Evade Hard Multiple Classifier Systems

Battista Biggio, Giorgio Fumera, and Fabio Roli

Abstract E

xperimental and theoretical evidences showed that multiple classifier systems (MCSs) can outperform single classifiers in terms of classification accuracy. MCSs are currently used in several kinds of applications, among which security applications like biometric identity recognition, intrusion detection in computer networks and spam filtering. However security systems operate in *adversarial environments* against intelligent adversaries who try to evade them, and are therefore characterised by the requirement of a high robustness to evasion besides a high classification accuracy. The effectiveness of MCSs in improving the hardness of evasion has not been investigated yet, and their use in security system is based mainly on intuitive and qualitative motivations, besides some experimental evidence. In this chapter we address the issue of investigating why and how MCSs can improve the hardness of evasion of security systems in adversarial environments. To this aim we develop analytical models of adversarial classification problems (also exploiting a theoretical framework recently proposed by other authors), and apply them to analyse two strategies currently used to implement MCSs in several applications. We then give an experimental investigation of the considered strategies on a case study in spam filtering, using a large corpus of publicly available spam and legitimate e-mails, and the SpamAssassin widely used open source spam filter.

Key words: Multiple classifier systems; adversarial classification; hardness of evasion; spam filtering.

Department of Electrical and Electronic Engineering, University of Cagliari, Piazza d'Armi, 09123 Cagliari, Italy {battista.biggio, fumera, roli}@diee.unica.it

1 Introduction

During the past ten years multiple classifier systems (MCS) have become an established approach to design pattern classification systems. A large body of both experimental and theoretical evidence shows that MCSs can outperform a single classifier in several real applications, in terms of classification accuracy (see for instance [8, 6]). In particular, several authors showed that MCSs can allow to improve the detection capability also in security applications like biometric authentication and intrusion detection in computer networks [4, 11]. It is also worth noting that the MCS classifier architecture is also used in commercial and open source spam filters. However, attaining a high classification accuracy or detection capability is not sufficient in security applications, and in particular in so-called *adversarial environments*, in which a security system faces an intelligent, adaptive adversary who exploits the available or acquired knowledge about the system just to evade it. Typical examples of this kind of applications are intrusion detection in computer networks and spam filtering. A crucial issue in this kind of applications, besides detection capability, is the *hardness of evasion*, which can be qualitatively defined as the effort required to the attacker to evade the system. While the effectiveness of MCSs in improving the detection capability has been deeply investigated so far, no work has formally analysed yet their effectiveness in improving the hardness of evasion of a classification system, although MCSs are widely used in adversarial classification tasks as mentioned above. Actually, the issue of the hardness of evasion has been addressed only recently in the machine learning and pattern recognition literature, often with respect to specific tasks, classification systems and type of attacks [5, 7] (for instance, the so-called *good words* attack against the text classifier used in most spam filters), and only in a few works under a more general perspective [1, 9, 3] aimed at developing analytical models of adversarial classification problems. With regard to the use of MCSs explicitly for the specific goal of improving the hardness of evasion, to our knowledge it was proposed by some authors for biometric tasks (see for instance [11]), but only by Perdisci et al. [10] for intrusion detection tasks, while no works considered MCSs for this specific goal in spam filtering tasks.

The aim of this chapter is to make a first step towards a better understanding of why and how MCSs can improve the hardness of evasion of a security system in adversarial classification problems, with respect to the use of a single classifier. We focus in particular on two strategies commonly used to design MCSs, and applied also in security applications. The first strategy is often applied when a set of heterogeneous features is available (as happens for instance in multimodal biometric identity verification tasks). In this case it could be better to combine different classifiers trained on disjoint and heterogeneous subsets of features (for instance, a face classifier and a fingerprint classifier) instead of designing a single, "monolithic" classifier based on all the available features [8]. This strategy is also used when the feature set is

very large (and not necessarily heterogeneous), since it is known that it can help avoiding over-fitting. In this chapter we investigate whether such strategy can be more effective than the use of a single classifier also in terms of hardness of evasion, in adversarial classification tasks.

The second strategy we consider is commonly used to update real spam filters as new kinds of attacks are detected, and was proposed in [10] explicitly for improving the hardness of evasion in intrusion detection tasks. This strategy consists in adding detectors (often, made up as classifiers) based on new features to a classification system. In this chapter we investigate whether this approach can be really effective to improve the hardness of evasion in adversarial classification tasks, trying to provide some argument more formal than the intuitive ones proposed in [10]. To this aim, we will exploit a theoretical framework proposed by Dalvi et al. [3] for adversarial classification problems.

Then, to experimentally investigate the above issues we consider a spam filtering task as a case study, using a large corpus of publicly available spam and legitimate e-mails, and a real and widely used open source spam filter, SpamAssassin.

The chapter is organised as follows. In section 2.1 we give an overview of the use of MCSs in security applications, and of the theoretical framework proposed in [3] for adversarial classification problems. In section 3 we describe how we model an adversarial learning task in which the classifier is a MCS using the framework in [3], and how we model such kinds of tasks to investigate the effectiveness of a MCS against a single classifier, when both classifiers use the same feature set. The experimental results are reported in section 4.

2 Related work

In this section we first give an overview of past work on MCSs for security applications, and then summarise a formal framework proposed in [3] to model adversarial classification problems. Such framework will be exploited in section 3.1 to analyse one of the two strategies mentioned in the introduction for improving the hardness of evasion of a security system.

2.1 Previous works on multiple classifiers for security applications

The use of MCSs to improve the detection accuracy has been recently proposed by several authors for security applications, and in particular biometric authentication and verification [8, 11, 6] and intrusion detection in com-

puter networks [4]. A common scenario in these applications is the availability of heterogeneous features coming from distinct pattern representations (for instance, features extracted from face and fingerprint images in biometric tasks). In this case it is natural to design a detection system based on the combination of different classifiers trained on disjoint feature subsets corresponding to the different pattern representations. This is a well known approach in the MCS field: if different sets of heterogeneous (and possibly loosely correlated) features are available, designing a MCS as described above can be simpler and more effective than designing a single classifier using all the available features (see for instance [8]). Moreover, if the overall number of features is large, a single classifier could be more prone to over-fitting than a MCS. Another motivation in the context of intrusion detection systems was pointed out in [4]: the ensemble approach "reflects the behaviour of network security experts, who usually look at different traffic statistics in order to produce reliable attack signatures."

The above arguments support the use of classifier ensembles to improve the effectiveness of security systems, in terms of attaining high detection rates. MCSs have also been explicitly proposed to improve the hardness of evasion in biometric tasks (see for instance [11]). To our knowledge, the only work which explicitly proposes MCSs to this aim for intrusion detection tasks is [10], while we are unaware of any work for spam filtering tasks. The approach followed in the mentioned works to improve the hardness of evasion is to add to a MCS one or more classifiers trained on new and different features. The motivations for biometric applications are very intuitive: for instance, in [11] it is claimed that using different biometrics like face, fingerprints, and speech allows to discourage attempts to evade a verification system, since this would require the construction of different kinds of fake biometric traits instead of only one. Similar qualitative arguments are given in [10] for intrusion detection systems: combining classifiers trained on different feature spaces "forces the attacker to devise a mimicry attack that evades multiple models of normal traffic at the same time, which is intuitively harder than evading just one model". We point out that, besides experimental evidences, all the above motivations in favour or the use of MCSs for the specific goal of improving hardness of evasion are only intuitive and qualitative, and are not based on formal and more compelling arguments.

Looking to real security systems, it turns out that the design of many spam filters and intrusion detection systems follows the approach based on combining an ensemble of detectors. Consider for instance two well-known open source systems: the SpamAssassin spam filter (`http://spamassassin.apache.org`) and the Snort intrusion detection system (`http://www.snort.org/`). Both SpamAssassin and Snort consist of a set of "tests" which check for different characteristics of input patterns (respectively e-mails and network packets) to detect the presence of "signatures" denoting a malicious origin of the pattern. Tests are often focused on specific signatures of known attacks. They can be of very different kinds, ranging from simple and fixed

feature detectors (like a keyword detector in a spam filter) to arbitrarily complex classifiers (like the text classifiers used in spam filters, also know as "bayesian classifiers" in the spam filtering jargon). The outputs of all tests are then properly combined to obtain a decision on the input pattern (either "legitimate" or not). In the case of SpamAssassin, a score (a real number) is associated to each test. The scores of the tests which are satisfied by an e-mail are first summed up, and then the e-mail is labelled as spam, if the overall score exceeds a predefined threshold; otherwise the e-mail is labelled as legitimate. In the case of Snort, a logic OR is computed on the boolean outcomes of all tests (in other words, a network packet is considered as an attack, if it satisfies at least one of the tests). The SpamAssassin and Snort architectures make it easy to add new tests based on different features as new kinds of attacks come up, and to delete existing tests related to attacks that are no more used. These architectures are supported by experience and intuition that suggest the designer of these kinds of security systems that the characteristics which allow detecting malicious patterns can be very different and heterogeneous, and can change over time due to new tricks used by spammers and hackers to defeat spam filters and intrusion detection systems.

We conclude by pointing out again that so far the hardness of evasion of security systems based on MCSs for adversarial classification problems, in particular for intrusion detection and spam filtering tasks, has been motivated only with intuitive and qualitative arguments.

2.2 A theoretical framework for adversarial classification problems

As explained above, the few works that proposed so far the use of MCSs for improving the hardness evasion in adversarial classification tasks were based only on informal end empirical motivations. This is actually true also for most works that proposed classification systems based on single classifiers. As mentioned in the introduction, just a few works addressed so far the hardness of evasion of machine learning systems under a more general perspective [1, 9, 3], in particular to formally analyse it. In this section we focus on the work by Dalvi et al. [3], who developed a formal framework (the only one so far, to our knowledge) for adversarial classification problems. In the following we summarise this framework, which will be applied in section 3 to model and analyse one of the MCS-based strategies considered in this chapter for improving the hardness of evasion (namely, adding classifiers trained on new features).

When machine learning or pattern recognition techniques are used in applications like spam filtering, intrusion detection, biometric authentication, etc., their task can be formalised as a two-class classification problem. Denoting with y the class label, instances belong either to a positive class made up of

malicious instances ($y = +$), or to a negative class made up of innocent or legitimate instances ($y = -$). Instances are represented as vectors of N feature values, and are considered as random variables $X = (X_1, \ldots, X_i, \ldots, X_N)$. A realisation of such random variable is denoted as $x = (x_1, \ldots, x_i, \ldots, x_N)$, where x_i is a possible value of the feature X_i. It is assumed that instances are generated i.i.d. according to a given distribution $P(X)$, which can be rewritten as $P(X) = P(X|+)P(+) + P(X|-)P(-)$. The feature space $\mathcal{X}$ is defined as the set of all possible realisations of X.

The framework in [3] considers tasks in which the adversary can modify positive instances (the ones generated by him) at the operation phase to make them being misclassified as legitimate by the classifier, but it can not modify any negative instance nor positive instances belonging to the training set. This happens in several real applications. For instance, if a spam filter is trained *off-line* on a hand-labelled corpora of e-mails, spammers can only modify their own spam e-mails to evade the filter, but can not modify legitimate e-mails nor any spam e-mail in the training set. In other cases, like intrusion detection systems trained *online*, the adversary can modify also training instances: the model in [3] can not be directly applied to these kinds of tasks.

In [3] it is further assumed that the classifier and the adversary act according to given utility and cost functions. Denoting with $y_{\mathcal{C}}(x)$ the decision function of the classifier, whose output is intended to be the label assigned to the instance x, the classifier's utility function is denoted as $U_C(y_{\mathcal{C}}, y)$, and represents the utility accrued by assigning to class $y_{\mathcal{C}}(x)$ an instance x belonging to class y. It is reasonable to assume that classifier's utility is positive for correctly classified instances and negative for misclassified ones, that is, $U_C(+,+) > 0$, $U_C(-,-) > 0$, $U_C(+,-) \leq 0$ and $U_C(-,+) \leq 0$. The cost for the classifier is assumed to be the one incurred for measuring feature values. In the following the cost for measuring the i-th feature of an input instance is denoted as V_i. The expression of the expected utility over the distribution $P(x, y)$ (the joint probability of pattern x being generated with true label y) can be obtained taking into account that the adversary acts by first generating a (positive) instance with a corresponding feature representation x, and then possibly modifying it to some different instance x' (if deemed necessary), with the aim of evading the classifier. Such modification is denoted as a function $\mathcal{A}(x)$. For example, spammers could add words which look legitimate to the body text of a spam e-mail, and this could result in a different e-mail feature representation (depending on the features used by the classifier). It follows that the expected utility for the classifier is given by:

$$U_{\mathcal{C}} = \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} P(x, y)[U_C(y_{\mathcal{C}}(\mathcal{A}(x)), y) - \sum_{i=1}^{N} V_i], \tag{1}$$

where $\mathcal{Y} = \{+, -\}$ (note that, by the above assumptions, $\mathcal{A}(x) = x$ if $y = -$, namely for each negative instance).

The adversary's utility function is denoted similarly with $U_A(y_c, y)$. In this case a reasonable assumption is that $U_A(y_c, y)$ takes on a positive value for positive instances misclassified by the classifier as legitimate ($U_A(-,+) > 0$), a negative value for correctly classified positive instances ($U_A(+,+) \leq 0$), and a zero value for negative instances ($U_A(-,-) = U_A(+,-) = 0$), whatever the label assigned by the classifier (in other words, the adversary's utility is not affected by the correct or incorrect classification of negative instances). The cost for the adversary is the one incurred for modifying an instance x, according to the function $\mathcal{A}(x)$. It is assumed that the cost is given by $W(x, \mathcal{A}(x)) = \sum_{i=1}^{N} W_i(x, \mathcal{A}(x))$, being W_i the cost for modifying the i-th feature. Of course, $W_i = 0$ if the i-th feature is not changed, $W_i > 0$ otherwise. The expected utility for the adversary is thus:

$$U_{\mathcal{A}} = \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P(x, y)[U_A(y_{\mathcal{C}}(\mathcal{A}(x)), y) - W(x, \mathcal{A}(x))]. \tag{2}$$

Using the above model, the adversarial classification problem is formulated as a game between classifier and adversary, in which the two players make one move at a time. A move by the classifier consists in choosing a decision function $y_{\mathcal{C}}(\cdot)$ to maximise his expected utility, taking into account both the training set and any knowledge it may have about the strategy $\mathcal{A}(\cdot)$ chosen in the previous move by the adversary. Analogously, a move by the adversary consists in choosing a strategy $\mathcal{A}(\cdot)$ to maximise his own expected utility (eq. 2), taking into account the available knowledge about the decision function chosen by the classifier in the previous move. Although game theory could in principle be applied to find the optimal sequence of moves by both players according to their utility and cost functions, it was shown in [3] that this is computational intractable, and anyway it requires the knowledge of the distribution $P(x, y)$, which is unrealistic. Therefore, a simplified single-shot version of the game was considered in [3]. Initially, the classifier constructs a decision function using a given training set, assuming it is untainted. Then the adversary chooses his strategy $\mathcal{A}(\cdot)$, assuming he has perfect knowledge of the utility and cost functions of the classifier, and also of his classification algorithm and the training set used. Finally, classifier moves again by choosing a new decision function, assuming he has perfect knowledge of the adversary's utility and cost functions, and that he also knows that the adversary has just made his move based on perfect knowledge on the classifier. Under these assumptions, the optimal adversary's strategy for choosing $\mathcal{A}(\cdot)$ turns out to be the following: for each positive instance x, the adversary has to find a modification x' which maximises the corresponding summand in the expression of the adversary's expected utility (2):

$$\mathcal{A}(x) = \operatorname*{argmax}_{x' \in \mathcal{X}} [U_A(y_{\mathcal{C}}(\mathcal{C}(x'), +) - W(x, x')]. \tag{3}$$

Given the above definition of the adversary's utility and cost functions, it is easy to see that the adversary will change any given instance x, only if it is correctly classified by the classifier as positive, and if there is any instance $\mathcal{A}(x) \neq x$ which is misclassified by the classifier as negative, and the modification cost $W(x, \mathcal{A}(x))$ is lower than the utility gain $U_A(-,+) - U_A(+,+)$. Otherwise, the best strategy is to leave the instance x unchanged, namely $\mathcal{A}(x) = x$.

In [3] the above framework was applied to find the optimal strategies of the adversary and the classifier, when the classifier is a Naive Bayes. Experiments on a spam filtering task quantitatively showed that the classifier performance can significantly degrade, if the adversarial nature of this task is not taken into account, while an adversary-aware classifier can perform much better.

The assumption that the adversary and the classifier have perfect knowledge of each other is rather unrealistic in practical applications, as well as the assumption that their behaviour can be modelled in terms of utility functions whose expected value they aim to maximise. Despite this, we point out that this framework is the first one proposed to model adversarial learning problems in the machine learning field, and it is thus worth taking it into account to formally analyse strategies to improve the hardness of evasion as the ones considered in this chapter.

3 Are multiple classifier systems harder to evade?

In this section we develop formal models of adversarial classification problems in which the classification system is made up of an MCS, with the aim of investigating whether MCSs could improve the hardness of evasion. In particular, the model in section 3.1 is based on the framework developed in [3].

3.1 Adding features to a classification system

In this section we consider a strategy proposed by several authors to design classifiers for different security applications, and commonly used in real security systems, as explained in section 2.1. The strategy consists in adding to a given classifier ensemble new classifiers trained on different features. We focus in particular on a simple kind of combining function, which consists in thresholding the weighted sum of the outputs provided by each classifier. The reason is that this combining function is equivalent to the ones commonly used in spam filters and intrusion detection systems, as the SpamAssassin and Snort software described in section 2.1. In the following we apply the framework of [3] to model a classifier based on this strategy, and analyse whether and how

such strategy can allow to improve the hardness of evasion of the classifier, according to this framework.

We consider a classifier ensemble made up of N classifiers trained on different feature subsets. We denote with $s_i(x)$ the output provided by the i-th classifier for the instance x (x can be considered the whole feature vector, while each classifier uses only a subset of these features), and with t the decision threshold. The decision function we consider can be defined as:

$$y_C(x) = \begin{cases} +, \text{ if } \sum_{i=1}^{N} s_i \geq t, \\ -, \text{ if } \sum_{i=1}^{N} s_i < t. \end{cases} \tag{4}$$

We also consider V_i and $W_i(x, \mathcal{A}(x))$ as the cost incurred respectively by the classifier for measuring the values of the i-th feature set of the instance x, and by the adversary for modifying the same features of x.

In the framework of [3] the classifier strategy against the adversary leads to choose a new decision function at each move. To apply the framework of [3] to our case, we add the constraint that the new decision function is obtained by adding one or more new classifiers to the previous ensemble. As in [3], we assume that the initial classifier ensemble is trained on a given training set of untainted instances, namely $\mathcal{A}(x) = x$. Then the adversary reacts by devising a strategy $\mathcal{A}(x)$, which is likely to decrease the classifier effectiveness. Next, some new classifiers are added to the previous classifier ensemble. The adversary can in turn react again by devising a new strategy to defeat the new version of the MCS, and so on. The question we will try to answer is: does adding new classifiers to a previous ensemble makes it harder to evade?

Let us now define in details the adversary strategy, namely the optimal way in which the adversary should choose the function $\mathcal{A}(x)$ against a given ensemble of N classifiers. To this aim, we assume that the adversary knows the feature set used by each of the individual classifiers, the score $s_i(x), i = 1, \ldots, N$ provided by each individual classifier for any positive instance x, and the threshold t. For the sake of simplicity, we also assume that the cost $W_i(x, \mathcal{A}(x))$ for modifying the i-th feature set is equal to the absolute difference between the corresponding scores $s_i(x)$ and $s_i(\mathcal{A}(x))$. This means that the total cost $W(x, \mathcal{A}(x))$ equals the Manhattan distance in the N-dimensional score space between the corresponding score vectors. In other words, the higher the score reduction the adversary would like to attain, the more the changes he has to make to his positive instance. We point out that this is only a simplifying assumption, since in practice a given reduction of the overall score could be attained by different modifications to the same instance at the expense of a different cost incurred by the adversary. However, modelling this fact is much more complex, without any specific assumption about the nature of instances, of the features used by the classifier and of the kinds of attacks used by the adversary. It should be noted that an assumption similar to ours about the cost of modifying an instance in a given

feature space was made in [9]: in that work, the cost was assumed to be a function of the distance between x and $\mathcal{A}(x)$ in the feature space.

The optimal strategy of the adversary in [3] is defined by eq. 3. In our case, denoting with ΔU_A the difference $U_A(-,+)-U_A(+,+)$, with the above definition of the adversary's cost function the optimal strategy against an ensemble of N classifiers can be rewritten as follows:

$$\mathcal{A}(x) = \begin{cases} x' \neq x, \text{ if } \exists x' : y_{\mathcal{C}}(x') = -, \Delta U_A > W(x,x'), \\ \qquad x' = \text{argmax}_{x'' \in \mathcal{X}} \, \Delta U_A - W(x,x''), \\ x, \qquad \text{otherwise.} \end{cases} \tag{5}$$

The above optimal strategy can be rephrased as finding, for any given instance x which is correctly classified as positive by the classifier, namely $\sum_{i=1}^{N} s_i(x) \geq t$, an instance $\mathcal{A}(x)$ which is misclassified as negative by the classifier, namely $\sum_{i=1}^{N} s_i(\mathcal{A}(x)) < t$, and for which the utility gain ΔU_A exceeds the cost for making the modification, which by the above assumptions is given by:

$$W(x, \mathcal{A}(x)) = \sum_{i=1}^{N} |s_i(x) - s_i(\mathcal{A}(x))|. \tag{6}$$

If no such instance can be found, then x is left unchanged. It is not difficult to see that the minimum cost the adversary has to incur so that the modified instance is misclassified as negative equals the difference between the total score given to x by the classifier and the decision threshold t: $\sum_{i=1}^{N} s_i(x) - t$.

It is now possible to give a formal explanation, according to the framework in [3], of the reasons why adding new classifiers to a given ensemble could improve the hardness of evasion, as well as the detection capability. We consider the simplest case in which the previous classifiers and the decision threshold t are left unchanged at each move. A consequence of adding M new classifiers ($M \geq 1$) to an existing ensemble of N classifiers is that the score of any positive instance could increase, under the reasonable assumption that the individual classifiers are well trained. In particular, consider the optimal strategy $\mathcal{A}(x)$ against N classifiers. As seen above, when this strategy leads the adversary to modify an instance x to a different instance $\mathcal{A}(x) \neq x$, this allows to evade the classifier. Denoting with $s^k(x)$ the score provided by an ensemble of k classifiers, this means that $s^N(x) = \sum_{i=1}^{N} s_i(\mathcal{A}(x)) < t$. However, the modified instance obtained with the strategy devised against N classifiers is not guaranteed to evade a new ensemble comprising M new classifiers. Indeed the new score $s^{N+M}(x)$ will be given by the sum of the previous score and of the scores of the new classifiers, $s^N(x) + \sum_{i=N+1}^{N+M} s_i(x)$, which could exceed t. This means that the optimal strategy of the adversary against N classifiers is not guaranteed to be optimal against $M+N$ classifiers. Accordingly, the detection capability can improve by adding new classifiers. Moreover, the evasion cost could increase by adding new classifiers. Indeed, if the score for some positive instance x correctly classified by the new classi-

fier ensemble increases from $s^N(x)$ to $s^{N+M}(x) > s^N(x)$, such increase could make the difference $s^{N+M}(x) - t$ larger than the utility, $U_A(-,+)$, that the adversary would gain by modifying x so that it is misclassified as negative. This implies that there could be some positive instances that the adversary can afford to modify to evade N classifiers, but not to evade $N+M$ classifiers. This means that the classifier has become harder to evade.

In the above model it is assumed that the adversary can modify only positive instances, and the analysis was focused only on the classifier's detection capability on positive instances (namely, on the false negative error rate). Before concluding this section it is worth discussing also the possible effects of adding classifiers, to the classification accuracy on negative instances (the false positive error rate). Under the same reasonable assumption above that the individual classifiers are well trained, a negative instance x which is correctly classified as negative by an ensemble of N classifiers (namely, $s^N(x) < t$) is likely to be classified as negative also by an ensemble of $N+M$ classifiers, if the sum of the new scores $\sum_{i=N+1}^{N+M} s^i(x)$, is lower than $t - s^N(x)$. Moreover, instances erroneously classified as positive by N classifiers (namely, $s^N(x) \geq t$) could be correctly recognised as negative by a larger ensemble, if $\sum_{i=N+1}^{N+M} s^i(x)$ is negative and lower than $t - s^N(x)$.

To sum up, the above analysis provides a first, formal support to the strategy of adding new classifiers trained on different features to a given classifier ensemble, to improve both the detection capability and the hardness of evasion. In section 4.1 we will give an experimental evaluation on a case study related to spam filtering.

3.2 Splitting features across an ensemble of classifiers

The second MCS-based strategy we consider is a design approach applied in several applications to simplify the classifier design and to improve classification accuracy, when the feature set is very large or is made up of heterogeneous feature subsets. The approach consists in combining different classifiers trained on disjoint feature subsets, instead of designing a single, "monolithic" classifier based on all the available features. It can be implemented naturally on heterogeneous features: a typical example is the combination of a face classifier and a fingerprint classifier in biometric tasks. The issue we address is the following: could this design approach be exploited to improve the hardness of evasion of a security system in adversarial classification tasks?

In this chapter we try to give a first answer to this question, without focusing to any specific application. We will not apply the model in [3] as made for the MCS-based strategy analysed in section 3.1, since in this case we are not analysing a defence strategy, but we are comparing two different classifier design approaches. We develop instead a simple, general model of a classification system based on these two classifier architectures (either a single

classifier trained on a given feature set, or an ensemble of classifiers trained on disjoint subsets of the same feature set), and a method for evaluating the corresponding hardness of evasion.

We first assume that a fixed feature set $x_1, \ldots, x_n$ is available, and that all the features are binary and take on the values 0 and 1. Without loosing generality we assume that the value 1 of any feature denotes the presence of some "malicious" characteristic in the input instance, while a 0 value denotes its absence. In the case of a single classifier, denoting with x a feature vector and with $y_{\mathcal{C}}(x)$ the decision function, we assume that $y_{\mathcal{C}}(x)$ is a thresholded weighted sum of the input features $x_1, \ldots, x_n$, with weights $w_1, \ldots, w_n$:

$$y_{\mathcal{C}}(x) = \begin{cases} +, \text{ if } \sum_{i=1}^{n} w_i x_i \geq t, \\ -, \text{ if } \sum_{i=1}^{n} w_i x_i < t. \end{cases} \tag{7}$$

Note that this kind of decision function, as well as the above assumption on the features, fit several real classification systems for security tasks, like SpamAssassin and Snort.

In the case of a MCS made up of N classifiers trained on N disjoint subsets of the same n features, we assume that the individual classifiers have the same kind of decision function (7). As combining function we consider the logical OR between the N boolean outputs of the individual classifiers, where the logical value *true* is assumed to denote the positive class $y = +$ (in other words, for an input pattern being labelled as positive by the MCS, it is sufficient that at least one of the individual classifiers labels it as positive). We consider a non-linear combining function because a linear one (a linear combination of the soft outputs of the individual classifiers) would lead to the same kind of decision function as the one of the monolithic classifier (since also the decision functions of individual classifiers is linear). We consider in particular the logical OR because of its simplicity, and because it is particularly suited to keep the false negative error rate low. We remind the reader that this combining function is used in Snort. In principle, it could also be used to combine different spam filters or different intrusion detection systems, whose outputs can be viewed as the features. Given that the value 1 of any feature denotes the presence of some "malicious" characteristic in the input instance, it follows that all the weights of both the monolithic classifier and the N individual classifiers of the ensemble are non-negative (because it is reasonable that the presence of a "malicious" characteristic must not decrease the overall score of a classifier). A scheme of the two classifier architectures is shown in figure 1.

In the two classifier architectures above, the parameters to be set during the training phase are the number of individual classifiers in the MCS, the feature subset associated to each individual classifier, and the values of the weights and of the decision thresholds in the decision functions of the linear classifiers. These choices will affect the effectiveness of the classifiers. The effectiveness has to be measured in terms of both the classification accuracy

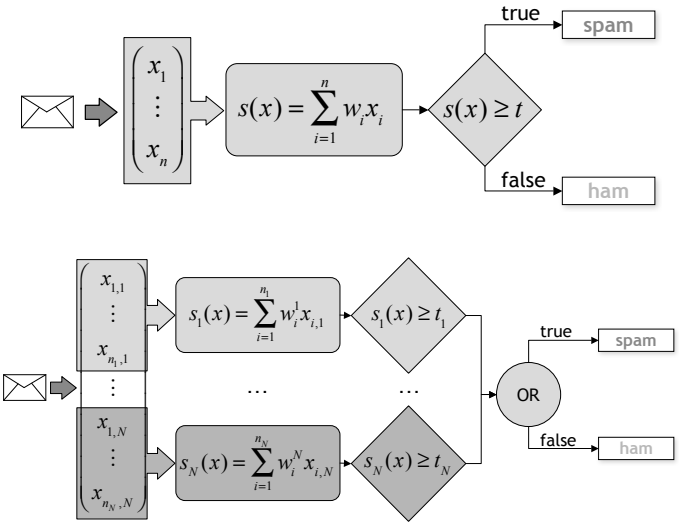

Fig. 1 The two classifier architectures considered in this section. A single, linear classifier working on n features (top). N linear classifiers working on disjoint subsets of the same n features, whose decisions are combined using the OR logical function (bottom).

and the hardness of evasion. Note that in adversarial classification problems the classification accuracy should be intended as a "static" characteristic of a classifier, in the sense that it is related to a *fixed* strategy used by the adversary. Such strategy can be considered as represented by the training instances. The hardness of evasion measures instead how easy is for the adversary to evade the classifier using one or more different specific strategies. In other words, it measures how vulnerable the classifier is to specific kind of attacks, different than the ones represented in the training set. Therefore, when comparing different security systems both measures have to be taken into account, as in the scheme of figure 2. Ideally, a security system should be characterised both by a high accuracy ad a high hardness of evasion. In practice a trade-off between the two goals could be needed.

The classification accuracy can be evaluated in terms of the false positive and false negative classification rates. Usually, the suitable trade-off between these misclassification rates is application-dependent. How to measure the hardness of evasion is clearly application-dependent as well. In particular, it could depend on the kind of classification system, on the knowledge the adversary has about it, and on the kinds of attacks he could make. However in this section we consider a measure of the hardness of evasion focused on comparing the two classifier architectures we are interested in, without making any specific assumption on the application. Concerning the adversary,

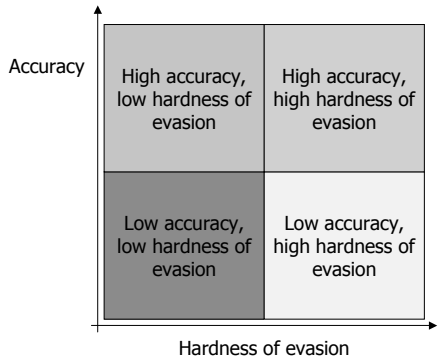

Fig. 2 An example of the two measures which should be used to evaluate the performance of a classifier in a security system: the classification accuracy against a given strategy used by the adversary (represented by training instances), and the hardness of evasion against new kinds of attacks. Ideally, the classifier should exhibit both a higher accuracy and a high hardness of evasion (green region in the accuracy-hardness of evasion plane).

we consider the worst case scenario for the classification system, as in the framework in [3]: we assume that the adversary has full knowledge of the classifier architecture, of the features and of the parameter values, and is capable to evade any feature (namely, to turn the value of any feature from 1 to 0). We further assume that the adversary has to make the same effort to evade any feature. We point out that this last assumption could not be true in practice. However, taking into account different costs for evading different features would make our model much more complex, which is out of the scope of this chapter. Under the above assumptions, the hardness of evasion can be defined as follows:

> *For a given feature set, the hardness of evasion is defined as the expected value of the minimum number of features which have to be modified to evade the classifier.*

Accordingly, a classifier A will be harder to evade than a classifier B, if the average minimum number of features the adversary has to evade for evading A is higher than for evading B. The whole classifier performance could thus be measured in the accuracy-hardness of evasion plane of figure 2 by using a proper combination of false positive and false negative classification rates in the Y axis (note that, in this case, the accuracy increases for *decreasing* values in the corresponding axis) and the average minimum number of features to evade for evading the whole classifier in the X axis.

It is now possible to discuss, at least informally, whether and how the MCS classifier architecture discussed above could be harder to evade than the monolithic classifier. Consider a given positive instance whose feature vector x is correctly classified by the monolithic classifier, namely, $s(x) \geq t$ (see figure 1). Under the above assumptions, to evade such classifier the adversary will have to modify such instance to some instance with feature

vector x' with the aim of turning to 0 those features which exhibit in x a value of 1 and are associated to the largest weights, until the overall output of the classifier becomes lower than the threshold t: $s(x') < t$. Instead, to evade the MCS the adversary has to evade all individual classifiers which correctly classify an instance as positive, since they are combined with the logical OR function. Since the individual classifiers are assumed to implement the same kind of decision function (7) as the monolithic classifier, the adversary will have to apply the same strategy above against *all* the individual classifiers. More precisely, let us denote with x_m the feature vector of the m-th individual classifier. Assuming it correctly classifies x_m as positive (namely, $s_m(x_m) \geq t_m$), the adversary will have to modify his original instance to some other instance with feature vector x'_m, in which the features exhibiting in x_m a value of 1 and are associated to the largest weights are turned to 0, until the overall output of the m-th classifier becomes lower than the threshold t_m: $s_m(x'_m) < t_m$. We point out again that this has to be done for each individual classifier which correctly classifies an instance as positive. It follows that a proper choice of the feature subsets could force the adversary to evade on average a higher number of features to evade a MCS with the above architecture, than to evade the monolithic classifier. It should however be noted that the kind of MCS considered in this section could exhibit a higher false positive error rate than the monolithic classifier, since each individual classifier of the MCS is trained with a smaller feature set, and an input instance is labelled as positive if at least one individual classifier labels it as positive. Accordingly, the attainable advantage of the MCSs in terms of hardness of evasion could need to be traded-off for an increase in false positives. In the following section we investigate experimentally the hardness of evasion of these two classifier architectures on the same case study as the one considered for the MCS-based strategy analysed in section 3.1.

4 A case study in spam filtering

In this section we apply the two formal models of section 3 to a case study of a spam filtering task, to experimentally analyse the hardness of evasion of the two corresponding MCS-based strategies considered in this chapter. For our experiments we use the well known open source SpamAssassin spam filter, whose architecture has been described in section 2.1, and a large and publicly available corpus of real spam and legitimate e-mails.

We used the latest versions of SpamAssassin available at the time of carrying out our experiments: version 3.2.4 for the experiments in section 4.1, and 3.2.5 for the ones in section 4.2. We used the filter configuration named "bayes+net", which includes all the available tests (several hundreds). The outputs of all tests are binary (either 0 or 1). Nine of the tests are associated to a text classifier with features corresponding to terms in the e-mails' header

and body. The continuous-valued output of the text classifier is discretized by default into nine disjoint intervals, each of which is associated with a binary test. All the remaining tests consist in fixed feature detectors. SpamAssassin is deployed with a default value for the weight of each test, and a default value of 5 for the detection threshold. All these values can be modified by the user. All the details about SpamAssassin, including the description of its tests, can be found in `http://spamassassin.apache.org/`

The e-mail corpus we used is the TREC 2007 e-mail data set, available at `http://plg.uwaterloo.ca/~gvcormac/treccorpus07/`. It is made up of 75,419 real e-mail messages, received by a mail server between April 2007 and July 2007, and contains 25,220 legitimate and 50,199 spam messages.

In sections 4.1 and 4.2 we describe the experiments aimed at evaluating the hardness of evasion attainable respectively by adding new classifiers to a classifier ensemble, and by using an ensemble of classifiers trained on disjoint subsets of given feature set, instead of a single classifier trained on the whole feature set.

4.1 Adding features to a spam filter

In this section we evaluate the hardness of evasion of the SpamAssassin filter, attainable by adding new tests (which can be thought as classifiers, as explained above) to a previous set of tests (equivalently, to a previous classifier ensemble). We point out that in our experiments the SpamAssassin tests can be considered as classifiers, although most of them are fixed feature detectors, because we consider the case in which the previous classifiers of the ensemble are *not* retrained, and neither their weight nor the decision threshold is changed, after new classifier are added. For these experiments we used the first 10,000 messages of the TREC 2007 corpus, in chronological order (1,969 legitimate e-mails and 8,031 spam e-mails), to train the SpamAssassin text classifier. The remaining 65,419 e-mails were used as a test set.

In the model of section 3.1 the adversary was assumed to be capable to modify his instances to attain any modification he would like on the classifier's outputs. In practice this could be not always possible. However it was very difficult to check whether this is possible or not for all SpamAssassin's tests. For the sake of simplicity, we kept the above assumption and avoided to devise real modifications to e-mails to attain the desired output changes. We point out that this assumption is totally in favour of the adversary, since we are not setting any constraint on the actual modifications which can be made on spam e-mails by him.

In the experiments we considered the following utility function of the classifier: $U_C(+,+) = 1, U_C(-,+) = -10, U_C(+,-) = -1, U_C(-,-) = 1$, namely, it gains 1 for correct classifications, looses 1 for misclassifying a spam e-mail as legitimate, and looses 10 for misclassifying a legitimate e-mail as spam.

This is coherent with the considered application, in which it is generally agreed that false positive errors are much more harmful than false negative ones. The utility function of the adversary was set to 0, except for the gain accrued for evading the classifier, namely for spam e-mails misclassified as legitimate, for which two different values (1 and 5) were considered: $U_A(+,+) = U_A(+,-) = U_A(-,-) = 0, U_A(-,+) = 1,\ 5$. We considered two different values of $U_A(-,+)$ to evaluate scenarios characterised by a different value of the maximum cost the adversary can (or wants) to pay to evade a spam filter. We point out that the above choices of the relative values of the utility functions is somewhat arbitrary, besides the obvious constraints mentioned above, due to the fact that such costs can not be precisely evaluated in practice (and it could also be questionable that the *real* behaviour of a classification system and of an adversary can be modelled in terms of such utility functions, as pointed out in section 2.2). However, we are interested here to the qualitative behaviour of the classifier's and adversary's performance (in terms of its expected utility), and different choices of the utility values would affect only the absolute values of their expected utilities, not their qualitative behaviour.

Finally, we assumed that the cost V_i faced by the classifier for measuring the features associated to the i-th text (or classifier) is zero, since such cost is just a negative constant added to the expected value of the utility function in the framework of section 3.1). The cost for the adversary was defined as explained in section 3.1, as the manhattan distance in classifiers' outputs space between the outputs given to a positive instance after and before the modification made by the adversary.

The addition of new classifiers at each move of the game was modelled as follows. Since the number of SpamAssassin tests is rather high (several hundreds), we did not add just one test at each move. Instead we subdivided them into n disjoint subsets $S_1, \ldots, S_n$, and added at each step all the tests of a given subset. For the purposes of these experiments we chose $n = 6$. The number of test was set to 119 for S_1 and to 100 for all the other subsets, for a total of 619 tests. This choice was made since only 619 out of all tests gave an output value of 1 for at least one of the e-mails in our data set: we considered therefore only these 619 tests. In the real SpamAssassin filter new tests are usually added in response to new spammers' tricks. Accordingly, it would have been reasonable to subdivide the tests taking into account their chronological order. Unfortunately the time in which each test was introduced is not reported in the SpamAssassin documentation. So we had to resort to a random subdivision. To make experiments easily reproducible, we sorted all tests alphabetically according to their names as listed in `http://spamassassin.apache.org/tests_3_2_x.html`. The only exception were the nine tests related to the text classifier, which were included in the first subset since it is known that text classifiers are used in spam filters since several years.

The moves of the classifier and the adversary at each step of the game were implemented according to the following procedure. At each step, we first evaluate the performance of the classifier and the adversary after a new set of tests is added to the classifier, and the adversary uses the strategy which was optimal against the previous set of tests (in the first step, this means that the adversary does not modify his instances). This simulates what happens in real cases, as soon as a spam filer is updated. Then the optimal strategy of the adversary against the new set of tests is computed, and the performances of the classifier and the adversary are evaluated again. This simulates what happens when spammers devise new tricks to evade the last version of a spam filter. This procedure can be formalised as follows:

1. $R \leftarrow \emptyset$, $\mathcal{A}^0(x) = x$ for all x
2. For $k = 1, \ldots, n$:
   - 2.1 $R \leftarrow R \bigcup S_k$
   - 2.2 Evaluate the expected utility of the classifier and of the adversary, when the classifier uses the tests in R and the adversary uses the strategy $\mathcal{A}^{k-1}(x)$ which was optimal for the previous set of tests
   - 2.3 Compute the optimal adversarial strategy $\mathcal{A}^k(x)$ against tests in R
   - 2.4 Evaluate the expected utility of the classifier and of the adversary, when the classifier uses the tests in R and the adversary uses the corresponding optimal strategy $\mathcal{A}^k(x)$

The adversary's optimal strategy $\mathcal{A}^k(x)$ at step k was computed as follows, according to section 3.1. Denoting the set of tests $S_1 \bigcup \ldots \bigcup S_k$ used by the classifier as R, for any positive instance x correctly classified by the filter (namely $y_{\mathcal{C}}(x) = +$, or equivalently $\sum_{i \in R} s_i(x) \geq t$), we compute the set of feasible values s'_i for the scores of tests in R which would correspond to an instance x' classified as negative (namely $\sum_{i \in R} s'_i < t$), such that the corresponding cost $W(x, x') = \sum_{i \in R} |s'_i - s_i(x)|$ is minimum and is lower than the utility gain. If such scores exist, then we assume that the adversary evades the filters by modifying x, otherwise it is assumed that the adversary can not afford to modify x to evade the filter.

The results are shown in figures 3 and 4, for both utility functions considered for the adversary, in terms of the expected utility of the adversary and of the classifier, as a function of the number of tests used in SpamAssassin. The results in the top-left graph refer to the case in which the adversary does not modify his instances. As one could expect, the expected utility of the classifier increases as the number of test increases while the opposite happens for the adversary. This means that adding new classifiers (tests) based on different features (without modifying the previous ones nor the detection threshold) allowed to improve the detection capability. The only exception is when going from 419 to 519 tests. The bottom-left graph shows what happens when the adversary uses the optimal strategy against each set of tests. The expected utility of the adversary significantly improves with respect to the previous case. The expected utility of the classifier still increases as the number of

tests increases, but obviously attains lower values than in the previous case. However, it is worth noting that the improvement attained by the adversary, reported in the top-right graph, tends to decrease as the number of tests increases. Similarly, the decrease in the classifier's expected utility tends to be higher for lower number of tests. The reason is that the modification cost the adversary has to face to evade the classifier increases as the number of tests increases, until it exceeds the utility gain for some positive instances, making it no more convenient to modify them. This is a clear evidence that adding new classifiers based on different features can allow to improve not only the classifier's discriminant capability, but also its hardness of evasion. Consider finally the bottom-right graph, corresponding to the case when the classifier adds new tests, and the adversary uses the strategy which was optimal against the *previous* set of tests. For lower number of tests (up to 319), the expected utility of the adversary is between the ones of the first two graphs: this is reasonable, because it is now trying to evade only *some* of the tests used by the classifier. However, for higher number of tests its expected utility is even worse than the one it attained *without* trying to evade any test. The expected utility of the classifier is instead very close to the one it attained when the adversary did not try to evade any test. This means that the addition of new tests allowed to compensate the actions made by the adversary to evade the previous tests. In other words, most spam e-mails which evaded the previous version of the filter were detected by the new tests.

The behaviour of the expected utility for the two different values of the gain accrued by the adversary for evading the classifier (figure 3 vs. figure 4) is similar, with the obvious difference that the expected utility of the adversary is higher in the graphs of figure 4 than in figure 3, since it can afford a higher cost to modify instances. The opposite happens for the classifier.

These experimental results on a real case study give thus a quantitative confirmation to the theoretical explanation given in section 3.1 on the effectiveness of adding new classifiers based on different features in improving both the detection capability and the hardness of evasion of a security system like a spam filter.

4.2 Splitting the features of a spam filter across an ensemble of classifiers

In this section we give an experimental evaluation of the classification accuracy and the hardness of evasion of the two classifier design architectures modelled in section 3.2: a single linear classifier trained on a given set of features, and an ensemble of linear classifiers trained on disjoint subsets of the same features and combined with the OR logical function. The experiments were carried out on the TREC 2007 e-mail corpus described in section 4.1. We used as feature set the tests of the SpamAssassin filter (version 3.2.5)

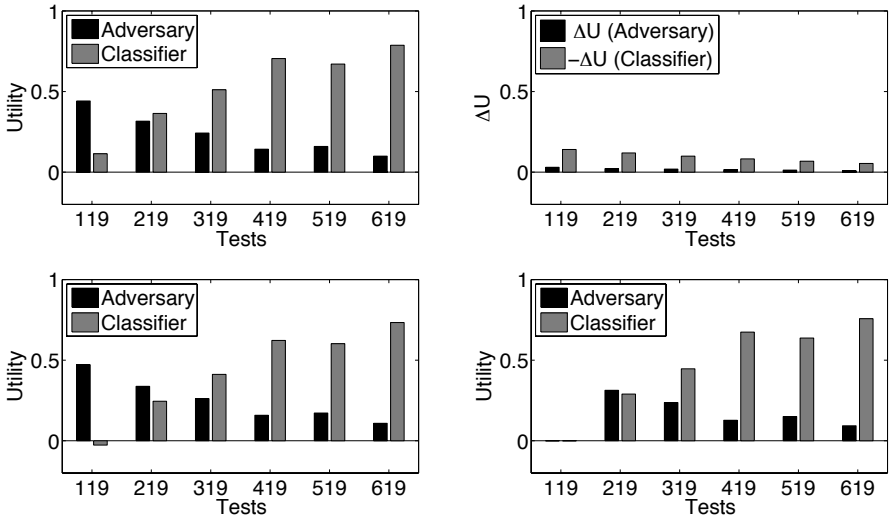

Fig. 3 Expected utility for the adversary and the classifier, as a function of the size of the classifier ensemble, when $U_A(-,+) = 1$. Top-left: the adversary does not modify his instances. Bottom-left: the adversary uses the optimal strategy against the classifier. Top-right: the gain and the loss in expected utility attained respectively by the adversary and the classifier, when passing from the situation in the top-left graph to that in the bottom-left one, as a function of the ensemble size. Bottom-right: for each ensemble size, the adversary uses the optimal strategy against the *previous* set of rules.

which gave an output value of 1 for at least one of the e-mails of this data set. Their number was 549. To implement the monolithic linear classifier and the individual classifiers of the ensemble we used a support vector machine (SVM) with the linear kernel. Since nine of the SpamAssassin tests are associated with a text classifier, they were not used as features of the MCS. The text classifier itself was instead used as one of the individual classifiers of the ensemble. In this case, given that its output is a real number in the interval $[0, 1]$ (with small values denoting legitimate e-mails), we set a decision threshold of 0.5.

The first 10,000 e-mails of the data set, in chronological order (1,969 legitimate e-mails and 8,031 spam e-mails), were used to train the SpamAssassin text classifier, and the individual classifiers of the MCS. The next 10,000 e-mails were used to train the monolithic classifier (we avoided using to this aim the same first 10,000 e-mails used to train the text classifier, since its outputs were used as features of the monolithic classifier). The remaining 55,419 e-mails were used as test set. The SVMs were trained using the publicly available `libsvm` software [2]. To carry out multiple runs of the experiments, all the classifiers were trained on 2,000 e-mails randomly extracted from the corresponding training sets described above.

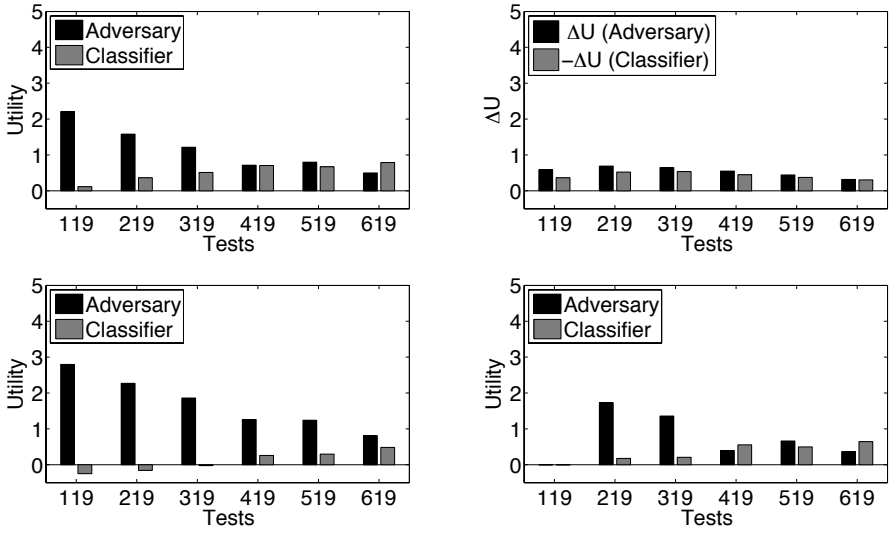

Fig. 4 Expected utility for the adversary and the classifier, as a function of classifier ensemble size, when $U_A(-,+) = 5$. See caption of figure 3 for the other details.

The SVM parameters of the monolithic classifier were set through a 5-fold cross validation on the training set, by minimising an objective function given by $100 \times FP + FN$, being FP and FN the two kinds of error rates. In other words, the cost of false positive errors was fixed to be one hundred times higher than the cost of false negative errors. The decision threshold of the SVM was fixed by minimising the same objective function. The same procedure was used to set the parameters and the decision threshold of the individual classifiers of the MCS. However in this case we fixed the cost of false positive errors to be *one thousand* times higher than the cost of false negative errors. The reason is that, differently from the features of the monolithic classifier, the individual classifiers of the MCS have been combined with the OR function, implying that the MCS labels an e-mail as spam, if at least one of the individual classifiers labels it as spam; it is therefore necessary to keep the false positive error rate of the individual classifiers of the MCS as low as possible. The parameters of the SVMs were the regularisation parameter C and the relative cost of false positive errors with respect to false negative ones, used in the `libsvm` objective function of the SVM learning algorithm (denoted in the following as w_{FP}).[1] For the monolithic classifier, the parameter values were chosen among all the possible combinations of the C values $\{0.001, 0.01, 0.1, 1, 10, 100\}$, and w_{FP} values $\{2, 5, 7, 10, 50, 100\}$. For

[1] We point out that the cost parameter w_{FP} of the SVM learning algorithm could not reflect the *real* cost considered in the task at hand: therefore its optimal value could be different from the real cost.

the individual classifiers of the MCS we considered the same C values above and the w_{FP} values $\{10, 50, 100, 500, 1000\}$.

As mentioned above, the MCS based on SpamAssassin tests was made up of the SpamAssassin text classifier and of $N-1$ linear classifiers trained on disjoint subsets of the 541 tests not associated to the text classifier. We considered two different ensemble sizes: $N = 3$ (namely, the text classifier and two linear classifiers) and $N = 11$. The 541 available tests (features) were distributed uniformly among the linear classifiers. The choice of which features associate to each classifier should be made in principle taking into account the kinds of the features. For instance, heterogeneous features could be fed to different classifiers. For the sake of simplicity, in our experiments we randomly split the features into N disjoint subsets (we just checked whether the false positive error rate of the MCS, estimated on the 2,000 e-mails of the training set, was higher than 0.01: in that case we disregarded the corresponding feature splitting).

The experiments described above were repeated five times on different randomly extracted training sets of the SVMs. Let us consider first the accuracy of the two kinds of classifiers (the monolithic classifier and the MCS). We report it in the Receiver Operating Characteristic (ROC) plane, in which the Y axis corresponds to the true positive classification rate (TP) and the X axis to the false positive rate (FP). In figure 5 (left) we show the TP and FP values of the monolithic classifier and of the two MCSs on the e-mails in the test set, obtained in the five runs of the experiments, as well as their average values across the five runs. For the sake of completeness we also report the whole ROC curve of the monolithic classifier (obtained by varying the decision threshold of the SVM). It can be seen that the accuracy of the monolithic classifier is slightly higher than the one of the two MCSs (it exhibits both higher TP and lower FP values). We remind the reader that this accuracy refers to the case when the adversary does not attack the classifiers. The hardness of evasion is instead shown in figure 5 (right). Since we are assuming that the adversary can only modify positive instances, only the FN rate can change under attack. Accordingly, in figure 5 (right) we report the FN rate as a function of the maximum number of features the adversary can evade. The FN rates for zero evaded tests correspond to the values reported in figure 5 (left). Figure 5 (right) clearly shows that, although the MCSs have a worse FN rate than the monolithic classifier when they are not under attack, they are harder to evade. For instance, if the adversary evades at most one feature, the FN rate of the two MCSs is between about 0.35 and 0.45, while the FN rate of the monolithic classifier is about 0.70, and so on. In other words, evading an MCS in which features are split across different classifiers required to evade a higher number of features than in the case they were processed by a monolithic classifier, as argued in section 3.2, although this comes at the expense of a slight increase in the false positive rate.

Consider finally a comprehensive plot showing the trade-off between the accuracy (when the adversary does not attack) and the hardness of evasion,

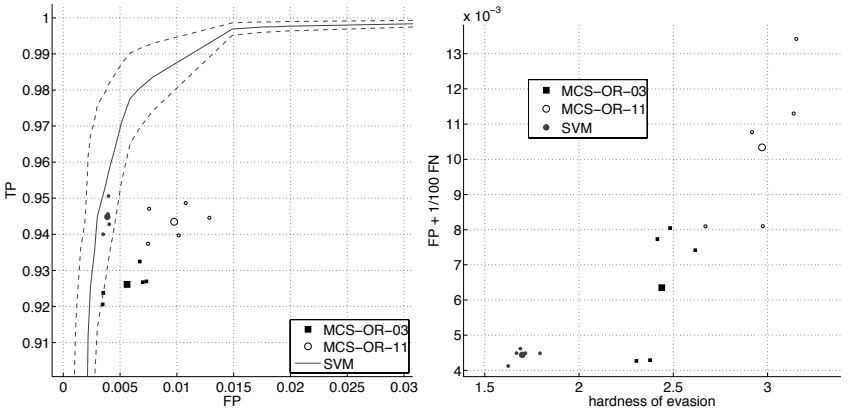

Fig. 5 Left: classification accuracy of the monolithic classifier (solid circles) and of the two MCSs (squares: three classifiers; white circles: eleven classifiers) in the TP, FP (ROC) plane. Small circles and small squares represent the values attained in the five runs of the experiments, while large ones represent the corresponding average values. The average ROC curve of the monolithic classifier is also shown (solid line), together with its standard deviation (dashed lines). Right: average FN rates with standard deviation as a function of the maximum number of features the adversary can evade. The FN rates when no feature is evaded correspond to the ones in the left plot.

as the in the scheme of figure 2. The accuracy (Y axis) is measured using the same trade-off between FP and FN rates as in the objective function of the monolithic classifier: $100 \times FP + FN$. The hardness of evasion is measured as explained in section 3.2, as the average minimum number of features that have to be evaded to evade the whole classifier. The accuracy-hardness of evasion trade-off attained by the monolithic classifier and by the MCSs is shown in figure 6. From this plot it is easy to see that the monolithic classifier attains a slightly higher accuracy (two to three times better than the MCSs), at the expense of a lower hardness of evasion (up to two times lower than that of the MCSs).

To sum up, the results presented in this section can be considered as a first experimental evidence, based on a formal setting, that the MCS architecture based on splitting features across different classifiers can be exploited in security tasks to improve the hardness of evasion in security systems.

5 Conclusions

Taking into account explicitly the presence of an intelligent, adaptive adversary in the design of classification systems for security applications, with the aim of making a classifier harder to evade, is a topic which has been addressed only recently in the machine learning and pattern recognition literature. So

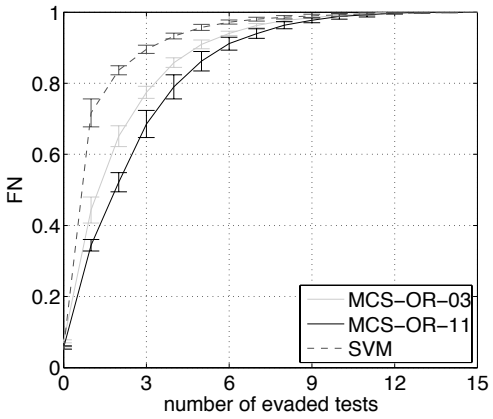

Fig. 6 Trade-off between the average classification accuracy (Y axis) and the average hardness of evasion (X axis) of the monolithic classifier (dashed line) and of the two MCSs (solid lines), over five runs of the experiments. Classification accuracy is measured as the false negative error rate. The hardness of evasion is measured in terms of the average minimum number of features the adversary has to evade, for evading the whole classifier. Horizontal bars represent the standard deviation of classification accuracy over the five runs of the experiments. Note that the area of the plot corresponding to the best accuracy-hardness of evasion trade-off is the bottom-right one.

far no general frameworks exist yet to deal with this problem. In this chapter we addressed this issue focusing on tasks like spam filtering and intrusion detection in computer networks, and on a classifier architecture based on the combination of an ensemble of classifiers. This architecture has been recently proposed by several authors and is used in commercial and open source products, but is supported so far only on by empirical and intuitive motivations. We tried to give a first answer, based on more formal motivations, to the questions of whether and how multiple classifier systems could allow to improve the hardness of evasion of a classifier. We considered in particular a defence strategy consisting in adding classifiers based on new features to a previous ensemble (as usually done in spam filters and intrusion detection systems to deal with new kinds of attacks), and to a design approach based on combining classifiers trained on disjoint subsets of features, instead of designing a monolithic classifier trained on the same features. We developed formal models of the corresponding classification systems and of possible adversary's strategies used to attack them (exploiting the framework developed in [3] to analyse the former strategy). We then gave an experimental evaluation on a case study related to the spam filtering task, using a real spam filter and a large and publicly available corpus of real spam e-mails.

Our results can be exploited as a starting point of future works aimed at formulating practical guidelines for the design of more robust classification systems in security applications.

Acknowledgements

We would like to thank Nilesh Dalvi and Mausam for providing us the code used in [3].

References

1. Barreno, M., Nelson, B., Sears, R., Joseph, A.D., Tygar, J.D.: Can machine learning be secure? In: ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security, pp. 16–25. ACM, New York, NY, USA (2006). DOI http://doi.acm.org/10.1145/1128817.1128824
2. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001). Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
3. Dalvi, N., Domingos, P., Mausam, Sanghai, S., Verma, D.: Adversarial classification. In: Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 99–108. Seattle (2004)
4. Giacinto, G., Roli, F., Didaci, L.: Fusion of multiple classifiers for intrusion detection in computer networks. Pattern Recognition Letters **24**, 1795–1803 (2003)
5. Globerson, A., Roweis, S.T.: Nightmare at test time: robust learning by feature deletion. In: W.W. Cohen, A. Moore (eds.) ICML, *ACM International Conference Proceeding Series*, vol. 148, pp. 353–360. ACM (2006)
6. Haindl, M., Kittler, J., Roli, F. (eds.): Multiple Classifier Systems, 7th International Workshop, MCS 2007, Prague, Czech Republic, May 23-25, 2007, Proceedings, *Lecture Notes in Computer Science*, vol. 4472. Springer (2007)
7. Jorgensen, Z., Zhou, Y., Inge, M.: A multiple instance learning strategy for combating good word attacks on spam filters. Journal of Machine Learning Research **9**, 1115–1146 (2008)
8. Kittler, J., Hatef, M., Duin, R.P., Matas, J.: On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence **20**(3), 226–239 (1998)
9. Lowd, D., Meek, C.: Adversarial learning. In: A. Press (ed.) Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). Chicago, IL. (2005)
10. Perdisci, R., Gu, G., Lee, W.: Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In: International Conference on Data Mining (ICDM), pp. 488–498. IEEE Computer Society (2006)
11. Ross, A.A., Nandakumar, K., Jain, A.K.: Handbook of Multibiometrics. Springer Publishers (2006)