Transactions on Machine Learning
and Data Mining
Vol. 8, No. 1 (2015) 3-39
ISSN 1864-9734
ISBN: 879-3-942952-33-0

ibai Publishing
www.ibai-publishing.org

Machine Learning in Security Applications

Davide Ariu, Igino Corona, Roberto Tronci, and Giorgio Giacinto

Department of Electrical and Electronic Engineering, University of Cagliari,
[davide.ariu, igino.corona, roberto.tronci,
giacinto]@diee.unica.it

Abstract. One of the most important assets to be protected is information, as every aspect of the life of a society deeply depends on the available information. Nowadays, information is stored, processed, and communicated by computers. It turns out that computers represent the most critical tool in modern society. A number of protection mechanisms are available so far, such as antivirus software tools, and biometric access control systems. For their effectiveness, frequent updates are needed, due to the rapid evolution of attack patterns. In fact, attacks are often devised and spread by running computer programs, which can produce new effective attacks in a short time frame. It turns out that machine learning techniques with their generalization capability are one of the favorite approaches to deploy protection and attack detection mechanisms. In this paper, we discuss the approaches that should be followed when devising machine learning techniques for security applications. In particular, we will focus on testing methodologies, performance measures, and techniques aimed at reducing the intrinsic variability of performance that often machine learning application exhibit in real-world scenarios.

Keywords: Machine Learning, Computer Security, Security Tests, Security by Design

1 Introduction

It is commonly recognized that nowadays we live in an information society as people continuously share opinions, findings, and discuss about their ideas. Internet is at the origins of this revolution that began less than twenty years ago. Modern people also travel a lot worldwide. They do it for both pleasure and work. As a consequence of these aspects, each individual gets continuously in touch with a large number of people whose reciprocal knowledge is often quite limited.

Under these premises, to provide people a feeling of security is fundamental in order to let them living fine within this context. Among all the elements that contribute to generate this feeling, we consider two of them as particularly relevant. One element is the feeling of security provided by the places where people live or spend their time. This is especially true in the case of crowded places, such as airports or train stations. The second one is the trust into the systems people commonly use in their daily activities such as ATM machines, the Internet, the mobile-phone network, etc.

A plenty of different systems and solutions is implemented to provide people security within all these contexts. Video-surveillance cameras, as well as biometric authentication systems (based on fingerprints, iris, etc.) are commonly installed in critical places such as airports or banks. A number of solutions is also implemented to provide secure surfing on the Internet: almost everybody has an anti-virus software installed on his own computer. More critical infrastructures, such as enterprise networks, also deploy firewalling mechanisms for traffic filtering, and Intrusion Detection (and or Prevention) appliances to guarantee network and computer security.

In this scenario of high-risk applications Machine Learning algorithms play a fundamental role, especially in applications related to biometrics and computer security.

1.1 Biometrics

One of the main problems in today's networked society, is to obtain a correct, and reliable verification of the identity of a person. However, "traditional" methods such as passwords, and PIN (Personal Identification Number), are unreliable because a personal code (a sequence of letters or digits) can be stolen or duplicated, and used by other people for illegal aims. In this context, biometrics is a very active research field. Its aim is to find reliable personal identification techniques based on human characteristics such as face, fingerprint, retina, signature, iris, gait and so on. A biometric system assures a more reliable identification of a person, since fingerprint, face etc. are unique for each person and cannot be stolen or easily duplicated [32, 33].

A biometric system can be built to face two different problems: the "authentication" (or verification) of a user, or the "recognition" (or identification) of a user. The authentication refers to the problem of confirming or denying a person's claimed identity. Recognition refers to the problem of establishing a subject's identity, for example in forensic applications.

The most faced problem in daily activities is the "authentication" one:

- at first, a biometry is acquired from a user by the biometric system, and the raw data are extracted (e.g. the image of a fingerprint or a face);
- then, the raw data is enhanced through the use of different algorithms to improve the quality of the data and remove the noise;
- from this enhanced data, the biometric system extracts the features (for example from a fingerprint image features regarding the minutiae can be extracted);
- when the features are available, a matching algorithm is run;
- the matching algorithm, with respect to the identity claimed by the user, compares the extracted features to those stored in the system and associated to the genuine identity. Finally, according to the outcome of the matching process, a decision is taken, and the identity is authenticated or not.

In order for the biometric system to work properly, machine learning algorithms are typically employed. In fact, different aspects have to be taken into account within the identification process: enhancing and parsing the raw data, feature extraction, feature selection, and identity matching [33].

1.2 Computer Security

Reason why machine learning is important for computer security applications are different. At present, the most part of the computer security issues involves Web Applications. The reason for this is that Web-based services and social networking platforms are quite common, and their number is still increasing as the Web-based architecture is the most frequently used in software deployments. The results of a recent study by the X-Force team show that approximately 50% of vulnerabilities discovered during 2010 affected Web applications [31]. In consequence of this, the security of Web applications is a key topic in computer security. In order to protect them, Web Application Firewalls are one of the most frequently used protection tools. Typically, they rely on a set of rules written by the administrator, who therefore must have an in-depth knowledge of the applications to be protected. Even if there are solutions as ModSecurity [1] that offer automated rule update functionalities, tools that rely only on rule-based approaches do not seem to guarantee sufficient protection to Web applications. The main reason is that zero-day attacks are particularly critical for Web applications, because exploiting a vulnerability in an application with a large number of users might allow to quickly infect a large number of victim clients (e.g., the so-called drive by download attacks). In our opinion, anomaly detection approaches based on machine learning techniques allow addressing the limitation of rule-based systems, thus providing for an effective solution.

Anomaly-based systems rely on a model of the normal behavior of Web applications. We share the definition of "normal behavior" provided by Maggi et. al. in [43]: *the term normal behavior generally refers to a set of characteristics (e.g., the distribution of the characters of string parameters, the mean and standard deviation of the values of integer parameters) extracted from HTTP messages that are observed during normal operation*. Starting from this definition, it is quite straightforward defining "anomalous" all those behaviors that significantly deviate from the statistical model of the normal activity. Obviously this allows anomaly based IDS to fight off also zero-day attacks. Initially, the main obstacle to the large scale deployment of anomaly based solutions has been the too high false positive rate, as not all the detected anomalies are actually related to attack attempts. Nowadays anomaly detection mechanisms are also deployed within some commercial products [2–4].

1.3 Contributions of this work

This book focus on the problem of Machine Learning Methods and the problem that arises from the Standardization point of view when systems are learnable. From this point of view, in the scenario of high-risk applications, is difficult to define "standards". This is a consequence of the constant rapid evolution of sensor and algorithms (e.g.

some times a solution to a specific problem have a short life time), and to the fact the context of security problems are in a continuous evolution.

In the rest of this chapter, we will make use of the following definitions:

– A *Biometric Access Control System* (shortly a “*Biometric System*”) will be intended as a system that disciplines the user access to a given asset by using biometrics (e.g. a fingerprint, the face, the iris). For the purpose of this analysis the nature of the asset to be protected will be not of interest: then, it could be indifferently a computer system, a bank account, or even a restricted area of a building.
– A “*Computer Security System*” will be meant as a system aimed at detecting or preventing software attacks against a single computer or a computer network. Without any regard of the compromised system, attacks can be distinguished depending on their effects. Very common examples are those attacks that can guarantee unauthorized access to a given resource (e.g. unauthorized access to a bank account), or that can compromise the availability of a given resource (a “denial of service” attack), or that can hijack the activity of a computer system (e.g. a malware infects a computer and opens a back-door for malicious remote access). *Intrusion Detection* (and/or *Prevention*) *Systems*, *Anti-virus*, and *Firewalls* are the Computer Security Systems the most commonly deployed for protecting computer networks.

The rest of this chapter will be organized as follows. Section 2 will discuss the issues concerning the application of Machine Learning algorithms in security applications. Section 3 will describe the performance measures typically used for the evaluation of biometric and computer security systems. Section 4 will provide an overview of the possible sources of variability for the performance of systems working in security-related applications. Section 5 will introduce the *Multiple Classifiers Systems* paradigm and will show some of the most commonly used combination strategies. Section 6 will provide the experimental results that concretely show which benefits can descend (in terms of performance) from using multiple classifiers in security applications. Finally, conclusions will be drawn in Section 7.

2 High Risk Applications in Machine Learning

In this section, we discuss the issues that must be taken into account when machine learning algorithms are applied to high-risk applications. Machine Learning algorithms are implemented within both *Anomaly-based (Computer Security) systems*, and *Biometric Systems* to perform a statistical Pattern Recognition task.

Unfortunately, the deployment of these systems in real-scenarios requires them being able to meet hard and often conflicting constraints. In particular, *Computer Security systems* are typically required to generate a very low number of false alarms and, at the same time, to be very accurate in detecting attack attempts. On the other side, “usability” is particularly important in biometric systems, since users can not be asked to follow too complicated procedures in order to make themselves recognized by the system. Obviously, usability must be achieved without affecting the capability of the biometric system to provide a strong and reliable authentication mechanism.

These requirements can be successfully met only if the system designer carefully considers several critical issues related to the choice of the most suitable statistical model used to represent (and to solve) the problem. In particular, the issues we identified are the following:

1. **What is the most appropriate model for the problem?** In security applications we typically want to assess if a given object has to be considered as "normal" (that is legitimate) or as "anomalous" (that is as an attack). In order to solve this problem we can:
   - Make use of our knowledge of both the normal and anomalous patterns. In this case the system learns the differences among the two classes of patterns, and creates a rule that allows separating these two classes as better as possible. A similar approach is suitable for malware detection applications, where we can easily obtain both large amounts of malware samples and "benign" (that is non malicious) executables. Therefore, since the availability of a large amount of samples from both populations is not a problem, we can easily represent the problem with a **two-classes model** [56]. For what concerns biometric, "recognition" problems too arise to this category, since the identity of a single user is checked against those of all the remaining users into the database.
   - Make use of our knowledge of the normal class only. In this case the system creates a statistical model of the normal class: all those patterns which distance from normal patterns exceeds a pre-defined threshold are labeled as anomalous. A typical case is that of web applications security, where it is quite simple to collect large amounts of "legitimate" patterns whereas it is indeed more difficult to obtain representative datasets of attacks, given that attacks are specific for each application. In a similar situation a **one-class** model that is built on the basis of legitimate requests only represents the most reasonable choice. Patterns labeled as attacks are those that are statistically too diverse from the normal ones [14].
2. **Which kind of pattern we should look at to detect attempts of intrusion?** Let us assume that we want to build a biometric system to restrict the access to a particular area of a building. In biometrics it is well known that looking at the iris or at the fingerprint is a good way to verify the identity of a person [33]. It is also known that both iris and fingerprints are definitely better than other biometric properties, such as for example the voice or the hand geometry [33]. And this is substantially due to the "amount of information" about the identity which is larger in iris and fingerprint with respect to that within other biometric properties. But what about Intrusion Detection? If we want to protect a web application it is better to look at the HTTP traffic incoming to the web server or to monitor the log files? Whatever the choice, it seems a quite artificial dilemma because neither the network traffic nor the web server logs are an "inner property" of the web application such as fingerprints for a person.
3. **Given a pattern, what is the best choice of features?** Obviously this is a concern not only for anomaly-based IDS but it is a general question which involves the design of every Pattern Recognition System. Suppose that we decided to monitor the network traffic toward a web server to detect attacks against web applications that

it is hosting. It is enough to model only the HTTP payload or should we consider also informations within the IP header? Assumed that we decided to model just the HTTP payload, is an analysis of the bytes' distribution enough or should we put into the model the a priory knowledge about the structure of the HTTP payload? This is not a trivial question to answer. In general, the more a priory knowledge is used, the more accurate the resulting system is in classifying patterns. It is worth noting, however, that the amount of false alarms and the detection rate are not the only parameters that must be considered in the evaluation of an IDS. For example a network based IDS has to meet severe real time constraints, and this means that the representation of the pattern into the features space can't be computationally too expensive. Anyway a discussion of the choice of features goes beyond the focus of this work.

4. **Which is the algorithm the most suitable?** The choice of the most suitable algorithm to separate normal and malicious patterns is tricky and heavily depends on the features chosen. Typically, the two options are supervised, and unsupervised algorithms. Successful applications of both supervised [55] and unsupervised [76] methods exist. For example, in [40] the authors show that in network intrusion detection problems, supervised methods provide superior performances. Once a choice has been made between a supervised or an unsupervised approach, a further choice has to be made for a specific algorithm: a variety of alternatives exists. The expertise of the designer together with a clear understanding of the problem domain, and with a in-depth knowledge of the algorithms, are the only factors that can drive this choice.
5. **Is a single classifier sufficient or should the IDS be designed using an ensemble of classifiers?** Classifier ensembles are generally used to increase the classification accuracy with respect to that of a single classifier. The price that must be paid for this gain in accuracy is an increased complexity of the resulting system. This complexity might cause, for example, an increase of the computational cost. In security-related applications this might become a critical issue: a system such as a network IDS must be able to keep up with the network speed; a biometric authentication system must be able to produce a decision within the time frame allocated for the task. Thus, the trade-off between complexity and computational cost must be carefully evaluated.
   A critical review on the popularity of the approach based on multiple classifiers has been expressed by Ho [29]:

   > Instead of looking for the best set of features and the best classifier, now we look for the best set of classifiers and then the best combination method. One can imagine that very soon we will be looking for the best set of combination methods and then the best way to use them all. If we do not take the chance to review the fundamental problems arising from this challenge, we are bound to be driven into such an infinite recurrence, dragging along more and more complicated combination schemes and theories and gradually losing sight of the original problem.

   Nevertheless, the combination of classifiers has been deeply investigated and many successful applications exist in fields such as intrusion and spam detection or biometrics [10,15,19,23,26,35,39,48,55]. In biometrics, intrusion and spam detection, the employment of multiple classifiers is further motivated by the fact that not only the classification accuracy, but also the robustness against attempts of evasion is a crucial parameter to evaluate a system. Usually, it is more difficult for an attacker to evade multiple classifiers instead of a single one. In Section 5 we will provide a brief description of Multiple Classifiers Systems and we will illustrate how they can improve performances in high-risk applications.

2.1 One vs. Two-classes Pattern Classification

The aim of a security system is generally that of detecting (or even blocking) any kind of malicious activity leaving as much as possible undisturbed all the normals. Within the computer security community the malicious objects are usually referred to as "*attacks*", whereas in biometrics the term"*impostors*" is typically used. Malicious objects are also referred to as"*Positive*" patterns both in computer security and in biometrics. On the other hand, the legitimate (that is "*negative*") patterns are usually referred to as "*normal*" patterns in computer security, and "*genuine*" in biometrics.

Basically, every statistical pattern recognition system requires two different phases:

– A **training** or **learning** phase, where the parameters of the models are estimated.
– An **operational** phase, that is the phase where a system performs its activity in a real world scenario, and provides the desired functions (attacks detection, in the case of an IDS, or user authentication, in the case of a biometric system).

During the training phase, the models' parameters are estimated based on a population of examples that represent the objects the system has to classify. In principle the training set, i.e., the data used in the training phase, should provide a good representation of the real data: that is it should contain a number of samples of each class large enough to obtain a good estimate of the distribution of the real population. This is generally true for what concerns patterns of the normal class, since in computer security it is quite simple to obtain large volumes of samples of legitimate patterns (e.g., network traffic traces, web-server log files). Unfortunately this is not always true in the case of the positive class: we mentioned in the previous section that might be reasonably simple collecting samples of malware whereas to collect attacks against a web application might be not so easy. This might be true also in the case where it is actually possible to collect a large number of malicious samples, but this number is negligible with respect to the number of possible attacks.

Thus, depending on the availability of attack samples, two different approaches can be adopted to train the classifier:

– If we have enough samples of the positive and negative classes, a **two-classes** model might be employed. During the training phase, the classifier models both the normal and the attack class. During the detection phase, the pattern is assigned to one of the two classes.
– If we do not have enough samples for the attack class, a **one-class** model might be employed. During the training phase, the classifier learns how to model patterns belonging to the normal class. During the detection phase, it estimates if the pattern belongs to the normal class and, if it does not, it is considered as **anomalous**.

As a consequence of the above discussion, it should result clear why an anomaly based system is able to deal and identify never-before seen attacks. The reason is that the system does not make any assumption about the distribution of the attacks within the features' space, but it only represents the distribution of legitimate patterns. Then, during the detection phase, it labels as anomalous all the patterns that do not fit within the model created for this distribution.

In the following subsections 2.1 and 2.1 we will discuss in more details the two-classes and one-class classification approaches.

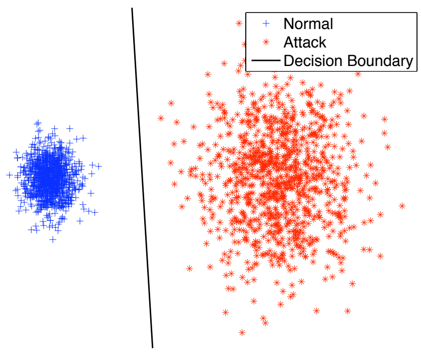


Fig. 1. A possible representation of the problem of Intrusion Detection as a two class problem. In this case the two distributions of patterns are perfectly separable with a linear classifier.

Two-classes Pattern Classification In figure 1 we propose a possible representation of a two class problem. We have two populations of samples, one representing normal patterns, and the other representing attacks.

During the training phase, the classifier estimates a decision boundary which separates as better as possible the two classes. In this example, the problem is quite simple given that even a linear classifier can separate the two classes perfectly. Unfortunately, the scenario depicted in 1 is almost unrealistic. In fact, in real cases a certain overlap usually exists between the two classes we aim to separate.

An additional example is presented in figure 2. Here the two distributions are not linearly separable and a more sophisticated classifier is necessary to separate the two classes. By using the boundary drawn in the figure, the classifier results quite accurate in separating the two classes. In fact, only two patterns out of two thousands are misclassified. In spite of this result (that would be great for a large number of real applications), a system designer should be aware of several issues that, once the system is deployed in the real environment, can affect the performance estimated during the training phase. In order to better clarify this point, let us focus on Figure 3 which depicts a detail of the features' space taken from Figure 2.

What is important to notice is that there are several points from both classes that are classified correctly but that are very close to the boundary. We indicated patterns

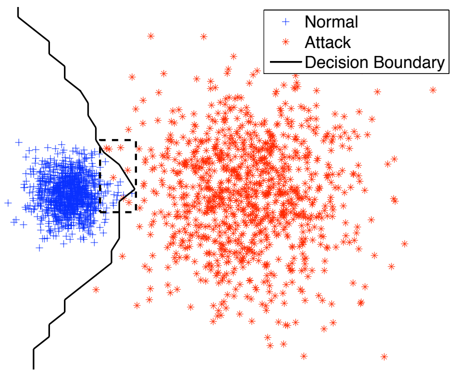


Fig. 2. A possible representation of the problem of Intrusion Detection as a two class problem. The two classes are separated with a K-Neirest Neighbor classifier. A more detailed view of the region within the rectangle is proposed in figure 3.

from the normal class that are in this situation such as *False Positive Candidates*. The presence of these points close to the boundary means that a little change in the behavior of normal patterns might produce a high increase in the rate of false positives. In the case of a network-based Intrusion Detection System (IDS) this might be a very risky situation, since that, with high volumes of traffic, the amount of false alarms might become very large. In addition, an attacker might decide to exploit this situation sending "fake attacks" that are not dangerous at all but that make the IDS generating a huge number of alarms [74]. Despite this is not strictly dangerous since these attacks do not produce any damage, the false alarms represent noise that an attacker could cause to masquerade the real attacks.

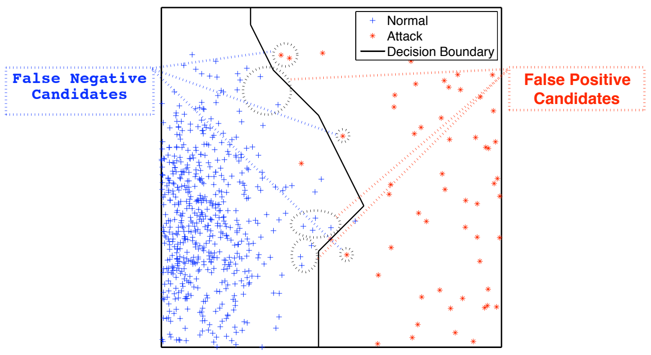


Fig. 3. A detail of the figure 2. The presence of several points very close to the decision boundary indicates that the system is neither resilient to attempts of evasion nor robust respect to variations of the normal patterns.

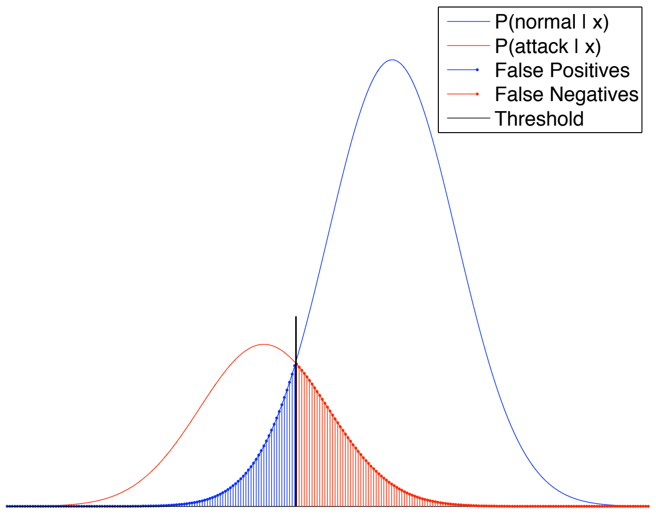


Fig. 4. Posterior probabilities distribution for a two-classes one-dimensional problem.

On the contrary, the presence of attack patterns close to the boundary is dangerous for what concerns the robustness of the system against attempts of evasion. We indicated these patterns as *False Negative Candidates*. The proximity of these attacks to the boundary represents for an attacker a great opportunity of evading the system with little modifications of the attack patterns. A practical example is that of "*polymorphic engines*" (e.g. CLET [18]) that can be used to modify the statistical properties of intrusive network traffic so that they resemble those of the normal traffic and are consequently able to evade a network-based IDS.

A simple mathematical model of the two-class problem is that based on the *Bayesian Decision Theory* [20]. Class-labels are assigned by a *Bayesian classifier* on the basis of the *a posteriori* probabilities. Given a generic class "**c**" and a pattern "**x**" the a posteriori probability of **c** given **x** is the *probability of having the class* ***c*** *given that the pattern is* ***x***. Basically, the a posteriori probability (also known as *posterior*) indicates how much the pattern **x** is likely to belong to the class **c**.

In an intrusion detection problem formulated as a two-class problem, the two possible classes are obviously "**normal**" and "**attack**". According to the *Bayes decision rule* if

$$P(\textbf{normal}|x) > P(\textbf{attack}|x) \tag{1}$$

x is assigned to the normal class; otherwise it is assigned to the attack class.

The probability of error for the rule is:

$$P(\textbf{error}|x) = min[P(\textbf{normal}|x), P(\textbf{attack}|x)] \tag{2}$$

An example of posteriors for a simple one-dimensional problem is proposed in figure 4. From the figure it is possible to infer easily how the false positive and negative rates variates in consequence of the threshold. Moving the threshold toward the attacks' posterior, the false positive rate reduces but increases the number of unrecognized attacks. On the contrary, a threshold which moves in the direction of the normal class posterior will increase the detection rate but also the amount of false positives.

One-class Pattern Classification One-class classification techniques are particularly useful in the case of *two-class* learning problems whereby one of the classes, referred to as *target class*, is well-sampled, whereas the other one, referred to as *outlier class*, is severely under sampled. The low number of examples from the outliers class may be motivated by the fact that it is too difficult or expensive to obtain a significant number of training patterns of that class [67]. The goal of one-class classification is to construct a decision surface around the examples from the target class in order to distinguish between *target objects* and all the other possible objects, i.e., the *outliers* [67].

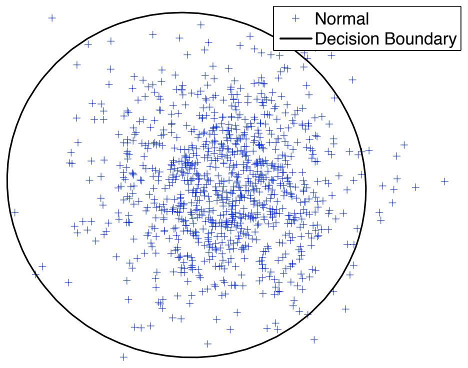


Fig. 5. A possible representation of the problem of Intrusion Detection as a one class problem. A closed surface is drawn around the distribution of normal patterns which leaves outside a certain fraction of rejected samples.

Given an unlabeled training dataset that is deemed to contain mostly target objects, a *rejection rate* is usually chosen during the training phase so that a certain percentage of training patterns lies outside the constructed decision surface. The rejection rate takes into account the possible presence of noise (i.e., unlabeled outliers), and allows us to obtain a more precise description of the target class [67]. In the case when the training set contains only "pure" target patterns, this rejection rate can be interpreted as a *tolerable* false positive rate. This situation is represented in figure 5.

The decision boundaries obtained with two different classifiers are compared in figure 6. The "Decision Boundary - 1" is obtained using a quadratic discriminant classifier which realizes a closed surface around the distribution of normal patterns. The "Decision Boundary - 2" is obtained using a polynomial of 3rd degree. Obviously neither of the two classifiers can do anything against the attacks that falls exactly over the distribution of normal patterns. Additional features would be necessary to detect these attacks. Nevertheless, the closed surface realized by the quadratic classifier is by far better than the decision boundary drawn by the polynomial.

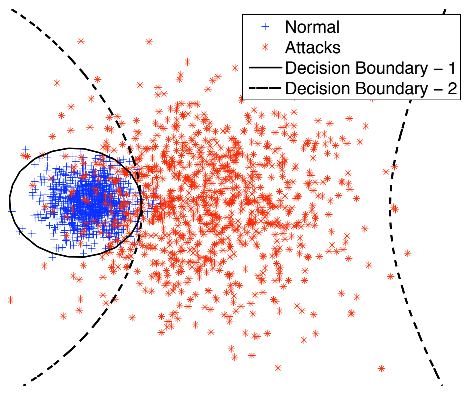


Fig. 6. A possible representation of the problem of Intrusion Detection as a one class problem. Two different decision boundaries are represented, belonging to a *quadratic discriminant classifier* (*Boundary -1*) and to a *polynomial of 3rd degree* (*Boundary - 2*) respectively. The figure shows that *Boundary - 1* guarantees a higher robustness against attempts of attack.

The main problem with the polynomial classifier is that the region assigned to the normal class is considerably wider than the space effectively covered by normal patterns. As a direct consequence of this, an attacker that is trying to evade the system is not strictly required to create attacks that appear similar to legitimate patterns: what is important to achieve evasion is that the attack falls into the region assigned to normal patterns. Obviously, the larger this region, the easier the task of evading the system. On other side, to evade a system based on the quadratic classifier an attack has to fall exactly over the distribution of normal patterns.

A possible issue related to the choice of the quadratic classifier is the limited tolerance against changes in the distribution of legitimate patterns. If this change occurs, a decision surface such as that drawn by the quadratic classifier easily lets the system generating large numbers of false alarms. The solution to this is usually represented by system re-training. In Section 4 we will say more on this while talking about *Moving Targets*.

3 Evaluation of Pattern Recognition Systems for Security Applications

Evaluating pattern recognition systems in security applications is an inherently difficult task. We can identify several issues:

– a statistically representative training set must be obtained; that is, patterns within the training set should be representative of patterns that will be encountered during the operating phase. Even if this aspect is in common with every pattern recognition problem, security applications make this task more challenging, due to:
  - **undersampled data**: some pattern classes may be severely undersampled depending on the security application. For instance, this may be the case of intrusive patterns, since intrusions are typically “rare” events.

  - **time-variant data**: the statistical distribution of representative patterns may evolve rapidly, and new features may be required (e.g. once new intrusion techniques have been discovered).
  - **privacy and authorization issues**: training data employed by security applications is often privacy-sensitive. Thus, obtaining and sharing such data may be difficult, as it requires authorization, and its storage requires privacy-safe databases. For instance, traffic towards web applications may contain confidential data about web users (e.g. passwords, personal data, credit card numbers).
- a clever, adaptive adversary, who aims to evade or divert the expected behavior of the pattern recognition system, should be taken into account. For instance, an adversary may exploit any vulnerability in the decision boundary of a classifier to evade it. Moreover, she may target a classifier at training phase, by inserting "poisoning" patterns within training data, to mislead the learning algorithm. Pattern recognition systems for security applications should be robust to such attacks.
- it is difficult to identify "standard" evaluation metrics, because these metrics may change according to application-specific constraints and challenges.

With these challenges in mind, Section 3.1 aims to outline the general metrics to the evaluation of classification accuracy in security applications. These metrics can be viewed as "best practices" for performance evaluation, as they are widely accepted by the scientific community working on pattern recognition and computer security. We will employ such metrics for the experimental evaluation of security solutions described in Section 6.

3.1 "Two-class" problems and ROC Curves

In Section 2 we have pointed out that high risk applications can be formulated in terms of a *one-class* or a *two-class* Machine Learning problem. This aspect regards only the classification approach used, but from the evaluation point of view they can be both treated as a *two-class* problem. These two classes are usually denoted with the terms *positive* class **p** (i.e., the class of patterns we want to locate), and *negative* class **n** (i.e., a "true" class in the two-class approach, and the "the rest of the world" in the one class approach).

Thus, in the evaluation process of a *two-class* problem for a generic pattern y, four possible outcomes can be obtained [22]:

- y is a *positive* pattern and it is classified as belonging to the *positive* class, then it is a *true positive*
- y is a *positive* pattern and it is assigned to the *negative* class, then it is a *false negative*
- y is a *negative* pattern and it is assigned to the *negative* class, then it is a *true negative*
- y is a *negative* pattern and it is assigned to the *positive* class, then it is a *false negative*

If we measure the number of patterns falling in each of the above four cases, different metrics can be derived:

$$true\ positive\ rate\ (TPR) = \frac{positives\ correctly\ classified}{total\ positives}$$

$$false\ positive\ rate\ (FPR) = \frac{negatives\ incorrectly\ classified}{total\ negatives}$$

$$true\ negative\ rate\ (TNR) = \frac{negatives\ correctly\ classified}{total\ negatives}$$

$$false\ negative\ rate\ (FNR) = \frac{positives\ incorrectly\ classified}{total\ positives}$$

$$precision = \frac{true\ positives}{true\ positives\ +\ false\ positives}$$

$$accuracy = \frac{true\ positives\ +\ true\ negatives}{total\ positives\ +\ total\ negatives}$$

$$specifity = \frac{true\ negatives}{true\ negatives\ +\ false\ positives}$$

it is worth noting that

$$TPR + FNR = 1$$

$$TNR + FPR = 1$$

Moreover, depending from the applications at hand, different terminology are used. In some Pattern Recognition and Machine Learning application the terms "hit rate" or "recall" (i.e., the *true positive* rate), and "false alarm rate" (i.e., the *false positive* rate) are used. In biometric authentication, the terms "false non matching rate" or "false rejection rate" (i.e., the *false negative* rate), "false matching rate" or "false acceptance rate" (i.e., the *false positive* rate), and "true matching rate" or "genuine acceptance rate" (i.e., the *true positive* rate) are used, as they are obtained by comparing the unknown pattern y to a known pattern x_i to verify if their identities match (if so, the pattern correspond to a "genuine" user) [32, 44].

In this kind of problems the outcome of a classifier can be of two types: in the case of a two-class formulation, a pattern is assigned either to the (*positive* or *negative*) class, while in the case of one-class formulations, a similarity score (or rank) is assigned to each pattern with respect to patterns belonging to the *positive* class. In the latter case a class can be assigned to the pattern by setting a decision threshold th: if the score is greater then th than the pattern is assigned to the positive class, otherwise to the negative one, it is important to notice that the decision threshold can be set to fulfill specific constraints. The use of a similarity score is the one generally used in the case of high risk real-world applications as it allows to fix a decision threshold in agreement with the trade-off imposed by the required level of security.

To assess the global performance in *two-class* problems, the Receiver Operating Characteristic curve (ROC) is the generally used. This curve plots the *true positive* rate against the *false positive* rate as the decision threshold th varies along the score range [22]. Other two well used performance measures related to the ROC curve are: the Area Under the ROC Curve (AUC), and the Equal Error Rate (EER). The AUC summarizes the performance for all the values of the decision threshold.

$$AUC = \int (TPR(th))\mathrm{d}FNR(th)$$

it is worth noting that the larger the AUC, the better the ROC [11]. The EER represents the point of the ROC curve where the *false positive rate* and the *false negative rate* are equal, and a good system should keep this value as small as possible. The relations between the ROC, the AUC, and the EER are illustrated in Figure 7.

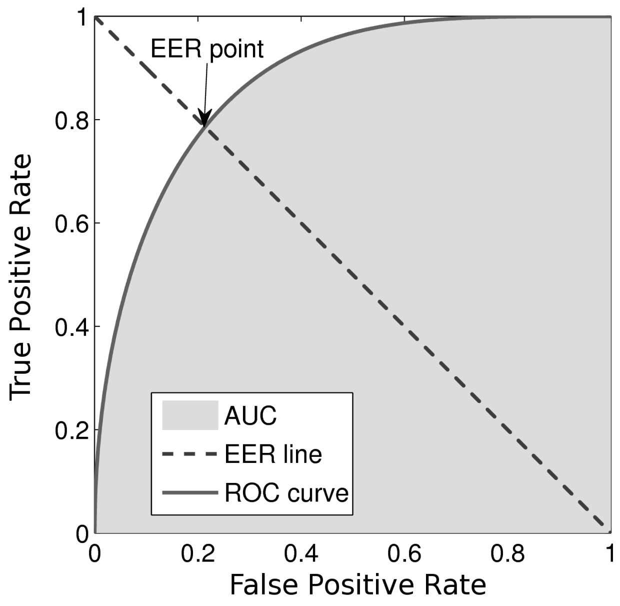


Fig. 7. An example of a ROC curve, its AUC and its EER.

One way to compute the AUC is to use the formulation based on the the Wilcoxon-Mann-Whitney statistic [28,45] instead of computing a numeric integral of the curve:

$$AUC = \frac{\sum_{a=1}^{n_+} \sum_{b=1}^{n_-} I(s_{a,j}^{\mathbf{p}}, s_{b,j}^{\mathbf{n}})}{n_+ \cdot n_-} \tag{3}$$

where n_+ is the number of *positive* patterns and n_- is the number of *negative*, **p** and **n** indicate the scores of the patterns belonging to the *positive* and the *negative* class respectively, and the function $I(s_{a,j}^{\mathbf{p}}, s_{b,j}^{\mathbf{n}})$ is:

$$I(s_{a,j}^{\mathbf{p}}, s_{b,j}^{\mathbf{n}}) = \begin{cases} 1 & s_{a,j}^{\mathbf{p}} > s_{b,j}^{\mathbf{n}} \\ 0 & s_{a,j}^{\mathbf{p}} < s_{b,j}^{\mathbf{n}} \end{cases}$$

Moreover the AUC can be statistically interpreted as follows: given two randomly chosen users, $X^{\mathbf{p}}$ belonging to the set of the positive patterns and $Y^{\mathbf{n}}$ belonging to the set of the negative, the AUC is the probability $P(X^{\mathbf{p}} > Y^{\mathbf{n}})$, i.e. the probability of correct pair-wise ranking [28].

All the measurements illustrated so far can be used for assessing the performance in a two-class problem. In particular, the ROC and AUC measures globally indicate how good the performance of a system are.

Usually, when dealing with high risk applications, the interest is focused on "local" performances of the classification systems, i.e., the performances measured in a particular range of values of input patterns, rather than on the average performance. To this end, a number of measures have been devised. The EER is a measure typically used to evaluate biometric systems, that takes into account a particular working point, i.e., the point where the errors are balanced (i.e., the FPR and the FNR are equal). Other measures obtained in specific working points are generally used to evaluate biometric systems:

- *1% FPR*: the value of FNR or TPR when the FPR is 1%
- *1% FNR*: the value of FPR when the FNR is 1%
- *0% FPR*: the value of FNR or TPR when the FPR is 0%
- *0% FNR*: the value of FPR when the FNR is 0%

The *1% FNR* is a measure that is also widely used in the assessment of computer security systems.

Another well know "local" measure is the partial AUC (AUC_p). The AUC_p is nothing but the AUC computed in a specific range of FPR. Generally it is used to measure the partial area under the ROC in the range $[0,d]$, where d is the maximum error in terms of FPR that can be accepted for a specific security system.

4 Causes of performances' variability

In this section, we briefly describe all the possible sources of performance' variability of systems based on statistical algorithms. In particular, this analysis will concentrate on the accuracy of the systems in correctly classifying the analyzed patterns.

For the purpose of this discussion, we distinguish the sources of variability in "*internal*" (or intrinsic) and "*external*".

We consider a source of variability as *intrinsic* if it concerns the design (i.e., the architecture) of the system and its implementation and deployment in the real world. In this sense, the performance of the system can be influenced by many factors, such as:

- Classifier parameters setting. In some cases, an initial estimate of the parameters must be provided before the learning phase. This estimate can just be a random choice of the values of the parameters, or it can be calculated on the basis of the application constraints, and of the dataset that will be used to train the classifier.
- In the case of ensemble systems, the number of classifiers in the ensemble also influences the performance of the system. As we will show in section 6.1, performance usually become more stable as the number of combined classifiers increases.
- The strategy used for combining classifiers. It obviously has a strong impact on the accuracy achieved. This issue will be discussed in 5.
- The dataset used to train the system. It must be large enough and representative of the real scenario where the system will be deployed.

On the other side, we consider *external* sources of variability the issues descending from the fact that the distribution of patterns to be classified is non stationary over time. The last one is a well known problem in the statistical pattern recognition literature, and it can be referred as the problem of the "Moving Targets." However, with respect to the traditional definition of non stationarity, i.e., modifications of the statistical distribution of patterns over time, security related problems are affected by the intentional actions of an adversary who craft new patterns in order to mislead the learning phase, and to have them being undetected by the protection system.

It is quite easy to see that the success of pattern recognition applications requires the detailed definition of the objects that have to be classified, and the detailed definition of the characteristics of the classes the objects have to be assigned to. In order to perform classification, measurable features must be extracted so that the classification can be performed on the basis of the values of the features. In this sense, among security application scenarios, the detection of computer attacks is actually one of the most challenging problems in security scenarios for three main reasons. One reason is related to the difficulty in predicting the behavior of software programs in response to every input data. Software developers typically define the behavior of the program for legitimate input data, and design the behavior of the program in the case the input data is not correct. However, in many cases it is a hard task to exactly define all possible incorrect cases. In addition, the complexity and the interoperability of different software programs make this task extremely difficult. It turns out that software products always exhibit weaknesses, a.k.a. vulnerabilities, which cause the software to behave in an unpredicted way in response to some particular input data. The impact of the exploitation of these vulnerabilities often involves a large number of computers in a very short time frame. Thus, there is a huge effort in devising techniques able to detect never-seen-before attacks. The main problem is the exact definition of the behavior that can be considered as being normal and which cannot. One of the reasons relies in the fact that the vast majority of computers are general-purpose. Thus, the user may run different kind of programs, at any time, and switch among them in any fashion. It turns out that the normal behavior of one user is typically different to that of other users. In addition, new programs and services are rapidly created, so that the behavior of the same user changes over time. Finally, as soon as a number of measurable features are selected to define the normal behavior, attackers are able to craft their attacks so that it fits the typical feature values of normal behavior. The above discussion, clearly show that the target of attack detection tasks rapidly moves, as we have an attacker whose goal is to be undetected. As a consequence, each move made by the defender to secure the system can be made useless by a countermove made by the attacker [65]. It is also worth noting that the defender has a partial knowledge of the attack, as the attacker may be able to craft the attacks in a way that drives the defender to select accidental features of the attacks as the most discriminative features [54]. The rapid evolution of the computer scenarios makes the detection problem quite hard, as the speed of creation and diffusion of attacks increases with the computing power of today machines.

Apart from the above peculiarities of computer security, security systems must be designed by taking into account the characteristics of the adversarial environment in which they will operate. This is quite clear when physical security systems are designed, while this is a quite new topic in the field of machine learning systems for new security applications such as computer security and biometric authentication. In the following subsection we will briefly summarize the major threats against machine learning systems that are originated by the so-called "adversarial environment".

4.1 Machine Learning Systems in an Adversarial Environment

Security tasks like intrusion detection in computer networks and biometric authentication can be viewed as adversarial classification tasks [17], i.e., tasks where intelligent, malicious, and adaptive adversaries can manipulate their samples to mislead a pattern recognition system. Adversaries can either exploit vulnerabilities in its learning algorithm (poisoning [50] or causative [7] attacks) or imprecisions in its decision boundary (classification attacks). In the following, we briefly discuss about these two classes of attacks with reference to intrusion detection and biometric authentication.

Poisoning attacks. In intrusion detection skilled adversaries may inject poisoning patterns to mislead the learning algorithm which infers the profile of legitimate activities [38] or intrusions [13, 57]. In biometric authentication systems adversaries may spoof biometric traits in order to mislead template update methods (e.g. gradually "inject" erroneous templates in the system) [34].

Classification attacks. In intrusion detection clever adversaries can modify their attack patterns either to evade a classifier [49, 71], or inject a large amount of false alarms, in order to hide their "real" attacks from security operators, as a needle in a haystack (overstimulation) [52, 75]. Clever adversaries can also spoof biometrics to evade biometric authentication systems [5, 46, 47, 70].

It is easy to see that such adversarial actions can cause performance degradation. In fact, one of the main objectives in adversarial classification tasks is the design of a *robust classifier*, namely, a classifier whose performance degrades as gracefully as possible under attack [17]. A key issue of robustness evaluation is the correct "simulation" of adversarial actions against a pattern recognition system. This analysis is far from being trivial, as it should take into account the characteristics of such a system (e.g., processed data, selected features, learning algorithm, decision function) and the capabilities of adversaries in the *operating environment* where this system will be deployed. This is perhaps the reason why adversarial classification is attracting a growing interest from the pattern recognition and machine learning communities [41].

5 Multiple Classifier Systems

In the previous section we have outlined that, when a pattern recognition system is built, several causes of performance variability should be taken into account. This effect is quite severe in the case of security applications, where typically the performance attained by individual experts does not provide the necessary reliability. For this reason, the Multiple Classifier Systems approach was introduced.

Multiple Classifier Systems (MCS) are widely used in Pattern Recognition applications as they allow to avoid the process of designing/choosing the "best" classifier for a given problem and a given set of patterns [9, 25, 27, 35–37, 39, 51, 60, 61, 63, 73]. In fact, the combination of classifiers typically provide better performance than those provided by individual experts [39]. Moreover, in the combination of classifiers, the single classifiers can be based on different input sources, so that complementary information can be exploited, and the resulting combination is robust with respect to noise [39].

Basically, an MCS exploits the decisions made by an ensemble of classifier, and combines these decision to obtain a "better" classification. According to [19], there are at least three reasons for which an ensemble results more accurate and robust of any classifier in the ensemble:

– The **statistical** reason. A learning algorithm can be viewed as a search for the best hypothesis in a space $\boldsymbol{H}$ of hypotheses. In consequence of the finite size of the training set, the learning algorithm will usually end up with a number of classifiers that achieve the same accuracy on the training data. These classifiers, however, may not produce the same accuracy on unseen data. By constructing an ensemble out all of them, the risk of choosing the wrong classifier can be reduced.
– The **computational** reason. In many cases the optimal training of a classifier is a NP-hard problem: consequently, most learning algorithms usually aim at finding a local optimum of the target function. This optimum usually depends on the starting point. This means that by running the local search from different starting points, and using the obtained classifiers to build an ensemble, a better approximation of the true unknown function can be attained.
– The **representational** reason. In most machine learning applications, the true function for the problem at hand cannot be represented by any of the functions available in $\boldsymbol{H}$. The use of weighted sums of hypotheses drawn from $\boldsymbol{H}$ may allow expanding the space of representable functions.

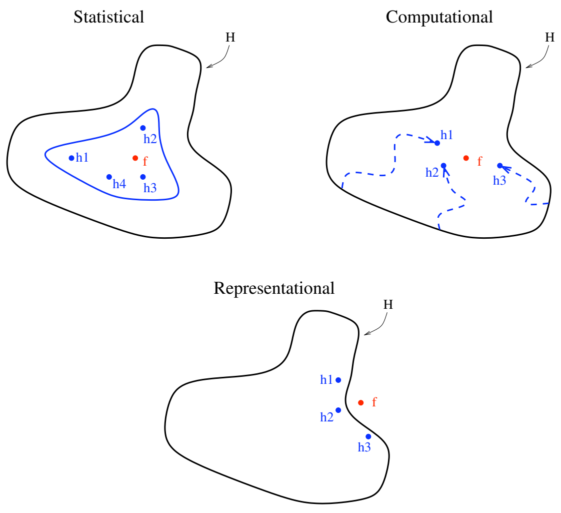


Fig. 8. Three fundamental reasons why an ensemble may work better than a single classifier [19]

The combination rules for building an MCS introduced so far can be distinguished by the combination level and the combination strategy. The combination strategy can be distinguished into two approaches, namely, the “fusion”, and the “selection” approach. “Fusion” mechanisms aim to combine the outputs of different classifiers into a new one. On the other hand, “selection” mechanisms aim to select one of the outputs of the classifiers in the ensemble. Combination can be performed at different levels: the raw data level, the feature level, the score (or rank) level, and the decision level. The raw data combination is performed at sensor/physical level. The feature level combination combines different features sets to a new feature. The decision level combination relays only to the final class decision. Finally, the score level combination combines the score outcomes from the classifier into a new one. In the case of security application the score level is the most used as it allows to combine different combination of sources and classifiers because it relays only on the score (e.g. at raw data level the source and the sensor must be compatible).

In the following we are going to illustrate some combination rules that works at score level.

5.1 Classifier Score Fusion

The basic assumptions of the fusion strategy is that the ensemble of classifiers to be combined are considered as competitive rather than complementary. Thus, this strategy fuses the score outcomes of an ensemble of classifiers to produce a “new” single score, usually different from those produced by the classifiers. A large number of fusion functions is available from the literature, each one with its pros and cons, and different complexity.

In this work we will consider the Maximum, the Minimum, the Mean and the Geometric mean rules [35].

– *the Maximum rule*:
$$s_i^* = \max\{s_{ij}\} \tag{4}$$
– *the Minimum rule*:
$$s_i^* = \min\{s_{ij}\} \tag{5}$$
– *the Mean rule*:
$$s_i^* = \frac{1}{K}\sum_{j=1}^{K} s_{ij} \tag{6}$$
– *the Geometric mean rule*:
$$s_i^* = \left[\prod_{j=1}^{K} s_{ij}\right]^{\frac{1}{K}} \tag{7}$$

These static rules are widely used in Pattern Recognition to combine classifiers because they allow achieving good results in spite of their simplicity. Nevertheless, trained combination rules have been also proposed to better exploit additional knowledge of the domain at hand. As pointed out in [21], combining classifiers using static rules is a suboptimal solution, whereas trained combination rules are asymptotically optimal.

Despite this, in this work we used static rules for two main reasons. One reason is related to the issues involved in building a trained combiner that make its design a non trivial task especially in high security applications. The other reason is that static rules are very fast to be computed, and thus the additional computational cost is very small compared to the one typically required by trained combination rules.

5.2 Classifier Score Selection

Classifier Selection is based on the assumption that each classifier in a given ensemble exhibits a higher "expertise" than others on a subset of patterns. For each pattern to be classified, the system selects the classifier which is considered to provide the highest accuracy for the pattern at hand. It is easy to see that the main difficulty with this approach is the development of the selection criterion. On the other hand, it can be easily shown that if the selector works properly, very high accuracy can be attained.

This selection could be "static" or "dynamic". With the term "static" we mean that once the classifier is selected it will be used for all the patterns to be combined. Instead with the term "dynamic" we mean that the classifier is selected according with the patterns to be combined. A general schema of a selection strategy is shown in Figure 9.

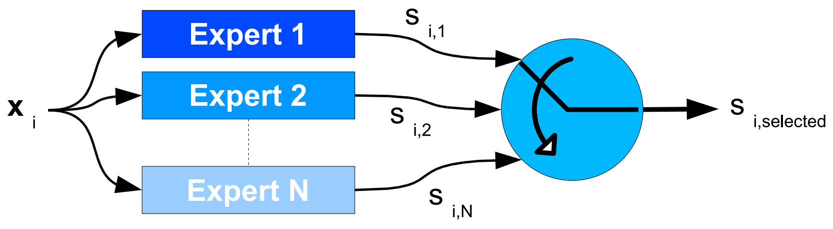


Fig. 9. A general schema of an score selection system.

In the case of combining similarity scores it is difficult to assess witch are the optimal performance achievable with a selection rule. For this purpose the "Ideal Score Selector", proposed in [68], represents the upper bound of the selection strategy. The output of such Ideal Score Selector can be computed as:

$$s_i^* = \begin{cases} \max\{s_{ij}\} & \text{if } x_i \text{ is a } \textit{positive} \text{ pattern} \\ \min\{s_{ij}\} & \text{if } x_i \text{ is a } \textit{negative} \text{ pattern} \end{cases} \qquad (8)$$

It can be seen that the selector is "ideal" as the selection function requires the knowledge of the true class the pattern belongs to.

An example of the results attained by the Ideal Score Selector is shown in figure 10, where two classifiers are combined. In particular, for each classifier the distribution of the output values for the two classes is shown. It is easy to see that the distribution of the output values of the Ideal Score Selector allows a better separation between the classes with respect to each of the combined classifiers.

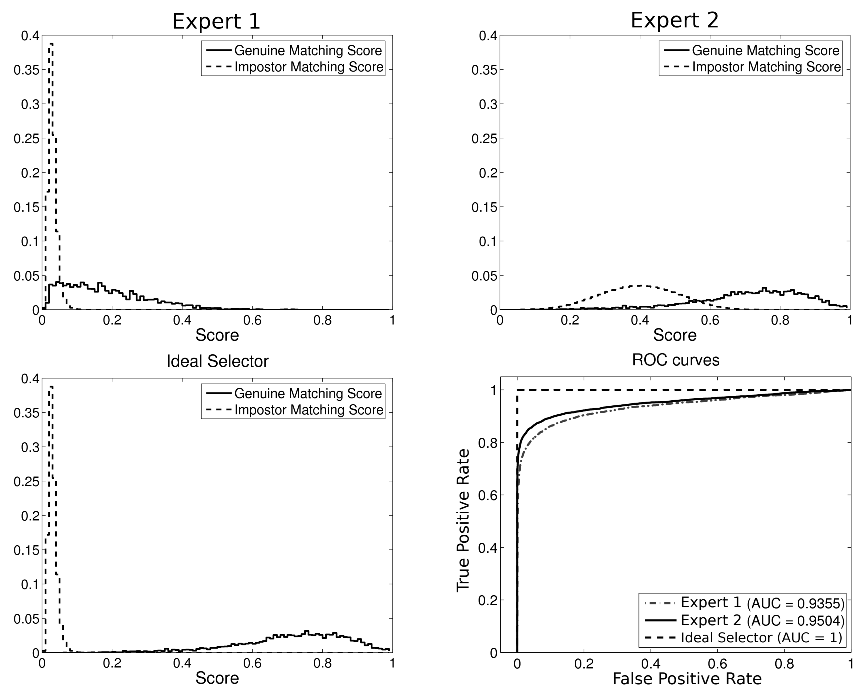


Fig. 10. An example of *ideal score selector* with two biometric experts from a real dataset.

It can be easily seen that the above Ideal Score Selector exhibits a better ROC curve than the ROC curves of each individual classifiers used in the combination, and consequently a larger AUC [68]. Moreover, it has also been shown that the Ideal Score Selector always attains a larger AUC than that obtained by the linear combination, whatever the value of the weights, and the number of classifiers [68].

6 Performance Evaluation of Machine Learning Techniques for Computer Security and Biometric Authentication

In this Section, based on our experience in the field, we will address a number of practical issues related to the evaluation of performance of machine learning systems for computer security and biometric authentication systems. In Section 3 we presented the performance evaluation metrics which are typically employed in two-class problems. In the following, such metrics are used to evaluate Multiple Classifier Systems (MCS) for different security applications. In fact, MCS actually represent a successful approach when security related applications are involved. The rest of the Section will provide a number of arguments to support this claim. In Section 6.1 we show that systems based on multiple classifiers can achieve results, in terms of classification accuracy, higher than those achieved by the classifier combined independently considered. To this end, we show the experimental results achieved with *HMMPayl* [6]. *HMMPayl* is a network-based Intrusion Detection System designed for detecting attacks against a web-server (and the hosted web-applications) trough the analysis of the HTTP traffic toward the server. In Section 6.2 we show that a thorough choice of score combination rules can significantly enhance MCS performance. As a case of study, we will refer to a biometric authentication system. Finally, in Section 6.3 we analyze the problem of poisoning attacks. We show that MCS based on weighted bagging can significantly reduce performance degradation caused by such attacks. To this end, we refer to an intrusion detection system (IDS) for web services.

It is interesting to note that performance evaluation may focus on one or more of the measurements described in Section 3, depending on the application. For instance, a false positive rate higher than 1% is often unacceptable in intrusion detection problems. So, intrusion detection performance is typically evaluated using the partial AUC. On the other hand, the EER is often used in biometric authentication problems, as it is important to verify which is a reasonable trade-off between the false positive rate and the false negative rate. Moreover, other specific working points are explored in biometrics with the aim of finding how the designed system works in different security constraints. These evaluation differences are quite intuitive, because each application may reflect different operating constraints and challenges in a real environment.

6.1 Number of Classifiers and MCS Accuracy

As we already mentioned, *HMMPayl* [6] is a network based IDS for the detection of the attacks against a web server. *HMMPayl* relies on the assumption that HTTP-requests carrying attack attempts appear different with respect to the normal traffic from the point of view of the bytes' distribution. In particular, *HMMPayl* uses Hidden Markov Models [59] to represent the bytes' distribution of legitimate HTTP payloads. It is worth to remind here that the HTTP payload is the portion of the network packet that carries the HTTP-request sent by the client to the web-server.

Hidden Markov Models allows to extract high-order statistics from the payload and to create an accurate model of it. In this sense, *HMMPayl* addresses limitations of previously proposed approaches based on the $n-gram$ analysis [72] or on approximation of it [53].

Hidden Markov Models are trained using the Baum-Welch algorithms [8] which basically relies on the Expectation-Maximization procedure. Starting from an initial estimate of the model parameters, the Baum-Welch algorithm estimates the final value for these parameters by finding a local maximum of the *likelyhood* of the sequences within the training set. The value of this maximum (as well as the final estimate for the parameters), is obviously influenced by the initial value set for the parameters. In pattern recognition applications is quite frequent to set this value randomly, when a criteria can not be clearly identified to estimate the initial value.

Thus, the final behavior of a classifier (roughly speking the "*classification accuracy*" that it is able to achieve) on a given dataset, will obviously depend on the initial parameter estimate. In order to mitigate this effect, a countermeasure which is frequently adopted consists in training several classifiers (starting from different initializations) and then combining them in order to create a MCS.

As *HMMPayl* too follows this approach, interesting considerations can be made by analyzing how the IDS behaves as the number of combined classifiers increases. Results are shown in Figure 11. The figure shows the partial AUC achieved on two different attacks datasets labeled respectively *Generic* and *XSS-SQL*. These datasets contains respectively 66 and 38 common attacks against web-servers and web-applications. The partial AUC has been calculated considering a number of classifiers from 1 to 5. Classifiers are combined using the minimum rule. All the possible combinations of 2, 3, and 4 classifiers have been considered and the average AUC_p has been calculated.

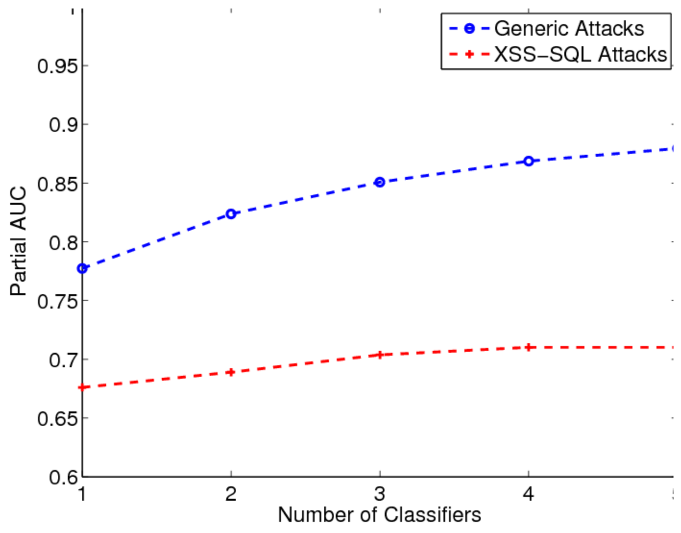


Fig. 11. Increase of the AUC_p with the number of classifiers. The AUC_p is the average of those obtained considering all the possible combinations of 2, 3, and 4 classifiers. Classifiers are combined using the minimum rule.

It is important to observe that the AUC_p increases as the number of classifiers combined. We limited to five the maximum number of classifiers in the ensemble since from some preliminary experiments we observed that five was a good trade-off between accuracy and computational cost. In particular the figure clearly shows that the increase of the AUC_p obtained adding a classifier to the ensemble becomes smaller as the size of the ensemble increases.

In addition, let us consider table 1 where the AUC_p achieved by single classifiers is compared to that achieved using all the five HMM and the minimum rule.

With respect to **XSS-SQL** attacks we can observe that HMM3 and HMM5 perform very poorly. In spite of their presence within the ensemble, the AUC_p achieved by the ensemble is only of a 0.44% smaller of that achieved by the best classifier. This result is particularly interesting if we consider that it is not trivial to establish which classifier will provide the best performance on different test sets. This is clearly shown in table 1 where HMM3 performs very bad on XSS-SQL attacks while it is the second

Table 1. Value of the AUC_p for individual classifiers. The MCS includes all of the five HMM combined with the minimum rule.

	HMM1	HMM2	HMM3	HMM4	HMM5	MCS
Generic	0.8036	0.8163	0.8564	0.8973	0.7944	0.918
XSS-SQL	0.8559	0.8584	0.6155	0.8221	0.6717	0.8546

best (after HMM4) on the Generic dataset. Further on the Generic attack dataset, the AUC_p obtained with the MCS is higher than that achieved by the best classifier in the ensemble.

6.2 Score Combination Rules and MCS Accuracy

In Section 5 we have pointed out that in the case of security applications such as biometrics and intrusion detection the combination is usually performed at the score level. This level of combination is the most used, especially in biometrics, because it allows combining different biometric traits, and different identity matching algorithms at the same time [62]. This aspect is very important in building a more reliable and robust biometric system because every biometric trait has its pros and cons. Thus, we'll show on a biometric dataset how different combination rules increase the performance with respect the single classifiers. Moreover, we remark again that one of the advantages of using an MCS is the one of avoiding to choose the classifier that provided the "best" performances with respect to a given reference set, thus allowing the constructed system to provide acceptable performances in different working conditions.

The experiments of this section have been performed on the *Biometric Authentication Fusion Benchmark Database* (BA-Fusion), a multimodal database of similarity scores artificially created from experiments carried out on the XM2VTS face and speaker verification database [58]. This dataset contains similarity scores from 8 classifiers (different biometric traits and/or different matching algorithms), and the scores have been normalized by the *Tanh* rule [62].

Reported experiments aim at assessing the performance of the proposed techniques in terms of different performance measures. In particular, the AUC, the EER, have been used, as well as error measures at four operating points that are generally used to test security systems, namely FPR 1%, FPR 0%, FNR 1% and FNR 0%. Thus, the FNR (FPR) attained when the FPR (FNR) is equal to 1% or 0% are measured, respectively.

Experiments have been carried out by creating ensembles whose size ranges from 2 to 8, i.e., from the minimum to the maximum ensemble sizes. For each size, all possible classifier ensembles have been considered. In order to get unbiased results, a 4-fold cross-validation technique has been used. The dataset has been subdivided into 4 subsets, so that one subset at a time was used for training, while the other three have been used for testing. Results are reported in terms of average and standard deviation over the four trials, and over all the possible ensemble of classifiers for a given ensemble size. In this way we can show that what we have proven in the previous section holds also for biometrics.

In these experiments the comparisons are made using the following combination rules: the Mean rule as static combination rule, the dynamic linear score combination rule, the Score Decidability Index-mean, the Δ-voting, and the Δ-mean proposed in [42]. Performance are also compared to the best performance provided by the individual classifiers included in the ensemble. It is worth noting that for each measure of performance, the best value for the "best classifier" can be related to a different classifier in the ensemble to be combined.

The dynamic score combination rules are all derived from the logical schema of the ideal score selector, and they exploit the *Score Decidability Index* (SDI) [42] that is related to the likelihood the pattern x_i is drawn either from the positive or negative distributions of scores:

$$\Delta(s_{ik}) = r_+(s_{ik}) - r_-(s_{ik})$$

where $r_-(s)$ represents the probability that the score s is lesser than a score coming from the *positive* distribution, and $r_+(s)$ represents the probability that the score s is larger than a score coming from the *negative* distribution.

The *Dynamic Linear Combination* is defined as:

$$s_i^* = \sum_{k=1}^{\mathbf{N}} \alpha_{ik} \cdot s_{ik} \tag{9}$$

where

$$\alpha_{ik} = \frac{\Delta(s_{ik}) + 1}{2} \tag{10}$$

We will refer to this technique as DLC.

The SDI-mean is computed as:

$$s_i^* = \frac{1}{N} \sum_{k=1}^{N} \frac{\Delta_{ik} + 1}{2} \tag{11}$$

The Δ-voting and the Δ-mean exploit the SDI in the framework of a simplified combination scheme called Dynamic Score Combination (DSC) [69]. The Δ-voting is computed as follows:

$$s_i^* = \beta_{1i} \cdot \max_k(s_{ik}) + \beta_{2i} \cdot \min_k(s_{ik})$$

$$\beta_{1i} = \frac{1}{N} \sum_{k=1}^{N} I(\Delta(s_{ik}), \alpha)$$

$$\beta_{2i} = \frac{1}{N} \sum_{k=1}^{N} I(-\Delta(s_{ik}), \alpha)$$

where the "likelihood" of the sample belonging either to the *positive* or the *negative* class, by counting the fraction of the classifiers that exhibit a decidability index larger than an offset α.

The Δ-mean is computed as follows:

$$s_i^* = \beta_i \cdot \max_k(s_{ik}) + (1 - \beta_i) \cdot \min_k(s_{ik})$$

Table 2. Performance in terms of average and standard deviation (between brackets) for all the ensembles of 5 classifiers. Results with a ∘ indicate that the difference in performance from those achieved by the Mean-rule are not statistically significant according to the t-test with a 95% confidence. The best performance are in italics.

	AUC	**EER**
Mean-rule	*0.9998(±0.0002)*	0.0058(±0.0019)
Best expert	0.9984(±0.0014)	0.0125(±0.0046)
DLC	*0.9998(±0.0002)*	*0.0045(±0.0017)*
SDI mean	*0.9998(±0.0002)*	0.0049(±0.0023)
Δ Voting	0.9997(±0.0005)	0.0045(±0.0021)
Δ mean	∘ 0.9998(±0.0003)	0.0047(±0.0019)

	FPR-0%	**FPR-1%**	**FNR-1%**	**FNR-0%**
Mean-rule	0.0941(±0.0342)	0.0040(±0.0026)	0.0023(±0.0017)	0.0719(±0.0827)
Best expert	0.3518(±0.1148)	0.0135(±0.0092)	0.0192(±0.0181)	0.1237(±0.1120)
DLC	*0.0886(±0.0469)*	0.0028(±0.0024)	*0.0008(±0.0010)*	*0.0532(±0.0599)*
SDI mean	∘ 0.0931(±0.0455)	∘ 0.0038(±0.0029)	0.0011(±0.0020)	0.0598(±0.0619)
Δ Voting	0.2017(±0.1622)	*0.0026(±0.0024)*	0.0014(±0.0013)	0.1250(±0.1981)
Δ mean	∘ 0.1015(±0.0723)	0.0029(±0.0025)	0.0010(±0.0013)	0.0895(±0.1150)

where the parameter β_i by taking into account the average and the standard deviation of Δ among all the classifiers as follows:

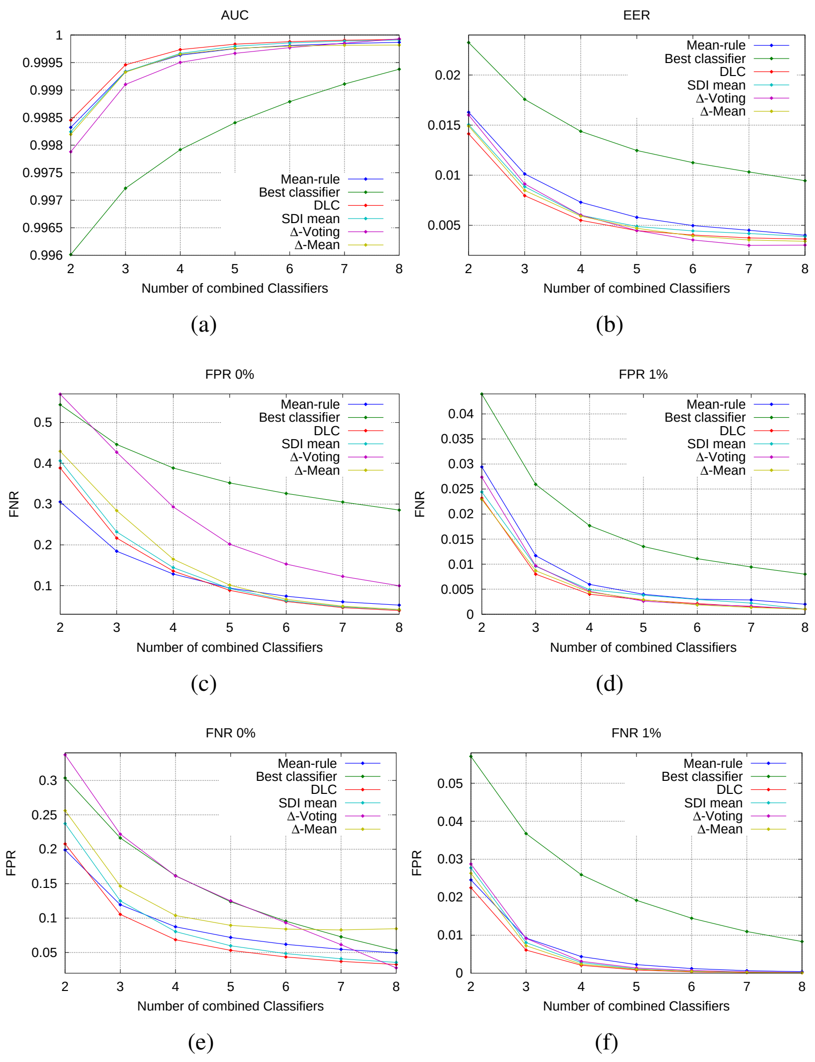
$$\Delta^*(\mathbf{s}_i) = \frac{\frac{1}{N}\sum_{k=1}^{N} \Delta(s_{ik})}{\sigma_{\Delta(s_{ik})}}$$

$$\beta_i = \frac{1}{1+e^{-\gamma \cdot \Delta^*(\mathbf{s}_i)}}$$

Results reported in Fig.12 show that the average performance improve as the size of the ensemble increases. As we already show in the previous section, this result is not surprising as each classifier provides complementary information by construction. In particular, the combination of classifiers always allows outperforming the best classifier, and provide very low error rates. By inspecting the figure, an ensemble size equal to five can be a good compromise between performance and ensemble complexity. For this reason, Table 2 shows the detailed numerical results in terms of the average and standard deviation for an ensemble size equal to five.

Fig. 12(a) shows the results in terms of the AUC. It is easy to see that all the combination methods provide very high AUC values, very close to each other. Fig.s 12(b)-(d) show the performance in terms of EER and FPR 1%, respectively. Regardless the ensemble size, all the dynamic methods outperform those of the mean rule. However, when the EER is considered, the DLC outperform all other measures for ensemble sizes smaller than or equal to five, while Δ-voting provides the best performance for sizes greater than five. On the other hand, when the FPR 1% is considered, the DLC provides the best performance for small ensemble sizes, while differences among the combination mechanisms tends to be negligible as the ensemble size is greater than 5.

Fig. 12. Average performance for each ensemble size.



A similar behavior can be also seen in Fig. 12(f) where the performance for FNR 1% are shown. A different behavior can be seen in Fig.s 12(c)-(e), where the working point is set to 0% FPR or FNR respectively. In these cases, Δ-voting provides the worst performance, while the DLC and SDI-mean outperform the *Mean-rule* for any ensemble size in the case of FNR 0%, while in the case of FPR 0% performance improvements are shown for ensemble sizes greater than or equal to 5. Thus, we can conclude that the combination mechanisms allows exploiting the complementarity of different classifiers, especially in the case of large ensemble size.

The inspection of the values in the Table 2 clearly shows that the AUC does not allow to see any significant difference among the considered combination mechanisms. On the other hand, the values related to the operating point related to very low error rates, show the effectiveness of the combination mechanism. This effectiveness has been also validated by performing the t-test with a 95% confidence on the difference in performance with the *Mean-rule*. All the differences, except those marked with a circle, are statistically significant. In addition, it is worth noting that in security applications even small differences in performances are of great value.

The reported results allow to conclude that the use of an MCS allows exploiting effectively the complementarity among different classifiers. In addition, the performance measure used to assess the effectiveness must be selected in accordance with the requirements of the application scenario at hand.

6.3 Fighting Poisoning Attacks through MCS

In this section we show that MCS can be exploited to enhance robustness against poisoning attacks. We consider a relevant application scenario: the detection of web application attacks.

Web applications are largely employed in simple websites, as well as in security-critical environments such as medical, financial, military and administrative systems. A web application is a software program which generates informative content in real time, e.g., a HTML page that is produced dynamically based on user inputs (queries). Cyber-criminals may divert the expected behavior of a web application by submitting malicious queries, either to access confidential information, or to cause a denial of service (DoS) [30]. To detect such intrusions, the following approach can be used:

1. web application queries can be represented as *sequences* of tokens;
2. a Hidden Markov Model (HMM) can be used to model normal (legitimate) sequences, using real traffic towards web applications (most of traffic is usually legitimate);
3. web application attacks can be detected as anomalous queries (sequences).

Basically, this is the approach proposed in [16], and has been shown experimentally to be very effective. Unfortunately, it may be vulnerable to a clever adversary. For instance, an adversary may deliberately inject well-crafted queries (targeted noise, poisoning) into the pool of queries employed for training, so as to “deviate” the classification algorithm from learning a correct model of legitimate queries, and further evade detection.

However, attackers can control only a small percentage of training samples. It follows that poisoning patterns should be *outliers* in order to be effective. If this were not true, namely, poisoning samples were similar to other samples within the same class (or even to novel samples which represent the normal evolution of the system), their effect would be negligible. Thus, the poisoning problem can be approached by reducing the influence of outliers samples in training data. To this end, we tested the effectiveness of systems employing bagging ensembles.

Bagging, short for *bootstrap aggregating*, was originally proposed in [12] to improve the classification accuracy over an individual classifier, or the approximation error in regression problems. The underlying idea is to perturb the training data by creating a number of bootstrap replicates of the training set, train a classifier on each bootstrap replicate, and aggregate their predictions. This allows to reduce the variance component of the classification or estimation error (in regression) (e.g., [12, 24]). To further reduce the influence of the most outlying observations in training data, *weighted* bagging was proposed in [64, 66]. The rationale behind this approach is to resample the training set by assigning a probability distribution over training samples, in particular, lower probability weights to the most outlying observations. The method can be summarised as follows. Given a training set $T_n = \{\mathbf{x}_i, y_i\}_{i=1}^n$, and a set of probability weights $w_1, \dots, w_n$, for which it holds $\sum_{i=1}^n w_i = 1$:

1. create m bootstrap replicates of T_n by sampling $(\mathbf{x}_i, y_i)$ with probability w_i, $i = 1, \dots, n$;
2. train a set of m classifiers, one on each bootstrap replicate of T_n;
3. combine their predictions, e.g., by majority voting, or averaging.

Note that this corresponds to the standard bagging algorithm [12] when $w_i = 1/n$, $i = 1, \dots, n$, and the majority voting is used as combining rule. The set of weights $w_1, \dots, w_n$ was estimated in [64, 66] using a kernel density estimator. Since kernel density estimation can be unreliable in highly dimensional feature spaces, the authors exploited a *boosted* kernel density estimate, given by

$$f(\mathbf{x}_i) = \sum_{j=1}^{n} \frac{w_j}{(2\pi)^{d/2}\sigma^d} k(\mathbf{x}_i, \mathbf{x}_j), \tag{12}$$

where $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma||\mathbf{x}_i - \mathbf{x}_j||^2)$ is a Gaussian kernel, and the set of weights $w_1, \dots, w_n$ is iteratively estimated as follows. Initially, all samples are equally weighted, i.e., $w_i = 1/n$, $i = 1, \dots, n$. Each weight is then iteratively updated according to $w_i^{(k+1)} = w_i^{(k)} + \log(f^{(k)}(\mathbf{x}_i)/g^{(k)}(\mathbf{x}_i))$, where k represents the current iteration, and $g(\mathbf{x}_i)$ is the "leave-one-out" estimate of $f(\mathbf{x}_i)$, given by

$$g(\mathbf{x}_i) = \sum_{j=1}^{n} \frac{w_j}{(2\pi)^{d/2}\sigma^d} k(\mathbf{x}_i, \mathbf{x}_j) I(j \neq i), \tag{13}$$

where $I(j \neq i)$ equals 0 (1) only when $j = i$ ($j \neq i$). Once convergence or a maximum number of iterations is reached, the final weights are inverted and normalized as

$$w_i = \frac{1}{w_i^{(k)}} \Big/ \sum_{j=1}^{n} \frac{1}{w_j^{(k)}}, \tag{14}$$

so that weights assigned to outlying observations exhibit lower values.

Bagging vs Poisoning: Experiments In our experiments we computed AUC_p for FP rate less or equal than 1%. We experimented with a dataset which reflected real traffic on a production web server employed by our academic institution. We collected `69,001` queries towards the principal web application, in a time interval of `8` months.

We detected `296` intrusive attempts among them. The first `10,000` legitimate queries (in chronological order) were used as the training set, while the remaining `58,705` legitimate queries and the intrusive queries were used as the test set.

Each web application query q has the form $a_1 = v_1 \& a_2 = v_2 \& \ldots \& a_n = v_n$, where a_i is the i-th attribute, v_i is its corresponding value, and n is the number of attributes of q. We encoded each query as the sequence of attributes and their values[1]. The HMM was trained using the Baum-Welch algorithm [8], to exploit the underlying structure of legitimate sequences, and consequently detect intrusions by assigning them a lower likelihood. To build a simple and effective model, we initialized the HMM with two states: one associated to the emission of symbols in even positions, and the other associated to the emission of symbols in odd positions. The emission probability of each symbol was initialized as its relative frequency in even or odd positions, depending on the state. The state transition matrix was randomly initialized.

We carried out experiments with 3, 5, 10, 20 HMMs per ensemble, and the simple average as combining rule. In order to apply the kernel density estimator used in weighted bagging (remind that we deal with sequences of non-fixed length), we first extracted all possible bigrams (i.e. contiguous subsequences of length two) from legitimate and intrusive sequences. Then, we represented each sequence as a Boolean feature vector, in which each value denotes either the absence (0) or presence (1) of the corresponding bigram in the given sequence. The length of each feature vector (total number of bigrams) turned out to be $N = 205$. The default value of γ has been computed as the inverse of the cardinality of the feature space, i.e., $1/N \approx 5E^{-3}$, and $f(\mathbf{x})$ and $g(\mathbf{x})$ were estimated using a subset of 50 training samples. The sensitivity of weighted bagging to γ was further studied by varying $\gamma \in \{5E^{-4}, 2.5E^{-3}\}$.

To simulate a poisoning attack, we generated poisoning queries with (1) a different structure with respect to legitimate queries, and (2) portions of structures similar to intrusive sequences. In particular, poisoning sequences contained only bigrams which were not present in legitimate sequences, but which might have been present in intrusive sequences. As this attack turned out to be very effective[2], we evaluated the performance of the considered classifiers by varying the fraction of poisoning attacks in $[0, 0.02]$ with steps of 0.2%. Results were averaged over 5 repetitions, as poisoning samples were randomly generated and standard deviation values turned out to be negligible.

Fig. 13 shows the results of our experiments. First, note that AUC values decreased for increasing percentage of poisoning, as expected. When no poisoning attack is performed (0%), all classifiers behaved similarly, and, in particular, bagging and weighted bagging only slightly outperformed the corresponding single classifiers. Under attack, instead, bagging and weighted bagging significantly outperformed the single classifiers. In particular, the performance improvement was marked when the injected amount of poisoning attacks significantly affected the single classifier's performance (see, for instance, 0.5% of poisoning).

[1] The whole data set is available at `http://prag.diee.unica.it/pra/system/files/dataset\_hmm\_mcs2011.zip`

[2] On the contrary, we noted that the same classifier was very robust to the injection of random sequences or of the intrusive ones. Classifier performances were not affected significantly even for large amount of such kinds of noise (10%).

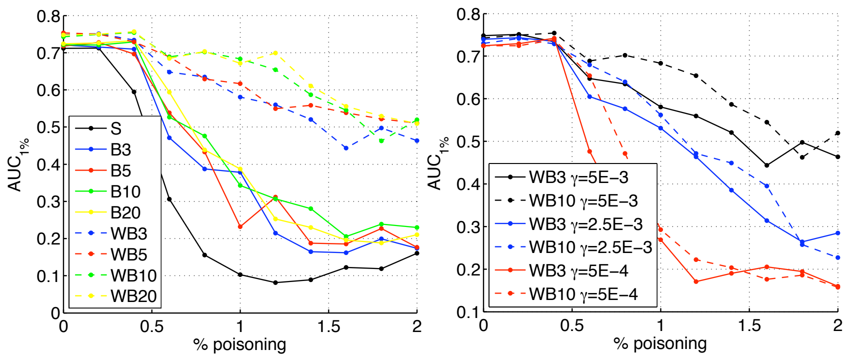


Fig. 13. *Left:* performance of the HMM-web classifier (S), bagging (B), and weighted bagging with default γ (WB) against percentage of poisoning attacks in training data, for different ensemble sizes (3,5,10,20). *Right:* performance of WB with ensemble sizes of 3 and 10 against percentage of poisoning attacks in training data, for different γ.

Increasing the ensemble size of bagging classifiers turned out to not significantly improve the performance of the system under attack. The underlying reason could be that bagging can effectively drop the variance of the classification error by increasing the ensemble size (as shown in [24]); thus, increasing the ensemble size may be effective only when poisoning attacks introduce a substantial variance in the classification error (whereas this may be not true when the error is highly biased). This aspect can be a promising research direction to investigate.

We focus now on weighted bagging, which significantly improved the performance over standard bagging, as expected. This is clearly due to the use of a kernel density estimator, which basically imputes outliers in training data, and reduces their influence. To investigate the effectiveness of weighted bagging more in depth, we considered different values of the γ parameter, as explained in the previous section. The rationale was to alter the performance of the kernel density estimator. Fig. 13 shows that the higher γ, the more gracefully the performance of weighted bagging decreased. It is worth noting that weighted bagging can worsen performance with respect to standard bagging, even in absence of poisoning, if the weights assigned by the kernel density estimator to samples in the same class exhibit a large variance. The reason is that this leads to obtain a set of bootstrap replicates of the training set which do not reflect the correct probability distribution of training samples.

To sum up, standard bagging can provide a significant improvement in performance over an individual classifier, in particular against some kinds of poisoning attacks. The effectiveness of weighted bagging is strictly related to the capability of estimating a reliable set of weights, namely, on the capability of the kernel density estimator to correctly impute the outlying observations. However, when this happens (as in our experiments) weighted bagging can provide a great performance improvement. Besides this, when using a good kernel density estimator the adversary is required to spend more "effort" to build a poisoning attack which misleads weighted bagging.

7 Conclusions

This chapter provided an analysis of the issues related to the application of machine learning algorithms in high-risk applications. In particular, we focused the analysis on computer security applications and on biometric systems. We first investigated the issues concerning the choice of the most suitable model for the problem, by discussing the aspects that must be considered while choosing between one or two-class models for the problem. Our discussion clearly pointed out that the choice of the appropriate model affects not only the classification accuracy, but also has a substantial impact on the robustness of the system against the attempts of evasion. After an overview of the performance measures that are commonly used to evaluate both biometric and computer security systems, we described the possible sources of performance variability for security systems based on machine learning algorithms. In particular, we showed that systems based on multiple classifiers are able to mitigate the effects of this sources of variability. In fact, multiple classifiers allow increasing the classification accuracy, and usually this accuracy increases with the number of combined classifiers. MCS performance can be also improved by thoroughly tuning score combination rules, in order to better exploit the diversity between classifiers within an ensemble. Finally, MCS can be also exploited to strengthen pattern recognition systems against poisoning attacks. This aspect is very important when classification must be performed in an adversarial environment, where a clever adversary can inject malicious noise in the training set.

References

1. Breach Security Inc. - ModSecurity: Open Source Web Application Firewall. http://www.modsecurity.org, November 2009.
2. Breach Security Inc. - WebDefend, November 2009.
3. Citrix Systems Inc. - Netscaler Application Firewall. http://www.citrix.com, November 2009.
4. F5 Networks Inc. - BIG-IP Application Security Manager, November 2009.
5. André Anjos and Sébastien Marcel. Counter-measures to photo attacks in face recognition: a public database and a baseline. In *International Joint Conference on Biometrics 2011*, October 2011.
6. Davide Ariu, Roberto Tronci, and Giorgio Giacinto. HMMpayl: An Intrusion Detection System Based On Hidden Markov Models. *Computers & Security*, 30(4):221 – 241, 2011.
7. Marco Barreno, Blaine Nelson, Anthony D. Joseph, and J. D. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.
8. L.E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The Annals of Mathematical Statistics*, 41(1):164–171, 1970.
9. Jon Atli Benediktsson, Josef Kittler, and Fabio Roli, editors. *Multiple Classifier Systems, 8th International Workshop, MCS 2009, Reykjavik, Iceland, June 10-12, 2009. Proceedings*, volume 5519 of *Lecture Notes in Computer Science*. Springer, 2009.
10. B. Biggio, G. Fumera, and F. Roli. Adversarial pattern classification using multiple classifiers and randomisation. In Niels da Vitoria Lobo, Takis Kasparis, Fabio Roli, James Tin-Yau Kwok, Michael Georgiopoulos, Georgios C. Anagnostopoulos, and Marco Loog, editors, *SSPR/SPR*, volume 5342 of *Lecture Notes in Computer Science*, pages 500–509. Springer, 2008.

11. Andrew P. Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, 1997.
12. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
13. Simon P. Chung and Aloysius K. Mok. Advanced allergy attacks: Does a corpus really help? In Christopher Krugel, Richard Lippmann, and Andrew Clark, editors, *RAID*, volume 4637 of *Lecture Notes in Computer Science*, pages 236–255. Springer, 2007.
14. I. Corona, D. Ariu, and G. Giacinto. HMM-Web: A framework for the detection of attacks against web applications. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1–6, June 2009.
15. I. Corona, G. Giacinto, C. Mazzariello, F. Roli, and C. Sansone. Information fusion for computer security: State of the art and open issues. *Information Fusion*, 10:274–284, 2009.
16. Igino Corona, Davide Ariu, and Giorgio Giacinto. Hmm-web: a framework for the detection of attacks against web applications. In *Proceedings of the 2009 IEEE international conference on Communications*, ICC'09, pages 747–752, Piscataway, NJ, USA, 2009. IEEE Press.
17. Nilesh N. Dalvi, Pedro Domingos, Mausam, Sumit K. Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 99–108, Seattle, Washington, USA, August 22-25 2004.
18. T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Underduk. Polymorphic shellcode engine using spectrum analysis. *Phrack*, 0x0b(0x3d), 2003.
19. Thomas G. Dietterich. Ensemble methods in machine learning. In Josef Kittler and Fabio Roli, editors, *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.
20. R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.
21. R.P.W. Duin. The combining classifier: to train or not to train? In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, volume 2, pages 765–770 vol.2, 2002.
22. Tom Fawcett. An introduction to roc analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.
23. Giorgio Fumera and Fabio Roli. A theoretical and experimental analysis of linear combiners for multiple classifier systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27:942–956, 06/2005 2005.
24. Giorgio Fumera, Fabio Roli, and Alessandra Serrau. A theoretical analysis of bagging as a linear combination of classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(7):1293–1299, 2008.
25. Neamat El Gayar, Josef Kittler, and Fabio Roli, editors. *Multiple Classifier Systems, 9th International Workshop, MCS 2010, Cairo, Egypt, April 7-9, 2010. Proceedings*, volume 5997 of *Lecture Notes in Computer Science*. Springer, 2010.
26. Giorgio Giacinto, Roberto Perdisci, Mauro Del Rio, and Fabio Roli. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Information Fusion*, 9(1):69 – 82, 2008. Special Issue on Applications of Ensemble Methods.
27. M. Haindl, J. Kittler, and F. Roli, editors. *Multiple Classifier Systems - 7th International Workshop (MCS2007)*, volume 4472 of *Lecture Notes in Computer Science*, Prague, Czech Republic, 2007. Springer.
28. James A. Hanley and Barbara J. McNeil. The meaning and the use of the area under a receiver operanting charateristic curve. *Radiology*, 143:29 – 36, 1982.
29. T.K Ho. Multiple classifier combination: Lessons and next steps. In A. Kandel and H. Bunke, editors, *Hybrid Methods in Pattern Recognition*, pages 171–198. World Scientific Publishing, 2002.

30. IBMThreat. X-force 2010 trend and risk report. http://www-935.ibm.com/ services/us/iss/xforce/trendreports/, 2010.
31. Internet Security Systems - IBM-ISS. X-force 2009š trend and risk report. Technical report, IBM Global Technology Services, 2010.
32. Anil K. Jain, R. Bolle, and S. Pankanti. *BIOMETRICS: Personal Identification in Networked society*. Kluwer Academic Publishers, 1999.
33. Anil K. Jain, Patrick Flynn, and Arun A. Ross, editors. *Handbook of Biometrics*. Springer, 2008.
34. Anil K. Jain, Karthik Nandakumar, and Abhishek Nagar. Biometric template security. *EURASIP J. Adv. Signal Process*, 2008:113:1–113:17, January 2008.
35. J. Kittler, M. Hatef, R.P.W. Duin, and J. Matas. On combining classifiers. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 20(3):226–239, Mar 1998.
36. J. Kittler and F. Roli, editors. *Multiple Classifier Systems - First International Workshop (MCS2000)*, volume 1857 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2000. Springer.
37. J. Kittler and F. Roli, editors. *Multiple Classifier Systems - Second International Workshop (MCS2001)*, volume 2096 of *Lecture Notes in Computer Science*, Cambridge, UK, 2001. Springer.
38. M. Kloft and P. Laskov. Online anomaly detection under adversarial impact. In *In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 405–412, 2010.
39. L. Kuncheva. *Combining Pattern Classifiers*. Wiley, 2004.
40. Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning intrusion detection: Supervised or unsupervised? In Fabio Roli and Sergio Vitulano, editors, *ICIAP*, volume 3617 of *Lecture Notes in Computer Science*, pages 50–57. Springer, 2005.
41. Pavel Laskov and Richard Lippmann. Machine learning in adversarial environments. *Machine Learning*, 81:115–119, November 2010.
42. Carlo Lobrano, Roberto Tronci, Giorgio Giacinto, and Fabio Roli. Dynamic linear combination of two-class classifiers. In *Proceedings of the 2010 joint IAPR international conference on Structural, syntactic, and statistical pattern recognition*, SSPR&#38;SPR'10, pages 473–482, Berlin, Heidelberg, 2010. Springer-Verlag.
43. Federico Maggi, William K. Robertson, Christopher Krügel, and Giovanni Vigna. Protecting a moving target: Addressing web application concept drift. In Engin Kirda, Somesh Jha, and Davide Balzarotti, editors, *RAID*, volume 5758 of *Lecture Notes in Computer Science*, pages 21–40. Springer, 2009.
44. D. Maltoni, Maio D., A.K. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer, 2003.
45. H.B. Mann and D.R. Whitney. On a test whether one or two random variable is stochastically larger than the other. *Annals of Mathematical Statistics*, 18(1):50 – 60, 1947.
46. Emanuela Marasco, Peter Johnson, Carlo Sansone, and Stephanie Schuckers. Increase the security of multibiometric systems by incorporating a spoofing detection algorithm in the fusion mechanism. In *Proceedings of the 10th international conference on Multiple classifier systems*, MCS'11, pages 309–318, Berlin, Heidelberg, 2011. Springer-Verlag.
47. Gian Luca Marcialis, Aaron Lewicke, Bozhao Tan, Pietro Coli, Dominic Grimberg, Alberto Congiu, Alessandra Tidu, Fabio Roli, and Stephanie Schuckers. First international fingerprint liveness detection competition–livdet 2009. In *Proceedings of the 15th International Conference on Image Analysis and Processing*, ICIAP '09, pages 12–23, Berlin, Heidelberg, 2009. Springer-Verlag.
48. Gian Luca Marcialis, Fabio Roli, and Luca Didaci. Personal identity verification by serial fusion of fingerprint and face matchers. *Pattern Recognition*, 42(11):2807 – 2817, 2009.

49. D. Mutz, C. Kruegel, W. Robertson, G. Vigna, and R.A. Kemmerer. Reverse engineering of network signatures. In *Proceedings of the AusCERT Asia Pacific Information Technology Security Conference (Gold Coast, Australia), University of Queensland*, 2005.
50. Blaine Nelson, Marco Barreno, Fuching Jack Chi, Anthony D. Joseph, Benjamin I. P. Rubinstein, Udam Saini, Charles Sutton, J. D. Tygar, and Kai Xia. *Misleading learners: Co-opting your spam filter*, chapter Machine Learning in Cyber Trust: Security, Privacy, and Reliability, pages 17–51. Springer, 2009.
51. N.C. Oza, R. Polikar, J. Kittler, and F. Roli, editors. *Multiple Classifier Systems - 6th International Workshop (MCS2005)*, volume 3541 of *Lecture Notes in Computer Science*, Seaside, CA, USA, 2005. Springer.
52. S. Patton, W. Yurcik, and D. Doss. An achilles' heel in signature-based ids: Squealing false positives in snort. In *Proceedings of fourth International Symposium on Recent Advances in Intrusion Detection*, volume 10, page 12, october 2001.
53. R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. Mcpad: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, 53(6):864 – 881, 2009. Special Issue on Traffic Classification and Its Applications to Modern Networks.
54. R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleadingworm signature generators using deliberate noise injection. In *IEEE Symposium on Security and Privacy*, 2006.
55. R. Perdisci, G. Gu, and W. Lee. Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In *Data Mining, 2006. ICDM '06. Sixth International Conference on*, pages 488–498, Dec. 2006.
56. R. Perdisci, A. Lanzi, and W. Lee. Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters*, 29(14):1941 – 1946, 2008.
57. Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul Sharif. Misleading worm signature generators using deliberate noise injection. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P'06)*, Washington, DC, USA, 2006. IEEE Computer Society.
58. Norman Poh and Samy Bengio. Database, protocol and tools for evaluating score-level fusion algorithms in biometric authentication. In *Fifth Int'l. Conf. Audio- and Video-Based Biometric Person Authentication AVBPA*, 0 2005.
59. L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77, pages 257–286, 1989.
60. F. Roli and J Kittler, editors. *Multiple Classifier Systems - Third International Workshop (MCS2002)*, volume 2364 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2002. Springer.
61. F. Roli, J. Kittler, and T. Windeatt, editors. *Multiple Classifier Systems - 5th International Workshop (MCS2004)*, volume 3077 of *Lecture Notes in Computer Science*, Cagliari, Italy, 2004. Springer.
62. Arun A. Ross, Karthik Nandakumar, and Anil K. Jain. *Handbook of Multibiometrics*. Springer-Verlag, 2006.
63. Carlo Sansone, Josef Kittler, and Fabio Roli, editors. *Multiple Classifier Systems - 10th International Workshop, MCS 2011, Naples, Italy, June 15-17, 2011. Proceedings*, volume 6713 of *Lecture Notes in Computer Science*. Springer, 2011.
64. Santi Segui, Laura Igual, and Jordi Vitria. Weighted bagging for graph based one-class classifiers. In *Proceedings of the 9th International Workshop on Multiple Classifier Systems*, volume 5997 of *Lecture Notes in Computer Science*, pages 1–10. Springer-Verlag, 2010.
65. Frederick T. Sheldon and Claire Vishik. Moving toward trustworthy systems: R&d essentials. *Computer*, 43:31–40, September 2010.
66. Albert D. Shieh and David F. Kamm. Ensembles of one class support vector machines. In *Proceedings of the 8th International Workshop on Multiple Classifier Systems*, MCS '09, pages 181–190. Springer-Verlag, Berlin, Heidelberg, 2009.

67. D. M. J. Tax. *One-Class Classification, Concept Learning in the Absence of Counter Examples*. PhD thesis, Delft University of Technology, Delft, Netherland, 2001.
68. Roberto Tronci, Giorgio Giacinto, and Fabio Roli. Dynamic score selection for fusion of multiple biometric matchers. In Rita Cucchiara, editor, *ICIAP*, pages 15–22. IEEE Computer Society, 2007.
69. Roberto Tronci, Giorgio Giacinto, and Fabio Roli. Dynamic score combination: A supervised and unsupervised score combination method. volume 5632, pages 163–177, Leipzig, Germany, 2009. Springer, Springer.
70. Roberto Tronci, Daniele Muntoni, Gianluca Fadda, Maurizio Pili, Nicola Sirena, Gabriele Murgia, Marco Ristori, and Fabio Roli. Fusion of multiple clues for photo-attack detection in face recognition systems. 2011.
71. Giovanni Vigna, William Robertson, and Davide Balzarotti. Testing network-based intrusion detection signatures using mutant exploits. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 21–30, New York, NY, USA, 2004. ACM.
72. K. Wang and S. J. Stolfo. Anomalous payload-based network intrusion detection. In Erland Jonsson, Alfonso Valdes, and Magnus Almgren, editors, *RAID*, volume 3224 of *Lecture Notes in Computer Science*, pages 203–222. Springer, 2004.
73. T. Windeatt and F. Roli, editors. *Multiple Classifier Systems - 4th International Workshop (MCS2004)*, volume 2709 of *Lecture Notes in Computer Science*, Guilford, UK, 2003. Springer.
74. W. Yurcik. Controlling intrusion detection systems by generating false positives: squealing proof-of-concept. In *Local Computer Networks, 2002. Proceedings. LCN 2002. 27th Annual IEEE Conference on*, pages 134–135, Nov. 2002.
75. William Yurcik. Controlling intrusion detection systems by generating false positives: Squealing proof-of-concept. In *LCN*, pages 134–135. IEEE Computer Society, 2002.
76. Stefano Zanero and Sergio M. Savaresi. Unsupervised learning techniques for an intrusion detection system. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 412–419, New York, NY, USA, 2004. ACM.