Università degli Studi di Cagliari

PHD DEGREE
Electronic and Computer Engineering

Cycle XXXIII

TITLE OF THE PHD THESIS

Explaining Vulnerability of Machine Learning to Adversarial Attacks

Scientific Disciplinary Sector(s)

ING-INF/05

PhD Student: Marco Melis

Supervisors: Prof. Fabio Roli
Dott. Ing. Battista Biggio

Final exam. Academic Year 2019 – 2020
Thesis defence: March 2021 Session

The science of today is the technology of tomorrow.

– Edward Teller

Abstract

Pattern recognition systems based on machine learning techniques are nowadays widely used in many different fields, ranging from biometrics to computer security. In spite of the huge performance often provided by these systems, there is a general consensus that their reliability should be carefully assessed, in particular when applied to critical applications like medicine, criminal justice, financial markets, or self-driving cars. Especially in the era of big data, the chance of these systems inadvertently making wrong decisions while misguided from artifacts or spurious correlations in the training information is not negligible. Accordingly, to increase the trust of the users and identify the potential design flaws of the algorithms, many scientists started to explore the research field of *explainable machine learning*, with the goal of designing systems that are not only able to perform a pattern recognition task accurately, but that are also *interpretable*, *i.e.*, they can "explain or present their decisions in understandable terms to a human". In parallel, another research field raised more than 10 years ago: *adversarial machine learning*. In the context of security tasks like spam filtering or malware detection, skilled and adaptive adversaries (human beings) may modify legitimate samples to defeat a system, by creating the so-called *adversarial attacks*. Thus, scientists started to consider such adversarial environments during the engineering process, by evaluating the potential vulnerabilities, measuring the performance in terms of robustness against these attacks, and designing potential countermeasures. Despite the vast amount of work in this direction, providing a thorough definition of the effects of adversarial attacks is still an open issue, especially if the systems are not able to provide an explanation alongside their automated decisions.

In this thesis, we conduct a systematic investigation of the connections between explainability techniques and adversarial robustness, in order to gain a better understanding of the reasons behind the brittleness of modern machine learning algorithms, and with the goal of designing more robust systems that can be trusted by the users to safely operate in an adversarial environment. To this end, we start by proposing a novel optimization framework for crafting different adversarial attacks under the same unified mathematical formulation, which eases the study of their different security properties, like one of the most sudden, *transferability*. After providing a formal definition of this property and different quantitative metrics for its evaluation, we apply a novel explainability method based on highly-interpretable *relevance vectors*, that allows one to compare different models with respect to their learned behavior, and to get insights on their security properties, including *adversarial robustness* and their resilience to transfer attacks. Finally, to facilitate the practical application of these concepts, we also present `secml`, an open-source Python library that integrates all the tools required for developing and evaluating secure and explainable machine learning based systems, without the need of leveraging multiple third-party libraries.

Acknowledgments

The author sincerely thanks all the people involved in the research works included in this thesis.

This thesis has been partly supported by the following research projects: the EU H2020 project ALOHA, under the European Union's Horizon 2020 research and innovation programme (grant no. 780788); the PIS-DAS project, funded by the Sardinian Regional Administration (CUP E27H14003150007); the PRIN 2017 project RexLearn (grant no. 2017TWNMH2), funded by the Italian Ministry of Education, University and Research; the COMET Programme managed by FFG in the COMET Module S3AI and by BMK, BMDW, and the Province of Upper Austria.

Version of February 2021

Contents

List of Figures

List of Tables

Notation

Samples and Datasets

d	Number of features
$\mathcal{F}$	Feature set
$\boldsymbol{x} \in \mathbb{R}^{1\times \mathsf{d}}$	A single sample
$x^{(j)}$	j-th feature of sample $\boldsymbol{x}$
N	Number of samples
$\mathcal{X} \in \mathbb{R}^{\mathsf{N}\times \mathsf{d}}$	Set of all the samples (dataset)
$\boldsymbol{x}_i \in \mathcal{X}$	i-th sample in the dataset
y_i	True label of the i-th sample in the dataset
c	Number of classes
$\mathcal{Y} \in \mathbb{R}^{\mathsf{c}}$	Set of all the possible classes
$\omega_k \in \mathcal{Y}$	A single class
n	Number of training samples
$\mathcal{D}_{\text{tr}} \in \mathbb{R}^{\mathsf{n}\times \mathsf{d}}$	Training set
$\mathcal{Y}_{\text{tr}} \in \mathbb{R}^{\mathsf{n}}$	True labels of the training set's samples
m	Number of validation samples
$\mathcal{D}_{\text{val}} \in \mathbb{R}^{\mathsf{m}\times \mathsf{d}}$	Validation set
M	Number of test samples
$\mathcal{D}_{\text{ts}} \in \mathbb{R}^{\mathsf{M}\times \mathsf{d}}$	Test set

Classifiers

$f^\star$	(optimal) Decision function of a classification problem
f	(learned) Decision function of a classification problem
$g_1, g_2, \ldots, g_{\mathsf{c}}$	Discriminant functions (one for each class) of a classification problem defined by the canonical model
$\mathbf{w}$	Weights (parameters) of a linear classifier
b	Bias of a linear classifier
y^*	Predicted label for sample $\boldsymbol{x}$ by the classifier
$\hat{f}$	Decision function of a surrogate model for classifier f

$\hat{\mathcal{D}}_{\text{tr}}$	Surrogate training set used for learning $\hat{f}$
$\hat{\mathbf{w}}$	Weights (parameters) of the surrogate model $\hat{f}$

Adversarial Attacks

$\boldsymbol{x}^\star$	Optimal adversarial (attack) sample
$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{\delta}$	Adversarial (attack) sample obtained by adding the perturbation $\boldsymbol{\delta}$ to a sample $\boldsymbol{x}$
ε	Bound on the size of the adversarial perturbation, defined as the maximum ℓ_{p} norm between $\boldsymbol{x}$ and $\boldsymbol{x}'$
$\hat{\boldsymbol{x}} = \boldsymbol{x} + \hat{\boldsymbol{\delta}}$	Adversarial (attack) sample obtained by adding to a sample $\boldsymbol{x}$ the perturbation $\hat{\boldsymbol{\delta}}$ optimized against the surrogate classifier $\hat{f}$
$\mathcal{D}_\varepsilon$	Set of ε-sized adversarial (attack) samples

Explanations

$\boldsymbol{x}_0$	Baseline sample
$\boldsymbol{r} = [r^{(1)}, \ldots, r^{(\text{d})}]$	A feature-based explanation vector

Greek Letters with Pronunciation

Character	Name	Character	Name
α	alpha *AL-fuh*	ν	nu *NEW*
β	beta *BAY-tuh*	ξ, Ξ	xi *KSIGH*
γ, Γ	gamma *GAM-muh*	o	omicron *OM-uh-CRON*
δ, Δ	delta *DEL-tuh*	π, Π	pi *PIE*
ϵ	epsilon *EP-suh-lon*	ρ	rho *ROW*
ζ	zeta *ZAY-tuh*	σ, Σ	sigma *SIG-muh*
η	eta *AY-tuh*	τ	tau *TOW (as in cow)*
θ, Θ	theta *THAY-tuh*	υ, Υ	upsilon *OOP-suh-LON*
ι	iota *eye-OH-tuh*	ϕ, Φ	phi *FEE, or FI (as in hi)*
κ	kappa *KAP-uh*	χ	chi *KI (as in hi)*
λ, Λ	lambda *LAM-duh*	ψ, Ψ	psi *SIGH, or PSIGH*
μ	mu *MEW*	ω, Ω	omega *oh-MAY-guh*

Capitals shown are the ones that differ from Roman capitals.

Introduction 1

Machine learning has been defined by the pioneering work of Arthur Lee Samuel as the "field of study that gives computers the ability to learn without being explicitly programmed" [1]. In fact, one of the main abilities of human beings and some animal species is to recognize and classify objects in a fast and efficient way, making decisions after observing the surrounding environment. For instance, most people encounter no difficulty recognizing a friend in a picture or avoiding an obstacle while walking. In the latest decades, many scientists studied how to translate these processes into algorithms, without being able however to replicate them with enough accuracy for all the different tasks a human usually performs. The underlying reason is that these processes are carried out subconsciously, thus it is neither possible to explain nor understand in detail how they work. While in the early days it has been theorized that computers will be able to flawlessly perform tasks like automatic reading of handwritten texts, or understanding of human speech in real-time, these are still open research problems today. Generally, the machine learning field tries to address the problem of *automated pattern recognition*, including the above applications as well as many others, like identity verification from biometric traits, text categorization, malware detection in computer systems, or, more recently, objects recognition for robot vision and self-driving cars.

Pattern recognition, "the act of taking in raw data and taking an action based on the category of the pattern" [2], is often used as a synonym for machine learning [3]. In fact, it represents the study of algorithms that, given a set of samples (the raw data), whose category (or class) is unknown, assign such samples to one of the possible (predetermined) classes. This typically requires the system to have some previous knowledge of the problem, acquired through a set of pre-collected data. Then, after the recognition process is completed, *i.e.* the classification phase, similar to what a human would do, an action may be taken. For instance, when a walking robot recognizes an object in front of it, it may choose to slow down or change direction to avoid crashing. Similarly, when a mobile app is recognized to have malicious behaviors, a detection system may decide to delete it from its memory.

Although pattern recognition systems based on machine learning had shown impressive accuracy on many different tasks, when they started to penetrate critical application areas like medicine, criminal justice, financial markets, self-driving cars, the general public raised questions about the reliability of these algorithms. For example, while not correctly classifying a spam email may have no serious consequence, a self-driving car not accurately identifying a road sign may endanger the lives of the transported passengers. Thus, many industrial companies are very cautious in applying of machine learning in these critical applications. For example, Tesla, nowadays considered to offer one of the most advanced autopilot systems, explicitly tells the users that this technology is not supposed to replace the human driver, which always has to keep attention

to the street; or Waymo, the new autonomous vehicles by Google, have only been recently authorized to operate autonomously, yet only in specific limited areas.

Another inherent risk of these systems is the possibility of inadvertently making wrong decisions while misguided from artifacts or spurious correlations in the training information. Especially in the era of *big data*, by virtue of the increased scalability and high-performance of the infrastructures, a considerable amount of information is often acquired from social networks or other mass media sources (*e.g.*, list of purchases from e-commerce sites, comments under social posts). From them, machine learning models may inherit human biases and prejudices during the training process, possibly leading to unfair and wrong decisions.

Even though a mistake made by an autonomous system is usually easy to detect, given that some supervision mechanism is involved, the underlying reason why it happened may be hard to understand. To troubleshoot these systems when they make resounding errors, and thus increasing the users' trust, scientists started to explore a new research field called *explainable machine learning* (or explainable Artificial Intelligence, xAI). The goal in this case is to design algorithms which are not only able to perform a pattern recognition task accurately, but that are also *interpretable*, *i.e.*, they can "explain or present their decisions in understandable terms to a human" [4].

While there is little to no consensus on a formal definition of interpretability, the benefits of employing an explainable machine learning algorithm are widely recognized [5, 4, 6]. Firstly, by acquiring an explanation for a certain wrong decision, the source of the mistake may be easier to *track and fix*. Understanding it can also directly impact the progression of science, as the insights provided by an explanation can lead to *new discoveries*. In addition, interpretability may also help *knowledge transfer*, when machine learning models support human decision makers (*e.g.*, for medical diagnosis). In this case, the explanations may highlight hidden patterns in the data that are evident for the machine, but possibly unknown for the human. Lastly, there are also *legal reasons* behind the study of explainable machine learning. The European Parliament has recently adopted a new legislation on data protection, the General Data Protection Regulation (GDPR). An innovative aspect of the GDPR is the clauses on automated decision makers, which introduce the right for all individuals to obtain "meaningful explanations of the logic involved" when automated decision-making occurs [7]. All things considered, there is a general agreement on the urgent need for the implementation of such explainability principle in machine learning based systems, and it represents today a huge open scientific challenge.

In parallel, another research field started to rise more than 10 years ago [8]: *adversarial machine learning*. It embraces all the techniques necessary to evaluate machine learning algorithms employed on security-related applications, like spam filtering, or malware detection. In this case, the task is usually to discriminate between benign and malicious samples, *e.g.*, legitimate and intrusive network traffic. In this context, skilled and adaptive adversaries (human beings) may modify benign samples to defeat the system, creating the so-called *adversarial attacks*. In spam filtering, for example, spammers usually alter the text in their messages

to have them misclassified as benign, by misspelling a specific keyword (like "viagra" changed to "v14gr4"), or by adding some known good words to spam e-mails [9]. In intrusion detection systems, a hacker may camouflage intrusive traffic by mimicking legitimate transmissions, like using the same packet size or content [10].

For these reasons, scientists started to consider such *adversarial environments* when designing the pattern recognition systems, by explicitly taking into account the possible presence of malicious entities. This includes identifying the vulnerabilities which an adversary can exploit to make the system ineffective; evaluating the performance not only in terms of recognition accuracy, but also in terms of robustness against the attacks (by simulating the possible attack strategies); and designing the necessary countermeasures to these threats.

Adversarial machine learning has recently gained attention also in the computer vision field. It happened in fact that a very popular type of machine learning algorithm named *neural networks*, despite the impressive performance, was making bizarre mistakes without an apparent reason, *e.g.*, recognizing a bus as an ostrich. Trying to interpret their (wrong) decisions, it has been found that these systems can be fooled using legitimate images that are carefully perturbed by an adversary to be misclassified, namely *adversarial examples* [11]. These attacks are often (but not necessarily[1]) indistinguishable from normal data, making them an even more sudden threat. Despite the vast amount of work on this issue, providing a thorough definition of the effects of adversarial attacks is still an open problem, especially if the systems are not able to provide an explanation alongside their decisions.

1: This misconception about adversarial examples, *i.e.*, they should be minimally perturbed, has held popularity in the scientific community for many years [8]. Only recently, high-confidence attacks started to be considered better suitable for the purpose of security evaluation. See Section 2.2.2 for more details.

In this thesis we thus argue that the process of designing a pattern recognition system that operates in an adversarial environment, should not only be based on a quantitative evaluation of the most critical properties of adversarial attacks, but may also directly benefit from the insights provided by explainable machine learning techniques. As this research direction has only been sparsely explored in the current literature [12, 13, 14, 15], we hope that our work may contribute to the formulation of more robust learning algorithms in the future.

1.1 Outline

We start in Chapter 2 by providing a state of the art on machine learning algorithms for classification tasks, as well as describing the main concepts and methods for their security evaluation in adversarial environments, and to provide an explanation for their decisions.

Then, after presenting in Chapter 3 a summary of the main limitation of the current state of art and our contributions in that context, we start by proposing a novel unified framework for crafting the two different types of adversarial attacks, *i.e.*, evasion and poisoning, through a gradient-descent optimization procedure (Chapter 4).

In Chapter 5 we study one of the most sudden properties of these attacks, *transferability*, which is their ability to be effective even on different systems from the one which have been crafted on. We provide a formal

definition of transferability and three quantitative metrics to evaluate the security of machine learning algorithms to this threat.

Afterwards, in Chapter 6, we discuss how gradient-based explanation methods can be used effectively to compare different machine learning models with respect to their learned behavior, and get insights into their security properties, like adversarial robustness or the resilience to transfer attacks.

In Chapter 7 we present `secml`, an open-source Python library that aims to integrate all the tools required for developing and evaluating secure and explainable machine learning based systems, without the need of leveraging multiple third-party libraries.

Subsequently, in Chapter 8, we conduct an experimental investigation of the transferability of adversarial attacks on three different applicative cases, and we statistically evaluate the connections between the uniformity of gradient-based explanations and adversarial robustness.

The conclusive remarks and the future research directions are finally discussed in Chapter 9.

1.2 Related Publications

This thesis includes research work part of the following publications:

- A. Demontis, M. Melis, *et al.*, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks", in Proceedings of the 28th USENIX Security Symposium, 2019 [16];
- M. Melis, *et al.*, "Explaining black-box android malware detection", in Proceedings of the 26th European Signal Processing Conference (EUSIPCO), 2018 [17];
- M. Melis, *et al.*, "Do gradient-based explanations tell anything about adversarial robustness to android malware?", arXiv preprint arXiv:2005.01452, 2020 [18];
- M. Melis, *et al.*, "Secml: A python library for secure and explainable machine learning", arXiv preprint arXiv:1912.10013, 2019 [19].

Moreover, the author of this thesis collaborated to the following publications before and during his Doctorate:

- M. Melis, *et al.*, "Is deep learning safe for robot vision? adversarial examples against the icub humanoid.", in Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW), 2017 [20];
- A. Demontis, M. Melis, *et al.*, "Yes, machine learning can be more secure! a case study on android malware detection.", IEEE Transactions on Dependable and Secure Computing, 2017 [21];
- A. Sotgiu, M. Melis, *et al.*, "Deep neural rejection against adversarial examples", EURASIP Journal on Information Security, 2020 [22];
- F. Crecchi, M. Melis, *et al.*, "FADER: Fast Adversarial Example Rejection.", arXiv preprint arXiv:2010.09119, 2020 [23].

For a full list of publications by the author of this thesis see:

`https://www.linkedin.com/in/melismarco`

Background 2

In this chapter we provide an overview of the necessary steps to build a pattern recognition system based on *machine learning* (Section 2.1), including a description of the most used learning algorithms and techniques in popular applications like computer vision, malware detection, and biometric recognition. We also discuss how to evaluate its performance on the specific recognition tasks. Within this state of the art, we describe how easily many of these techniques can be fooled by an adversary, crafting powerful *adversarial attacks* (Section 2.2). We highlight the *attack surface* available to the adversary, and present a generalized *threat model*. Then, we review both test-time and training-time attacks, as well as few of the defence techniques proposed in the literature. In the last part of the section, we discuss how the adversary can successfully attack a system even if he possesses limited or no knowledge of it, leveraging a property of adversarial attacks called *transferability*. Finally, in Section 2.3, we examine the issue of *trusting* these automated systems, especially when applied to safety-critical applications, like self-driving cars or medical diagnosis. In these cases, the user may require an *explanation* of the decisions, which could be a challenge depending on how *interpretable* is the underlying machine learning model.

2.1 Machine Learning Systems

We start by considering the task of automatically select a specific species of flower from a set on unknown samples. Instinctively, this represents a *classification* problem, as the system must be capable of distinguish between a limited number of species, *i.e.* classes, and assign the correct one to each sample.

Figure 2.1: Architecture of a typical machine learning based system.

As depicted in Figure 2.1, to build a machine learning system able to tackle this classification task the first phase should be *data acquisition*, when the samples to be analyzed are collected. As the information about the real objects is transferred to the system using particular interfaces, for example, cameras, microphones, movement sensors, it cannot understand the characteristics of each sample in their entirety. Thus, it is fundamental to choose the more appropriate representation for the specific application. In the case of flower species classification, an intuitive choice could be to acquire an image of the sample to analyze, while in other applications it could be more effective to jointly consider two or more representation of the same object, like a video and an audio sample for a voice recognition task, by following a *multimodal* paradigm [24, 25].

The data acquisition phase is a critical step as it can be disrupted by undesirable contamination, *e.g.*, noise in an audio track, causing problems in the subsequent analysis phase. In particular, it is always preferable to acquire all data in the same conditions, *e.g.*, same illumination or prospective in the case of images, so that will be easier for the system to distinguish between all the differences or similarities that could actually exist between the samples. To mitigate the problem, data acquisition it is often followed by a *pre-processing* phase, when different techniques are used trying to uniform the representation of all the samples and remove, as much as possible, the noise. For example, in the case of images a standard procedure involves normalizing the samples to make them independent from translation, rotation or distortion [26].

The next question which should be answered when designing a machine learning system is: which aspect of the samples can be measured? From the same acquired data, in fact, it is possible to extract many different characteristics, namely *features*, not always relevant for the specific classification task. In the case of flower species recognition, one could think about measuring the width and the length of the petals, as they may allow to *discriminate* between the different classes, while it could be inappropriate to analyze the color of the flowers as it does not represent a peculiar characteristic of a species. The set of extracted features forms the *feature space*, where the learning system works in. A 2-dimensional representation example of a feature space constituted by the length of the sepals and the petals of different flowers, extracted from the famous *iris-flower* dataset [27], is reported in Figure 2.2.

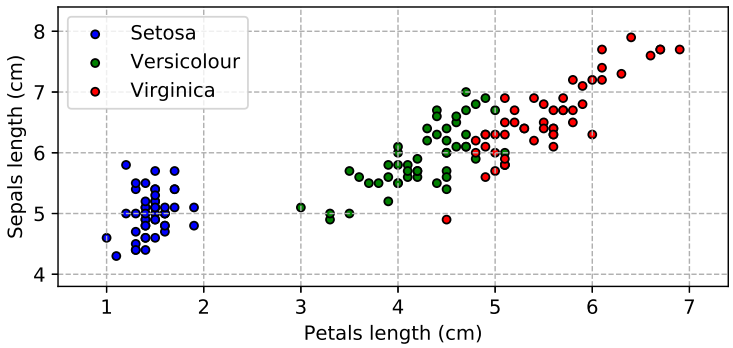


Figure 2.2: Feature space of the 3-classes *iris-flower* dataset [27]. Each dot represents a sample from the three different species, *setosa* (blue), *versicolour* (green), *virginica* (red). Each sample is defined by two features, sepals and petals length.

The choice of which features to extract from the data, however, should not be based only on the relevance for the specific application. It should take into account the computational cost of analyzing a large number of characteristics and, generally, the specificities of the learning algorithm which will be used by the system. Some algorithms, in fact, work better when the feature space is large, while others provide a higher performance with a small set of independent features. In the latter case, to the feature extraction phase often follows a *feature reduction* or *feature selection* step, when specific methods (*e.g.* PCA, LDA) are employed to compact the size of the feature space [28, 29, 30].

Notation. In the following, for each sample we define its vector representation $\boldsymbol{x} = [x^{(1)}, x^{(2)}, \dots, x^{(\mathsf{d})}]$, with d the number of features. We denote the set of all possible features as $\mathcal{F}$. The dataset with the N acquired samples is defined as $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^{\mathsf{N}}$. If known, the label representing

the class of the i-th sample is indicated as y_i. Each label y is part of the set of all the c classes ω, defined as $y \in \mathcal{Y} = \{\omega_k\}_{k=1}^{\mathsf{c}}$.

2.1.1 Classification Algorithms

After building the feature space, it is necessary to choose a mathematical function that performs the *classification*, *i.e.*, allows the system to assign a certain numerical, or textual, value to each input sample, representing the predicted *class*. This value usually takes the name of *label*. Formally, this requires selecting a *decision function* $f^\star : \mathcal{X} \mapsto \mathcal{Y}$ which exactly maps each sample $\boldsymbol{x} \in \mathcal{X}$ to the corresponding true class $y \in \mathcal{Y}$. From a statistical point of view, this translates into defining a function f that estimates as precisely as possible the value of $f^\star(\boldsymbol{x})$, *i.e.* minimizes the prediction error $f(\mathcal{X}) \neq Y$.

To this end, let's extract a subset of n samples from all the acquired data, which is usually called the *training set*, denoted as $\mathcal{D}_{\text{tr}} = \{\boldsymbol{x}_i\}_{i=1}^{\mathsf{n}}$, with $\boldsymbol{x}_i \in \mathcal{X}$. The learning algorithm has the task of selecting the best function f through the information contained in this data and the set of the corresponding true labels $\mathcal{Y}_{\text{tr}} = \{y_i\}_{i=1}^{\mathsf{n}}$, with $y_i \in \mathcal{Y}$. This type of classification paradigm is named *supervised*. Conversely, if the set $\mathcal{Y}_{\text{tr}}$ is only partially known or unknown, the learning process is defined *semi-supervised* or *unsupervised* [2, 3].

Considering again the task of automatically select the correct flower species, this is defined as a classification problem where the set of all the possible predictions is limited to only 3 elements, *i.e.* $\mathcal{Y} = \{\omega_1, \omega_2, \omega_3\}$. Computing the mapping function f actually involves partitioning the feature space into different *decision regions*, one for each class. The border that separates each region is called *decision boundary*. Depending on the chosen learning algorithm, one could obtain quite different decision regions, as illustrated in Figure 2.3.

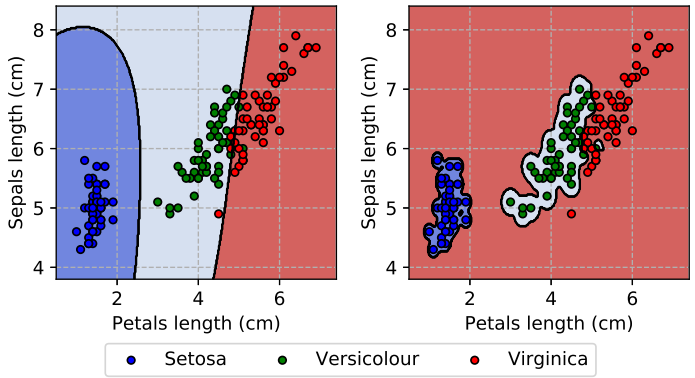


Figure 2.3: Decision regions learned by different machine learning algorithms in the feature space of the 3-classes dataset *iris-flower* [27]. To the samples inside each region is assigned a different class: *setosa* (blue region), *versicolour* (green region), *virginica* (red region). The decision boundaries are reported in black. The decision function on the right is *overfitting* the training data, while the (left) one *generalizes* better on never-before-seen samples.

The choice of the learning algorithm to employ should take into account different factors related to the behavior expected during the prediction phase. For example, the decision function reported in Figure 2.3 (left) probably performs better in terms of *generalization*, meaning that may be able to assign the correct class to never-before-seen samples more accurately. Conversely, the function in Figure 2.3 (right) minimizes the prediction error on the samples from the training set, while probably loosing accuracy on new data. The latter case is called *overfitting* [2, 3]. To evaluate the behavior of the system in this sense, one can compute the

prediction error on a different set of samples $\mathcal{D}_{\text{val}} \not\subset \mathcal{D}_{\text{tr}}$, usually called *validation set*, which are not used for learning the decision function.

A more complete definition of a classification problem is provided by the *canonical model*, schematized in Figure 2.4 [2, 31]. The output of the decision function f, *i.e.*, the predicted label y^*, is obtained by evaluating a set of c *discriminant functions* $\{g_1(\boldsymbol{x}), \ldots, g_{\mathsf{c}}(\boldsymbol{x})\}$, each of them providing a *score* for the input sample w.r.t. each class $\omega \in \mathcal{Y}$. Typically, the predicted class is the one corresponding to the discriminant function g_k which outputs the higher score, following the maximum support rule [31]:

$$y^* = f(\boldsymbol{x}) = \omega_k \in \mathcal{Y} \quad \text{with} \quad \omega_k = \arg\max_{k=1,\ldots,\mathsf{c}} g_k(\boldsymbol{x}) \ . \tag{2.1}$$

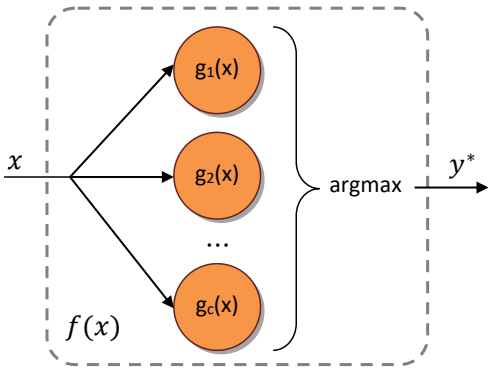


Figure 2.4: Canonical classification model. The predicted class y^* corresponding to the sample $\boldsymbol{x}$ is chosen by computing the arg max on the outputs of the c discriminant functions $\{g_1(\boldsymbol{x}), \ldots, g_{\mathsf{c}}(\boldsymbol{x})\}$.

Linear Models

As previously mentioned, the choice of the learning algorithm should take into account the specific application requirements and many other factors. While does not exist a system that perfectly tackles any recognition task providing the same performance (*no free lunch theorem*, [32]), few machine learning algorithms demonstrate good generalization abilities even if applied to wide range of applications. This is the case of the *linear models*, which decision function is given by the following:

$$y^* = f(\boldsymbol{x}) = \text{sign}(\mathbf{w}^\top \boldsymbol{x} + b) \ , \tag{2.2}$$

with $\mathbf{w}$ a vector of parameters and b the bias, representing the distance of the decision boundary from the origin of the chosen reference system. From Equation 2.2 it is clear that the predicted class y^* can only take two values, depending on $\text{sign}(\mathbf{w}^\top \boldsymbol{x} + b)$, which are $+1$ (-1) if $f(\boldsymbol{x}) \geq 0$ $(f(\boldsymbol{x}) < 0)$. As a result, these classifiers are considered *binary* algorithms, as the set of all the possible labels is limited to $\mathcal{Y} = \{-1, +1\}$. The class corresponding to $y = -1$ is usually named the *negative*, while the *positive* class corresponds to $y = +1$. The wording *linear model* derives from the form of the decision boundary, given by $\mathbf{w}^\top \boldsymbol{x} + b = 0$, which is linear the feature space.

The parameters $\mathbf{w}$ and the bias b are not given along with the decision algorithm, and should be chosen by solving the following optimization

problem [33] over all the training samples $\boldsymbol{x} \in \mathcal{D}_{\text{tr}}$:

$$\min_{\mathbf{w},b} R(\mathbf{w}) + C \sum_{i=1}^{n} L(y_i, \mathbf{w}^\top \boldsymbol{x}_i + b) \ . \tag{2.3}$$

In Equation 2.3 we find L, which is the so-called *loss* function, a measure of the difference between the classifier's predictions and the true class y_i of each training sample $\boldsymbol{x}_i$; the regularizer R, which goal is to limit the magnitude of the parameters $\mathbf{w}$ (usually leading to less overfitting); and C, a constant to be chosen during the design phase that balances the contribution of L and R, which represents a *hyperparameter* of the system. Depending on the chosen mathematical notation, this latter constant may be denoted with α.

Support Vector Machines

By a particular choice of the loss function and the regularizer, a decision region with specific properties can be obtained [33]. One of the most common choices is the *hinge loss* and the *quadratic regularizer* $R = \|\mathbf{w}\|_2^2$, which constitutes a particular type of linear classifiers named *Support Vector Machines* (SVMs) [34]. The rationale behind these models is to maximize the separation, *i.e.*, the margin $\frac{2}{\|\mathbf{w}\|}$, between the training samples of the two different classes, as schematized in Figure 2.5.

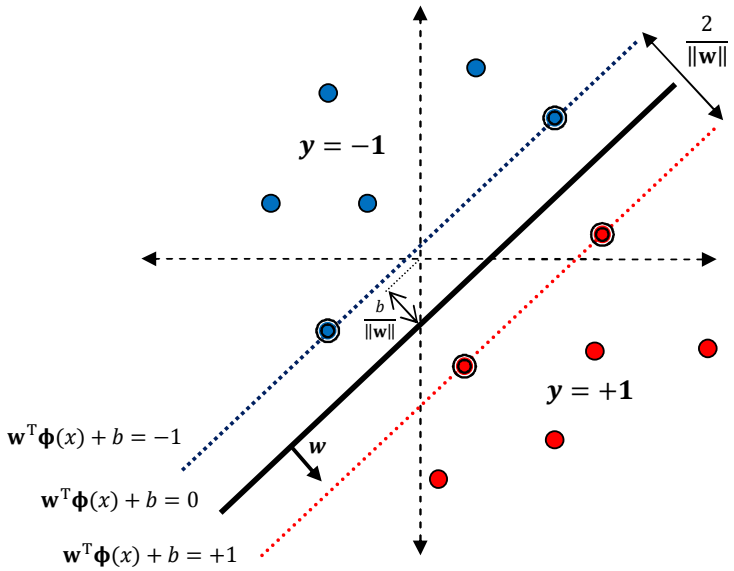


Figure 2.5: A binary classification problem in the feature space solved by a Support Vector Machine (SVM). The double-circled points are the *support vectors*, blue for the negative class, red for the positive class. The decision *hyperplanes* are given by $\text{sign}(g_{+1}(\boldsymbol{x}))$, with g_{+1} the discriminant function for the positive class. ϕ is a function that maps the samples into a high-dimensional space, where the classes are more likely to be linearly separable. Lastly, $\frac{b}{\|\mathbf{w}\|}$ is the distance of the decision boundary from the origin.

The optimization problem in Equation 2.3 can be rewritten in this case using the following *primal form*:

$$\begin{aligned} &\min_{\mathbf{w},\xi,b} \frac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^{n} \xi_i \\ &\text{s.t. } \xi_i \geq 0 \ , \\ &\qquad \xi_i \geq 1 - y_i(\mathbf{w}^\top \boldsymbol{x}_i + b) \ . \end{aligned} \tag{2.4}$$

For the mathematical properties of the SVMs [35], the parameters vector $\mathbf{w}$ can be expressed as a linear combination of the training samples,

resulting in the following Wolfe *dual form* [36] of Equation 2.4, which only depends on a series of scalar products:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \; & \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{n} \alpha_i \\ \text{s.t. } & 0 \leq \alpha_i \leq C \;\; , \\ & \sum_{i=1}^{n} y_i \alpha_i = 0 \;\; . \end{aligned} \tag{2.5}$$

The solution found by Equation 2.5 is *sparse*, meaning that $\alpha_i > 0$ only for a subset of the dual parameters $\boldsymbol{\alpha}$, as a result of the constraint $0 \leq \alpha_i \leq C$. The training samples associated with the non-null α values take the name of *support vectors* (SVs).

Finally, one can compute the discriminant function for the input $\boldsymbol{x}$ associated with the positive class, which will depend solely on a scalar product between the training samples:

$$g_{+1}(\boldsymbol{x}) = \sum_{i=1}^{n} y_i \alpha_i K(\boldsymbol{x}, \boldsymbol{x}_i) + b \;\; . \tag{2.6}$$

Kernel trick. The scalar products in Equation 2.5 and Equation 2.6 are expressed using the function $K(\cdot, \cdot)$, which takes the name of *kernel*. Using this *kernel trick*, it is possible to implicitly solve the optimization problem in Equation 2.5 by simply knowing the value of the scalar products between the training samples. It has been proved that only for the kernel functions under the Mercer theorem, the optimization problem is convex and the solution unique[1] [38]. An additional advantage of the kernel trick is that allows to implicitly map the optimization problem into a high-dimensional space Φ, using a function $\phi : \mathcal{X} \mapsto \Phi$, which can also be *non-linear*. In this space, it is more likely that the samples from the different classes are linearly separable, which can be useful in the case of complex classification problems like biometric recognition [39] or computer vision. To this end, one of the most used kernels is the Radial Basis Function (RBF), given by:

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left(\frac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2}\right) \;\; , \tag{2.7}$$

where $\|\cdot\|$ is the ℓ_2 norm operator and σ is a free tuning constant. Other notable kernel functions are the Polynomial Kernel, the Laplacian Kernel, and the Histogram Intersection Kernel.

1: If the optimization problem is not convex, the solution converges to a local minimum, which is still an acceptable outcome in most cases [37].

Multiclass Classification

If the set $\mathcal{Y} = \{\omega_1, \ldots, \omega_c\}$ of all the possible classes is constituted by more than two elements, the corresponding classification task could not be directly tackled using a binary learning algorithm, like the support vector machines. However, it has been demonstrated that a *multiclass* classification task can be solved effectively by transforming it to a series of binary, *i.e.*, 2-classes, problems and combining the outputs following the canonical model (Figure 2.4) [2, 3].

One of the most used strategies to decompose a multiclass problem is called *One-vs-All* (OvA)[2] , which consists on learning a binary discriminant function for each of the available classes, for a total of c different classifiers, by considering the training samples from that class as positives, while all the other samples are treated as negatives [3]. A different method, called *One-vs-One* (OvO), considers $\mathsf{c}(\mathsf{c}-1)/2$ binary discriminant functions, one for each pair of available classes [3]. To assign the predicted class, in the case of OvA, the maximum support rule is used (Equation 2.1), while in the OvO case, a common choice is to follow a most-votes paradigm, by assigning the class which is picked more often by the binary classifiers (positive value of the score) [40].

2: Occasionally called One-vs-Rest (OvR) or One-Against-All (OAA).

These techniques allow to employ linear classifiers like the support vector machines in multiclass problems [41] by learning, for example in the case of One-vs-All, one SVM for each class, indexed using $k = \{1, \dots, \mathsf{c}\}$, and predicting the label y^* of an input sample $\boldsymbol{x}$ as follows:

$$y^* = f(\boldsymbol{x}) = \arg\max_{k=1,\dots,\mathsf{c}} g_k(\boldsymbol{x}) = \sum_{i=1}^{\mathsf{n}} y_i^k \alpha_i^k K(\boldsymbol{x}, \boldsymbol{x}_i) + b^k \quad , \tag{2.8}$$

where y^k are the true labels binarized w.r.t the k-th class following the OvA scheme[3] , and g_k, $\boldsymbol{\alpha}^k$, b^k are, respectively, the decision function, the dual parameters and the bias of the k-th binary classifier.

3: The OvA binarization process works by assigning the label +1 to all samples from the k-th class, and the label −1 to the rest of the samples.

Although, even in this case, the solution is sparse for each binary classifier taken separately, as only a subset of the dual parameters $\{\alpha_i^k\}_{i=1}^{\mathsf{n}}$ is not null, for the multiclass system as a whole the combination of all the support vectors is often coincident with the entire training set, and thus the global solution will be dense.

Neural Networks

The human brain excels in performing many different tasks as it is formed by a thousand billions of interconnected *neurons*, a basic unit that reacts in a very fast way to specific patterns in the input electrical signals. It implements a massive parallelized structure and, more importantly, it is able to learn from real time data received from the skin, the eyes, the ears, etc., by changing or creating new connections between units.

Neural Networks are classifiers inspired by the architecture of the human brain [3]. Their basic unit is called *perceptron*, developed in the 1950s and 1960s by the scientist Frank Rosenblatt [42], inspired by the earlier work of Warren McCulloch and Walter Pitts [43]. With a very simple structure, a perceptron takes several inputs $\{x^{(1)}, x^{(2)}, \dots\} \in \boldsymbol{x}$ and produces a single binary output $g(\boldsymbol{x})$, which is determined as 0 or 1 depending on whether the weighted sum $\sum_j \mathrm{w}^{(j)} x^{(j)}$ is less than or greater than some threshold value th. This can be rewritten as a dot-product to obtain the decision function of a perceptron as follows:

$$g(\boldsymbol{x}) = \begin{cases} 0, \text{ if } \mathbf{w}^\top \boldsymbol{x} + \mathrm{th} \leq 0 \\ 1, \text{ if } \mathbf{w}^\top \boldsymbol{x} + \mathrm{th} > 0 \end{cases} \quad , \tag{2.9}$$

which is identical to the sign function in Equation 2.2, with $\mathrm{th} \equiv b$, making perceptrons equivalent to a classifier based on the linear model.

By combining multiple instances of this basic unit into different layers, complex *networks* can be built, like the one depicted in Figure 2.6, which takes the name of *Multi Layer Perceptron* (MLP). In this way, a many-layers network of perceptrons can engage in sophisticated decision making.

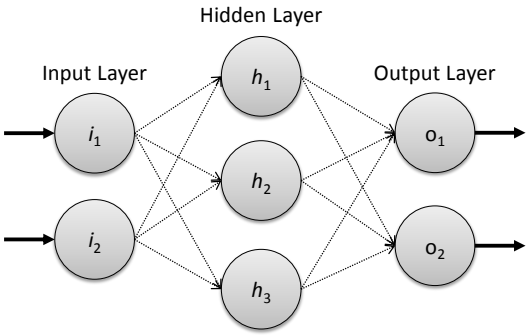


Figure 2.6: Example of *multi layer perceptron* with two neurons in the input layer, three in the hidden layer, and two final output units, one for each class.

The *input layer* of a MLP is constituted by one perceptron for each feature of the input sample. The intermediate layer(s) are referred to as *hidden layers*, being their inputs and outputs not explicitly visible in the final network's output. Each connection between perceptrons represents the weight used in Equation 2.9 to compute the decision. These parameters must be carefully chosen to reproduce a desired input-output behaviour, according to the set of training samples. Finally, the *output layer* is usually constituted by one unit for each existing class, making neural networks suitable for multiclass classification tasks.

To learn the appropriate weights from the training set, an efficient algorithm was devised in the 1980s, which is called *back-propagation* [44]. At the beginning, all the edge weights are initialized randomly, or with a specific criteria like symmetry breaking [45]. Then, for each sample in the training dataset, the output of the network is observed and compared, using an error function E, with the desired output, *i.e.* the true labels of the input samples. The error value is *back-propagated* through the network to adjust the parameters, usually leveraging a gradient-based learning procedure like *gradient descent*.[4] This process is repeated until the output error is below a predetermined threshold. For a network with a single output unit (*e.g.*, for binary classification tasks), an elementary error function is the *mean squared error*:

$$E(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\left(g(\boldsymbol{x}_i) - y_i\right)^2 \quad . \tag{2.10}$$

For multiclass classification problems, more complex error functions are usually employed, including the well-known *cross-entropy loss*. To mitigate the problem of overfitting, the error function may include a regularization term, similarly to Equation 2.3.

As the back-propagation algorithm requires the function of each basic unit to be continuous and differentiable, the sign function in Equation 2.9 and, thus, the decision function $g(\cdot)$ in Equation 2.10, is replaced by a so-called *activation function*. A common choice is the *sigmoid*[5] :

$$\sigma(\boldsymbol{x}) = \frac{1}{1 + e^{-g(\boldsymbol{x})}} \quad . \tag{2.11}$$

The choice of the activation function, *e.g.*, Gaussian, Radial Basis Function,

4: For a complete overview on the theory behind back-propagation see [46].

5: Sigmoids allow small changes in the weights and bias to cause only minimal variations in the output, making the learning process more stable. Conversely, for standard perceptrons a little change in the weights and bias can sometimes cause the output to completely flip.

Hyperbolic Tangent, Rectified linear unit (ReLU) [47], may lead to very different outcomes of the learning process.

Deep Learning

While basic neural networks like multi layer perceptrons provide impressive performance for a wide range of tasks, due to the flexibility in their structure (one may tune the number of layers and basic units, choose a different loss and activation functions, etc.), to tackle very complex classification problems like a human brain does, a more advanced architecture, *e.g.*, compared to the one reported in Figure 2.6, is required. To represent, often non-linear, patterns of increasing complexity, researchers in the 1980s and 1990s started designing *Deep* Neural Networks (DNNs), which have the basic structure of MLPs, but with *many* hidden layers. Nowadays, DNNs are widely employed for many pattern recognition tasks, with different variants of a few basic approaches depending on the specific application [48].

Recurrent Neural Networks (RNNs) [44, 49] are designed to take as input a data series with no predetermined limit on size, like temporal data. To this end, the hidden units form a directed graph along the sequence, creating a memory of the inputs. This characteristic makes them particularly suitable for language modeling tasks [50, 51]. When unfolded, a RNN may appear as depicted in Figure 2.7.

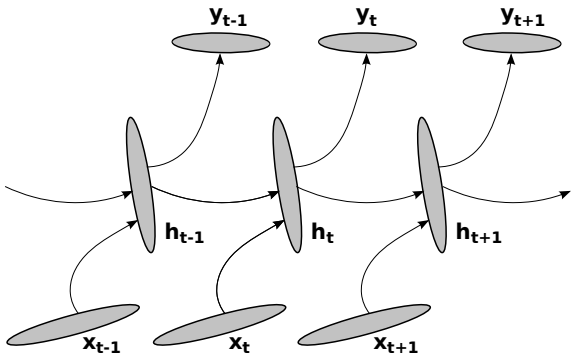


Figure 2.7: A standard Recurrent Neural Network (RNN) unfolded in time [49]. The hidden units form a directed graph along the sequence, generating in the network a memory of the inputs.

In the field of computer vision, a widely employed set of architectures takes the name of *Convolutional Neural Networks* (CNNs) [52]. In this case the network is not fully-connected (where all units are linked, like MLPs), but it exploits the spatial adjacency in grid-like inputs, like images pixels. This design is inspired by the vision processes of living organisms. Early work by Hubel and Wiesel [53] showed that the animal visual cortex contains neurons that individually respond to small regions of the visual field, known as the *receptive fields*. The receptive fields of neighboring neurons partially overlap such that they cover the entire visual field.

A standard CNN design is reported in Figure 2.8. The hidden layers carry out specific image processing operations, alternating two kinds of layers: (i) *convolutional filtering layers*, whose connection weights that determine the filter implemented are learnt during training; (ii) *pooling layers*, which have predefined connection weights, and carry out a down-sampling operation on the outputs of the previous layer. In particular, each neuron in the filtering layer is connected to a small region of the input neurons, called the *local receptive field*. Then, the pooling operation replaces the

output of the layer with a summary statistic of nearby outputs, *e.g.*, by taking the maximum in a certain region. Finally, the upper layers of the CNN usually consist of standard, fully-connected, networks similar to a MLP.

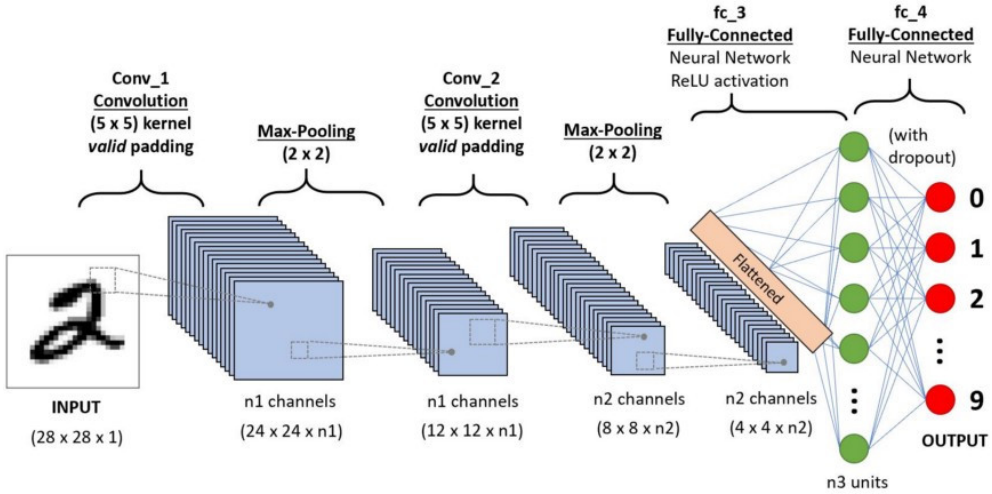


Figure 2.8: A Convolutional Neural Network (CNN) to classify handwritten digits [54]. The network is formed by two repeated pairs of filtering-pooling layers and a fully-connected top MLP with one hidden layer and ReLU as the activation function.

Surprisingly, by visualizing the output of the activation functions of each pair of filtering-pooling layers, can be clearly seen how a CNN gradually learns more complex and abstract notions from the training data. While shallow layers show only borders and color traces, in the deeper layers patterns of actual objects can be easily recognized, as shown in Figure 2.9 [55, 56]. An analysis of these intermediate outputs can help the understanding of how these classifiers work, and possibly give an insight of why they show high vulnerability to external attacks like adversarial examples (see Section 2.2.2) [11].

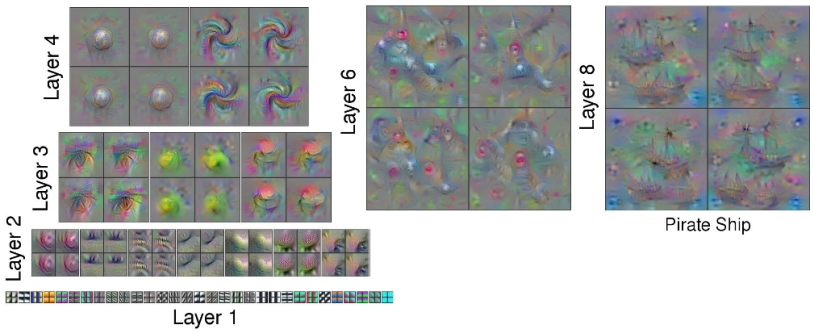


Figure 2.9: Visualization of the activation functions of six layers of a deep convolutional neural network, from the shallower (layer 1) to the deeper (layer 8), obtained using an input sample of class *pirate ship* [56]. This shows the increase in complexity and variation on higher layers, comprised of simpler components from lower layers. The distinctive features of the input objects can be clearly recognized at the output (deeper) layer.

Another interesting type of DNN architectures which gained a lot of attention in the latest years are the Generative Adversarial Networks (GANs) [57]. They are based on two separate neural networks, usually CNNs if applied to computer vision tasks, that contest with each other in a zero-sum game, where one gain is the other's loss. In other words, one network (the *generator* or *decoder*) learns to generate new data from the training set, while the other network (the *discriminator* or *encoder*) tries to distinguish between real examples and synthetic ones. For example, a GAN trained on real-world images can generate new artificial images that look authentic to a human observer, having learned to replicate many of the characteristics of the depicted objects or subjects. These systems

raised many security concerns lately, as are being used to produce the so-called *DeepFakes* [58], highly realistic, possibly incriminating, material, including fake celebrity videos, fake audio tracks, and fake news.

2.1.2 Performance Evaluation

To evaluate the performance of a machine learning algorithm, it is generally necessary to compute the *prediction error*, *i.e.* how the predicted value (label) y_i^* for sample $\boldsymbol{x}_i$ is consistent with its true class y_i. This procedure is repeated for each sample in a new dataset $\mathcal{D}_{\text{ts}} = \{\boldsymbol{x}_i\}_{i=1}^{\mathsf{M}}$, disjointed from the training set $\mathcal{D}_{\text{tr}}$ and the validation set $\mathcal{D}_{\text{val}}$, which is called *test set*.

One of the most used metrics to compute the prediction error in the case of classification problems is the *accuracy*, which represents the portion of correct predictions with respect to the total number of tested samples:

$$\text{acc}(\boldsymbol{y}*, \boldsymbol{y}) = \frac{1}{\mathsf{M}} \sum_{i=1}^{\mathsf{M}} (y_i^* == y_i) \ . \tag{2.12}$$

For a binary classification problem, it is also possible to define few different metrics to evaluate specific aspects of the learning algorithm. Considering the samples from the positive class only, we define the *True Positives* (TP) as the set of samples which classes have been correctly predicted, and the *False Positives* (FP) as the set of samples to which have been assigned the wrong labels. Considering the samples from the negative class, similar definitions can be provided in terms of the *True Negatives* (TN) and the *False Negatives* (FN).

Ideally, the value of the discriminant function computed for a specific class ω_c on the samples from that same class will be always higher than the values obtained by computing the function on samples from the other classes $\omega_{k \neq c}$. As a consequence, a single threshold θ_c chosen *a priori* may perfectly separate the set of positives and negatives scores. As depicted in Figure 2.10, however, the two sets can overlap, for example, if high positive score is given to a sample from the negative class, meaning it becomes a *false positive*, or vice-versa, if high negative score is given, resulting in a *false negative*.

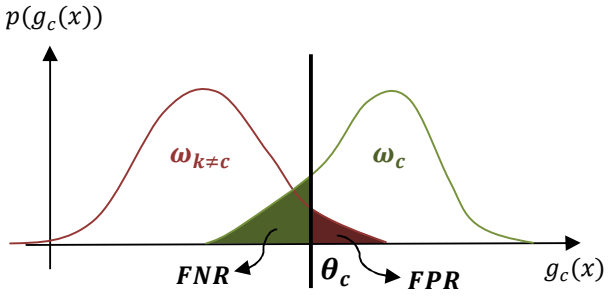


Figure 2.10: Plotting of the positives (green) and the negatives (red) scores to determine the False Positive Rate (FPR), the False Negative Rate (FNR) and the decision threshold θ_c.

The choice of the decision threshold θ_c is fundamental, as in specific applications one may prefer to have, for example, as few as possible false positives (like in a biometric recognition systems applied to access control for a bank). To correctly estimate the best threshold for each case,

the performance of the system is usually measured using the following metrics:

- False Positive Rate (FPR), representing the probability that a negative sample is recognized as positive. A common range for this metric is 1-5% FPR.
- False Negative Rate (FNR), representing the probability that a positive sample is recognized as negative.

To graphically analyse the performance in terms of FPR and FNR metrics, one can plot the *Receiver Operating Characteristic* (ROC) curve. It shows the ratio between the false negatives and the false positives at increasing values of the decision threshold. From the ROC curve, the system performance can be directly observed and, in addition, the threshold θ_c can be easily balanced in order to obtain the desired value for the two metrics depending on the task to tackle.

2.2 Secure Machine Learning Systems

Machine learning techniques are nowadays being extensively employed in security applications, like spam filtering, intrusion detection systems, or malware detectors. The underlying reason is that traditional security systems struggle to generalize on never-before-seen samples, *i.e.*, detect new attacks. Instead, we already illustrated how many different classification algorithms have indeed shown high generalisation capability. Since the 90s, computer viruses, malware and Internet scams have increased not only in volume, but also in terms of inconsistency and sophistication, in response to the growing complexity of defense systems. Automatic tools to design novel attacks have been developed, making large-scale automatization of subtle attacks practical for non-skilled attackers too. A straightforward example of this is provided by phishing kits, which automatically compromise legitimate (vulnerable) websites, and hide phishing web-pages within them [59, 60]. The increasing availability of such attack vectors, malware and other threats, is also strongly motivated by a worrying underground economy around them, which enables the attackers an easy monetization on vulnerable systems.

On the other hand, the introduction of machine learning algorithms in such applications raises itself an issue, namely, if these techniques are themselves secure [61]. In fact, they turn out to introduce specific vulnerabilities that skilled attackers can exploit to compromise the whole system, *i.e.*, machine learning itself can be the *weakest link* in the security chain. These threats have to be first identified, then the behavior of the system under attack can be analysed to eventually propose any possible countermeasures.

2.2.1 Attack Surface and Threat Model

In principle, an adversary may devise attacks at any stage of a machine learning based system (see Figure 2.11). For instance, an attacker may compromise the training set used to learn the decision function, by injecting carefully designed samples during the data acquisition phase [62]. Also, an attacker can mislead the data pre-processing step (*e.g.*,

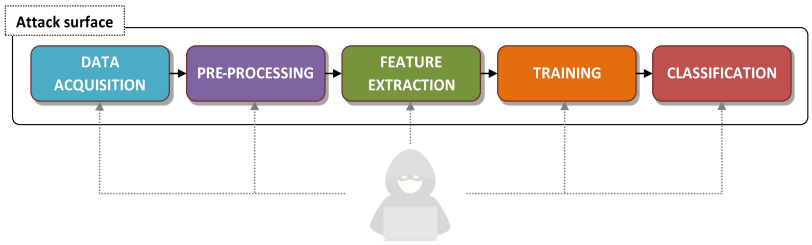


Figure 2.11: The attack surface of a typical machine learning based system.

in spam filtering, different techniques can be used to avoid correct parsing of e-mails), as well as the feature extraction processes (*e.g.*, samples may be altered to make the module or algorithm which performs feature extraction ineffective). In addition, an adversary may exploit some characteristics of the selected learning algorithm to design effective attacks at the operating phase. For example, a spammer may be able to understand which are the most relevant words used by a spam filter to classify legitimate e-mails, and use them to craft a more powerful attack, namely perform a *good word attack* [63, 64, 9].

All these possible security issues raise from the fact that machine learning techniques are not designed from the ground up to be attack-robust. In other words, they were not originally thought to operate in an *adversarial environment*. The design process should instead explicitly take into account that malicious adversaries can, at least in principle, attack the system at any stage. To this end, the designers should put themselves in the adversary's shoes and try to anticipate his attacks. So the first step to design secure pattern recognition systems is to model threats and thoroughly evaluate their security against the corresponding attacks. To this end, we describe in the following a framework based on the popular attack taxonomy proposed in [61, 65, 66] and subsequently extended in [20, 67, 68, 69, 70], which enables a designer to envision future attack scenarios, and to implement the corresponding attack strategies. In this context, we characterize the attackers by: (i) their objective in attacking the system; (ii) their knowledge of the system; (iii) their capabilities in influencing the system through manipulation of the input data [8].

Attacker's Goal

The adversary's goal or objective in attacking the system can be defined in terms of *security violation*, *attack specificity*, and *error specificity*.

Security Violation. The attacker may aim to cause: an *integrity* violation, to penetrate the system without compromising its normal operation (minimizing the chance of being detected); an *availability* violation, to compromise the normal system functionalities available to legitimate users; or a *privacy* violation, to obtain private information about the system or its users, usually by reverse-engineering the underlying machine learning model.

Attack Specificity. We distinguish between *targeted* and *indiscriminate* attacks, depending on whether the attacker aims to cause misclassification of a specific set of samples, or of any available data. The former attacks are usually performed to target a specific system user or protected service.

Error Specificity. If the adversary aims to have a sample misclassified as a precise class we consider the attack as *specific*; or *generic*, if the attacker aims to have a sample misclassified as any of the available classes excluding its true one. Other works [71] mix error and attack specificity by defining targeted and indiscriminate attacks depending on whether the attacker aims to cause specific or generic errors.

Attacker's Knowledge

We characterize the attacker's knowledge $\boldsymbol{\kappa}$ as a tuple in an abstract knowledge space $\mathcal{K}$ consisting of four main dimensions, respectively representing knowledge of: (k.i) the training data $\mathcal{D}_{\text{tr}}$; (k.ii) the feature set $\mathcal{F}$; (k.iii) the decision function f, along with the objective function (loss) $\mathcal{L}$ minimized during training; and (k.iv) the parameters $\mathbf{w}$ learned during the training process. This categorization enables the definition of many different threats, ranging from *white-box* attacks with full knowledge of the target classifier, to *gray-box* or *black-box* attacks in which the adversary has only limited or no information about the target system.

White-Box Perfect-Knowledge (PK) Attacks. We assume in this case that the attacker has full knowledge of the target classifier, *i.e.*, $\boldsymbol{\kappa} = (\mathcal{D}_{\text{tr}}, \mathcal{F}, f, \mathbf{w})$. This *white-box* setting allows one to perform a worst-case evaluation of the security of a machine learning algorithm, providing empirical upper bounds on the performance degradation expected under attack (see Section 2.2.4).

Gray-Box Limited-Knowledge (LK) Attacks. One may consider in this category different settings, depending on the attacker's knowledge about each of the components (k.i)-(k.iv). Typically, the attacker is assumed to know, at least, the input feature representation $\mathcal{F}$. For example, in case of images this means that he knows that the input features have the form of pixels' values, which is an assumption adopted in many recent works[72, 73]. At the same time, the attacker may know the form of the decision function f, *e.g.*, if the classifier is linear, or it is a neural network with a given architecture, etc., but neither the training data $\mathcal{D}_{\text{tr}}$ nor the classifier's parameters $\mathbf{w}$.

The attacker, however, may be able to collect a surrogate dataset $\hat{\mathcal{D}}_{\text{tr}}$, ideally sampled from the same underlying distribution of $\mathcal{D}_{\text{tr}}$, and train on such data a *surrogate model* $\hat{f}$ that approximates as close as possible the target function f. Then, the attacks can be crafted against $\hat{f}$ and *transferred* to the target classifier f [74, 72, 73] (see Section 2.2.6). By denoting the limited knowledge of a given component of the system with the *hat* symbol, these *gray-box* attacks can be defined as $\boldsymbol{\kappa} = (\hat{\mathcal{D}}_{\text{tr}}, \mathcal{F}, f, \hat{\mathbf{w}})$, or as $\boldsymbol{\kappa} = (\hat{\mathcal{D}}_{\text{tr}}, \mathcal{F}, \hat{f}, \hat{\mathbf{w}})$ if the type of the target classifier is unknown. It is worth remarking that surrogate models are used in the field of black-box mathematical optimization to find the optima of functions which are non-differentiable, nor analytically tractable. In these cases, the gradient information from a (differentiable) surrogate function $\hat{f}$ (that resembles the target f) can be leveraged to speed up the optimization process.

Black-Box Zero-Knowledge (ZK) Attacks. Classifiers working in an adversarial environment can also be threatened without any substantial knowledge of the feature space, the learning algorithm, or the training data, if the attacker can query the system in a *black-box* manner and get an output in terms of the predicted labels or the classification scores [73, 75, 76, 77, 78]. It should be noted however, that the attacker must at least know that the classifier is designed to perform some precise task (*e.g.*, identity recognition through fingerprint scanning, or malware classification), and must have a clear idea on how to alter the data to cause some feature changes, otherwise no variation of the outputs can be obtained, nor any useful information can be extracted from the system. For example, if one attacks a malware detector based on dynamic analysis by injecting static code which is never executed, there will be no impact at all on the classifier's decisions. This means that the attacker, to be effective against the system, must know at least (or has to get to know) which kind of features are used (*e.g.*, features based on static or dynamic analysis in malware detection). In other words, knowledge of the feature representation may be partial, but not completely absent. This is even more evident for algorithms trained on images, where the attacker generally knows if the input features are, *e.g.*, the pixels. Similar considerations hold regarding the knowledge of the training data. If the attacker knows that the system is used for a specific task, it may also know what kind of data has been used to learn its decision function. For example, if a deep network is trained to recognize road signs, then it is highly probable that the training set is constituted by images of such objects. Hence, also in this case, the attacker may not know the exact training set but, at least, should have some knowledge of the data used during the learning process.

We thus characterize this setting as $\boldsymbol{\kappa} = (\hat{\mathcal{D}}_{\text{tr}}, \hat{\mathcal{F}}, \hat{f}, \hat{\mathbf{w}})$. Even if surrogate classifier are not necessarily used when simulating these attacks, as described in [75, 76, 77, 78], as well as in pioneering work on black-box attacks [79, 80], one may learn a surrogate function anyway (potentially on a different feature representation) to evaluate the *transferability* of the attacks to the targeted classifier (see Section 2.2.6). Feedback from the classifier's decisions on carefully-crafted query samples can then be used to refine the surrogate model, as shown in [73].

Attacker Capability

This characteristic defines how the attacker can influence the system, and how the data can be manipulated based on application-specific constraints. If the adversary can manipulate both training and test data, the attack is said to be *causative*. It is instead referred to as *exploratory*, if the attacker can only manipulate test data. These scenarios are more commonly known as *evasion* [74, 11, 81, 82] (Section 2.2.2) and *poisoning* [83, 84, 85, 68, 86] (Section 2.2.3).

Another aspect related to the attacker's capability depends on the presence of application-specific constraints on data manipulation. For example, to attack a malware detector, malicious code has to be modified without compromising its intrusive functionality. This may be done against systems leveraging static code analysis, by injecting instructions that will be never executed [74, 87, 21, 88]. These constraints can be

generally accounted for in the definition of the optimal attack strategy, by assuming that an initial attack sample $\boldsymbol{x}'$ can only be changed according to a space of possible modifications, which we denote with $\Phi(\boldsymbol{x}')$, or by mapping the feature space $\mathcal{F}$ in terms of some value constraints, *e.g.*, by imposing that the feature corresponding to occurrences of some instructions can only be incremented [74, 87, 21].

Taxonomy of Adversarial Attacks

We provide in Table 2.1 a simplified categorization of the main attacks against machine learning algorithms based on the aforementioned threat model; in particular, considering the attacker's goal and his main capabilities. As mentioned before, the most common threats are evasion and poisoning availability attacks (aimed to maximize the test error). A different kind of poisoning integrity attacks which manipulate the training data to cause specific misclassifications have also been studied under the name of *backdoor* and *trojaning* attacks [89, 90]. These attacks maliciously manipulate pre-trained models to create subtle vulnerabilities which, when the corrupted models are publicly released, can be activated using specific input samples that are misclassified as desired, and thus allow gaining access to the system to achieve the malicious goal.

All the aforementioned attacks can be staged under different levels of the attacker's knowledge. When knowledge is limited or absent, as in the gray-box and black-box cases, privacy or confidential attacks can be performed to gain further knowledge about the target classifier or its users. A few practical examples of such threats include *model-extraction* attacks aimed to steal machine learning models, and *model-inversion* against biometric systems used to steal the face and fingerprint templates of their users (or any other sensitive information) [67, 75, 91, 92, 93, 94].

Table 2.1: Taxonomy of the attacks against machine learning based systems under the threat model described in Section 2.2.1.

	Attacker's Goal		
Attacker's Capability	*Integrity*	*Availability*	*Privacy*
Test data	Evasion (adversarial examples)	Sponge examples (to maximize energy consumption) [95]	Model stealing and model inversion
Training data	Poisoning (to allow future intrusion; *e.g.* backdoors, trojans)	Poisoning (to maximize classification error)	-

2.2.2 Evasion Attacks (Adversarial Examples)

Evasion attacks consist on manipulating input data to have them misclassified, *i.e.*, to evade a trained classifier at test time. These include, *e.g.*, altering images to mislead object recognition, adding specific words to an email to fool a spam filter, or manipulating malware code to have the corresponding sample misclassified as legitimate. A sample that has been modified to this end is often called an *adversarial example*. We denote the changes, *i.e.*, the perturbation, added to a sample to make it an attack input, with δ. In the following, we consider the formulation we reported in [20] for generic multiclass recognition problems, which extends the framework proposed in [74] for binary classifiers.

Two evasion settings, different by their error specificity, are conceptually depicted in Figure 2.12: *error-generic* and *error-specific*. In Chapter 4 we provide an optimization framework to solve both settings through a straightforward gradient-based attack, for differentiable learning algorithms (including neural networks, SVMs with differentiable kernels, etc.) [20, 74]. Non-differentiable learning algorithms, like decision trees and random forests, can be attacked with more complex strategies [96], or by using a differentiable surrogate classifier [97].

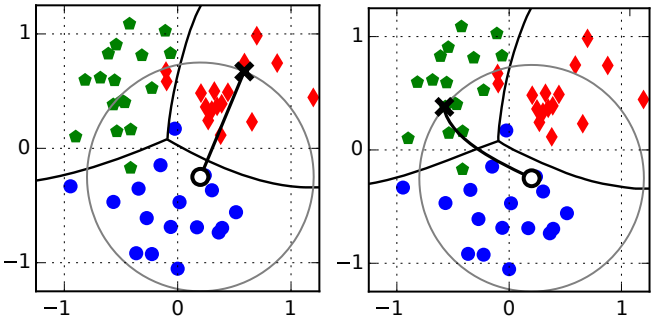


Figure 2.12: Examples of evasion attacks by error specificity, *error-generic* (left) and *error-specific* (right) [20]. Decision boundaries among the three classes (blue, red and green points) are shown as black lines. In the error-generic case, the initial sample (from blue class) is shifted towards the red class, as it is the closest class to the initial sample. In the error-specific case, instead, it is shifted towards the green class, as it is selected as target. The gray circle represents the feasible domain, given as an upper bound on the ℓ_2 distance between the initial and the adversarial example.

In the error-generic case, the attacker is interested in misleading classification, regardless of the predicted output class. The underlying idea behind this attack formulation, similarly to [98], is to ensure that an attack sample to which the system assigns the class ω_k, is rather misclassified as a sample of the closest candidate class, *i.e.*, the one exhibiting the highest value of the decision function among the remaining classes. In the error-specific setting instead, the attacker aims to mislead classification by requiring the adversarial example to be misclassified as an input from a specific class. The rationale in this case is to maximize the confidence assigned to the wrong target class $\omega \neq \omega_k$, while minimizing the probability of correct classification [98, 20].

Minimal Perturbation vs High Confidence Attacks In many works on evasion attacks, one of the main definitions states that *adversarial examples should be minimally perturbed*, meaning that they are modified as little as possible compared to the original sample, but enough to fool the classifier in the desired way. This *misconception* has taken hold in the community since the initial work by Szegedy *et al.* [11], where the notion of adversarial examples was introduced, which analyzed the instability of deep neural networks, *i.e.*, their sensitivity to minimal input perturbations, but not to perform a detailed security evaluation of the machine learning algorithms. Instead, it is more reasonable to assume that the attacker aims to maximize the classifier's confidence, *i.e.* the score, on the desired output class $\omega \neq \omega_k$, rather than only minimally perturbing the original samples. For this reason, while minimally-perturbed adversarial examples can be used to analyze the sensitivity of a learning algorithm, high- or maximum-confidence attacks are more suitable for an accurate security assessment of machine learning based systems under attack [8, 74, 69] (see Section 2.2.4), as well as to evaluate transferability across different models (see Chapter 5). More recent works by Carlini and Wagner [82, 99] and follow-up [100, 101], also showed that several defenses proposed against minimally-perturbed adversarial examples are vulnerable to maximum-confidence ones, which use a stronger perturbation. Figure 2.13 depicts

the difference between maximum-confidence and minimum-distance evasion attacks with respect to the decision boundary of the targeted classifier.

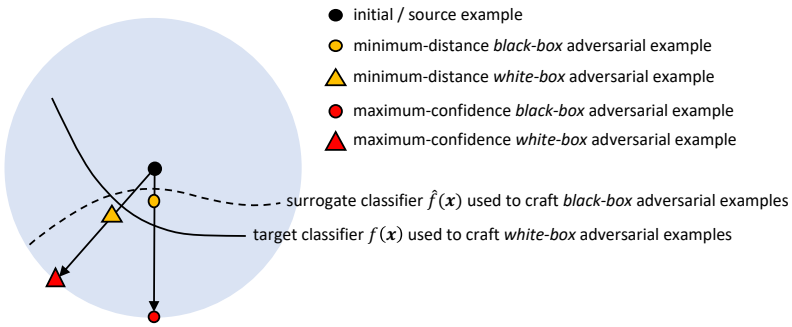


Figure 2.13: Conceptual representation of maximum-confidence evasion attacks vs. minimum-distance adversarial examples [16], within an ℓ_2 ball of radius ε. Maximum-confidence attacks tend to be more powerful as they are misclassified with higher confidence, even though are generated using a heavier perturbation.

2.2.3 Poisoning Attacks

Poisoning attacks consist on manipulating training data to either compromise normal system operation by causing a denial of service, or to favor intrusions without affecting normal system operation. The former are referred to as *poisoning availability attacks*, while the latter are known as *poisoning integrity attacks* [98, 21]. To this end, a small fraction of poisoning samples is injected into the training data, often by simply duplicating existing samples but with flipped label (assuming a binary classification problem). These attacks, conversely to evasion, are performed at the training phase. A conceptual example of how poisoning attacks work is given in Figure 2.14.

As for evasion attacks, we distinguish two settings by error specificity, namely error-generic and error-specific poisoning attacks, both in a perfect-knowledge scenario, given that the extension to gray-box and black-box is trivial through the use of surrogate learners [68]. In the error-generic case, the attacker aims to cause a denial of service by inducing the system to misclassify as many samples as possible, regardless of the classes in which they occur. Conversely, in the error-specific setting the attacker aims to cause specific misclassifications, usually to allow future intrusion (see Figure 2.15).

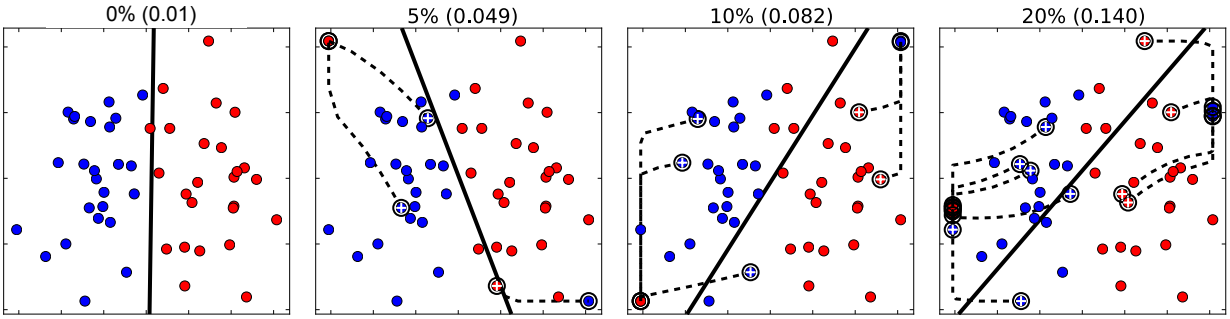


Figure 2.14: Conceptual example of poisoning attacks against a linear classifier [83, 84]. The training points (red and blue dots) and the decision boundary of the trained classifier (black solid line) are shown on each plot. The fraction of injected poisoning points w.r.t. the training set size is reported on top of each plot, along with the test error (in parentheses) of the resulting poisoned classifier. The attacks are initialized by cloning the legitimate points denoted with white crosses and flipping their label. The optimization process moves the samples to some local optima, following the black dashed trajectories, to obtain the final poisoning points (highlighted with black circles).

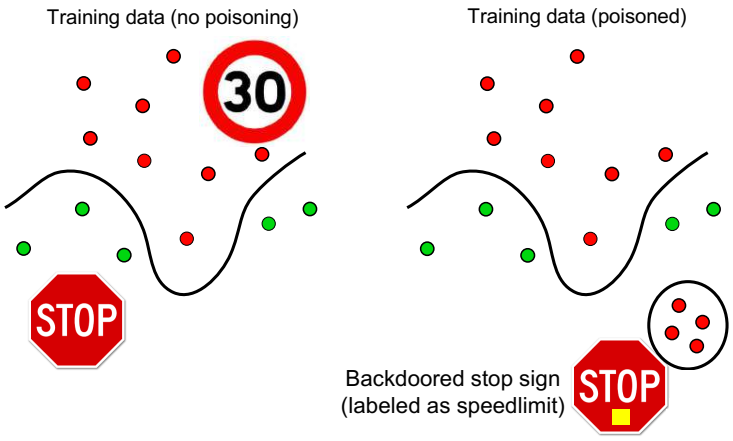


Figure 2.15: Conceptual representation of the impact of error-specific poisoning integrity attacks on the decision function of a trained model [89, 8]. The attacker places mislabeled training points in a region of the feature space far from the rest of data, leading the learning algorithm to consider them as legit samples. As a result, attack points are misclassified as desired by the adversary at test time, allowing a successful intrusion in the system.

Similarly to evasion attacks, in Chapter 4 we provide an optimization framework to craft poisoning attacks through a straightforward gradient-based algorithm, which allows solving a bi-level optimization problem by replacing the inner optimization with its equilibrium conditions [83, 84, 85, 68]. This enables gradient computation in closed form and, thus, the derivation of gradient-based attacks. However, gradient-based poisoning attacks are much more computationally demanding compared to their evasion counterpart, as they require retraining the classifier iteratively on the modified attack samples. This can be unpractical for advanced classifiers like deep neural networks, due to computational complexity and instability of the closed-form gradients. To tackle this issue, a more efficient technique has been recently proposed, named *back-gradient poisoning*, which relies on automatic differentiation and on reversing the learning procedure to compute the gradient [68].

2.2.4 Security Evaluation

Assessing the performance of a machine learning algorithm on a set of previously collected and labelled data, according to methodologies like the ones described in Section 2.1.2, may provide an optimistic estimate of the real performance of the system, especially when it tackles a security-related task. The reason behind this, is that the collected data either does not contain attack samples at all, or it does contain attack samples which are not targeted against the system being designed [102]. In other words, the performance of a classifier is typically assessed without taking into account its *robustness under attack*.

Instead, the existence of carefully crafted attacks aimed to evade the system should be considered during the design process and the performance evaluation. This includes analyzing the attacker's knowledge, capability or *strength*, and his possible goals (see Section 2.2.1). In fact, a system which is more robust under attack may be a better choice with respect to a system which is more accurate (according to the standard evaluation process), since the performance of the latter may decrease faster at the operating phase, as it gets targeted by the adversary's attacks.

To provide a thorough security evaluation of machine learning algorithms by taking into account the attacker's strength, one can vary the maximum amount ε of perturbation used to craft evasion attacks, or the number of poisoning attack points injected into the training data. The resulting

security evaluation curve, conceptually represented in Figure 2.16, shows the extent to which the performance of a learning algorithm drops more or less gracefully under attacks of increasing strength. This ensures that, *e.g.*, if the noise applied to the input data is within an ℓ_2 ball of radius ε, then the classification performance should drop no more than a certain value Δ. This is crucial to enable a fairer comparison among different attack algorithms and defenses [69, 70]. It should be noted that, by using minimally-perturbed adversarial examples (see Section 2.2.2), one can only provide guarantees against an average level of perturbation (rather than a worst-case bound).

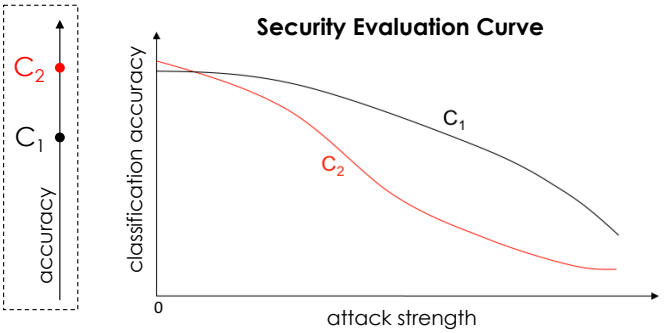


Figure 2.16: Security evaluation curve of two hypothetical classifiers C_1 and C_2 [8]. Accuracy of the C_2 classifier in absence of attack (zero strength) is higher then C_1 accuracy, and so it may appear as the best choice as a model for the ML-based system. However, simulating attacks of increasing (maximum) strength, the C_1 classifier is revealed to be more robust to adversarial input perturbations, and it's then a better choice from a security perspective.

2.2.5 Defenses against Adversarial Attacks

In this section we describe the *proactive* defenses aimed to prevent adversarial attacks to machine learning based systems. The majority of methods proposed so far in the literature can be categorized according to the paradigms of *security by obscurity* and *security by design* [8].

Security by obscurity. This category of defenses, also known as disinformation techniques [61, 65, 66], follow a paradigm where the information is hidden to the attacker with the goal of improving security. These defenses aim to counter gray-box and black-box attacks in particular, as a probing mechanism (*i.e.*, send multiple samples to the system to receive its output score or prediction) is often used to improve the surrogate models, or to refine evasion attempts by querying the target classifier. We categorize obscurity-based defenses as follows [103]: (*i*) randomization of training data collection (at different timings, and locations); (*ii*) use of classifiers that are difficult to reverse-engineer (*e.g.*, ensembles); (*iii*) access denial to the learned model or training data; and (*iv*) randomization of outputs to give variable feedback to the attacker. Regarding the latter approach, even if applied successfully to counter adversarial examples [104, 105, 106], it is still an open issue to understand whether and to which extent randomization may be used to make it harder for the attacker to learn a proper surrogate model, and to implement privacy-preserving mechanisms [107] against model inversion [67, 75, 91, 108, 93, 94]. Finally, gradient masking has been proposed to hide the gradient direction used when crafting adversarial examples with gradient-based algorithms [71, 109], but it has been shown that this can be easily circumvented with surrogate learners [73, 82, 74].

Security by design. This paradigm encourages the designer to devise a robust system from the ground up, namely, build an *adversary-aware* system. Based on this idea, several learning algorithms have been adapted to explicitly take into account the possible existence of different kinds of adversarial data manipulations. Moreover, this paradigm often assumes the worst-case white-box scenario, where the attacker has a perfect knowledge of the target system. In the following, we describe different techniques to design robust machine learning algorithms against evasion and poisoning adversarial attacks.

Defenses against Evasion Attacks

The first adversary-aware classification algorithm against evasion attacks has been proposed in 2004, which is based of simulating attacks and iteratively retrain the classifier on them [110]. More recently, similar techniques took the name of *adversarial training*, and employed to counter adversarial examples against deep neural networks [11, 81], or to harden particular classifiers like decision trees and random forests [96].

As the aforementioned techniques are all based on heuristics, with no formal guarantees on convergence and robustness properties, more structured approaches relying on game theory have been proposed, like zero-sum games to learn invariant transformations like feature insertion, deletion and rescaling [111, 112, 113]. Also, other works introduced Nash and Stackelberg games for secure learning derive formal conditions for existence and uniqueness of the game equilibrium under the assumption that each player knows everything about the opponents and the game [114, 115], or by randomizing players [106] and uncertainty on the players' strategies [116]. Despite these approaches seem promising, machine learning in an adversarial environment is not a board game with well-defined rules [8]. Understanding the extent to which the resulting attack strategies are representative of practical scenarios remains an open issue [117, 118]. Also, the scalability of these methods to large datasets and high-dimensional feature spaces is in doubt, as it may be too computationally costly to generate a sufficient number of attack samples to correctly represent the data distribution.

Robust optimization is a more computational efficient defense approach. It formulates machine learning in adversarial settings as a min-max problem in which the inner problem maximizes the training loss by manipulating the training points under worst-case, bounded perturbations, while the outer problem encourages the learning algorithm to minimize the corresponding worst-case training loss [81, 119, 120, 121]. The inner problem can be solved in closed form, for linear support vector machines, yielding a standard regularized loss formulation that penalizes the classifier parameters using the dual norm of the input noise [119], and for non-linear classifiers [122]. A direct result derived from these techniques is the equivalence between regularized learning problems and robust optimization, which has enabled approximating computationally-demanding secure learning models, like the aforementioned game-theoretical ones, with more efficient strategies based on the regularization of the objective function in a specific manner [21, 123, 97]. In fact, the main effect of these methods is to smooth out the decision function of the classifier, reducing the norm of the input gradients, and

thus making it less sensitive to worst-case input changes. To achieve this, few works proposed to improve the so-called *evenness* of the classifier's parameters [102, 124].

In [21] we proposed a more principled approach to smooth out the decision function of a machine learning model, derived from the idea of *bounding sensitivity* to feature changes. This starts with the definition of the classifier's sensitivity as:

$$\Delta f(\boldsymbol{x}, \boldsymbol{x}') = \frac{f(\boldsymbol{x}) - f(\boldsymbol{x}')}{\|\boldsymbol{x} - \boldsymbol{x}'\|_p} = \frac{\mathbf{w}^\top (\boldsymbol{x} - \boldsymbol{x}')}{\|\boldsymbol{x} - \boldsymbol{x}'\|_p} \ , \tag{2.13}$$

where $\|\cdot\|_p$ is the ℓ_p norm operator. This evaluates the decrease of f when a sample $\boldsymbol{x}$ is manipulated as $\boldsymbol{x}'$, with respect to the required amount of modifications, given by $\|\boldsymbol{x} - \boldsymbol{x}'\|_p$. Let us assume now, without loss of generality, that $\mathbf{w}$ has unary ℓ_1-norm and the features are normalized in $[0, 1]$.[6] We also assume that, for simplicity, the ℓ_1-norm is used to evaluate $\|\boldsymbol{x} - \boldsymbol{x}'\|_p$. Under these assumptions, it is not difficult to see that $\Delta f \in \left[\frac{1}{d}, 1\right]$, where the minimum is attained for equal absolute weight values, and the maximum is attained when only one weight is not null, confirming the intuition that more *evenly-distributed* parameters should improve classifier security under attack. This can also be formally shown by selecting $\boldsymbol{x}, \boldsymbol{x}'$ to maximize $\Delta f(\boldsymbol{x}, \boldsymbol{x}')$, which gives:

$$\Delta f(\boldsymbol{x}, \boldsymbol{x}') \leq \frac{1}{k} \sum_{k=1}^{k} |w^{(k)}| \leq \max_{j=1,\ldots,d} |w^{(j)}| = \|\mathbf{w}\|_\infty \ . \tag{2.14}$$

Here, $k = \|\boldsymbol{x} - \boldsymbol{x}'\|_1$ corresponds to the number of modified features and $|w^{(1)}|, \ldots, |w^{(d)}|$ denote the weights sorted in descending order of their absolute values, such that we have $|w^{(1)}| \geq \ldots \geq |w^{(d)}|$. The last inequality shows that, to minimize classifier sensitivity to feature changes, one can minimize the ℓ_∞-norm of $\mathbf{w}$. This tends to promote solutions which exhibit the same absolute weight values, a well-known effect of ℓ_∞ regularization [125], and are also more robust to evasion attacks. Moreover, the intuition behind more evenly-distributed parameters to improve security also led to the discovery of a strong *connection* between the robustness to adversarial examples and gradient-based explanations, as we discuss in Chapter 6.

Practically, to build a defense based on these principles, one can define, for example, a secure linear support vector machine by adding a box constraint on the weights $\mathbf{w}$, obtaining the following robust optimization algorithm, which we call Sec-SVM [21]:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \tfrac{1}{2}\mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^{n} \max\left(0, 1 - y_i f(\boldsymbol{x}_i)\right) \\ \text{s.t.} \quad & w_{lb}^{(k)} \leq w^{(k)} \leq w_{ub}^{(k)} \ , \ k = 1, \ldots, d \quad , \end{aligned} \tag{2.15}$$

where the lower and upper bounds on $\mathbf{w}$ are defined by the vectors $\mathbf{w}_{lb} = [w_{lb}^{(1)}, \ldots, w_{lb}^{(d)}]$ and $\mathbf{w}_{ub} = [w_{ub}^{(1)}, \ldots, w_{ub}^{(d)}]$, which are application dependent. As shown in Figure 2.17, the parameters learned by Sec-SVM have a lower absolute value compared to a standard support vector machine, making the resulting model more robust to evasion attacks.

Another line of defenses against evasion attacks is based on detecting and rejecting samples which are *sufficiently far* from the training data in feature space [20, 105, 126, 127, 128]. These samples are usually referred to

6: Note that this is always possible without affecting system performance, by dividing f by $\|\mathbf{w}\|_1$, and normalizing the feature values on a compact domain before the learning process.

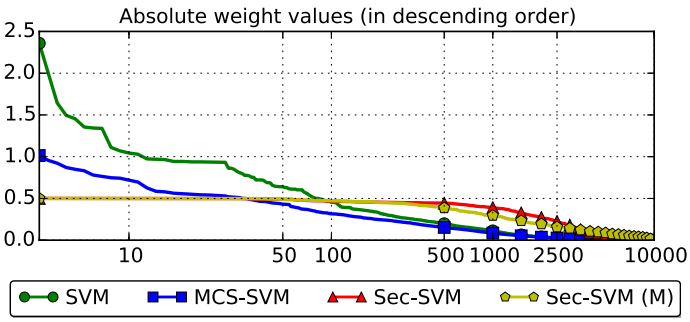


Figure 2.17: Comparison of the absolute values of the feature weights, in descending order (*i.e.*, $|w^{(1)}| \geq \ldots \geq |w^{(d)}|$), between two standard Support Vector Machines (SVM and MCS-SVM) and our robust Sec-SVM [21]. In the case of Sec-SVM (M) a different bound has been used for a subset of the feature space. Flatter curves correspond to more evenly-distributed feature weights and more secure classifiers.

as *blind-spot* evasion points, as they appear in regions of the feature space scarcely populated by training data, and can be countered by a *rejection* mechanism, *i.e.*, instructing the system to avoid any decision on them. In fact, this is a consequence of the stationarity assumption underlying many machine learning algorithms (according to which training and test data should come from the same distribution) [129, 130], and such rejection-based defenses simply aim to overcome this issue. In Figure 2.18, it is conceptually depicted a rejection-based defense applied to a support vector machine with RBF kernel [20]. We can observe how the decision boundaries are substantially changed after applying the defense (middle and right plots), with the stationary training classes tightly enclosed. This in turn may require one to trade-off between the security against potential attacks and the number of misclassified (stationary) samples at test time which may end up in the rejection region.

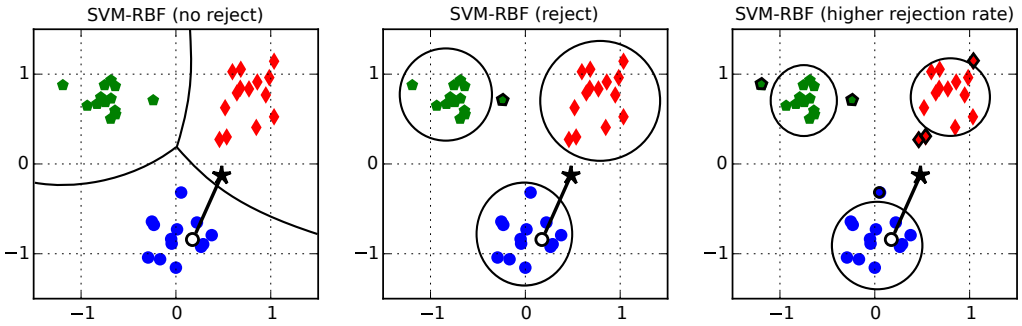


Figure 2.18: Effect of class-enclosing defenses against blind-spot adversarial examples on multiclass SVMs with RBF kernels, adapted from [20]. Rejected samples are highlighted with black contours. The attack (black star) is misclassified only by the standard SVM (left figure), while SVM with rejection correctly identifies it as an adversarial example (middle figure). Class enclosure can be tightened to improve classifier security (right figure), at the expense of misclassifying more legitimate samples.

It is worth remarking that, by using the aforementioned rejection-based algorithms, one can counter blind-spot evasion samples, but definitely not adversarial examples whose feature vectors become *indistinguishable* from those training samples belonging to the other classes. In the latter case, indeed, any learning algorithm would not be able to tell such samples apart [131]. In the case of deep convolutional neural networks, for example, most of the problems arise from the fact that the feature representation violates the smoothness assumption of learning algorithms: samples that are close in input space may be very far in the mapped deep space. To counter this vulnerability, one may re-train or re-engineer the deeper layers of the network (and not only the last ones) [11, 20].

Finally, classifier ensembles have also been exploited to improve the robustness against evasion attempts, *e.g.*, by implementing rejection-

based mechanisms or secure fusion rules [102, 60, 132, 127, 133, 134]. The underlying rationale in this case is that a Multi-Classifier System (MCS) can improve security since, at least in principle, it forces the adversary to evade more than one model. However, it may also worsen security if the base classifiers are not properly combined [60, 127].

Defenses against Poisoning Attacks

Compared to the number of works aimed to study methods to counter evasion attacks, only a few defenses against poisoning attacks have been proposed [135, 136, 127, 137, 138, 139, 140, 86]. This is mainly due to the fact that, to compromise a learning algorithm during the training process, an attacker must know the characteristics of the training samples and produce attacks that are different from the rest of the data (otherwise they would have no impact at all) [127]. As a result, poisoning attacks can be often detected as *outliers*, and countered using *data sanitization*, *i.e.*, attack detection and removal [127, 138, 140].

Another strategy to counter poisoning attacks is to employ robust techniques, where the learning algorithm is based on secure statistics that are intrinsically less sensitive to outlying training samples, *e.g.*, via bounded losses or employing kernel functions [136, 139, 86, 141, 142, 107]. Moreover, a few defenses to strengthen support vector machines have been proposed: Label Noise Robust SVM (LN-SVM) [143], where the classifier is heuristically enforced to increase the number of support vectors (enhancing its stability against outliers); and Least-Square SVM (LS-SVM) [144], where a quadratic loss function is used in place of the hinge loss to reduce the sparsity of the solution, by making all the training point have some weight in the prediction instead of just using the support vectors.

2.2.6 Transferability of Adversarial Attacks

Crafting poisoning and evasion attacks is not a trivial task and, generally, requires at least some knowledge of the system's architecture and the data which, in the real world, are rarely fully disclosed publicly. For example, a commercial machine learning based mobile malware detector can rely on a publicly known architecture, but use proprietary data collected from end hosts, and a mixture of known features (*e.g.*, system calls of an app), and undisclosed features (*e.g.*, reputation scores of the app). As a result, attackers are often forced to craft their attacks in black-box settings against a surrogate model $\hat{f}$, possibly built by querying the target model f [72], with no guarantee that the attack will be actually effective.

Unfortunately, several works demonstrated that adversarial attacks generated for one machine learning model may also successfully fool another model. Such property is referred to as *transferability*. This represents a serious threat for any machine learning based system, as it actually increases the potential strength of many existing adversarial attacks. Additionally, it should be noted that attacks may transfer not only *intra-technique*, *i.e.*, when both $\hat{f}$ and f are trained using the same algorithm (while, possibly, using different datasets or hyperparameters), but also *cross-technique*, when the surrogate and the target classifiers use different learning techniques [72].

Attack transferability of adversarial examples was first examined in [11] between different models trained over the same dataset, and between the same or different models trained over disjoint subsets of a dataset. Later, in [81] the phenomenon was attributed to the fact that adversarial perturbations are highly aligned with the parameters vector of a model. Additionally, [145] suggested that decision boundaries learned by the original and target models must be extremely close to one another for the attacks to successfully transfer.[7] With the advancement of machine learning based cloud services, many works studied the possibility of constructing a surrogate model by querying an online system to obtain predictions. In [73], black-box attacks are shown to be feasible towards machine learning services hosted by Amazon, Google, and MetaMind. Alongside, other works reported empirical findings about the transferability of evasion attacks [74, 101, 146, 145, 147]. Finally, [148] showed the existence of an *universal perturbation*, which is image-agnostic and can transfer with high probability across many different samples, making them adversarial examples without specific modifications.

7: In Chapter 8 we provide a similar insight after comparing the global explanations of different classification models.

Only recently, the transferability of poisoning integrity attacks has gained some attention [149], largely because existing black-box attacks have few shortcomings similar to white-box attacks, as they assume a threat model where the attacker has full control over the training process of the target classifier [150, 151, 152]. In fact, error specific poisoning attacks [135, 153, 90] require control of the labeling phase, which in the real-world is rarely part of the capabilities of an attacker. For example, consider the case where a malware creator provides to the system a poisoning sample to be added to the training set, but the learning process involves that the label, malicious or benign, is assigned by a human operator or a community voting process. These attacks can be easily detected by existing defenses, as they may stand out from the rest of the training set. Moreover, they also risk of causing collateral damage by accidentally lowering the accuracy on legitimate samples, which may alert the system's maintainer and thus the potential impact of the attack.

In spite of these efforts, the question of *when and why do adversarial attacks transfer remains largely unanswered*, including the *absence of reliable metrics* to evaluate the security of machine learning systems against these threats. In the next chapters we provide a formal definition of transferability and present the first comprehensive evaluation of this property for both evasion and poisoning availability attacks.

2.3 Explain Machine Learning Systems

Procedures like performance evaluation (Section 2.1.2) are necessary steps in the build process of an automated classification system, as no machine learning model is able to perfectly tackle all existing tasks (*no free lunch theorem* [32]), and the one that best suits the needs of the specific application should be carefully chosen. But then, if a ML-based system is able to provide the expected performance level, why do not just trust its decisions and leave it without monitoring? One reason is that these automated systems are shown to be vulnerable to attacks such as adversarial examples (see Section 2.2), which may lead to unexpected

decisions. In this case, a security evaluation procedure (see Section 2.2.4) can help identify the vulnerability of a model to such external attacks.

Unfortunately, performance and security metrics are an incomplete description of most real-world cases, especially for safety-critical applications [4]. Let's consider for example a self-driving car which has learned to perfectly recognize cyclists, but it suddenly fails when there is a side bag which partially covers one of the wheels; or a system which has to automatically accept or reject credit-cards transactions, which happens to block all operations made by users from a certain country. This type of unexpected situations may lead to the users asking for an *explanation* of the system's decisions, which requires the learned machine learning model to be *interpretable*. To interpret means to give or provide the meaning in understandable terms to a human [4]. Thus, an explanation should be a comprehensible interface between humans and the decision-maker, in this case, the machine learning based system [154]. Also, the explanation must be self-contained, *i.e.*, no further interpretation of it should be required. It will help assessing *safety*, *fairness* (by detecting biases), and *reliability* (by facilitate the debug process). Also, most automated systems do not guarantee *causality*, and thus explanations may help to highlight undesired spurious correlations between the input data and the output decision. Finally, while single performance or security metrics may only provide a quantitative measure of some specific system's property, explanations may also be useful for understanding previously unknown aspects of the learning algorithm, facilitating *scientific research* [5, 4].

Interpretability of a ML-based system is defined by the *transparency* of the learned model, which connotes the sense of understanding one may have of the working mechanisms. We distinguish between *simulatability*, which denotes transparency at the level of the entire model, and *decomposability*, when individual components are understandable [5]. In the former case, the user should be able to contemplate the entire model at once, *i.e.*, it can be readily presented to the user in visual or textual form [155]. This requirement mainly comes from the capacity of human cognition of processing only a limited amount of information [5]. In the latter case, each part of the model, *e.g.*, the feature space, the weights, the algorithm, should be intelligible [5, 156]. As we discuss in Section 2.3.1, however, while some well-known algorithms, like decision trees and linear models, are often considered to posses these properties by-design, it may be difficult to actually explain them depending on, *e.g.*, the number of features or the complexity of the learned rules. Also, for gray-box or black-box systems, when limited to no knowledge is available of the underlying mechanisms, a direct explanation of the model's behavior can be impossible to provide. In this case, additional techniques are used for interpretability, like post-hoc explanation methods (see Section 2.3.2).

Local Interpretability vs Global (Model) Interpretability

When it comes to explain a machine learning based system, one needs to identify what are the aspects of the system which need to be or can be interpreted [4]. In fact, properties like *safety* and *fairness*, *i.e.*, if the learned model generates decisions that are free from discrimination of specific inputs, especially when working with sensitive data, may be difficult to evaluate by only explaining the model's predictions. For example, a

system that makes automated medical diagnoses and a profanity filter for social media posts may both have learned biases against a certain ethnic group, even if the outputs take different forms in the two tasks. In other cases however, like for computer vision applications, it may be more useful to understand why the system made certain decisions by analyzing the components relevant for the outputs themselves.

In this context, we distinguish between *global interpretability*, which implies being able to understand the whole patters learned by a model and the working mechanisms that produce all the different possible decisions, and *local interpretability*, which implies knowing the reasons that lead to a specific prediction [4, 154]. As shown in Figure 2.19, in the former case explanations are provided to the user as an independent representation of the learned model as a whole (*global explanations*), while in the latter they are given as an additional information alongside a prediction (*local explanations*).

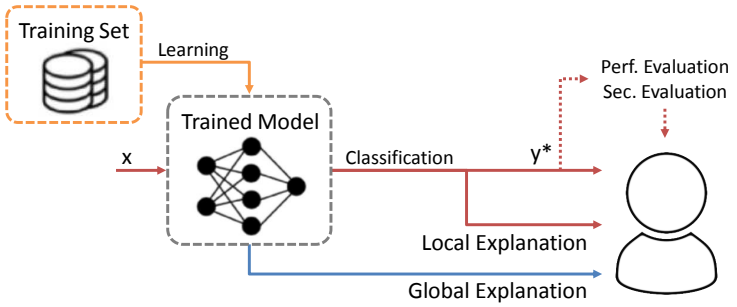


Figure 2.19: Architecture of an interpretable system. After the learning process from the training set is finished (orange), explanations may be provided to the user as an interpretation of the whole learned model (*global explanation*, blue), or during the classification process alongside a prediction y^* and, optionally, a performance or security metric (*local explanation*, red).

Most of the works in the current literature propose methods to produce explanations that achieve the latter, as it is arguably more difficult to describe a whole model compared to interpret its single predictions, especially if it is learned, from example, by complex algorithms like neural networks [5]. Conversely, global or model explanations are a research topic which has only been sparsely tackled in the past. Most works are focused in this case on producing *interpretable models*, which can be directly decomposed and simulated, and, thus, globally explained, or by fusing the local explanations computed on different samples to produce a global interpretation from the predictions [155, 17]. In Chapter 6 we describe a structured method to produce highly-interpretable model explanations for any machine learning based system, including black-boxes, which can be used to compare multiple models and also understand their intrinsic security properties, like adversarial robustness or the vulnerability to transfer attacks (see Section 2.2.6).

2.3.1 Interpretable Models

The easiest way to achieve interpretability is to purposely design systems based on a learning algorithm that, *i.e.* which predictions and behaviors are, *directly (human) understandable*. Among many others, decision trees, rule sets, and linear models are generally considered self-explaining algorithms, especially when applied to classification problems involving a limited number of features and classes.

Decision Trees and Rule Sets

Decision trees are classifiers that during the training process split multiple times the data into different subsets according to certain cutoff values in the features (*e.g.*, by checking if a feature has a value lower or greater than a threshold). The intermediate subsets are the split nodes, while the final ones are called terminal or leaf nodes, and represent a class label. To make a prediction, the input sample is evaluated through the tree until a leaf is reached, making its associated label the output class. The path followed from the root node to the leaf node to make a prediction is called *decision rule*. This classifiers are widely adopted in situations where the relation between the features and the outcome is non-linear, or when the features are not independent from each other.

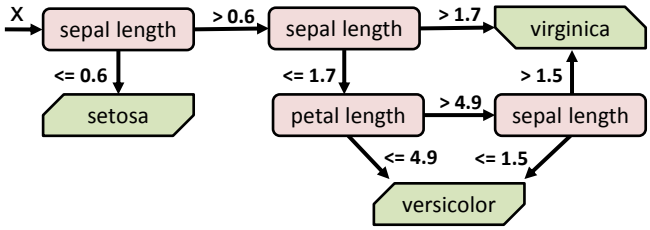


Figure 2.20: Example of a decision tree classifier for the 3-classes dataset *iris-flower* [27]. The decision rules are easily interpretable as they depend on the value of the features *sepal length* and *petal length*. For example, if the sepal length is ≤ 0.6, the predicted class for sample $\boldsymbol{x}$ will be *setosa*; otherwise if the sepal length is > 1.7, then *virginica* will be predicted.

Decision trees are often considered explainable by-design. Looking at Figure 2.20 for example, even a non-expert human can understand why the classifier makes a certain prediction given the value of the features of the input sample. Although, when the number of features or classes increases, or when the model is overfitting the training data, the depth of the tree can quickly grow, as more split nodes are added. The deeper the tree, the more difficult it becomes to interpret the decision rules [157]. Thus, generally, decision trees should not be considered explainable by design as they can lack *simulatability*. To address this issue, many works in literature proposed methods to post-process a tree classifier by simplifying its decision rules [158, 159].

Another possible solution to represent the patterns learned by a decision tree is to use *rule sets*. They are listings of ordered IF-THEN statements consisting of a condition (also called *antecedent*) and an *outcome*. To make a prediction, each rule should be evaluated in the order they are presented[8] until a `true` one is found, which outcome will be the predicted class. For example, the decision tree from Figure 2.20 can be represented using the following rules:

```
IF sepallength <= 0.6 THEN setosa
ELSE IF sepallength > 1.7 THEN virginica
ELSE IF petallength <= 4.9 THEN versicolor
ELSE IF sepallength <= 1.5 THEN versicolor
ELSE IF sepallength > 1.5 THEN virginica
```

All decision trees can be linearized into a set of rules using the IF-THEN form [160, 161], while keeping the expressivity of the former with the more compact representation of the latter. Also, the IF-THEN structure semantically resembles the natural language and the way a human generally thinks, making it more easily interpretable. Unfortunately, even for rule sets, there is no guarantee on *simulatability* as the number of

8: Other types of rule sets can be used: (i) unordered listings, where each rule is evaluated independently; (ii) *m-of-n* rules, where, given a set of n conditions, if m are verified, then the outcome of the rule is considered `true`.

conditions can quickly grow with the number of features or the possible outcomes [157]. Furthermore, to mitigate the cases where no rule applies for never-before-seen samples, a default condition can be introduced which will be a failover if no other rule applies.

Linear Models

As described in Section 2.1.1, a linear model predicts the class of an input sample $\boldsymbol{x}$ by computing a weighted sum of the features. As the learned relationship between the features and the output is linear, they are often considered highly-interpretable models. In fact, the value of each parameter in $\mathbf{w}$ represents the *relevance* of the corresponding feature towards a prediction [155]. If the value of a certain weight $\mathrm{w}^{(i)}$ is positive (negative), then the corresponding feature $x^{(i)}$ contributes by increasing (decreasing) the model's output by $\mathrm{w}^{(i)} \cdot x^{(i)}$. Also, if a feature has a higher contribution than another, it means that it has for the model a higher relevance on the prediction. As shown in Figure 2.21, by visualizing the sign and the magnitude of each weight, one can directly understand what the algorithm has learned from the training data (producing a *global explanation*). In addition, by plotting the contribution of each weight-feature pair for an input sample, one can directly explain why the model made a certain prediction and which are the most relevant components towards it (producing a *local explanation*).

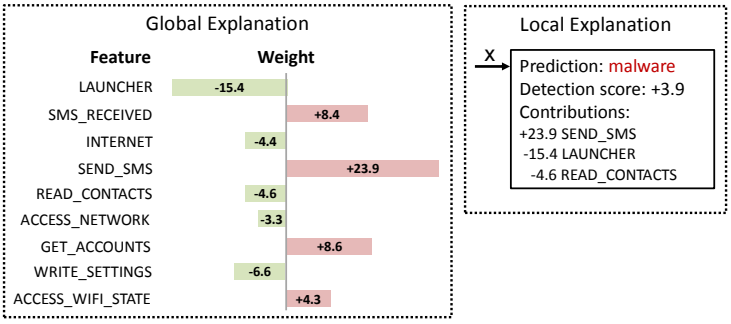


Figure 2.21: Conceptual example of weight-based feature importance global and local explanations for a malware detection system based on a linear model and Boolean features. We plot the sign and the magnitude of each weight, discovering that, for example, the `LAUNCHER` component has a negative value, meaning that the model learned that it defines benign samples. Conversely, the `SEND_SMS` permission is representative of malware, having a positive value. Given an input $\boldsymbol{x}$, we report the components present in the sample along with their contribution towards the prediction.

An intrinsic problem of linear models, however, is that they can quickly loose *decomposability* and *simulatability* [5]. Especially when the model does not optimally fit the training data, the interpretation of a specific weight can be counter-intuitive, because it may depend on all the other features. Thus, the correlation with a certain class outcome may be hard to interpret for a human. Also, similarly to decision trees and rule sets, linear models are actually directly interpretable only if the number of features is limited. To mitigate this, there are ways to encourage a more *sparse* solution, *i.e.*, where the weight associated to many features will be 0, for example by using in Equation 2.3 a ℓ_1-norm regularizer [3]. Alternatively, algorithms like support vector machines (see Section 2.1.1) generally learn a very sparse model which can be easier to interpret. Unfortunately, as highlighted in Section 2.2.5, models which use a limited number of features for the decision may be more vulnerable to adversarial attacks. The trade-off between interpretability and adversarial robustness is an integral part of a machine learning based system design process.

Generalized Additive Models

Recently, *Generalized Additive Models* (GAMs) [162] have been proposed as highly-interpretable yet accurate classifiers that may overcome the downsides of linear models. Conversely to the latter, in fact, where the relation between the weighted sum of the features and the output is linear, GAMs combine the sum of single-feature models called *shape functions* $h(\cdot)$ with a, possibly non-linear, *link function* $g(\cdot)$ to obtain the prediction y^*. The resulting expression is the following:

$$g(y^*) = \sum_{j=1}^{d} h_j(x^{(j)}) \ . \tag{2.16}$$

By visualizing the form of the shape functions, the actual contribution of each feature can be independently explained [156]. However, any link function that is not the identity complicates the interpretation, and complex non-linear shape functions can also be difficult to fully understand and decompose.

2.3.2 Post-hoc Explanations

Giving a proper explanation for all machine learning based systems could be challenging, especially because, in many different applications, may not be viable to only employ interpretable models, due to performance or security specifications, or because, as we described in the previous section, these algorithms may not actually provide the desired transparency. In addition, in case of gray-box or black-box systems where the learning function, its parameters and the training dataset are not or only partially know or, even worse, if the feature set is unspecified, it may be impossible to directly provide an interpretation of the system's behavior.

To tackle these cases, different *post-hoc* explanation methods have been proposed, which goal is to provide an interpretation for systems which algorithms are not necessarily explainable by their own, or which are partially or completely unknown. These methods can be considered as a way to reverse-engineer trained models in order to understand the information which has been acquired from the data [154]. Depending on which aspects of the system the interpretation focuses on, post-hoc methods may explain a model locally or globally.

In the following, we distinguish post-hoc explanation methods by how the information is presented to the human. We broadly consider three categories: rule- and tree- based, feature-based, and prototype-based explanations. Furthermore, we distinguish between methods that rely in approximating the decision function f of the learning system with a surrogate function $\hat{f}$, which may be easier to interpret, or can provide some properties that the original function does not have, *e.g.*, differentiability. Also, if the original system is gray-box or black-box, learning an interpretable approximation can often be the only viable method. In this latter case, the explanation methods are often called *agnostic*, as they are not tied to a particular type of function to approximate.

Tree- and Rule- based methods

A widespread type of techniques to provide explanations for a generic model, even black-boxes, is called *single-tree approximation*. They rely on training a comprehensible decision tree which accurately approximates the original model by only evaluating its predictions. Single-tree approximations were first presented in 1996 [163], based on maximizing the gain ratio of the information represented by neural networks. Other methods use *genetic programming* instead, to evolve a decision tree that mimics the behavior of a network [164, 165]. While these methods are often exclusively applied to neural networks, they can potentially be adopted as agnostic explanators, due to their exclusive reliance on the queried predictions.

Another commonly used solution is to determine a set of rules that accurately describes the behavior of the target system. A variety of methods have been proposed to this end, especially targeting neural networks [166]. Typically, they transform the rule extraction process, which is basically a search problem, into a learning one. If the prediction of the network to a certain input sample is not covered by the rules set, then a new *m-of-n* rule is added [163]. This ensures that all target classes have been covered. Conversely to decision tree -based techniques, these approaches are often strongly dependent on the type of classifier to explain and the specific form of rules used, thus are rarely generalizable to multiple cases. Other solutions are based on *Bayesian* frameworks which do not rely on the traditional greedy rules-finding methods [167, 168, 169, 170]. Worth mentioning, few works proposed methods to convert support vector machines into rule sets [171, 172].

We finally note that, when decision trees or rule sets are used for post-hoc explanation, the provided interpretations are almost always at global level. In fact, the methods described above try to generate the most compact and easier representation that allows to understand the whole model, following the notions of simulatability and decomposability [5]. To our knowledge, only a handful of methods provide local explanations using rule sets. One example is *Anchors*, which is based on computing a decision rule that ties a certain prediction locally in the feature space, such that similar instances covered by the same anchor have the same prediction outcome [173].

Feature-based methods

In many common pattern recognition applications the feature set is composed by human-understandable components, like the pixel values of an image, or by features to which a text description can be associated with, like the components of a mobile application (*e.g.*, Figure 2.21). Thus, many explainability methods try to associate a value to each feature, representative of the meaning that component has for the classifier. These set of values are called *attributions*, *saliency masks* [174], or *explanation vectors* [175]. Figure 2.22 shows different feature-based explanations for a handwritten digit from the MNIST dataset, obtained using a few of the methods described in this section.

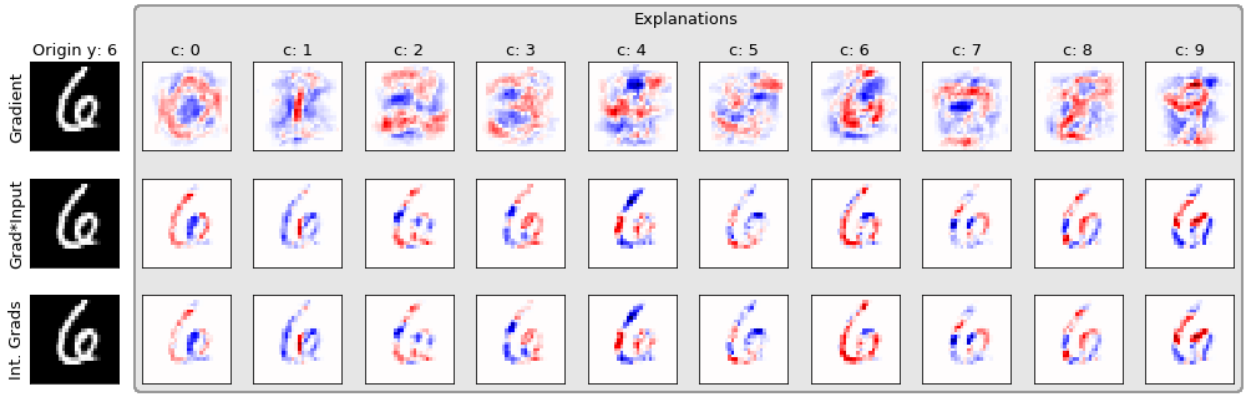


Figure 2.22: Local feature-based explanations of a handwritten digit (true label 6) from the MNIST dataset w.r.t. each different class, obtained using three different gradient-based explanation methods, namely, *Gradient* [175], *Gradient*Input* [176, 177, 17], and *Integrated Gradients* [178].

Many feature-based methods are derived from the computation of the gradient of the classifier's output, *i.e.*, the value of the discriminant function f, with respect to the input sample $\boldsymbol{x}$ [175, 174, 179, 176, 177, 178, 180, 181]. In this case, a non-null value of an attribution indicates that the corresponding feature is relevant for the class prediction of the input sample. The general rationale behind these approaches is that the sign of each element of the gradient indicates whether the classification score would increase or decrease when the corresponding feature is changed, and the absolute value of each element gives the amount of influence in the change of the score [175], which is similar to the intuition behind interpretable linear models (Section 2.3.1).

Gradient. The simplest method to obtain an explanation for the prediction on a sample $\boldsymbol{x}$ is by considering the gradient as it is [175]. For the j-th feature it is computed as:

$$\text{Gradient}_j(\boldsymbol{x}) := \frac{\partial f(\boldsymbol{x})}{\partial x^{(j)}} \quad . \tag{2.17}$$

By taking the attribution for each feature, one obtains the *explanation vector* [175] or, as later named, the *saliency map* [174] for the sample $\boldsymbol{x}$.

Arguably, if the decision function f is highly non-linear, it may be unclear how this provides accurate attributions for the input [174]. In fact however, the value of the decision function f computed on $\boldsymbol{x}$ can be approximated[9] using the Taylor expansion at some well-chosen baseline $\boldsymbol{x}_0$ [177]:

$$\begin{aligned} f(\boldsymbol{x}) &\approx f(\boldsymbol{x}_0) + \frac{\partial f}{\partial \boldsymbol{x}}\Big|_{x=x_0} \cdot (\boldsymbol{x} - \boldsymbol{x}_0) = \\ &= f(\boldsymbol{x}_0) + \sum_j \frac{\partial f}{\partial x^{(j)}}\Big|_{x=x_0} \cdot (x^{(j)} - x_0^{(j)}) \quad . \end{aligned} \tag{2.18}$$

To simplify this relation, it is possible to choose a baseline for which the score is equal or near zero, *i.e.*, $f(\boldsymbol{x}_0) \simeq 0$. As for most classifiers, including deep networks, the all-zeros vector $\mathbf{0}$ satisfies this property

9: Most of the higher-order terms in the expansion involves several features at the same time. Thus, for simplicity, only the first-order terms are considered for computing the attributions [177].

(especially on image recognition tasks) [178], Equation 2.18 becomes:

$$f(\boldsymbol{x}) \approx \sum_j \underbrace{\frac{\partial f(\boldsymbol{x})}{\partial x^{(j)}}}_{\text{Gradient}_j(\boldsymbol{x})} \cdot x^{(j)} \ , \tag{2.19}$$

showing how the gradient $\nabla f(\boldsymbol{x})$ defines the relevance of each feature in $\boldsymbol{x}$ for computing the classification score $f(\boldsymbol{x})$.

Gradient*Input. Other works [176, 177] showed that the saliency map for a certain input $\boldsymbol{x}$ can be substantially improved by considering the entire Equation 2.19 as an explanation, *i.e.*, by multiplying the gradient with the sample itself. Firstly proposed in [176] and utilized in our work [17] (see Section 6.1), the Gradient*Input method computes the attribution for the j-th feature as follows:

$$\text{Gradient*Input}_j(\boldsymbol{x}) := \frac{\partial f(\boldsymbol{x})}{\partial x^{(j)}} \cdot x^{(j)} \quad . \tag{2.20}$$

By projecting the gradient $\nabla f(\boldsymbol{x})$ onto $\boldsymbol{x}$, it ensures that the relevance is proportional to the input features. In mathematics, this notion takes the name of *directional derivative* [17].

Integrated Gradients. As an evolution of the Gradient*Input method, Sundararajan et al. [178] proposed an explanation technique that, instead of computing the gradient of the decision function only at the input sample $\boldsymbol{x}$, considers the straight-line path from the baseline $\boldsymbol{x}_0$ to the input sample, and computes the gradient at each point along that path. Once accumulated, this gradients define the explanation vector. Formally, the relevance along the j-th dimension for an input $\boldsymbol{x}$ and a baseline $\boldsymbol{x}_0$ is defined as:

$$\text{IntegratedGrads}_j(\boldsymbol{x}) := \left(x^{(j)} - x_0^{(i)}\right) \cdot \int_{\alpha=0}^{1} \frac{\partial f\left(\boldsymbol{x}_0 + \alpha \cdot (\boldsymbol{x} - \boldsymbol{x}_0)\right)}{\partial x^{(j)}} d\alpha \ . \tag{2.21}$$

To efficiently approximate the previous integral, one can add up the gradients computed at p fixed intervals along the path from $\boldsymbol{x}$ to $\boldsymbol{x}_0$:

$$\text{IntegratedGrads}_j^{\text{approx}}(\boldsymbol{x}) := \left(x^{(j)} - x_0^{(j)}\right) \cdot \sum_{k=1}^{\mathsf{p}} \frac{\partial f\left(\boldsymbol{x}_0 + \frac{k}{\mathsf{p}} \cdot (\boldsymbol{x} - \boldsymbol{x}_0)\right)}{\partial x^{(j)}} \cdot \frac{1}{\mathsf{p}} \ . \tag{2.22}$$

From Equation 2.22, it is clear that for classifiers which gradient of the decision function is independent from the input, like the linear models where $\partial f / \partial x^{(j)} = \mathrm{w}^{(j)}$, the Integrated Gradients method is in fact equivalent to Gradient*Input when the all-zeros vector $\boldsymbol{x}_0 = \mathbf{0}$ is chosen as baseline, which, as previously discussed, is a well-suited choice in many applications.

Layer-Wise Relevance Propagation. Another set of post-hoc explanation methods, specific for neural networks, are based on performing a backward pass from the output to the input layer, in order to *backpropagate* the effects of a decision on a certain sample up to the input level. This technique takes the name of Layer-wise Relevance Propagation (LRP) [179]. The basic idea is derived from a layer-wise conservation principle, which forces the propagated quantity, *e.g.*, the score of the sample w.r.t. the predicted class, to hold between consecutive layers. Supposing $R_l^{(j)}$ is the relevance of the j-th neuron at layer l, the conservation principle requires for the adjacent layer $l+1$:

$$\sum_j R_l^{(j)} = \sum_k R_{l+1}^{(k)} \ , \tag{2.23}$$

where k is the index of each neuron in the layer $l+1$ instead. By applying this condition through all the neural network, it results that the sum of the attributions in the pixel space is equal to the total relevance detected by the classifier for the input sample. In the first explanation method leveraging this technique, called Pixel-wise Decomposition (PWD) [179], the input images are encoded using a bag-of-world representation and the resulting saliency maps are referred to as *heatmaps*. Figure 2.23 shows a schematization of the process of producing a local explanation for a computer vision application using the PWD method.

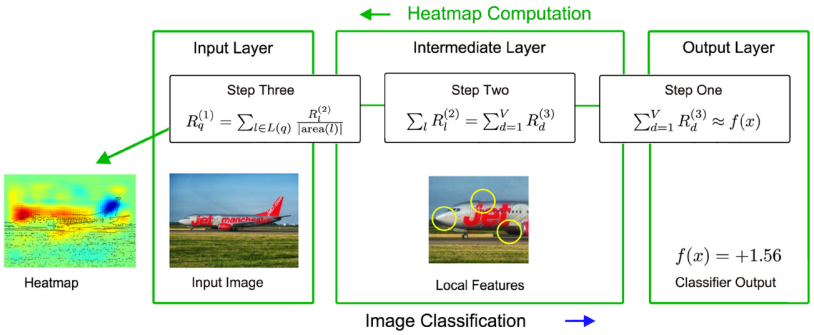


Figure 2.23: Process of computing a *heatmap* using the Pixel-wise Decomposition (PWD) method to locally explain a prediction in a computer vision application based on a neural network with three layers, adapted from [179]. After the classification phase, the score for the input sample is backpropagated through the network to obtain the final relevance map $R_q^{(1)}$ at the input layer. The layer-wise conservation principle ensures that the propagated quantity holds between consecutive layers, resulting in a heatmap which is equal to the total relevance detected by the classifier for the input image.

It is worth nothing that, when all activation functions are linear and the bias terms are included in the calculation, the LRP method is basically equivalent to Gradient*Input [182], and thus derived from Equation 2.18. In fact, an evolution of LRP is the Deep Taylor Decomposition (DTD) method [177], where the relevance is backpropagated from the top layer down to the input by applying the Taylor decomposition locally at each layer. Even in this case, the layer-wise conservation principle ensures that the relevance of the score for a certain sample is propagated intact to the bottom layer, producing an accurate heatmap for the input.

CAM, grad-CAM and grad-CAM++. Another line of research, strictly related to neural networks, embraces few explanation methods that produce saliency maps incorporating layers activations into the attributions. As firstly proposed in [183], the Class Activation Mapping (CAM) technique produces a local explanation for an input w.r.t. each output class, indicating the discriminative features that identify each label. This is achieved by feeding global average pooled convolutional feature maps

computed at the penultimate layer of the network into the top one. The Grad-CAM method generalizes CAM by allowing use of the technique in a wider range of CNN architectures, which may include fully-connected layers or provide structured outputs [180]. In this case, the attributions are obtained as a linear combination of the penultimate layer's feature maps and the output-specific weights. Finally, an extension of Grad-CAM named grad-CAM++ has been proposed to produce more accurate explanations when the relevance features for a specific class are scattered in the input sample [181].

Explanation of non-differentiable models. All gradient-based explanation methods work under the assumption that the decision function f is *differentiable* w.r.t the input $\boldsymbol{x}$, and that its gradient $\nabla f(\boldsymbol{x})$ is sufficiently smooth to provide meaningful information at each point. When $f(\boldsymbol{x})$ is not differentiable (*e.g.*, for decision trees and random forests), or its gradient vanishes (*e.g.*, if $f(\boldsymbol{x})$ becomes constant away from $\boldsymbol{x}$), one can approximate f by means of *surrogate models*, similarly to the procedure described in Section 2.2 to perform adversarial attacks when limited or no knowledge of the target system is available. In this case, a differentiable approximation $\hat{f}$ of the original function f is used in the standard gradient-based methods to compute the explanations [175, 155, 17].

LIME, LEMNA and SHAP. Inspired by the notion of surrogate models to explain hardly interpretable, non-differentiable, or black-box classifiers, few research works proposed *agnostic* methods that query the target system in the neighborhood of the sample to explain, producing a *local approximation* of the original decision function which can be used for explanation purposes.

In [155], the authors of LIME (Local Interpretable Model-agnostic Explanations) start by querying the target system using the synthetic samples $\mathcal{X} = \{\tilde{\boldsymbol{x}}_i, \dots, \tilde{\boldsymbol{x}}_l\}$, that are randomly generated in the vicinity of the input $\boldsymbol{x}$. Then, an interpretable model g is trained on these inputs to minimize the following objective:

$$\arg\min_{g} \sum_{\tilde{\boldsymbol{x}} \in \mathcal{X}} \pi_{\boldsymbol{x}}(\tilde{\boldsymbol{x}}) \cdot \big(f(\tilde{\boldsymbol{x}}) - g(\tilde{\boldsymbol{x}})\big)^2 + \Omega(g) \ , \tag{2.24}$$

where $\pi_{\boldsymbol{x}}$ is a proximity measure between $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$, and $\Omega(g)$ is a complexity function that encourages generation of simpler explanations (*e.g.*, the level of sparsity of the weights for linear models). While Equation 2.24 may accurately approximate a decision function which is sufficiently linear in the neighborhood of $\boldsymbol{x}$, it may be less accurate for more complex models like SVMs with non-linear kernels or neural networks.

A similar approach, called LEMNA [184], tries to overcome this limitation by employing a *mixture regression model*, which allows to locally approximate even non-linear decision boundaries by performing a weighted sum of K linear models. This linear combination of functions is furtherly enhanced by *fused lasso*, a penalty term which forces the solution to group related features together, producing more relevant explanations. Figure 2.24 shows how accurately a mixture regression model can approximate a complex decision function compared to the single linear regression used

by LIME. Formally, this techniques minimizes the following objective over n training samples:

$$\sum_{i=1}^{n} || \sum_{k=1}^{K} \pi_k(\boldsymbol{\beta}_k \boldsymbol{x}_i + \epsilon_k) - y_i || \tag{2.25}$$

$$\text{s.t.} \sum_{j=2}^{d} \left\| \beta_k^{(j)} - \beta_k^{(j-1)} \right\| \leq S, \; k = 1, \ldots, K \; , \tag{2.26}$$

where π_k and $\boldsymbol{\beta}_k$ represent, respectively, the weight and the regression coefficients of each linear model, and $\epsilon_k \sim N(0, \sigma)$ is a random perturbation variable that originates from a normal distribution. Finally, Equation 2.26 represents the fused lasso term that bounds the dissimilarity of the coefficients assigned to adjacent features within a small threshold S.

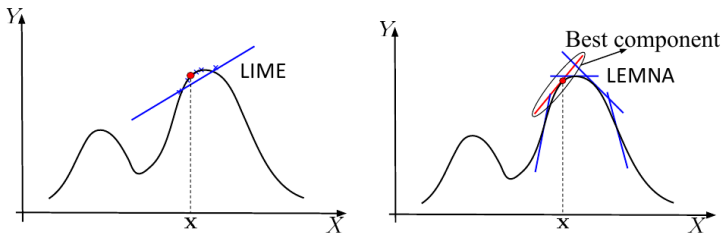


Figure 2.24: Comparison of the approximation procedure of a non-linear decision function, in the vicinity of sample $\boldsymbol{x}$, by a single linear regression (LIME) and a mixture regression model (LEMNA) [184].

SHAP (SHapley Additive exPlanations) [185] method takes a similar approach to LIME and LEMNA, where the explanations have the form of the, game theoretically optimal, Shapley values [186]. This coalitional game is designed so that the feature values of an input sample act as players in a coalition and the Shapley values represent how to fairly distribute the payout, *i.e.*, the classification score, among the features. Starting from Equation 2.24, to obtain Shapley compliant weighting, the $\pi_x(\tilde{\boldsymbol{x}})$ term is given by the SHAP Kernel:

$$\pi_x(\tilde{\boldsymbol{x}}) = \frac{M - 1}{\binom{M}{|\tilde{\boldsymbol{x}}|} |\tilde{\boldsymbol{x}}| \, (M - |\tilde{\boldsymbol{x}}|)} \; , \tag{2.27}$$

where M is the maximum coalition size and $|\tilde{\boldsymbol{x}}|$ is the number of non-null features in the instance $\tilde{\boldsymbol{x}}$. This variant of the SHAP method, called KernelSHAP, may be impractical to use when the Shapley values for many instances should be calculated, due to its computational complexity. To overcome this downside, few model-specific versions of SHAP have been proposed, including LinearSHAP (for linear models) or DeepSHAP (for deep neural networks) [185].

Prototype-based methods

The final category of post-hoc explanation methods we describe are based on selecting particular instances of the training dataset to explain the behavior of a machine learning model. Prototype-based explanations are particularly helpful as humans usually construct mental models in an *associative* or, more frequently, *contrastive* way [6], *i.e.* by relating an event to other similar or different ones that have been experienced in the past. In the same way, a machine learning based system can provide the set of samples which more strictly represent a certain information

acquired from the data (*e.g.*, the most related ones or the most un-related ones). In this context, *prototypes* are instances representative of all the dataset or some clusters of it, while a *criticism* is a data instance that is not well represented by any set of prototypes. The purpose of criticisms is to provide insights together with prototypes, especially for under-represented samples like data outliers. Nevertheless, these methods can only be used if the samples are represented in a humanly understandable way, like images or texts. Conversely, it may be unhelpful to present a sample as an explanation if, for example, it consists on tabular features or it is a list of the components in a mobile application, because the explanation may consist of hundreds or thousands of elements.

k-medoids. Arguably the simplest technique to find the prototypes for a dataset is called *k-medoids* [187], a clustering algorithm related to k-means where the centroids are always actual data points (while for k-means this is not guaranteed). This method however, like any other clustering algorithm, only finds the prototypes and not the criticisms, which are a fundamental part of the contrastive philosophy to prototype-based explanations.

Maximum Mean Discrepancy. In a recent work by Kim *et al.* [188], a technique that combines prototypes and criticisms in a single framework is presented under the name of MMD-critic. This method selects prototypes by minimizing the Maximum Mean Discrepancy (MMD) between the distribution of the data and the distribution of the selected samples. Specifically, data points in areas with high density are good prototypes, while samples from regions that are not well represented by the prototypes are chosen to be criticisms. To estimate the density of the different data distributions, a kernel function is used, like the Radial Basis Function (RBF). After minimizing MMD through a greedy search algorithm, a so-called *witness function* is computed, which identifies the portions of the input space that most misrepresents the dataset. This is achieved by measuring for each point the average proximity with the rest of the samples and with the set of prototypes. Finally, after choosing the two sets of relevant samples, one may provide global explanations of the machine learning model by computing the predictions for both the prototypes and criticisms and analyze in which cases is the algorithm right or wrong. An important drawback of this method is the arbitrariness of the choice of the number of prototypes and criticisms, which may be not enough to well represent the dataset, or be too high, and leading to explanations that are more difficult to analyze.

Influence functions. While an analysis of the dataset to find the most representative (or under-representative) samples may be helpful, to explain the behavior of a machine learning model using samples, it may be necessary to trace the predictions through the learning algorithm back to the training data, from which the parameters have been learned. To this end, similarly to feature-based explanation methods where each relevance value ultimately represents how the prediction score would change if the corresponding feature is changed, we may ask how the model would change if one or more training instances are removed from the training data during the learning process. Achieving this by

retraining the model multiple times after perturbing the data can be extremely expensive. Thus, Koh and Liang [189] proposed a method to compute the *influence functions*, which are a measure of how strongly a prediction depends on a certain training instance. The key idea is to modify a certain training instance $z \in \mathcal{D}_{\text{tr}}$ by an infinitesimally small step ϵ and compute how much the loss L computed using the new parameters $\hat{\mathbf{w}}$ consequently changes. Formally, the influence $\mathcal{I}(z, x)$ of each training sample z at a test point x can be computed as follows:

$$\mathcal{I}(z, x) = -\nabla_{\mathbf{w}} L(x, \hat{\mathbf{w}})^{\top} H_{\hat{\mathbf{w}}}^{-1} \nabla_{\mathbf{w}} L(z, \hat{\mathbf{w}}) \tag{2.28}$$

where $\nabla_{\mathbf{w}} L(\cdot, \hat{\mathbf{w}})$ is the loss gradient with respect to the parameters, and $H_{\hat{\mathbf{w}}}^{-1}$ is the inverse of the Hessian matrix, the second derivative of the loss with respect to the changed model parameters. Figure 2.25 shows an example of prototype-based explanations for two handwritten digits from the MNIST dataset obtained using the influence functions method. In this case, we can observe a direct correspondence between the most influential prototypes and the true class of the test samples, which should increase the trust in the predictions of the machine learning model.

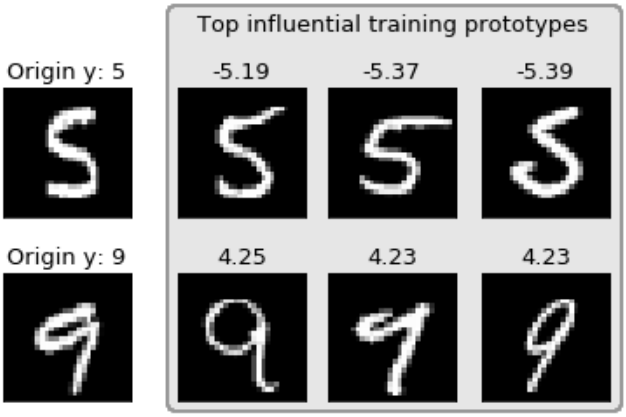


Figure 2.25: Examples of prototype-based explanations for two handwritten digits from the MNIST dataset classified by a binary SVM with RBF kernel, computed using the Koh and Liang method [189]. The influence value for each training prototype has a negative or positive sign depending on the corresponding class label. We can observe a direct correspondence between the most influential prototypes and the true class of each test sample, validating the information learned by the model.

As an interesting connection between prototype-based explanations and adversarial attacks, Koh and Liang demonstrated that influence functions can also be used to craft poisoning attacks that are minimally perturbed and can flip a model's prediction on a separate test image [189], which is a procedure mathematically equivalently to the label-flipping attacks described in [83] (see Section 2.2.3).

2.3.3 Evaluating Explanations

When it comes to perform a complete evaluation of a machine learning algorithm one may want to asses which are the methodologies and metrics that allow to objectively validate if some specific requirements are met. We described in Section 2.1.2, for example, that the prediction error (*i.e.*, accuracy) is one of these broadly used metrics. Comparing the accuracy obtained on a set of test samples with a minimum required level of precision, allows to establish if the learned model is satisfying enough for the given application. Similarly, evaluating the robustness to adversarial attacks is commonly achieved through the computation of security evaluation curves, which allow to fairly compare different models under the same assumptions of the attacker's knowledge and strength, as

described in Section 2.2.4. Regarding explanations however, there is little to no consensus in the scientific literature on how to evaluate the methods described in the previous sections, and how to quantify or measure how well they represent the behavior of the system to interpret.

In fact, due to the nature of interpretability itself, which is the ability to explain something to humans [4], it may be something that cannot be directly measured or quantified, as it relates to the different nature of the users of the system and their needs in each different application [190, 191]. Rather, interpretability should be considered a latent property that is influenced by multiple aspects of the machine learning model, *e.g.*, the number of features or the sparsity of the parameters, and has a direct impact on how the users perceive the working mechanisms, like their ability to simulate or debug the system's behavior and, more importantly, how they *trust* its automated decisions [190]. Thus, few works proposed structured experiments to evaluate explanations, which are based on human subjects challenged with appropriate and specific tasks.

Nevertheless, the lack of quantitative and objective measures to assess the quality of the explanations is problematic for engineers trying to choose amongst the different machine learning models depending on how secure or trustworthy they are [192]. For this reason, other works proposed well-defined criteria and metrics to compare explanation methods when applied to different models and pattern recognition tasks [193, 177, 184].

Later, in Chapter 6, we also discuss how explanation methods can be evaluated and compared in terms of the specific information they may highlight about a machine learning based system, including its security properties [18].

Human-centered evaluation

Many works proposed structured methodologies to evaluate the interpretability of a system based on the judgement of human subjects. As human beings have limited knowledge, abilities and available time, the main challenge of these approaches is to properly design the experiments so that the required tasks can be efficiently and effectively performed.

Arguably, the best approach to assess if an explanation is relevant and the corresponding system can be trusted, is by means of *application-grounded* evaluations, where domain experts judge the automated predictions and the obtained explanations, providing feedback such as identification of errors or biases, or discovering of new facts [4]. While finding humans with sufficient domain knowledge and enough willingness or available time could be challenging but achievable, the main drawback of these approaches is that they directly test the objective the system is built for, often obtaining strong but biased evidence of success.

Human-grounded evaluations are more appealing approaches when experimenting with a domain-specific community is challenging. In this case, human subjects are provided with simplified tasks to perform, which try to maintain the essence of the target system [4]. Generally, three cognitive tasks are part of these evaluations [4, 194, 190]:

- **Simulation**, where humans have to predict the system's decision after being presented with an explanation and a set of inputs. The performance on this task is usually computed as the absolute deviation between the model's prediction and the subject's guess;
- **Verification** or **Detection**, where humans have to judge the system's behavior given a set of inputs, the corresponding predictions and an explanation. Figure 2.26 shows an example of user interface for this type of evaluation task;
- **Counterfactual**, where humans decide how an input should be modified to make the system's decision change based on the provided explanation.

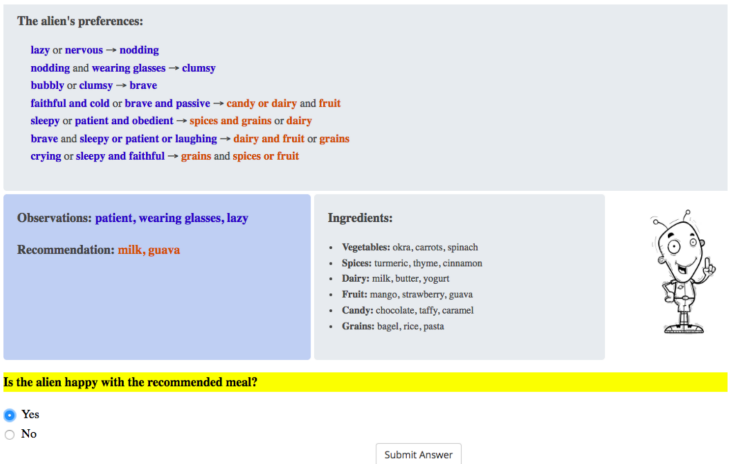


Figure 2.26: Example of user interface for a human-centered evaluation based on a verification task [194]. The subject is challenged to decide if the system's recommendation is correct, based on the features (observations) of the input sample and the rule-based explanations.

Since the scope of these experiments is twofold, *i.e.*, evaluate the system in terms of precision and user's trust, but also verify the consistency of the explanations, there are multiple ways in which the tasks described above can be varied to generate a comprehensive set of test cases. For example, the size of the explanations, *e.g.*, how many relevant features are displayed, greatly impacts the response time and the willingness of subjects [194]. Also, the model complexity, *e.g.*, the number of parameters, or how much knowledge of it the subject has, usually affects his ability to simulate or verify the system's predictions, but can also impact the percentage of detected mistakes on unusual samples. Specifically, humans are more likely to trust the decisions of a simulatable and known model, but may also be less prone to correct its inaccurate predictions, as their caution level is lower [190].

Finally, to measure how an explanation improves the credibility of the system, and thus the trust of the users in its decisions, Schmidt and Biessmann [192] proposed a test based on a metric called *Information Transfer Rate* (ITR), which can be independently applied to different models, tasks and explanation methods. ITR builds around the concept that the better an explanation is, the faster and more accurately an user will reproduce the decisions of the model. Formally, this is given by:

$$\text{ITR} = \frac{I(y^*_H, y^*_{ML})}{t} \ , \tag{2.29}$$

$$\text{with } I(y^*_H, y^*_{ML}) = \sum_{y^*_H, y^*_{ML}} p(y^*_H, y^*_{ML}) \log \frac{p(y^*_H, y^*_{ML})}{p(y^*_H) p(y^*_{ML})} \ ,$$

where y^*_H, y^*_{ML} are, respectively, the predictions of the human subject and

the machine learning model, $p(\cdot,\cdot)$ and $p(\cdot)$ are the joint and marginal probability functions, and t is the average response time of human subject. By computing ITR before and after showing the explanations to the user, one can measure how the trust in the system has improved. In addition, to capture if the humans are too biased towards the decisions of the automated system, *i.e.*, over-trust it, one can compute the *trust coefficient* $\bar{T}$ as follows:

$$\bar{T} = \frac{\mathrm{ITR}_{y^*_{ML}}}{\mathrm{ITR}_{y}} \quad , \tag{2.30}$$

where $\mathrm{ITR}_{y^*_{ML}}$ is given by Equation 2.29, and ITR_{y} is obtained by considering the true labels instead of the system's predictions. $\bar{T}$ should be small if the users are able to correctly recognize the system's mistakes, and big if they still agree with the system even if the decisions are wrong.

Metric-centered evaluation

Few works address the problem of systematically evaluate the quality of explanations and the trust of the users in the relative systems, by proposing different well-defined mathematical criteria. We broadly divide these techniques in two categories: *fidelity metrics*, which validate how the explanations accurately represent the learned model; and *interpretability metrics*, which give an indication on how capable the explanations are to provide insights into the system's behavior and improve its trustworthiness.

Fidelity metrics. To evaluate the consistency (fidelity) of a feature-based explanation method, one possibility is to consider the set of relevant features obtained from a sample of a certain class, as representative of the class itself. In this case, three different tests can be performed [184]: (*i*) *feature deduction*, (*ii*) *feature augmentation*, and (*iii*) *synthesization*. In case (*i*), if the explanation for sample $\boldsymbol{x}$ is correct, then removing the set of relevant features from $\boldsymbol{x}$ should lead the system to assign a different label to the sample. Similarly, in case (*ii*) adding the explanation for sample $\boldsymbol{x}$ to a sample $\boldsymbol{x}'$ from a different class, should induce the model to assign the latter the same label of $\boldsymbol{x}$. Finally, case (*iii*), a consistent explanation allows to synthesize a sample which only contains the relevant features and that will be recognized as a sample from the same class of $\boldsymbol{x}$.

A similar idea is proposed in [177] and [193], where a *pixel-flipping* or *descriptive accuracy* test is performed to measure the fidelity of an explanation. The rationale in this case is that a method that produces correct explanations, assigns relevance to the features that have the strongest impact on the decision function. Thus, removing the set of relevant features from $\boldsymbol{x}$ should lead to a sharp drop of the value of $f(\boldsymbol{x})$, measurable as a metric to compare different explanation methods.

Interpretability metrics. A different set of metrics allow to assess how successfully the explanations can be used to interpret the system's decisions or behavior, by providing a quantitative measure of some of the characteristics proper of interpretable models (Section 2.3.1), like simulatability or decomposability. As an example, a *descriptive sparsity* test allows to measure how large the set of relevant features is, by computing

a normalized histogram h of them, and calculate the *Mass Around Zero* (MAZ) [193], defined as:

$$\text{MAZ}(r) = \int_{-r}^{r} h(\boldsymbol{x})dx \ , \quad \text{for } r \in [0,1] \ . \tag{2.31}$$

Sparser explanations show a steep rise in MAZ close to 0, and are flatter around 1, consequence of a large set of non-relevant features.

Conversely, dense, uniform explanations are often obtained from systems which are more robust to adversarial attacks, as we describe and measure in Chapter 6 through means of different *evenness metrics* [18].

2.3.4 Explainable Machine Learning in Adversarial Environments

Explainable Machine Learning and Adversarial Machine Learning have historically raised almost independently. Although, in the recent years, many researchers started to hypothesize that a deep connection exists between explanations and adversarial attacks [12, 13, 14, 15, 195, 17, 18]. While in this thesis we mainly focus on how the process of designing a pattern recognition system which operates in an adversarial environment may directly benefit from the insights provided by explainability techniques, allowing robustness and other security-related properties (*e.g.*, attack transferability) to be evaluated through the analysis of explanations [17, 18], few works proposed to leverage explanations for both generating and detecting attack samples.

We previously discussed how, in the context of evasion attacks, the goal of the adversary is to manipulate input data so that it gets misclassified by the machine learning algorithm (see Section 2.2.2). Assuming that the attacker aims to maximize the classifier's error, he may also need to modify a set of features as small as possible, especially when altering particular types of samples like mobile applications, where any change of the components can potentially break the executable. Thus, the most impacting features for a specific sample towards the classifier's decision are usually modified first. Rosenberg *et al.* [195] proposed to leverage explainability algorithms, including Integrated Gradients, LRP, and SHAP (see Section 2.3.2), to intuitively obtain these most relevant features in a malware classification task. From this feature set, the easier to modify (*e.g.*, the static components) are selected to be used to craft the adversarial examples. It should be noted, however, that many older attack algorithms against linear models leverage a similar idea [74, 21, 16], where the features to modify are ranked by their (absolute) weight value. In fact, this is equivalent to computing the explanations using the Gradient method first (Equation 2.17), and then sorting the resulting features by their (absolute) relevance value. Despite this, more complex explainability techniques like IG, LRP or SHAP, may arguably lead to better results.

One of the underlying motivations behind the effectiveness of these feature-ranking methods is given by the work of Ilyas *et al.*, where it is argued that the existence of adversarial examples is actually an intrinsic property of the dataset itself [196]. The authors distinguish between *non-robust features*, which are highly predictive, yet very fragile and prone to change drastically even after a small perturbations of the input; and

robust features, which are both highly predictive and less subject to be influenced by small changes in the input. As non-robust features usually take a larger role in the learned model [196], this may explain why altering the most relevant components to produce evasion attacks has a higher impact on the target classifier.

Fidel *et al.* start from the same hypothesis to propose a highly accurate detector of adversarial examples, based on the SHAP values computed for the internal layers of a DNN classifier [14]. As different patterns in the importance of robust vs. non-robust features can be distinguished between normal and adversarial inputs, a supervised binary classifier can be trained to discriminate between the two. A major downside of this technique is that, in order to obtain a good generalization result, it requires a large training dataset of attacks crafted using various algorithms, which may be unfeasible to generate.

Unfortunately, a direct problem of including explanation techniques in the design process of a machine learning based system is that the attack surface (Figure 2.11) available to the adversary actually increases. For example, Dombrowski *et al.* show how saliency maps can be manipulated arbitrarily by applying perturbations to the input, while keeping the model's output approximately constant [197]. This is a worst case scenario where not only the prediction of the system is wrong (the perturbed malicious point is evading detection), but also the explanation which may be used to identify the vulnerability is compromised. An adversarial explanation attack $\boldsymbol{x}'$ can be obtained from an input $\boldsymbol{x}$ by optimizing:

$$\min_{x'} \|\boldsymbol{r}(\boldsymbol{x}') - \boldsymbol{r}^{\mathrm{t}}\|^2 + \gamma \|f(\boldsymbol{x}') - f(\boldsymbol{x})\| \ , \tag{2.32}$$

where $\boldsymbol{r}^{\mathrm{t}}$ is a target explanation produced by any of the post-hoc techniques described in Section 2.3.2, $\|\cdot\|$ is the ℓ_2 norm operator, and γ is a constant hyperparameter. An example of such attack is reported in Figure 2.27. As a counter-measure to increase robustness, a smoothing mechanism can be applied to the explanation method, basically leading to less noisy and less sensitive saliency maps.

Figure 2.27: Example of adversarial explanation attack [197]. The machine learning model predicts the same class for both the original and the manipulated image, while the saliency map corresponding to the latter shows a text stating "this explanation was manipulated".

State-of-the-art Limitations and Contributions of This Thesis

3

In the previous chapter we briefly described the state-of-art machine learning algorithms (Section 2.1) and discussed how an adversary can leverage any part of a pattern recognition system based on these automated decision-makers to perform different types of attacks, including the so-called evasion and poisoning attacks (Section 2.2). We also focused on many techniques for explaining the decisions of these systems, as ensuring the user's trust in them is a fundamental step before the adoption in any real-world application (Section 2.3).

One of the missing items in the current literature on machine learning algorithms applied in adversarial environments is a *common framework to encompass both evasion and poisoning attacks* targeting a specific learned model. In fact, most of the research works are focused on crafting either one of these threats. In addition, many of the proposed techniques are *hardly applicable to comprehensive threat models* (like the one described in Section 2.2.1), as they focus on specific malicious goals, and on specific levels of knowledge or capabilities available to the adversary.

Also, many security-critical properties of these attacks are only sparsely studied, like for example their *transferability between different models*. Worryingly, the question of when and why do adversarial attacks transfer remains mostly unanswered, mainly for the absence of a formal mathematical definition. In addition, the designers of machine learning based systems are currently missing a *reliable metric* to quantitatively evaluate the robustness of the learned models to transfer attacks, which may bring a massive benefit in terms of proactive defense to this threat.

However, transferability is not the only issue that should be evaluated during the design of a machine learning based system. The general *adversarial robustness* of a model, for example, is often analyzed mainly through means of the security evaluation curves. These are obtained by running *costly* simulations of the attacks, especially when advanced learning algorithms like neural networks are employed. Also, the *user's trust* in the system's decisions cannot be directly measured using the standard performance metrics, as it is strictly related to the ability of the system to clearly explain their outputs to the human. How to *compare the behavior of different systems* based on these types of user-dependent aspects is still an open question, mainly due to the strict connection between most explanation methods with the specific machine learning model they try to interpret (*e.g.*, its feature space or the hyperparameters).

Finally, when it comes to practically implementing a machine learning based system, the *Python programming language* has established itself as one of the most popular of the latest years, mainly due its nature of general-purpose platform for fast prototyping. However, despite the huge number of third-party libraries that allow a straightforward implementation of machine learning algorithms, only a few choices are available for evaluating these automated systems in an adversarial environment. Also, the issue of interpretability is only sparsely tackled in

the Python ecosystem, mainly with research code in support of scientific publications. As a result, engineers are often required to *integrate a plethora of very specific libraries* case-by-case for their application, slowing down the development process and the time to market.

3.1 Contributions

In this thesis, we first propose a *unified framework* for crafting evasion and poisoning attacks through a gradient-descent optimization procedure. This fully supports the threat model described in Section 2.2.1, including the different adversarial goals (integrity and availability), the available knowledge (white-box, gray-box, black-box), as well as the different adversarial capabilities (causative or exploratory). We derive and present the mathematical formulation required to craft both types of attacks, including the hypergradients that define the poisoning bi-level optimization problem.

Secondly, we provide a *formal definition of transferability* integrated with our unified attack framework, and present the first comprehensive evaluation of this property for both evasion and poisoning availability attacks. In particular, we discuss the intriguing connection among transferability, *input gradients* and *model complexity*, and we highlight the factors impacting transferability between a surrogate and a target model. We also provide *three reliable metrics* to measure the robustness of machine learning based systems to transfer attacks, allowing to compare the security of different models to these threats.

Afterward, we present a study on how to leverage gradient-based explanation methods to analyze and compare different machine learning based systems, in particular with respect to their security-related properties. We start by describing a novel method to compute local and global explanations based on *highly-interpretable relevance vectors*, also applicable to non-differentiable models or black-box systems. Our explanations not only allow to compare the behavior learned by the different models, but also allow to directly get an insight into the vulnerability to transfer attacks, without computing any additional specific metric. Then, motivated by the intuition that classifiers whose attributions are more evenly distributed should also be the more robust (as they rely on a broader set of features for the decision, see Section 2.2.5), we statistically investigate how gradient-based explanations may effectively be leveraged to provide a quantitative measure of the adversarial robustness of a machine learning model, without the need of running a full security evaluation. To this end, we propose few synthetic metrics that allow correlating the *uniformity* of the attributions with the *adversarial robustness*.

Finally, we also present `secml`, an open-source Python library that aims to provide engineers with all the tools necessary for developing and evaluating secure and explainable machine learning based systems without leveraging multiple third-party libraries. `secml` seamlessly integrates the most popular scientific libraries for machine learning, including *numpy*, *scipy*, *PyTorch*, and *TensorFlow*, within our unified framework to perform an empirical security evaluation against evasion and poisoning attacks,

and provides an easy-to-use implementation of many techniques for post-hoc local and global interpretation described in Section 2.3.2.

Outline of the next chapters

In the following chapters, we focus on each specific contribution of this thesis, as outlined below:

- Chapter 4: we describe a generalized optimization framework for crafting gradient-based evasion and poisoning attacks [16];
- Chapter 5: we provide a formal definition of transferability and different metrics to evaluate this property [16];
- Chapter 6: we discuss the possible connections between gradient-based explanations and adversarial robustness;
  - Section 6.1: we describe a highly-interpretable approach to explain and compare any machine learning model [17];
  - Section 6.2: we propose different quantitative metrics to statistically correlate the uniformity of explanations with adversarial robustness [18];
- Chapter 7: we describe `secml`, a Python library for secure and explainable machine learning [19];
- Chapter 8: we conduct an experimental investigation of the transferability of adversarial attacks on three different applicative cases, and we statistically evaluate the correlation between gradient-based explanations and the robustness to adversarial attacks using the metrics proposed in Chapter 6.

An Optimization Framework for Gradient-Based Adversarial Attacks

4

This chapter describes a unified framework for crafting evasion (see Section 2.2.2) and poisoning (see Section 2.2.3) adversarial attacks through a gradient-descent optimization procedure [16]. This generalizes existing attacks proposed by previous works for evasion [74, 11, 81, 82, 121] and poisoning [83, 85, 84, 189, 68, 86]. The framework supports the threat model described in Section 2.2.1, with the different adversarial goals (integrity and availability), amount of knowledge available to the adversary (white-box and black-box), as well as the different adversarial capabilities (causative or exploratory).

We start by providing a general projected-gradient solving algorithm (Section 4.1), and then we describe the specific procedures to craft evasion (Section 4.2) and poisoning (Section 4.3) attacks. The latter are more difficult to derive than evasion ones, as they require computing hyper-gradients for a bi-level optimization problem, to capture the dependency on how a machine learning model changes while the training poisoning points are modified [83, 85, 84, 189, 68, 86].

4.1 Gradient-based Optimization Algorithm

Given the attacker's knowledge $\boldsymbol{\kappa} \in \mathcal{K}$ (as defined in Section 2.2.1), and an attack sample $\boldsymbol{x}' \in \Phi(\boldsymbol{x})$ along with y, the true label of $\boldsymbol{x}$, the attacker's goal can be defined in terms of an objective function $\mathcal{A}(\boldsymbol{x}', y, \boldsymbol{\kappa}) \in \mathbb{R}$, which measures how effective the attack sample $\boldsymbol{x}'$ is on the target classifier (*e.g.*, how the classification loss changes under attack). Thus, the optimal attack point $\boldsymbol{x}^\star$ can be obtained as:

$$\boldsymbol{x}^\star \in \arg\max_{\boldsymbol{x}' \in \Phi(\boldsymbol{x})} \mathcal{A}(\boldsymbol{x}', y, \boldsymbol{\kappa}) \ . \tag{4.1}$$

Equation 4.1 only considers a single attack sample, but this formulation can be easily extended to account for multiple ones. In particular, the attacker can maximize the objective by optimizing one attack point at a time [8, 84].

Algorithm 1 provides a general projected gradient-ascent algorithm that can be used to solve Equation 4.1 for both evasion and poisoning attacks. It iteratively updates the attack sample $\boldsymbol{x}'$ along the gradient of the objective function $\mathcal{A}$, ensuring the resulting point to be within the feasible domain through a projection operator Π_Φ. The gradient step size η is determined at each iteration using a line-search procedure based on the bisection method, which solves:

$$\max_{\eta} \mathcal{A}(\boldsymbol{x}'(\eta), y, \boldsymbol{\kappa}), \ \text{ with } \ \boldsymbol{x}'(\eta) = \Pi_\Phi\left(\boldsymbol{x} + \eta \nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \boldsymbol{\kappa})\right) \ . \tag{4.2}$$

As Equation 4.2 requires the classifier decision function to be differentiable, non-differentiable learning algorithms like decision trees and random forests can be attacked using the gradient-based optimization against a differentiable surrogate learner [72, 97], or with more complex strategies [96, 198].

Algorithm 1: Gradient-based Evasion and Poisoning Attacks

input : $(\boldsymbol{x}, y)$, the input sample and its true label; $\mathcal{A}(\boldsymbol{x}, y, \boldsymbol{\kappa})$, the attacker's objective; $\boldsymbol{\kappa} = (\mathcal{D}, \mathcal{F}, f, \mathbf{w})$, the attacker's knowledge on the target model; Φ, the set of feasible manipulations that can be made on $\boldsymbol{x}$; $t > 0$, a small number.

output: $\boldsymbol{x}'$, the adversarial attack.

Initialize the attack sample: $\boldsymbol{x}' \leftarrow \boldsymbol{x}$;
repeat
  Store the attack from previous iteration: $\boldsymbol{x} \leftarrow \boldsymbol{x}'$;
  Choose the optimal step size η by solving Equation 4.2;
  Update the attack point: $\boldsymbol{x}' \leftarrow \Pi_{\Phi}\left(\boldsymbol{x} + \eta \nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \boldsymbol{\kappa})\right)$;
until $|\mathcal{A}(\boldsymbol{x}', y, \boldsymbol{\kappa}) - \mathcal{A}(\boldsymbol{x}, y, \boldsymbol{\kappa})| \leq t$;
return $\boldsymbol{x}'$

4.2 Crafting Evasion Attacks

As described in Section 2.2.2, the goal of an adversary in case of evasion attacks is to perturb a legitimate sample so that it gets wrongly labeled at test time. The intuition here is that he has to maximize the loss on the adversarial example w.r.t the original class, to cause misclassification to another class (*e.g.*, the opposite one in case of binary problems).

When full knowledge of the target model is acquired (including the classifier parameters $\mathbf{w}$), *i.e.*, white-box evasion, the optimization problem given in Equation 4.1 can be rewritten as:

$$\max_{\boldsymbol{x}'} \quad \mathcal{A}(\boldsymbol{x}', y, \mathbf{w}) \tag{4.3}$$

$$\text{s.t.} \quad \|\boldsymbol{x}' - \boldsymbol{x}\|_p \leq \varepsilon \ , \tag{4.4}$$

$$\boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{\text{ub}} \ , \tag{4.5}$$

where $\|\cdot\|_p$ is the ℓ_p norm operator. As demonstrated in [74], the classification score can be used as an objective by considering:

$$\mathcal{A}(\boldsymbol{x}', y, \mathbf{w}) = -y f(\boldsymbol{x}') \ . \tag{4.6}$$

Also, Equation 4.3 can be directly extended to the black-box case by leveraging the parameters $\hat{\mathbf{w}}$ of a surrogate classifier $\hat{f}$.

Manipulation constraints. The set of feasible manipulations Φ that can be made on the sample $\boldsymbol{x}$ are given in terms of: (4.4) a distance constraint $\|\boldsymbol{x}' - \boldsymbol{x}\|_p \leq \varepsilon$, which sets a bound on the maximum input perturbation between $\boldsymbol{x}$ and $\boldsymbol{x}'$; and (4.5) a box constraint $\boldsymbol{x}_{\text{lb}} \preceq \boldsymbol{x}' \preceq \boldsymbol{x}_{\text{ub}}$ (where $\boldsymbol{u} \preceq \boldsymbol{v}$ means that each element of $\boldsymbol{u}$ has to be not greater than the corresponding element in $\boldsymbol{v}$), which bounds the values of the attack sample $\boldsymbol{x}'$. For images, the former distance constraint is used to implement

either *dense* or *sparse* evasion attacks [123, 97, 20]. Normally, the ℓ_2 and the ℓ_∞ distances between pixel values are used to cause an indistinguishable image blurring effect (by slightly manipulating all pixels). Conversely, the ℓ_1 distance corresponds to a sparse attack in which only few pixels are significantly manipulated, yielding a salt-and-pepper noise effect on the image [123, 97]. Similarly, in case of attacks towards a malware detection system, ℓ_1 distance leads to changing only one component of the original application at each optimization step. The latter box constraint, instead, can be used to bound each feature inside a given range, *e.g.*, the pixel values between 0 and 255, or to ensure manipulation of only a specific region of the image. For example, if some pixels should not be manipulated, one can set the corresponding values of $\boldsymbol{x}_{\text{lb}}$ and $\boldsymbol{x}_{\text{ub}}$ equal to those of $\boldsymbol{x}$.

Initialization. When it comes to performing evasion attacks in both white-box and black-box settings, the choice of the start point $\boldsymbol{x}$ plays an important role in the effectiveness of the resulting adversarial example $\boldsymbol{x}'$. A good strategy to improve the solution found through Equation 4.3 consists on running the attack starting from different initialization points, thus mitigating the problem of getting stuck in poor local optima [74, 199, 101]. In addition, for non-linear classifiers one can also consider starting the gradient ascent from the projection of a randomly-chosen point of class $c \neq y$ onto the feasible domain. This double-initialization strategy helps finding better local optima, through the identification of more optimization paths towards evasion [16, 199, 101, 147].

4.3 Crafting Poisoning Attacks

By injecting adversarial points into the training set, an adversary can perform poisoning attacks (see Section 2.2.3). His goal in this case can be to either favor intrusions without affecting normal system operation, or to purposely compromise normal system operation to cause a denial of service. The former are referred to as integrity attacks, while the latter are known as availability attacks [8, 84]. Our framework focuses on poisoning availability attacks, as crafting integrity attacks has a much more modest goal of modifying prediction only for a small set of targeted points. The attacker's capability in this case is limited by assuming that he can inject only a fraction $\alpha \cdot \mathsf{n}$, with $\alpha \in (0, 1]$, of poisoning points into the training set of size n.

As for the evasion case, we provide the derivation of poisoning attacks in a white-box setting, given that the extension to black-box is immediate through the use of surrogate learners. The optimization problem is now bi-level, with the outer formula that maximizes the attacker's objective $\mathcal{A}$ (typically, a loss function L computed on m untainted samples), while the inner formula amounts to learning the classifier on the poisoned training data [83, 84, 85]. By rewriting Equation 4.1 accordingly, we obtain:

$$\max_{\boldsymbol{x}'} \quad L(\mathcal{D}_{\text{val}}, \mathbf{w}^\star) = \sum_{j=1}^{\mathsf{m}} \mathcal{A}(\boldsymbol{x}_j, y_j, \mathbf{w}^\star) \ , \tag{4.7}$$

$$\text{s.t.} \quad \mathbf{w}^\star \in \arg\min_{\mathbf{w}} \mathcal{L}(\mathcal{D}_{\text{tr}} \cup (\boldsymbol{x}', y), \mathbf{w}) \ , \tag{4.8}$$

where $\mathcal{D}_{\text{tr}}$ and $\mathcal{D}_{\text{val}}$ are the training and validation datasets available to the attacker. The former, along with the poisoning point $\boldsymbol{x}'$, is used to train the learner on poisoned data (minimizing its training loss $\mathscr{L}$), while the latter is used to evaluate its performance on untainted data, through the loss function $L(\mathcal{D}_{\text{val}}, \mathbf{w}^\star)$. Notably, the objective function $\mathscr{A}$ implicitly depends on $\boldsymbol{x}'$ through the parameters $\mathbf{w}^\star$ of the poisoned classifier.

4.3.1 Gradient of validation loss

While poisoning points can be easily optimized via projected-gradient procedures like Algorithm 1, the main challenge with solving Equation 4.7 is computing the gradient of the validation loss L with respect to each poisoning point. In fact, this gradient has to capture the implicit dependency of the optimal parameters vector $\mathbf{w}^\star$ (learned after training) on the poisoning point being optimized, as the classification function changes while this point is updated.

Provided that the attacker objective $\mathscr{A}$ is differentiable w.r.t. $\mathbf{w}$ and $\boldsymbol{x}$, its gradient can be computed using the chain rule [83, 84, 68, 8, 85]:

$$\nabla_x \mathscr{A} = \nabla_x L + \frac{\partial \mathbf{w}}{\partial \boldsymbol{x}}^\top \nabla_{\mathbf{w}} L \ , \tag{4.9}$$

where the term $\frac{\partial \mathbf{w}}{\partial \boldsymbol{x}}$ captures the implicit dependency of the parameters $\mathbf{w}$ on the poisoning point $\boldsymbol{x}$. Under some regularity conditions, this derivative can be obtained by replacing the inner optimization problem with its stationarity Karush-Kuhn-Tucker (KKT) conditions, *i.e.*, with its implicit equation $\nabla_{\mathbf{w}} \mathscr{L}(\mathcal{D}_{\text{tr}} \cup (\boldsymbol{x}', y), \mathbf{w}) = \mathbf{0}$ [85, 68].[1] By differentiating this expression w.r.t. the poisoning point $\boldsymbol{x}$, one yields:

$$\nabla_x \nabla_{\mathbf{w}} \mathscr{L} + \frac{\partial \mathbf{w}}{\partial \boldsymbol{x}}^\top \nabla^2_{\mathbf{w}} \mathscr{L} = \mathbf{0} \ . \tag{4.10}$$

Finally, rearranging Equation 4.10, we obtain $\frac{\partial \mathbf{w}}{\partial \boldsymbol{x}}^\top = -(\nabla_x \nabla_{\mathbf{w}} \mathscr{L})(\nabla^2_{\mathbf{w}} \mathscr{L})^{-1}$, which can be substituted in Equation 4.9 to obtain the required gradient:

$$\nabla_x \mathscr{A} = \nabla_x L - (\nabla_x \nabla_{\mathbf{w}} \mathscr{L})(\nabla^2_{\mathbf{w}} \mathscr{L})^{-1} \nabla_{\mathbf{w}} L \ . \tag{4.11}$$

Depending on the type of the target classifier, Equation 4.11 can be computed using a non-approximated closed form. In the following, we report the derivation for both support vector machines, as already proposed in [83], and for logistic regression classifiers.

1: More rigorously, one can write the KKT conditions in this case as $\nabla_{\mathbf{w}} \mathscr{L}(\mathcal{D}_{\text{tr}} \cup (\boldsymbol{x}', y), \mathbf{w}) \in \mathbf{0}$, given that the solution may not be unique.

Gradients for Support Vector Machines

By considering $\mathscr{L}$ as the dual form learning problem of a support vector machine (see Equation 2.5), and L as the hinge loss (in the outer optimization), Equation 4.11 becomes:

$$\nabla_x \mathscr{A} = -\alpha_c \frac{\partial \boldsymbol{k}_{kc}}{\partial \boldsymbol{x}_c} \boldsymbol{y}_k + \alpha_c \begin{bmatrix} \frac{\partial \boldsymbol{k}_{sc}}{\partial \boldsymbol{x}} & 0 \end{bmatrix} \begin{bmatrix} \mathbf{K}_{ss} & \mathbf{1} \\ \mathbf{1}^\top & 0 \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{K}_{sk} \\ \mathbf{1}^\top \end{bmatrix} \boldsymbol{y}_k \ . \tag{4.12}$$

Here we use c, s, and k to index, respectively, the attack point, the support vectors, and the validation points for which $\mathscr{A}(\boldsymbol{x}, y, \mathbf{w}) > 0$ (corresponding to a non-null derivative of the hinge loss). The coefficient α_c is the dual parameter assigned to the attack point by the learning algorithm, and (vector) $\boldsymbol{k}$ and (matrix) $\mathbf{K}$ contain the kernel values between the corresponding indexed sets of points.

Gradients for Logistic Regression

Logistic regression is a linear classifier that estimates the probability of the positive class using the sigmoid function σ (Equation 2.11). A derivation of poisoning attacks against this algorithm has been already proposed in [85], yet by maximizing a different outer objective and not the validation loss directly. The following is a novel formulation that computes the gradients under our general optimization framework.

Using the logistic loss as the attacker's loss, Equation 4.11 for logistic regression can be computed as:

$$\nabla_{\boldsymbol{x}} \mathscr{A} = -\begin{bmatrix} \nabla_{\boldsymbol{x}} \nabla_{\mathbf{w}} \mathscr{L} \\ C\ z_c\ \mathbf{w} \end{bmatrix}^{\top} \begin{bmatrix} \nabla^2_{\mathbf{w}} \mathscr{L} & \mathbf{X}\ \boldsymbol{z}\ C \\ C\ \boldsymbol{z}\ \mathbf{X} & C \sum_i^n z_i \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{X}(\boldsymbol{y} \circ \boldsymbol{\sigma} - \boldsymbol{y}) \\ \boldsymbol{y}^{\top}(\boldsymbol{\sigma} - 1) \end{bmatrix} C \ , \tag{4.13}$$

where $\mathbf{X}$ and $\boldsymbol{y}$ are, respectively, the matrix of the training samples and the vector of their true labels, $\mathbf{w}$ are the classifier weights, and $\circ$ is the element-wise product. $\boldsymbol{z} = \boldsymbol{\sigma}(1 - \boldsymbol{\sigma})$ includes the contribution of the signed decision function, with each element given by:

$$\sigma_i = \frac{1}{1 + e^{-y_i f_i(\boldsymbol{x}_i)}} \ . \tag{4.14}$$

Finally, the derivatives of the training loss $\mathscr{L}$ in Equation 4.13 are:

$$\nabla^2_{\mathbf{w}} \mathscr{L} = C \sum_i^n \boldsymbol{x}_i z_i \boldsymbol{x}_i^{\top} + \mathbb{1} \ , \tag{4.15}$$

$$\nabla_{\boldsymbol{x}} \nabla_{\mathbf{w}} \mathscr{L} = C(\mathbb{1} \circ (y_c \sigma_c - y_c) + z_c \mathbf{w} \boldsymbol{x}^{\top}) \ , \tag{4.16}$$

with $\mathbb{1}$ denoting the identity matrix.

Transferability of Adversarial Attacks

5

Transferability captures the ability of an attack against a machine learning model to be effective against a different one, potentially unknown. While previous works have reported empirical findings about this property of adversarial attacks in both evasion [74, 101, 81, 146, 148, 73, 72, 11, 145, 147] and, limitedly, poisoning integrity [149] cases, the question of when and why do adversarial attacks transfer remains largely unanswered, including the absence of reliable metrics to evaluate the security of machine learning based systems against these threats.

In this chapter [16] we start by providing a formal definition of transferability in case of evasion and poisoning availability attacks (Section 5.1). Then, we discuss the intriguing connection among transferability, input gradients and model complexity, and highlight the factors impacting transferability between a surrogate and a target model (Section 5.2).

It is important to emphasise right from the start, that model complexity is a measure of the capacity of a learning algorithm to fit the training data. It is typically penalized to avoid overfitting by reducing either the number of classifier parameters to be learnt or their size (*e.g.*, via regularization) [3]. Given that complexity is essentially controlled by the hyperparameters (*e.g.*, the number of neurons in the hidden layers of a neural network, or the regularization hyperparameter C of a SVM), *only models that are trained using the same learning algorithm should be compared in terms of complexity*. As we later discuss, this is an important point to correctly interpret the results of our analysis.

Afterward, in Chapter 8, we provide a comprehensive experimental evaluation of transferability by considering three different application cases: (i) handwritten digit recognition; (ii) Android malware detection; (iii) identity verification via face recognition.

Notation. For convenience, in this chapter we denote an adversarial (attack) sample as $\hat{\boldsymbol{x}} = \boldsymbol{x} + \hat{\boldsymbol{\delta}}$, where $\boldsymbol{x}$ is the initial point, and $\hat{\boldsymbol{\delta}}$ is the adversarial perturbation optimized by the attack algorithm against the *surrogate* classifier $\hat{f}$, for both evasion and poisoning attacks.

5.1 Formal Definition

In the following we provide a formal definition of transferability in case of evasion and poisoning availability attacks.

5.1.1 Evasion Attacks

Given an attack point $\hat{\boldsymbol{x}}$, crafted against a surrogate learner $\hat{f}$ (parameterized by $\hat{\mathbf{w}}$), we define its *transferability* as the attack loss attained by the target classifier f (parameterized by $\mathbf{w}$) on that point, *i.e.*, $T = \mathcal{A}(\boldsymbol{x}+\hat{\boldsymbol{\delta}}, y, \mathbf{w})$. This can be linearly approximated for sufficiently-small input perturbations as:

$$T = \mathcal{A}(\boldsymbol{x} + \hat{\boldsymbol{\delta}}, y, \mathbf{w}) \approxeq \mathcal{A}(\boldsymbol{x}, y, \mathbf{w}) + \hat{\boldsymbol{\delta}}^\top \nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \mathbf{w}) \ . \tag{5.1}$$

Equation 5.1 may also hold for larger perturbations if the classification function is linear or has a small curvature (*e.g.*, if it is strongly regularized).

Under this linear approximation, for any given point $\boldsymbol{x}$, the evasion problem in Equation 4.3 (without considering the feature bounds in Equation 4.5) can be rewritten as:

$$\hat{\boldsymbol{\delta}} \in \arg\max_{\|\boldsymbol{\delta}\|_\mathsf{p} \leq \varepsilon} \mathcal{A}(\boldsymbol{x} + \boldsymbol{\delta}, y, \hat{\mathbf{w}}) \ , \tag{5.2}$$

where the perturbation $\boldsymbol{\delta}$ is bounded within an ℓ_p ball of radius ε.

Notably, the previous also corresponds to the maximization of an inner product over the ℓ_q ball of radius ε:

$$\max_{\|\boldsymbol{\delta}\|_\mathsf{p} \leq \varepsilon} \boldsymbol{\delta}^\top \nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}}) = \varepsilon \|\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}})\|_\mathsf{q} \ , \tag{5.3}$$

where ℓ_q is the dual norm of ℓ_p. For the common values of the latter, this is maximized as follows:

- $\mathsf{p} = 2$, the maximum is $\hat{\boldsymbol{\delta}} = \varepsilon \frac{\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}})}{\|\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}})\|_2}$;
- $\mathsf{p} = \infty$, the maximum is $\hat{\boldsymbol{\delta}} \in \varepsilon \cdot \operatorname{sign}(\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}}))$;
- $\mathsf{p} = 1$, the maximum is achieved by setting the values of $\hat{\boldsymbol{\delta}}$ that correspond to the maximum absolute values of $\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}})$ to their sign, *i.e.*, ± 1, or 0 otherwise.

After substituting the relevant value of $\hat{\boldsymbol{\delta}}$ from above into Equation 5.1, we compute the loss increment $\Delta\mathcal{A} = \hat{\boldsymbol{\delta}}^\top \nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \mathbf{w})$ under a transfer attack in closed form; *e.g.*, for $\mathsf{p} = 2$, it is given as:

$$\Delta\mathcal{A} = \varepsilon \frac{\nabla_{\boldsymbol{x}} \hat{\mathcal{A}}^\top}{\|\nabla_{\boldsymbol{x}} \hat{\mathcal{A}}\|_2} \nabla_{\boldsymbol{x}} \mathcal{A} \leq \varepsilon \|\nabla_{\boldsymbol{x}} \mathcal{A}\|_2 \ , \tag{5.4}$$

where, for compactness, we use $\hat{\mathcal{A}} = \mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}})$ and $\mathcal{A} = \mathcal{A}(\boldsymbol{x}, y, \mathbf{w})$.

In Equation 5.4, the left-hand side is the increase in the loss function in the black-box case, while the right-hand side corresponds to the white-box case. The upper bound is obtained when the parameters $\hat{\mathbf{w}}$ of the surrogate classifier are equal to the parameters $\mathbf{w}$ of the target (white-box attacks). These results also hold for $\mathsf{p} = 1$ and $\mathsf{p} = \infty$ (using the dual norm in the right-hand side).

5.1.2 Poisoning Attacks

A similar derivation can be followed for poisoning attacks. Instead of defining transferability in terms of the attacker objective computed on a single input sample, we define it in terms of the validation loss attained by the target classifier under the influence of the poisoning points, similarly to Equation 4.7.

Using the same linear approximation described in the previous section, this yields:

$$T \approxeq L(\mathcal{D}_{\text{val}}, \mathbf{w}) + \hat{\boldsymbol{\delta}}^{\top} \nabla_{\boldsymbol{x}} L(\mathcal{D}_{\text{val}}, \mathbf{w}) \ , \tag{5.5}$$

where $\mathcal{D}_{\text{val}}$ are the untainted validation points, and $\hat{\boldsymbol{\delta}}$ is the perturbation applied to the initial poisoning point $\boldsymbol{x}$ against the surrogate classifier. As in Equation 4.7, L depends on the poisoning point through the classifier parameters $\mathbf{w}$, and the gradient $\nabla_{\boldsymbol{x}} L(\mathcal{D}_{\text{val}}, \mathbf{w})$ here is equivalent to the generic one defined by Equation 4.11.

It is clear now that the perturbation $\hat{\boldsymbol{\delta}}$ maximizes the (linearized) objective when it is best aligned with its derivative $\nabla_{\boldsymbol{x}} L(\mathcal{D}_{\text{val}}, \mathbf{w})$, according to the constraint used, as in the previous case.

5.2 Evaluation metrics

The formal definitions given in the previous section reveal some interesting connections between the transferability of adversarial attacks, model complexity (controlled by the classifier hyperparameters), and the input gradients, allowing the definition of few simple and computationally-efficient metrics to quantitatively measure this property.

5.2.1 Size of Input Gradients

The first interesting observation is that transferability depends on the size of the gradient of the objective $\mathcal{A}$ computed using the *target* classifier, regardless of the surrogate: the larger this gradient is, the larger the attack impact may be. This is inferred from the upper bound in Equation 5.4. We define the corresponding metric $S(\boldsymbol{x}, y)$ as:

$$S(\boldsymbol{x}, y) = \|\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, y, \mathbf{w})\|_{\mathsf{q}} \ , \tag{5.6}$$

where ℓ_{q} is again the dual of the perturbation norm ℓ_{p}.

The size of the input gradient also depends on the complexity of the given model, controlled, *e.g.*, by its regularization hyperparameter. Less complex, strongly-regularized classifiers tend to have smaller input gradients, *i.e.*, they learn smoother functions that are more robust to attacks, and vice-versa. Notably, this holds for evasion attacks, but also for poisoning attacks, as the latter's gradient in Equation 4.12 is proportional to α_c (the dual parameter assigned to the attack point by the learning algorithm), which is larger when the model is weakly regularized. In Figure 5.1 we report an example showing how increasing regularization (*i.e.*, decreasing complexity) for a neural network trained on MNIST89

(see Section 8.1.1), by controlling its *weight decay*, reduces the average size of its input gradients, improving adversarial robustness to evasion.

It should be noted however that, since complexity is a model-dependent characteristic, *the size of input gradients cannot be directly compared across different learning algorithms*; *e.g.*, if a linear SVM exhibits larger input gradients than a neural network, we cannot conclude that the former is more vulnerable.

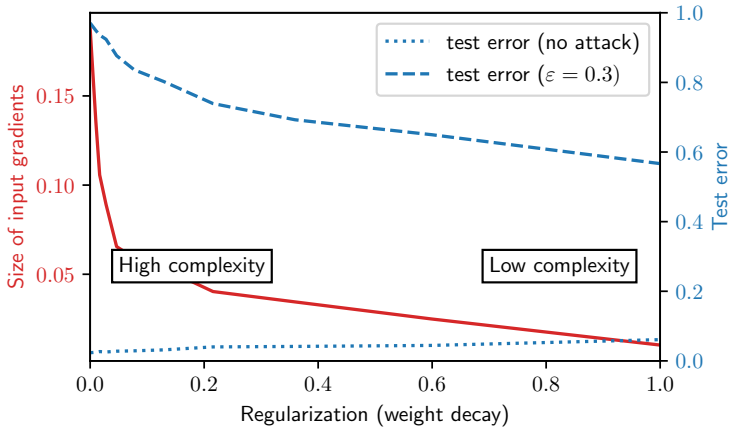


Figure 5.1: Size of test set's input gradients and test error (in the absence and presence of evasion attacks), against regularization (controlled via weight decay) for a neural network trained on MNIST89 (see Section 8.1.1). Note how the size of the input gradients and the test error under attack decrease as regularization (complexity) increases (decreases).

Another interesting observation is that, if a classifier has large input gradients (*e.g.*, due to high-dimensionality of the input space and low level of regularization), for an attack to succeed it may suffice to apply only tiny, *imperceptible* perturbations. As we experimentally demonstrate in Section 8.1, this explains why adversarial examples against deep neural networks can often be only slightly perturbed and still mislead detection while, when attacking less complex classifiers in low dimensions, the necessary modifications become more evident.

5.2.2 Gradient Alignment

The second relevant impact factor on transferability is based on the alignment of the input gradients of the loss (objective) function computed using the target and the surrogate learners. If we compare the increase in the loss function in the black-box case (the left-hand side of Equation 5.4) against that corresponding to white-box attacks (the right-hand side), we find that the relative increase in loss, at least for ℓ_2 perturbations, is given by the value of the gradient alignment metric $R(\boldsymbol{x}, y)$:

$$R(\boldsymbol{x}, y) = \frac{\nabla_{\boldsymbol{x}} \hat{\mathcal{A}}^{\top} \nabla_{\boldsymbol{x}} \mathcal{A}}{\|\nabla_{\boldsymbol{x}} \hat{\mathcal{A}}\|_2 \|\nabla_{\boldsymbol{x}} \mathcal{A}\|_2} \ , \tag{5.7}$$

where the gradient terms are indicated as in Equation 5.4.

Notably, this is exactly the cosine of the angle between the gradient of the objective computed on the surrogate and the one computed on the target classifier. This is a novel finding which explains why the cosine angle metric between the target and surrogate gradients can well characterize the transferability of attacks, confirming empirical results from previous work [146]. For other perturbation norms this definition slightly changes, but gradient alignment can be similarly evaluated. Differently from the gradient size S, gradient alignment R is a pairwise metric, allowing

comparisons across different surrogate models; *e.g.*, if a surrogate SVM is better aligned with the target model than another surrogate, we can expect that attacks targeting the surrogate SVM to transfer better.

5.2.3 Variability of the Loss Landscape

We define here another useful metric to characterize attack transferability. The idea is to measure the variability of the loss function, *i.e.*, the attacker's objective $\hat{\mathcal{A}}$, when the training set used to learn the surrogate model changes, even though it is sampled from the same underlying distribution. In fact, if this loss landscape changes dramatically even when simply resampling the surrogate training set $\hat{\mathcal{D}}_{\text{tr}}$ (which may happen, *e.g.*, for surrogate models exhibiting a large error variance, like neural networks and decision trees), it is very likely that the local optima of the corresponding optimization problem change too, and this may in turn imply that the attacks will not transfer correctly to the target learner.

We define the variability $V(\boldsymbol{x}, y)$ of the loss landscape as its *variance*:

$$V(\boldsymbol{x}, y) = \mathbb{E}_{\hat{\mathcal{D}}_{\text{tr}1}}\{\mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}})^2\} - \mathbb{E}_{\hat{\mathcal{D}}_{\text{tr}2}}\{\mathcal{A}(\boldsymbol{x}, y, \hat{\mathbf{w}})\}^2 \ , \tag{5.8}$$

where $\mathbb{E}_{\hat{\mathcal{D}}_{\text{tr}}}$ is the expectation taken with respect to different (surrogate) training sets. This is similar to what is typically done to estimate the variance of classifiers' predictions. Figure 5.2 reports a conceptual representation of this notion. As for the size of input gradients S, the loss variance V should only be compared across models trained with the same learning algorithm.

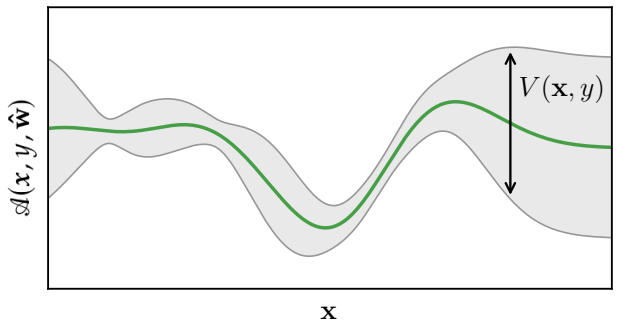


Figure 5.2: Conceptual representation of the variability of the loss landscape $V(\boldsymbol{x}, y)$. The green line represents the expected loss with respect to different training sets used to learn the surrogate model, while the gray area represents the variance of the loss landscape. If the variance is too large, local optima may change, and the attack may not successfully transfer.

Explaining Vulnerability to Adversarial Attacks and Attack Transferability

6

In Chapter 5 we explained how to compare machine learning based systems, through the use of few reliable and efficient metrics, to assess their vulnerability to transfer attacks, *i.e.*, attacks that are not only effective against the model they have been crafted on, but can also successfully fool other ones. Transferability, however, is not the only security-related property that should be evaluated while comparing different classifiers. The general adversarial robustness of a model, for example, is often analysed mainly through means of the security evaluation curves (see Section 2.2.4), obtained by running costly simulations of the attacks.

We also discussed how properties like the user *trust* in the system's decisions cannot be measured using standard performance metrics, as they are strictly related to the ability of the system to *explain* their outputs (see Section 2.3). How to compare the behavior of different systems based on these user-dependent aspects is still an open question in the literature, due to the strict connection between most explanation methods with the machine learning model they try to interpret.

In this chapter we discuss how gradient-based explanations can be leveraged effectively to evaluate the security of machine learning based systems. We start in Section 6.1 by describing a novel method to compute local and global explanations based on *highly-interpretable relevance vectors*, which can be used to compare different machine learning models, even non-differentiable ones, or black-box systems. Notably, by applying this method, we are able to obtain some insights on the vulnerability to transfer attacks, without computing any additional specific metric. Then, in Section 6.2, we provide few quantitative metrics that allow to *correlate* the *gradient-based explanations* with the *adversarial robustness*, obtaining insights on the vulnerability of a machine learning model, without the need of running a full security evaluation.

Later, in Chapter 8, we conduct a thorough experimental analysis of the proposed techniques by evaluating different machine learning models trained for the task of detecting Android malware.

Notation. In the following, we denote a feature-based explanation as $\boldsymbol{r} = [r^{(1)}, r^{(2)}, \ldots, r^{(\mathsf{d})}]$, where $r^{(k)}$ is the attribution (relevance) of the k-th component (feature) in the sample $\boldsymbol{x}$.

6.1 Relevance Vectors for Model (Global) Explanations

In the following, we describe a novel local and global explainability method which, in contrast to other post-hoc feature-based techniques (Section 2.3.2), allows to compare different machine learning models

through highly-interpretable *relevance vectors* [17]. As our approach is based on the Gradient*Input method, it is more suitable than many other gradient-based techniques (like considering the simple gradient, see Equation 2.17), especially when the feature vectors are sparse, because they often assign high relevance to features which corresponding components are *not* present in the considered samples, *e.g.*, when applied to malware detection applications. Moreover, our method can be directly extended to explain, and compare, non-differentiable models and black-box systems, through the use of surrogate models.

6.1.1 Local relevance vectors

Given an input sample $\boldsymbol{x}$ which prediction given by the classifier f should be locally explained, we start by taking the output $\boldsymbol{v}$ of the Gradient*Input method [176, 177] (Section 2.3.2). For reference, we report a compact formulation of its original Equation 2.20 below:

$$\boldsymbol{v} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) \cdot \boldsymbol{x} \ , \tag{6.1}$$

where $\cdot$ denotes the element-wise product between two vectors. As already discussed, this method basically projects the gradient of the decision function w.r.t. the input sample, *i.e.*, $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$, onto $\boldsymbol{x}$ itself, thus ensuring that the relevance is proportional to the input features, including setting as zero the attribution of the null features of $\boldsymbol{x}$.

The magnitude of the explanation $\boldsymbol{v}$ however, directly depends on the magnitude of the classification scores given by the decision function f (through its gradient), and on the value of each feature in $\boldsymbol{x}$. Thus, to compare explanations obtained on different decision functions, we propose to normalize $\boldsymbol{v}$ to have an unary ℓ_1 norm, obtaining the *local relevance vector*:

$$\boldsymbol{r} = \boldsymbol{v} / \|\boldsymbol{v}\|_1 \ . \tag{6.2}$$

The explanations obtained using Equation 6.2 are highly-interpretable as the attribution of the k-th feature is limited to $r^{(k)} \in \{0, 1\}$, *e.g.*, if the classifier assigns all the relevance to a single feature, the value of the corresponding attribution will be 1, and all the other $\mathsf{d} - 1$ components will have attribution equal to 0. The vector $\boldsymbol{r}$ can then be displayed as normally done for saliency maps, or its absolute values can be ranked in ascending/descending order to identify the most influential features for the input sample, *e.g.*, as done in Figure 2.21, for interpretable models.

6.1.2 Global relevance vectors

Equation 6.2 can be directly extended to provide an interpretation of the learned model as a whole. Our idea is to average the relevance vectors $\boldsymbol{r}$ computed over different samples, obtaining the *global relevance vector*:

$$\bar{\boldsymbol{r}} = \sum_i \boldsymbol{r}_i \ , \tag{6.3}$$

where $\boldsymbol{r}_i$ represents the local relevance vector computed on a generic sample $\boldsymbol{x}_i \in \mathcal{X}$. Then, as in the local case, the absolute values of the average relevance vector $\bar{\boldsymbol{r}}$ can be displayed in the form of saliency maps, to get a feature-based snapshot of the behavior learned by the model, or its values ranked in ascending/descending order to identify the most influential *global* features.

It should be noted however, that the choice of which set of samples to consider when computing Equation 6.3 plays a big role in the interpretability of the obtained global explanation. In fact, especially for highly unbalanced datasets, *i.e.*, where the samples of a certain class outnumber the samples from few or all other classes, the attribution corresponding to the features that are highly relevant only for a limited set of samples can be massively soften by the averaging process. Thus, it is often more useful to compute a global relevance vector separately for the samples from each class in a dataset, or for each logically-related set of samples. Part of our experiments, in Chapter 8 we show an example of this methodology applied to a malware detection system (Figure 8.25), where the global explanations are computed separately for benign and malware samples, and for different types of malicious apps in the dataset.

6.1.3 Explaining black-box systems

Our explanation method, similarly to other gradient-based techniques, works under the assumption that the decision function f is differentiable w.r.t the input $\boldsymbol{x}$, and that its gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ is sufficiently smooth to provide meaningful information at each point. However, when $f(\boldsymbol{x})$ is not differentiable (*e.g.*, for random forests), one can approximate f by means of a surrogate model $\hat{f}$ [175, 155, 17], using the latter to compute the relevance vectors. The resulting explanations should provide a good approximation of the attributions assigned by the original classifier to the features in the input sample, as we report in Chapter 8.

6.2 The connection between Explanations and Adversarial Robustness

We now discuss the possible connections between gradient-based explanations and the robustness of machine learning models to adversarial attacks. Motivated by the experimental findings obtained using our global relevance vectors (see Chapter 8), and by the intuition that classifiers whose attributions are more evenly distributed should also be the more robust (as they rely on a broader set of features for the decision, see Section 2.2.5), we propose few synthetic metrics that allow to correlate the *uniformity* of the explanation vectors with the *adversarial robustness* of a machine learning model.

6.2.1 Explanation Evenness

We start by describing two different metrics that allow to compute the evenness of a relevance vector, *i.e.*, the uniformity of its values.

The following has been originally proposed in [102] and later expanded in [124]. Let's define a function $F(\boldsymbol{r}, \mathsf{k})$ which, given a relevance vector $\boldsymbol{r}$, computes the ratio of the sum of the k highest relevance values to the sum of all absolute relevance values. This, for $\mathsf{k} = 1, \ldots, \mathsf{D}$, is given by:

$$F(\boldsymbol{r}, \mathsf{k}) = \frac{\sum_{j=1}^{\mathsf{k}} |r^{(j)}|}{\|\boldsymbol{r}\|_1} \ , \tag{6.4}$$

where the relevance values are sorted in descending order of their absolute values, *i.e.*, $|r^{(1)}| \geq |r^{(2)}| \geq \ldots \geq |r^{(\mathsf{D})}|$, and D is the number of considered relevance values (with $\mathsf{D} \leq \mathsf{d}$). Since in many applications each sample only exhibits a small number of non-zero features compared to the entire, very large, d-dimensional feature space (*e.g.*, in Android malware detection systems), the corresponding relevance vector computed by Equation 6.2 will be sparse. Thus, to reduce computational cost, one may restrict the analysis to the first non-null D components.

For Equation 6.4, the evenest distribution (the one for which all attributions are equal), corresponds to $F(\boldsymbol{r}, \mathsf{k}) = \mathsf{k}/\mathsf{D}$, whereas the most uneven is attained when only one relevance differs from zero and, in this case, it yields $F(\boldsymbol{r}, \mathsf{k}) = 1$ for each k value. However, to avoid dependence on k and to obtain a single scalar value, the final evenness metric can be computed as follows [124]:

$$\mathscr{E}_1(\boldsymbol{r}) = \frac{2}{\mathsf{D}-1} \left[\mathsf{D} - \sum_{k=1}^{\mathsf{D}} F(\boldsymbol{r}, \mathsf{k}) \right] \ , \tag{6.5}$$

which value is limited to $\mathscr{E}_1 \in [0, 1]$. In this case, $\mathscr{E}_1 = 0$ and $\mathscr{E}_1 = 1$ indicate, respectively, the most uneven and the most even vector.

A different metric has been proposed in [123], based on the ratio between the ℓ_1 and ℓ_∞ norms:

$$\mathscr{E}_2(\boldsymbol{r}) = \frac{1}{\mathsf{D}} \cdot \frac{\|\boldsymbol{r}\|_1}{\|\boldsymbol{r}\|_\infty} \ , \tag{6.6}$$

which value in this case is limited to $\mathscr{E}_2 \in [\frac{1}{\mathsf{D}}, 1]$, with $\mathscr{E}_2 = \frac{1}{\mathsf{D}}$ being the uniformity of a relevance vector with only one component different from zero, and $\mathscr{E}_2 = 1$ if the attribution is identical for all the features.

To obtain a broader perspective of the attributions evenness, we compute the metrics on multiple samples, and we average the results. Formally, we define the *explanation evenness* as:

$$E = \frac{1}{\mathsf{M}} \sum_{i=1}^{\mathsf{M}} \mathscr{E}(\boldsymbol{r}_i) \ , \tag{6.7}$$

where $\boldsymbol{r}_i$ is the relevance vector computed on each sample of the M-sized test dataset $\mathscr{D}_{\text{ts}}$, and $\mathscr{E}$ is any of the uniformity metrics described before. Specifically, in our experiments, we represent the averaged evenness computed considering the per-sample metric $\mathscr{E}_1$ ($\mathscr{E}_2$) with E_1 (E_2).

6.2.2 Adversarial Robustness

To compute the correlation between explanation evenness and the robustness to evasion attacks, we need a metric that compactly represents the information usually given by means of security evaluation curves (see Section 2.2.4). We thus propose to quantitatively measure the robustness of a classifier to the adversarial examples crafted using a maximum input perturbation of size ε, as follows:

$$\mathcal{AR}(\mathcal{D}_\varepsilon, f) = \frac{1}{\mathsf{M}} \sum_{i=1}^{\mathsf{M}} e^{-\mathcal{A}_i} \ , \tag{6.8}$$

where $\mathcal{A}_i = \mathcal{A}(\boldsymbol{x}'_i, y_i, \mathbf{w})$ is the adversarial loss (see Equation 4.4) attained by the classifier f on each sample from the M-sized set $\mathcal{D}_\varepsilon$ of ε-perturbed adversarial examples. This measure can then be averaged over the security evaluation curve, by considering multiple $\mathcal{D}_\varepsilon$ with different ε, obtaining the final *adversarial robustness* metric:

$$AR = \mathbb{E}_{\mathcal{D}_\varepsilon}\{\mathcal{AR}(\mathcal{D}_\varepsilon, f)\} \ . \tag{6.9}$$

Explanation evenness (Equation 6.7) and adversarial robustness (Equation 6.9) are the two metrics we correlate in Chapter 8 to investigate the connections between gradient-based explanations and the robustness of machine learning based systems to adversarial (evasion) attacks.

`secml`: a Python library for Secure and Explainable Machine Learning

7

In the previous chapters we discussed how a machine learning based system can be evaluated over many different aspects, from the accuracy of its decisions, to its robustness to adversarial attacks. Moreover, we shown that being able to explain the automated decisions is a fundamental step to acquire enough users trust so that they will employ the system in a real-world scenario. When it comes to practically implement all these procedures, the *Python programming language* has established itself as one of the most popular choices of the latest years [200, 201]. It provides a general-purpose platform for fast algorithmic development, scientific data analysis and, thanks to a vast ecosystem of libraries, it is often an attractive option for specific industry applications.

Despite the huge number of handy libraries that allow a straightforward implementation of many machine learning algorithms [202, 200, 203], including neural networks [204, 205], and allow data analysis and visualization [206, 207, 208], only a few choices are available for testing these automated systems in an adversarial environment [209, 210, 211]. This should include being able to simulate evasion and poisoning attacks, and perform a thorough security evaluation of the adversarial robustness of the models, along with testing of their vulnerability to transfer attacks. Lastly, the issue of interpretability is only sparely tackled in the Python ecosystem, mainly with specific libraries implementing research code in support of scientific publications [155, 185].

In this chapter, we present `secml` [19], an open-source Python library that aims to tackle all the aforementioned aspects of developing and testing pattern recognition systems in a single framework, thus favoring an easier development of more secure and explainable learning algorithms. To this end, `secml` implements: (*i*) a seamless integration between the most popular scientific libraries for machine learning, including *numpy*, *scipy*, *PyTorch* and *TensorFlow*; (*ii*) a single unified data structure which allows developing algorithms working in both dense and sparse features spaces; (*iii*) a unified framework to perform empirical security evaluation against evasion and poisoning attacks; and (*iv*) different techniques for local and global interpretation of machine learning models, including feature-based and prototype-based explanation methods. With respect to other popular libraries mainly focused on implementing attacks against deep neural networks [209, 210, 211], `secml` provides training-time poisoning attacks and computationally-efficient test-time evasion attacks against many traditional algorithms, including support vector machines and random forests.

`secml` is a project born in 2014 and open-sourced in August 2019. Thanks to an emerging community of users and developers from both our GitLab[1] and GitHub[2] repositories, it is constantly updated to enrich it with new functionalities, novel attacks and defenses, and wrappers for supporting other third-party libraries.

1: `https://gitlab.com/secml/secml`
2: `https://github.com/pralab/secml`

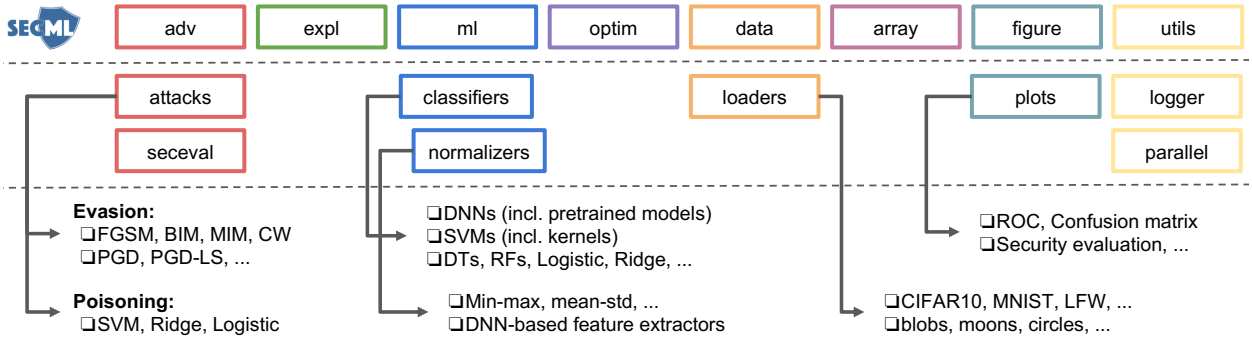


Figure 7.1: Architecture and main packages of `secml`.

7.1 Architecture and Implementation

`secml` has a modular architecture oriented to code reuse, depicted in Figure 7.1. We define one or more abstract interfaces for each component, including (but not limited to) loss functions, regularizers, optimizers, classifiers and attacks. Notably, the definition of an optimization problem is separated from the algorithm used to solve it, so that one can easily implement novel attacks or classifiers (in terms of constrained optimization problems), and then use different optimizers to obtain a solution. This is a great advantage with respect to other libraries like *CleverHans* [209] as, *e.g.*, we can switch from white-box to black-box attacks by just changing the optimizer (from a gradient-based to a gradient-free solver), without redefining the entire optimization problem.

`secml` integrates different components via easy-to-use wrapping classes. We provide many attacks as implemented by *CleverHans*, but we also extended them to compute the values of the loss function, and of the intermediate points optimized during the attack iterations, as well as the number of function and gradient evaluations. This is useful to debug and compare different attacks, *e.g.*, by checking their convergence to a local optimum, and properly tune their hyperparameters (*e.g.*, step size and number of iterations). `secml` also supports deep neural networks via a dedicated *PyTorch* wrapper, which can be extended, if desired, to other popular deep-learning frameworks, like *Keras*. Notably, this functionality allows to run attacks that are natively implemented in *CleverHans* also against *PyTorch* models.

Main packages

The `adv` package implements a framework to perform both evasion and poisoning attacks, following the formulation described in Chapter 4. It also provides the functionalities necessary to perform a full security evaluation of the machine learning models against both attacks, and test their transferability. Moreover, it encompasses the evasion attacks provided by *CleverHans*, which are directly integrated in our framework. The related package `optim`, provides an implementation of the Projected Gradient Descent (PGD) optimization algorithm, and the more efficient version of it that runs a bisect line-search along the gradient direction (PGD-LS) to reduce the number of gradient evaluations (Algorithm 1).

The ml package provides an implementation of many classifiers from *scikit-learn*, and of the deep neural networks from *PyTorch*. We extended the *scikit-learn* classifiers with the gradients required to run evasion and poisoning attacks, which are analytically implemented in closed form. Our library also supports chaining different modules (*e.g.*, scalers and classifiers), and can automatically compute the corresponding end-to-end gradient via the chain rule.

The explanation package implements many feature- and prototype-based explanation methods described in Section 2.3, including Gradient, Gradient*Input, and Integrated Gradients techniques [175, 174, 176, 177, 178], as well as influence functions [189], and our gradient-based explanation method based on *relevance vectors*, presented in Section 6.1.

Packages array and data provide the structures necessary to load and manage input data, integrating those provided by *scikit-learn* and *PyTorch*. Unique to our library, the main data structure CArray is a higher-level interface for both dense (*numpy*) and sparse (*scipy*) arrays, enabling an efficient execution of attacks especially on sparse data representations, *e.g.*, for malware detection systems.

Finally, package figure implements some advanced plotting functions based on *matplotlib* (*e.g.*, to visualize and debug attacks); while package utils provides functionalities for logging and parallel code execution.

Testing, documentation, and contributions

Our library is extensively tested on Ubuntu and Windows operating systems, via a dedicated continuous-integration server. We also run frequent tests on the latest macOS versions. This ensures a high-quality source code, and reduces the possibility of encountering problems during use. To furtherly enhance the user experience, we provide a complete documentation online at https://secml.gitlab.io, along with many examples in jupiter-notebook format on how to use both the basic and the advanced functionalities of the library.

secml is an open-source project, and we are committed to encourage code contributions by maintaining an up-to-date developers' guide and documentation on how to extend each package with more attacks, classifiers, wrappers of deep-learning frameworks, etc. The latest version of the dev-docs is available at https://secml.gitlab.io/developers.

7.2 Applications

In this section, we show how to use secml to build, explain, attack, and evaluate the security of different machine learning based systems for practical application tasks, including support vector machines for Android malware detection and deep neural networks for computer vision. The following examples are extracted from the user guides available at https://secml.gitlab.io, to which we refer for the full source code and documentation.

7.2.1 Android Malware Detection

This first example shows how to use `secml` for building and testing an Android malware detection system. In particular, we simulate the Drebin architecture [212], depicted in Figure 8.12, by learning a linear Support Vector Machine (SVM) on a subset of the *Drebin* dataset. Our library allows to leverage all the computational advantages of SVMs, which provide remarkable performance even when working on high-dimensional but sparse features spaces (over 1 million features in this case, mostly zeros).

Training and Performance Evaluation

We start by loading a toy dataset of Android applications, consisting of 12,000 benign and 550 malicious samples extracted from the *Drebin* dataset. Then, we train the support vector machine classifier, using the `secml`'s `CClassifierSVM` class, on half the dataset, while using the rest for testing. The single hyperparameter of the SVM has been set to $C = 0.1$ via a cross-validation procedure, directly available in the library's class.

After the learning procedure is finished, we assess the performance of the classifier on the task of recognizing benign and malicious applications. The results are reported in Figure 7.2 by means of a Receiver Operating Characteristic (ROC) curve. `secml` provides all the necessary functions to compute and visualize ROC curves, as well as other performance metrics to evaluate any trained classifier, including neural networks.

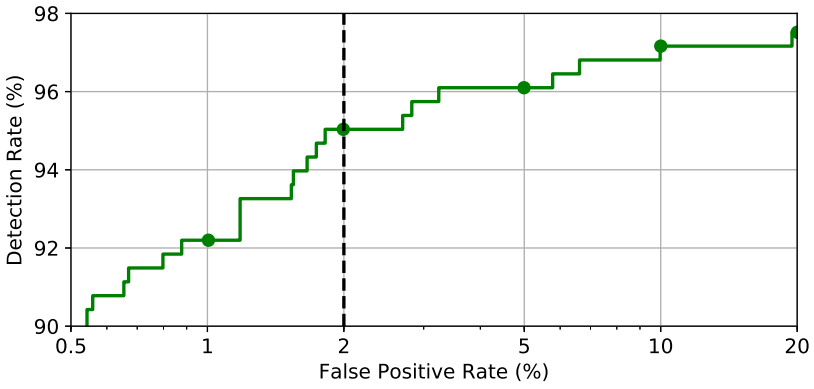


Figure 7.2: ROC curve to evaluate the performance of a linear support vector machine implemented in `secml`, trained on a subset of the *Drebin* data [212]. The *detection rate* represents the probability that a malicious sample is correctly labeled by the classifier. The false positive rate (2%) which is used for computing the security evaluation curve (Figure 7.3) is highlighted with a dashed black line.

Explanations

The post-hoc explanations for the trained malware detector can then be computed to understand which components of the Android applications are more relevant for the classifier during the decision (classification) phase. For two samples from the test set, one benign and one malicious, we compute the local *relevance vectors* leveraging our explanation method (Equation 6.2). In `secml`, this can be obtained using the standard Gradient*Input implementation provided by the `CExplainerGradientInput` class and dividing the output by its ℓ_1 norm.

The top-10 influential features and the corresponding relevance are reported in Table 7.1. Looking at the first sample, a benign application, we observe how the majority of the features have a negative relevance towards

Explanations for sample 137 (true class: 0)		Explanations for sample 138 (true class: 1)	
r (%)	**Feature Name**	r (%)	**Feature Name**
-8.30	android.permission.CALL_PHONE	15.18	TelephonyManager;->getNetworkOperator
-6.59	suspicious_calls::android/net/Uri;->toString	12.55	android.permission.SEND_SMS
5.60	android.permission.INTERNET	8.69	android.permission.READ_SMS
-5.39	used_permissions::ACCESS_NETWORK_STATE	5.83	android.permission.INTERNET
-5.08	api_calls::android/media/MediaPlayer;->start	5.58	android.intent.action.BOOT_COMPLETED
-4.24	used_permissions::VIBRATE	-4.41	used_permissions::VIBRATE
-3.65	android.intent.category.LAUNCHER	3.96	android.intent.category.HOME
-3.56	android.permission.ACCESS_FINE_LOCATION	-3.80	android.intent.category.LAUNCHER
-3.43	used_permissions::ACCESS_FINE_LOCATION	3.66	com.google.android.c2dm.C2DMBroadcastReceiver
-2.63	LocationManager;->getLastKnownLocation	3.39	android.permission.READ_PHONE_STATE

Table 7.1: Top-10 influential features and corresponding relevance as computed by our explanation method (Equation 6.2) implemented in `secml`, for one benign sample (left) and one malware (right) from the *Drebin* dataset [212]. Notice how the majority of the features in the benign sample have a negative relevance towards the decision, meaning that, for the classifier, are an indication of a benign behavior. Conversely, most features for the malware have a positive relevance and, thus, considered malicious components.

the decision, meaning that, for the SVM, those are an indication of a benign behavior. Conversely, we obtain the opposite for the second sample, as most of the features have a positive relevance value and, thus, considered malicious components. We also observe how the classifier identifies different cellular- and SMS-related features as malicious, coherently with the actual behavior of many malware apps, which goal is often to send SMS messages to premium-rate numbers. Similar results are obtained in the experiments discussed in Chapter 8.

We also discover that more than ~50% of the relevance is assigned to only 10 features in both cases. This highlights a known behavior of these classifiers, which tend to assign most of the weight to a small set of features, making them highly vulnerable to evasion attacks as we described in Section 2.2.

Security Evaluation

We now set up a gradient-based maximum-confidence evasion attack to craft adversarial examples against the linear SVM classifier, as described by Algorithm 1 and implemented in `secml` by the `CAttackEvasionPGDLS` class. The attacks are then used to perform a security evaluation of the malware detection system.

The solver parameters should be chosen according to the specific application. As we are working with sparse Boolean features (each can take either 0 or 1 value), we use a ℓ_1 norm constraint (Equation 4.4). Secondly, the lower and the upper bound constraints for the features (Equation 4.5) are critical in this case. In fact, to create malware able to fool a classifier, an attacker may, in theory, both adding and removing features from the original applications. However, in practice, feature removal is a non-trivial operation that can easily compromise the malicious functionalities of the application and, generally, only be performed for components not from the `manifest` [3] [21]. Feature addition is a safer operation, especially when the injected features belong to the `manifest`; for example, adding permissions does not influence any existing functionality of the app. In `secml`, to only allow feature addition one can set $\boldsymbol{x}_{\text{lb}} = \text{'x0'}$ and $\boldsymbol{x}_{\text{ub}} = 1$. To also allow feature removal, the lower bound can be set to $\boldsymbol{x}_{\text{lb}} = 0$.

3: See Section 8.1.2 for some details on the structure of Android applications.

Finally, secml provides different handy classes to produce and visualize a security evaluation curve using the computed adversarial samples. The process for this example should last around 60 seconds, despite a feature space of over 1 million components, as all the functions are optimized to work on sparse data. The resulting plot is reported in Figure 7.3, with the detection rate at 2% False Positive Rate (FPR) for increasing values of the ℓ_1-order maximum perturbation ε. We can observe how this malware detector is highly vulnerable to adversarial attacks, and after changing less than 10 features half of the malicious samples are incorrectly classified as benign applications. This known vulnerability has also been highlighted when we listed the top influential features (Table 7.1), and observed that most of the relevance is assigned to a very limited set of features.

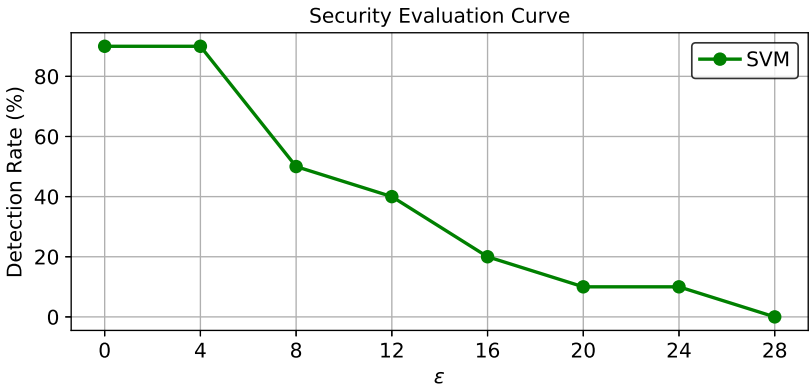


Figure 7.3: Security evaluation of the Drebin malware detector implemented in secml against white-box evasion attacks. Detection rate at 2% FPR against an increasing number of added features ε. We can observe how after changing less than 10 features, half of the malicious samples are incorrectly classified as benign applications, due to the high vulnerability of the linear SVM to this type of attacks.

7.2.2 Computer Vision

In the following, we show another possible application of secml, running different evasion attacks against ResNet-18, a convolutional neural network trained on the ImageNet dataset, available from *torchvision*. This example demonstrates how our library enables running also attacks from third-party libraries, like *CleverHans* (implemented in *TensorFlow*), against *PyTorch* models. In particular, we aim to have the race car depicted in Figure 7.4 (leftmost plot) misclassified as a tiger.

Experimental settings

We use three different ℓ_2-norm targeted attacks: Carlini-Wagner (CW) [82] (from *CleverHans*); our Projected Gradient Descent (PDG) with line-search (Algorithm 1); and a variant of the latter, namely PGD-patch, where we restrict the attacker to only change the pixels of the image corresponding to the license plate, using a box constraint [20]. All the attacks run for 50 iterations, adjusting the step size to reach convergence within this range. For CW, we set the confidence parameter $\kappa = 10^6$ to generate high-confidence misclassifications, and $c = 0.4$, yielding an ℓ_2 perturbation size of $\varepsilon = 1.87$. We add a bound constraint to the PGD attack to create an adversarial image with the same perturbation size. For PGD-patch instead, we do not bound the perturbation size to avoid unreasonably limiting of the capabilities of the attack.

Adversarial Attacks

The crafted adversarial images are shown in Figure 7.4, for each different attack algorithm. In Figure 7.5 (left) we show how the attack losses (scaled linearly in $[0, 1]$ to enable comparison) are minimized while the attacks iterate. Also, Figure 7.5 (right) shows how the confidence assigned to the class *race car* (dashed line) decreases in favor of the confidence assigned to the class *tiger* (solid line) for each attack, across different iterations. These plots are particularly useful to tune the attack's hyperparameters (*e.g.*, the step size or the number of iterations), and to check the converge to a good local optimum. Also, such visualizations may help avoid common pitfalls in the security evaluation of learning algorithms, facilitating understanding and configuration of the attacks.

Figure 7.4: Adversarial images representing a *race car* misclassified as a *tiger*, produced by three different attack algorithms implemented in `secml`: Carlini-Wagner (CW); Projected Gradient Descent with line-search (PDG); and PDG-patch, where a box constraint restricts the attack area. For PGD-patch, we also report the explanations computed using the integrated gradients method (Equation 2.22).

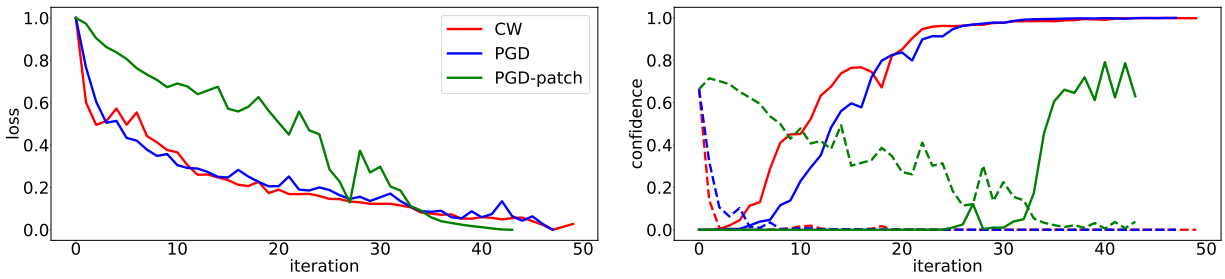


Figure 7.5: Analysis of the optimization process for the different attack algorithms implemented in `secml`: CW (red); PGD (blue); PDG-patch (green). *Left*: minimization of the loss; *Right*: confidence of source class (*race car*, dashed lines) vs confidence of target class (*tiger*, solid lines).

Explanations

For the adversarial example crafted using the PGD-patch algorithm, we highlight the most relevant pixels that lead the deep neural network to classify the attack sample as a tiger (Figure 7.4, rightmost plot). To produce this explanation, we use the *integrated gradients* method (Equation 2.22), which is implemented in `secml` by the `CExplainerIntegratedGradients` class. We observe how the most relevant pixels are found around the perturbed region containing the license plate, unveiling the presence of a potential adversarial manipulation.

8 Experimental Analysis

This chapter presents a systematic experimental evaluation of the techniques we proposed in this thesis. We start in Section 8.1 by analysing the *transferability* property of adversarial attacks on three different applicative cases: (*i*) handwritten digit recognition; (*ii*) Android malware detection; (*iii*) identity verification via face recognition. To this end, we test and compare the quantitative metrics proposed in Section 5.2: (*j*) size of input gradients; (*jj*) gradient alignment; (*jjj*) variability of the loss landscape. Then, in Section 8.2, we follow up on the experiments on Android malware detection by verifying the applicability of our method for computing global model explanations through highly-interpretable *relevance vectors* (Section 6.1). Finally, in Section 8.3, we statistically evaluate the correlation between the *uniformity* of gradient-based explanations and the *adversarial robustness* of a machine learning model to evasion attacks, using the corresponding metrics described in Chapter 6.

We proudly remark that all the experiments are implemented using `secml`, our open-source Python library described in Chapter 7.

8.1 Transferability of Adversarial Attacks

In this section, we evaluate the transferability of both evasion and poisoning attacks across a range of machine learning models. In particular, we highlight some interesting findings about this security-critical property by measuring the three synthetic metrics described in Section 5.2. Given their dependency on the initial attack sample $\boldsymbol{x}$ and its original label y, we compute the mean values averaging on different points.

We consider three practical applications: handwritten digit recognition (Section 8.1.1); Android malware detection (Section 8.1.2); identity verification via face recognition (Section 8.1.3). Evasion and poisoning attacks are both performed in the digits recognition case, and specific evasion and poisoning attacks are crafted against the malware detection and face recognition systems, respectively. Different machine learning algorithms are tested, including support vector machines (with both linear and non-linear decision functions), logistic and ridge classifiers, and neural networks. It should be noted however, that designing efficient poisoning availability attacks against neural networks is still an open problem due to the complexity of the bi-level optimization (Equation 4.7) and the non-convexity of the inner learning problem.[1] For this reason, we test this type of classifiers only against evasion attacks. Similarly, poisoning random forests is not feasible with gradient-based attacks, and we are not aware of any existing work considering this ensemble algorithm.

We conclude the analysis by summarizing our findings in Section 8.1.4.

1: Previous work mainly considered integrity poisoning attacks against neural networks [189, 149, 8], and it is believed that neural networks are much more resilient to poisoning availability attacks due to their memorization capability.

Experimental parameters. Equally for all the considered cases, we set the maximum number of iterations of our gradient-based optimization algorithm to 1,000, yet convergence (Algorithm 1, line 6) is typically reached only after a hundred iterations. Also, we allow a maximum of 20 iterations for the line-search step (Algorithm 1, line 4), which reduces the overall time required to reach a local or global optimum.

8.1.1 Handwritten Digit Recognition

The goal of a machine learning model in this case is to assign the correct label to each input digit. We use the *MNIST*89 data, which includes the MNIST handwritten digits from classes 8 and 9. Each image consists of 784 pixels ranging from 0 to 255, normalized in $[0, 1]$ by dividing such values by 255. Few example images used for the experiments are reported in Figure 8.1. The two original classes, *i.e.*, 8 and 9, are remapped to 0 and 1 (respectively), defining a binary classification problem.

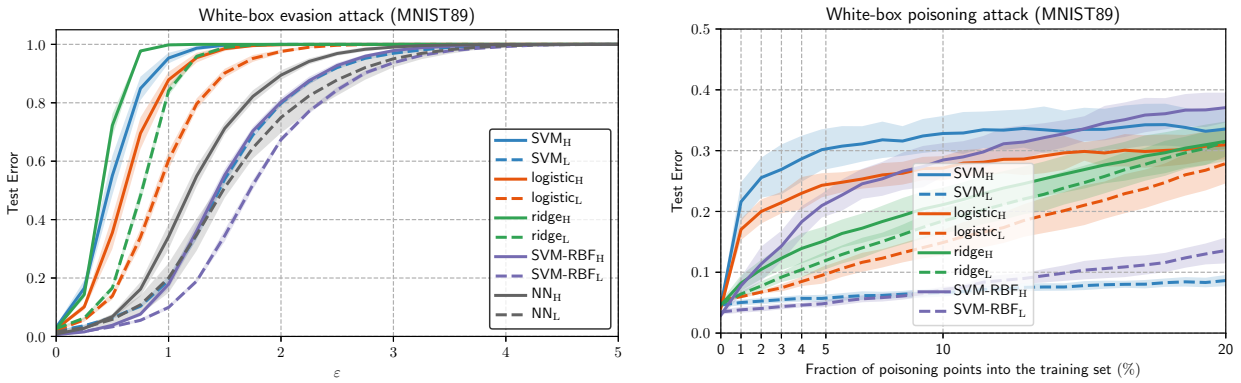
Figure 8.1: Example of handwritten digits from the *MNIST*89 dataset.

Experimental Setup (Evasion)

We run 10 independent repetitions to average the results on different training-test splits. In each repetition, white-box and black-box attacks are performed, using 5,900 samples to train the target classifier, 5,900 distinct samples to train the surrogate classifier (without even relabeling the surrogate data with labels predicted by the target classifier; *i.e.*, we do not query the target). 1,000 samples are used for testing. We perturb the test digits to craft the optimal attacks using Algorithm 1 under the ℓ_2 distance constraint $\|\boldsymbol{x} - \boldsymbol{x}'\|_2 \leq \varepsilon$, with $\varepsilon \in [0, 5]$.

For each of the following learning algorithms, we train a high-complexity (H) and a low-complexity (L) model, by changing its hyperparameters: (i) SVMs with linear kernel (SVM_H with $C = 100$ and SVM_L with $C = 0.01$); (ii) SVMs with RBF kernel (SVM-RBF_H with $C = 100$ and SVM-RBF_L with $C = 1$, both with $\gamma = 0.01$); (iii) logistic classifiers (logistic_H with $C = 10$ and logistic_L with $C = 1$); (iv) ridge classifiers (ridge_H with $\alpha = 1$ and ridge_L with $\alpha = 10$);[2] (v) fully-connected neural networks with two hidden layers including 50 neurons each, and ReLU activations (NN_H with no regularization, *i.e.*, weight decay set to 0, and NN_L with weight decay set to 0.01), trained via cross-entropy loss minimization; and (vi) random forests consisting of 30 trees (RF_H with no limit on the depth of the trees and RF_L with a maximum depth of 8). These configurations are chosen to evaluate the robustness of classifiers that exhibit similar test accuracies but different levels of complexity.

2: The level of regularization in the standard formulation of these classifiers increases as α increases, and as C decreases.



(a) Test error against increasing maximum perturbation ε

(b) Test error against an increasing fraction of poisoning points

Figure 8.2: Security evaluation curves for white-box evasion (a) and poisoning (b) attacks on the handwritten digits *MNIST89* dataset.

Experimental Setup (Poisoning)

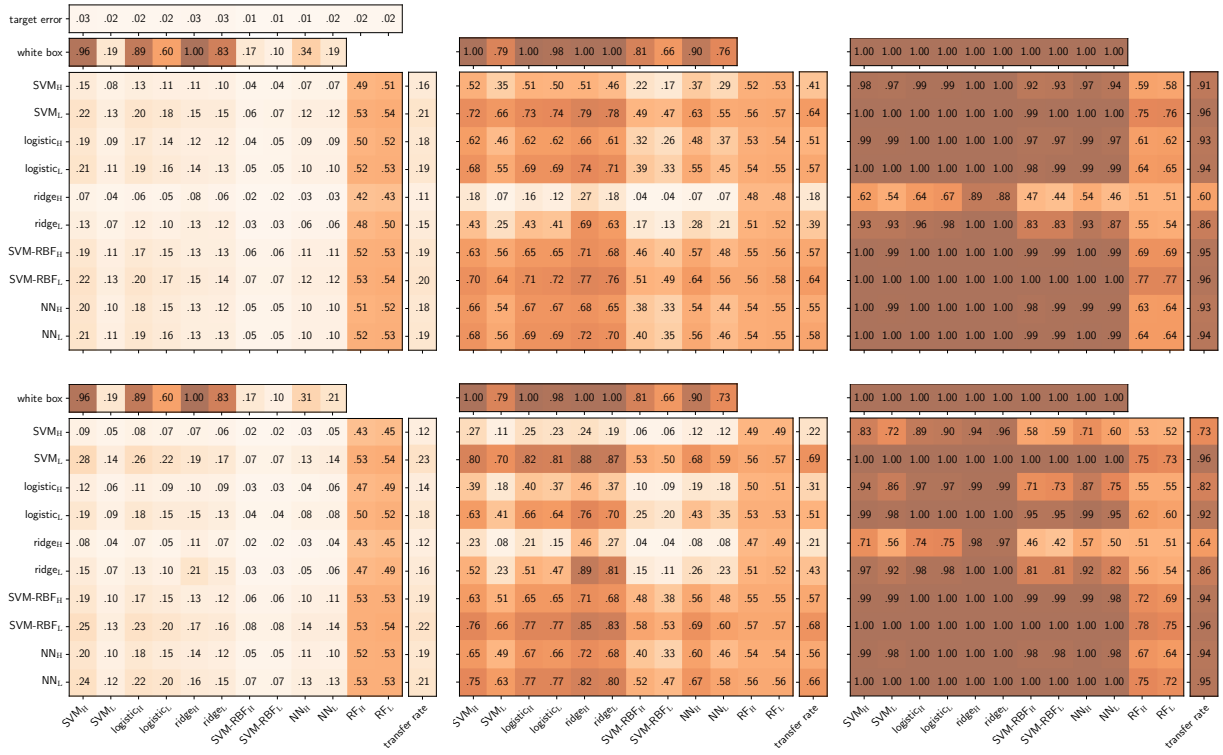
We consider 500 training samples, 1,000 validation samples to compute the attacks, and a separate set of 1,000 test samples to evaluate the error. The latter is computed against an increasing number of poisoning points fed into the training set, from 0% to 20% (corresponding to 125 poisoning points). The reported results are averaged on 10 independent, randomly-drawn data splits.

Similarly to the evasion case, we employ as surrogate learners: (*i*) linear SVMs with $C = 0.01$ (SVM_L) and $C = 100$ (SVM_H); (*ii*) logistic classifiers with $C = 0.01$ (logistic_L) and $C = 10$ (logistic_H); (*iii*) ridge classifiers with $\alpha = 100$ (ridge_L) and $\alpha = 10$ (ridge_H); and (*iv*) SVMs with RBF kernel with $\gamma = 0.01$ and $C = 1$ (SVM-RBF_L) and $C = 100$ (SVM-RBF_H). We additionally consider as target classifiers: (*v*) random forests with 100 base trees, each with a maximum depth of 6 for RF_L, and with no limit on the maximum depth for RF_H; (*vi*) feed-forward neural networks with two hidden layers of 200 neurons each and ReLU activations, trained via cross-entropy loss minimization with different regularization (NN_L with weight decay 0.01 and NN_H with no decay); and (*vii*) the Convolutional Neural Network (CNN) used in [99].

Results (Evasion)

The results for white-box evasion attacks are reported for all classifiers that fall under our framework and that can be tested for evasion with gradient-based attacks (SVMs, logistic, ridge, and NN). This excludes random forests, as they are not differentiable. We report the complete security evaluation curves in Figure 8.2(a), showing the mean test error (over 10 runs) against an increasing maximum admissible distortion ε.

In Figure 8.3 we report the results for black-box evasion, in which the attacks against the surrogate models (in rows) are transferred to the target models (in columns). The top plots show the results for surrogates trained using only 20% of the surrogate training data, while in the bottom plots the surrogates are trained using all surrogate data, *i.e.*, a training set of the same size as that of the target. These plots report (from left to right) the results for $\varepsilon \in \{1, 2, 5\}$.



(a) $\varepsilon = 1$ (b) $\varepsilon = 2$ (c) $\varepsilon = 5$

Figure 8.3: Black-box (transfer) evasion attacks on *MNIST*89. Each cell contains the test error of the target classifier (in columns) computed on the attack samples crafted against the surrogate (in rows). Matrices in the top (bottom) plots correspond to attacks crafted against surrogate models trained with 20% (100%) of the surrogate training data, and (from left to right) for $\varepsilon \in \{1, 2, 5\}$. The test error of each target classifier in the absence of attack (target error) and under (white-box) attack are also reported for comparison, along with the mean transfer rate of each surrogate across targets (rightmost column of each plot). Darker colors mean higher test error, *i.e.*, better transferability.

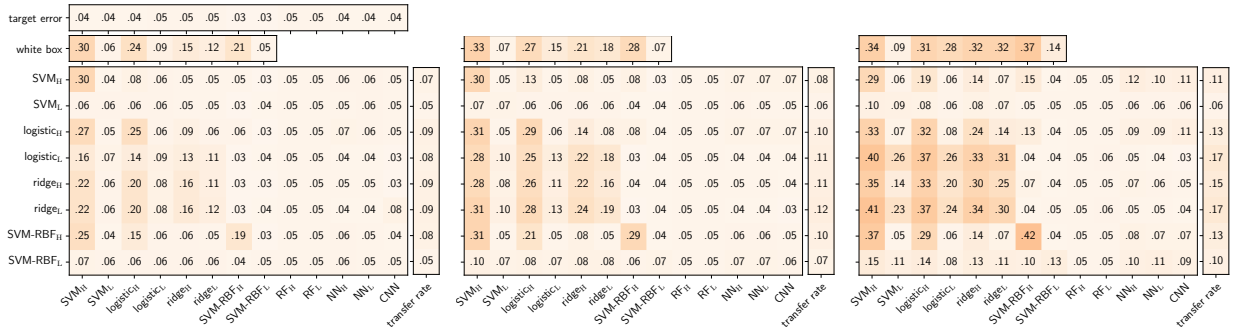
Results (Poisoning)

The results for white-box poisoning are reported in Figure 8.2(b), by means of security evaluation curves showing the average test error against an increasing fraction of poisoning points (over 10 runs).

Figure 8.4 reports the results for black-box poisoning. Conversely to evasion case, here we only use surrogates trained on all the surrogate data, *i.e.*, a training set of the same size as the one used for the target. The three plots report the results for (from left to right) {5%, 10%, 20%} fraction of poisoning points.

How does model complexity impact evasion attack success in the white-box setting?

In Figure 8.5 (left) we report the mean test error after the evasion attacks at $\varepsilon = 1$ for each target model against the size of its input gradients (metric S, averaged over the test samples and over the 10 runs). These results show that, for each learning algorithm, the low-complexity model has smaller input gradients, and it is less vulnerable to evasion than its high-complexity counterpart, confirming our theoretical hypotheses. This is also confirmed by the p-values reported in Table 8.1 (evasion, MNIST89, left column), obtained by running a binomial test for each learning algorithm to compare the white-box test error of the corresponding high-



(a) 5% poisoning (b) 10% poisoning (c) 20% poisoning

Figure 8.4: Black-box (transfer) poisoning attacks on *MNIST*89. Each cell contains the test error of the target classifier (in columns) computed on the attack samples crafted against the surrogate (in rows). Each plot reports the results for an increasing fraction (from left to right) {5%, 10%, 20%} of poisoning points. The test error of each target classifier in the absence of attack (target error) and under (white-box) attack are also reported for comparison, along with the mean transfer rate of each surrogate across targets (rightmost column of each plot). Darker colors mean higher test error, *i.e.*, better transferability.

	Evasion				**Poisoning**			
	MNIST89		**DREBIN**		**MNIST89**		**LFW**	
	$\epsilon = 1$	$\epsilon = 1$	$\epsilon = 5$	$\epsilon = 30$	5%	20%	5%	20%
SVM	<1e-2	<1e-2	<1e-2	<1e-2	<1e-2	<1e-2	<1e-2	0.75
logistic	<1e-2	<1e-2	<1e-2	0.02	<1e-2	<1e-2	0.10	0.21
ridge	<1e-2	<1e-2	<1e-2	<1e-2	0.02	<1e-2	0.02	0.75
SVM-RBF	<1e-2	<1e-2	<1e-2	<1e-2	<1e-2	<1e-2	<1e-2	0.11
NN	<1e-2	<1e-2	<1e-2	0.02				

Table 8.1: Statistical significance of our results. For each attack, dataset and learning algorithm, we report the p-values of two two-sided binomial tests, to respectively reject the null hypothesis that: (i) for white-box attacks, the test errors of the high- and low-complexity target follow the same distribution; and (ii) for black-box attacks, the transfer rates of the high- and low-complexity surrogate follow the same distribution. Each test is based on 10 samples, obtained by comparing the error of the high- and low-complexity models for each learning algorithm in each repetition. In the first (second) case, success corresponds to a larger test (transfer) error for the high-complexity target (low-complexity surrogate).

and low-complexity models. All the p-values are smaller than 0.05, which confirms 95% statistical significance. Recall that these results hold only when comparing models trained using the same learning algorithm. This means that we can compare, *e.g.*, the S metric of SVM_H against SVM_L, but not that of SVM_H against logistic_H. In fact, even though logistic_H exhibits the largest value of S, it is not the most vulnerable classifier. Another interesting finding is that non-linear classifiers tend to be less vulnerable than linear ones.

Similarly to the evasion case, high-complexity models with larger input gradients, as shown in Figure 8.6 (left), are more vulnerable to poisoning attacks than their low-complexity counterparts (*i.e.*, given that the same learning algorithm is used). This is also confirmed by the statistical tests of Table 8.1 (poisoning, MNIST89, left column). Therefore, model complexity plays a large role in the model's robustness also against poisoning attacks, confirming our analysis.

How do evasion attacks transfer between models in black-box settings?

For evasion attacks, it can be noted from Figure 8.3 that lower-complexity models (with stronger regularization) provide, on average, better surro-

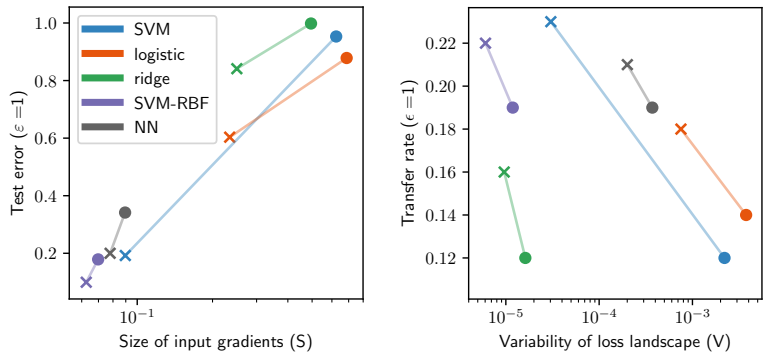


Figure 8.5: Evaluation of our transferability metrics for evasion attacks on *MNIST*89. *Left:* test error under attack ($\varepsilon = 1$) vs average size of input gradients (S) for low- (denoted with '×') and high- complexity (denoted with '∘') classifiers. *Right:* average transfer rate ($\varepsilon = 1$) vs variability of loss landscape (V).

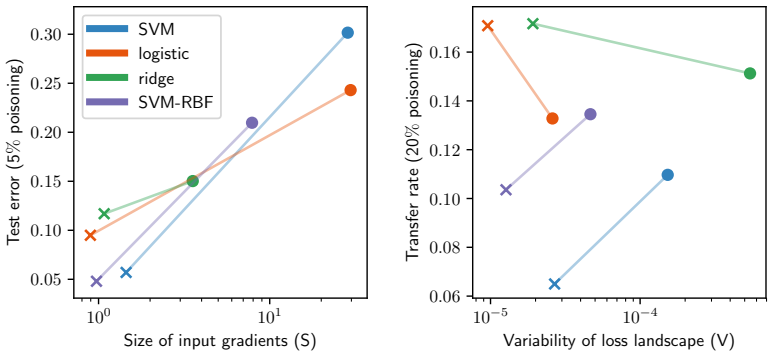


Figure 8.6: Evaluation of our transferability metrics for poisoning attacks on *MNIST*89. *Left:* test error under attack (5% poisoning points) vs average size of input gradients (S) for low- (denoted with '×') and high- complexity (denoted with '∘') classifiers. *Right:* average transfer rate (20% poisoning points) vs variability of loss landscape (V).

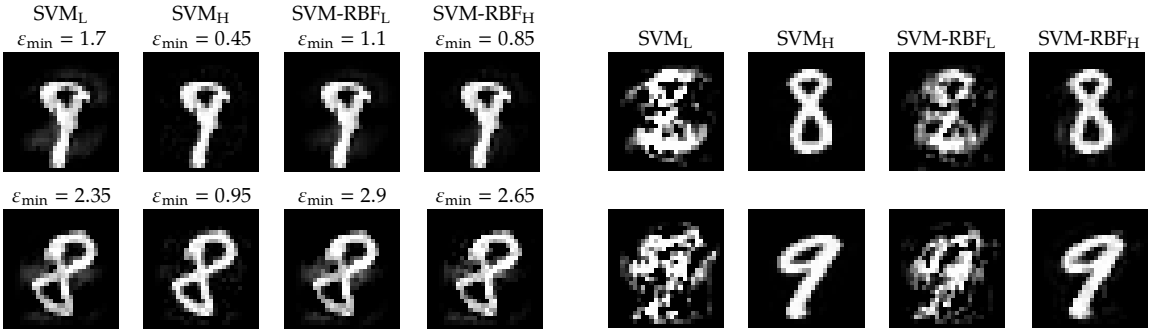
gate models. In particular, this can be seen best in the middle column for medium level of perturbation, in which the lower-complexity models (SVM_L, logistic_L, ridge_L, and SVM-RBF_L) provide on average higher error when transferred to other models. The reason is that they learn smoother and stabler functions, that are capable of better approximating the target function. Surprisingly, this holds also when using only 20% of training data, as the black-box attacks relying on such low-complexity models still transfer with similar test errors. This means that most classifiers can be attacked in this black-box setting with almost no knowledge of the model, and no query access, but provided that one can get a small amount of data similar to that used to train the target model.

These findings are also confirmed by looking at the variability of the loss landscape, computed as discussed in Section 5.2 (by considering 10 different training sets), and reported in Figure 8.5 (right) against the average transfer rate of each surrogate model. It is clear that higher-variance classifiers are less effective as surrogates than their less-complex counterparts, as the former tend to provide worse, unstable approximations of the target classifier. To verify this result, for each learning algorithm we also compare the mean transfer errors of high- and low- complexity surrogates with a binomial test, reported in Table 8.1 (evasion, MNIST89, right column). All the p-values are smaller than 0.05 even in this case, confirming 95% statistical significance.

For poisoning attacks (Figure 8.4), the best surrogates are those matching the complexity of the target, as they tend to be better aligned and to share similar local optima, except for low-complexity logistic and ridge surrogates, which seem to transfer better to linear classifiers. Thus, according to our findings, reducing the variability of the loss landscape (V) of the surrogate model is less important than finding a good alignment between the surrogate and the target. In fact, from Figure 8.6 (right) it is

evident that increasing V is even beneficial for SVM-based surrogates, result also confirmed by the statistically significant tests reported in Table 8.1 (poisoning, MNIST89, right column).

It is worth nothing that, after a visual inspection of the adversarial examples (Figure 8.7(a)) and poisoning points (Figure 8.7(b)), it appears that the digits crafted against high-complexity classifiers are only minimally perturbed, while the ones computed against low-complexity classifiers exhibit larger, visible perturbations. This is due to the instability induced by high-complexity models into the loss function, whose sudden changes cause the presence of closer local optima to the initial attack point.



(a) Adversarial examples (evasion attacks) (b) Poisoning attacks

Figure 8.7: Adversarial digits crafted from the *MNIST*89 dataset to attack a linear SVM and the SVM-RBF. Larger perturbations are required to mislead low-complexity classifiers (L), while smaller ones suffice to fool high-complexity classifiers (H). For evasion attacks, the values of ε_{min} reported here correspond to the minimum perturbation required to evade detection.

On the vulnerability of random forests. A noteworthy finding is that random forests can be successfully attacked at test-time by small perturbations optimized using most of the other models (see last two columns of each plot in Figure 8.3). We looked at the learned models and discovered that trees are actually often susceptible to small changes. In one example, a node of the tree checked if a particular feature value was above 0.002, and classified the samples as digit 8 if that condition holds (or as digit 9 otherwise). The attack modified that feature from 0 to 0.028, causing it to be immediately misclassified. This vulnerability is intrinsic in the selection process of the threshold values used by these decision trees to split each node. The threshold values are selected among the existing values in the dataset (to correctly handle categorical attributes). Therefore, for pixels which are highly discriminant (*e.g.*, mostly black for one class and white for the other), the threshold will be either very close to one extreme or the other, making it easy to subvert the prediction by a small change. Since ℓ_2-norm attacks change almost all feature values, with high probability the attack modifies at least one feature on every path of the tree, causing misclassification.

Surprising however, random forests are instead quite robust to poisoning (see Figure 8.4), as well as NNs, when attacked with low-complexity linear surrogates. The reason may be that these target classifiers have a large capacity, and can thus fit *outlying* samples (like the digits crafted against low-complexity classifiers in Figure 8.7(b)) without affecting the classification of the other training samples.

Is gradient alignment an effective transferability metric?

In Figure 8.8, we report for the evasion case the gradient alignment (R) computed between the surrogate and the target models (left plot), and (right plot) the Pearson correlation coefficient $\rho(\hat{\boldsymbol{\delta}}, \boldsymbol{\delta})$ between the perturbation optimized against the surrogate (*i.e.*, the black-box perturbation $\hat{\boldsymbol{\delta}}$) and that optimized against the target (*i.e.*, the white-box perturbation $\boldsymbol{\delta}$). We observe immediately that gradient alignment provides an accurate measure of transferability: the higher the cosine similarity, the higher the correlation (meaning that the adversarial examples crafted against the two models are similar). We correlate these two measures in Figure 8.9 (left), and show the statistical significance for both Pearson and Kendall coefficients (p-values $\ll 0.05$ in both cases). In Figure 8.9 (right) we also correlate gradient alignment with the ratio between the test error of the target model in the black- and white- box setting (extrapolated from the matrix corresponding to $\varepsilon = 1$ in the bottom row of Figure 8.3), as suggested by our theoretical derivation. The corresponding permutation tests confirm once again statistical significance.

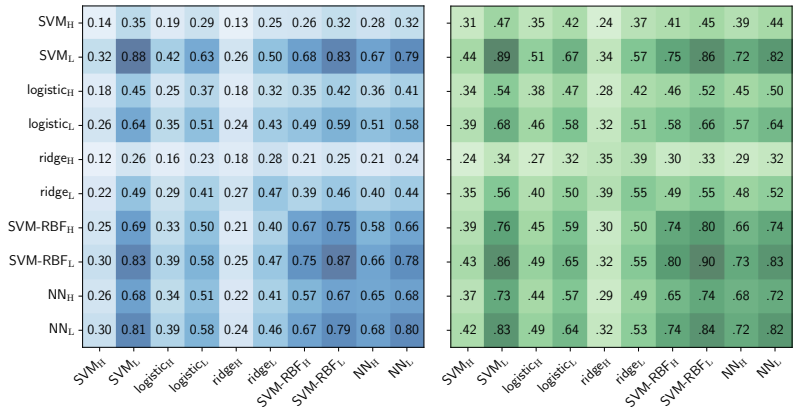


Figure 8.8: Gradient alignment and perturbation correlation for evasion attacks on *MNIST*89. *Left:* gradient alignment R between surrogate (rows) and target (columns) classifiers, averaged on the unmodified test samples. *Right:* Pearson correlation coefficient $\rho(\boldsymbol{\delta}, \hat{\boldsymbol{\delta}})$ between white-box and black-box perturbations for $\varepsilon = 5$ maximum ℓ_2 distance.

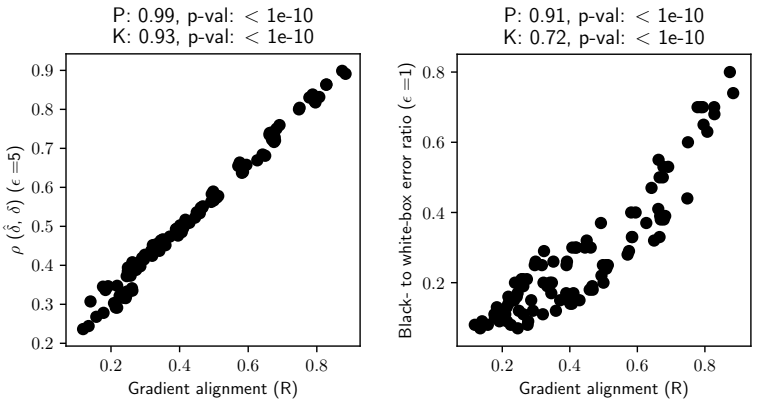


Figure 8.9: Evaluation of our transferability metrics for evasion attacks on *MNIST*89. Pearson (P) and Kendall (K) correlations are reported along with the p-values obtained from a permutation test to assess statistical significance. *Left:* Pearson coefficient $\rho(\hat{\boldsymbol{\delta}}, \boldsymbol{\delta})$ between black-box ($\hat{\boldsymbol{\delta}}$) and white-box ($\boldsymbol{\delta}$) perturbations (values in Figure 8.8, right) vs gradient alignment R (values in Figure 8.8, left) for each target-surrogate pair. *Right:* correlation of gradient alignment with the ratio between the test error of the target model in the black- and white- box setting.

The same can be witnessed for poisoning attacks by looking to the gradient alignment metric reported in Figure 8.10, which is again not only correlated to the similarity between black- and white- box perturbations (Figure 8.11, left), but also to the ratio between the black- and white-box test errors (Figure 8.11, right). Interestingly, these error ratios are larger than 1 in some cases, meaning that attacking a surrogate model can be more effective than running a white-box attack against the target. A similar phenomenon has been observed for evasion attacks [72], and

it is due to the fact that optimizing attacks against a *smoother* surrogate may find better local optima of the target function (*e.g.*, by overcoming gradient obfuscation [100]).

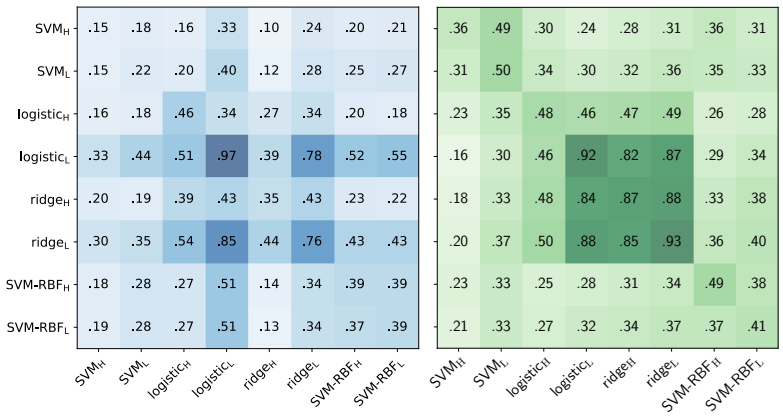


Figure 8.10: Gradient alignment and perturbation correlation for poisoning attacks on *MNIST*89. *Left:* Gradient alignment R between surrogate (rows) and target (columns) classifiers, averaged on the unmodified test samples. *Right:* Pearson correlation coefficient $\rho(\boldsymbol{\delta}, \hat{\boldsymbol{\delta}})$ between white-box and black-box perturbations using 20% poisoning points.

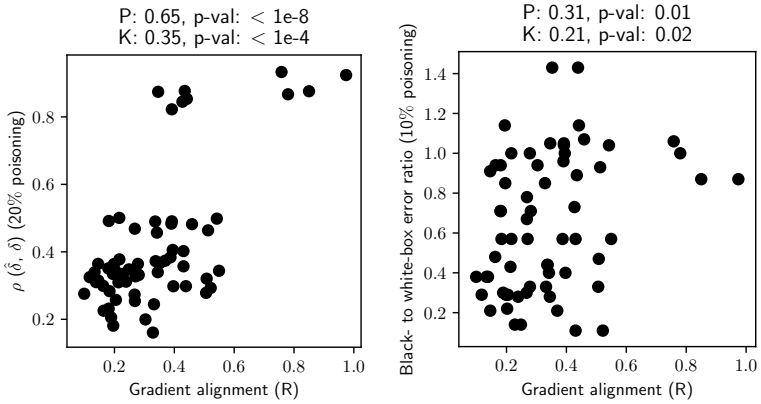


Figure 8.11: Evaluation of our transferability metrics for poisoning attacks on *MNIST*89. Pearson (P) and Kendall (K) correlations are reported along with the p-values obtained from a permutation test to assess statistical significance. *Left:* Pearson coefficient $\rho(\hat{\boldsymbol{\delta}}, \boldsymbol{\delta})$ between black-box ($\hat{\boldsymbol{\delta}}$) and white-box ($\boldsymbol{\delta}$) perturbations (values in Figure 8.10, right) vs gradient alignment R (values in Figure 8.10, left) for each target-surrogate pair. *Right:* correlation of gradient alignment with the ratio between the test error of the target model in the black- and white- box setting.

8.1.2 Android Malware Detection

This section describes another application case where a machine learning model is trained to distinguish between malicious and benign applications for the Android mobile operating system. In this binary classification task, the system should assign the class 1 to malign samples, and the label 0 to all the other inputs. We first provide some background on the structure of Android applications [18]; secondly, we describe Drebin [212], the Android malware detector that we consider in our case study; third, we report the experimental analysis on the transferability property of evasion attacks against these systems.

Android Background

Android applications are compressed in `apk` files, *i.e.*, archives that contain the following elements: (*i*) the `AndroidManifest.xml` file; (*ii*) one or more `classes.dex` files; (*iii*) resource and asset files, such as native libraries or images; (*iv*) additional `xml` files that define the application layout. Since Drebin only analyzes the `AndroidManifest.xml` and the `classes.dex` files, we briefly describe them below.

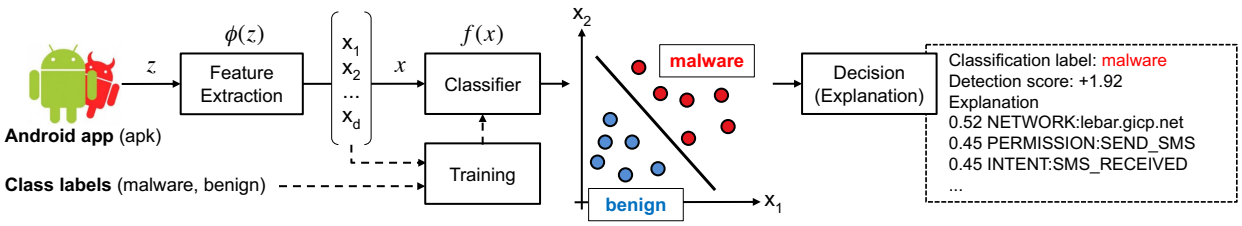


Figure 8.12: A schematic representation ([21]) of Drebin [212]. First, applications are represented as binary vectors in a d-dimensional feature space. A linear classifier is then trained on an available set of malware and benign applications, assigning a weight to each feature. During classification, unseen applications are scored by the classifier by summing up the weights of the present features: if $f(\boldsymbol{x}) \geq 0$, they are classified as malware. Drebin also explains each decision by reporting the most suspicious (or benign) features present in the app, along with the weight assigned to them by the linear classifier.

Android Manifest (`manifest`). The basic information about the Android application is held in the `AndroidManifest.xml`, including its package name or the supported API levels, together with the declaration of its *components*, *i.e.*, parts of code that perform specific actions. For example, one component might be associated with a screen visualized by the user (*activity*), or to the execution of background tasks (*services*). App components can also perform actions (through *receivers*) on the occurrence of specific events, *e.g.*, a change in the device's connectivity status (`CONNECTIVITY_CHANGE`) or the opening of an application (`LAUNCHER`). The `manifest` also contains the list of *hardware components* and *permissions* requested by the application to properly work (*e.g.*, Internet access).

Dex bytecode (`dexcode`). The `classes.dex` file embeds the compiled source code of the application, including all the user- implemented methods and classes; the bytecode can be executed with the Dalvik Virtual Machine (until Android 4.4) or the Android runtime (ART). The `classes.dex` may contain specific API calls that access sensitive resources such as personal contacts (*suspicious calls*). Additionally, it contains all, system-related, *restricted API calls* that require specific permissions (*e.g.*, writing to the device's storage). Finally, it can also contain references to *network addresses* that might be contacted by the application.

Drebin

The majority of the approaches for Android malware detection employ static and dynamic analyses, extracting information such as permissions, communications through Inter-Component Communication (ICC), and system- and user-implemented API calls [212, 213, 214, 215, 216].

Drebin [212] is among the most popular and used architectures. It performs the detection of Android malware through a static analysis of the applications. In a first phase (training), it employs a set of benign and malicious apps provided by the user to determine the features that will be used for detection (meaning that the feature set will be strictly dependent on the training data). Such features are then embedded into a *sparse*, high-dimensional vector space. Finally, after learning a linear model like a support vector machine, the system is able to perform the classification of previously-unseen apps. An overview of the architecture of this system is given in Figure 8.12.

Feature extraction. Drebin processes the data by statically analyzing a set of n training Android applications to construct a suitable feature space. The extracted features are presented as *strings* and organized in 8 different feature sets, as listed in Table 8.2.

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

Table 8.2: Overview of feature sets in the Drebin system [212].

Android applications are then mapped onto the feature space as follows. Let us assume that an app is represented as an object $z \in \mathcal{Z}$, being $\mathcal{Z}$ the abstract space of all `apk` files. We denote with $\Phi : \mathcal{Z} \mapsto \mathcal{F}$ a function that maps an `apk` file z to a d-dimensional feature vector $\boldsymbol{x} = [x^{(1)}, \ldots, x^{(d)}] \in \mathcal{F} = \{0,1\}^d$, where each feature is set to 1 (0) if the corresponding *string* is present (absent) in the `apk` file z. An application encoded in this feature space may thus look like the following:

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

Learning, Classification, and Explanation. Drebin employs a linear model such as a Support Vector Machine (SVM) to perform detection of malicious apps, which can be efficiently trained in a sparse feature space. Then, during classification, unseen apps are labeled as malware if the score is $f(\boldsymbol{x}) \geq 0$, or as benign otherwise.

Leveraging interpretable models (see Section 2.3.1), explanations are provided by-design by Drebin, reporting, for any given application, the most influential features, *i.e.*, the ones that are assigned the highest absolute weights by the classifier. For instance, in Figure 8.12 we report an example of an app correctly identified as malware since it connects to a suspicious URL and uses SMS as a side-channel for communication. It should be noted, however, that in the standard Drebin implementation the relevance of each feature corresponds exactly to the value of the relative parameter in $\mathbf{w}$, thus components not actually present in a certain input app may appear as relevant for the system.

Experimental Setup

The *Drebin* data [212] consists of around 120,000 legitimate and around 5000 malicious Android applications, cataloged using the VirusTotal service. A sample is labeled as malicious (or positive, $y = +1$) if it is classified as such by at least five out of ten anti-virus scanners, while it is flagged as legitimate (or negative, $y = -1$) otherwise. The sparse feature vector resulting from the static analysis of the applications, consists of

more than a million binary feature. Since we are dealing with *sparse binary features*, we use the ℓ_1 norm for the attack.

We use 30,000 samples to learn the surrogate and the target classifiers, and the remaining 66,944 samples are used for testing. The classifiers and their hyperparameters are the same used for the experiments on *MNIST*89, apart from (*i*) the number of hidden neurons for NN_H and NN_L, set to 200, (*ii*) the weight decay of NN_L, set to 0.005; and (*iii*) the maximum depth of RF_L, set to 59.

We perform feature selection to retain those 5,000 features which maximize information gain, *i.e.*, $|p(x^{(k)} = 1|y = +1) - p(x^{(k)} = 1|y = -1)|$. While this process does not significantly affect the detection rate (which is only reduced by 2%, on average, at 0.5% false positive rate), it reduces even more the computational complexity of the classification step.

In each experiment, we run white-box and black-box evasion attacks on 1,000 distinct malware samples (randomly selected from the test data) against an increasing number of modified features in each malware $\varepsilon \in \{0, 1, 2, \ldots, 30\}$. This is achieved by imposing the ℓ_1 constraint $\|\boldsymbol{x}' - \boldsymbol{x}\|_1 \leq \varepsilon$ (Equation 4.4). As in previous works, we further restrict the capabilities of the attacker to only *inject* features into each malware sample (using the bounds in Equation 4.5), to avoid compromising its intrusive functionality [21, 74].

Results

To evaluate the impact of the aforementioned evasion attack, we measure the evasion rate (*i.e.*, the fraction of malware samples misclassified as legitimate) at 0.5% false positive rate (*i.e.*, when only 0.5% of the legitimate samples are misclassified as malware).

The results for white-box and black-box evasion attacks, reported in Figure 8.13, and Figure 8.14, respectively, along with the metrics evaluation in Figure 8.15, Figure 8.16, and Figure 8.17, and the statistical tests in Table 8.1, confirm the main findings described before on the *MNIST*89 data. One significant difference is that random forests are much more robust in this case. The reason is that the ℓ_1-norm attacks (differently from ℓ_2) only change a small number of features, and thus the probability that all the features considered by the ensemble trees are modified is very low.

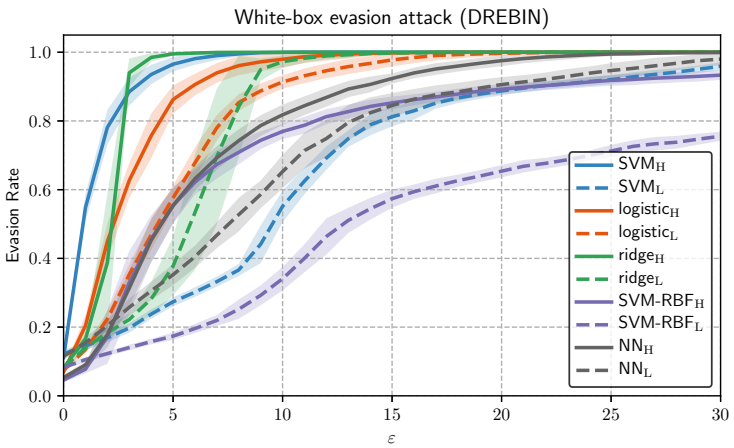
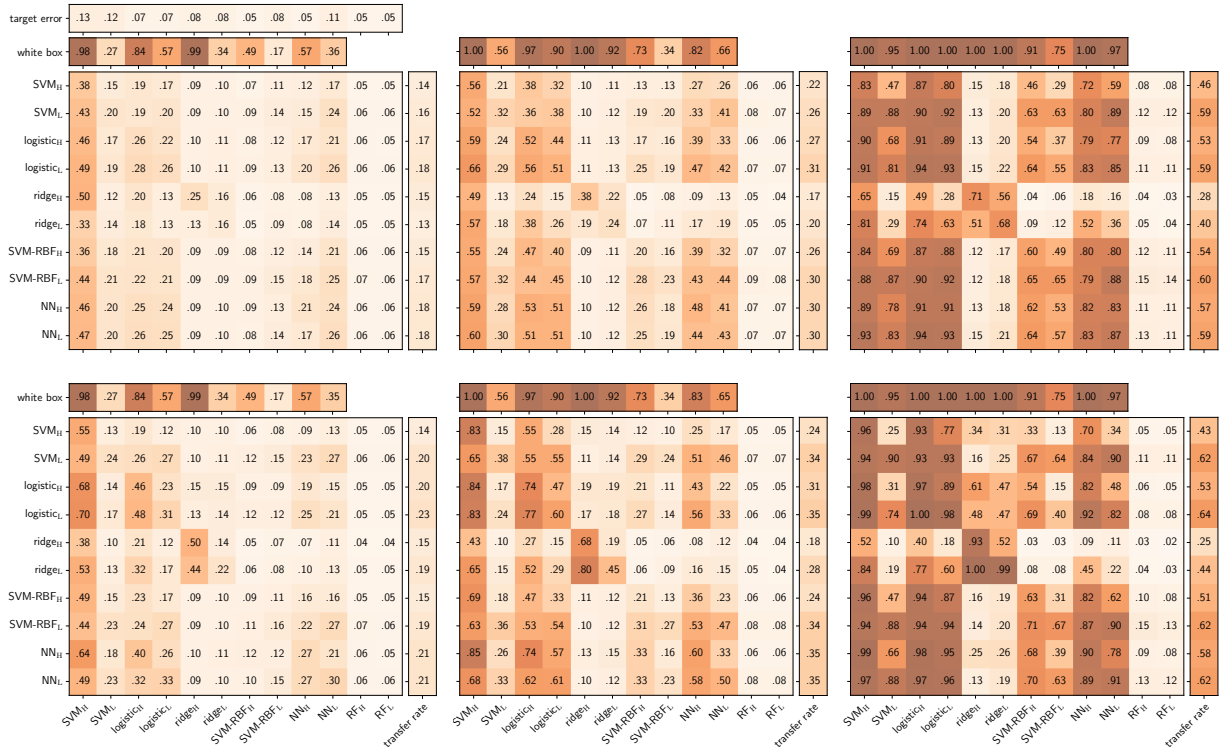


Figure 8.13: Security evaluation curves for white-box evasion attacks on *Drebin*. Evasion rate against increasing maximum perturbation ε.



(a) $\varepsilon = 5$ (b) $\varepsilon = 10$ (c) $\varepsilon = 30$

Figure 8.14: Black-box (transfer) evasion attacks on *Drebin*. Each cell contains the test error of the target classifier (in columns) computed on the attack samples crafted against the surrogate (in rows). Matrices in the top (bottom) plots correspond to attacks crafted against surrogate models trained with 20% (100%) of the surrogate training data, and (from left to right) for $\varepsilon \in \{5, 10, 30\}$. The test error of each target classifier in the absence of attack (target error) and under (white-box) attack are also reported for comparison, along with the mean transfer rate of each surrogate across targets (rightmost column of each plot). Darker colors mean higher test error, *i.e.*, better transferability.

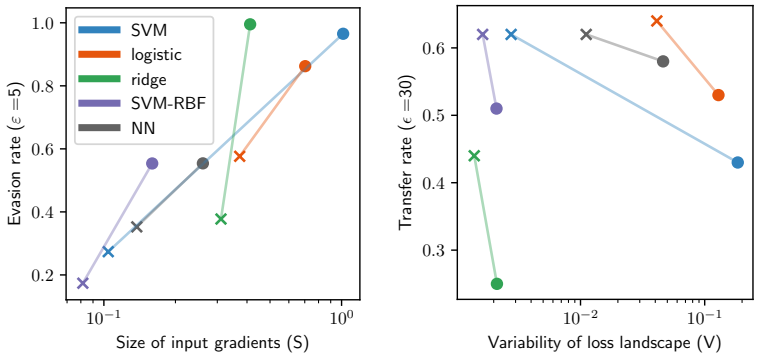


Figure 8.15: Evaluation of our transferability metrics for evasion attacks on *Drebin*. *Left:* test error under attack vs average size of input gradients (S) for low- (denoted with '×') and high- complexity (denoted with '∘') classifiers. *Right:* average transfer rate vs variability of loss landscape (V).

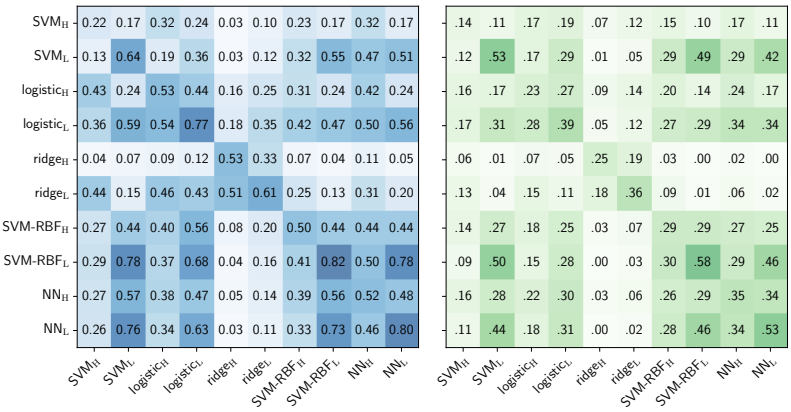


Figure 8.16: Gradient alignment and perturbation correlation for evasion attacks on *Drebin*. *Left:* Gradient alignment R between surrogate (rows) and target (columns) classifiers, averaged on the unmodified test samples. *Right:* Pearson correlation coefficient $\rho(\boldsymbol{\delta}, \hat{\boldsymbol{\delta}})$ between white-box and black-box perturbations for $\varepsilon = 30$ maximum ℓ_1 distance.

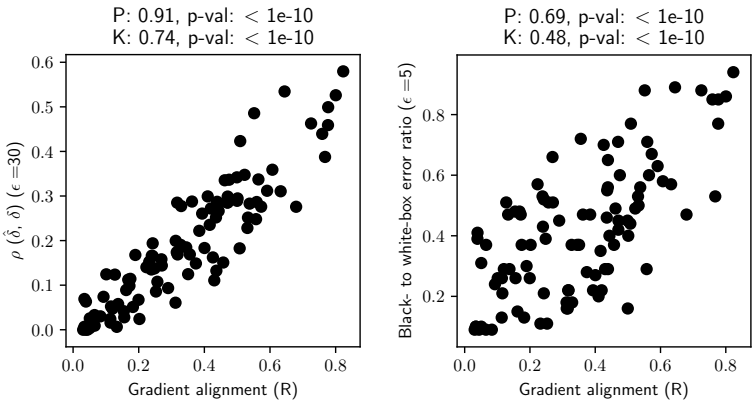


Figure 8.17: Evaluation of our transferability metrics for evasion attacks on *Drebin*. Pearson (P) and Kendall (K) correlations are reported along with the p-values obtained from a permutation test to assess statistical significance. *Left:* Pearson coefficient $\rho(\hat{\boldsymbol{\delta}}, \boldsymbol{\delta})$ between black-box ($\hat{\boldsymbol{\delta}}$) and white-box ($\boldsymbol{\delta}$) perturbations (values in Figure 8.16, right) vs gradient alignment R (values in Figure 8.16, left) for each target-surrogate pair. *Right:* correlation of gradient alignment with the ratio between the test error of the target model in the black- and white- box setting.

8.1.3 Face Recognition

The last case we consider is a machine learning model which has to verify the identity of specific persons from an image of their face. We use the Labeled Faces in the Wild (*LFW*) dataset [217, 218], which includes faces of famous peoples collected on Internet. We consider the six identities with the largest number of images in the dataset, assigning to the person with most images the positive class, and labeling all the others as negative. The resulting dataset consists of 530 positive and 758 negative images. The classifiers and their hyperparameters are the same used for *MNIST*89 experiments, except that we set: (i) $C = 0.1$ for logistic_L, (ii) $\alpha = 1$ for ridge_H, (iii) $\gamma = 0.001, C = 10$ for SVM-RBF_L, (iv) $\gamma = 0.001, C = 1000$ for SVM-RBF_H, and (v) weight decay to 0.001 for NN_L. We run 10 repetitions with 300 samples in each training, validation and test set.

Results

The results for white-box and black-box poisoning attacks are shown in Figure 8.18 and Figure 8.19, respectively. The main findings on transferability discussed for *MNIST*89 are confirmed by the analysis of our metrics reported by Figure 8.20, Figure 8.21, and Figure 8.22, as well as by the statistical tests for significance reported in Table 8.1. In this case, there is no significant distinction between the mean transfer rates of high- and low- complexity surrogates, probably due to the reduced size of the used training sets.

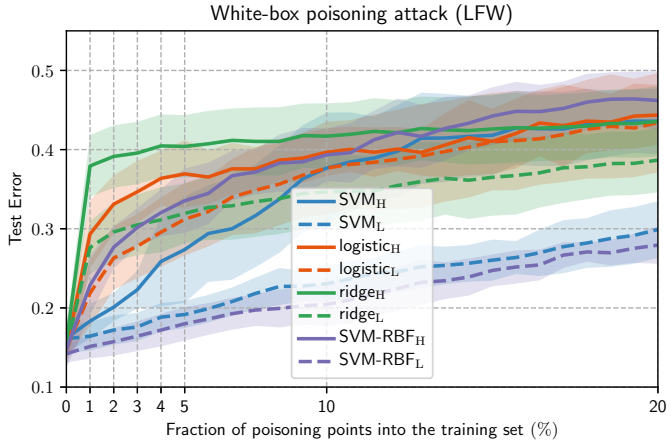
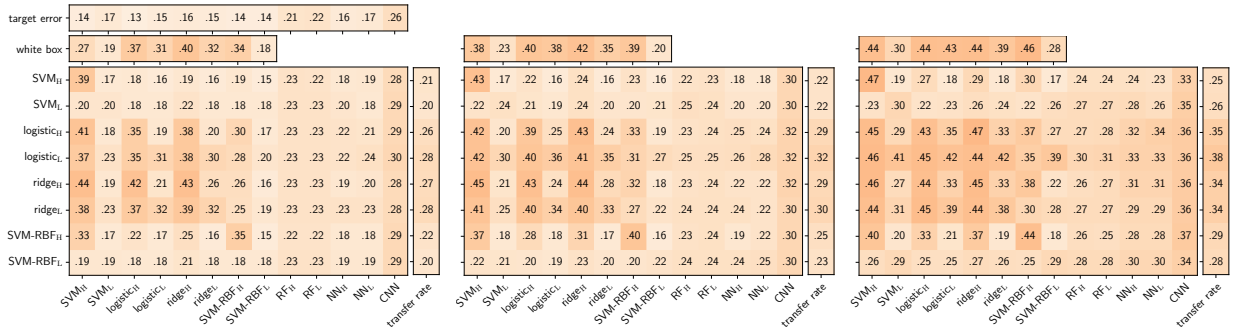


Figure 8.18: Security evaluation curves for white-box poisoning attacks on *LFW*. Test error against an increasing fraction of poisoning points.



(a) 5% poisoning (b) 10% poisoning (c) 20% poisoning

Figure 8.19: Black-box (transfer) poisoning attacks on *LFW*. Each cell contains the test error of the target classifier (in columns) computed on the attack samples crafted against the surrogate (in rows). Each plot reports the results for an increasing fraction (from left to right) {5%, 10%, 20%} of poisoning points. The test error of each target classifier in the absence of attack (target error) and under (white-box) attack are also reported for comparison, along with the mean transfer rate of each surrogate across targets (rightmost column of each plot). Darker colors mean higher test error, *i.e.*, better transferability.

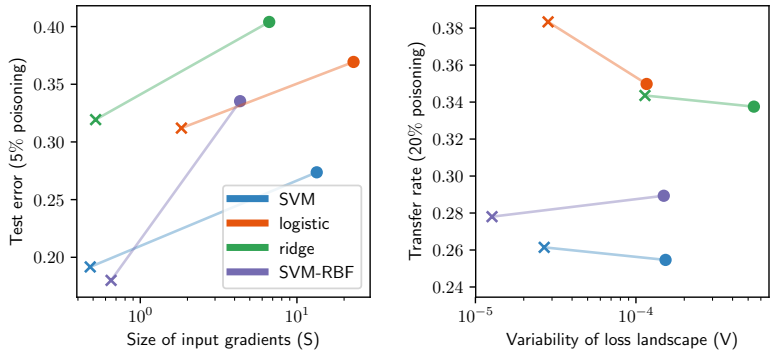


Figure 8.20: Evaluation of our transferability metrics for poisoning attacks on *LFW*. *Left:* test error under attack vs average size of input gradients (S) for low- (denoted with '×') and high- complexity (denoted with '○') classifiers. *Left:* average transfer rate vs variability of loss landscape (V).

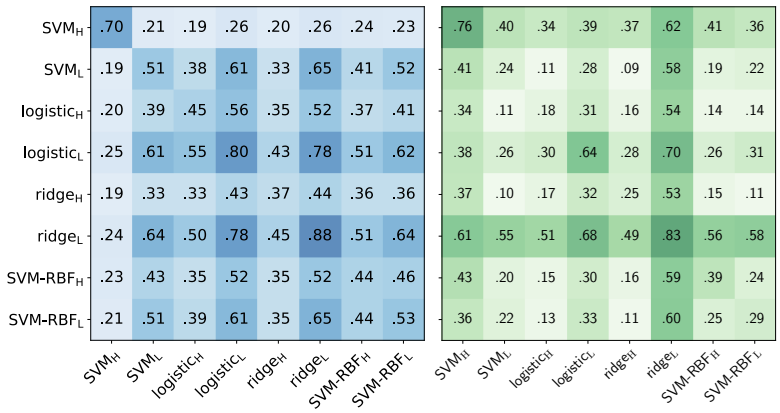


Figure 8.21: Gradient alignment and perturbation correlation for poisoning attacks on *LFW*. *Left:* Gradient alignment R between surrogate (rows) and target (columns) classifiers, averaged on the unmodified test samples. *Right:* Pearson correlation coefficient $\rho(\boldsymbol{\delta}, \hat{\boldsymbol{\delta}})$ between white-box and black-box perturbations at 20% poisoning.

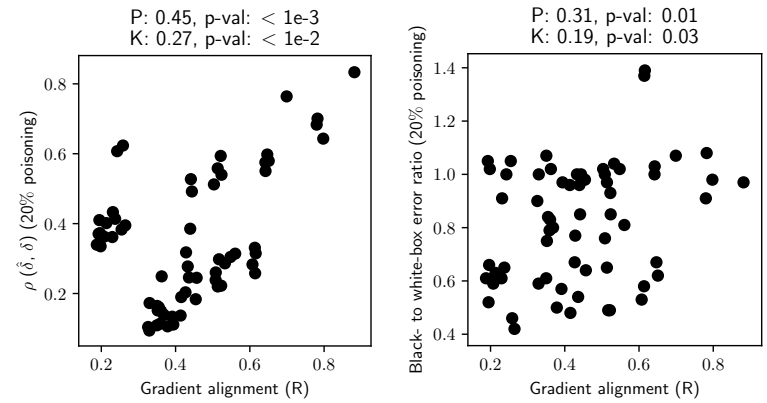


Figure 8.22: Evaluation of our transferability metrics for poisoning attacks on *LFW*. Pearson (P) and Kendall (K) correlations are reported along with the p-values obtained from a permutation test to assess statistical significance. *Left:* Pearson coefficient $\rho(\hat{\boldsymbol{\delta}}, \boldsymbol{\delta})$ between black-box ($\hat{\boldsymbol{\delta}}$) and white-box ($\boldsymbol{\delta}$) perturbations (values in Figure 8.21, right) vs gradient alignment R (values in Figure 8.21, left) for each target-surrogate pair. *Right:* correlation of gradient alignment with the ratio between the test error of the target model in the black- and white- box setting.

8.1.4 Summary

The experimental results reported in the previous sections demonstrate the main finding of our analysis: *attack transferability strongly depends on the complexity of the target model*, *i.e.*, on its inherent vulnerability. Also, this security-critical property of adversarial attacks can be measured effectively by our proposed metrics, with interesting insights provided by each of them.

Size of input gradients. Firstly, reducing the size of input gradients, *e.g.*, via regularization, may allow to learn more robust classifiers not only against evasion [219, 220, 221, 222] but also against poisoning availability attacks. Also, in general, non-linear models are more robust than linear models to both threats.

Gradient alignment. Secondly, even though it cannot be directly measured in black-box scenarios, transferability is also impacted by the surrogate model's alignment with the target model. For evasion attacks, low-complexity surrogate classifiers provide stabler gradients which are better aligned, on average, with those of the target models; thus, it is generally preferable to use strongly-regularized surrogates. For poisoning attacks, instead, gradient alignment tends to improve when the surrogate matches the complexity (regularization) of the target (which may be estimated using techniques from [223]).

Variability of the loss landscape. Third, surrogate loss functions that are stabler and have lower variance tend to encourage gradient-based attack algorithms to find better local optima (see Figure 8.23). As less complex models exhibit a lower variance of their loss function, they typically result in better surrogates.

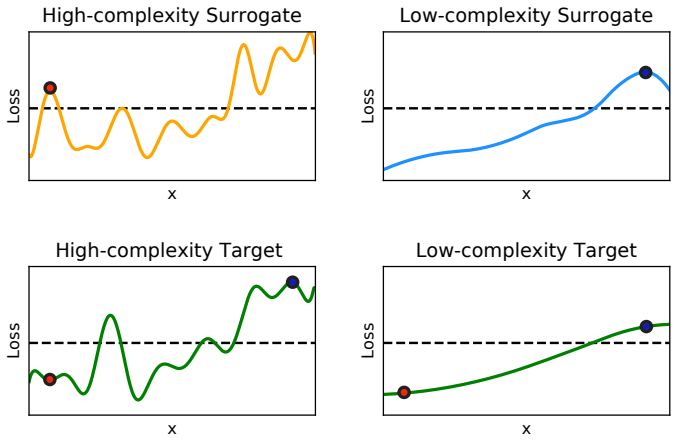


Figure 8.23: Conceptual representation of transferability [16]. We show the attack loss as a function of a single feature, denoted in this context as x. The top row includes two surrogate models (*high* and *low* complexity), while the bottom row includes two models as targets. The adversarial samples are represented as red dots for the high-complexity surrogate, and as blue dots for the low-complexity surrogate. If the adversarial loss is below a certain threshold (*i.e.*, the black horizontal dashed line), the point is correctly classified, otherwise it is misclassified. The attack computed against the high-complexity model (top left) lays in a local optimum due to the irregularity of the objective. This point is not effective even against the same classifier trained on a different dataset (bottom left) due to the variance of the high-complexity classifier. The attack computed against the low complexity model (top right), instead, succeeds against both low- and high- complexity targets (bottom left and bottom right, respectively).

8.2 Relevance Vectors for Model (Global) Explanations

In this section, we leverage our explainability technique based on highly-interpretable relevance vectors (Section 6.1), to provide local and global interpretations for linear and non-linear (including non-differentiable) classifiers employed on Drebin [212], the Android malware detection system introduced in Section 8.1.2. As previously hinted, this analysis also provides some interesting insights on the security of such algorithms against adversarial manipulations [21, 224].

Experimental Setup

Similarly to Section 8.1.2, we randomly select 60,000 apps from the *Drebin* data to train the learning algorithms, and use the rest for testing. The results are averaged over 5 independent repetitions.

We compare the standard Drebin implementation based on a linear Support Vector Machine (SVM), against a SVM with the RBF kernel (SVM-RBF), and a (non-differentiable) Random Forest (RF). As previously discussed, a surrogate model is needed to interpret the latter; to this end, SVMs with the RBF kernel have been proposed as a reliable approximation of ensemble classifiers [225]. The surrogate in this case is trained on the training set relabeled by the random forest, providing an approximation of the original decision function more than 99% accurate, on average, on the corresponding test sets.

We optimize the hyperparameters of each classifier through a 3-fold cross-validation procedure. In particular, we optimize $C \in \{10^{-2}, 10^{-1}, \ldots, 10^{2}\}$ for both linear and non-linear SVMs, the RBF kernel parameter $\gamma \in \{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$ for the SVM-RBF, and the number of estimators $\mathsf{ne} \in \{5, 10, \ldots, 30\}$ for the random forest.

Results

To validate the detection performance of each classifier, we start by reporting in Figure 8.24 the Receiver Operating Characteristic (ROC) curve averaged over the 5 repetitions. We recall that, in the context of malware detection, the *detection rate* represents the probability that a malicious sample is correctly labeled by the classifier.

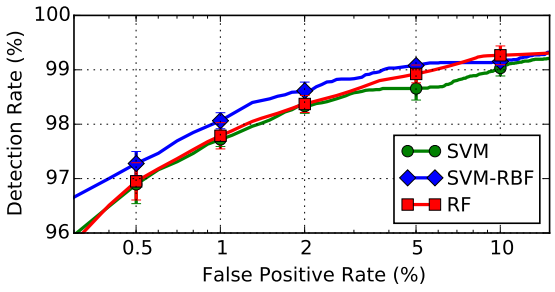


Figure 8.24: Average ROC curves for the given classifiers trained on the *Drebin* data. The *detection rate* represents the probability that a malicious sample is correctly labeled by the classifier.

We then perform an analysis of the models learned by each algorithm by applying our global explanation method (Equation 6.3) to M tests

samples. As previously discussed, to improve the interpretability of the results, we average the local relevance vectors $\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_M$ over different groups of samples: benign apps, malware, and the apps from the top-15 malware families with the largest number of available samples in the *Drebin* dataset (see Table 8.3).

Family	#	Family	#	Family	#
FakeInstaller	901	BaseBridge	318	Geinimi	88
DroidKungFu	640	Iconosys	149	DroidDream	81
Plankton	609	Kmin	144	LinuxLotoor	69
Opfake	591	FakeDoc	128	MobileTx	68
GingerMaster	332	Adrd	88	GoldDream	67

Table 8.3: Top 15 malware families by number of samples in the *Drebin* test set.

The results are shown in Figure 8.25. For each group of samples (benign apps, malware, `FakeInstaller` apps, etc.), we report a *compact* and a *fine-grained* analysis of the global relevance vectors $\bar{\boldsymbol{r}}$. In the compact analysis (top plots), we further average these vectors over each feature sets $S_1, \ldots, S_8$ (Table 8.2). In the fine-grained analysis (bottom plots), we report the global relevance for the top-44 features, selected by aggregating the top 5 features with the highest average relevance value for each group of samples. The components are sorted from top to bottom by their corresponding feature sets.

This representation demonstrates how *highly-interpretable* are the relevance vectors produced by our method, even for a non-skilled user. Being the values normalized, considering their absolute values in the [0, 100] range, one can immediately observe to which components is assigned the highest or the lower relevance, and thus compare the behavior learned by each classifier. Using other explainability techniques for which the magnitude of each attribution depends on the characteristics of the model, a similar comparative analysis can be harder or impossible to perform. Moreover, the tested classifiers, after being explained by our technique, can be considered both *simulatable* and *decomposable*, as the user is able to contemplate the entire model at once (using the compact analysis), or evaluate the contribution of each of the relevant components (using the fine-grained representation).

Discussion

The compact analysis highlights the importance of permissions (S_2) and suspicious API calls (S_7 group) for the detection of malware. This is reasonable, as the majority of the malicious samples require permissions to perform specific actions, like stealing contacts and opening SMS and other side communication channels. The fine-grained analysis provides a more detailed characterization of the aforementioned behavior, highlighting how each classifier learns a specific *behavioral signature* for each group of samples. In particular, the different malware families are characterized by their communication channels (*e.g.*, SMS and HTTP), by the amount of stolen information and accessed resources, and by specific application components or URLs (S_3 and S_8).

This analysis also highlights a fundamental security-related vulnerability of these classifiers: they tend to assign high relevance to a very small set of features in each decision, both at a local and at a global scale. In

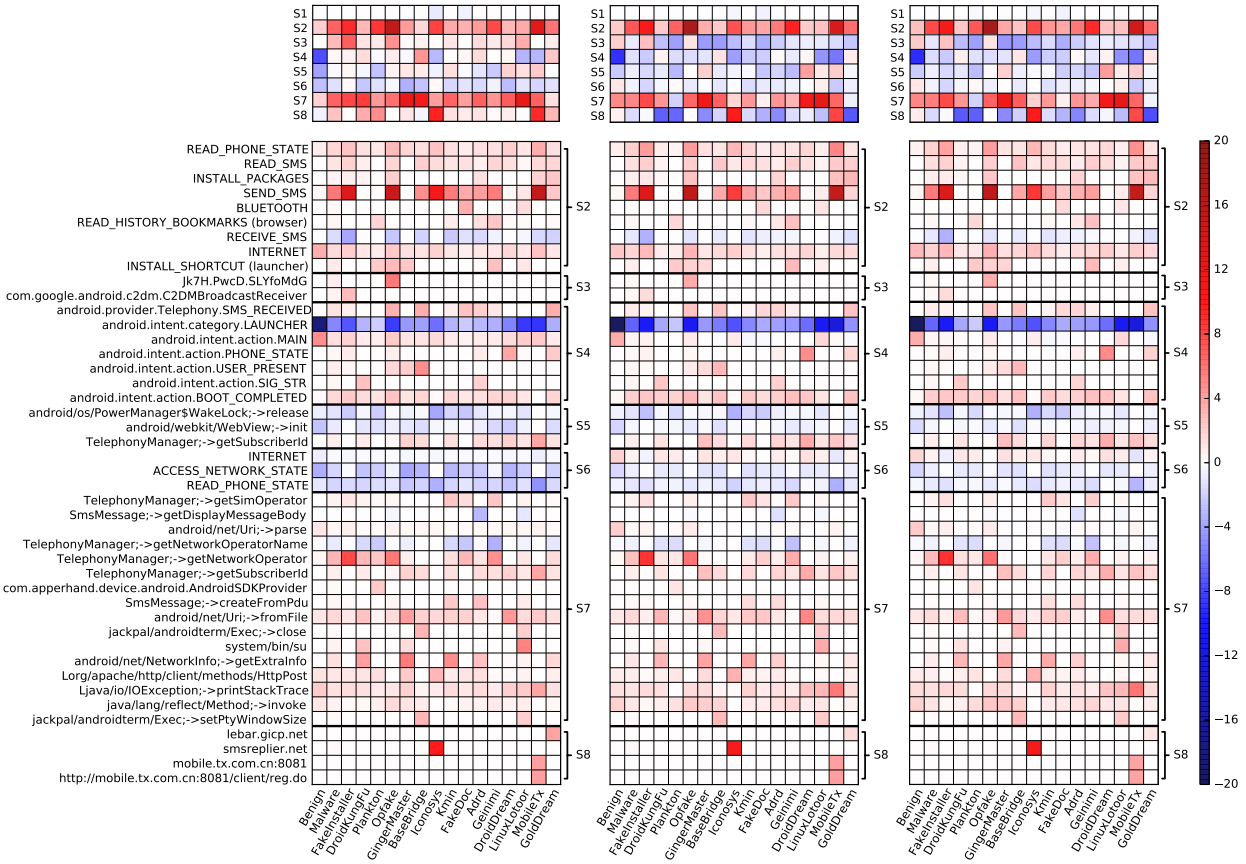


Figure 8.25: Average attribution assigned to each feature as computed by our global explanation method (Equation 6.3) with respect to benign apps, malware, and the apps from the top-15 malware families by number of available samples in the *Drebin* dataset (Table 8.3), for SVM (*left*), SVM-RBF (*middle*) and RF (*right*). The compact representation (*top*) reports the feature relevance averaged over the feature sets $S_1, \ldots, S_8$ (Table 8.2). The fine-grained representation (*bottom*) reports relevance values for the top-44 features with the highest average relevance score, aggregated for each group of samples (benign apps, malware, `FakeInstaller` apps, etc.). Positive (negative) relevance values denote malicious (benign) behavior.

fact, as discussed in Section 2.2.5, if the decision of a classifier relies on a limited set of features, it is intuitive that detection can be easily evaded by manipulating only few components, as also confirmed in previous works [21, 224]. Conversely, if a model distributes the relevance more evenly among features, evasion may be more difficult (*i.e.*, requires a higher number of manipulations, which may not be always feasible).

Another interesting observation relates to the *transferability* of evasion attacks across different models, especially our findings on the gradients alignment (see Chapter 5). It is clear that in the tested cases the explanations depend more on the available training data rather than on the specific learning algorithm: the three considered classifiers learn very similar patterns of features relevance, as highlighted by both the compact and the fine-grained representations in Figure 8.25. Being our explanation method based on the gradient of the decision function of each classifier, this simply means that models with similar global explanations, *i.e.*, aligned gradients, can be fooled by the same adversarial perturbation, coherently with the experimental results from Section 8.1. As a follow-up to these findings, in Section 8.3 we statistically demonstrate how gradient-based explanations are directly correlated to the robustness of machine learning models to evasion attacks.

8.3 The connection between Explanations and Adversarial Robustness

In this section, we practically evaluate whether the synthetic metrics introduced in Section 6.2 can be used effectively to analyse the security properties of machine learning models in an adversarial environment. Motivated by the insights on transferability obtained in Section 8.2 using our gradient-based explanation method on three different models for Android malware detection, and by the intuition that classifiers whose attributions are more evenly distributed should also be the more robust (as they rely on a broader set of features for the decision, see Section 2.2.5), we statistically correlate the *uniformity* of the attributions (Equation 6.7) with the *adversarial robustness* (Equation 6.9) to evasion attacks.

Experimental Setup

We expand the experimental framework introduced in Section 8.1.2 and Section 8.2 by comparing, in addition to the standard Drebin linear Support Vector Machine (SVM) and a SVM with RBF kernel (SVM-RBF), a logistic regression (logistic), a ridge regression (ridge), and the *secured* linear SVM we proposed in [21], as defined by Equation 2.15 (Sec-SVM).

Using a 5-fold cross-validation procedure, we optimize the hyperparameters of each classifier in this case to maximize the detection rate (*i.e.*, the fraction of detected malware) at 1% false positive rate (*i.e.*, the fraction of legitimate applications misclassified as malware). In particular, we optimize $C \in \{10^{-2}, 10^{-1}, \ldots, 10^{2}\}$ for both linear and non-linear SVMs and logistic, the kernel parameter $\gamma \in \{10^{-4}, 10^{-3}, \ldots, 10^{2}\}$ for the SVM-RBF, and the parameter $\alpha \in \{10^{-2}, 10^{-1}, \ldots, 10^{2}\}$ for ridge. For Sec-SVM, we optimize the parameters $-\mathbf{w}_{\text{lb}} = \mathbf{w}_{\text{ub}} \in \{0.1, 0.25, 0.5\}$ and $C \in \{10^{-2}, 10^{-1}, \ldots, 10^{2}\}$. When similar detection rates (±1%) are obtained for different configurations of the hyperparameters, we select the configuration corresponding to a more regularized classifier, as less complex models are expected to be more robust under attack, according to what we demonstrated in Section 8.1. The typical values of the aforementioned hyperparameters found after cross-validation are $C = 0.1$ for SVM, $\alpha = 10$ for ridge, $C = 1$ for logistic, $C = 1$ and $-\mathbf{w}_{\text{lb}} = \mathbf{w}_{\text{ub}} = 0.25$ for Sec-SVM, $C = 10$ and $\gamma = 0.01$ for SVM-RBF.

We compute the explanations for 1,000 malware samples randomly chosen from the *Drebin* test set. We compare the relevance vectors, w.r.t the malicious class, obtained using the Gradient*Input method Equation 2.20[3] , with the explanations based on the simple Gradient (Equation 2.17), and the ones computed using Integrated Gradients (Equation 2.22), with $\boldsymbol{x}_0 = \mathbf{0}$ as the baseline. We recall that a positive (negative) relevance value in our analysis denotes malicious (benign) behavior. Given the high sparsity ratio of the *Drebin* dataset, we use $\mathsf{D} = 1,000$ to compute the explanation evenness metrics, as detailed in Section 6.2.1.

3: For a fairer comparison of the results between the different explanation methods, in this analysis we use the standard (non-normalized) version of Gradient*Input.

Results

We start by expanding the detection performance analysis previously reported in Figure 8.24, with the addition of logistic, ridge, and Sec-SVM

classifiers. The new Receiver Operating Characteristic (ROC) curves, averaged over the 5 repetitions, are reported in Figure 8.26. We recall that the *detection rate* represents the probability that a malicious sample is correctly labeled by the classifier (true positive rate, see Section 2.1.2).

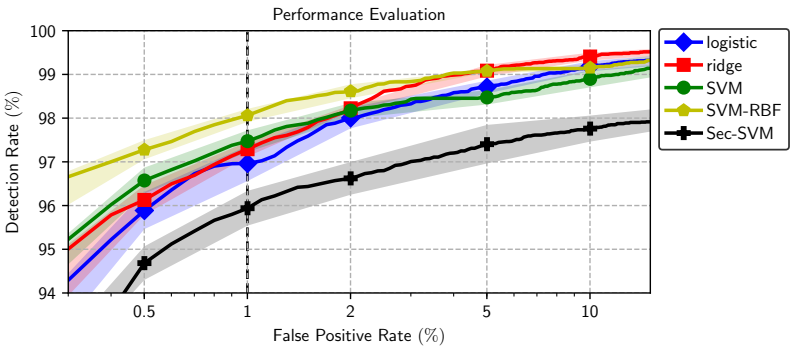


Figure 8.26: Average ROC curves for the given classifiers trained on the *Drebin* data. Sec-SVM is the *secured* linear SVM we proposed in [21]. The *detection rate* represents the probability that a malicious sample is correctly labeled. The false positive rate (1%), which is used for computing the security evaluation curves in Figure 8.27, is highlighted with a dashed black line.

To obtain a reference indication of the security of each classifier, to be later compared with the results of our correlation analysis, we perform a white-box evasion attack against each model. 1,000 malware samples are randomly chosen from the *Drebin* dataset to this end, and we simulate an adversary trying to make them misclassified as benign. The resulting security evaluation curves are shown in Figure 8.27, which reports the variation of the detection rate at 1% false positive rate as the number of modified features ε increases. We can notice that, while the Sec-SVM classifier provides a slightly worse detection rate compared to the other models (Figure 8.26), it is highly robust against evasion attacks. In fact, more than 25 features (on average) should be altered in each malicious sample to evade this robust classifier.[4] Conversely, all the other models are evaded after 5 to 10 feature additions. Notably, the non-linearity of the SVM-RBF helps this model maintain a higher robustness to adversarial examples compared to the other linear classifiers.

4: Our previous work [21] provides an in-depth experimental analysis of the Sec-SVM algorithm.

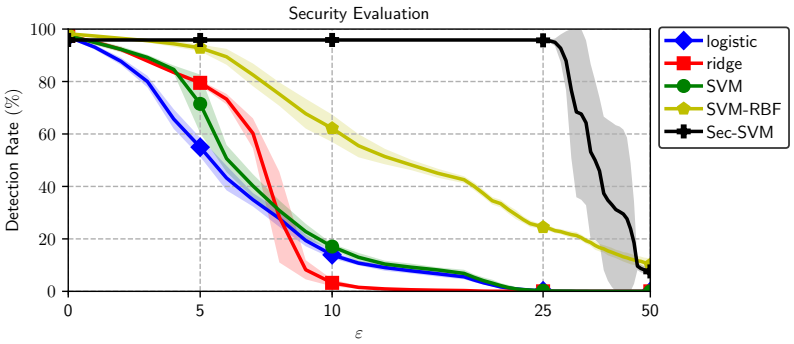


Figure 8.27: White-box evasion attacks on the *Drebin* data. Detection rate at 1% false positive rate against an increasing number of added features ε. We can see how the Sec-SVM, despite providing a slightly lower detection rate compared to the other tested classifiers (Figure 8.26), requires on average more than 25 different new feature additions to the original apps to be fooled by the attacker.

Is adversarial robustness correlated with explanation evenness?

We now investigate the connection between adversarial robustness and the evenness of gradient-based explanations. We start with two illustrative examples based on *local* explanations. Table 8.4 shows the top-10 influential features for two malware samples of the `FakeInstaller`[5] and `Plankton`[6] families, reported for the SVM-RBF and Sec-SVM algorithms, and obtained through the Gradient*Input technique. Both classifiers correctly label the samples as malware.

5: f8bcbd48f44ce973036fac0bce68a5d5

6: eb1f454ea622a8d2713918b590241a7e

SVM-RBF ($\mathscr{E}_1$ = 46.24%, $\mathscr{E}_2$ = 22.47%, $\varepsilon_{\min}$ = 6)

Set	Feature Name	r (%)
S2	SEND_SMS	10.35
S7	android/telephony/TelephonyManager ;->getNetworkOperator	10.05
S4	LAUNCHER	-8.89
S5	android/os/PowerManager$WakeLock ;->release	-8.01
S2	READ_PHONE_STATE	5.03
S2	RECEIVE_SMS	-5.00
S3	c2dm.C2DMBroadcastReceiver	4.56
S2	READ_SMS	3.52
S4	DATA_SMS_RECEIVED	3.50
S5	android/app/NotificationManager ;->notify	-3.49

Sec-SVM ($\mathscr{E}_1$ = 73.04%, $\mathscr{E}_2$ = 66.24%, $\varepsilon_{\min}$ = 31)

Set	Feature Name	r (%)
S2	READ_PHONE_STATE	3.51
S7	android/telephony/TelephonyManager ;->getNetworkOperator	3.51
S2	SEND_SMS	3.51
S3	c2dm.C2DMBroadcastReceiver	3.51
S2	INTERNET	3.44
S3	com.software.application.ShowLink	3.39
S3	com.software.application.Main	3.39
S3	com.software.application.Notificator	3.39
S3	com.software.application.Checker	3.39
S3	com.software.application.OffertActivity	3.39

SVM-RBF ($\mathscr{E}_1$ = 60.74%, $\mathscr{E}_2$ = 25.84%, $\varepsilon_{\min}$ = 31)

Set	Feature Name	r (%)
S4	LAUNCHER	-1.89
S7	android/net/Uri;->fromFile	1.34
S5	android/os/PowerManager$WakeLock ;->release	-1.25
S2	INSTALL_SHORTCUT	1.23
S7	android/telephony/SmsMessage ;->getDisplayMessageBody	-1.21
S7	android/telephony/SmsMessage ;->getTimestampMillis	-1.20
S2	SET_ORIENTATION	-1.20
S2	ACCESS_WIFI_STATE	1.15
S4	BOOT_COMPLETED	1.08
S5	android/media/MediaPlayer;->start	-1.06

Sec-SVM ($\mathscr{E}_1$ = 63.14%, $\mathscr{E}_2$ = 52.70%, $\varepsilon_{\min}$ = 39)

Set	Feature Name	r (%)
S2	ACCESS_NETWORK_STATE	0.93
S2	READ_PHONE_STATE	0.93
S6	READ_HISTORY_BOOKMARKS	0.93
S7	android/telephony/TelephonyManager ;->getNetworkOperatorName	-0.93
S6	ACCESS_NETWORK_STATE	-0.93
S7	android/telephony/SmsMessage;->getDisplayOriginatingAddress	0.93
S7	android/telephony/TelephonyManager ;->getNetworkOperator	0.93
S7	android/net/Uri;->getEncodedPath	-0.93
S2	SET_ORIENTATION	-0.93
S7	java/lang/reflect/Method;->invoke	0.93

Table 8.4: Top-10 influential features and corresponding Gradient*Input relevance (r %) for two malware apps of the *Drebin* dataset, one from the `FakeInstaller` family (top) and one from the `Plankton` family (bottom). Notice that the minimum number $\varepsilon_{\min}$ of features to add to evade the classifiers increases with the evenness metrics $\mathscr{E}_1$ and $\mathscr{E}_2$.

Looking at the relevant features of the first sample, the `FakeInstaller` malware, we discover how both the classifiers identify the cellular- and SMS-related features, *e.g.*, the `GetNetworkOperator()` method or the `SEND_SMS` permission, as highly relevant. This is coherent with the actual behavior of the malware, since its goal is to send SMS messages to premium-rate numbers. With respect to the relevance values, the first aspect to point out comes from their relative magnitude, expressed as a percentage in Table 8.4. In particular, we observe that the top-10 relevance values for SVM-RBF vary, regardless of their signs, from 3.49% to 10.35%, while for Sec-SVM the top values lie in the 3.39%–3.51% range. This suggests that SVM-RBF assigns high prominence to few features; conversely, Sec-SVM distributes the relevance values more evenly. This behavior is also represented by the synthetic evenness measures $\mathscr{E}_1$ and $\mathscr{E}_2$, reported in Table 8.4, both showing higher values for the Sec-SVM.

In Table 8.4 we also report the $\varepsilon_{\min}$ value, *i.e.*, the minimum number of features to add to the malware to evade the classifier. We can notice how the $\varepsilon_{\min}$ measure is strictly related to the evenness distribution, since higher values of $\mathscr{E}_1$ and $\mathscr{E}_2$ correspond to higher values of $\varepsilon_{\min}$. This is in accordance to the hypothesis that a higher effort from the attacker is required to evade a model which gradient-based attributions are more uniform. In the case of this first malware, it is possible to identify a clear difference between the behavior of SVM-RBF and Sec-SVM: the diversity of their evenness metrics, which causes the $\varepsilon_{\min}$ values to be quite different as well, indicating that the SVM-RBF is more susceptible to an evasion attack compared to the Sec-SVM.

Conversely, the attributions (regardless of the sign) and the evenness metrics computed on the second sample present similar values, inside the 1.89%–0.93% range. Such behavior is also reflected by the associated $\varepsilon_{\min}$ values, which are over 30 (feature additions) for both models. In this case, the relevance values are more evenly distributed, which indicates that the adversary needs to modify more components of the application in order to evade each classifiers.

We now correlate the explanation evenness metrics with the *adversarial robustness*, as defined by Equation 6.9. Figure 8.28 shows the measures computed on 100 randomly-selected samples from the test set of *Drebin*, reported for each explainability technique. From this broader view, we can observe how the evenness values calculated on top of the Gradient*Input and Integrated Gradients explanations present a significant connection to the adversarial robustness. Conversely, it seems that such relation does not subsist for the Gradient technique, and specifically against the linear classifiers (SVM, logistic, and ridge), whose occurrences in Figure 8.28 are perfectly vertical-aligned. A simple explanation of this behavior can be provided by recalling that the gradient of the decision function w.r.t the input sample is given, for these classifiers, by the vector of the parameters itself. As a result, the explanations computed by the Gradient method are constant across all the samples, and the values of the relative evenness metrics are constant as well.

In order to assess the statistical significance of these plots, in Table 8.5 we also report the associated correlation values, computed with three different metrics: Pearson (P), Spearman Rank (S), Kendall's Tau (K). As we obtain p-val $\ll 0.05$ in almost all cases[7] , these statistical tests confirm the validity our findings.

7: The only off-scale value is Pearson (P) on $\mathscr{E}_1$ for the Gradient method, probably caused by its sensitivity to outliers.

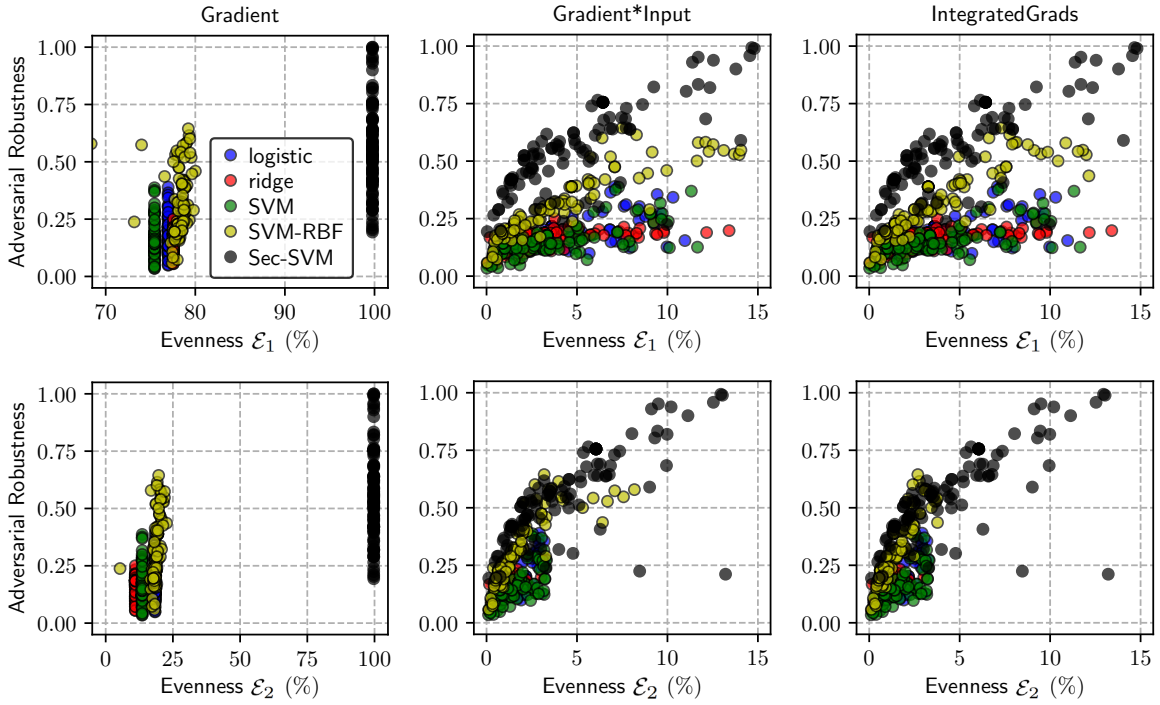


Figure 8.28: Evaluation of the adversarial robustness AR against the evenness $\mathscr{E}_1$ (top plots), $\mathscr{E}_2$ (bottom plots) metrics for the different gradient-based explanation techniques applied to 1,000 randomly-selected samples of the test set (only 100 samples are shown for compactness). The statistical significance of these plots is assessed in Table 8.5 by means of the relative correlation coefficients and p-values.

Is adversarial robustness correlated with detection rate?

Finally, we inquire whether the connection between the evenness metrics and the detection performance of a classifier can provide a global assessment of its robustness. Figure 8.29 reports the correlation between the explanation evenness and the mean detection rate under attack,

		Gradient		Gradient*Input		Int. Gradients	
		$\mathscr{E}_1$	$\mathscr{E}_2$	$\mathscr{E}_1$	$\mathscr{E}_2$	$\mathscr{E}_1$	$\mathscr{E}_2$
logistic	P			0.67, <1e-5	0.75, <1e-5	0.67, <1e-5	0.75, <1e-5
	S			0.67, <1e-5	0.72, <1e-5	0.67, <1e-5	0.72, <1e-5
	K			0.51, <1e-5	0.54, <1e-5	0.51, <1e-5	0.54, <1e-5
ridge	P			0.48, <1e-5	0.56, <1e-5	0.48, <1e-5	0.56, <1e-5
	S			0.58, <1e-5	0.67, <1e-5	0.58, <1e-5	0.67, <1e-5
	K			0.41, <1e-5	0.49, <1e-5	0.41, <1e-5	0.49, <1e-5
SVM	P			0.68, <1e-5	0.70, <1e-5	0.68, <1e-5	0.70, <1e-5
	S			0.66, <1e-5	0.73, <1e-5	0.66, <1e-5	0.73, <1e-5
	K			0.49, <1e-5	0.54, <1e-5	0.49, <1e-5	0.54, <1e-5
SVM-RBF	P	0.03, 0.769	0.46, <1e-5	0.82, <1e-5	0.82, <1e-5	0.89, <1e-5	0.91, <1e-5
	S	0.46, <1e-5	0.70, <1e-5	0.94, <1e-5	0.94, <1e-5	0.93, <1e-5	0.93, <1e-5
	K	0.34, <1e-5	0.51, <1e-5	0.81, <1e-5	0.80, <1e-5	0.78, <1e-5	0.77, <1e-5
Sec-SVM	P			0.73, <1e-5	0.76, <1e-5	0.73, <1e-5	0.76, <1e-5
	S			0.76, <1e-5	0.78, <1e-5	0.76, <1e-5	0.78, <1e-5
	K			0.62, <1e-5	0.67, <1e-5	0.62, <1e-5	0.67, <1e-5

Table 8.5: Correlation between the adversarial robustness AR (Equation 6.9) and the evenness metrics $\mathscr{E}_1$ and $\mathscr{E}_2$ (respectively, Equation 6.5 and Equation 6.6). Pearson (P), Spearman Rank (S), Kendall's Tau (K) coefficients along with corresponding p-values. The linear classifiers lack a correlation value since the evenness is constant (being the gradient constant as well), thus resulting in a non-defined correlation.

calculated for ε in the range $[1, 50]$. Similarly to the previous tests, the synthetic metrics computed on the explanations from Gradient*Input and Integrated Gradients methods present a significant connection to the detection rate in most cases, also witnessed by the p-values largely under 0.05, while the correlation of the Gradient technique is again lower, due to the explanations being constant for the linear models.

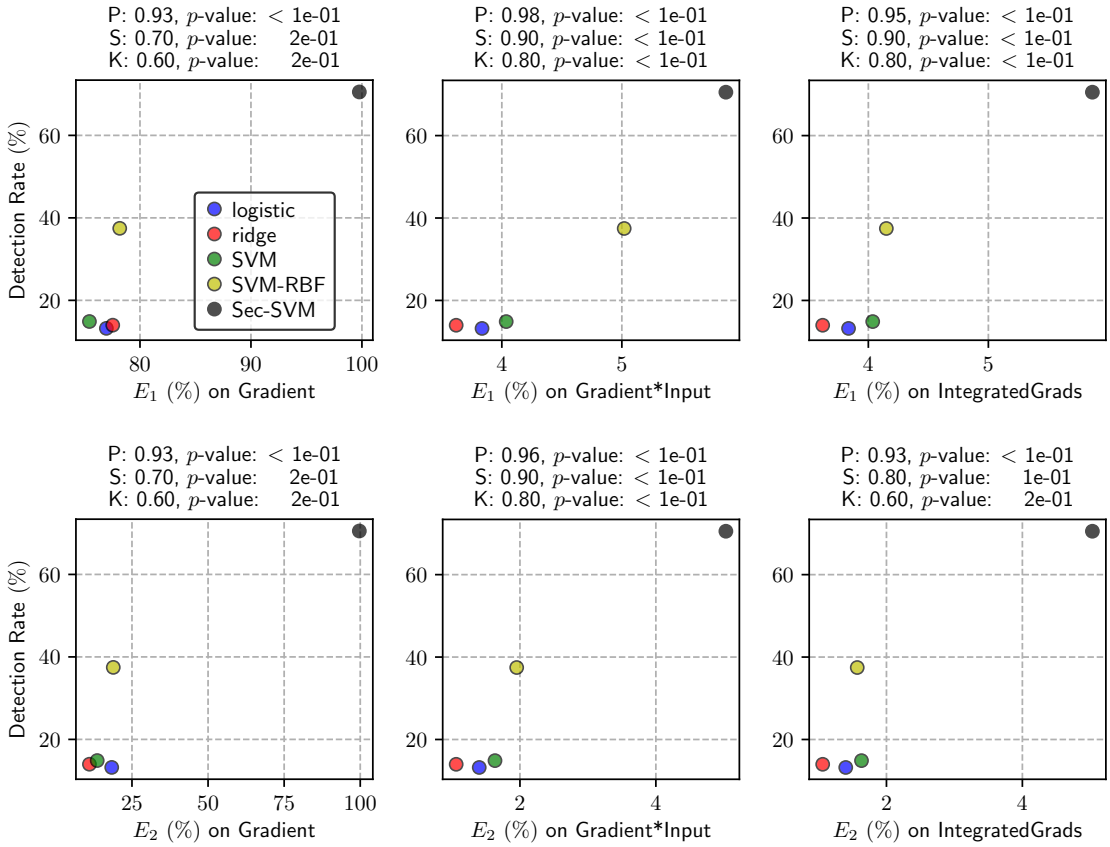


Figure 8.29: Evaluation of the evenness metrics E_1 (top plots) and E_2 (bottom plots) against the detection rate (at FPR 1%) for the different gradient-based explanation techniques applied on 1,000 randomly-selected samples from the test set. On top of each plot we report the correlation values, using Pearson (P), Spearman Rank (S), Kendall's Tau (K) coefficients, along with the corresponding p-values.

9 Conclusions and Limitations

While pattern recognition systems based on machine learning models are increasingly being applied to many practical applications, the research on defending them against adversarial attacks should be considered still at a very early stage. The number of scientific works published daily on this topic, reporting new discovered vulnerabilities and new attack strategies, is a worrying signal for the general public, which leads to a very low trust in these automated decision-makers. In addition, the relatively limited work that has been done until now on the techniques to explain their behavior has not helped to this end. Machine learning based systems are still in need of a single clear theoretical framework that defines their behavior when employed in adversarial environments, including proper metrics to quantitatively evaluate their performance, security, and trustworthiness. We believe this thesis provided several contributions toward this direction.

First, in Chapter 4 we proposed a *unified framework* for crafting evasion and poisoning attacks through a gradient-descent optimization algorithm. As by our knowledge, this was the first structured attempt to jointly tackle the two different types of adversarial threats with a single mathematical formulation. Notably, we provided the challenging derivation of poisoning attacks for both support vector machines and logistic regression. Our attack framework can be applied effectively to *comprehensive threat models* that describe a wide range of possible applications, by including the different goals an adversarial may have (integrity and availability), the different knowledge of the model which may be available (white-box, gray-box, or black-box), as well as the different capabilities the attacker may have to influence the targeted system.

In Chapter 5 we provided the first thorough evaluation of one of the arguably most insidious properties of adversarial attacks: *transferability*, the ability of an attack crafted against a machine learning model to be effective against a different one, potentially unknown. We provided a formal definition in the context of our unified attack framework, and then, in Section 8.1, we presented a comprehensive evaluation considering both evasion and poisoning availability attacks. Our experiments demonstrated that attack transferability strongly depends on the *complexity* of the target model, *i.e.*, on its inherent vulnerability. In particular, we observed that for evasion attacks, an adversary might prefer to decrease the complexity of the surrogate model by adjusting the hyperparameters of its learning algorithm, as this results in adversarial examples that transfer better to a wide range of models. For poisoning attacks, the best surrogates are generally models with similar levels of regularization as the target. The reason is that the poisoning objective function has a relatively low variance for most classifiers, and the gradient alignment between the surrogate and the target becomes a more critical factor. These conclusions have been supported by an analysis of *three newly proposed metrics*, (i) size of input gradients, (ii) gradient alignment, and (iii) variability of the loss landscape, which can be used to quantitatively

measure transferability and compare the robustness of different machine learning models. We believe this analysis may increase the awareness of the community on the issue of attack transferability, especially to design more robust algorithms in the future.

Afterward, in Chapter 6, we presented a study on how gradient-based explanation methods can be leveraged to compare different models with respect to their learned behavior and get insights into their security properties, like adversarial robustness or the resilience to transfer attacks. Also, concepts like the user's trust in the system's decision cannot be measured using standard performance metrics, as they are strictly related to the ability of a system to clearly explain its outputs to humans. Thus, comparing the behavior of different systems based on these user-dependent aspects is still an open question, also because of the strict connection between most interpretability techniques with the attributes of the machine learning model they try to explain, like the hyperparameters or the magnitude of the output scores.

To overcome this limitation, in Section 6.1 we proposed a novel technique that, by normalizing the attributions obtained using the traditional Gradient*Input method, allows to compare different classifiers independently from their specificities. Our local and global *relevance vectors* are highly-interpretable, as the attribution assigned to each feature is limited inside a definite interval. Notably, our method can be successfully applied even in the case of black-boxes or non-differentiable models, allowing to analyze any machine learning based system. Moreover, this approach can also help to understand the security-related properties of the algorithms, including the vulnerability to transfer attacks. From our experiments of Section 8.2, in fact, it became clear that many classifiers learn very similar patterns of feature relevance from the available training data. This behavior is fascinating as, being our method based on the gradient of the decision function w.r.t. the input sample, it merely means that models with similar global explanations, *i.e.*, aligned gradients, can be fooled by the same adversarial perturbations, in agreement to what we observed in Section 8.1 evaluating transferability.

In Section 6.2 we proposed to correlate few synthetic metrics that measure the *uniformity of the attributions* and the *adversarial robustness* to statistically demonstrate this evident connection between gradient-based explanations and the security properties of machine learning models. Inspired by one of the known vulnerabilities of linear classifiers, *i.e.*, that models that distribute the relevance over a broader set of features should also be more robust, we showed in Section 8.3 that more uniform explanation vectors correspond to less vulnerability to evasion attacks. We believe that this finding will greatly ease the design process of machine learning based systems, as gradient-based explanations may be leveraged to provide a useful indication of the adversarial robustness of a model, without the necessary need of performing a costly full security evaluation, especially when advanced learning algorithms like neural networks are involved.

Finally, in Chapter 7 we presented `secml`, an open-source Python library that aims to provide all the tools for developing and evaluating secure and explainable machine learning based systems, without the need of integrating multiple third-party libraries. `secml` not only implements most of the functions available in the most popular scientific libraries,

including *numpy*, *scipy*, and *PyTorch*, but also allows to conduct a full security evaluation of the implemented system by crafting evasion and poisoning attacks using the built-in algorithms. Last but not least, it provides many techniques to produce post-hoc local and global explanations, which may help the engineers increase the users' trust in their newly developed algorithms.

Future Work

Different avenues for future research works are still open after this thesis. Firstly, our analysis of the transferability issue has been carried out on few different application cases, but all of them designed to use binary classifiers. While our unified attack framework can be easily extended to the multiclass case, similarly to what we have already done for evasion attacks in [20], it would be interesting to prove the applicability of our proposed metrics in a multiclass classification setting. We may also consider in the future a range of gray-box models in which attackers have only partial knowledge of the machine learning based system. The different attack scenarios may in fact provide additional constraints that could impact the transferability of the attacks in interesting ways.

Secondly, different strategies to provide global explanations using our proposed technique could be explored. In fact, merely averaging the local relevance vectors can have the effect of softening the attribution of features that are highly relevant only for a few samples. In the future, we may test the effect of weighting the explanations before averaging, or more advanced solutions like the combination of related explanation vectors (*i.e.*, crafting global relevance prototypes).

We also plan to validate our analysis on the connection between gradient-based explanations and adversarial robustness on other malware detectors in addition to Drebin, as well as considering other applicative domains (*e.g.*, PDF malware detection or web application security), learning algorithms (*e.g.*, neural networks), and poisoning attacks. Moreover, it could be interesting to verify if our correlation metrics can be successfully applied when the attacker does not know the classifier parameters, or when the model is not differentiable, *e.g.*, by leveraging a surrogate classifier. Finally, another interesting research avenue may be to modify the objective functions used for learning the models by adding a penalty term inversely proportional to the explanation evenness. This should force the algorithms to learn models with more evenly distributed attributions and, consequently, increase the robustness to adversarial attacks.

Closing Remarks

We hope that the contributions of this thesis will capture the attention of the scientific community to improve the research in both the adversarial machine learning and explainable machine learning fields. There is indeed an increasing number of applications that deal with these problems, and thus the demand for more secure and interpretable implementations of pattern recognition and machine learning techniques is expected to soar in the following years.

Bibliography

[1] A. L. Samuel, "Some studies in machine learning using the game of checkers", *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[2] R. O. Duda, P. E. Hart, *et al.*, *Pattern classification and scene analysis*. Wiley New York, 1973, vol. 3.

[3] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, Oct. 2007.

[4] F. Doshi-Velez and B. Kim, "Towards a rigorous science of interpretable machine learning", *arXiv preprint arXiv:1702.08608*, 2017.

[5] Z. C. Lipton, "The mythos of model interpretability", in *ICML Workshop on Human Interpretability in Machine Learning*, 2016, pp. 96–100.

[6] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences", *Artificial Intelligence*, vol. 267, pp. 1–38, 2019.

[7] B. Goodman and S. Flaxman, "European Union regulations on algorithmic decision-making and a "right to explanation"", *ArXiv e-prints*, 2016.

[8] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning", *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[9] G. L. Wittel and S. F. Wu, "On attacking statistical spam filters", in *First Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2004.

[10] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, "Polymorphic blending attacks", in *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, Vancouver, B.C., Canada: USENIX Association, 2006, pp. 241–256.

[11] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks", in *International Conference on Learning Representations*, 2014.

[12] A. Ignatiev, N. Narodytska, and J. Marques-Silva, "On relating explanations and adversarial examples", in *Advances in Neural Information Processing Systems*, 2019, pp. 15 883–15 893.

[13] K. Xu, S. Liu, G. Zhang, M. Sun, P. Zhao, Q. Fan, C. Gan, and X. Lin, "Interpreting adversarial examples by activation promotion and suppression", *arXiv preprint arXiv:1904.02057*, 2019.

[14] G. Fidel, R. Bitton, and A. Shabtai, "When explainability meets adversarial learning: Detecting adversarial examples using shap signatures", in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–8.

[15] W. Woods, J. Chen, and C. Teuscher, "Adversarial explanations for understanding image classification decisions and improved neural network robustness", *Nature Machine Intelligence*, vol. 1, no. 11, pp. 508–516, 2019.

[16] A. Demontis, M. Melis, M. Pintor, M. Jagielski, B. Biggio, A. Oprea, C. Nita-Rotaru, and F. Roli, "Why do adversarial attacks transfer? explaining transferability of evasion and poisoning attacks", in *28th USENIX Security Symposium (USENIX Security 19)*, Santa Clara, CA: USENIX Association, Aug. 2019, pp. 321–338.

[17] M. Melis, D. Maiorca, B. Biggio, G. Giacinto, and F. Roli, "Explaining black-box android malware detection", in *2018 26th European Signal Processing Conference (EUSIPCO)*, IEEE, 2018, pp. 524–528.

[18] M. Melis, M. Scalas, A. Demontis, D. Maiorca, B. Biggio, G. Giacinto, and F. Roli, "Do gradient-based explanations tell anything about adversarial robustness to android malware?", *arXiv preprint arXiv:2005.01452*, 2020.

[19] M. Melis, A. Demontis, M. Pintor, A. Sotgiu, and B. Biggio, "Secml: A python library for secure and explainable machine learning", *arXiv preprint arXiv:1912.10013*, 2019.

[20] M. Melis, A. Demontis, B. Biggio, G. Brown, G. Fumera, and F. Roli, "Is deep learning safe for robot vision? adversarial examples against the icub humanoid", in *ICCV Workshop on Vision in Practice on Autonomous Robots (ViPAR)*, 2017.

[21] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection", *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2017.

[22] A. Sotgiu, A. Demontis, M. Melis, B. Biggio, G. Fumera, X. Feng, and F. Roli, "Deep neural rejection against adversarial examples", *EURASIP Journal on Information Security*, vol. 2020, pp. 1–10, 2020.

[23] F. Crecchi, M. Melis, A. Sotgiu, D. Bacciu, and B. Biggio, "Fader: Fast adversarial example rejection", *arXiv preprint arXiv:2010.09119*, 2020.

[24] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning", in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, 2011, pp. 689–696.

[25] N. Srivastava and R. R. Salakhutdinov, "Multimodal learning with deep boltzmann machines", in *Advances in neural information processing systems*, 2012, pp. 2222–2230.

[26] S.-C. Pei and C.-N. Lin, "Image normalization for pattern recognition", *Image and Vision computing*, vol. 13, no. 10, pp. 711–723, 1995.

[27] R. A. Fisher, "The use of multiple measurements in taxonomic problems", *Annals of eugenics*, vol. 7, no. 2, pp. 179–188, 1936.

[28] H. Hotelling, "Analysis of a complex of statistical variables into principal components.", *Journal of educational psychology*, vol. 24, no. 6, p. 417, 1933.

[29] P. Comon, "Independent component analysis, a new concept?", *Signal processing*, vol. 36, no. 3, pp. 287–314, 1994.

[30] P. Cunningham, "Dimension reduction", in *Machine learning techniques for multimedia*, Springer, 2008, pp. 91–112.

[31] L. I. Kuncheva, *Combining Pattern Classifiers: Methods and Algorithms*. Hoboken, N. J.: Wiley, 2004.

[32] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms", *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

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

[34] C. Cortes and V. Vapnik, "Support-vector networks", *Machine Learning*, vol. 20, pp. 273–297, 3 1995.

[35] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition", *Data mining and knowledge discovery*, vol. 2, pp. 121–167, 2 Jun. 1998.

[36] R. Fletcher, *Practical methods of optimization*. John Wiley & Sons, 1987.

[37] B. Haasdonk, "Feature space interpretation of svms with indefinite kernels", *IEEE Transactions on pattern analysis and machine intelligence*, vol. 27, no. 4, pp. 482–492, 2005.

[38] B. Schölkopf, "The kernel trick for distances", in *Advances in neural information processing systems*, 2001, pp. 301–307.

[39] B. Biggio, G. Fumera, and F. Roli, "Learning sparse kernel machines with biometric similarity functions for identity recognition", in *IEEE Fifth International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, 2012, pp. 325–330.

[40] D. M. Tax and R. P. Duin, "Using two-class classifiers for multiclass classification", in *Object recognition supported by user interaction for service robots*, IEEE, vol. 2, 2002, pp. 124–127.

[41] K.-B. Duan and S. S. Keerthi, "Which is the best multiclass svm method? an empirical study", in *International workshop on multiple classifier systems*, Springer, 2005, pp. 278–285.

[42] F. Rosenblatt, "Principles of neurodynamics: Perceptions and the theory of brain mechanisms", 1962.

[43] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity", *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[44] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[45] G. Thimm and E. Fiesler, "Neural network initialization", in *International Workshop on Artificial Neural Networks*, Springer, 1995, pp. 535–542.

[46] R. Hecht-Nielsen, "Theory of the backpropagation neural network", in *Neural networks for perception*, Elsevier, 1992, pp. 65–93.

[47] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines", in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, 2010, pp. 807–814.

[48] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey", *Heliyon*, vol. 4, no. 11, e00938, 2018.

[49] R. Pascanu, C. Gulcehre, K. Cho, and Y. Bengio, "How to construct deep recurrent neural networks", *arXiv preprint arXiv:1312.6026*, 2013.

[50] H. Sak, A. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling", in *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[51] X. Li and X. Wu, "Constructing long short-term memory based deep recurrent neural networks for large vocabulary speech recognition", in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2015, pp. 4520–4524.

[52] Y. LeCun *et al.*, "Generalization and network design strategies", *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[53] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex", *The Journal of physiology*, vol. 148, no. 3, p. 574, 1959.

[54] S. Saha. (2018). A comprehensive guide to convolutional neural networks, [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 12/15/2018).

[55] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks", in *European conference on computer vision*, Springer, 2014, pp. 818–833.

[56] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization", *arXiv preprint arXiv:1506.06579*, 2015.

[57] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets", in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[58] B. Chesney and D. Citron, "Deep fakes: A looming challenge for privacy, democracy, and national security", *California Law Review*, vol. 107, p. 1753, 2019.

[59] X. Han, N. Kheir, and D. Balzarotti, "Phisheye: Live monitoring of sandboxed phishing kits", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16, New York, NY, USA: ACM, 2016, pp. 1402–1413.

[60] I. Corona, B. Biggio, M. Contini, L. Piras, R. Corda, M. Mereu, G. Mureddu, D. Ariu, and F. Roli, "Deltaphish: Detecting phishing webpages in compromised websites", in *22nd European Symposium on Research in Computer Security (ESORICS)*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds., ser. LNCS, vol. 10492, Cham: Springer International Publishing, 2017, pp. 370–388.

[61] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?", in *Proceedings of the 2006 ACM Symposium on Information, Computer and Comm. Security*, 2006, pp. 16–25.

[62] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, "Misleading worm signature generators using deliberate noise injection", in *IEEE Symposium on Security and Privacy (S&P'06)*, May 2006, 15 pp.–31.

[63] J. Graham-Cumming, "How to beat an adaptive spam filter", in *MIT Spam Conference*, Cambridge, MA, USA, 2004.

[64] D. Lowd and C. Meek, "Good word attacks on statistical spam filters", in *Second Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2005.

[65] M. Barreno, B. Nelson, A. Joseph, and J. Tygar, "The security of machine learning", *Machine Learning*, vol. 81, pp. 121–148, 2 2010.

[66] 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)*, Chicago, IL, USA, 2011, pp. 43–57.

[67] B. Biggio, I. Corona, B. Nelson, B. Rubinstein, D. Maiorca, G. Fumera, G. Giacinto, and F. Roli, "Security evaluation of support vector machines in adversarial environments", in *Support Vector Machines Applications*, Y. Ma and G. Guo, Eds., Cham: Springer International Publishing, 2014, pp. 105–153.

[68] L. Muñoz-González, B. Biggio, A. Demontis, A. Paudice, V. Wongrassamee, E. C. Lupu, and F. Roli, "Towards poisoning of deep learning algorithms with back-gradient optimization", in *10th ACM Workshop on Artificial Intelligence and Security*, B. M. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, Eds., ser. AISec '17, New York, NY, USA: ACM, 2017, pp. 27–38.

[69] B. Biggio, G. Fumera, and F. Roli, "Security evaluation of pattern classifiers under attack", *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 4, pp. 984–996, Apr. 2014.

[70] B. Biggio, G. Fumera, and F. Roli, "Pattern recognition systems under attack: Design issues and research challenges", *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 28, no. 7, 2014.

[71] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings", in *IEEE European symposium on security and privacy (EuroS&P)*, IEEE, 2016, pp. 372–387.

[72] N. Papernot, P. McDaniel, and I. Goodfellow, "Transferability in machine learning: From phenomena to black-box attacks using adversarial samples", *arXiv preprint arXiv:1605.07277*, 2016.

[73] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning", in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17, New York, NY, USA: ACM, 2017, pp. 506–519.

[74] 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 *Machine Learning and Knowledge Discovery in Databases (ECML PKDD), Part III*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds., ser. LNCS, vol. 8190, Springer Berlin Heidelberg, 2013, pp. 387–402.

[75] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis", in *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX: USENIX Association, 2016, pp. 601–618.

[76] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers", in *Proceedings of the 23rd Annual Network & Distributed System Security Symposium (NDSS)*, The Internet Society, 2016.

[77] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models", in *10th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '17, New York, NY, USA: ACM, 2017, pp. 15–26.

[78] H. Dang, Y. Huang, and E. Chang, "Evading classifiers by morphing in the dark", in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Comm. Security*, ser. CCS '17, ACM, 2017, pp. 119–133.

[79] D. Lowd and C. Meek, "Adversarial learning", in *Proceeding of the 11th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining (KDD)*, Chicago, IL, USA: ACM Press, 2005, pp. 641–647.

[80] B. Nelson, B. I. Rubinstein, L. Huang, A. D. Joseph, S. J. Lee, S. Rao, and J. D. Tygar, "Query strategies for evading convex-inducing classifiers", *The J. of Mach. Learn. Res.*, vol. 13, pp. 1293–1332, May 2012.

[81] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples", in *International Conference on Learning Representations*, 2015.

[82] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks", in *IEEE Symposium on Security and Privacy*, IEEE Computer Society, 2017, pp. 39–57.

[83] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines", in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, J. Langford and J. Pineau, Eds., Omnipress, 2012, pp. 1807–1814.

[84] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?", in *Proceedings of the 32nd International Conference on International Conference on Machine Learning (ICML)*, F. Bach and D. Blei, Eds., vol. 37, 2015, pp. 1689–1698.

[85] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners", in *Proceedings of the 29th AAAI Conference Artificial Intelligence (AAAI '15)*, 2015.

[86] M. Jagielski, A. Oprea, B. Biggio, C. Liu, C. Nita-Rotaru, and B. Li, "Manipulating machine learning: Poisoning attacks and countermeasures for regression learning", in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, IEEE, 2018, pp. 19–35.

[87] N. Šrndic and P. Laskov, "Practical evasion of a learning-based classifier: A case study", in *IEEE Symposium on Security and Privacy (S&P)*, ser. SP '14, Washington, DC, USA: IEEE CS, 2014, pp. 197–211.

[88] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. D. McDaniel, "Adversarial examples for malware detection", in *European Symposium on Research in Computer Security (ESORICS)*, ser. LNCS, vol. 10493, Springer, 2017, pp. 62–79.

[89] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain", *arXiv preprint arXiv:1708.06733*, 2017.

[90] X. Chen, C. Liu, B. Li, K. Lu, and D. Song, "Targeted backdoor attacks on deep learning systems using data poisoning", *arXiv preprint arXiv:1712.05526*, 2017.

[91] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures", in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15, New York, NY, USA: ACM, 2015, pp. 1322–1333.

[92] A. Adler, "Vulnerabilities in biometric encryption systems", in *5th International Conference on Audio- and Video-Based Biometric Person Authentication (AVBPA)*, T. Kanade, A. K. Jain, and N. K. Ratha, Eds., ser. LNCS, vol. 3546, Hilton Rye Town, NY, USA: Springer, Jul. 2005, pp. 1100–1109.

[93] J. Galbally, C. McCool, J. Fierrez, S. Marcel, and J. Ortega-Garcia, "On the vulnerability of face verification systems to hill-climbing attacks", *Pattern Recognition*, vol. 43, no. 3, pp. 1027–1038, 2010.

[94] M. Martinez-Diaz, J. Fierrez, J. Galbally, and J. Ortega-Garcia, "An evaluation of indirect attacks and countermeasures in fingerprint verification systems", *Pattern Recognition Letters*, vol. 32, no. 12, pp. 1643–1651, 2011.

[95] I. Shumailov, Y. Zhao, D. Bates, N. Papernot, R. Mullins, and R. Anderson, "Sponge examples: Energy-latency attacks on neural networks", *arXiv preprint arXiv:2006.03463*, 2020.

[96] A. Kantchelian, J. D. Tygar, and A. D. Joseph, "Evasion and hardening of tree ensemble classifiers", in *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, vol. 48, JMLR.org, 2016, pp. 2387–2396.

[97] P. Russu, A. Demontis, B. Biggio, G. Fumera, and F. Roli, "Secure kernel machines against evasion attacks", in *9th ACM Workshop on Artificial Intelligence and Security*, New York, NY, USA: ACM, 2016, pp. 59–69.

[98] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2574–2582.

[99] N. Carlini and D. A. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods", in *10th ACM Workshop on Artificial Intelligence and Security*, B. M. Thuraisingham, B. Biggio, D. M. Freeman, B. Miller, and A. Sinha, Eds., ser. AISec '17, New York, NY, USA: ACM, 2017, pp. 3–14.

[100] A. Athalye, N. Carlini, and D. Wagner, "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples", in *Proceedings of the International Conference on Machine Learning (ICML)*, 2018, pp. 274–283.

[101] Y. Dong, F. Liao, T. Pang, H. Su, J. Zhu, X. Hu, and J. Li, "Boosting adversarial attacks with momentum", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 9185–9193.

[102] A. Kolcz and C. H. Teo, "Feature weighting for improved classifier robustness", in *Sixth Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2009.

[103] A. D. Joseph, P. Laskov, F. Roli, J. D. Tygar, and B. Nelson, "Machine Learning Methods for Computer Security (Dagstuhl Perspectives Workshop 12371)", *Dagstuhl Manifestos*, vol. 3, no. 1, pp. 1–30, 2013.

[104] B. Biggio, G. Fumera, and F. Roli, "Adversarial pattern classification using multiple classifiers and randomisation", in *12th Joint IAPR International Workshop on Structural and Syntactic Pattern Recognition (SSPR 2008)*, ser. Lecture Notes in Computer Science, vol. 5342, Orlando, Florida, USA: Springer-Verlag, Apr. 2008, pp. 500–509.

[105] D. Meng and H. Chen, "MagNet: A two-pronged defense against adversarial examples", in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[106] S. Rota Bulò, B. Biggio, I. Pillai, M. Pelillo, and F. Roli, "Randomized prediction games for adversarial machine learning", *IEEE Trans. on Neural Net's and Learn. Systems*, vol. 28, no. 11, pp. 2466–2478, 2017.

[107] B. I. P. Rubinstein, P. L. Bartlett, L. Huang, and N. Taft, "Learning in a large function space: Privacy-preserving mechanisms for svm learning", *J. of Privacy and Conf.*, vol. 4, no. 1, pp. 65–100, 2012.

[108] A. Adler and S. A. C. Schuckers, "Security and liveness, overview", in *Encyclopedia of Biometrics*, S. Z. Li and A. K. Jain, Eds., Springer US, 2009, pp. 1146–1152.

[109] J. Lu, T. Issaranon, and D. Forsyth, "Safetynet: Detecting and rejecting adversarial examples robustly", in *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[110] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, "Adversarial classification", in *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, Seattle, 2004, pp. 99–108.

[111] A. Globerson and S. T. Roweis, "Nightmare at test time: Robust learning by feature deletion", in *Proceedings of the 23rd International Conference on Machine Learning*, W. W. Cohen and A. Moore, Eds., vol. 148, ACM, 2006, pp. 353–360.

[112] C. H. Teo, A. Globerson, S. Roweis, and A. Smola, "Convex learning with invariances", in *Advances in Neural Information Processing Systems 20*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds., Cambridge, MA: MIT Press, 2008, pp. 1489–1496.

[113] O. Dekel, O. Shamir, and L. Xiao, "Learning to classify with missing and corrupted features", *Machine Learning*, vol. 81, pp. 149–178, 2 2010.

[114] M. Brückner, C. Kanzow, and T. Scheffer, "Static prediction games for adversarial learning problems", *The Journal of Machine Learning Research*, vol. 13, pp. 2617–2654, Sep. 2012.

[115] W. Liu and S. Chawla, "Mining adversarial patterns via regularized loss minimization.", *Machine Learning*, vol. 81, no. 1, pp. 69–83, 2010.

[116] M. Großhans, C. Sawade, M. Brückner, and T. Scheffer, "Bayesian games for adversarial regression problems", in *30th International Conference on Machine Learning (ICML)*, vol. 28, 2013, pp. 55–63.

[117] M. Wooldridge, "Does game theory work?", *Intelligent Systems, IEEE*, vol. 27, no. 6, pp. 76–80, 2012.

[118] G. Cybenko and C. E. Landwehr, "Security analytics and measurements", *IEEE Security & Privacy*, vol. 10, no. 3, pp. 5–8, 2012.

[119] H. Xu, C. Caramanis, and S. Mannor, "Robustness and regularization of support vector machines", *The Journal of Machine Learning Research*, vol. 10, pp. 1485–1510, Jul. 2009.

[120] Z. Qi, Y. Tian, and Y. Shi, "Robust twin support vector machine for pattern classification", *Pattern Recognition*, vol. 46, no. 1, pp. 305–316, 2013.

[121] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks", in *International Conference on Learning Representations*, 2018.

[122] E. Wong and Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope", in *International Conference on Machine Learning*, PMLR, 2018, pp. 5286–5295.

[123] A. Demontis, P. Russu, B. Biggio, G. Fumera, and F. Roli, "On security and sparsity of linear classifiers for adversarial settings", in *Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, A. Robles-Kelly, M. Loog, B. Biggio, F. Escolano, and R. Wilson, Eds., ser. LNCS, vol. 10029, Cham: Springer International Publishing, 2016, pp. 322–332.

[124] B. Biggio, G. Fumera, and F. Roli, "Multiple classifier systems for robust classifier design in adversarial environments", *International Journal of Mach. Learn. and Cybernetics*, vol. 1, no. 1, pp. 27–41, 2010.

[125] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[126] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models", in *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, 2017, pp. 625–642.

[127] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, "Bagging classifiers for fighting poisoning attacks in adversarial classification tasks", in *10th International Workshop on Multiple Classifier Systems (MCS)*, C. Sansone, J. Kittler, and F. Roli, Eds., ser. Lecture Notes in Computer Science, vol. 6713, Springer-Verlag, Jun. 2011, pp. 350–359.

[128] A. Bendale and T. E. Boult, "Towards open set deep networks", in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1563–1572.

[129] J. G. Moreno-Torres, T. Raeder, R. Alaiz-Rodriguez, N. V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification", *Pattern recognition*, vol. 45, no. 1, pp. 521–530, 2012.

[130] I. Pillai, G. Fumera, and F. Roli, "Multi-label classification with a reject option", *Pattern Recognition*, vol. 46, no. 8, pp. 2256–2266, 2013.

[131] D. Maiorca, A. Demontis, B. Biggio, F. Roli, and G. Giacinto, "Adversarial detection of flash malware: Limitations and open issues", *Computers & Security*, p. 101 901, 2020.

[132] N. Šrndić and P. Laskov, "Detection of malicious pdf files based on hierarchical document structure", in *Proc. of the 20th Annual Network & Distributed Sys. Sec. Symposium (NDSS)*, The Internet Society, 2013.

[133] P. Wild, P. Radu, L. Chen, and J. Ferryman, "Robust multimodal face and fingerprint fusion in the presence of spoofing attacks", *Pattern Recognition*, vol. 50, pp. 17–25, 2016.

[134] B. Biggio, G. Fumera, G. L. Marcialis, and F. Roli, "Statistical meta-analysis of presentation attacks for secure multibiometric systems", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 3, pp. 561–575, Mar. 2017.

[135] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. P. Rubinstein, U. Saini, C. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter", in *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, San Francisco, California: USENIX Association, 2008, pp. 1–9.

[136] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S.-h. Lau, S. Rao, N. Taft, and J. D. Tygar, "Antidote: Understanding and defending against poisoning of anomaly detectors", in *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference*, ser. IMC '09, New York, NY, USA: ACM, 2009, pp. 1–14.

[137] B. Nelson, B. Biggio, and P. Laskov, "Understanding the risk factors of learning in adversarial environments", in *4th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '11, Chicago, IL, USA, 2011, pp. 87–92.

[138] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, "Casting out demons: Sanitizing training data for anomaly sensors", in *IEEE Symposium on Security and Privacy*, Los Alamitos, CA, USA: IEEE Computer Society, 2008, pp. 81–95.

[139] C. Liu, B. Li, Y. Vorobeychik, and A. Oprea, "Robust linear regression against training data poisoning", in *10th ACM Workshop on Artificial Intelligence and Security*, ser. AISec '17, ACM, 2017, pp. 91–102.

[140] J. Steinhardt, P. W. Koh, and P. Liang, "Certified defenses for data poisoning attacks", in *NIPS*, 2017.

[141] G. Xu, Z. Cao, B.-G. Hu, and J. C. Principe, "Robust support vector machines based on the rescaled hinge loss function", *Pattern Recognition*, vol. 63, pp. 139–148, 2017.

[142] A. Christmann and I. Steinwart, "On robust properties of convex risk minimization methods for pattern recognition", *The Journal of Machine Learning Research*, vol. 5, pp. 1007–1034, 2004.

[143] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise", in *Journal of Machine Learning Research - Proc. 3rd Asian Conf. Machine Learning*, vol. 20, Nov. 2011, pp. 97–112.

[144] J. A. Suykens and J. Vandewalle, "Least squares support vector machine classifiers", *Neural processing letters*, vol. 9, no. 3, pp. 293–300, 1999.

[145] F. Tramèr, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "The space of transferable adversarial examples", *arXiv preprint arXiv:1704.03453*, 2017.

[146] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks", in *Proceedings of 5th International Conference on Learning Representations*, 2017.

[147] L. Wu, Z. Zhu, C. Tai, and W. E, "Enhancing the transferability of adversarial examples with noise reduced gradient", *arXiv preprint arXiv:1802.09707*, 2018.

[148] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[149] O. Suciu, R. Marginean, Y. Kaya, H. Daume III, and T. Dumitras, "When does machine learning fail? generalized transferability for evasion and poisoning attacks", in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1299–1316.

[150] C. Yang, Q. Wu, H. Li, and Y. Chen, "Generative poisoning attack method against neural networks", *arXiv preprint arXiv:1703.01340*, 2017.

[151] Y. Liu, S. Ma, Y. Aafer, W.-C. Lee, J. Zhai, W. Wang, and X. Zhang, "Trojaning attack on neural networks", in *25th Network and Distributed System Security Symposium*, 2017.

[152] T. Gu, B. Dolan-Gavitt, and S. Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain", *arXiv preprint arXiv:1708.06733*, 2017.

[153] M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha, "Systematic poisoning attacks on and defenses for machine learning in healthcare", *IEEE journal of biomedical and health informatics*, vol. 19, no. 6, pp. 1893–1905, 2014.

[154] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, and D. Pedreschi, "A survey of methods for explaining black box models", *ACM computing surveys (CSUR)*, vol. 51, no. 5, pp. 1–42, 2018.

[155] M. T. Ribeiro, S. Singh, and C. Guestrin, ""why should i trust you?": Explaining the predictions of any classifier", in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, ser. KDD '16, New York, NY, USA: ACM, 2016, pp. 1135–1144.

[156] Y. Lou, R. Caruana, and J. Gehrke, "Intelligible models for classification and regression", in *Proceedings of the 18th ACM SIGKDD International Conf. on Knowledge discovery and data mining*, 2012, pp. 150–158.

[157] J. Huysmans, K. Dejaeger, C. Mues, J. Vanthienen, and B. Baesens, "An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models", *Decision Support Systems*, vol. 51, no. 1, pp. 141–154, 2011.

[158] M. Bohanec and I. Bratko, "Trading accuracy for simplicity in decision trees", *Machine Learning*, vol. 15, no. 3, pp. 223–250, 1994.

[159] J. R. Quinlan, "Simplifying decision trees", *International Journal of Human-Computer Studies*, vol. 51, no. 2, pp. 497–510, 1999.

[160] J. R. Quinlan, "Generating production rules from decision trees", in *Proceedings of the 10th International Joint Conference on artificial Intelligence*, Citeseer, vol. 87, 1987, pp. 304–307.

[161] R. Agrawal, R. Srikant, *et al.*, "Fast algorithms for mining association rules", in *Proceedings of the 20th Int'l Conf. on Very Large Data Bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[162] T. J. Hastie and R. J. Tibshirani, *Generalized additive models*. CRC press, 1990, vol. 43.

[163] M. Craven and J. W. Shavlik, "Extracting tree-structured representations of trained networks", in *Advances in neural information processing systems*, 1996, pp. 24–30.

[164] R. Krishnan, G. Sivakumar, and P. Bhattacharya, "Extracting decision trees from trained neural networks", *Pattern recognition*, vol. 32, no. 12, 1999.

[165] U. Johansson and L. Niklasson, "Evolving decision trees using oracle guides", in *2009 IEEE Symposium on Computational Intelligence and Data Mining*, IEEE, 2009, pp. 238–244.

[166] R. Andrews, J. Diederich, and A. B. Tickle, "Survey and critique of techniques for extracting rules from trained artificial neural networks", *Knowledge-based systems*, vol. 8, no. 6, pp. 373–389, 1995.

[167] F. Wang and C. Rudin, "Falling rule lists", in *Artificial Intelligence and Statistics*, 2015, pp. 1013–1022.

[168] H. Lakkaraju, S. H. Bach, and J. Leskovec, "Interpretable decision sets: A joint framework for description and prediction", in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1675–1684.

[169] H. Lakkaraju, E. Kamar, R. Caruana, and J. Leskovec, "Interpretable & explorable approximations of black box models", *arXiv preprint arXiv:1707.01154*, 2017.

[170] T. Wang, C. Rudin, F. Doshi-Velez, Y. Liu, E. Klampfl, and P. MacNeille, "A bayesian framework for learning rule sets for interpretable classification", *The Journal of Machine Learning Research*, vol. 18, no. 1, pp. 2357–2393, 2017.

[171] G. Fung, S. Sandilya, and R. B. Rao, "Rule extraction from linear support vector machines", in *Proceedings of the eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDDM)*, 2005, pp. 32–40.

[172] H. Núñez, C. Angulo, and A. Català, "Rule extraction from support vector machines.", in *Esann*, 2002, pp. 107–112.

[173] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations.", in *AAAI*, vol. 18, 2018, pp. 1527–1535.

[174] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps", *arXiv preprint arXiv:1312.6034*, 2013.

[175] D. Baehrens, T. Schroeter, S. Harmeling, M. Kawanabe, K. Hansen, and K.-R. Müller, "How to explain individual classification decisions", *The J. Mach. Learn. Res.*, vol. 11, pp. 1803–1831, 2010.

[176] A. Shrikumar, P. Greenside, A. Shcherbina, and A. Kundaje, "Not just a black box: Learning important features through propagating activation differences", *arXiv preprint arXiv:1605.01713*, 2016.

[177] G. Montavon, S. Lapuschkin, A. Binder, W. Samek, and K.-R. Müller, "Explaining nonlinear classification decisions with deep taylor decomposition", *Pattern Recognition*, vol. 65, pp. 211–222, 2017.

[178] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks", in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, JMLR. org, 2017, pp. 3319–3328.

[179] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation", *PloS one*, vol. 10, no. 7, 2015.

[180] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization", in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.

[181] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks", in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, IEEE, 2018, pp. 839–847.

[182] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences", in *International Conference on Machine Learning*, 2017, pp. 3145–3153.

[183] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization", in *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2921–2929.

[184] W. Guo, D. Mu, J. Xu, P. Su, G. Wang, and X. Xing, "Lemna: Explaining deep learning based security applications", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 364–379.

[185] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions", in *Advances in neural information processing systems*, 2017, pp. 4765–4774.

[186] L. S. Shapley, "A value for n-person games", *Contributions to the Theory of Games*, vol. 2, no. 28, pp. 307–317, 1953.

[187] L. Kaufmann and P. Rousseeuw, "Clustering by means of medoids", in *Proc. Statistical Data Analysis Based on the L1 Norm Conference*, 1987, pp. 405–416.

[188] B. Kim, R. Khanna, and O. O. Koyejo, "Examples are not enough, learn to criticize! criticism for interpretability", in *Advances in neural information processing systems*, 2016, pp. 2280–2288.

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

[190] F. Poursabzi-Sangdeh, D. G. Goldstein, J. M. Hofman, J. W. Vaughan, and H. Wallach, "Manipulating and measuring model interpretability", *arXiv preprint arXiv:1802.07810*, 2018.

[191] R. Tomsett, D. Braines, D. Harborne, A. Preece, and S. Chakraborty, "Interpretable to whom? a role-based model for analyzing interpretable machine learning systems", *arXiv preprint arXiv:1806.07552*, 2018.

[192] P. Schmidt and F. Biessmann, "Quantifying interpretability and trust in machine learning systems", *arXiv preprint arXiv:1901.08558*, 2019.

[193] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, "Evaluating explanation methods for deep learning in security", *arXiv preprint arXiv:1906.02108*, 2019.

[194] I. Lage, E. Chen, J. He, M. Narayanan, B. Kim, S. Gershman, and F. Doshi-Velez, "An evaluation of the human-interpretability of explanation", *arXiv preprint arXiv:1902.00006*, 2019.

[195] I. Rosenberg, S. Meir, J. Berrebi, I. Gordon, G. Sicard, and E. O. David, "Generating end-to-end adversarial examples for malware classifiers using explainability", in *2020 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2020, pp. 1–10.

[196] A. Ilyas, S. Santurkar, L. Engstrom, B. Tran, and A. Madry, "Adversarial examples are not bugs, they are features", *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[197] A.-K. Dombrowski, M. Alber, C. J. Anders, M. Ackermann, K.-R. Müller, and P. Kessel, "Explanations can be manipulated and geometry is to blame", *arXiv preprint arXiv:1906.07983*, 2019.

[198] A. Ilyas, L. Engstrom, A. Athalye, and J. Lin, "Black-box adversarial attacks with limited queries and information", in *International Conference on Machine Learning*, 2018, pp. 2137–2146.

[199] F. Zhang, P. Chan, B. Biggio, D. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks", *IEEE Transactions on Cybernetics*, vol. 46, no. 3, pp. 766–777, 2016.

[200] T. E. Oliphant, "Python for scientific computing", *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[201] K. J. Millman and M. Aivazis, "Python for scientists and engineers", *Computing in Science & Engineering*, vol. 13, no. 2, pp. 9–12, 2011.

[202] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, *et al.*, "Array programming with numpy", *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[203] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *The J. of Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[204] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library", in *Advances in neural information processing systems*, 2019, pp. 8026–8037.

[205] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems", *arXiv preprint arXiv:1603.04467*, 2016.

[206] W. McKinney *et al.*, "Pandas: A foundational python library for data analysis and statistics", *Python for High Performance and Scientific Computing*, vol. 14, no. 9, 2011.

[207] J. D. Hunter, "Matplotlib: A 2d graphics environment", *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[208] M. Waskom, O. Botvinnik, P. Hobson, J. Warmenhoven, J. Cole, Y. Halchenko, J. Vanderplas, S. Hoyer, S. Villalba, E. Quintero, *et al.* (2014). Seaborn: Statistical data visualization, [Online]. Available: https://seaborn.pydata.org (visited on 05/15/2017).

[209] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, *et al.*, "Technical report on the cleverhans v2. 1.0 adversarial examples library", *arXiv preprint arXiv:1610.00768*, 2016.

[210] J. Rauber, W. Brendel, and M. Bethge, "Foolbox: A python toolbox to benchmark the robustness of machine learning models", *arXiv preprint arXiv:1707.04131*, 2017.

[211] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, *et al.*, "Adversarial robustness toolbox v1. 0.0", *arXiv preprint arXiv:1807.01069*, 2018.

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

[213] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: Efficient and Comprehensive Mobile App Classification through Static and Dynamic Analysis", in *2015 IEEE 39th Annual Computer Software and Applications Conference*, vol. 2, Jul. 2015, pp. 422–433.

[214] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "Stormdroid: A streaminglized machine learning-based system for detecting android malware", in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 377–388.

[215] M. Scalas, D. Maiorca, F. Mercaldo, C. A. Visaggio, F. Martinelli, and G. Giacinto, "On the effectiveness of system api-related information for android ransomware detection", *Computers & Security*, vol. 86, pp. 168–182, 2019.

[216] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: Effective android malware detection and categorization via app-level profiling", *IEEE Transactions on Information Forensics and Security*, vol. 14, no. 6, pp. 1455–1470, 2018.

[217] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments", University of Massachusetts, Amherst, Tech. Rep. 07-49, Oct. 2007.

[218] G. B. H. E. Learned-Miller, "Labeled faces in the wild: Updates and new reporting procedures", University of Massachusetts, Amherst, Tech. Rep. UM-CS-2014-003, May 2014.

[219] C.-J. Simon-Gabriel, Y. Ollivier, L. Bottou, B. Schölkopf, and D. Lopez-Paz, "Adversarial vulnerability of neural networks increases with input dimension", 2018.

[220] A. S. Ross and F. Doshi-Velez, "Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients", *arXiv preprint arXiv:1711.09404*, 2017.

[221] D. Varga, A. Csiszárik, and Z. Zombori, "Gradient Regularization Improves Accuracy of Discriminative Models", *ArXiv e-prints ArXiv:1712.09936*, 2017.

[222] C. Lyu, K. Huang, and H.-N. Liang, "A unified gradient regularization family for adversarial examples", in *2015 IEEE International Conference on Data Mining (ICDM)*, Los Alamitos, CA, USA: IEEE Computer Society, 2015, pp. 301–309.

[223] B. Wang and N. Z. Gong, "Stealing hyperparameters in machine learning", in *2018 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2018, pp. 36–52.

[224] A. Calleja, A. Martin, H. D. Menendez, J. Tapiador, and D. Clark, "Picking on the family: Disrupting android malware triage by forcing misclassification", *Expert Systems with Applications*, vol. 95, pp. 113–126, 2018.

[225] L. Breiman, "Some infinity theory for predictor ensembles", Technical Report 579, Statistics Dept. UCB, Tech. Rep., 2000.