Università degli Studi di Cagliari

DOTTORATO DI RICERCA

INGEGNERIA ELETTRONICA ED INFORMATICA

XXIX Ciclo

SECURE MACHINE LEARNING AGAINST EVASION AND POISONING ATTACKS

Settore scientifico disciplinare di afferenza

ING-INF/05: Sistemi di elaborazione delle informazioni

Presentata da:	**Paolo Russu**
Coordinatore Dottorato	**Prof. Ing. Fabio Roli**
Tutor	**Prof. Ing. Fabio Roli**
Co-Tutor	**Dr. Ing. Battista Biggio**

Esame finale anno accademico 2015 – 2016

Tesi discussa nella sessione d'esame aprile 2017

Dedicated to my family

Abstract

In the last decades, machine learning has been widely used in security applications like spam filtering, intrusion detection in computer networks and biometric identity recognition. The adoption of such techniques has been mainly due to their high generalization capability, which allows one to identify also new kinds of attacks.

However, in these applications, machine learning has to deal with intelligent and adaptive adversaries that aim to subvert its proper functioning to achieve their malicious scope. Since machine learning has not been designed to take into account the presence of attackers, it may exhibit novel, specific vulnerabilities that can be exploited in the wild. Accordingly, identifying potential vulnerabilities and proposing new design schemes for pattern recognition and machine learning techniques in adversarial environments are not only two open problems, but also two among the major goals of adversarial classification. This situation has led to an arms race between attackers and developers. In order to limit the effects of the attackers, the designers should follow a proactive approach, *i.e.*, they should figure out how the adversary can interact with the system, in terms of points and methods of attack, and to develop appropriate countermeasures. This should enforce the attackers to spend a greater effort (in terms of time, skills, and resources) to find and exploit less intuitive vulnerabilities. In literature there are several attempts to create secure systems, based on models that allow one to characterize the adversary's behaviour with respect to a particular classifier-application scenario. However, the frameworks used to represent the attacker are customized for a specific setting, so it is difficult to readily apply them to different applications. Moreover, the adoption of the proposed robust solutions in practice is hampered by different factors, as the difficulty of meeting specific theoretical requirements, the complexity of implementation, and scalability issues, in terms of computational time and space required during training.

The main goal of this work regarded the development of methods to design pattern recognition algorithms that are secure from the ground up, which can effectively cope with malicious agents.

To start with, we show the usefulness of a recently proposed general threat model, which allows one to analyse the security of several application domains, applying it to a specific scenario. We show how to use this framework to analyse the security of biometric recognition systems from a novel perspective, by enabling the categorization of known and

novel vulnerabilities, along with the corresponding attacks, countermeasures and defence mechanisms.

Using the previous threat model as starting point to analyse systems in adversarial settings, we develop novel solutions to improve the security of several types of classifier. With respect to state-of-the-art methods, our non-trivial goal is to propose secure learning algorithms that are not computationally more demanding than their non-secure counterparts. We show that an adequate choice of the classifier's parameters, related to the specific hypothesized attack scenario, enables us to improve significantly system security.

Contents

List of Figures

List of Tables

Chapter 1

Machine Learning and Pattern Recognition

During their evolution, animals and, in particular, human beings have evolved a capacity, more or less pronounced, to recognize in a fast and efficient way the reality in which they are placed. This ability has been indispensable to ensure their own survival, from the research for food (e.g., the separation between edible and not plants) to the social interaction (recognition of a friend from an enemy or stranger).

The categorization problem is called *pattern recognition*, that is the act of taking in raw data and taking an action based on the category of the pattern [2]. It resolves the problem of classification, so the assignment of a new object to one or a set of classes which are known beforehand. Furthermore, as soon as the sample is classified, an action may be taken: eat or not the plant, avoid or fight or meet up with the stranger.

As an instance, if you were given a set of dogs and cats, you are perfectly and immediately able to separate the objects in these two classes, based on the dimensions, the length and the color of the fur, the muzzle shape and so on. The ease with which we perform this task, or we recognize a face, understand spoken words, read handwritten characters, belies the amazingly complex processes that underlie these acts of pattern recognition, the sophisticated neural and cognitive systems that we have evolved for such tasks.

It is natural that we should seek to design and build machines that can recognize patterns. From automated speech recognition, fingerprint identification, optical character recognition, DNA sequence identification and much more, it is clear that reliable, accurate pattern recognition by machine would be immensely useful.

Although the problem of distinguishing dogs and cats does not look very complicated, automating this process turns out to be fairly complicated. After all, what should be the basis for the categorization of an animal as 'cat' or as a 'dog'? Usually this decision is taken by combining all the stimulations that we can perceive with our five senses. Also, if some parts of an item are missing, our brain is able to override this lack, connecting the object to something that we have already seen and allowing us to recognize it. Even

Figure 1.1: Design stages of a typical pattern recognition system.

the top computer scientists will not be able to translate this process into an algorithm (as accurate as you), since the recognition process is carried out subconsciously. In other words, it is very difficult to explain in detail how the recognition process works.

Pattern recognition and machine learning address the problem of automatic pattern classification, including the above mentioned problems as well as many others. They offer a variety of algorithms which can be used in many classification tasks. They are typically referred to as classification algorithms, or classifiers for short. A classifier is usually trained on a set of collected samples whose class labels are known (referred to as training set), and infers some properties from the data. When a sample has to be classified, the classifier exploits the acquired knowledge to assign it to one among the given classes. The main advantage of such algorithms is their generalisation capability, *i.e.*, their ability to correctly classify unseen samples (that is, samples which were not present in the training set). The generalization capability depends on several factors, like the characteristics of the data used to train the classifier, the number of patterns, the type and number of features used, just to cite a few. Indeed, there is no best classifier, but it is possible to exploit different techniques to select the classification algorithm which performs better on the task at hand.

In the next section we will see the typical design scheme for pattern recognition systems, the steps to follow to design a good classifier. Then, we introduce the classical parameters used to measure the performance of a classifier, to decide if it works well or not for a specific setting. Finally, we give an overview on the type of classifiers that we have used for the work of this thesis.

1.1 Learning from examples

In many classification problems explicit rules do not exist, but examples can be obtained easily. It is difficult to design a classifier, *i.e.*, a function which outputs a class label for each input object, from known rules. Therefore, in pattern recognition, one tries to infer a classifier from a (limited) set of training examples. The use of examples thus elevates the need to explicitly state the rules for the classification by the user. The goal is to obtain models and learning rules to learn from the examples and predict the labels of future objects.

Broadly speaking, the design stages of a pattern recognition system can be summarised as depicted in Fig.1.1. Firstly, a set of training data has to be collected and potentially pre-processed. For instance, this could be the case of a face recognition system, where the acquired face images need to be pre-processed before feature extraction. After that, features can be extracted from each sample and used to train a classifier, chosen among different classification algorithms. Once a classifier is trained, a set of unseen samples may be used to estimate its performance. In the following, we give you a more detailed explanation of the setting steps, with a more formal view of these problems, and highlighting their main issues.

1.1.1 Collecting Data

The notion of "object" is taken very broadly, it can range from apples and pears to handwritten digits, from speech signals to the Internet traffic. A certain quantity of data should be collected to allow an appropriate representation of the classes distribution. The number of the necessary samples depends on the complexity of the classifier that we will use and on the specific application. For example, in a biometric recognition system, the training usually requires a number of traits (faces, fingerprints) to categorize each user. In security setting, instead, it needs at least a number of samples in the order of thousands, so as to represent all the potential intra and extra classes variations.

It is quite simple to collect a sufficient number of objects for the application at hand. In biometric systems, the users give spontaneously their traits. In medical settings, evidences of healthy and ill patients are easily collected. In computer security applications, the designer can monitor the environment for a period of time, or develop a honeypot. [1], in order to have examples of legitimate and illegal traffic.

Usually, in statistical pattern recognition the unknown distribution $p(\mathbf{X}, \mathbf{Y})$ is supposed to be independent and identically-distributed (i.i.d.). Identically Distributed means that it is stationary, the samples drawn for both training and test sets come from the same $p(\mathbf{X}, \mathbf{Y})$. Independent means that the samples are all independent events; in other words, they are not connected to each other in any way.

1.1.2 Data Pre-processing

Once collected, input samples should be processed to remove or mitigate potential natural or sensor noise. Only rarely, indeed, they can be readily used in the successive phases of the design process. For this reason, they have to be 'cleaned' and 'uniformed', in order to reduce the quantity of useless information that should be analysed, to simplify subsequent

[1] A honeypot is a computer security mechanism set to detect attempts at unauthorized use of information systems. Generally, a honeypot consists of data (for example, in a network site) that appears to be a legitimate part of the site but is actually isolated and monitored, and that seems to contain information or a resource of value to attackers, which are then blocked.

operations, and to uniform them. These operations are influenced by the knowledge of the usage scenario, only in this case we are able to understand how to handle the data so that we can improve their representativeness. For instance, in a face recognition system, we apply filters to the images to reduce differences of lighting, to mitigate the noise introduced by the camera. We identify the faces in the photos and separate them from the background (*i.e.*, the segmentation process). We manipulate the extracted faces in order to have, more or less, the same dimensions for every acquisition and to compare them in a fair way. Typical procedures are:

- *Data cleaning.* Reduction of the noise of the samples, due to the acquisition device, the overlapping from other signal sources, etc.
- *Data integration.* Combination in a single coherent data of the information originated from different sources. It allows one to build samples that are better representative for a particular classification setting.
- *Data reduction.* Extraction of the main relevant features from the samples. For example, the separation of an object from the background in a photo. This operation permits to reduce substantially the amount of data per sample that will be analyse and/or stored, leaving the distinctness of the object, more or less, unchanged.
- *Data transformation.* The samples have to be manipulated in order to be comparable with each other; for instance, they are rescaled, normalized, translated or re-oriented.

1.1.3 Feature Representation

The decision about which characteristics to use to represent the objects is, probably, the most important operation in the design of a pattern recognition system. This choice strongly influences the ability of the classifier to recognize the nature of an unseen sample in the operating phase. Starting from the pre-processed samples, we select particular characteristics and we measure their values or occurrences. What are the main characteristics that one should select to represent data in a compact manner? To be able to make a distinction between objects and classes of objects, the measurements should contain enough information to distinguish the objects, namely, those values are significantly different (similar) for objects belonging to different classes (the same class). In other words, they should have enough discriminative power such that a classifier will show sensible generalization. When only noise measurements are available, we cannot expect to infer a good classification. The feature extraction allows us to further reduce the amount of data to represent each sample. We assume that objects are described by vectors containing a set of d real valued measurements, thus an object i is represented by the feature vector

$$\mathbf{x}_i = (x_{i,1}, ..., x_{i,d}),\ x_{i,j} \in \mathbb{R} \tag{1.1}$$

Each object is thus represented as a point in a feature space $\mathbb{R}^d$. Furthermore, we assume that all components in the vector are known and that there are no missing values. In practice it might happen that some measurements are not performed, due to costs in time, money and effort, or because it is expected that they do not provide useful information (this can happen, for instance, in medical applications). The missing values introduce extra complications and we will not consider them. We assume all objects are characterized with the same set of measurements.

1.1.4 Model Selection and Sparsity

Now, a classification algorithm may be trained on the training data to find a classification function, namely, a function which classifies any sample in the feature space in one of the classes, according to its feature values. Notice that this corresponds to partition the feature space in different decision regions, each one corresponding to a different class. The boundary which separates such regions in the feature space is named *decision boundary*. Different classification functions may be obtained by training different algorithms on the given data. Since pattern recognition could be used in several settings, one should choose the most appropriate classification model for the task at hand, based on its performance estimation. This evaluation is mainly focused on assessing the generalisation capability of classifiers using specific performance measures, although sometimes it may involve other criteria as well, like computational efficiency, depending on the specific application domain. To estimate the classifier's performance, usually the entire initial dataset is divided in two separate parts, the training set, used to train the system, and the test set, used to evaluate the operative characteristics.

For example, a test set can be built by simply extracting (and removing) a number of samples from the training set, before training the classifier. It will then provide a predicted class label for each sample. The overall performance can be evaluated, as an instance, by comparing the estimated class labels with the true ones. If the classification performance meets the application requirements, the system can be finally released.

The two-class problem is considered as the basic classification problem; also a general multiclass setting can be decomposed in several two-class classification problems, where the classes are taken one by one and compared with the remaining ones.

In a two-class classification problem, the two classes ω_1 and ω_2 will be labelled by -1 and $+1$ respectively. The training set is a set of objects for which a label y_i is attached to each object $\mathbf{x}_i$:

$$D = \{(\mathbf{x}_i, y_i) \mid i = 1, .., N\} \tag{1.2}$$

Now, the classification problem can be stated as the problem of finding a classification function which optimally maps any input pattern $\mathbf{x}$ to the corresponding class

$$f^* = \mathbb{R}^d \rightarrow \{-1, +1\} \tag{1.3}$$

In statistical pattern recognition, both the input pattern and its class label are assumed to be random variables, and the aim is to find a function f (i.e., an estimate of f^*) which minimises the classification errors. Notice that other measures of performance can be adopted, depending on the given application. The function f assigns one of the two labels (two-class problem) to each point in the feature space thresholding a continuous discriminant function

$$g : \mathbb{R}^d \mapsto \mathbb{R} \tag{1.4}$$

Usually, we assume that $f(\mathbf{x}) = -1$ if $g(\mathbf{x}) < 0$, and $+1$ otherwise. In the case of $g(\mathbf{x}) = 0$, the label is assigned indifferently to one class. So, we can state that $f(\mathbf{x}_i) = sign(g(\mathbf{x}_i))$

In Fig.1.2 we can see three examples of how the classifier works in the feature space. The plots are sorted in ascending order of complexity of the function $g(\mathbf{x})$. The samples are taken from two noisy banana-shape distributions, one in front of the other. [2]

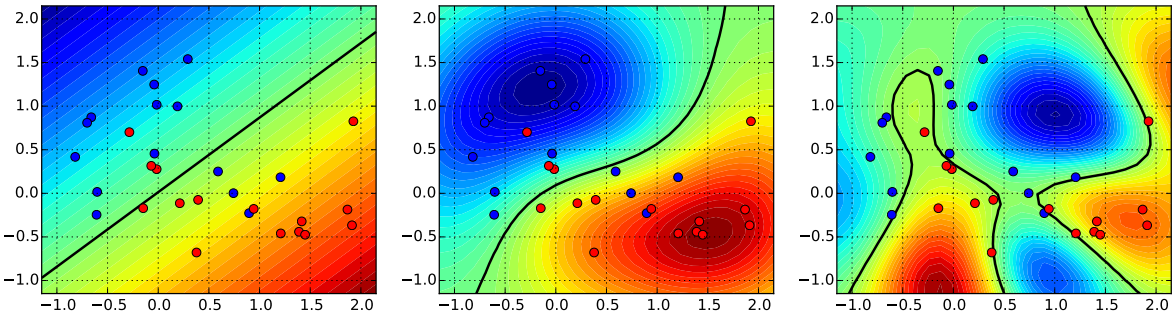


Figure 1.2: Three examples of classifier in the feature space, sorted by complexity. The samples are taken from two noisy banana-shape distributions. In each, the function $g(\mathbf{x})$ is plotted with a color map with high values (red-orange-yellow) for the class labelled with $+1$ (red points), and low values (green, cyan-blue) for the class labelled with -1 (blue points). The decision boundaries are shown in black.

Under the previous considerations, we can re-write the classifier's function as $f(g(\mathbf{x})) = f(\mathbf{x}; \mathbf{w})$ to state explicitly the dependence on the parameters or weights $\mathbf{w}$ of the discriminant function.

Data-Dependent Error. To find the optimal parameters $\mathbf{w}^*$ for the function f on a given training set D, an error function $\mathcal{E}(f, \mathbf{w}, D)$ has to be defined. The independence assumption on training data allows you to decompose it as the sum of the classification errors on the single samples:

$$\min_{\mathbf{w}} \mathcal{E}(f, \mathbf{w}, D) = \min_{\mathbf{w}} \frac{1}{N} \sum_i e(f(\mathbf{x}_i; \mathbf{w}), y_i) \tag{1.5}$$

[2] `http://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html`

Different definitions for the error function are possible, according to the type of $f(\mathbf{x}_i; \mathbf{w})$. The first possibility, the simplest one, is the 0-1 loss. This counts the number of wrongly classified objects:

$$e(f(\mathbf{x}_i; \mathbf{w}), y_i) = \begin{cases} 0, & \text{if } f(\mathbf{x}_i; \mathbf{w}) = y_i \\ 1, & \text{otherwise} \end{cases} \tag{1.6}$$

Other more complex error functions are:

- square loss: $(1 - y_i f(\mathbf{x}_i))^2$,
- hinge loss: $\max(0, 1 - y_i f(\mathbf{x}_i))$,
- logistic loss: $\frac{1}{\ln 2} \ln(1 + \exp -y_i f(\mathbf{x}_i))$.

By minimizing the error $\mathcal{E}$ on the training set, one hopes to find a good set of weights $\mathbf{w}$ such that a good classification is obtained.

Note that the choice of a more accurate classification model is not always an advantage. In many pattern recognition problems, the amount of data we can obtain easily is often quite limited. They are a sampling of the true probability distributions of the categories. So, using an overly complex model may allow you to reach a perfect classification of the training samples, but probably it does not give a good classification of novel patterns. This situation is known as *overfitting* [1,2]. In general, it occurs when the model is excessively complex, such as having too many parameters relative to the number of observations.
In Fig.1.2, we can see that by increasing the complexity of the classifier we can reach higher classification performance: in the third plot the samples are perfectly discriminated in the respectively classes. However, this solution doesn't approximate properly the real distributions of the samples, so it potentially will misclassify the unseen samples in the operation phase, resulting useless.

Regularizer. To avoid overfitting, we can rewrite the optimization problem as the trade-off between two error terms

$$\min_{\mathbf{w}} \mathcal{E}_{tot}(f, \mathbf{w}, D) = \min_{\mathbf{w}} \mathcal{E}_D(f, \mathbf{w}, D) + \lambda \mathcal{E}_{\mathbf{w}}(\mathbf{w}) \tag{1.7}$$

The first term is the data-dependent term, while the second one, $\mathcal{E}_{\mathbf{w}}(\mathbf{w})$, is the *regularizer*. It controls the overfitting of the vector $\mathbf{w}$ on the training data, trying to limit the amplitude of the function coefficients. Usually, this term corresponds to a 2-norm, or Euclidean norm, of the vector $\mathbf{w}$ and it is called quadratic regularizer or ***ridge***. The ridge regressor tends to penalize the square of the coefficients of $\mathbf{w}$, so it tries to push down the elements with big weights with respect to the small ones. In this way every coefficient, more or less, is important in the classification task.

In general, we can use any type of norm as regularizer. To take into account applications where there are constraints on energy consumption and computational complexity,

it has been introduced the ***LASSO*** regularizer [6], that computes the 1-norm of the $\mathbf{w}$ in the regularization term. It has the opposite behaviour respect to the ridge, it drives the smaller elements to zero and leading to a ***sparse*** solution. Having a sparse $\mathbf{w}$ vector reduces significantly the computational requests and makes more intuitive the classification task: the presence of a few non-zero weights highlights the most discriminant features for the objects recognition.

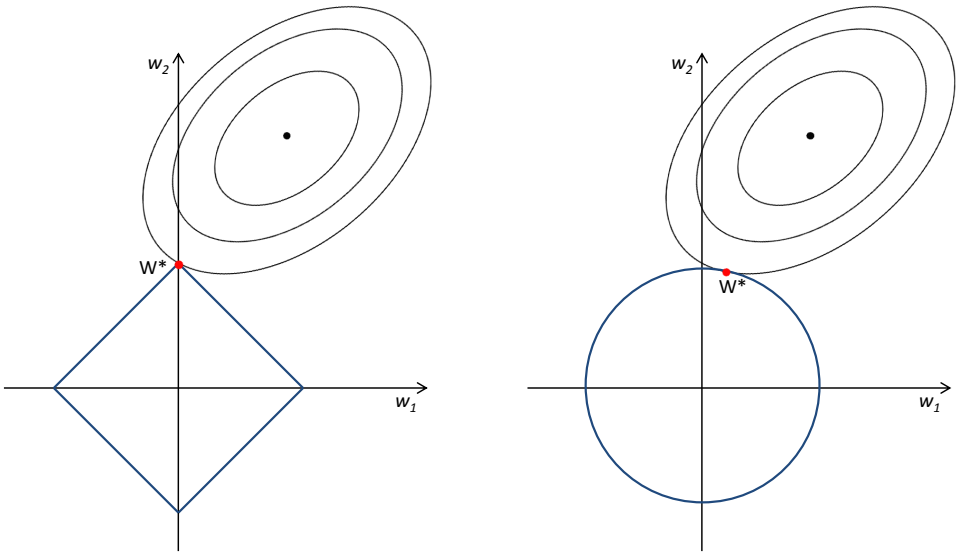
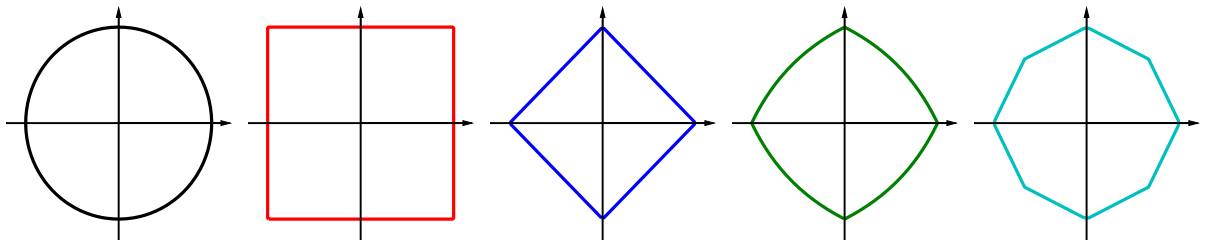


Figure 1.3: Plot of the countour of the error function(black) along with the constraint region for the LASSO (*left*) and ridge (*right*) regularizers, on which the optimum value for the parameter vector $\mathbf{w}$ is denoted by $\mathbf{w}^*$ [1].

Although the LASSO has shown success in many situations, it has some limitations:

- In the $p > n$ case, with p the number of features and n the number of samples, the LASSO selects at most n variables before it saturates, because of the nature of the convex optimization problem.
- If there is a group of variables among which the pairwise correlations are very high, then the LASSO tends to select only one variable from the group and does not care which one is selected; for instance, if in Fig.1.3-*left* the ellipse is narrow and parallel or orthogonal to an edge of the ℓ_1 ball, the solution could be indifferently one of the two vertices.
- For usual $n > p$ situations, if there are high correlations between predictors, it has been empirically observed that the prediction performance of the LASSO is dominated by ridge [6].



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

Figure 1.4: Unit balls for different norms. We will discuss the octagonal norm in Chap.4.

To overcome these limitations, Zou et al. [78] have proposed the ***elastic-net*** regularizer, that is a tradeoff between ridge and LASSO:

$$\|\mathbf{w}\|_{el\_net} = (1-\lambda)\|\mathbf{w}\|_1 + \tfrac{\lambda}{2}\|\mathbf{w}\|_2^2. \tag{1.8}$$

The addition of the quadratic penalty term to the LASSO makes the regularizer strictly convex, and therefore it has a unique minimum.

Another important regularizer, that we will use in Chap.4, is the ℓ_∞ norm. We remind the reader that $\|\mathbf{w}\|_\infty = \max_{j=1,\ldots,\mathsf{d}} |w_j|$; so, in the optimization problem, this regularizer tries to force the coefficients to have, more or less, the same value.

In Fig.1.4 we can see the different unit balls corresponding to different regularizers.

One of the most important areas of research in statistical pattern classification is determining how to adjust the complexity of the model, not so simple that it cannot explain the differences between the categories, yet not so complex as to give poor classification during the operating phase. We might be satisfied with slightly poorer performance on the training samples if it means that our classifier will have better performance on novel patterns.

So far, several classification algorithms have been proposed in the literature, like support vector machines (SVMs), neural networks, bayesian classifiers, decision trees and k-nearest neighbour classifiers, just to cite a few. Broadly speaking, classification algorithms can be divided into discriminative and generative models. The former are based on the direct estimation of the decision boundaries between classes (like SVMs and neural networks), while the latter (e.g., bayesian classifiers) are based on the estimation of the class-conditional pattern distributions, which consequently provide a partition in the feature space corresponding to the different decision regions. Unfortunately, as previously mentioned, there is not an absolute best classification algorithm, since the generalisation capability of any classifier mainly depends on the specific application and the characteristics of the data, like number of features and training samples, and correlation between feature values. Some classifiers use very simple models for classification, either in terms of complexity of the decision function or dimensionality of the feature space. Several times

they may perform better than more complex models, although complex models may be intuitively expected to have a higher generalisation capability.

To reach higher performance, the more complicated models typically require a higher number of parameters to set. Consequently, larger training sets are required to provide a reliable estimation of such parameters, and to avoid the problem of overfitting. Moreover, also the number of features used to build a classifier influences its generalisation capability. Intuitively, one may expect that the more features are used the better, since more detailed information about the samples is given to the classifier. However, as the number of features increase, the volume of the feature space increases exponentially, and a very large number of training samples is typically required to obtain a good estimation of the classification function. The reason behind is that high-dimensional functions can be much more complex than low-dimensional ones, and such complexities are harder to learn from data (unless a very large amount of data is available). This problem is generally known in the literature as *curse of dimensionality* [1,2]. A possible method to overcome this problem is to include some knowledge about the data during the classifier design, namely, assuming some model for the class-conditional pattern distributions, or a specific kind of classification function (*i.e.*, linear, quadratic). Another possibility is to use feature selection or dimensionality reduction approaches, like principal component analysis (PCA), or linear discriminant analysis (LDA) [1, 2]. The above mentioned issues are typical examples of why simple classification models may be preferred in some applications.

1.2 Performance Measurements

To compare different classifiers and to decide which of them is the most suitable for a specific problem, we evaluate their performances on the test set, which give us an idea, an approximation of how the classifiers will work in the operational condition.

The most straightforward method for error computation is to compute the rate of misclassified patterns of the test set n_{err}/n, that is called *apparent error*, because it is a very optimistic estimate of the true error that our classifier will do in the future. However this measure is restrictive, it is preferable to separate the error terms on the classes, because usually the recognition task is associated to actions will be taken, and which have different importance for the different classes. For example, in cancer recognition settings, misclassify an ill patient as healthy is much worse than the opposite case, a healthy patient classified as ill. So, it is better to choose a classifier that minimizes the error on the class linked to the most critical action. For this reason the overall error is splitted in two parts:

- the **False Positive Rate (FPR)**: the probability to assign a sample from the class labelled with -1 (the negative one) to the class labelled with 1 (the positive one). The complement is the **True Negative Rate (TNR)**;

- the **False Negative Rate (FNR)**: the probability to assign a sample from the class labelled with 1 to the class labelled with -1. The complement is the **True Positive Rate (TPR)**

$$FPR = \frac{n_p}{N} \qquad FNR = \frac{p_n}{P} \tag{1.9}$$

where N and P are the total number of negative and positive examples, and n_p (p_n) are the negative (positive) samples classified as positives (negatives). These measures define the working point of the classifier.

Another possible measure is the **receiver operating characteristic (ROC)**, or ROC curve; it is a graphical plot that shows how the performances of a binary classifier change in terms of true positive rate against the false positive rate, as its discrimination threshold is varied. The best possible prediction method would yield a point in the upper

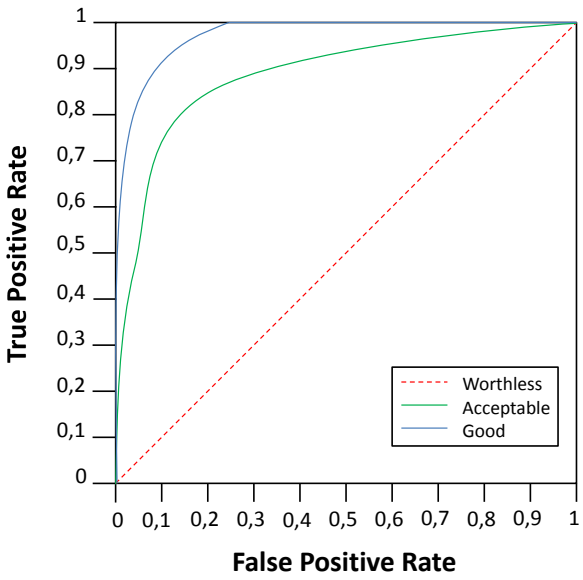


Figure 1.5: Example of ROC curves.

left corner or coordinate (0,1) of the ROC space, representing 100% of right classification of the positive and negative samples. It corresponds to have an optimal system, stable for any value of decision threshold. A completely random guess would give a point along a diagonal line (the so-called line of no-discrimination) from the left bottom to the top right corners (regardless of the positive and negative base rates). This curve represents the worst case. The diagonal divides the ROC space. Curves above the diagonal represent good classification results (better than random), curves below the line indicate the

necessity to invert the decision function to have a good classifier.

Often a more concise performance measure is used, given by the **Area Under the ROC Curve (AUC)**

$$AUC = \int_0^1 TP(FP)\, dFP \tag{1.10}$$

The AUC values range from 0 to 1: AUC=0 corresponds to a classifier which misclassify all the samples. Notice that, by inverting the decisions of such a classifier, we obtain a perfect classifier (namely, a classifier which correctly classifies all samples), whose AUC value is equal to 1. When AUC=0.5, the classifier is not able to distinguish between the two classes. Basically, it classifies a sample completely at random, namely, either as negative or as positive with the same probability. Sometimes it can be more useful to look at a specific region of the ROC rather than at the whole curve. As an instance, in security settings is preferable to focus to the classifiers behaviour within low values of FP, because the systems usually work in this range (if a classifier has a high working FP, it results inadequate). So, it is possible to compute the *partial* AUC as:

$$pAUC = \int_0^a TP(FP)\, dFP\, \frac{1}{a} \tag{1.11}$$

where a indicates the FP value within you want to compute the partial AUC; note that the result is divided by a to rescale it in the 0-1 range, otherwise it would be too small.

1.3 Type of classifiers

Now, we give a short description of the classifiers used throughout this thesis.

1.3.1 Support Vectors Machine

A Support Vector Machine (SVM) is basically a linear classifier, namely, it finds a hyperplane in the feature space which separates the two classes. Its decision function can be generally written as

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} - b\,, \tag{1.12}$$

$\mathbf{w}$ e b are parameters which determine the position of the decision hyperplane in the feature space: its orientation by the hyperplane normal $\mathbf{w}$ and its displacement by b. In general, there are many hyperplanes that might correctly classify the data (Fig.1.6-*left*).

1.3.1.1 Hard margin

If the training data are linearly separable, we can select two parallel hyperplanes which separate the two classes, so that the distance between the closest positive and negative

samples is as large as possible. This distance is called "margin", and the maximum-margin hyperplane is the one that lies halfway in between the two parallel hyperplanes (Fig.1.6-*center*). The latter can be described by the equations

$$\mathbf{w}^T\mathbf{x} - b = 1 \qquad \textit{and} \qquad \mathbf{w}^T\mathbf{x} - b = -1\,. \tag{1.13}$$

Geometrically, the distance between these two hyperplanes is $\frac{2}{\|\mathbf{w}\|}$. By requiring each data point to lie on the correct side of the margin, we obtain the following constrained optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \tag{1.14}$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x} - b) \geq 1\,. \tag{1.15}$$

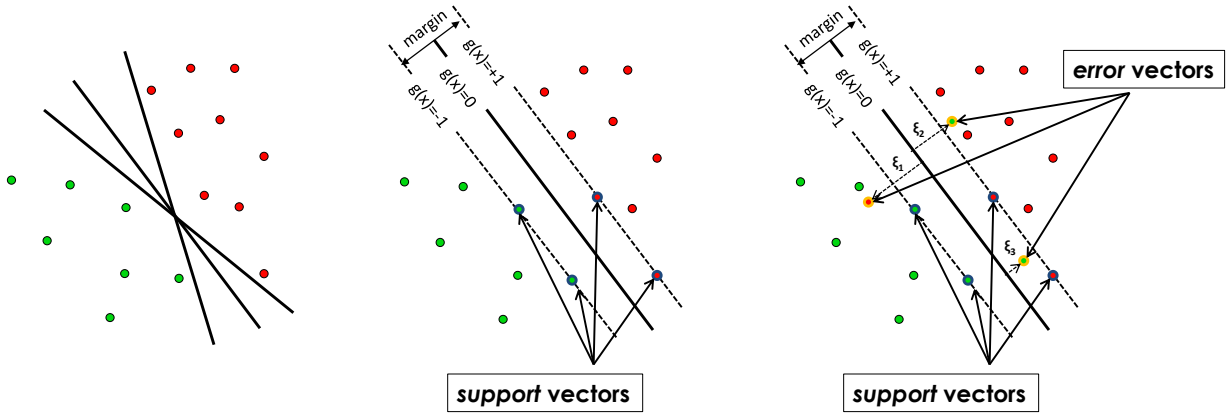


Figure 1.6: *Left.* Several hyperplanes can be used to separate two distributions. *Center.* SVM in hard margin case (distribution linearly-separable). The support vectors are circled in blue. *Right.* SVM in soft margin case. The support and error vectors are circled in blue and yellow, respectively.

1.3.1.2 Soft margin

The presence of linearly-separable data is an ideal condition, almost never found in a real situation. In [5], SVMs were subsequently extended to cope with this type of problem, for which some samples are allowed to violate the margin (Fig.1.6-*right*). In this case, the

problem becomes

$$\min_{\mathbf{w},b,\xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \xi_i \tag{1.16}$$

$$\text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x} - b) \geq 1 - \xi_i, \quad i = 1, .., n, \tag{1.17}$$

$$\xi_i \geq 0, \quad i = 1, .., n. \tag{1.18}$$

where the variables ξ_i, referred to as slack variables, represent the extent to which the samples $\mathbf{x}_i$, violate the margin. The parameter C tunes the trade-off between reducing classification error on the training data (measured using the hinge-loss function), and margin maximization (enforced by penalizing the ℓ_2 norm of the classifier weights). Problems 1.14/1.16 are referred to as the primal problems, and they are solved by means of their dual Lagrange multipliers. Their solutions are given by

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{1.19}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.20}$$

$$\begin{cases} \alpha_i \geq 0, & \text{if hard margin} \\ 0 \leq \alpha_i \leq C, & \text{if soft margin.} \end{cases} \tag{1.21}$$

where α_i are the Lagrange multipliers. Interesting properties of the SVM arise from its dual form. Firstly, the normal to the hyperplane $\mathbf{w}$ can be expressed as a convex linear combination of the training samples. Secondly, the solution to the dual problem is sparse, and only samples that lie on or within the hyperplane margin exhibit a non-zero contribution: if $\alpha = C$, the sample violates the margin (*error vector*); if $0 < \alpha < C$, the sample lies exactly on the margin (*support vector*); if $\alpha = 0$ the sample is correctly classified. As a consequence, the displacement b is typically determined by averaging $\mathbf{w}^T\mathbf{x}_i - y_i$ over the set of support vectors.

1.3.1.3 Kernel Trick

Sometimes, a linear classifier can not separate in appropriate way the two class distributions, it is not complex enough. To overcome this problem, we can use a non-linear function $\phi : \mathbb{R}^d \to \Phi$, that maps the training samples into a higher dimensional feature space. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space, that may be non-linear in the original input space (Fig.1.7).

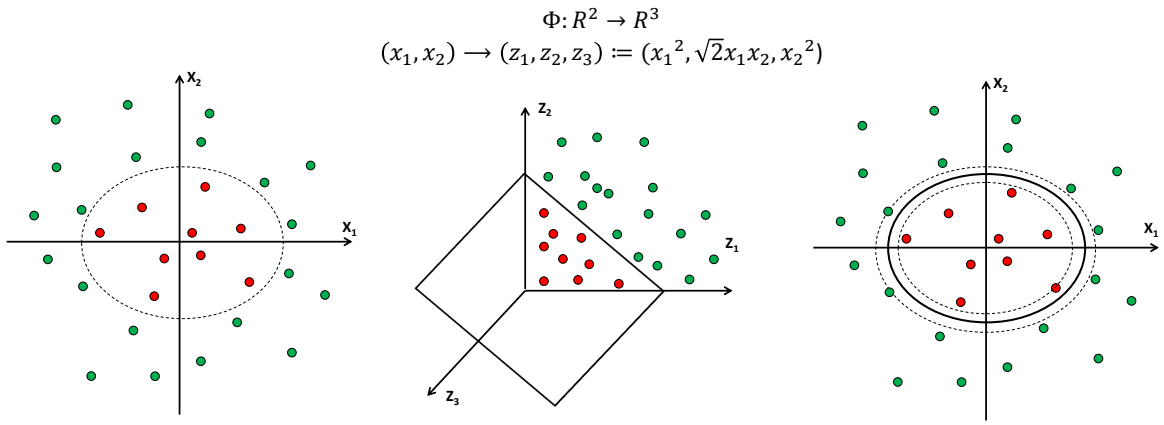


Figure 1.7: Example of two concentric distributions, that are impossible to separate with a hyperplane [3]. Mapping the training samples in a transformed feature space allows an easy linear separation. Note that the hyperplane in the transformed feature space becomes an ellipsoid in the starting feature space.

The above solution can be derived by the method of Lagrangian multipliers which yields the dual problem. In matrix form, this dual can be expressed as

$$\min_{\alpha} \frac{1}{2}\alpha^T \mathbf{Q}\alpha - \mathbf{1}_n^T\alpha, \tag{1.22}$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad i = 1, .., n, \tag{1.23}$$

$$\textstyle\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{1.24}$$

where $\mathbf{Q} = \mathbf{K} \circ yy^T$ (the Hadamard product of $\mathbf{K}$ and yy^T), and $\mathbf{1}_n$ is a column vector of n ones. $\mathbf{K}$ is the kernel matrix, whose elements are $K_{i,j} = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ ($K_{i,j} = \mathbf{x}_i^T\mathbf{x}_j$ in the linear case). Since only inner products between samples are required to compute the solution and predict the class labels, one does not need to know ϕ explicitly, but only the corresponding kernel function. This is known as the *kernel trick*, and, due to its wide adoption, SVMs are often learned by directly solving the dual problem. The most used kernel is the **gaussian Radial Basis Function (RBF)**:

$$\mathbf{K}(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \parallel \mathbf{x}_i - \mathbf{x}_j\parallel^2), \quad for\ \gamma > 0 \tag{1.25}$$

1.3.1.4 One-Class SVM

For particular applications, it can be difficult or expensive to find samples from each class. An example is the classification of the operational status of a nuclear plant as 'normal';

in this scenario, there are (fortunately) few or no examples of catastrophic system states, only the statistics of normal operation are known.

To cope with this problem, Schölkopf et al. [4] have introduced the One-Class SVM. It infers the properties of normal cases and from these properties it can predict effectively if the inputs correspond to a normal situation or not. The OC-SVM basically separates all the data points from the origin, in feature space, and maximizes the distance from this hyperplane to the origin. This results in a binary function which captures regions in the input space where the probability density of the data lives. The quadratic programming minimization function is slightly different from the two-class version stated above:

$$\min_{\mathbf{w},\rho,\xi} \frac{1}{2} \|\mathbf{w}\|^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \xi_i - \rho \tag{1.26}$$

$$\text{s.t.} \quad (\mathbf{w}^T \phi(\mathbf{x}_i) \geq \rho - \xi_i, \quad i = 1, .., n, \tag{1.27}$$

$$\xi_i \geq 0, \quad i = 1, .., n. \tag{1.28}$$

where ρ indicates the distance from the origin and ν corresponds to the fraction of outliers, *i.e.*, training examples misclassified, that we accept in the learning phase. Moreover, it represents a lower bound on the number of training examples used as support vectors. Due to the importance of ν parameter, this machine is also called ν-SVM. Solving the problem with Lagrange multipliers we obtain

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i) \tag{1.29}$$

$$\alpha_i \leq \frac{1}{\nu n} \tag{1.30}$$

$$\sum_{i=1}^{n} \alpha_i = 1 \tag{1.31}$$

1.3.2 Decision Tree

Decision tree is a type of supervised learning algorithm that works for both categorical and continuous input and output variables. In this technique, we split the population of samples into two or more homogeneous sets (or sub-populations), through a sequence of questions, in which the next question asked depends on the answer to the current one. This approach is particularly useful for non-metric data, since all of the questions can be asked in a "yes/no" or "true/false" or "value(property) $\in$ set of values" style that does not require any notion of metric [2]. The structure of a decision tree follows the shape of a real tree, it starts from the root, up to the canopy; we can see in Fig.1.8 an example

of decision tree. The first node, or root node, is displayed at the top, that asks for the value of a particular property of the pattern. It is connected by successive (directional) branches to other nodes, each one corresponds to different possible values of the answer. The next step is to make the decision at the appropriate subsequent node, which can be considered the root of a sub-tree. We continue this way until we reach a leaf node, which has no further question. Each leaf node bears a category label and the test pattern is assigned to the category of the leaf node reached.

The most important step while training this classifier is the choice of the attribute to consider for each node. Different algorithms use different metrics to select these attributes, based on the homogeneity of the target variable within their values. This yields different classification functions.

Temperature	Outlook	Humidity	Windy	Play Tennis
hot	sunny	high	false	no
hot	sunny	high	true	no
hot	overcast	high	false	yes
cool	rain	normal	false	yes
cool	overcast	normal	true	yes
mild	sunny	high	false	no
cool	sunny	normal	false	yes
mild	rain	normal	false	yes
mild	sunny	normal	true	yes
mild	overcast	high	true	yes
hot	overcast	normal	false	yes
mild	rain	high	true	no
cool	rain	normal	true	no
mild	rain	high	false	yes

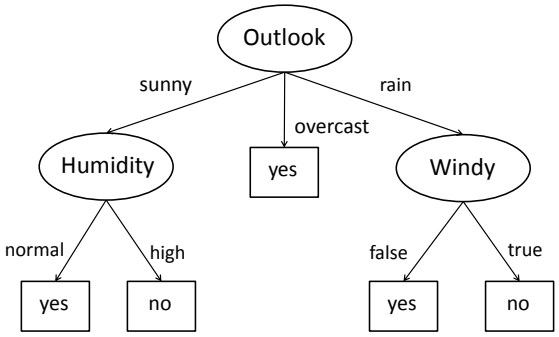


Figure 1.8: Typical example of decision tree building, regarding the decision to play tennis based on the weather. Note that in different paths we evaluate different attributes (humidity on the left and windy on the right) and that not all of the properties are considered (temperature).

1.3.3 Random Forest

Even if we identify the most discriminant rules for each node, the Trees could become very deep and tend to learn highly irregular patterns, overfitting their training sets. To overcome this possible behaviour, the so-called Random Forest classification algorithm has been proposed. It is an ensemble learning method for classification, which operates by constructing a multitude of Decision Trees. Each Decision Tree is trained using the *bootstrap aggregating* procedure: starting from the initial training set D, this technique

generates m new training sets, each one for a different tree, obtained by sampling data from D uniformly and with replacement. This clearly means that some samples from D may repeatedly appear in several of the new training sets. During operation, the Random Forest assigns a sample to the class corresponding to the mode of the predicted ones from the individual trees.

1.4 Applications and Limitations

Thanks to its generalization capability, machine learning is widely used in computer security applications. Nowadays, computer technology is pervasively entered in our everyday life, and a lot of activities depend on it. We are inclined to share, more or less consciously, sensitive information about us. We access and manage resources, from personal computers to industrial control plants, through computer systems.

These systems can be targeted by attacks from malicious agents, motivated by economical (*e.g.*, credit card numbers theft) or terrorist (*e.g.*, sabotage of nuclear plants) scopes. To ensure that these attacks target as many users as possible, a lot of malicious software (malware) have been developed during these years. Such software mainly aim to take the control of the victim's device, in order to steal their private information, encrypt their data, or perform additional malicious actions that address other systems.

Due to their diffusion and dangerousness, it is crucial to provide adequate protections against malware. A number of anti-malware commercial solutions have been developed (*e.g.*, Antiviruses, IDSs) that consistently increase the users' security, based on signature mechanisms or some rules/heuristics. These software detect the attacks finding out the presence of malicious characteristic feature, as the MD5 value of the codes, or the presence of a particular command, stored in a reference database. These solutions are able to protect the users from many known threats and, in many cases, they are able to clean detected infections. However, polymorphism and various obfuscation techniques are widely used by the attackers. This forces the detectors to continuously update their reference databases, in order to detect the new malware 'versions'. As an alternative way to detect malware in a more reliable way, several authors have shown that machine learning can be very effective. Despite the introduction of this advantage, machine learning is not immune to malicious alterations of the input objects. In fact, as we will discuss in Chap.2, it is not designed to cope with intelligent and adaptive adversaries; it potentially introduces novel vulnerabilities that can be exploited from the attackers to evade detection or to mislead the learning process. Previous work has proposed several techniques to prevent or to limit the attacker's actions, among which we highlight [37, 38]. In these works the authors proposed a general attacker's behaviour model that allows one to assess the security of machine learning algorithms, also accounting for some application-specific issues. The proposed model is based on simulating how the attacker may interact with the system, allowing one to discover new unknown vulnerabilities and, if possible, to design

countermeasures to prevent malicious actions. Following this secure-by-design approach, in the last years adversary-aware learning algorithms have been developed ([32, 52, 53]), exploiting secure optimization and game-theoretical models to incorporate knowledge of potential adversarial data manipulations into the learning algorithm. Despite these techniques have been shown to be effective in some adversarial learning tasks, their adoption in practice is hindered by different factors, including the difficulty of meeting specific theoretical requirements, the complexity of implementation, and scalability issues, in terms of computational time and space required during training.

1.5 Contributions of this thesis

Throughout this thesis, we present the work carried out to achieve the design of more secure classifier systems.

We firstly show, in Chap.2, the state of the art concerning the works proposed to counter the adversaries in security settings. We introduce the general framework to represent the attacker's behaviour, that allows us to understand how she can interact with the systems and how the designer can reduce the classifier vulnerabilities. It is shown that, following this taxonomy, it is possible to categorize the attacks in two scenarios: poisoning and evasion. We finally present the attempts proposed to counter these two attack categories.

In Chap.3, we show how the aforementioned general model for the attacker can be used to specific setting, in particular to review previous work on biometric security. This allows us to highlight novel insights on the security of biometric systems when operating in the presence of intelligent and adaptive attackers. We show how this framework enables the categorization of known and novel vulnerabilities of biometric recognition systems, along with the corresponding attacks, countermeasure and defence mechanisms.

Then, in Chap.4, we propose some techniques about the development of secure systems against evasion attacks that are not computationally more demanding than their non-secure counterparts. We show that, based on the specific application setting and the possible attacker's moves, the choice of proper parameters for the classifier is crucial to improve the security of a system.

Finally, in Chap.5, we highlight the conclusions about our proposals and the future works.

Chapter 2

Adversarial Machine Learning

In the last decades, machine learning has been increasingly used in security-sensitive applications, such as spam filtering, malware detection, network intrusion detection and biometric recognition systems. The reason is that traditional security systems were not able to generalise, namely, to detect new (never-before-seen) kinds of attacks, while pattern recognition algorithms have indeed a good generalisation capability.

These applications differ from the classical machine-learning setting in which the underlying data distribution is assumed to be stationary (training and testing data follow the same distribution). Conversely, in security-sensitive applications, samples (and, thus, their distribution) can be actively manipulated by an intelligent, adaptive adversary to undermine classifier operation.

Machine learning was originally designed to reach optimal accuracy, precision performances in classification task, not to deal with adversaries. As a consequence, these algorithms can introduce additional, specific vulnerabilities that can be exploited by carefully-crafted attacks to cause different security violations. This may eventually compromise the whole system security.

The attacks can be devised at any stage of the design process, as well as at operation phase. As an instance, an adversary may compromise the training set used to build a classifier, by injecting carefully designed samples during the data acquisition phase [7] (training phase). Moreover, she can devise some attacks to mislead the feature extraction, *e.g.*, to avoid detection, spam emails are often modified by obfuscating common spam words or inserting words associated with legitimate emails [8–11] (operating phase).

There are also examples of adversarial classification tasks not directly related to security: in data analysis and information retrieval environments, a malicious webmaster may manipulate search engine rankings to artificially promote her web site [12, 13].

To secure a system, a common approach used in engineering and cryptography is **security by obscurity**, that relies on keeping secret some of the system details to the adversary. This approach often leads to a *reactive* arms race between the adversary and the classifier designer, in which each player attempts to achieve his goal by reacting to the

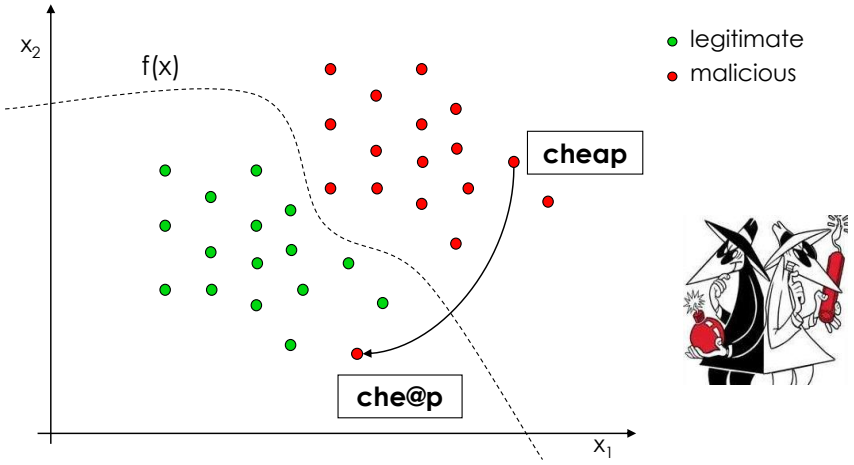


Figure 2.1: Common example of attack against an email spam filter.

changing behaviour of his opponent. At each step, the adversary analyses the system, and develops an attack strategy to exploit the potential vulnerabilities. The designer reacts by analysing the novel attack samples, and searches patches to counter the attacks, trying to do that as soon as possible.

For instance, a spammer may gather some knowledge of the words used by the targeted anti-spam filter to block spam, and then manipulate the textual content of spam emails accordingly; *e.g.*, words like "cheap" that are indicative of spam can be misspelled as "che4p". Secondly, the pattern recognition system designer reacts by analysing the novel attack samples and updating the system consequently; *e.g.*, by retraining the classifier on the newly collected samples, and/or by adding features that can better detect the novel attacks. In the previous spam example, this amounts to retraining the filter on the newly collected spam and, thus, to adding novel words into the filter's dictionary (che4p may be now learned as a spammy word). This reactive arms race continues in perpetuity as illustrated in the left plot in Fig.2.2.

However, reactive approach to this arms race does not anticipate the next generation of security vulnerabilities, *i.e.*, they do not attempt to forecast future attacks and thus the system potentially remains vulnerable to new attacks.

Adversarial Machine Learning is a novel research field that allows to overcome these problems, following a different paradigm, **security by design**. It advocates that systems should be designed from the ground-up to be secure, without assuming that the adversary may ever find out some important system details. Accordingly, the system designer should

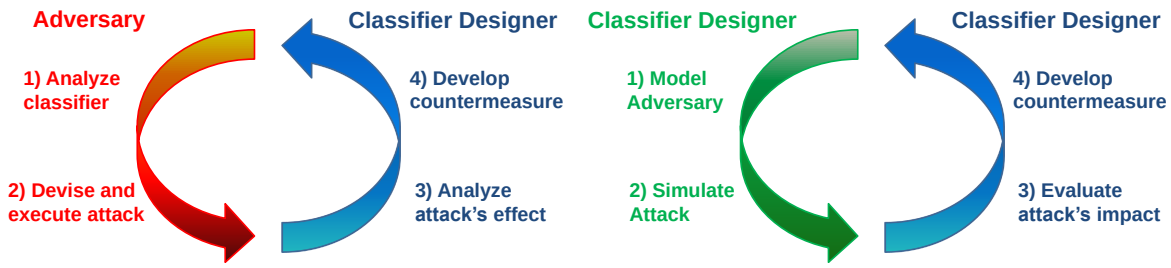


Figure 2.2: Reactive (left) and Proactive (right) arms race.

anticipate the adversary by simulating a *proactive* arms race to *(i)* figure out the most relevant threats and attacks and *(ii)* devise proper countermeasures, before deploying the classifier (see Fig.2.2-right). This paradigm typically improves security by delaying each step of the reactive arms race, as it requires the adversary to spend a greater effort (time, skills, and resources) to find and exploit vulnerabilities. The scope of security evaluation is to address issue *(i)* above, *i.e.*, to simulate a number of realistic attack scenarios that may incur during operation phases, and to assess the impact of the corresponding attacks on the targeted classifier to highlight the most critical vulnerabilities. This amounts to performing a what-if analysis [14], which is a common practice in security. Although security evaluation may also suggest specific countermeasures, the design of secure classifiers, *i.e.*, issue *(ii)* above, remains a distinct open problem. In this way, pattern recognition systems that are properly designed according to the *reactive* and *proactive* security approaches should remain useful for a longer time, with less frequent supervision or human intervention and with less severe vulnerabilities.

2.1 Threat Model

The key point to design a more secure system following the secure-by-design approach is the construction of an adversary's model, based on the knowledge of her goals and capabilities. This way the designer can identify himself with the attacker and find out new vulnerabilities in the system, in order to, if possible, remedy them.

In the literature we can find works where the authors performed security evaluation as a what-if analysis, based on empirical simulation methods ([10], [15] - [25]), and others that proposed analytical methods to evaluate the security of learning algorithms or of some classes of decision functions ([26] - [32]). Their goal was either to point out a previously unknown vulnerability, or to evaluate security against a known attack. In some cases, specific countermeasures were also proposed, according to a proactive/security-by-design approach.

The most relevant proposal is represented by the taxonomy presented in [17], [18], and subsequently extended in [27], that is shown in Tab.2.1. However, in these studies the attacks were simulated by manipulating training and testing samples according to application-specific criteria only, without reference to more general guidelines; consequently, such techniques can not be directly exploited by a system designer in more general cases.

Table 2.1: The attack model proposed by Barreno *et al.* [17]

		Integrity	Availability
Causative:	*Targeted*	Permit a specific intrusion	Create sufficient errors to make system unusuable for one person or service
	Indiscriminate	Permit at least one intrusion	Create sufficient errors to make classifier unusable
Exploratory:	*Targeted*	Find a permitted intrusion from a small set of possibilities	Find a set of points misclassified by the classifier
	Indiscriminated	Find a permitted intrusion	

Starting from the previous proposals and considerations, a more general framework has been formalized, which allows to modelling the adversary's behaviour in every application, making explicit presumptions on the attacker's goal, knowledge of the target system and capabilities of manipulating the input data or the system's components [37, 38]. This allows one to derive the corresponding optimal attack strategy.

The aim is threefold: *(i)* to provide a well-structured categorization of the vulnerabilities of the systems and of the corresponding attacks, also through the definition of different, pertinent attack scenarios; *(ii)* to provide a formal characterization of existing attacks within the framework, and envision more sophisticated and effective attack strategies; and *(iii)* to identify suitable countermeasures and defences inspired by previous work on adversarial machine learning.

We give now a short overview on this general taxonomy.

Adversary's goal.
As in [29], it is formulated as the optimization of an objective function. This function is based on the desired ***security violation***

- *integrity*: to gain unauthorized access to the system,
- *availability*: to generate many classification errors to compromise the normal system operations,

- *privacy*: to obtain confidential information from the classifier

and on the ***attack specificity***

- *targeted*: the attack focus is on a few specific samples,
- *indiscriminate*: the focus is on all malicious samples

according to the taxonomy in [17,27]. For instance, the goal of an indiscriminate integrity violation may be to maximize the fraction of misclassified malicious samples [9, 10, 27]; the goal of a targeted privacy violation may be to obtain some specific, confidential information from the classifier (e.g., the biometric trait of a given user enrolled in a biometric system) by exploiting the class labels assigned to some "query" samples, while minimizing the number of query samples that the adversary has to issue to violate privacy [16,27].

Adversary's knowledge.
Assumptions on the adversary's knowledge have only been qualitatively discussed in previous works, mainly depending on the application at hand. Here it is proposed a more systematic scheme for their definition, with respect to the knowledge of the single components of a pattern classifier

- the training data: the adversary might know the data D or only a portion of it. More realistically, she may not know D exactly, but she may be able to obtain a surrogate dataset sampled from the same distribution as D.
- The feature set: the adversary could know how the features are extracted from each sample. Similarly to the previous case, she may know how to compute the whole feature set, or only a subset of the features.
- The learning algorithm and the kind of decision function (*e.g.*, a linear SVM).
- The classifier's decision function and its parameters (*e.g.*, the feature weights of a linear classifier).
- The feedback available from the classifier, if any (*e.g.*, the class labels assigned to some "query" samples that the adversary issues to get feedback [16, 27]).

It is worth noting that realistic and minimal assumptions about what can be kept fully secret from the adversary should be made, as discussed in [27].
The worst-case scenario in which the attacker has full knowledge of the attack system is usually referred to as *perfect knowledge* (**PK**) case, and it allows one to empirically evaluate an upper bound on the performance degradation that can afflict the system under attack. Attacks with *limited knowledge* (**LK**) have also been considered, to simulate more realistic settings. In this case, the attacker is assumed to have only partial knowledge of

the system: for example, she can collect data to create a surrogate dataset, but she knows the feature set and the feature selection algorithm. She can thus replicate the behaviour of the attacked system using the surrogate data, to construct a set of attack samples. The effectiveness of these attacks is then assessed against the target system (trained on the true data).

Adversary's capability.
It refers to the control that the adversary has on training and testing data, and where she locates the attacks. This task defines the malicious actions in terms of

- the attack influence:
  - *causative*, if the attack undermines the learning algorithm to cause subsequent misclassification,
  - *exploratory*, if it exploits knowledge of the trained classifier to cause misclassification, without affecting the learning algorithm

  as defined in [17, 27]. Thus, causative attacks may influence both training and testing data, or only training data, whereas exploratory attacks affect only testing data;
- whether and to what extent the attack affects the class priors;
- how many and which training and testing samples can be controlled by the adversary in each class;
- which features can be manipulated, and to what extent, taking into account application-specific constraints (e.g., correlated features can not be modified independently, and the functionality of malicious samples can not be compromised [9, 10]).

It is clear that excessive obfuscation or manipulation of the samples is not possible without compromising the functionality of the attack. This behaviour has been formalized in terms of application-dependent constraints in previous works [40, 41]; in particular, in terms of bounds on the distance between the initial malicious sample $\mathbf{x}$ and the manipulated one $\mathbf{x}'$ in feature space.
As discussed in [41], two kinds of constraints have been mostly used when we model real-world adversarial settings, leading one to define *sparse* (ℓ_1) and *dense* (ℓ_2) attacks. Bounding the ℓ_1 distance between $\mathbf{x}$ and $\mathbf{x}'$ yields a sparse attack, as it represents the case where the cost depends on the number of modified features. For example, when instances correspond to text (*e.g.*, the email's body) and each feature represents the occurrences of a given term in the text, the attacker usually aims to change as few words as possible. Instead, the ℓ_2 distance yields a dense attack, as it represents the case in which the cost to modify features is proportional to the distance between the original and modified sample

in Euclidean space. For instance, when considering images as the input data, usually the attacker prefers making small changes to many or even all pixels, rather than significantly modifying only few of them. This amounts to only slightly blurring the image, instead of obtaining a salt-and-pepper noise effect (as produced by sparse attacks), and the final effect is less visible to the human eye.

Adversary's strategy.
One can finally define the optimal attack strategy, namely, how training or testing data should be quantitatively modified to optimize the objective function characterizing the adversary's goal. Such modifications are defined in terms of:

- how the class priors are modified;
- what fraction of samples of each class is affected by the attack;
- how features are manipulated by the attack.

According to the adversary's goal (generally expressed in terms of an objective function $g(a;\theta)$), knowledge (given in terms of the parameter vector $\theta \in \Theta$) and capability (which defines the feasible set of attack strategies $a \in \mathcal{A}$), an optimal attack strategy can be defined to implement the attack, as:

$$\max_{a \in \mathcal{A}} g(a;\theta) \tag{2.1}$$

2.2 Categorization of attack scenarios

Looking at the attacks from the broader perspective opened by the previous framework allows us to identify two main scenarios.

2.2.1 Evasion

Evasion attacks are the most prevalent type of attacks that may be encountered in adversarial settings during system operations. In this setting, malicious samples are modified at test time to evade detection in order to be misclassified as legitimate. No influence over the training data is assumed. Here we formalize evasion attacks in terms of the model previously explained.
Attacker's goal. In evasion attacks, the goal is to cause an *integrity violation*: the attacker modifies malicious samples (*e.g.*, a spam email) to have it misclassified as legitimate (with the largest confidence) by the classifier, without the necessity to compromise

its functionality. Moreover, the specificity can be *targeted or indiscriminate*, it depends on how many samples she wants to change.

Attacker's knowledge. The attacker can have different levels of knowledge of the targeted classifier; she may have *limited or perfect knowledge* about the training data, the feature set, and the classification algorithm [37,40]. Usually we focus on perfect-knowledge (worst-case) attacks. Although it may be overly pessimistic to assume that the attacker knows everything about the targeted system, this often reveals interesting properties of learning algorithms, as it highlights the worst possible performance degradation that may be incurred under attack.

Attacker's capability. In evasion attacks, the attacker operates during the operating phase, without altering the training process (*exploratory influence*), and she can modify only malicious samples. The amount of feasible manipulations is often bounded, as the malicious data has to preserve its intrusive functionality. For example, malware has to embed a valid exploitation code for the attack to be effective, and spam emails have to remain readable by humans.

Attack strategy. Having defined the attacker's goal, knowledge and capability, one can finally formalize the attack strategy, *i.e.*, the procedure for obfuscating malicious data to evade detection, in terms of an optimization problem. Let us denote the legitimate and malicious class labels respectively with -1 and $+1$, and assume that the classifier's decision function is $f(\mathbf{x}) = \text{sign}\,(g(\mathbf{x}))$, where $g(\mathbf{x}) \in \mathbb{R}$ is the discriminant function, and $\mathbf{x}$ is the representation of a sample in a d-dimensional feature space. For example, for linear classifier, $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b \in \mathbb{R}$, where $\mathbf{w} \in \mathbb{R}^{\mathsf{d}}$ are the feature weights, and $b \in \mathbb{R}$ is the bias. Given a malicious sample $\mathbf{x}$, the goal is to find the sample $\mathbf{x}^*$ that minimizes the classifier's discriminant function $g(\cdot)$ (*i.e.*, that is classified as legitimate with the highest possible confidence) subject to the constraint that $\mathbf{x}^*$ lies within a distance $d_{\max}$ from $\mathbf{x}$:

$$\mathbf{x}^* = \quad \arg\min_{\mathbf{x}'} g(\mathbf{x}') \tag{2.2}$$

$$\text{s.t.} \quad d(\mathbf{x}', \mathbf{x}) \leq d_{\max}\ ,\ \mathbf{x}_{\text{lb}} \preceq \mathbf{x}' \preceq \mathbf{x}_{\text{ub}}\ , \tag{2.3}$$

where $\mathbf{x}_{\text{lb}} \preceq \mathbf{x}' \preceq \mathbf{x}_{\text{ub}}$ represents a box constraint (it defines the min and max values allowed for each feature: for example, in case of features normalized between 0 and 1, the attacker have to modify the features of her samples remaining in $[0:1]$ range), and the distance measure $d(\cdot,\cdot)$ is defined in terms of the cost of data manipulation (*e.g.*, the number of modified words in each spam) [9,11,28,37,40]. Sparse and dense evasion attacks are simply defined based on whether $d(\cdot,\cdot)$ corresponds to the ℓ_1 or to the ℓ_2 distance, respectively. Examples of evasion attacks are shown in Fig.2.1,2.3.

2.2.2 Poisoning

Machine learning algorithms are often re-trained on data collected during the operation phase to adapt to changes in the underlying data distribution. Within this scenario, an

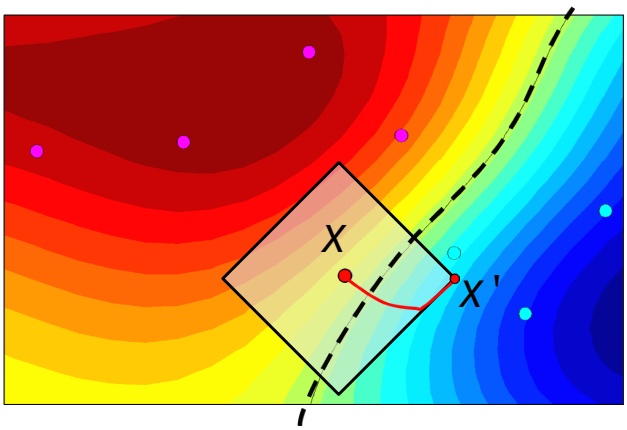


Figure 2.3: Example of ℓ_1 (sparse) attack against a kernel SVM [40].

attacker may poison the training data by injecting carefully designed samples to eventually compromise the whole learning process and consequently lead to misclassify as many samples as possible at test time. Poisoning may thus be regarded as an adversarial contamination of the training data.

Attacker's goal. For a poisoning attack, the attacker's goal is to find a set of points whose addition to the training set maximizes the classifier's errors. She causes both *integrity violation*, because after the attack she can evade the classifier without modifying her malicious samples, and *availability violation*, because the system does not recognize anymore the legitimate accesses. The specificity of the attacks can be *targeted or indiscriminate*, depending on how many samples the attacker wants to modify.

Attacker's knowledge. As for the evasion scenario, we assume that the adversary knows the training samples, the feature representation and the classification algorithm. In the same way, usually we focus on the *perfect-knowledge* attacks, so as to consider the worst-case and to compute the maximum error rate that the adversary can inflict through the poisoning.

Attacker's capability. In this scenario the attacker has a *causative influence*, she operates during the learning phase, adding attack samples into the training set in order to modify the class priors and the class-conditional distributions. Even here, she can alter the feature values of the attack data within some lower and upper bounds.

Attack strategy. Under the previous setting, we can formalize the attack strategy in terms of an optimization problem. As in evasion setting, let us denote the legitimate and malicious class labels respectively with -1 and $+1$, $\mathbf{x}$ is the representation of a sample in a d-dimensional feature space, the classifier's decision function is $f(\mathbf{x}) = \text{sign}\,(g(\mathbf{x}))$,

where $g(\mathbf{x}) \in \mathbb{R}$ is the classifier's discriminant function. Moreover, we introduce the function $Loss(\mathbf{x}')$, that expresses the error computed by the classifier on the samples in the operating phase, after the injection of the malicious sample $\mathbf{x}'$ inside the training set. The optimization problem can be thus written as:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}'} Loss(\mathbf{x}') = \sum_{k=1}^{m} (1 - y_k f_{\mathbf{x}'}(\mathbf{x}_k)) \tag{2.4}$$

$$\text{s.t.} \quad d(\mathbf{x}', \mathbf{x}) \leq d_{\max} \ , \ \mathbf{x}_{\text{lb}} \preceq \mathbf{x}' \preceq \mathbf{x}_{\text{ub}} \ , \tag{2.5}$$

where $f_{\mathbf{x}'}(\cdot)$ is the initial decision function modified by the addition of $\mathbf{x}'$ in the training set. The constraint on the maximum possible modifications is defined as for the evasion case.

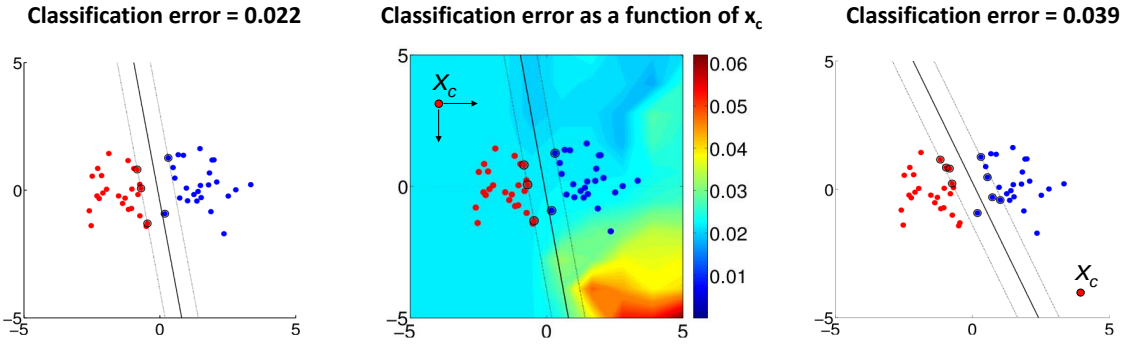


Figure 2.4: Poisoning attack against a linear classifier. The attacker injects a malicious sample that causes the greatest classification error possible. Example adapted from [24].

2.3 Constructing real-world attack samples

The attack strategies discussed in this section allows one to find a malicious sample in terms of a set of desired feature values. Clearly, feature mappings in real-world security-related tasks can be very difficult to reverse-engineer and, accordingly, constructing the corresponding real-world attack samples (*e.g.*, malware samples) may not be trivial. As widely discussed in previous work [27, 37, 38, 40], it is clear that this problem demands for application-specific solutions, and it is thus out of the scope of this work. On the other hand, in some cases, feature mappings can be easily inverted and the corresponding samples easily constructed.

2.4 Related Work

According to the paradigm of security-by-design, several recent works have proposed modifications to learning algorithms and novel learning techniques, that explicitly take into account a specific kind of adversarial data manipulation.

Evasion. For linear classifiers, the underlying rationale has been that of devising classifiers with more evenly-distributed feature weights, to intuitively enforce the attacker to manipulate more feature values to evade detection. Based on this intuition, different *heuristic* techniques have been proposed [10, 38].

Bruckener at al. [32] have proposed to model the interactions between the attacker and the classifier as a game in which players aim to maximize their own payoff function. The adversary's goal is to evade detection by minimally manipulating some attack samples, while the classifier is retrained to correctly classify them. This procedure is iterated until a Nash equilibrium point is reached, *i.e.*, when no player can improve his payoff function unilaterally by playing a different strategy. To simulate the attacker's actions, one can resolve the optimization problem with many well-known techniques, like gradient descent, or quadratic techniques such as Newton's method, BFGS, or L-BFGS. If the function is not differentiable, as in Random Forest, the attacks are simulated using black-box optimization strategies, like genetic algorithm [58] and greedy descent techniques [59]. Another method is the learning with invariances, essentially a minimax approach in which the learning algorithm is modified to minimize the maximum loss incurred under worst-case manipulations of the attack samples. In particular, different variants of the SVM learning algorithm have been proposed to account for a maximum number of worst-case feature manipulations, including deletion, insertion, and rescaling [52, 53].

Poisoning. To significantly compromise the training phase of a learning algorithm, an attack has to exhibit some characteristics that are different from those shown by the rest of the training data, otherwise it would have no impact at all. Following this consideration, some proposed techniques generate a model of the plausible distribution of the classes, in order to sanitize the training from the malicious outlier [54]; other solutions exploit robust statistics [49, 50] to mitigate the outliers' impact on learning (*e.g.*, robust principal component analysis [19]).

2.5 Open Issues

Through this chapter we have seen how important the proactive approach is to design a more secure system. The framework that we previously presented provides to designers general guidelines to empirically evaluate the classifier security in any adversarial environment.

In Chap.3 we show how to apply this framework to the case of biometric recognition systems. We highlight that it enables one to generate a well-structured categorization of

the systems' vulnerabilities and of the corresponding known attacks. Moreover, we show that it allows one to identify novel attacks and mechanisms to improve the security of biometric systems. In particular, we present an efficient countermeasure for a recently proposed type of poisoning attack.

The use of this framework to study a particular scenario allows the designers to understand how and where attackers can attack the classifier. Accordingly, by taking into account such potential threats, one may design specific countermeasures to increase the system security level. State-of-the-art solutions, as shown in the last section, present however several problems that limit their usability in real applications:

- standard optimization techniques can be used to solve the optimization problems corresponding to evasion and poisoning attacks (Eq.2.2-2.4), but they can be very computationally demanding depending on the scenario at hand. As an example, standard optimizers are not very efficient when dealing with discrete input spaces. To overcome this problem, in Chap.4 we present an evasion algorithm based on gradient descent, specifically tailored to the given kind of attack and characteristics of the input feature space. In particular, the latter two considerations allow making our algorithm faster and more suitable for every application setting with respect to the other solvers.

- Regarding non-differentiable classifiers, as Random Forests, the strategies to simulate the attacks are very slow and with high computational complexity. To make the evasion attack practicable in this setting, we construct a differentiable surrogate of the non-differentiable classifier, trained on the training set re-labelled from the latter one, that will be evaded with the aforementioned evasion algorithm.

- In evasion settings,

  - for linear classifiers, the countermeasures based on learning more evenly-distributed feature weights to improve security are mainly heuristic. Thus, a clear understanding of the conditions under which they can be effective, or even optimal, is still lacking. In Chap.4 we provide a clearer and theoretically-sound explanation to this approach, and show how to improve it.
  - Some solutions propose to repeatedly attack and retrain the classifier on the attack samples. These mechanisms enable us to account for potential attacks during the design phase, although they may be computationally expensive. In Chap.4 we propose some solutions that are not computationally more demanding than their non-secure counterparts.

It is worth noting that the aim of the application examples presented in the biometric chapter is to show, in a simple and clear manner, how the threat model works in a

particular setting and how it can help finding new vulnerabilities and developing novel countermeasures.

In the secure learning chapter, we present a more rigorous study on the proposed countermeasures to improve the security of a classifier.

Chapter 3

Adversarial Biometric Recognition

In the previous chapter we presented a general threat taxonomy to model the attacker's behaviour, highlighting its flexibility to be used in any adversarial pattern recognition scenario, in order to analyse vulnerabilities of the systems and potential threats. We now show how it is possible to use this framework in a specific application setting, involving biometric recognition systems. The results of this work were published in [39].

Biometric identity recognition is a clear example of a widespread and still growing application field in which security is a key issue, and pattern recognition techniques play a major role. Different vulnerabilities of biometric systems, specific attacks that can exploit them, and some corresponding countermeasures have been analysed in the literature [33, 42] and in research projects. However, all existing efforts disregarded the potential, specific vulnerabilities introduced by pattern recognition algorithms used in biometric systems, and thus the investigation of the corresponding attacks and countermeasures.

In our work, we argued that looking at biometric system security from the perspective of adversarial machine learning not only provides an original categorization of existing attacks against such systems, but it also allows us to consider more sophisticated attacks targeting vulnerabilities of the learning algorithms used in these systems, along with the countermeasures already proposed in the field of adversarial machine learning.

This chapter is organised as follows. First we introduce the standard architecture of a biometric recognition system. Then we give the definition of a more complete taxonomy of attacks against biometric systems, based on the model explained in Chap.2, which allows one to identify novel attack scenarios associated to specific vulnerabilities of machine learning and pattern recognition algorithms, besides encompassing known attacks. Furthermore, we show how this model enables us to design corresponding countermeasures, building on solutions proposed in adversarial machine learning, which can give rise to the design of novel, secure-by-design algorithms capable of improving adversarial biometric identity recognition. We finally discuss two application examples of the possible aforementioned achievements. We first show how a skilled attacker may fabricate more ef-

fective face spoofing attacks, and then highlight a new vulnerability of adaptive biometric systems, devising the corresponding attack and a possible countermeasure.

3.1 Architecture of a Biometric Recognition Systems

Biometric recognition systems operate either in *enrollment* or in *recognition* mode [33]. During **enrollment**, each client provides his biometric traits and identity, in the presence of a human supervisor. A set of *reference templates* for each client is then stored in the *template database* along with the corresponding identity. During **recognition**, the biometric system is expected to recognize a previously-enrolled client by comparing the submitted traits with those stored in the template database. Biometric systems may operate either in an *verification* or in a *identification* setting. In *verification* settings, biometric systems are often used to control access to protected resources, including confidential information or services. A user aiming to access them has to provide his/her biometric trait and claim an identity. The system then verifies whether the claim is genuine (*i.e.*, the user's identity is the claimed one) or not (*i.e.*, the user is an impostor trying to impersonate another client), and allows access only in the former case. This procedure is illustrated in Fig. 3.1, which is general enough to also account for multi-biometric systems; in this case, the fusion rule s_c should aggregate the matching scores coming from *all* biometric traits. In *identification* settings, instead, no identity claim is made: a user provides only the requested biometric trait $\mathbf{x}$, and the system is expected to correctly recognize the identity among the ones in the template database, by matching $\mathbf{x}$ against all the known clients' templates; the corresponding scores $s_c(\mathbf{x})$ are then sorted in descending order to provide a list of the most likely candidate identities.

To account for natural changes of biometric traits over time (*i.e.*, biometric *aging*), and changes in the environmental or acquisition conditions during verification, *adaptive* biometric systems have been proposed, to automatically update the stored templates during verification [34, 35]. A popular technique is template self-update: if the submitted trait is sufficiently similar to the reference templates of the claimed identity, one of such templates is updated by exploiting information coming from the submitted trait, according to a given policy. A simple update policy is called *nearest-out self-update*, and replaces the most similar template to the submitted trait with the latter [24, 36].

3.1.1 The Attack Surface

Previous work has identified the main attack points and vulnerabilities of biometric recognition systems, along with the corresponding attacks [33, 42]. First, any system is subject to **intrinsic failures** not produced by adversarial attempts, *i.e.*, rejected genuine claims and accepted *zero-effort* impostors (*i.e.*, impostors that do not exert any special effort to intrude). Besides this, a number of **adversarial attacks** have been also considered in early work, leading to the identification of eight potentially vulnerable attack points

highlighted by the red circled numbers 1–8 in Fig. 3.1 [42]. We additionally consider here points 9-11: they correspond to vulnerabilities of *adaptive* biometric systems that update clients' templates during operation, which we recently exploited to implement a *template poisoning* attack [24, 36]. The set of all attack points defines the *attack surface* of a biometric recognition system. The corresponding attacks can be categorized into four main groups, according to the targeted system component [33]: attacks to the sensor (point 1), to interfaces and channels connecting different modules (points 2, 4, 7), to processing modules and algorithms (points 3, 5, 8–11), and to the template database (point 6). We discuss them below, along with the corresponding countermeasures proposed so far, that are also summarized in Table 3.1. It is also worth mentioning here a special category of attacks, known as *insider attacks*, where the attacker is colluded with a system administrator or exercises coercion to escalate privileges [33].

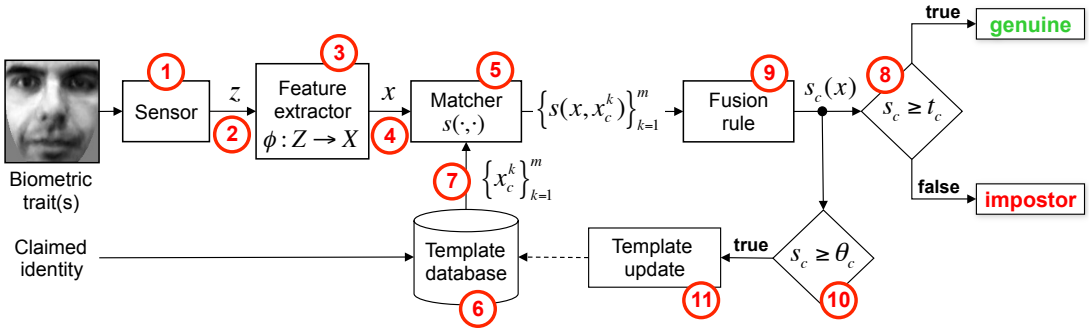


Figure 3.1: Architecture of a biometric verification system and corresponding attack points, highlighted with red circled numbers. During verification, the image $\mathbf{z} \in \mathcal{Z}$ (*e.g.*, a face image) acquired by the sensor is processed by a feature extractor $\phi : \mathcal{Z} \mapsto \mathcal{X}$ to obtain a compact representation $\mathbf{x} \in \mathcal{X}$ (*e.g.*, a graph). The templates $\{\mathbf{x}_c^k\}_{k=1}^m$ of the claimed identity c are retrieved from the template database, and compared to $\mathbf{x}$ using a matching algorithm $s : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$. The resulting scores $\{s(\mathbf{x}, \mathbf{x}_c^k)\}_{k=1}^m$ are combined by a fusion rule, producing an aggregated score $s_c(\mathbf{x})$ that expresses the degree to which $\mathbf{x}$ is likely to belong to c. The score $s_c(\mathbf{x})$ is then compared with a decision threshold t_c to decide whether the claim is genuine or impostor. If template self-update is implemented, and $s_c(\mathbf{x})$ is not lower than a self-update threshold θ_c, one of the templates in $\{\mathbf{x}_c^k\}_{k=1}^m$ is updated depending on $\mathbf{x}$, according to a given policy.

Spoofing attacks consist of fabricating a fake biometric trait to impersonate an enrolled client. They target the sensor (point 1), so they are also referred to as *direct attacks*. Current defenses are based on *liveness detection* methods, which aim to verify whether the submitted trait is "alive" or "fake" by looking at specific patterns (*e.g.*, perspiration

Table 3.1: Categorization of attacks and countermeasures for biometric systems. For each attack technique, we also report the targeted component (attack location) and the attack point(s), according to Fig. 3.1.

Attack Technique	Attack Location	Attack Point(s)	Defense
Spoofing	Sensor	1	Liveness Detection, Multibiometrics (Secure Fusion)
Replay	Interfaces / Channels	2, 4, 7	Encrypted Channel, Timestamp, Challenge-Response, Physical Isolation
Hill-Climbing	Interfaces / Channels	2, 4	Encrypted Channel, Timestamp, Challenge-Response, Physical Isolation, Score Quantization
Malware Infection	Modules / Algorithms	3, 5, 8-11	Secure Code, Specialized Hardware, Algorithmic Integrity
Template Theft, Substitution, and Deletion	Template Database	6	Template Encryption, Cancelable / Revokable Templates

patterns during fingerprint acquisition, or eye blinking during face verification). Multibiometric systems have been also proposed as a defense; however, to avoid spoofing them by only using a single fake trait, the matching scores coming from the different traits should be properly combined, using a *secure* score-level fusion rule [43, 44].

Replay attacks can be staged at interfaces between modules by replaying a stolen image of the biometric trait of the targeted client to the feature extractor (point 2), or directly the corresponding feature values to the matcher (point 4). An attacker may even replay a signal to replace the features of a given template of the claimed identity (point 7). This attack can be clearly staged if the corresponding communication channels are *insecure*, but also over encrypted channels, as the encrypted signal can be stolen and replayed into the channel directly. This can be avoided by encrypting a timestamp into the signal, or using challenge-response mechanisms. Another possible countermeasure is *physical isolation*, to avoid sending data over insecure channels (*e.g.*, the Internet) subject to man-in-the-middle attacks. A popular example of physical isolation is the use of smart cards performing match-on-card operations. However, this technique has its own disadvantages, including limitations in terms of computational resources and memory, and the fact that the user should always use the smart card to be authenticated [33].

Hill-climbing attacks, similarly to replay ones, affect insecure communication channels between modules, and, in particular, point 2 and 4 in Fig. 3.1. Their goal is to reconstruct a template image by iteratively sending a bunch of slightly perturbed images to the feature extractor (point 2), or their features to the matcher (point 4), and retaining the one that maximizes the matching score $s_c(\mathbf{x})$, where $\mathbf{x}$ is the current image (or set of features) submitted by the attacker. In practice, it is a gradient-ascent technique that approximates the gradient of $s_c(\mathbf{x})$ numerically. In this case, the attacker is assumed

to be able to observe $s_c(\mathbf{x})$ for any queried image, which may only be feasible if the system provides (or leaks) such information. In fact, besides the aforementioned channel protection schemes, an additional defense mechanism consists of quantizing the matching score to provide less accurate information to the attacker. However, attacks based on more sophisticated black-box optimization techniques, suited to quantized objective functions, can also be considered to make these countermeasures ineffective [45].

Malware Infection. The algorithmic implementations of the software modules (points 3, 5, 8–11) may exhibit vulnerabilities that can be exploited by a skilled attacker through well-known hacking techniques (*e.g.*, buffer overflow), to install malicious software, *i.e.*, *malware*, including worms, trojan horses, *etc.* This threat can be avoided or mitigated by exploiting well-known programming practices, like secure code programming, or using specialized hardware to perform some critical operations [33]. A secure programming practice is to check algorithmic integrity, *i.e.*, that each algorithm and function correctly handles any input parameter and never shows any unexpected behavior. For instance, if the matching algorithm expects a vector $\mathbf{x} \in \mathbb{R}^d$ as input, and instead receives an input with a different format, is it going to crash or provide anyway an output? In the latter case, how is such an output handled by the subsequent modules? Does it lead to accepting by error the given claim as genuine or not?

Template Theft, Substitution, and Deletion attacks target the template database (point 6). If templates are not protected properly, one may be able to steal them, and use them to create a spoof (*i.e.*, a fake template), to perform a replay attack, or to impersonate the targeted client on a different system and perform other operations, *e.g.*, searching on protected databases (function creep) [33]. Another possibility is to replace a template to impersonate a client without requiring any sophisticated attack as spoofing or replay; *e.g.*, an attacker may add his own fingerprint template to the set of templates belonging to another client. Additionally, templates of a given client can be deleted to cause a denial of service, *i.e.*, to avoid the targeted client to be recognized successfully. Countermeasures include template encryption, and also the use of cancelable / revokable templates, which can be used only on a specific system, and reissued if stolen. The idea is to encode the templates using a key or pin code that can be changed to re-enroll the user and create a novel, different encrypted template [33].

3.2 Biometric System Security reviews from the adversarial perspective

We analyse here biometric system security in terms of the previously-discussed framework, by making assumptions on the adversary's goal, knowledge, capability, and attack strategy, suited to biometric applications. Our aim is threefold: (*i*) to provide a well-structured categorization of the vulnerabilities of biometric systems and of the corresponding attacks, also through the definition of different, pertinent attack scenarios; (*ii*) to provide a formal

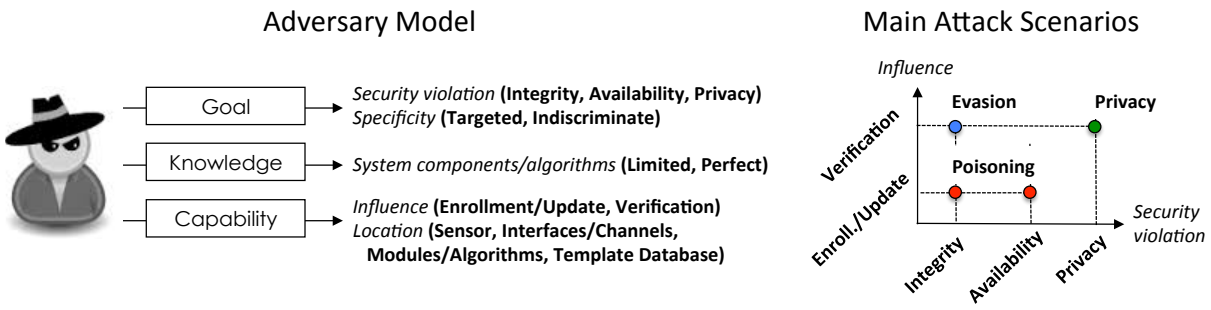


Figure 3.2: A conceptual representation of the adversary model and of the main attack scenarios (given in terms of the corresponding security violation and attack specificity) according to the framework.

characterization of existing attacks within the framework, and envision more sophisticated and effective attack strategies; and (*iii*) to identify suitable countermeasures and defenses inspired by previous work on adversarial machine learning.

Adversary's Goal. It is defined in terms of *security violation* and *attack specificity*. Biometric system security can be violated by an attacker that aims at impersonating a genuine user (**integrity** violation), at compromising the template galleries of genuine users to deny them access to the system, causing a denial of service (**availability** violation), or at violating the privacy of genuine users, *e.g.*, by inferring their templates through a hill-climbing attack (**privacy** violation). The attack specificity can be **targeted**, if the attack targets a specific set of clients, or **indiscriminate**, if any client may be affected.

Adversary's Knowledge. We define it by leveraging on the definition of the attack surface of biometric systems given in the previous section, by making specific assumptions on what the attacker knows of the system components and how they work. According to Fig. 3.1 and Table 3.1, the attacker may know: (*i*) the kind of **sensor** used (point 1), *e.g.*, an optical or capacitive fingerprint sensor; (*ii*) which **interfaces / channels** are used to implement connections (points 2, 4, 7), *e.g.*, if an insecure channel over the Internet is used to send the acquired images to the feature extractor (point 2); (*iii*) how the **modules / algorithms** work, and whether they are vulnerable or not (points 3, 5, 8-11), in particular, the feature mapping ϕ (point 3), the matching algorithm s (point 5), the decision threshold t_c (point 8), the fusion rule s_c (point 9), and, if template update is implemented, the self-update threshold θ_c (point 10) and the template update policy (point 11); (*iv*) some of the templates stored in the **template database** (point 6). The attacker may also be able to collect images of the same biometric traits using other techniques; *e.g.*, acquiring latent fingerprints, or collecting face images of the targeted clients from social networks. From a machine learning perspective, this amounts to having different levels of knowledge of the classifier's training data. In practice, it is worth noting that attackers typically have limited knowledge of the sensors and algorithms used, of the users' templates, and of any other system components (*e.g.*, communication channels, template encryption schemes,

etc.). Several previous works have however considered vulnerabilities of biometric systems without clearly pointing out the underlying assumptions on the adversary's knowledge required to perform the corresponding attack. Under the framework, such assumptions become clearly explicit.

Adversary's Capability. It can be also defined in terms of the *attack location*: (*i*) the sensor (point 1); (*ii*) interfaces / channels (2,4,7); (*iii*) the internals or even the output of modules and algorithms (3,5,8–11), *e.g.*, through malware infection attacks; and (*iv*) the template database (6). In addition, one has to define the *attack influence*, *i.e.*, the capability of manipulating the input data (*e.g.*, using fake biometric traits), and how such data may be used to update the system (*e.g.*, in adaptive biometric systems the attacker can produce spoofing attacks that can subsequently poison the clients' templates [24,36]). Accordingly, the attack can influence only **verification**, or also **enrollment / update**.

Attack Strategy. According to the adversary's goal (generally expressed in terms of an objective function $g(a;\theta)$), knowledge (given in terms of the parameter vector $\theta \in \Theta$) and capability (which defines the feasible set of attack strategies $a \in \mathcal{A}$), an optimal attack strategy can be defined to implement the attack, as explained before (see Eq. 2.1). For instance, assume that the attacker aims to impersonate an enrolled client (integrity targeted attack), and she is only able to acquire a latent fingerprint of the client, without having any other knowledge of the system components and algorithms. Then, the corresponding optimal attack strategy amounts to fabricating a fake fingerprint that is as similar as possible to the latent one, and using it to perform a spoofing attack. In this case, $g(a;\theta)$ can be regarded as a measure of the similarity between the fake and the latent fingerprint, as θ only contains information related to the latent fingerprint, and a corresponds to the fake fingerprint. A more skilled attacker may however also know the matching algorithm s and the fusion rule s_c used by the system, and may be able to collect more than a single fingerprint image of the targeted client. As an application example of the framework, we will show that, under this setting, more sophisticated and effective spoofing attacks can be fabricated. As another application example, we will also consider a poisoning attack against an adaptive face verification system, and propose a novel countermeasure based on the *sanitization* of the client's templates.

In the following, we define a set of representative attack scenarios to categorize known attacks according to the framework. The framework and the considered attack scenarios are also represented in Fig. 3.2.

3.3 Categorization of Biometric Attack Scenarios

Previous work has categorized attacks to biometric systems and countermeasures simply in terms of the attack points of Fig. 3.1 (*e.g.*, spoofing attacks to the sensor, countered by

Table 3.2: Examples of categorization of previous work on biometric security according to the three main attack scenarios defined in adversarial machine learning: evasion, poisoning, and privacy attacks.

	Goal		**Knowledge**	**Capability**		**Attack Strategy**
	Violation	*Specificity*		*Influence*	*Location*	
Evasion attacks						
Matsumoto *et al.* [47], Rodrigues *et al.* [43], Johnson *et al.* [46]	Integrity	Targeted, Indiscriminate	Perfect (consensual fake)	Verification	Sensor	Spoofing
Poisoning attacks						
Biggio *et al.* [24,36]	Integrity, Availability	Targeted, Indiscriminate	Perfect, Limited (unknown templates)	Enrollment / Update	Sensor	Spoofing (face)
Privacy attacks						
Adler [45], Galbally *et al.* [48], Martinez *et al.* [51]	Privacy	Targeted, Indiscriminate	Limited (unknown templates)	Verification	Matcher	Hill-climbing

liveness detection techniques, and attacks to a compromised channel, countered by channel encryption). Instead, looking at them from the broader perspective opened by the general framework [37, 38], allows us to identify three main *attack scenarios*, described in the following, in which the attacks and countermeasures discussed in the previous sections play different roles. A few examples of known attack and defenses are categorized in Table 3.2 in terms of these attack scenarios. This also opens the way both to identify novel, more sophisticated attacks against biometric systems, and to adapt the corresponding countermeasures from the adversarial learning field.

3.3.1 Evasion

The **goal** of this attack scenario is to impersonate a client (integrity, targeted/ indiscriminate attack). To this end, **knowledge** of the client's biometric trait is required, *e.g.*, to create a fake trait or to carry out a replay attack. Attacks exploiting perfect knowledge of the targeted client's biometric trait include the so-called consensual method (in which the targeted client voluntarily provides the required biometric trait to the attacker), and template stealing. Conversely, exploiting a latent fingerprint is an example of limited knowledge, since the attacker may only partially know or observe the required biometric trait. A limited knowledge about the rest of the biometric system can also be sufficient. In this scenario the **capability** of the attacker consists of manipulating data during the verification step, whereas no influence on the enrollment/update step is assumed (in particular, she can not access the template database). Most frequently, the attack strategy corresponding to an evasion attack consists of submitting a fake trait (spoof) to the sensor (point 1), or of replaying the acquired image (point 2) into the system. In rarer

cases (disregarded here), the biometric system can be infected by a malware, which allows the attacker to arbitrarily manipulate the functionality or the output of any system component.

3.3.2 Poisoning

This non-trivial attack scenario has been originally defined in the context of adaptive biometric systems in previous work [24,36], inspired by the adversarial learning framework, and by work on poisoning learning algorithms [17,27,37,38]. The **goal** of poisoning attacks can be either an integrity or availability security violation; it can be either targeted to a specific client, or indiscriminate (see below). The adversary's **knowledge** can be perfect or limited, depending on whether each of the system's components is known exactly to the attacker. More precisely, the attacker may have perfect (or limited) knowledge of each of the components discussed in Fig. 3.1, including the targeted clients' templates, the matching algorithm, the template update algorithm, and the decision and update thresholds. The attacker's **capability** consists of modifying the template database, either by directly manipulating it (*e.g.*, through malware infection), or, more realistically, by submitting fake traits that are erroneously used to update the template gallery of a given client. In terms of security violation, an integrity violation thus amounts to replacing a victim's template with an attacker's template, or to adding an attacker's template in the victim's gallery. This indeed allows the attacker to impersonate the victim without using any further spoofing or replay attack, but directly using her own biometric trait. The goal of an availability violation is to cause a denial of service, instead, by replacing or compromising the majority of templates in the victim's gallery. This will indeed deny the victim to correctly access the system. Under this setting, the **attack strategy** amounts to compromising the template gallery either by introducing an attacker's template in the victim's gallery (*i.e.*, integrity violation), or by compromising the maximum number of victim's templates (*i.e.*, availability violation). If the template database can not be compromised directly, the attacker can produce a well-crafted sequence of fake traits to gradually drift the victim's template gallery towards the desired set of templates, while minimizing the number of fake traits required to complete the attack. An example of such an attack will be given below.

3.3.3 Privacy

In this case, the **goal** is to retrieve confidential information (*i.e.*, one or more templates) about either a given set of clients (targeted attack) or about any client (indiscriminate attack). This is typically a preliminary step before performing another kind of attack (evasion or poisoning), when no simpler way to retrieve information on the victims' templates exists (*e.g.*, acquiring a face image through a social network, or a latent fingerprint). To this end, the attacker can gain **knowledge** from the system's feedback, *e.g.*, the outcome

of the verification decision (either accept or reject), or the score value $s_c(\mathbf{x})$ (as in hill-climbing attacks). In common settings (*i.e.*, disregarding cases like malware infection), the **capability** consists of sending a number of query images through a remote channel and observing the available feedback; *e.g.*, if the sensor and the matcher are remotely operating, and interconnected through the Internet, an attacker may perform a man-in-the-middle attack, and send replayed images through the channel. It is thus clear that the **attack strategy** in this case corresponds to an hill-climbing attack.

3.4 Secure-by-Design Biometric Systems

The considered adversary model can be exploited not only to provide a different categorization of known defense mechanisms for biometric recognition systems, but also to identify novel countermeasures among those proposed in adversarial learning, that can help countering attacks against machine learning and pattern recognition algorithms used in biometric systems. We discuss them below, with reference to the three aforementioned attack scenarios.

3.4.1 Countering Evasion

In this scenario, the main attack strategies involve spoofing and replay. As reported in Table 3.1, the pertinent defenses thus are: liveness detection, multibiometric systems with secure score-level fusion rules, encrypted channels and timestamp / challenge-response schemes, and physical isolation (*e.g.*, match-on-smart-cards). Novel defense mechanisms can also be devised, inspired by the adversarial machine learning field. In particular, to counter evasion attacks, one can consider *secure learning* techniques. They consist of modifying existing learning algorithms, and developing novel ones, that explicitly take into account a specific kind of adversarial data manipulation. They follow the paradigm of security by design, which advocates that a system should be designed from the ground up to be secure. In the context of biometric systems, secure learning techniques can be exploited to design trainable score-level fusion rules, such as those based on game theory, or on the framework of learning with invariances [9, 32, 52, 53]. Investigating this issue would be an interesting research direction for future work.

3.4.2 Countering Poisoning

Spoofing and replay are the main attack strategies also under this scenario and, thus, the same defenses listed in Table 3.1 can be also exploited in this case. In addition, other countermeasures can be considered, among those proposed in adversarial learning, to improve the *security of the training phase* in the presence of poisoning, which may occur when the system is retrained on data collected during operation [19, 54]. These

include *secure learning* (see above) and *data sanitization*. In a biometric system, the latter consists of detecting outlying template updates that may compromise the template gallery of a given client, *e.g.*, by adding an impostor's template to the targeted client's gallery, or by replacing some of the client's templates. We will give a concrete example of a novel defense based on *template sanitization* in the next section. We point out that these additional defenses can be considered complementary to those listed in Table 3.1, like liveness detection and channel encryption.

3.4.3 Preserving Privacy

Known defenses that can be exploited against attacks targeting the template database are mainly based on template encryption schemes [33] (Table 3.1). Score quantization has been also proposed to counter hill-climbing attacks, but it has already been shown to be ineffective [45]. Moreover, attacks proposed in adversarial learning have already been capable of reverse engineering the classifier by only exploiting only feedback on its decisions [16, 55]; thus, even by only looking at genuine or impostor classifications, an attacker may be able to successfully perform an hill-climbing attack. Among the proposed countermeasures that have not been yet considered for biometric systems, it would be worth investigating in future work the ones based on *randomization* and *disinformation*. They follow the paradigm of *security by obscurity*, aiming to improve system security by hiding information to the attacker. They have been suggested in adversarial learning to counter reverse-engineering attacks. This can be achieved by denying access to the actual classifier or training data, and randomizing the classifier's output to give imperfect feedback to the attacker [17, 27, 37, 38, 56].

3.5 Application Examples

We consider here two application examples that we studied by applying the framework to biometric recognition systems, respectively related to the development of sophisticated *spoofing* and *poisoning* attacks against face verification systems. For the simulations we use the DIEE dataset (Appendix A).

3.5.1 Improved Face Spoofing from Multiple Faces

Let us assume we are given a face verification system that authenticates clients by matching the acquired face image against the template gallery of the claimed identity (consisting of n images acquired during enrollment), and then thresholding the corresponding average score. According to the architecture depicted in Fig. 3.1, we assume that our system maps the submitted face image $\mathbf{z} \in \mathcal{Z}$ onto a reduced vector space $\mathcal{X}$ using principal component analysis (PCA), and computes the matching score for client c as $s_c(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} s(\mathbf{x}, \mathbf{x}_i)$,

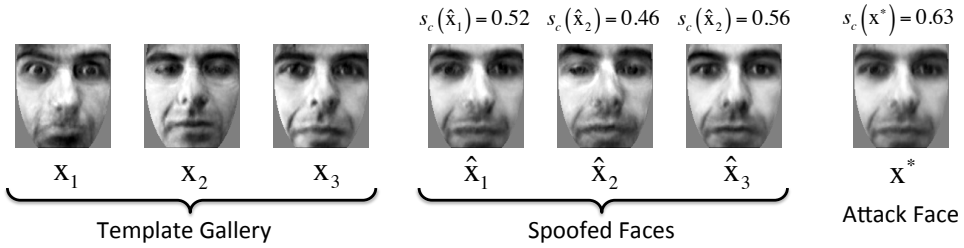


Figure 3.3: Face spoofing from multiple images. The client's templates $\{\mathbf{x}_i\}_{i=1}^3$, the spoofed faces $\{\hat{\mathbf{x}}_j\}_{j=1}^3$, and the final attack face $\mathbf{x}^*$ (obtained solving Prob. 3.1) are shown, along with the corresponding s_c values.

where $s(\mathbf{x}, \mathbf{x}_i) = \exp\{-\|\mathbf{x} - \mathbf{x}_i\|\}$, and $\mathbf{x}_i$ is the *i-th* template of the claimed identity. We further assume the attack scenario detailed below.

Adversary's Goal. The attacker aims to impersonate a targeted client (*integrity, targeted* attack).

Adversary's Knowledge. She is assumed to know: (i) the feature extraction algorithm, (ii) the matching algorithm s, (iii) the fusion rule s_c, and (iv) a set of n face images $\{\hat{\mathbf{x}}_j\}_{j=1}^n$ of the targeted client, different from those in the client's template gallery (*e.g.*, potentially collected from a social network).

Adversary's Capability. She can only submit printed photos of faces to the sensor, during verification. Regarding the type of attack, the adversary makes a dense (ℓ_2) attack. She makes small changes in many pixel, potentially all of them, in the images.

Adversary's Strategy. Under these assumptions, the attacker can approximate the score $s_c(\mathbf{x})$ computed by the targeted system for the claimed identity c using the collected face images of the victim, *i.e.*, she can compute an estimate $\hat{s}_c(\mathbf{x}) = \frac{1}{n}\sum_{j=1}^n s(\mathbf{x}, \hat{\mathbf{x}}_j)$. Accordingly, the optimal attack strategy is given by:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}} \hat{s}_c(\mathbf{x}) = \frac{1}{n}\sum_{j=1}^{n} s(\mathbf{x}, \hat{\mathbf{x}}_j)\,, \tag{3.1}$$

where $\hat{s}_c$ is the attacker's goal function $g(a, \theta)$, and $\mathbf{x}^*$ is the attack sample that maximizes the objective in the PCA-induced feature space. The above problem can be solved by a simple gradient-ascent algorithm, and the resulting attack sample $\mathbf{x}^*$ can then be projected back onto the space of face images $\mathcal{Z}$ (where each feature corresponds to the gray-level value of a pixel) by inverting the PCA-induced mapping. This is also possible if more sophisticated matching algorithms and feature representations are used, using well-crafted heuristics. We refer the reader to [36] for further details.

An example of this improved spoofing technique on a simple case involving $n = 3$ templates is shown in Fig. 3.3, where we also report the values of s_c for each of the

client's face images $\{\hat{\mathbf{x}}_j\}_{j=1}^{n}$. It can be seen that the final attack face $\mathbf{x}^*$ yields a higher probability (*i.e.*, s_c value) of successfully impersonating the victim, than using any of the available images $\{\hat{\mathbf{x}}_j\}_{j=1}^{3}$.

3.5.2 Poisoning Biometric Systems that Learn from Examples

We report now a different application example focusing on a *poisoning* attack against an *adaptive* face recognition system, and on the development of a corresponding defense. Recent works ([24,36]) have shown that an attacker may exploit the system's adaptation mechanism to compromise the templates of a given client by presenting a well-crafted sequence of fake faces to the camera, with the goal of denying him access to the system. At the same time, if the attacker replaces the targeted client's templates with her templates, she may also impersonate the client without presenting any fake trait to the sensor.

The face verification system considered in this example, as in [24, 36], authenticates clients based on matching the acquired face image with a stored *average* template, referred to as *centroid*. For each client, the centroid is updated using the self-update algorithm: if the submitted face image is similar enough to the centroid, the latter is updated incorporating the new image into the computation of the average face image. As in the previous case, we consider a PCA-based mapping to map face images from $\mathcal{Z}$ onto a reduced vector space $\mathcal{X}$. The matching score for client c is computed here as $s_c(\mathbf{x}) = \exp\{-\|\mathbf{x} - \mathbf{x}_c\|^2\}$, where $\mathbf{x}_c$ is the client's centroid. The centroid $\mathbf{x}_c$ is initially computed as the average of n templates and, when $s_c(\mathbf{x}) \geq \theta_c$, updated as $\mathbf{x}'_c = \mathbf{x}_c + \frac{1}{n}(\mathbf{x} - \mathbf{x}_c)$, *i.e.*, slightly drifted towards $\mathbf{x}$. Accordingly, in the PCA-based feature space, $\mathbf{x}_c$ is updated if the acquired image $\mathbf{x}$ is within an hypersphere centered on $\mathbf{x}_c$, with radius d_c dependent on the update threshold θ_c. The complete attack scenario is given below.

Adversary's goal. It is that of replacing the centroid $\mathbf{x}_c$ of a given client with an attacker's template $\mathbf{x}_a$, both to deny access to client c, and to allow the attacker to impersonate c using her own face (*i.e.*, a *targeted* attack violating both system *availability* and *integrity*).

Adversary's Knowledge. The attacker is assumed to know: (*i*) the feature extraction algorithm; (*ii*) the matching algorithm; (*iii*) the template update algorithm; and (*iv*) the decision and self-update threshold. In the case of *perfect* knowledge, she also knows the centroid $\mathbf{x}_c$ of the targeted client c, while when *limited* knowledge is considered, only a good estimate of $\mathbf{x}_c$ is available to the attacker, *e.g.*, a frontal face image of the victim collected from a social network.

Adversary's Capability. The attacker can modify the template database by presenting fake faces at the sensor that enable template self-update. Even here, as in the improved spoofing example, the adversary makes an ell_2 attack. The attack influence is thus over the enrollment / update phase.

Attack Strategy. Under these assumptions, the shortest sequence of fake traits required to replace the victim's centroid $\mathbf{x}_c$ with that of the attacker $\mathbf{x}_a$ can be found by solving the

following optimization problem, for each sample $\mathbf{x}$ in the attack sequence: $\min_{\mathbf{x}} \|\mathbf{x}-\mathbf{x}_a\|^2$, subject to the update condition $\|\mathbf{x}-\mathbf{x}_c\| \leq d_c$ (see Fig. 3.4-*upper-left*). At each iteration, this amounts to finding the closest attack sample to $\mathbf{x}_a$ that enables update of $\mathbf{x}_c$. The solution is simply given as $\mathbf{x} = \mathbf{x}_c + d_c\vec{a}$, where $\vec{a} = \frac{\mathbf{x}_a-\mathbf{x}_c}{\|\mathbf{x}_a-\mathbf{x}_c\|}$ is the so-called *attack direction*. In practice, each attack sample $\mathbf{x}$ is found at the intersection between the hypersphere corresponding to the update condition, and the line connecting $\mathbf{x}_a$ and $\mathbf{x}_c$. As in the previous example, the face images for the attack sequence can be obtained by projecting the attack samples from the PCA-induced space $\mathcal{X}$ onto the space of face images $\mathcal{Z}$. Then, the attacker can fabricate the corresponding fake faces (*e.g.*, by printing them on paper), and present them in the right order to the sensor. An example of how the victim's centroid is gradually updated by the corresponding sequence of attack faces, under the assumption of *perfect knowledge*, and for $n = 5$, is given in Fig. 3.4.*bottom*. In Fig. 3.4-*upper-right*, we show how the Genuine and False Acceptance Rate (GAR and FAR, other names to call the True and False Negative Rates) vary as the attack proceeds, under perfect and limited knowledge of the victim's template. Note how the probability of authenticating the attacker (without presenting any fake face) as the victim (*i.e.*, the FAR) increases, while the probability of correctly authenticating the victim as a genuine user (*i.e.*, the GAR) decreases, since the victim's template is gradually *morphed* towards the attacker's face during the attack progress. Results for the perfect and limited knowledge attacks are similar, despite the latter case requires more iterations (*i.e.*, submitting more fake faces) to compensate the lack of knowledge of the victim's template. The exact number of iterations required to complete both attacks can also be analytically computed [36].

3.5.2.1 Template Sanitization

In the remainder of this section, we present a novel countermeasure based on the idea of *sanitizing* the template gallery (*i.e.*, identifying *anomalous* template updates), inspired by the countermeasures proposed in adversarial machine learning against poisoning attacks [17, 27, 37, 38, 54].

The underlying idea is to analyze whether the sequence of the most recent k updated centroids $\mathbf{x}_c^{(i-k)}, \ldots, \mathbf{x}_c^{(i)}$ (where i denotes the current iteration) falls within a given region of the feature space, called *sanitization hypersphere* (see Fig. 3.5.*left*). If the current centroid $\mathbf{x}_c^{(i)}$ falls within the sanitization hypersphere, *i.e.*, if $\|\mathbf{x}_c^{(i)} - \mathbf{x}_c^{(i-k)}\| \leq d_s$, then the center of the sanitization hypersphere is updated to the next centroid in the sequence, *i.e.*, $\mathbf{x}_c^{(i-k+1)}$; otherwise, the current center of the sanitization hypersphere $\mathbf{x}_c^{(i-k)}$ is restored as the *current* centroid. In this case, an alert may be also (or alternatively) raised to the system administrator to report the *anomalous* update. The rationale of this approach is to identity sequences of centroid updates that consistently drift the centroid towards a given, biased direction, within a small number of iterations (as it happens in the presence of a poisoning attack), assuming that *genuine* updates exhibit a different (*i.e.*, less biased and more random) behavior. The parameter k and the hypersphere radius d_s of the

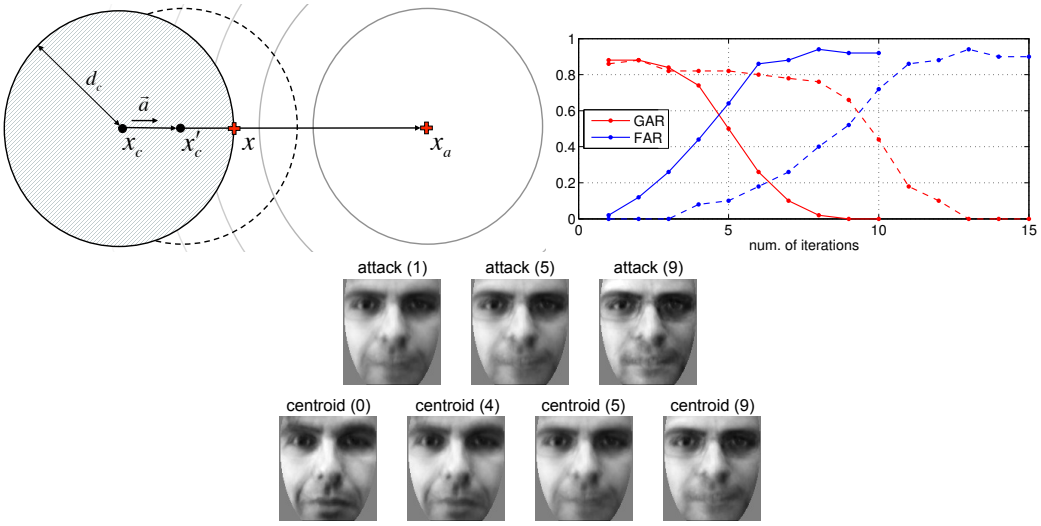


Figure 3.4: (*upper-left*): Poisoning attack with perfect knowledge. The circles centered on $\mathbf{x}_a$ represent the objective function $||\mathbf{x} - \mathbf{x}_a||$, minimized by the attack point $\mathbf{x}$ on the feasible domain $\|\mathbf{x} - \mathbf{x}_c\| \leq d_c$. The updated centroid $\mathbf{x}'_c$ and the feasible domain for the next attack iteration are also shown. (*upper-right*) GAR and FAR for poisoning with perfect (solid lines) and limited (dashed lines) knowledge, at different iterations. (*bottom*) Attack samples and victim's centroids for poisoning with perfect knowledge, at different iterations.

proposed approach should thus be chosen such that $d_s \leq k\frac{d_c}{n}$, otherwise poisoning attacks will not be detected, as they drift the centroid of an amount equal to $\frac{d_c}{n}$ at each iteration. In Fig. 3.5.*right*, we report an example using the same attacker and victim pair considered in the previous case. Initially, we simulate a number of random accesses to the system, including genuine and impostor attempts, which do not significantly affect GAR and FAR. Then, the attacker launches a poisoning attack, and, as in the previous case, the GAR decreases and the FAR increases. If template sanitization is not implemented, the attack succeeds, and system integrity and availability are compromised. Conversely, in the presence of template sanitization, the attack is detected after four iterations, and a previous centroid is restored. This avoids the attack to succeed, preserving normal system operation and security. Although this simple countermeasure can be misled by a poisoning attack in which the attack samples are closer to the current centroid (instead of lying exactly at the boundary of the feasible domain), this would require the attacker to perform a significantly higher number of iterations to complete the attack. We can thus conclude that the proposed sanitization technique helps improving system security.

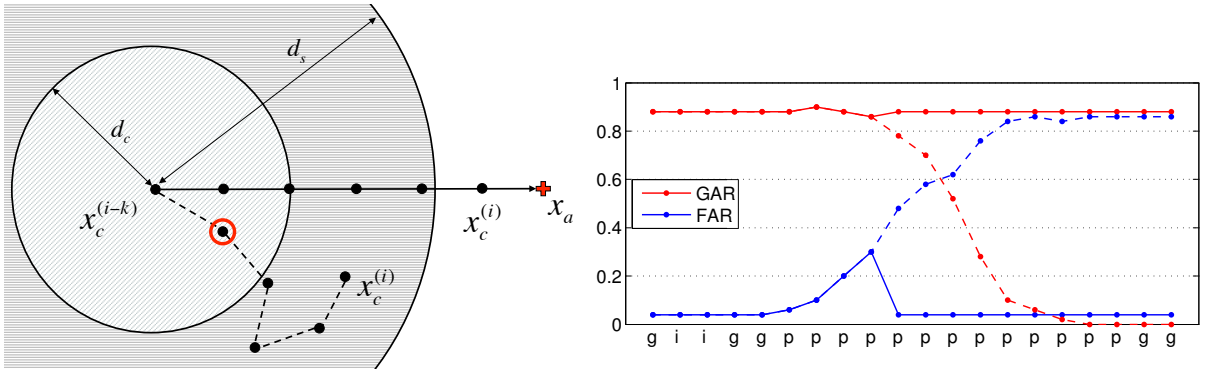


Figure 3.5: (*Left*) Template sanitization. If the current centroid $\mathbf{x}_c^{(i)}$ falls outside the sanitization hypersphere (dark grey area), as for the poisoning attack sequence (solid line), the centroid $\mathbf{x}_c^{(i-k)}$ is restored; otherwise, as for the hypothesized genuine update sequence (dashed line), the center of the sanitization hypersphere is updated to $\mathbf{x}_c^{(i-k+1)}$ (red circled point). (*Right*) GAR and FAR values in the presence (solid lines) and in the absence (dashed lines) of template sanitization, after different centroid updates, including genuine ('g') and impostor ('i') attempts, and poisoning attacks ('p') with perfect knowledge.

Chapter 4

Secure Learning against Evasion Attacks

As we have seen previously, learning algorithms have been originally designed by assuming that training and testing samples are drawn from the same distribution. Such assumption is clearly violated when attackers manipulate the input data either at training or testing time. From a very general theoretical viewpoint, this means that the class-conditional distribution of malicious samples observed at test time is different from that observed at training time. In the case of evasion attacks, only malicious data at test time is affected. Thus, by denoting the input samples with $\mathbf{x} \in \mathcal{X}$ (in a continuous feature space), and their class labels with $y \in \{-1, +1\}$ (respectively, for legitimate and malicious samples), this can be formalized as:

$$p_{\text{ts}}(\mathbf{x}'|y = +1) = \int_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}'|\mathbf{x}, y = +1) p_{\text{tr}}(\mathbf{x}|y = +1) d\mathbf{x}\,,$$

where $\mathbf{x}' = a(\mathbf{x})$, being $a : \mathcal{X} \mapsto \mathcal{X}$ a manipulation function representing the attack strategy, *i.e.*, defining how the attacker manipulates the initial malicious data $\mathbf{x}$ as $\mathbf{x}'$ to evade detection at test time. The term $p(\mathbf{x}'|\mathbf{x}, y = +1)$ characterizes the probability of having the initial malicious sample $\mathbf{x}$ modified as $\mathbf{x}'$, and p_{ts} and p_{tr} respectively denote the testing and training class-conditional distributions of the malicious samples. This straightforward model, albeit being not directly useful to design secure learning algorithms (see, *e.g.*, [57]), clarifies the connection between the manipulation function a and the *adversarial drift* that it induces in the probability distribution of malicious samples. To account for this potential, adversarial drift between training and testing distributions, adversary-aware learning algorithms have been developed, based on secure optimization, and probabilistic and game-theoretical models (see, *e.g.*, [32, 52, 53]). The underlying idea of these algorithms is that of incorporating knowledge of the potential adversarial data manipulations into the learning phase, either by simulating such manipulations at training time, through the definition of suitable manipulation functions $a(\mathbf{x})$, or by modeling

the distribution drift directly (in generative models). In practice, both options reflect a similar effect, *i.e.*, an adversarial shift of the malicious distribution, as witnessed by the aforementioned probability model. The only difference is the level at which assumptions on the attacker model are made, *i.e.*, either at the level of each malicious sample, or at the higher level of their global probability distribution. Clearly, making assumptions at the sample level allows one to more finely define the potential adversarial data manipulations, which can be advantageous when application-specific constraints on data manipulation can be accounted for. On the other hand, secure learning techniques based on this approach tends to exhibit a much higher training complexity, especially in terms of computational time and space. For example, game-theoretical approaches (as that advocated in [32]) require simulating the attacks during training, and iteratively adjust the classification function. Another issue is that such approaches require specific assumptions to be met, to guarantee existence of a unique equilibrium in the game. For instance, in [32] the objective function of the attacker is required to be twice differentiable, and this is clearly not the case for sparse attacks (since the ℓ_1 distance representing the attacker's cost to modify data is not differentiable). This means in turn that this approach can not deal with sparse attacks, at least in principle. The underling idea of the work in [52, 53] is instead to consider a worst-case loss suited to sparse attacks in which features can be set to their maximum/minimum values. In this case too, training complexity becomes higher with respect to the non-secure versions of the same algorithms. The huge computational necessity, in terms of calculus and memory space, introduced by these works is one of the main factors that hinders the adoption of these algorithms in practice, along with the difficulty of meeting some theoretical requirements. In this chapter, we aim to overcome these limitations by developing secure classifiers against evasion attacks that are not computationally more demanding than their non-secure counterparts.

4.1 Solving the evasion problem

Firstly, we need to find an evasion algorithm that would be efficient and effective. Depending on the kind of decision function and distance metric, Problem (2.2)-(2.3) can be casted in terms of a linear or a non-linear programming problem.[1] For linear classifiers, the global minimum can be found, either in the case of ℓ_1 or ℓ_2 constraints. For non-linear $g(\mathbf{x})$, the solution is typically found at a local minimum of the objective function. Problem (2.2)-(2.3) can be solved with standard solvers in both cases, although they may not be very efficient, as they do not exploit specific knowledge about the evasion problem (*e.g.*, sparsity of the solution, compact domain, etc.). We thus devise an ad-hoc solver based on exploring a descent direction aligned with the gradient of $g(\mathbf{x})$ by means of a bisect line search. Its basic structure is given as Algorithm 1 [74]. To reduce the number

[1]Note that Problem (2.2)-(2.3) becomes linear only for linear classifiers and sparse evasion attacks (using the ℓ_1 distance).

Algorithm 1 Evasion Attack

Input: $\mathbf{x}$: the malicious sample; $\mathbf{x}^{(0)}$: the initial location of the attack sample; $d_{\max}$: the maximum distance between $\mathbf{x}$ and $\mathbf{x}'$; $\mathbf{x}_{\text{lb}}$, $\mathbf{x}_{\text{ub}}$: the box constraint bounds (Eq. 2.3); ϵ: a small positive constant.

Output: $\mathbf{x}'$: the evasion attack sample.

$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $t' = \arg\min_t g(\mathbf{x}^{(i-1)} - t\nabla g(\mathbf{x}^{(i-1)}))$ (line search)
  $\mathbf{x}^{(i)} \leftarrow \mathbf{x}^{(i-1)} - t'\nabla g(\mathbf{x}^{(i-1)})$
  if constraints in Eq. (2.3) are violated **then**
    Project $\mathbf{x}^{(i)}$ onto the feasible domain
  end if
until $g(\mathbf{x}^{(i)}) - g(\mathbf{x}^{(i-1)}) > \epsilon$
return $\mathbf{x}^{(i)}$

of iterations, and ensure quick convergence, we explore one feature at a time in the case of ℓ_1 attacks (starting from the more promising feature, *i.e.*, the one exhibiting the highest gradient variation), as the solution will be sparse. Conversely, we simultaneously explore all the features in the case of ℓ_2 attacks, as the solution will be likely to modify all feature values. We also minimize the number of gradient and function evaluations to further speed up our evasion algorithm; *e.g.*, we only re-compute the gradient of $g(\mathbf{x})$ when no better point is found on the descent direction under exploration. Finally, in the case of non-linear $g(\mathbf{x})$, we exploit multiple initializations for $\mathbf{x}^{(0)}$ to mitigate issues related to the presence of multiple local minima. Usually, the data have to be normalized, in order to lead on a common scale the values measured on different feature. Our algorithm allows one to solve the optimization problem considering the radius of the distance constraint differently normalized for each dimension. Moreover, it is possible to find the solution in the presence of a discrete feature space: the evading sample obtained for the continuous case is compared with the discrete neighbourhood, to chose the one that minimizes the discriminant function remaining inside the distance bound. Examples of (sparse and dense) evasion attacks against Support Vector Machines (SVMs) and Random Forests are shown in Fig. 4.1. As Random Forests have a non-differentiable discriminant function $g(\mathbf{x})$, we construct a differentiable approximation $\hat{g}(\mathbf{x})$ by learning a surrogate SVM on the same training data used to learn the random forest, but replacing the (true) training labels with the classification labels assigned by the Forest to such data. Then, the surrogate SVM can be used to find a suitable descent direction, and run the evasion attack against the random forest. Learning a surrogate (differentiable) model to solve the evasion problem should be more computationally efficient, at least in principle.

4.2 Understanding classifier security

Observing the shape of the non-linear decision functions in Fig. 4.1 (third and fourth plot), and the corresponding evasion samples, one may interestingly note that, in some cases (see, *e.g.*, the evasion sample in the third plot), to evade detection, it suffices to create a sample that is *far enough* from the known malicious samples (learned by the classifier during training), without even mimicking any legitimate sample. In some other cases (see, *e.g.*, the evasion sample in the fourth plot), the attacker is instead required to *mimic* the characteristics exhibited by the *legitimate samples*, which can be a much harder task in real-world applications. This reveals an interesting insight on the security of non-linear classifiers, as also already pointed out in recent work [40,60], *i.e.*, that decision functions that better enclose the legitimate data tend to be more secure against evasion attacks. In practice, the main vulnerability of learning algorithms relies upon the fact that, sometimes, it is possible to evade detection by creating samples which are far enough from the rest of the training data.[2] In previous work [60], this vulnerability has been referred to as a vulnerability of the classification algorithm. Conversely, if the classifier only allows evasion if the attack sample is close enough to the legitimate data, and the attacker can nevertheless construct evasion samples successfully, then the vulnerability is related to the feature representation. In fact, if a legitimate and a malicious sample become indistinguishable from each other in terms of their feature values, then the vulnerability is clearly related to the choice of the feature representation.

4.3 Security and Regularization

We clarify here the connection between regularization and input data uncertainty highlighted by the recent findings in [63–67]. In particular, Xu *et al.* [63] have considered the following *secure* optimization problem:

$$\min_{\mathbf{w},b} \max_{\mathbf{u}_1,..,\mathbf{u}_n \in \mathcal{U}} \sum_{i=1}^{n} \left(1 - y_i(\mathbf{w}^\top(\mathbf{x}_i - \mathbf{u}_i) + b)\right)_+ , \tag{4.1}$$

where $(z)_+$ is equal to $z \in \mathbb{R}$ if $z > 0$ and 0 otherwise, $\mathbf{u}_1, ..., \mathbf{u}_n \in \mathcal{U}$ define a set of bounded perturbations of the training data $\{\mathbf{x}_i, y_i\}_{i=1}^{n} \in \mathbb{R}^n \times \{-1, +1\}^n$, and the so-called *uncertainty set* $\mathcal{U}$ is defined as

$$\mathcal{U} \triangleq \{(\mathbf{u}_1, \ldots, \mathbf{u}_n) | \sum_{i=1}^{n} \|\mathbf{u}_i\|^* \leq c\} , \tag{4.2}$$

being $\|\cdot\|^*$ the dual norm of $\|\cdot\|$. By definition, a norm is connected to its dual by the equation

$$\|\mathbf{x}\|^* = \max_{\|\mathbf{z}\| \leq 1} \mathbf{z}^T \mathbf{x} , \tag{4.3}$$

[2] They have been also referred to as *blind spots* in [55], and as *adversarial examples* in recent work related to the evasion of deep learning algorithms [61, 62].

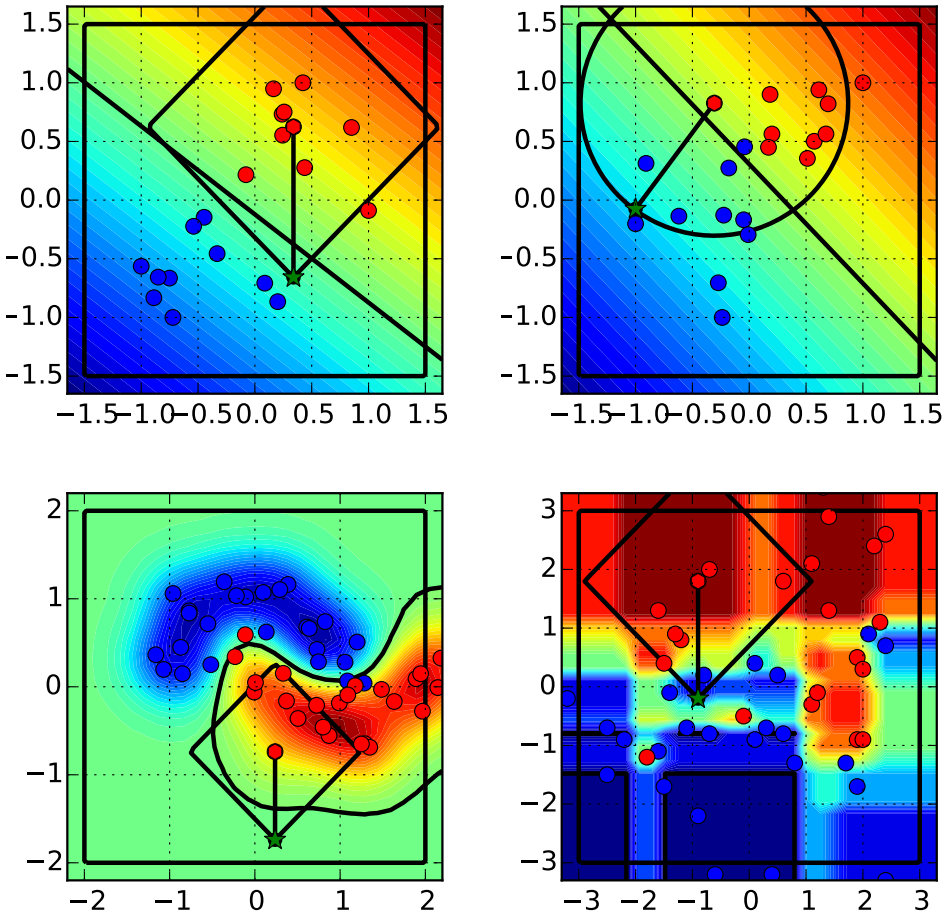


Figure 4.1: Evasion attacks against different classifiers, trained on blue (legitimate) and red (malicious) samples. A linear SVM classifier against sparse (first plot) and dense (second plot) evasion attacks, an SVM with the RBF kernel (third plot) and a random forest (fourth plot) against sparse evasion attacks. The initial malicious point $\mathbf{x}$ is found at the center of the distance constraint, while the evasion sample $\mathbf{x}^\star$ is denoted with a green star. For each classifier, $g(\mathbf{x})$ values are shown in colors, and the black line denotes the decision boundary.

where $\mathbf{x}$ and $\mathbf{z}$ are two vectors. More generally, Hölders's inequality shows that the dual of the ℓ_p norm is the ℓ_q norm, where q satisfies $\frac{1}{p} + \frac{1}{q} = 1$.

Typical examples of uncertainty sets according to the above definition include ℓ_1 and ℓ_2 balls [63, 64].

Problem (4.1) amounts to minimizing the hinge loss for a two-class classification problem under worst-case, bounded perturbations of the training samples $\mathbf{x}_i$, *i.e.*, a typical setting in secure optimization [63–67]. Under some mild assumptions easily verified in practice (including non-separability of the training data), the authors have shown that the above problem is equivalent to the following non-secure, regularized optimization problem (*cf.* Th. 3 in [63]):

$$\min_{\mathbf{w},b} \; c\|\mathbf{w}\| + \sum_{i=1}^{\mathsf{n}} \left(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\right)_+ . \tag{4.4}$$

This means that, if the ℓ_2 norm is chosen as the dual norm characterizing the uncertainty set $\mathcal{U}$, then $\mathbf{w}$ is regularized with the ℓ_2 norm, and the above problem is equivalent to a standard SVM. If input data uncertainty is modelled with the ℓ_1 norm, instead, the optimal regularizer would be the ℓ_∞ regularizer, and vice versa.[3]

Uncertainty sets and cost-sensitive learning The work by Xu *et al.* [63] only considers uncertainty sets of the same size, *i.e.*, the same perturbation is applied on both the legitimate and the malicious class. However, it is clear that, under evasion, the malicious samples are potentially affected by a stronger worst-case perturbation than legitimate data. Interestingly, in their recent work, Katsumata and Takeda [66] have shown that different uncertainty sets can be accounted for on each sample (and thus, on each class too), by simply modifying the cost of each classification error. This means that it suffices to penalize differently errors in different classes to consider uncertainty sets of different sizes. In the SVM learning algorithm, this can be simply accounted for by setting a different C value for legitimate and malicious samples. This in turn suggests that classifier security can be improved by unbalancing the costs of classification errors in different classes.

Kernelization. To conclude, note that these findings mostly hold for linear classifiers. In the case of non-linear (kernelized) classifiers, the authors have essentially repeated their analysis but considering perturbations directly in the feature space induced by the kernel function, instead of retaining them in the input space. Although this may be useful to understand how to regularize non-linear functions, the resulting perturbation in the feature/kernel space depends on the kernel mapping itself, and it is not trivial to understand how it could be modified by the kernel function. For instance, if one considers an ℓ_1 perturbation in input space, and an RBF kernel $k(\mathbf{x}, \mathbf{z}) = \exp(-\gamma\|\mathbf{x}-\mathbf{z}\|_2^2)$, the corresponding perturbation in the feature/kernel space is likely to become dense for sufficiently small γ values. Intuitively, this can be explained by the fact that a sparse modification on an input sample $\mathbf{x}$ tends to affect almost all kernel values computed

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

between $\mathbf{x}$ and the rest of the training data. This analysis is thus not very helpful in the case of evasion attacks, as the corresponding perturbations are clearly applied in the input space. As we will see in the next section, in fact, for non-linear classifiers it is not the choice of the regularization term that plays a crucial role for improving security, but rather the selection of a proper *kernel function*.

4.4 Classifier Security

We discuss here different strategies we have proposed in [73, 74, 76], that can be exploited to improve security of linear and non-linear classifiers. For linear and non-linear differentiable classifiers, our rationale is to show that the maximum variation of a classifier's discriminant function under an evasion attack can be bounded, highlighting the factors that may harm classifier security, and discussing how to limit their impact. This will give us a set of guidelines to help designing more secure learning algorithms against evasion. It is also worth remarking that, very interestingly, some of the results arising from our analysis corroborate the findings discussed in the previous section for linear classifiers. Differently, in case of non-differentiable discriminant functions, we can not apply the aforementioned considerations. So, we present a solution that follows the approach highlighted in Sect.4.2: the closure of the benign region in the feature space.

4.4.1 Linear Classifiers

Works in the adversarial machine learning literature have already investigated the security of linear classifiers to evasion attacks [10,68], suggesting the use of more evenly-distributed feature weights as a mean to improve their security. Such a solution is however based on heuristic criteria, and a clear understanding of the conditions under which it can be effective, or even optimal, is still lacking. In our works we have given a clearer explanation of this heuristic criteria and we have provided a theoretically-sound approach to devise secure linear classifiers. We start by analysing the worst-case variation of the discriminant function of a linear classifier under evasion. The discriminant function of a linear classifier is simply given as $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$. Assuming that $\mathbf{x}$ is an initial malicious sample, and $\mathbf{x}'$ the corresponding manipulated evasion sample, one yields:

$$\Delta g = g(\mathbf{x}) - g(\mathbf{x}') = \mathbf{w}^\top (\mathbf{x} - \mathbf{x}') \,. \tag{4.5}$$

Note that, from the attacker's perspective, this variation has to be maximized to increase chances of successfully evade the targeted classifier, as the attacker's strategy in Problem (2.2)-(2.3) aims to minimize $g(\mathbf{x}')$.

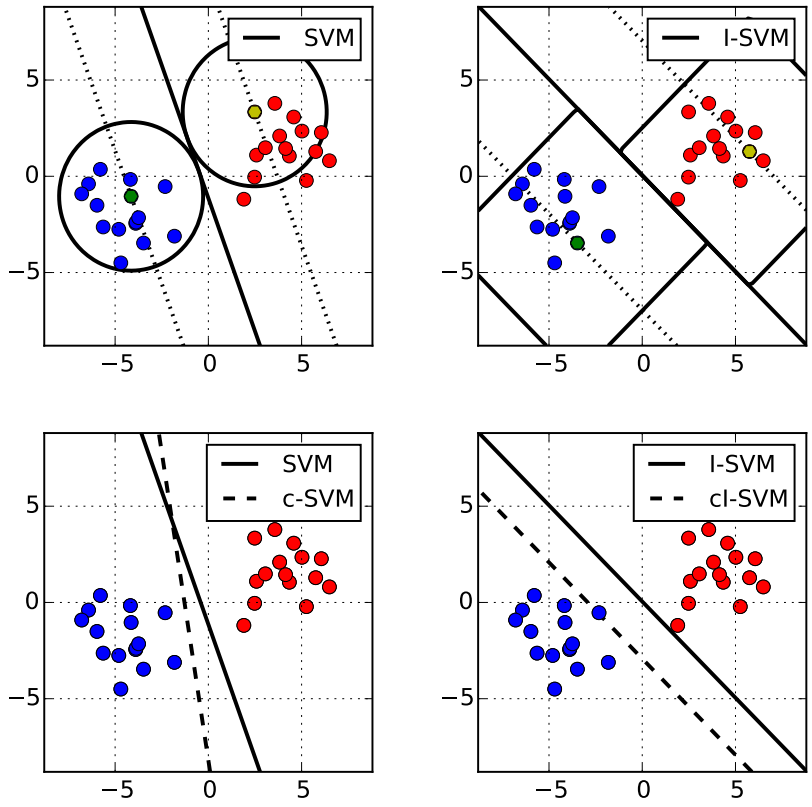


Figure 4.2: Decision boundaries for SVM (first plot), I-SVM (second plot), and their cost-sensitive versions, C-SVM (third plot) and cI-SVM (fourth plot). In the first and the second plot, we also report ℓ_2 and ℓ_1 balls over the margin support vectors, to visually clarify why the orientation of the decision hyperplane changes.

4.4.1.1 Dense Attacks

Under dense evasion attacks, instead, the worst-case increase of Δg corresponds to a linear shift of $\mathbf{x}$ towards the decision boundary (along the opposite direction to the hyperplane normal $\mathbf{w}$), *i.e.*, $\mathbf{x}' = \mathbf{x} - \|\mathbf{x} - \mathbf{x}'\|_2 \frac{\mathbf{w}}{\|\mathbf{w}\|_2}$ (see Fig. 4.1, second plot), which implies that:

$$\Delta g \leq \frac{\mathbf{w}^\top \mathbf{w}}{\|\mathbf{w}\|_2} \times \|\mathbf{x} - \mathbf{x}'\|_2 = \|\mathbf{w}\|_2 \times \|\mathbf{x} - \mathbf{x}'\|_2 \,. \tag{4.6}$$

4.4.1.2 Sparse Attacks

Under sparse evasion attacks, it is not difficult to see that Δg (from Eq. 4.5) is upper bounded by the following quantity:

$$\Delta g \leq \|\mathbf{w}\|_\infty \times \|\mathbf{x} - \mathbf{x}'\|_1 \,, \tag{4.7}$$

where we remind the reader that $\|\mathbf{w}\|_\infty = \max_{j=1,\ldots,\mathsf{d}} |w_j|$. In fact, for sparse attacks, the solution $\mathbf{x}'$ is found by modifying the features that have been assigned the highest absolute weight values (see, *e.g.*, Fig. 4.1, first plot). In the worst case, the maximum Δg is attained by modifying the most relevant feature of a quantity equal to $\|\mathbf{x} - \mathbf{x}'\|_1$.

The analysis of the worst-case Δg values for linear classifiers highlights two interesting facts. The former is that the feature values should be bounded, to bound the maximum variation of the relevant features. This is normally not a problem, if feature normalization is used, as normalization techniques often map the input samples onto a compact domain. The latter fact is that, under sparse attacks, one should bound the infinity-norm of $\mathbf{w}$, while under dense attacks, it is better to penalize its ℓ_2 norm. This means that it is better to use ℓ_∞ and ℓ_2 regularization respectively against sparse and dense evasion attacks. This novel result in the context of adversarial learning also confirms the findings by Xu *et al.* [63] related to the relationship between security and regularization of learning algorithms.

4.4.1.3 Countering Dense Attacks

Based on our previous discussion, and on the findings in [63], one should consider ℓ_2 regularization to counter ℓ_2 attacks. Furthermore, as suggested in [66], also using unbalanced classification costs may improve classifier security.

SVM. Accordingly, the SVM learning algorithm, with different values of C for each class, should guarantee a higher level of security against dense evasion attacks. It finds $\mathbf{w}$ and b by solving the following quadratic programming problem:

$$\min_{\mathbf{w},b} \quad \tfrac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{\mathsf{n}} c_i \left(1 - y_i g(\mathbf{x}_i)\right)_+ \,, \tag{4.8}$$

where $c_i = C_+$ ($c_i = C_-$) for malicious (legitimate) data.

4.4.1.4 Countering Sparse Attacks

For sparse attacks, our analysis, as that in [63, 66], suggests the use of ℓ_∞ regularization, potentially with unbalanced costs.

Infinity-norm SVM (I-SVM). We thus consider the SVM formulation, but changing the regularization term:

$$\min_{\mathbf{w},b} \quad \|\mathbf{w}\|_\infty + \sum_{i=1}^{\mathsf{n}} c_i \left(1 - y_i g(\mathbf{x}_i)\right)_+ \,, \tag{4.9}$$

where the c_i's are set as in the SVM case. Differently from the SVM learning problem, this one can be solved using a simpler linear programming approach.

Remark I. Examples of decision boundaries for the considered classifiers are shown in Fig. 4.3. Note that the effect of unbalanced classification costs tends to shift the decision boundary farther from the malicious class. Despite this may yield higher security, it may increase the fraction of misclassified legitimate samples (*i.e.*, the false positive rate). Therefore, its effectiveness as a valid defence strategy needs to be empirically assessed in detail, especially in those applications where keeping the false positive rate low is crucial.

Remark II. Another interesting observation is that the decision hyperplane of the I-SVM tends to yield more evenly-distributed weight values, *i.e.*, weights that are all equal in absolute value. This is clear from Fig. 4.3, as the hyperplane normal tends to align with a bisect line, *i.e.*, $\mathbf{w} \approxeq (1, 1)$. This is an important property for the security of linear classifiers, empirically validated in previous work [10, 68]. However, based on the interpretation of security and regularization in [63], and on our analysis, we have provided more theoretically-sound explanations behind the meaning of "evenly-distributed weights", and on how to enforce this behaviour with a proper regularizer (instead of exploiting heuristic techniques).

4.4.1.5 Security and Sparsity

There are some settings where the previous considerations are not applicable. In [73] we investigated on classifiers used for mobile and embedded systems; these applications usually demand for strict constraints on storage, processing time and energy consumption. So, the linear classifiers are the preferred choice as they provide easier-to-interpret decisions (with respect to non-linear classification methods). For instance, the widely-used SpamAssassin anti-spam filter exploits a linear classifier.[4] In this case, in presence of an ℓ_1-attack the I-SVM is unusable; a classifier with non-zero weight for each dimension of the feature space, with the increasing of the number of features used to represent the samples, requires a lot of memory and big computational requests to analyse each sample, and this conflicts with the application constraints. So, in this scenario, *sparse* weights are more desirable than evenly-distributed ones. For this reason we propose a novel *octagonal* (8gon) regularizer,[5] given as a linear (convex) combination of ℓ_1 and ℓ_∞ regularization (Fig.1.4):

$$\|\mathbf{w}\|_{8\text{gon}} = (1-\rho)\|\mathbf{w}\|_1 + \rho\|\mathbf{w}\|_\infty \,, \tag{4.10}$$

where the ℓ_1 pushes to a sparse solution, ℓ_∞ pushes to an evenly-distribute one, and $\rho \in (0, 1)$ can be increased to trade this two characteristics.

[4] See also `http://spamassassin.apache.org`.

[5] Note that octagonal regularization has been previously proposed also in [72]. However, differently from our work, the authors have used a pairwise version of the infinity norm, for the purpose of selecting (correlated) groups of features.

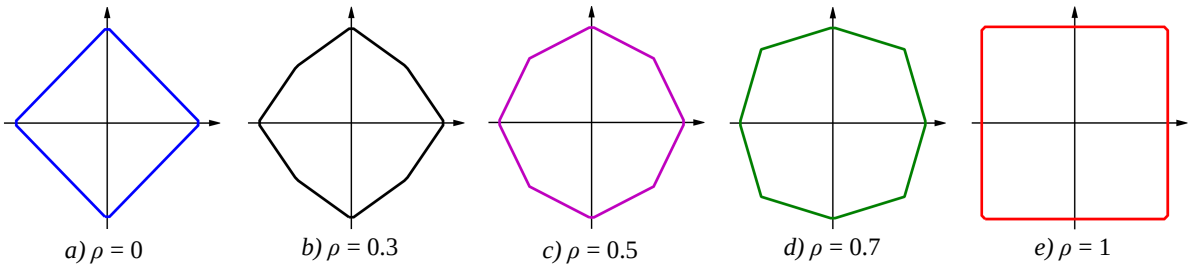


Figure 4.3: Shape of the Octagonal regularizer unit ball for different values of ρ.

4.4.2 Nonlinear Kernel Machines

Let us now analyse how to bound the maximum variation of Δg for decision functions of the form:

$$g(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\,, \tag{4.11}$$

where $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is the *kernel* function, and $\mathbf{x}_i$'s are the training samples. For example, for SVMs, the α_i's are not null only for the support vectors, and positive (respectively, negative) for malicious (legitimate) samples. The value of Δg in these cases is simply given as:

$$\Delta g = \sum_{i=1}^{n} \alpha_i \left(k(\mathbf{x}, \mathbf{x}_i) - k(\mathbf{x}', \mathbf{x}_i)\right)\,. \tag{4.12}$$

As we aim to obtain decision functions that can potentially enclose the legitimate data (as discussed in Sect. 4.2), we focus here on kernels with an exponential form, including the RBF and the Laplacian kernel. They are respectively given by $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|_p^p)$, with $p = 1, 2$. The reason is that such kernels yield decision functions whose values tend to decrease while getting farther from the training data, thus yielding enclosing decision functions around one of the two classes.[6] For these kernels, it is not difficult to see that:

$$k(\mathbf{x}', \mathbf{x}_i) = e^{-\gamma \|\mathbf{x}' - \mathbf{x}_i\|_p^p} \geq e^{-\gamma \|\mathbf{x}' - \mathbf{x}\|_p^p} e^{-\gamma \|\mathbf{x} - \mathbf{x}_i\|_p^p}\,, \tag{4.13}$$

where we use the triangle inequality $\|(\mathbf{x}' - \mathbf{x}) + (\mathbf{x} - \mathbf{x}_i)\|_p \leq \|\mathbf{x}' - \mathbf{x}\|_p + \|\mathbf{x} - \mathbf{x}_i\|_p$ to upper bound the ℓ_p norm (valid for $p = 1, 2$). Substituting the above lower bound for $k(\mathbf{x}', \mathbf{x}_i)$ into Eq. (4.12), you get:

$$\Delta g \leq \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) \left(1 - e^{-\gamma \|\mathbf{x}' - \mathbf{x}\|_p^p}\right)\,. \tag{4.14}$$

[6] This has also been discussed in [69], exploiting a probabilistic model defined for open-set recognition, where the goal is to find enclosed decision functions around known training classes, to be able to detect novel classes at test time.

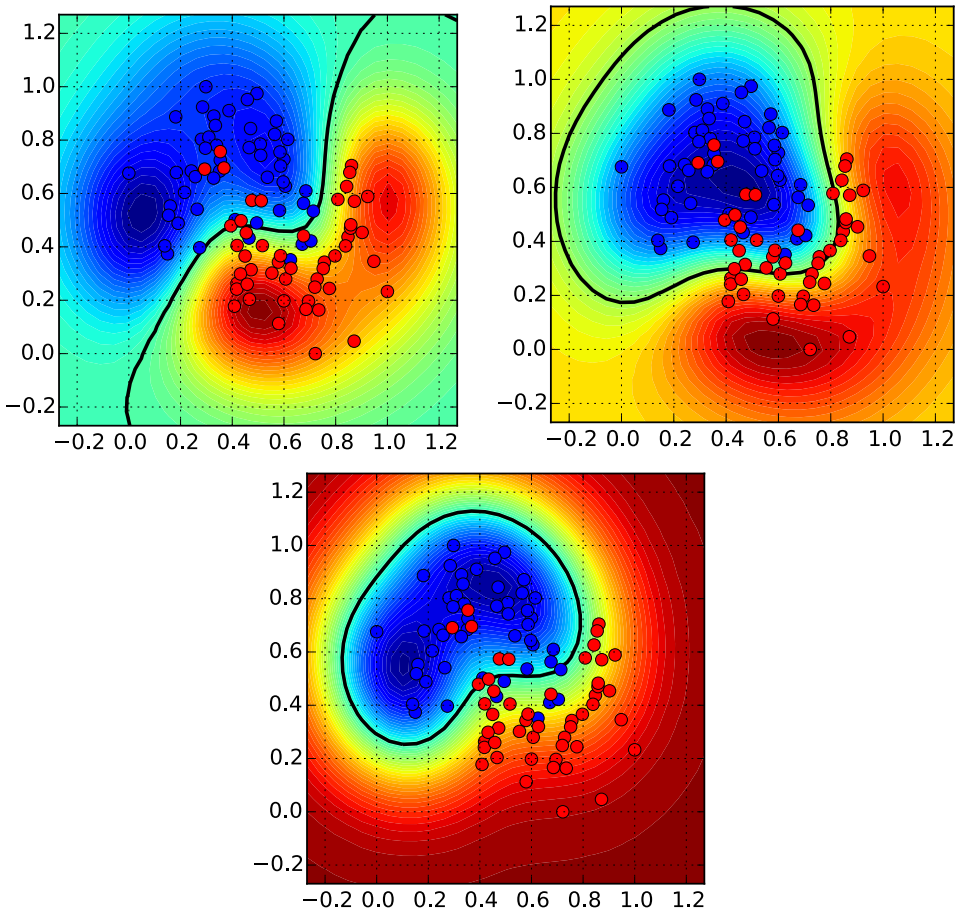


Figure 4.4: Decision boundaries, and $g(\mathbf{x})$ values (in colors), for RBF-SVM (first plot), cRBF-SVM (second plot), and γRBF-SVM (third plot). Note how the classifiers in the second and third plot provide a better enclosing of the legitimate data.

This upper bound reveals some interesting properties about the security of nonlinear kernels. First, it is clear that, if $\mathbf{x}' = \mathbf{x}$, $\Delta g = 0$. Instead, if $\|\mathbf{x}' - \mathbf{x}\|_p^p \to \infty$, $\Delta g = \sum_i \alpha_i k(\mathbf{x}, \mathbf{x}_i)$ and, thus, $g(\mathbf{x}') = b$. This means that, if $b \geq 0$ and $\mathbf{x}'$ is far enough from the training data, the decision function encloses the legitimate class, and $\mathbf{x}'$ is classified as malicious (and vice versa for $b < 0$), confirming the class-enclosing property of such kernels.

Kernel selection. The upper bound in Eq. (4.14) depends on $\|\mathbf{x}' - \mathbf{x}\|_p^p$. Since sparse and dense evasion attacks tend to minimize the ℓ_1 and the ℓ_2 distance between the same points, it should be clear that p should be chosen accordingly. Namely, if the evasion attack is sparse, then one should select the Laplacian kernel; otherwise, in case of dense

attacks, the RBF kernel should be preferred. The reason is that such choices will minimize the value of $\|\mathbf{x}' - \mathbf{x}\|_p^p$, *i.e.*, they will enable one to map the evasion samples in a region of the kernel space which is closer to the non-manipulated malicious samples (thus yielding a lower variation of g, and requiring more modifications to evade detection). This is an important observation, and it has a similar effect to the choice of the regularization term for linear classifiers; in fact, if one knows whether a sparse or a dense attack is deemed more likely, then a better regularizer (for linear classifiers) or kernel function (for non-linear classifiers) can be selected.

Cost-sensitive Learning. Another non-trivial suggestion coming from Eq. (4.14) is to set a lower value of the cost of classification for malicious samples. The reason is that the (absolute) α_i values obtained from SVM learning are bounded by the corresponding value of the SVM parameter C. Thus, if we set a lower C value for the malicious samples, their α_i values will decrease. This will in turn decrease the value of Δg, and thus, the impact of evasion attacks. Similarly, one may think of increasing C for legitimate data, to further decrease Δg. Recall however that the balance condition in SVM learning requires $\sum_{i=1}^{n} \alpha_i = 0$ and, thus, the final α_i values will clearly depend on the data at hand (it is not generally the case that they will be equal to the corresponding C value). Moreover, if one subsequently adjusts the value of b on a validation set to meet some specific requirements (*e.g.*, a desired false positive rate), then it may be even convenient to unbalance costs in a very different way. It is thus difficult to draw general conclusions from Eq. (4.14), despite the fact that cost-sensitive learning may be useful to shape the decision boundary in a different way, potentially improving security.

Kernel correction. Our analysis also suggests that reducing the value of γ should be beneficial, as it yields smoother functions. Furthermore, one may also think of assigning a different γ to each class, and reduce only that assigned to the malicious training samples (as they are in turn assigned positive α_i values). Despite this breaks the positive-semidefiniteness and symmetry of the kernel function, it could improve classifier security by reducing the maximum value of Δg. Although standard SVM learning algorithms may not converge if the kernel function is not symmetric, we can still learn a linear SVM in the similarity space induced by our kernel, by essentially using the kernel matrix as the set of input features. This is a well-known technique in similarity-based classification, which amounts to learning the SVM on the squared kernel [70, 71].

Similarly to the previous section, we consider here secure kernel machines against dense and sparse attacks.

4.4.2.1 Countering Dense Attacks

Based on the discussion in Sect. 4.4.2, to counter ℓ_2 attacks, one may train a standard SVM with the RBF kernel (RBF-SVM), potentially using unbalanced classification costs

(cRBF-SVM). A further option could be to assign distinct values of γ for the malicious and legitimate training samples (γRBF-SVM). Examples of decision boundaries for these classifiers are shown in Fig. 4.4.

4.4.2.2 Countering Sparse Attacks

In the case of sparse attacks, similar considerations can be made, despite the fact that one should use a Laplacian kernel. We thus consider the following classifiers: SVM with the Laplacian kernel (Lap-SVM), Lap-SVM with unbalanced costs (cLap-SVM), and Lap-SVM with different γ values for each class (γLap-SVM).

4.4.3 Non-Differentiable Classifiers

For non-differentiable classifiers, as the Decision Tree and the Random Forest, the previous considerations are not applicable. So, the only solution to improve the security of these classifiers is to find out a method that reduces as much as possible the region in the feature space assigned to the benign class. In a recent work [76], we have proposed a solution to cope against *mimicry* attacks, in particular for the scenarios where the attacker can modify her malicious samples only by adding features typically used in benign files, but without the possibility to remove them, so as to avoid the impairment of the intrusive functionality of the malware sample. It has been shown that this attack strategy is very effective against systems that are not designed to be secure against targeted evasion attempts [40]. To tackle this issue, we exploit an approach similar to that advocated in [60], named one-and-a-half-class (1.5C) classification. The underlying idea is to combine a two-class classifier with a one-class classifier to detect potential, anomalous samples during testing. In fact, most of the evasion samples constructed to evade detection by a two-class classifier can be considered anomalous with respect to the training (benign) data, and can be thus detected using this simple strategy. In particular, we build our 1.5C Multiple Classifier System (1.5C-MCS) using three distinct classifiers: (i) a Random Forest classifier trained on both benign and malicious data; (ii) a one-class SVM RBF trained only on benign data; and (iii) another one-class SVM RBF trained on the outputs of the aforementioned classifiers, using only benign data. In this way we can combine the very-good classification performance of the Forest in absence of attacks and the very-good anomaly-detection of the one-class in presence of them. The latter SVM will basically output an aggregated score to be thresholded to make the final decision. To better understand the behaviour of this system of classifiers, see Fig.4.11 in the next section.

4.5 Application Examples

In this section we report some experiments to support the considerations that we have made previously in this chapter. For our simulations we use several datasets, described in Appendix A. We point out that, concerning the MNIST dataset, we use the samples corresponding to 8 and 9, respectively for legitimate and malicious classes, in order to consider a binary problem.

4.5.1 Securing Linear Classifiers

In these experiments, we consider the SVM (*i.e.*, 2-norm regularization), the I-SVM (*i.e.*, infinity-norm regularization) and their version with unbalanced costs, as described in Sect. 4.4.1.

Experimental Setup. We randomly select 500 legitimate and 500 malicious samples from MNIST dataset, 2500 legitimate and 2500 malicious samples from the Spam dataset, and equally subdivide them to create a training and a testing set. We optimize the regularization parameter C (or the cost-sensitive parameters C_+, C_-) of each SVM through 3-fold cross-validation, maximizing a trade-off between the detection rate (*i.e.*, the fraction of correctly-classified malicious samples, also referred to as true positive rate, TP) at 1% false positive rate (FP) in the absence of attack, and under attack, estimated by simulating the attacks on a validation set (a subset of the training set), for different $d_{\max}$ values.

After classifier training, we perform sparse and dense evasion attacks on all malicious digit testing samples, and sparse evasion attacks on all malicious spam testing samples, for increasing values of $d_{\max}$. For the digit data, $d_{\max}$ represents either the ℓ_2 or ℓ_1 distance between the non-manipulated and the manipulated images, respectively, for dense and sparse attacks. In the case of sparse attacks, this corresponds to the number of gray-level pixel values modified by the attack. For spam filtering, it instead represents the number of modified words in each spam. We evaluate the corresponding performance in terms of TP at FP=1%, against an increasing value of $d_{\max}$ (recall that the performance in the absence of attack corresponds to $d_{\max} = 0$). We repeat this procedure five times, and report the average results on the original and modified test data. The corresponding (averaged) curves are called *security evaluation curves* [37, 40].

Experimental Results. Results are reported in Fig. 4.5. The reported security evaluation curves show that the main improvement in classifier security is due to the choice of a proper regularizer. In fact, I-SVM is much more secure than SVM against sparse attacks, and vice versa for dense attacks, confirming the discussion in Sect. 4.4.1. The use of different classification costs only introduces a slight improvement in terms of security, as it is difficult to achieve an improved level of security while retaining FP=1%. Images of manipulated digits under dense and sparse evasion attacks are reported in Fig. 4.6.

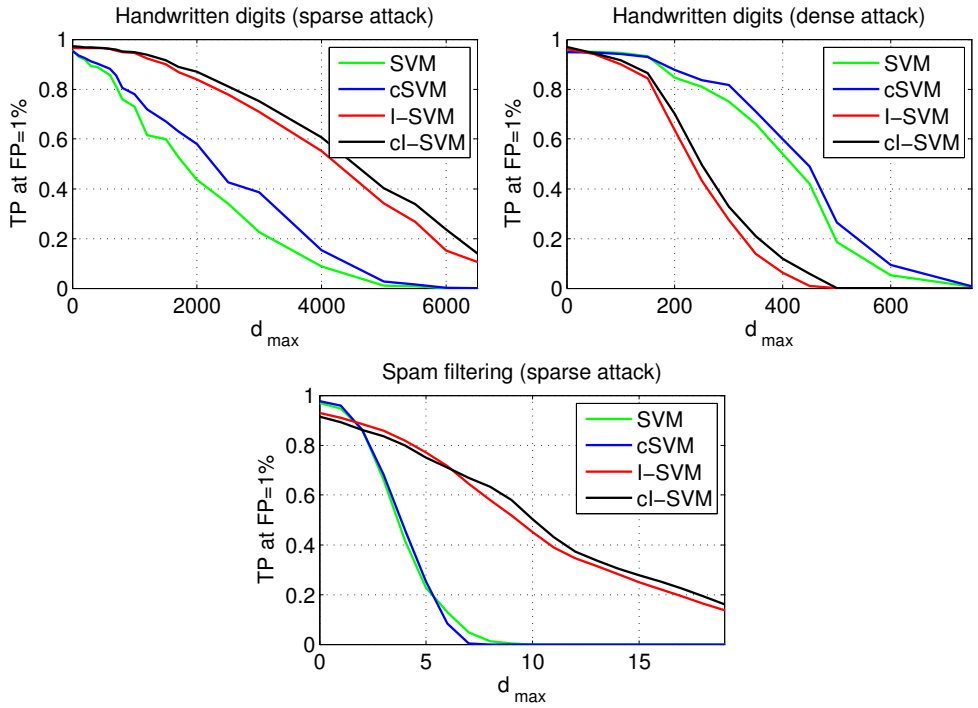


Figure 4.5: Security evaluation curves (TP at FP=1% vs $d_{\max}$) for the 9-vs-8 digit classification task against dense (first plot) and sparse (second plot) evasion attacks, and for the spam filtering data against sparse evasion attacks (third plot).

4.5.2 Securing Linear Classifiers with Limited Complexity

This setting refers to a situation in which the system has strict constraints in terms of computational complexity, as discussed in Chap.4.4.1.5.

Firstly, we consider dense and sparse attacks in the context of handwritten digit recognition, using the MNIST dataset, to visually demonstrate their blurring and salt-and-pepper effect on images. Then, we consider two real-world application examples including spam and PDF malware detection, using Spam and Lux0r datasets, investigating the behaviour of different regularization terms against (*sparse*) evasion attacks. As in [77], we select a subset of 100 features from the training data, by retaining those exhibiting higher values in malicious data, such that mimicking legitimate samples becomes more difficult. We consider different versions of the SVM classifier obtained using the different regularizers introduced in Sect.1.1.4 (2-norm, infinity-norm, 1-norm and elastic-net) and in Sect.4.4.1.5 (8-gon), shown in Fig. 1.4. We call them SVM, inf-SVM, 1-Norm SVM, Elastic-Net SVM and Octagonal SVM, respectively.

Sparsity and Security Measures. We evaluate the degree of sparsity S of a given

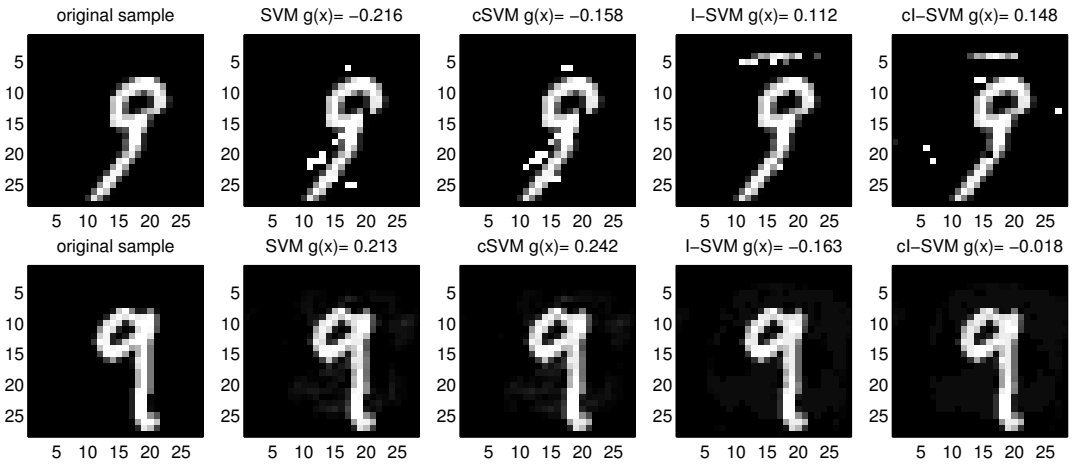


Figure 4.6: Original and manipulated handwritten digits at $d_{\max} = 3000$ by sparse attacks (top row), and at $d_{\max} = 250$ by dense attacks (bottom row), against SVM (second column), c-SVM (third column), I-SVM (fourth column), and cI-SVM (fifth column). Values of $g(\mathbf{x})$ are also reported for each digit and classifier, confirming that sparse attacks are less effective against I-SVM and cI-SVM, and that dense attacks are less effective against SVM and cSVM. Note also how the blurring effect induced by dense attacks is more difficult to spot for humans than the salt-and-pepper noise induced by sparse attacks.

linear classifier as the fraction of its weights that are equal to zero:

$$S = \tfrac{1}{\mathsf{d}} |\{w_j | w_j = 0, j = 1, \ldots, \mathsf{d}\}| \,, \tag{4.15}$$

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

To evaluate security of linear classifiers, we define a measure E of *weight evenness*, similarly to [10, 68], based on the ratio of the ℓ_1 and ℓ_∞ norm:

$$E = \frac{\|\mathbf{w}\|_1}{\mathsf{d}\|\mathbf{w}\|_\infty} \,, \tag{4.16}$$

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

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

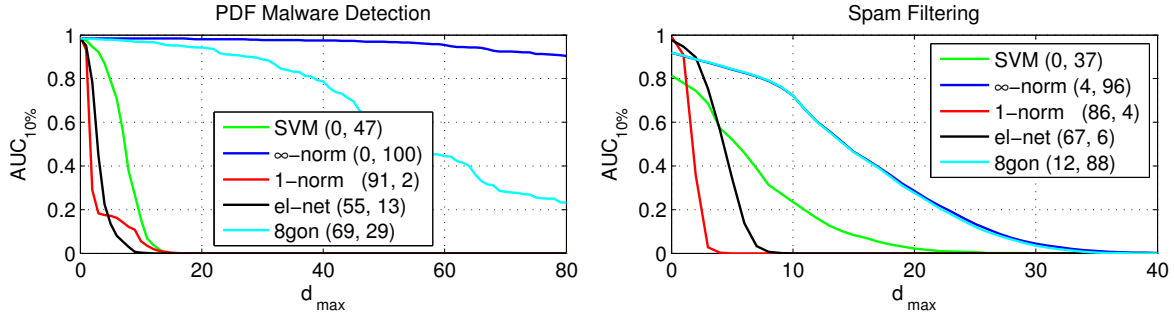


Figure 4.7: Classifier performance under attack for PDF malware and spam data, measured in terms of $\text{AUC}_{10\%}$ against an increasing number $d_{\max}$ of modified features. For each classifier, we also report (S, E) percentage values (Eqs. 4.15-4.16) in the legend.

following objective on the training data:

$$\text{AUC} + \alpha E + \beta S \tag{4.17}$$

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

Experimental Results. We consider first PDF malware and spam detection. In these applications, as mentioned before, only sparse evasion attacks make sense, as the attacker aims to minimize the number of modified features. In Fig. 4.7, we report the AUC at 10% false positive rate for the considered classifiers, against an increasing number of words/keywords changed by the attacker. This experiment shows that the most secure classifier under sparse evasion attacks is the Infinity-norm SVM, as expected, given that its regularizer corresponds to the dual norm of the attacker's cost/distance function. Notably, the Octagonal SVM yields reasonable security levels while achieving much sparser solutions, as expected. This experiment really clarifies how much the choice of a proper regularizer can be crucial in real-world adversarial applications.

By looking at the values reported in Fig. 4.7, it may seem that the security measure E does not properly characterize classifier security under attack; *e.g.*, note how Octagonal SVM exhibits lower values of E despite being more secure than SVM on the PDF data. The underlying reason is that the attack implemented on the PDF data only considers feature increments, while E generically considers any kind of manipulation. Accordingly,

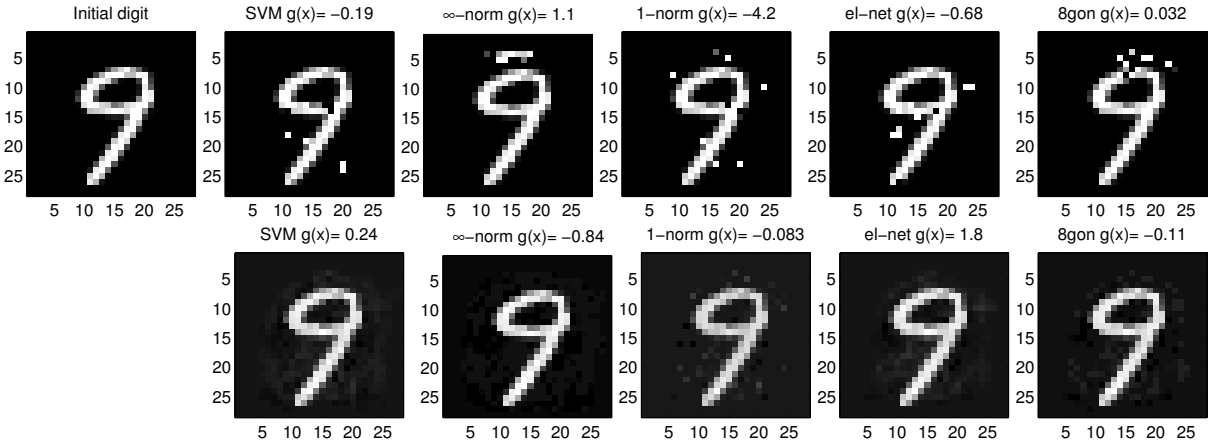


Figure 4.8: Initial digit "9" and its versions modified to be misclassified as "8". Each column corresponds to a different classifier (from *left* to *right*): SVM, Infinity-norm SVM, 1-norm SVM, Elastic-net SVM, Octagonal SVM. *Top row*: sparse attacks (ℓ_1), with $d_{\max} = 2000$. *Bottom row*: dense attacks (ℓ_2), with $d_{\max} = 250$. Values of $g(\mathbf{x}) < 0$ denote a successful classifier evasion (*i.e.*, more vulnerable classifiers).

one should define alternative security measures depending on specific kinds of data manipulation. However, the security measure E allows us to properly tune the trade-off between security and sparsity also in this case and, thus, this issue may be considered negligible.

Finally, to visually demonstrate the effect of sparse and dense evasion attacks, we report some results on the MNIST handwritten digits. In Fig. 4.8, we show the "9" digit image modified by the attacker to have it misclassified by the classifier as an "8". Note how dense attacks are more effective, as they only produce a slightly-blurred effect on the image, while sparse attacks create more evident visual artefacts. This simple example also confirms that Infinity-norm and Octagonal SVM are more secure against sparse attacks, while SVM and Elastic-net SVM are more secure against dense attacks.

4.5.3 Securing Kernel Machines

For nonlinear differentiable classifiers, we consider an adversarial application example involving the detection of malware in PDF files (Lux0r dataset). We apply the same feature selection explained in the setting of the previous example.

Experimental Setup. We consider the classifiers secure to sparse evasion attacks described in Sect. 4.4.2, namely, Lap-SVM, cLap-SVM, and γLap-SVM. We randomly split the dataset into training and testing sets of 5,000 samples each, and optimize the classifiers' parameters as done in the previous experimental setup.

To set the system, we simulate attacks starting from a subset of the training set, and

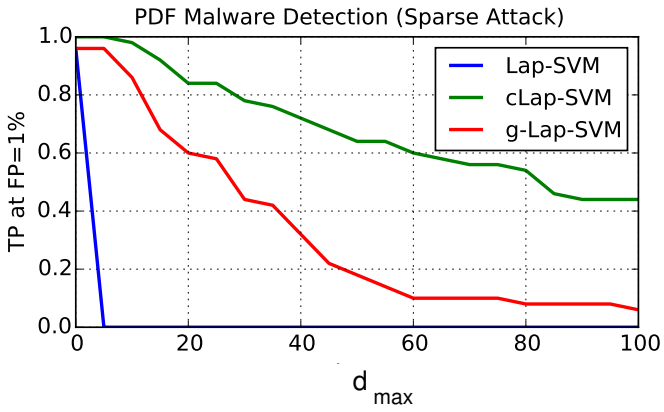


Figure 4.9: Security evaluation curves (TP at FP=1% vs $d_{\max}$) for PDF malware detection against sparse evasion attacks.

we chose the parameters that allow the best trade-off between the accuracy pre and post attacks. However, this mechanism is computational demanding, so we hold off on finding a better solution to next works.

Experimental Results. The results averaged over five repetitions are reported in Fig. 4.9. It is easy to appreciate how the secure variants of the Lap-SVM algorithms outperform the baseline algorithm. Another conclusion, that we can see from these results, is that the unbalancing of the costs on the error classification is preferable respect to the unbalancing of the γ values for each class.

4.5.4 Securing Random Forests

In this experiment, we tested the resilience of the proposed approach against the mimicry attacks, described previously, for Lux0r and SWF datasets.

Experimental Setup. For each dataset, we randomly split the data in a training and a test set, respectively consisting of 70% and 30% of the total number of samples (Lux0r contains 17826 samples, SWF 6776). The feature sets of the two datasets, consisting respectively of 3272 and 2587 elements, are reduced through feature selection, to obtain a more compact feature set and facilitate the training process of our classifiers, by tackling the so-called curse of dimensionality [1]. In particular, we exploit a feature selection criterion, slightly different from [77], based on *information gain*, and select the first 100 features with the highest *occurrence score* $S = |p(x_i|M) - p(x_i|B)|$, being x_i the i-th feature value, and M and B the sets of malicious and benign samples [1]. The selected features include functions, attributes and classes that are often used by malware to perform their actions. For example, selected features among `Flash` files are `flash.events.Event`, `flash.utils.ByteArray`, `Math` and other classes that are often used to manipulate memory to perform attacks. With respect to `JavaScript`, selected features include, among

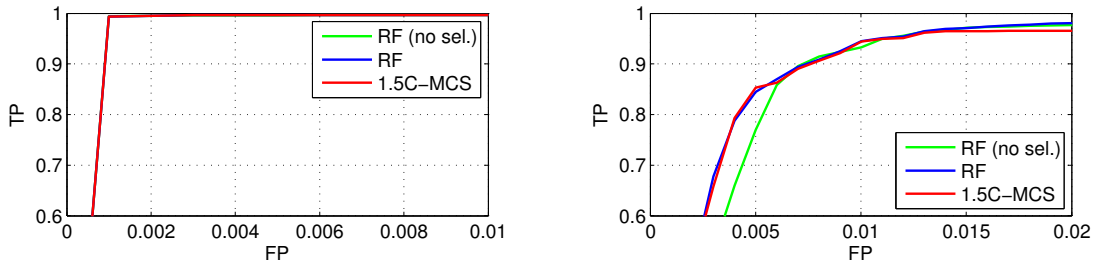
others, `app.ViewerVersion`, `app.['eval']`, `app.PlugIns`. Such features are often used to obfuscate code. We used Random Forest and SVM RBF classifiers, as described in Sect.4.4.3, and we optimize the parameters of each classifier through a 5-fold cross validation, maximizing the accuracy performed on the training set. Finally, the malicious files of the test set were modified according to the *mimicry* strategy. It is worth noting that we are not building the real sample corresponding to the manipulated malicious file, but we are only simulating the effect of the attack at the feature level, *i.e.*, we are simulating changes in the feature values of each malicious sample that can be practically implemented also to build a real malware sample. In particular, we only consider *adding* API calls from benign samples. As mentioned in Sect.4.4.3, removing API calls from a malicious sample may compromise the intrusive functionality of the embedded exploitation code. This process was repeated five times, to avoid biases due to the quality of a specific training-test split. For each split, we compare the classification performance, in terms of ROC curve, of the Random Forest, trained on the training set with all of the feature, with another Forest and the 1.5C-MCS proposed, both trained on the dataset with feature selection.

Then, we evaluate the security of the previous classifiers against mimicry attacks, showing their detection rate, at a given false positive rate, for each classifier (for `JavaScript`, we set $FP = 0.2\%$, whilst for `ActionScript` we set $FP = 1\%$).

Experimental Result. In Fig.4.10 we report the results for the standard performance of the classifiers. Notably, there are clear differences between the results attained on `JavaScript` and `ActionScript`. In particular, although the results are very good for both languages, classifying `ActionScript` files is significantly more difficult than classifying `JavaScript` files. The reason is that benign and malicious `ActionScript` files are more similar, in terms of API calls, than their `JavaScript` counterparts. Selecting features allows one to attain better performances with Random Forests for `Action-Script` codes. The 1.5C-MCS exhibits essentially the same performance of the best Random Forest classifier. This was somehow expected, as the one-class component has been designed specifically to detect targeted, anomalous attacks that significantly deviate from benign data.

Fig.4.12 shows how the performances of the considered classifiers decrease under mimicry attacks.

The first thing to observe is that the attack is tremendously effective against the `ActionScript` dataset. On the `JavaScript` dataset the effect is lower, but it can be increased by raising up the amount of added samples (up to 100, see [77]). Random Forests classifiers are considerably vulnerable to this attack, while the 1.5C-MCS remains significantly secure. The underlying reason is that, in the latter case, the one-class SVM is able to correctly spot the anomalous behaviour of the attack samples with respect to the rest of the training data used to learn the classifier. To better explain this phenomenon, in Fig. 4.11 we report a scatter plot that depicts benign (blue points), malicious (red



(a) Performances on `JavaScript` (b) Performance on `ActionScript`

Figure 4.10: ROC curves for PDF files embedding `JavaScript` (left) and for SWF files embedding `ActionScript` (right). These curves report the results for Random Forests (either by using all the features or the 100 most discriminant ones, and for the 1.5C-MCS.

.

points) and attack (green points) data in the space characterized by the outputs of the two combined classifiers. In addition, the decision function of the 1.5C-MCS is also shown. Differently to what happens with SVM RBF and stand-alone Random Forest, the circular shape of the 1.5C-MCS encloses all the benign samples, so that malicious and attack samples (which are located in a different position compared to standard malicious samples - see the green points) are considered anomalous. Notably, while the scores assigned to the attack samples by the Random Forest classifier are closer to those assigned by the same classifier to the benign data (, they would evade detection by this classifier), the one-class SVM is able to well-separate them from the rest of the data. This also applies to the 1.5C-MCS, which is able to correctly assign a high score value to the attack samples, and, therefore, to successfully detect them.

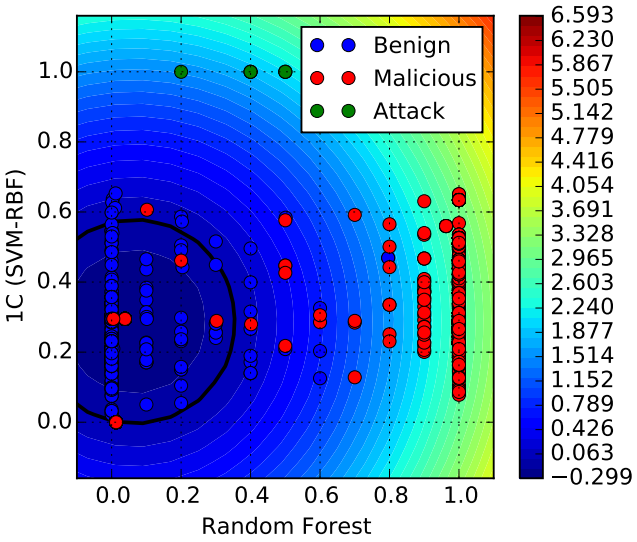
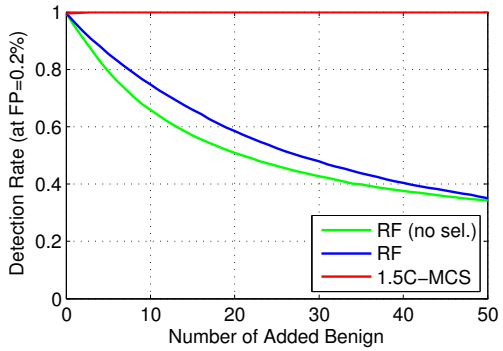
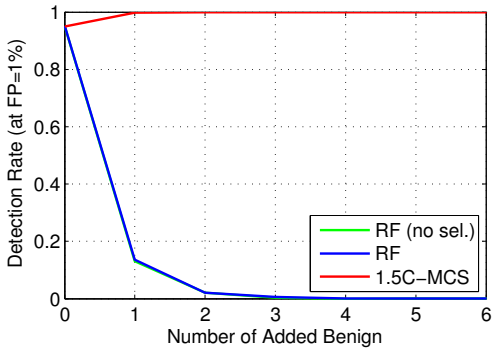


Figure 4.11: The decision function of our 1.5C-MCS (shown in colors) in the bi-dimensional space spanned by the outputs of the combined classifiers. The decision boundary is highlighted with a solid black line, while blue and red points respectively represent benign and malicious files. Malicious files manipulated with the mimicry attack strategy are reported as green points.



(a) Performances on `JavaScript` (fp=0.2%)



(b) Performances on `ActionScript` (fp=1%)

Figure 4.12: Detection rate of our classifiers against *mimicry* attacks in which an increasing number of benign samples is added to each malware sample, for `JavaScript` and `ActionScript` files.

Chapter 5

Conclusions

In recent years, machine learning has been widely used in security sensitive applications. Due to their intrinsic adversarial nature, these scenarios differ from the classical machine learning settings. They are characterized by the presence of intelligent adversaries who can deliberately attack the classifier by carefully manipulating malicious data to subvert the system normal operation. It is so clear the need to analyse this problem and find out countermeasures, in order to make the attacks ineffective, or at list to limit their influence.

In this work, we have proposed security mechanisms for classification systems to oppose the attackers' actions.

Firstly, we have presented a general framework to characterize the attacker's behaviour introduced by [37,38], that enables us to analyse the security of a system in any application setting. Then, we have shown how this model can be easily adapted to a specific scenario like the one related to biometric recognition systems. We have discussed how this novel perspective may not only inspire the simulation of more sophisticated attack scenarios, but also how, based on such scenarios, more effective countermeasures can be proactively developed. As concrete application examples, we have developed a sophisticated spoofing attack and a novel countermeasure to contrast a recently proposed poisoning attack against an adaptive face verification system. This countermeasure has proved to be efficient in the attack detection; the keypoint for its success is the appropriate choice of the radius of the 'normal-updates' hypersphere. One should choose it according to an hypothetical genuine behaviour, otherwise the attacks will not be detected or the number of rejected genuine updates will increase, depending on whether the radius is too big or too small. In both cases, the system would become useless. Note that the attacker could reach her scope anyway, if she limits the distance of each attack, increasing the number of attempts. However, there could be some mechanisms to limit the number of consecutive accesses or it may happen some genuine updates during the sequence of attacks. These problems would further slow the achievement of her goal, and the effort would become too big respect to the possible benefits.

Secondly, we have shown the current state of the art of the methods to make classi-

fiers more secure against adversaries' attacks, highlighting their disadvantages. To overcome the limitations concerned the countermeasures for evasion attacks at test time, we have proposed solutions to improve the security of classification systems retaining the same computational complexity of their non-secure versions. Our ideas essentially revolve around the intuition of finding more secure parameter configurations for existing algorithms, rather than overly-complicating existing ones. We believe that this would help overcoming the intrinsic limitations of current secure learning algorithms, namely, their strong theoretical requirements, complexity of implementation, and scalability issues due to their training complexity, to finally favour the wide adoption of more secure learning algorithms in practical settings. To represent the attacker, we have used the general taxonomy previously explained, and to simulate the manipulations she can make on the samples, we have designed a set of algorithms that includes knowledge about the application scenario at hand, obtaining more accurate and faster solutions with respect to other existing solvers. We have highlighted that an appropriate choice of the classifier parameters, connected to an adequate comprehension of the given scenario and the related attack, allows us to significantly reduce the effectiveness of the malicious actions. We have explained that, to improve the security of linear SVMs, it is necessary to select a correct type of regularizer and an appropriate balance of the cost of classification errors on the different classes. In fact, in the presence of sparse attacks, our experiments have shown that Infinity-norm SVMs can drastically outperform the security of standard SVMs. When dense attacks are instead deemed more likely, the standard SVM may be retained a good compromise. The results have also shown that a proper choice of the regularization term is the key factor for significantly improving the security of linear classifier, while unbalancing the costs of classification errors in different classes introduces only a slight enhancement. Moreover, we have proposed a new octagonal regularizer that enables trading sparsity for a marginal loss of security under sparse evasion attacks. This is extremely useful in applications where sparsity and computational efficiency at test time are crucial. In this setting, in presence of dense attacks, one may trade some level of security for sparsity using the Elastic-net SVM. The same considerations are valid for the kernel SMVs, where it is also possible to set the type of the kernel function and its parameters. With respect to the results obtained for the linear classifiers, in this case, unbalancing the classification costs introduces a significant improvement in security. For some classifiers, including Random Forests, it is clearly not possible to apply the aforementioned countermeasure. For this reason we have proposed a different defence strategy, based on enforcing the attacker to mimic as much as possible the characteristics of benign samples to evade detection. To this scope, we have combined the output of the Random Forest classifier with that of a One-Class SVM, aiming to detect outlying evasion samples. As shown in our results, this solution allows us to drastically improve the detection of evasive malicious samples, at the cost of only misclassifying a small fraction of benign objects.

5.1 Future Work

As short-term future work, we aim to better systematize the state of the art in secure learning, and to extend our experimental analysis also to current secure-learning approaches. It may be worth considering also application settings in which the attack can be a combination of sparse and dense attacks, and try to mitigate their impact by exploiting a convex combination of infinity-norm and ridge regularization. Moreover, we plan to investigate, in a similar manner, the security properties of learning algorithms against poisoning attacks. Since in poisoning scenario the attacker can only inject few samples into the training set to mislead learning, this type of attack can be considered as sparse (in terms of the number of samples). Thus, an interesting idea may be to consider an infinity-norm regularization on the values of non-linear kernel machines, such that more evenly-distributed weights are assigned to the training samples. This should indeed reduce the impact of each poisoning (outlying) sample in the training set, making learning more secure even in the presence of poisoning. Regarding the solution proposed to non-differentiable classifiers, we plan to test our approach against obfuscated samples and more sophisticated evasive attacks.

As long-term future work, we want to study and analyse the security of other learning algorithms, in particular deep neural networks. The technological progress has allowed us to apply these powerful classifiers to many areas, from robotic vision (*e.g.*, iCub[1]) to intelligent personal digital assistants (*e.g.*, Cortana, Siri). In some of these applications, such as the recognition of the surrounding environment in self-driving vehicles, or the central data management in IoT Smart Cities, the robustness against potential malicious attacks is very sensitive, because the effects would have a significant impact on people safety. Thus, it is evident that developing novel methods capable of improving the security level of these systems remains a relevant open research issue.

[1] http://www.icub.org/

Appendix A

Dataset

In this appendix we present the datasets used to the previous simulations.
MNIST dataset. It is a large database of handwritten digits from 0 to 9, contains around 70,000 images [80]. Each image is represented by a vector of 784 features, corresponding to its gray-level pixel values. As in [40], we simulate an adversarial classification problem where the digits 8 and 9 correspond to the legitimate and malicious classes, respectively.
TREC 2007. Most spam filters include an automatic text classifier that analyses the email's body text. In the simplest case Boolean features are used, each representing the presence or absence of a given term. For our experiments we use the TREC 2007 spam track data, consisting of about 25000 legitimate and 50000 spam emails [81]. We extract a dictionary of terms (features) from the first 5000 emails (in chronological order) using the same parsing mechanism of SpamAssassin, and then select the 200 most discriminant features according to the information gain criterion [79]. We simulate a well-known (*sparse*) evasion attack in which the attacker aims to modify only few terms. Adding or removing a term amounts to switching the value of the corresponding Boolean feature [10, 16, 28, 37, 40].
Lux0r dataset. The PDF-Reader is an application that is often targeted by attackers. A PDF file can host different kinds of contents, like Flash and JavaScript, making it an appealing vector to convey malware. In fact, such third-party applications can be exploited by attacker to execute arbitrary operations. To simulate an attack involving PDF files, we use the Lux0r dataset [77], which consists of 17,826 unique PDF documents embedding JavaScript code: 12,592 malicious samples and 5,234 benign samples. The whole dataset is the result of an extensive collection of PDF files until 2016 from security blogs such as "Contagio", and "Malware don't need Coffee", analysis engines such as VirusTotal, and search engines such as Google and Yahoo. Every file is represented using 736 features that correspond to the number of occurrences of a predefined set of Javascript function calls (API), where every API represents an action performed by one of the objects that are contained into the PDF file (*e.g.*, opening another document that is stored inside the file).

Note that an attacker cannot trivially remove an API from a PDF file without corrupting its functionality. Conversely, she can easily add new APIs by inserting new function calls. For this reason, we simulate this attack by only considering feature increments (*i.e.*, decrementing a feature value is not allowed). Accordingly, the most convenient strategy to mislead a malware detector (classifier) is to insert as many occurrences of a *given* API as possible, which is a sparse attack.[1]

SWF dataset. ActionScript, as JavaScript, is derived from ECMAScript, a standardized programming language maintained by Ecma with the ECMA-262 standard. While JavaScript is mostly used for web applications and to extend functionality of third parties formats such as PDF, ActionScript is used as an essential support for delivering Flash-based content. In particular, ActionScript is mainly employed in SWF files, although it can also be employed in PDF files to show Flash animations inside a document. This dataset contains 6,776 examples of SWF files, 4,425 of which are benign and the remaining 2,351 are malicious. Each sample is represented from 2,587 features. Differently from the Lux0r dataset, there are more benign files than malicious ones, as Flash-based attacks have only considerably increased since 2015. These files were collected until 2016 by using the VirusTotal service.

DIEE Face. It is a dataset consisting of 40 different clients with 60 images each, for a total of 2,400 face images. The face images of each client were collected into two session, using a commercial webcam, with a time interval of about two weeks between them, under different lighting conditions and facial expressions. This induced a high intra-class variability of the face images, which makes face recognition particularly challenging.

[1]Despite no upper bound on the number of injected APIs may be set, we set the maximum value for each API to the corresponding one observed during training.

Bibliography

[1] Christopher M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, October 2007

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. Wiley-Interscience Publication, 2000

[3] `http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf`

[4] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, R. C. Williamson, *Estimating the support of a high-dimensional distribution*. Neural Computation, 13(7):1443–1471, 2001.

[5] C. Cortes and V. Vapnik, *Support-vector networks*. Machine Learning, 20:273–297, 1995.

[6] R. Tibshirani, *Regression Shrinkage and Selection via the lasso*. Journal of the Royal Statistical Society. Series B (methodological) 58 (1). Wiley: 267–88, 1996

[7] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, *Misleading worm signature generators using deliberate noise injection*. In Security and Privacy. IEEE Symposium on, pages 15–31, 2006

[8] B. Biggio, G. Fumera, F. Roli, *Multiple classifer systems for robust classifer design in adversarial environments*. Int'l J. of Machine Learning and Cybernetics 1(1), 27–41, 2010

[9] N. Dalvi, P. Domingos, Mausam, S. Sanghai, D. Verma, *Adversarial classifcation*. In: 10th ACM SIGKDD Int'l Conf. on Knowl. Discovery and Data Mining (KDD). pp. 99–108, 2004

[10] A. Kolcz, C.H. Teo, *Feature weighting for improved classifer robustness*. In: Sixth Conf. on Email and Anti-Spam (CEAS). Mountain View, CA, USA, 2009

[11] D. Lowd, C. Meek, *Adversarial learning.* In: Press, A. (ed.) Proc. of the Eleventh ACM SIGKDD Int'l Conf. on Knowl. Disc. and D. Mining (KDD). pp. 641–647, Chicago, IL., 2005

[12] D. B. Skillicorn, *Adversarial knowledge discovery.* IEEE Intell.Syst., vol. 24, pp. 54–61, 2009

[13] D. Fetterly, *Adversarial information retrieval: The manipulation of web content.* ACM Computing Reviews, 2007

[14] S. Rizzi, *What-if analysis.* Enc. of Database Systems, pp. 3525–3529, 2009.

[15] G. L. Wittel and S. F. Wu, *On attacking statistical spam filters.* In 1st Conf. on Email and Anti-Spam, CA, USA, 2004.

[16] D. Lowd and C. Meek, *Good word attacks on statistical spam filters.* In 2nd Conf. on Email and Anti-Spam, CA, USA, 2005.

[17] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, *Can machine learning be secure?* In Proc. Symp. Inf., Computer and Commun. Sec. (ASIACCS) . NY, USA: ACM, 2006, pp. 16–25.

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

[19] B. I. Rubinstein, B. Nelson, L. Huang, A. D. Joseph, S. Lau, S. Rao, N. Taft, and J. D. Tygar, *Antidote: understanding and defending against poisoning of anomaly detectors* In Proc. 9thACM SIGCOMM Internet Measurement Conf., ser. IMC 09. NY,USA: ACM, 2009, pp. 1–14.

[20] M. Kloft and P. Laskov, *Online anomaly detection under adversarial impact.* In Proc. 13th Int'l Conf. on Artificial Intell. and Statistics, 2010, pp. 405–412.

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

[22] B. Biggio, G. Fumera, and F Roli, *Design of robust classifiers for adversarial environments* In IEEE Int'l Conf. on Systems, Man,and Cybernetics, 2011, pp. 977–982.

[23] B. Biggio, I. Corona, G. Fumera, G. Giacinto, and F. Roli, *Bagging classifiers for fighting poisoning attacks in adversarial environments.* In Proc. 10th Int'l Workshop on Multiple Classifier Systems, ser. LNCS, vol. 6713. Springer-Verlag, 2011, pp. 350–359.

[24] B. Biggio, G. Fumera, F. Roli, and L. Didaci, *Poisoning adaptive biometric systems.* In Structural, Syntactic, and Statistical Pattern Recognition, ser. LNCS, vol. 7626. Springer, 2012, pp. 417–425.

[25] B. Biggio, B. Nelson, and P. Laskov, *Poisoning attacks against support vector machines.* In Proc. 29th Int. Conf. on Machine Learning, 2012

[26] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma, *Adversarial classification.* In 10th ACM SIGKDD Int. Conf. On Knowl. Discovery and Data Mining, WA, USA, 2004, pp. 99–108.

[27] 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, IL, USA, 2011, pp. 43–57

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

[29] P. Laskov and M. Kloft, *A framework for quantitative security analysis of machine learning* In Proc. 2nd ACM Workshop on Security and Artificial Intelligence. NY, USA: ACM, 2009, pp. 1–4

[30] A. A. Cardenas, J. S. Baras, and K. Seamon, *A framework for the evaluation of intrusion detection systems.* In Proc. IEEE Symp. On Security and Privacy. DC, USA: IEEE CS, 2006, pp. 63–77.

[31] B. Biggio, G. Fumera, and F. Roli, *Multiple classifier systems for adversarial classification tasks* In Proc. 8th Int'l Workshop on Multiple Classifier Systems, ser. LNCS, vol. 5519. Springer, 2009, pp. 132–141.

[32] M. Bruckner, C. Kanzow, and T. Scheffer, *Static prediction games for adversarial learning problems* J. Mach. Learn. Res., vol. 13, pp.2617–2654, 2012

[33] A. K. Jain, K. Nandakumar, and A. Nagar, *Biometric template security.* J. Adv. Sign. Proc., vol. 2008, pp. 1–17, 2008.

[34] U. Uludag, A. Ross, and A. K. Jain, *Biometric template selection and update: a case study in fingerprints* Patt. Rec., vol. 37, no. 7, pp. 1533–1542, 2004.

[35] C. Ryu, H. Kim, and A. K. Jain, *Template adaptation based fingerprint verification* In Proceedings of the 18th Int'l Conf. on Pattern Recognition - Volume 04, ser. ICPR '06. Washington, DC, USA: IEEE CS, 2006, pp. 582–585.

[36] B. Biggio, L. Didaci, G. Fumera, and F. Roli, *Poisoning attacks to compromise face templates.* In 6th IAPR Int'l Conf. Biometrics, Madrid, Spain, 2013, pp. 1–7.

[37] B. Biggio, G. Fumera, and F. Roli, *Security evaluation of pattern classifiers under attack*. IEEE Trans. Knowl. Data Eng., vol. 26, no. 4, pp. 984–996, 2014.

[38] B. Biggio, G. Fumera, and F. Roli, *Pattern recognition systems under attack: Design issues and research challenges* Int'l J. Patt. Recogn. Artif. Intell., vol. 28, no. 7, 1460002, 2014.

[39] B. Biggio, G. Fumera, P. Russu, L. Didaci and F. Roli, *Adversarial Biometric Recognition: a review on biometric system security from the adversarial machine-learning perspective* IEEE Signal Processing Mag., vol. 32, no. 5, pp. 31–41, 2015.

[40] 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 H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, editors, European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD), Part III, volume 8190 of *Lecture Notes in Computer Science*, pages 387–402. Springer Berlin Heidelberg, 2013.

[41] F. Wang, W. Liu, and S. Chawla, *On sparse feature attacks in adversarial learning*. In IEEE Int'l Conf. on Data Mining (ICDM), pages 1013–1018. IEEE, 2014.

[42] N. K. Ratha, J. H. Connell, and R. M. Bolle, *An analysis of minutiae matching strength*. In AVBPA, ser. LNCS, J. Bigün and F. Smeraldi, Eds., vol. 2091. Springer, 2001, pp. 223–228.

[43] R. N. Rodrigues, L. L. Ling, and V. Govindaraju, *Robustness of multimodal biometric fusion methods against spoof attacks*. J. Vis. Lang. Comput., vol. 20, no. 3, pp. 169–179, 2009.

[44] B. Biggio, Z. Akhtar, G. Fumera, G. L. Marcialis, and F. Roli, *Security evaluation of biometric authentication systems under real spoofing attacks*. IET Biometrics, vol. 1, no. 1, pp. 11–24, 2012.

[45] A. Adler, *Vulnerabilities in biometric encryption systems*. in 5th Int'l Conf. Audio- and Video-Based Biometric Person Auth., ser. LNCS, Springer, July 20-22 2005, pp. 1100–1109.

[46] P. Johnson, B. Tan, and S. Schuckers, *Multimodal fusion vulnerability to non-zero effort (spoof) imposters*. in IEEE Int'l Workshop on Information Forensics and Security, 2010, pp. 1–5.

[47] T. Matsumoto, H. Matsumoto, K. Yamada, and S. Hoshino, *Impact of artificial "gummy" fingers on fingerprint systems*. Datenschutz und Datensicherheit, vol. 26, no. 8, 2002.

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

[49] F. R. Hampel, E. M. Ronchetti, P. J. Rousseeuw, and W. A. Stahel, *Robust Statistics: The Approach Based on Influence Functions.* Probability and Mathematical Statistics. John Wiley and Sons, NY, USA, 1986

[50] R. A. Maronna, R. D. Martin, and V. J. Yohai, *Robust Statistics: Theory and Methods.* Probability and Mathematical Statistics. John Wiley and Sons, NY, USA, 2006

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

[52] A. Globerson and S. T. Roweis, *Nightmare at test time: robust learning by feature deletion.* In Proceedings of the 23rd Int'l Conf. on Mach. Learn., W. W. Cohen and A. Moore, Eds., vol. 148. ACM, 2006, pp. 353–360.

[53] C. H. Teo, A. Globerson, S. Roweis, and A. Smola, *Convex learning with invariances.* In NIPS 20 : MIT Press, 2008, pp. 1489–1496.

[54] 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 Symp. Security and Privacy. Los Alamitos, CA, USA: IEEE CS, 2008, pp. 81–95.

[55] 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.* J. Mach. Learn. Res., vol. 13, pp. 1293–1332, 2012.

[56] 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.

[57] J. Bi and T. Zhang. *Support vector classification with input data uncertainty.* In Advances in Neural Information Processing Systems 17, 2004.

[58] W. Xu, Y. Qi, and D. Evans. *Automatically evading classifiers.* In Proc. 23rd Annual Network & Distributed System Security Symposium (NDSS). The Internet Society, 2016.

[59] A. Kantchelian, J. D. Tygar, and A. D. Joseph, *Evasion and hardening of tree ensemble classifiers.* In 33rd ICML, volume 48 of JMLR Workshop and Conference Proceedings, pages 2387–2396. JMLR.org, 2016.

[60] B. Biggio, I. Corona, Z.-M. He, P. P. K. Chan, G. Giacinto, D. S. Yeung, and F. Roli, *One-and-a-half-class multiple classifier systems for secure learning against evasion attacks at test time.* In F. Schwenker, F. Roli, and J. Kittler, editors, Multiple Classifier Systems, volume 9132 of Lecture Notes in Computer Science, pages 168–180. Springer International Publishing, 2015.

[61] A. M. Nguyen, J. Yosinski, and J. Clune, *Deep neural networks are easily fooled: High confidence predictions for unrecognizable images.* In IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pages 427–436. IEEE, 2015.

[62] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, *The limitations of deep learning in adversarial settings.* In Proc. 1st IEEE European Symposium on Security and Privacy, pages 372–387. IEEE, 2016.

[63] H. Xu, C. Caramanis, and S. Mannor, *Robustness and regularization of support vector machines.* Journal of Machine Learning Research, 10:1485–1510, July 2009.

[64] S. Sra, S. Nowozin, and S. J. Wright, *Optimization for Machine Learning.* The MIT Press, 2011.

[65] R. Livni, K. Crammer, A. Globerson, E.-i. Edmond, and L. Safra, *A simple geometric interpretation of SVM using stochastic adversaries.* In JMLR W&CP - Proc., volume 22 of AISTATS '12, pages 722–730, 2012.

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

[67] T. Miyato, S.-i. Maeda, M. Koyama, K. Nakae, and S. Ishii, *Distributional smoothing with virtual adversarial training.* In International Conference on Learning Representation, 2016.

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

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

[70] E. Pekalska, P. Paclik, and R. P. Duin, *A generalized kernel approach to dissimilarity based classification.* Journal of Machine Learning Research, 2:175–211, 2001.

[71] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti, *Similarity-based classification: Concepts and algorithms.* J. Mach. Learn. Res., 10:747–776, June 2009.

[72] Bondell, Reich, *Simultaneous regression shrinkage, variable selection, and supervised clustering of predictors with OSCAR*, (2008)

[73] A. Demontis, P. Russu, B. Biggio, G. Fumera, e F. Roli, *On Security and Sparsity of Linear Classifiers for Adversarial Settings.* In Joint IAPR Int'l Workshop on Structural, Syntactic, and Statistical Pattern Recognition, Merida, Mexico, 2016, vol 10029 of LNCS, pagg 322-332.

[74] P. Russu, A. Demontis, B. Biggio, G. Fumera, e F. Roli, *Secure Kernel Machines against Evasion Attacks.* In 9th ACM Workshop on Artificial Intelligence and Security, Vienna, Austria, 2016, pagg 59-69

[75] J. Zhu, S. Rosset, R. Tibshirani, T. Hastie, *1-norm support vector machines.* In Thrun, S., Saul, L., Schölkopf, B., eds.: NIPS 16, MIT Press (2004) 49–56.

[76] D. Maiorca, P. Russu, I. Corona, B. Biggio, G. Giacinto, *Detection of Malicious Scripting Code through Discriminant and Adversary-Aware API Analysis.* In Italian Conference on Cybersecurity (ITASEC), Venezia, Italy 2017.

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

[78] H. Zou, T. Hastie, *Regularization and variable selection via the elastic net.* Journal of the Royal Statistical Society, Series B, pagg. 301–320, 2005.

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

[80] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Müller, E. Säckinger, P. Simard, and V. Vapnik. *Comparison of learning algorithms for handwritten digit recognition.* In Int'l Conf. on Artificial Neural Networks, pages 53–60, 1995.

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

List of Publications Related to the Thesis

Published papers

- B. Biggio, G. Fumera, P. Russu, L. Didaci and F. Roli, *Adversarial Biometric Recognition: a review on biometric system security from the adversarial machine-learning perspective*. IEEE Signal Processing Mag., vol. 32, no. 5, pp. 31–41, 2015.

- A. Demontis, P. Russu, B. Biggio, G. Fumera, e F. Roli, *On Security and Sparsity of Linear Classifiers for Adversarial Settings*. In Joint IAPR Int'l Workshop on Structural, Syntactic, and Statistical Pattern Recognition, Merida, Mexico, 2016, vol 10029 of LNCS, pagg 322-332.

- P. Russu, A. Demontis, B. Biggio, G. Fumera, e F. Roli, *Secure Kernel Machines against Evasion Attacks*. In 9th ACM Workshop on Artificial Intelligence and Security, Vienna, Austria, 2016, pagg 59-69.

- D. Maiorca, P. Russu, I. Corona, B. Biggio, G. Giacinto, *Detection of Malicious Scripting Code through Discriminant and Adversary-Aware API Analysis*. In Italian Conference on Cybersecurity (ITASEC), Venezia, Italy 2017.