Adversarial Pattern Classification Using Multiple Classifiers and Randomisation

Battista Biggio, Giorgio Fumera, and Fabio Roli

Dept. of Electrical and Electronic Eng., University of Cagliari,
Piazza d'Armi, 09123 Cagliari, Italy
{battista.biggio,fumera,roli}@diee.unica.it
http://prag.diee.unica.it

Abstract. In many security applications a pattern recognition system faces an *adversarial classification* problem, in which an intelligent, adaptive adversary modifies patterns to evade the classifier. Several strategies have been recently proposed to make a classifier harder to evade, but they are based only on qualitative and intuitive arguments. In this work, we consider a strategy consisting in hiding information about the classifier to the adversary through the introduction of some randomness in the decision function. We focus on an implementation of this strategy in a multiple classifier system, which is a classification architecture widely used in security applications. We provide a formal support to this strategy, based on an analytical framework for adversarial classification problems recently proposed by other authors, and give an experimental evaluation on a spam filtering task to illustrate our findings.

1 Introduction

Pattern recognition techniques are currently applied to several security applications, like biometric personal authentication, intrusion detection in computer networks and spam filtering [1,2,3,4,5]. However, these kinds of application do not fit the standard pattern classification model [6]. The main reason is that, in these applications, a pattern classification system faces an intelligent, adaptive adversary who engineers patterns to defeat the system itself. The machine learning community is becoming aware of the relevance of this problem, as a recent workshop held as part of the NIPS 2007 conference shows.[1] To date, few works have explicitly addressed this problem [7,8,9,10]. In [9], the general issue of how to design machine learning and pattern classification systems which are hard to evade for an adaptive adversary was discussed. A taxonomy of attack types for different security applications was proposed and some potential approaches to design "evade hard" classification systems were suggested for future work. Practical solutions for securing classification systems have been also proposed in applications like intrusion detection in computer systems [3]. However, such

[1] Workshop on Machine Learning in Adversarial Environments for Computer Security, http://nips.cc/Conferences/2007/Program/event.php?ID=615

solutions are indeed based on qualitative and intuitive arguments, besides the experimental evidence. In [7], the first attempt of developing an analytical framework for these kinds of problem (named *adversarial classification* problems) was made. An adversarial classification problem was formalised as a two-player game between a classifier and an adversary, in which both players try to maximise their expected utility by exploiting the knowledge they have about each other. In [7] it was *formally* shown that an adversary-aware classifier can perform much better than traditional adversary-unaware classifiers.

The aim of this work is to investigate and to provide a formal support to a new technique for designing hard to evade classifiers in adversarial classification problems, using the analytical framework developed in [7] (summarised in Sect. 2). Besides making a classifier adversary-aware, we consider the possibility of introducing some randomness in the placement of the classification boundary, which is a possible implementation of a general strategy suggested in [9], based on hiding information about the classifier to the adversary. We analyse the randomisation strategy in Sect. 3. In Sect. 4 we discuss a possible implementation using multiple classifier systems, which are widely used in security applications. We finally give an experimental validation of our findings in Sect. 5.

2 A Framework for Adversarial Pattern Classification

In [7], adversarial classification problems were formalised as two-player games. Each player chooses in his ply the move which maximises his expected utility (computed for that ply only, so using a greedy strategy[2]), based on the knowledge he has about the other player. The classifier faces a two-class problem in which patterns can be either malicious (if they are generated by the adversary) or innocent. Patterns are represented as n-dimensional feature vectors x in a feature space $\mathcal{X}$, and are viewed as instances of a random variable X generated i.i.d. from a given distribution $P(X)$ (without loosing generality, $\mathcal{X}$ is assumed to be discrete in [7]). Class labels y are denoted with $+$ (malicious) and $-$ (innocent). It is assumed that the adversary strategy to defeat the classifier consists in defining at each move a function $a : \mathcal{X} \rightarrow \mathcal{X}$ which modifies malicious patterns at operation phase, with the aim of making them being misclassified as innocent by the decision function constructed by the classifier at its previous move. The adversary can not modify malicious patterns at training phase, nor any innocent pattern. We point out that this assumption holds in several, but not all real applications (for instance, it holds in spam filtering tasks in which classifier training is made offline on a controlled training set, while it does not hold in some intrusion detection systems trained online). The classifier strategy consists instead in constructing at each move a decision function $y_{\mathrm{C}} : \mathcal{X} \rightarrow \{+, -\}$ which discriminates malicious and innocent patterns, taking into account the knowledge he has about modifications introduced by the adversary in his previous move. For given $a(x)$ and $y_{\mathrm{C}}(x)$, the utility function of the adversary is defined

[2] As explained in [7], computing the optimal strategies according to standard game theory is intractable for a classification problem.

as $U_{\mathrm{A}}(x, y, a, y_{\mathrm{C}}) = G_{\mathrm{A}}(y_{\mathrm{C}}(x), y) - W(x, a(x))$, where $G_{\mathrm{A}}(y_{\mathrm{C}}(x), y)$ is the gain accrued when a pattern (x, y) is labelled as $y_{\mathrm{C}}(x)$ (with $G_{\mathrm{A}}(-,+) > G_{\mathrm{A}}(+,+)$, since the adversary's gain is higher when a malicious pattern is mislabelled as innocent than when it is correctly classified), and $W(x, a(x))$ is the cost faced to modify a pattern x to $a(x)$ ($W(x, a(x)) > 0$, if $a(x) \neq x$, $W(x, a(x)) = 0$ otherwise). The corresponding expected utility is:

$$E[U_{\mathrm{A}}(x, y, a, y_{\mathrm{C}})] = \textstyle\sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P(x, y)[G_{\mathrm{A}}(y_{\mathrm{C}}(a(x)), y) - W(x, a(x))]. \quad (1)$$

Analogously, the utility function of the classifier is defined as $U_{\mathrm{C}}(x, y, a, y_{\mathrm{C}}) = G_{\mathrm{C}}(y_{\mathrm{C}}(a(x)), y) - \sum_{i=1}^{N} V_i$, where $G_{\mathrm{C}}(y_{\mathrm{C}}(x), y)$ is the gain defined similarly as G_{A} (with $G_{\mathrm{C}}(+,+) > G_{\mathrm{C}}(-,+)$, and $G_{\mathrm{C}}(-,-) > G_{\mathrm{C}}(+,-)$), and $V_i > 0$ is the cost of measuring the i-th feature. The expected utility is thus:

$$E[U_{\mathrm{C}}(x, y, a, y_{\mathrm{C}})] = \textstyle\sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P(x, y)[U_{\mathrm{C}}(y_{\mathrm{C}}(a(x)), y) - \sum_{i=1}^{N} V_i]. \quad (2)$$

If the adversary has complete knowledge of the decision function constructed by the classifier at the previous move [7], its optimal strategy at any given move is the function $a(x)$ which maximises his expected utility (1):

$$a(x) = \mathrm{argmax}_{x' \in \mathcal{X}}[G_{\mathrm{A}}(y_{\mathrm{C}}(x'), +) - W(x, x')], \quad (3)$$

if $y = +$, while $a(x) = x$, if $y = -$ (since he can modify only malicious patterns). Note that the above $a(x)$ amounts to choose between two alternatives. The first one is to keep x unchanged, and the corresponding utility is $G_{\mathrm{A}}(+,+)$ (x is a malicious pattern correctly classified, and there is no cost since it is not modified). The second one is to modify it to some $x' \neq x$. The modification with the highest gain, denoted with x^*, is the one which leads to a pattern x^* misclassified as legitimate, with the minimum cost $W(x, x^*) = \mathrm{argmin}_{x' \in \mathcal{X}} W(x, x')$. Such modification results in a gain equal to $G_{\mathrm{A}}(-,+) - W(x, x^*)$. It follows that the adversary will keep a malicious pattern x unchanged, if $G_{\mathrm{A}}(+,+) > G_{\mathrm{A}}(-,+) - W(x, x^*)$, namely if the minimum cost required to evade the classifier is higher than the corresponding increase in the gain, $G_{\mathrm{A}}(-,+) - G_{\mathrm{A}}(+,+)$. Otherwise x would be modified to the x^* defined above. Analogously, under the assumption that the classifier knows the modification function $a(x)$ devised by the adversary at his previous move, the optimal strategy of the classifier is to compute at each move the decision function $y_{\mathrm{C}}(\cdot)$ which maximises its expected utility (2).

Algorithms for computing the optimal strategies of adversary and classifier under the above framework strongly depend on the kind of classifier, on the kinds of feature used (for instance, on whether they are continuous or discrete), on the kinds of modification the adversary can make to malicious patterns and on the modification cost $W(\cdot, \cdot)$. In [7], specific algorithms were derived for the case in which the classifier is the naive Bayes. Experiments were also carried out on a spam filtering task, using features corresponding to words in the e-mail body, and assuming that the adversary can modify his e-mails by replacing words with synonyms or adding new words (these are real tricks used by spammers). These experiments showed that adversary-aware classifiers can perform much better than adversary-unaware ones designed with the traditional approach.

In the above analytical framework the hardness of evasion of a classifier is improved by making it adversary-aware, namely modifying it according to the knowledge on the strategy currently used by the adversary to defeat it. Although in real tasks the assumption that both players have complete knowledge about each other is likely to be violated, it is plausible that they can make at least some guesses about each other parameters [7] (for instance, the adversary can try to reverse engineer the classifier [10]). Using the same framework, in the next section we will formally show that the hardness of evasion can also be improved by hiding information about the decision function to the adversary.

3 Hardening Classifiers by Randomisation

An intuitive strategy for securing a classifier (namely, making it harder to evade) is to hide information about it to the adversary. A possible implementation of this strategy was suggested with qualitative arguments in [9]: introducing some randomness in the placement of the classification boundary. This implementation was not further investigated in [9] or in other works, to our knowledge. The goal of this section is to provide a more formal support to it, based on the analytical framework of [7]. To this aim, we consider a *single-shot* version of the game: first, the classifier computes its decision function $y_{\mathrm{C}}(\cdot)$ assuming that the adversary does not modify his patterns. Then the adversary computes the optimal modification function $a(\cdot)$. Our aim is to compare the performance of both players (their expected utility) under two conditions: when the adversary has complete knowledge about the decision function of the classifier, as in [7] (named from now on the *deterministic* case), and when the classifier introduces some randomness in it (the *non-deterministic* case). In the following we will denote quantities related to the two cases with the superscripts "det" (for "deterministic") and "rnd" (for "randomisation").

In the deterministic case the optimal strategy $a^{\mathrm{det}}(x)$ is the one which maximises the expected value of the utility function $U_{\mathrm{A}}^{\mathrm{det}}(x, y, a, y_{\mathrm{C}})$ over the distribution $P(x, y)$, for given $y_{\mathrm{C}}(\cdot)$ and $a(\cdot)$. In the considered framework, introducing randomness in the decision function means that $y_{\mathrm{C}}(\cdot)$ becomes a *random variable* $Y_{\mathrm{C}}(\cdot)$ for the adversary. We denote with $\mathcal{Y}_{\mathrm{C}}$ the domain of $Y_{\mathrm{C}}(\cdot)$ (namely, the set of decision functions which the classifier can implement). Accordingly, in the non-deterministic case the expected utility has to be computed with respect to the distribution $P(x, y, y_{\mathrm{C}}(\cdot))$. This distribution can be rewritten as $P(y_{\mathrm{C}}(a(x))|x, y)P(x, y)$. Note that $P(Y_{\mathrm{C}}(a(x)) = +|x, y)$ (written in the rest of the paper as $P(+|x, y)$ for the sake of brevity) is the probability that the classifier chooses a decision function $y_{\mathrm{C}}(\cdot)$ such that $y_{\mathrm{C}}(a(x)) = +$ (analogously for $P(Y_{\mathrm{C}}(a(x)) = -|x, y)$). Note that such probability is given by $\sum_{y_{\mathrm{C}}(\cdot) \in \mathcal{Y}_{\mathrm{C}}} P(Y_{\mathrm{C}}(\cdot) = y_{\mathrm{C}}(\cdot)|x, y) \times I[y_{\mathrm{C}}(a(x)), +]$, where $I[a, b] = 1$, if $a = b$, $I[a, b] = 0$ otherwise. Now, taking into account that the term $W(x, a(x))$

in (1) does not depend on $y_{\mathrm{C}}(a(x))$, the expected utility $E[U_{\mathrm{A}}]$ in the non-deterministic case can be written as:

$$\begin{aligned}&\textstyle\sum_{(x,y,y_{\mathrm{C}}(a(x)))\in\mathcal{X}\times\mathcal{Y}\times\mathcal{Y}} P(x,y,y_{\mathrm{C}}(a(x)))[G_{\mathrm{A}}(y_{\mathrm{C}}(a(x)),y) - W(x,A(x))]\\ &= \textstyle\sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} P(x,y)[P(+|x,y)G_{\mathrm{A}}(+,y) + P(-|x,y)G_{\mathrm{A}}(-,y) - W(x,a(x))].\end{aligned} \tag{4}$$

The term between square brackets in the second row of (4) is the conditional *expected* utility for a given pattern (x,y), taken over the distribution $P(y_{\mathrm{C}}(a(x))|x,y)$ (namely, over the possible realisations of the class boundary). In the determination of the optimal strategy it plays the same role as the utility function $U_{\mathrm{A}}^{\mathrm{det}}(x,y,a,y_{\mathrm{C}})$ in the deterministic case, and can be denoted here as $U_{\mathrm{A}}^{\mathrm{rnd}}(x,y,a)$: the optimal strategy $a^{\mathrm{rnd}}(x)$ is the one which maximises $U_{\mathrm{A}}^{\mathrm{rnd}}(x,y,a)$.

The two strategies $a^{\mathrm{det}}(x)$ and $a^{\mathrm{rnd}}(x)$ could be different, since the latter is chosen by maximising not the adversary's utility function for the decision function chosen by the classifier (which we denote now as $y_{\mathrm{C}}^*(\cdot)$), but instead its *expected* utility over all the possible decision functions. This means that the adversary could make sub-optimal choices in the non-deterministic case, with respect to the true but unknown $y_{\mathrm{C}}^*(\cdot)$, namely choices which would lead to a lower overall expected utility. More precisely, it can be easily shown (see below) that in the non-deterministic case the adversary *underestimates* the actual utility of each modification which evades the classifier, while he *overestimates* the actual utility of each modification which does not evade the classifier. This means that the adversary could make too conservative choices with respect to the optimal ones, to the extent that $a^{\mathrm{rnd}}(x)$ could keep unchanged some malicious pattern x correctly classified by $y_{\mathrm{C}}^*(\cdot)$, while $a^{\mathrm{det}}(x)$ would allow to evade the classifier. A simple example is shown in Fig. 1. Even more, $a^{\mathrm{rnd}}(x)$ could modify some malicious pattern x which evades $y_{\mathrm{C}}^*(\cdot)$ (and is thus kept unchanged by $a^{\mathrm{det}}(x)$) to a pattern x' which is correctly classified by $a^{\mathrm{rnd}}(x)$.

To prove the above assertion, consider any malicious pattern x $(y = +)$ and any modification x' which evades the classifier, namely $y_{\mathrm{C}}^*(x') = -$. The adversary's expected gain for x' is $P(+|x,+)G_{\mathrm{A}}(+,+) + P(-|x,+)G_{\mathrm{A}}(-,+)$, while the true gain is $G_{\mathrm{A}}(-,+)$. Since $G_{\mathrm{A}}(-,+) > G_{\mathrm{A}}(+,+)$, it follows that $U_{\mathrm{A}}^{\mathrm{rnd}}(x,+,x') < U_{\mathrm{A}}^{\mathrm{det}}(x,+,x',y_{\mathrm{C}}^*(x'))$, namely the adversary underestimates the true utility of modifying x to x'. Instead, if x' is correctly classified ($y_{\mathrm{C}}^*(\cdot) = +$), then the expected gain is the same as above, while the true gain is $G_{\mathrm{A}}(+,+)$. It follows that $U_{\mathrm{A}}^{\mathrm{rnd}}(x,+,x') > U_{\mathrm{A}}^{\mathrm{det}}(x,+,x',y_{\mathrm{C}}^*(x'))$, namely the adversary overestimates the true utility of modifying x to x'.

To sum up, we formally showed under the framework of [7] that randomising the decision boundary makes a classifier harder to evade, since the lack of information about the exact position of the decision boundary leads the adversary to make too conservative or too risky choices when deciding whether and how to modify a malicious pattern. One could raise the objection that in real tasks the adversary hardly ever has complete knowledge about the classifier decision function, and thus in practice it always acts in the non-deterministic case. However the above analysis can be easily extended to this case, to show more generally that the classifier can benefit by increasing the level of uncertainty of the

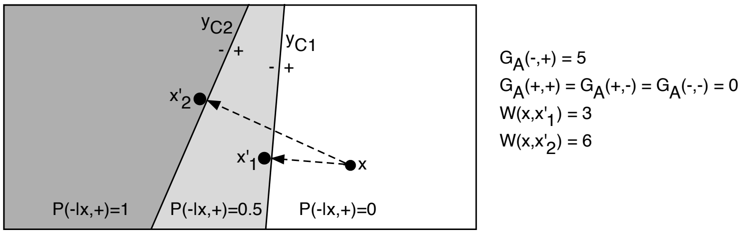

Fig. 1. Example of the non-optimality of the adversary's strategy in the non-deterministic case, for a given malicious pattern x in a 2-D feature space, assuming that the adversary's cost function is proportional to the Euclidean distance: $W(x, x') = |x - x'|$. The classifier chooses between two equiprobable decision functions y_{C_1} and y_{C_2} (both labelling patterns lying on the right as malicious), in a 2-D feature space. According to (3), in the deterministic case (when the adversary knows which decision function is used), the optimal strategy for x requires to choose between keeping x unchanged (with a gain of $G_A(+, +)$) and modifying it to x'_1 (if y_{C_1} is used, with a gain $G_A(-, +) - W(x, x'_1)$) or to x'_2 (if y_{C_2} is used, with a gain $G_A(-, +) - W(x, x'_2)$), being x'_1 and x'_2 the minimum cost modifications of x which allow to evade the corresponding decision function. If the adversary's gain and cost functions are the ones shown on the right, the optimal strategy against y_{C_1} is to modify x to x'_1, while against y_{C_2} is to keep x unchanged. In the non-deterministic case, patterns in the dark gray area are labeled as $-$ by both y_{C_1} and y_{C_2}, while patterns in the white area are labeled as $+$: accordingly, $P(-|, x, +)$ equals 1 in the former area and 0 in the latter. Patterns in the light gray area are labeled as $-$ by y_{C_1} only. Since y_{C_1} and y_{C_2} are equiprobable, $P(-|, x, +) = 0.5$ in that area. The optimal strategy in the non-deterministic case is to modify x to the pattern x' which maximises the conditional expected utility $P(+|x, +)G_A(+, +) + P(-|x, +)G_A(-, +) - W(x, x')$. This requires to choose between keeping x unchanged, modifying it to x'_1 or modifying it to x'_2 (any other choice has a higher cost but the same gain as one of these three choices). The best choice is to keep x unchanged. This means that the adversary's strategy in the non-deterministic case is optimal, only if the decision function is y_{C_2}. Otherwise the non-deterministic strategy does not allow to evade the classifier, while the deterministic strategy would have evaded it.

adversary, even starting from a non-zero uncertainty level (that is, by making $y_C(\cdot)$ still "more random"). On the other hand, it should be taken into account that an excessive randomisation could lead to select a classifier with a poor performance, to the extent that the advantage attained by hiding information to the adversary is lost. A trade-off is thus necessary between these two factors. Finally, we point out that randomisation can be viewed as a strategy to further improve the hardness of evasion of an adversary-aware classifier. Indeed, the analysis of the randomisation strategy can be extended to the repeated version of the game (when players continue to move indefinitely), allowing the classifier to retrain his set of decision functions according to the knowledge about the adversary's strategy $a^{\mathrm{rnd}}(x)$ (not reported here due to lack of space).

4 Evade Hard Multiple Classifiers Based on Randomisation Strategy

In real tasks the randomisation approach described in Sect. 3 can be implemented in several ways. It is important to understand that in the considered framework introducing randomness means preventing the adversary from having exact knowledge about one or more parameters involved in the design of the decision function. This can be achieved, for instance, tuning the classifier to a training set unknown by the adversary, or using some randomly selected parameter in the training phase (like the initial weights of a neural network). We propose here a specific implementation based on the use of MCSs. The reason is that MCSs turn out to be particularly suited in security applications as an alternative to the approach based on a "monolithic" classifier [2,3,1]. They are typically used for fusing classifiers each trained on a distinct feature-vector representation of patterns. The main motivations are derived from the field of classifier ensembles. It is indeed known that, if different sets of heterogeneous and loosely correlated features are available (as happens for IDSs), combining the outputs of different classifiers trained on different feature sets can be more effective than designing a single classifier in the feature space made up of all the available features (see for instance [11]). Moreover, if the overall number of features is large, such a single classifier would be more prone than a classifier ensemble to the so-called curse of dimensionality problem. Besides this, the MCS architecture is exploited (although often implicitly) in many commercial and open source spam filters and intrusion detection systems. Such kinds of system are usually made up of a set of independent modules which process a given subset of features, often focused on specific characteristics of malicious or innocent patterns, and provide a score denoting the likelihood that the input pattern is malicious. The final decision is then taken by fusing the scores with some combining function.

In this context, we point out that randomisation can be easily implemented exploiting one of the several techniques for constructing classifier ensembles, which are indeed based on randomisation like, for instance, bagging [12] and the random subspace method [13]. A concrete example of a security system with an MCS architecture is the SpamAssassin spam filter, in which filtering modules are provided with a default scoring system and the final decision is taken by thresholding the sum of the scores. However the user can modify the weights assigned to each module, for instance by tuning them to a given training set of legitimate and spam e-mails. Randomisation can be easily implemented here as the choice of a particular set of weights different than the default one. In sect. 5 we will present an experimental evaluation of the randomisation approach based on MCSs on a spam filtering task, using the SpamAssassin filter as the classifier.

5 Experimental Results

In this section we give an experimental evaluation on a spam filtering task of the effectiveness of class boundary randomisation for making a classifier harder to evade. We used the publicly available TREC 2007 data set,[3] made up of 25,220 legitimate and 50,199 spam e-mails. We used SpamAssassin as the classifier in its default configuration (all the available 619 different filtering rules were used, with their default scoring, and the decision threshold on the sum of the scores was set to 5), except for the embedded naive Bayes classifier, which was trained on the first 10,000 e-mails of the data set (in chronological order). We assumed that the adversary knew the decision threshold and the score produced by each module for any malicious pattern x. The randomisation was implemented as described in Sect. 3, by assigning different weights to each module. The weights were obtained using a support vector machine (SVM) classifier with linear kernel, trained on the scores produced by each filtering rule on 1,000 e-mails randomly drawn using the bootstrap technique from the 10,001st to 20,000th e-mails of the data set (the training set size of 1,000 was chosen since it provided a good trade-off between the uncertainty level of the adversary and the discriminant capability of the classifier). We computed 100 different sets of weights, and assumed that the classifier can choose one of them with identical probability. We also assumed that the adversary knew each set of weights and the fact that they are equiprobable.

Given that it was unfeasible to devise real modifications to the e-mails, tailored to each of the 619 filtering rules, we choose to simulate the effects of modifications by assuming that the adversary can arbitrarily reduce the score of each module. More precisely, let us denote with x_i the feature-vector representation of a given e-mail used by the i-th filtering rule, and with $s_i(x_i)$ the corresponding default score. We assumed that the adversary can modify any given spam e-mail x_i to any x'_i corresponding to any feasible value of the score such that $s_i(x_i) < s_i(x'_i)$[4] (note, however, that a modification is actually made by the adversary only if its cost is lower than the utility gain, as explained in Sect. 2). This simplifying assumption is in favour of the adversary, since we are not setting any constraint on how it can modify real spam e-mails. Given that it was also difficult to estimate the cost of modifications (or at least their relative difficulty) which allow to reduce the scores of the different modules, the modification cost $W_i(x_i, x'_i)$ for each module was assumed to be equal to the difference between the scores $s_i(x_i)$ and $s_i(x'_i)$ (which is always non-negative). Thus the total cost $W(x, x')$ is given by $\sum_{i=1}^{619}[s_i(x_i) - s_i(x'_i)]$. This is based on the reasonable assumption, in absence of more precise information, that attaining a higher score reduction requires a higher cost.

The experiments were made using the single-shot game setting explained in Sect. 3. Training the classifier corresponds to choose one out of the 100 weight sets. In the deterministic case, the adversary knows which weight set was chosen, and thus the corresponding decision function, and computes its optimal strategy $a^{\mathrm{det}}(\cdot)$ against that function. In the non-deterministic case, he has to compute the optimal strategy $a^{\mathrm{rnd}}(x)$ based on the distribution $P(y_{\mathrm{C}}|x, y)$ (see (4)). He evaluates such distribution based on the knowledge that one of the 100 possible

[3] `http://plg.uwaterloo.ca/~gvcormac/treccorpus07/`

[4] Note that almost all SpamAssassin filtering modules output discrete scores, often binary-valued.

Table 1. First row: SpamAssassin results on the 55,419 testing e-mails are shown for reference. Rows 2 and 3: average and standard deviation (between parentheses) of the expected utility of the adversary ($E[U_A]$) and of the classifier ($E[U_C]$), and of the percentage of false positive (FP) and false negative (FN) error rates. The superscripts 'det' and 'rnd' denote respectively the deterministic and non-deterministic case. Second row: the adversary does not modify malicious instances. Third row: the adversary modifies malicious instances according to the optimal strategy.

$E[U_A]^{det}$	$E[U_A]^{rnd}$	$E[U_C]^{det}$	$E[U_C]^{rnd}$	$FN^{det}(\%)$	$FN^{rnd}(\%)$	FP (%)
0.49	-	0.65	-	9.78	-	0.15
0.05 (0.04)	-	1.67 (0.38)	-	0.97 (0.81)	-	0.69 (0.37)
0.98 (0.13)	0.56 (0.05)	1.30 (0.41)	1.46 (0.38)	19.55 (2.56)	11.21(1.05)	0.69 (0.37)

and equiprobable weight sets was chosen. Note that, in our experimental setting, computing the optimal strategy amounts to evaluating, for each set of scores $s = (s_1(x_1), \ldots, s_{619}(x_{619}))$ corresponding to a given e-mail, the maximum of $U_A^{det}(s, s')$ and $U_A^{rnd}(s, s')$ (defined in Sect. 3) for all possible combinations of score values of the filtering rules.[5] We then computed the performance of both players both in the deterministic and non-deterministic case as their average expected utility over the 100 possible decision functions, evaluated on the last 55,419 e-mails of the data set (the ones not used for training). The optimal strategy of the adversary was computed on the same 55,419 e-mails. We used the following utility functions: $G_A(-,+) = 5, G_A(+,+) = G_A(-,+) = G_A(-,-) = 0$ (namely, the adversary gains 5 if it evades the classifier, and 0 if a spam e-mail is correctly classified, or whatever happens to legitimate e-mails), $G_C(-,+) = -1, G_C(+,-) = -100, G_C(+,+) = G_C(-,-) = 1$ (namely, the classifier looses 1 for misclassifying a spam e-mail as legitimate, he looses 100 for misclassifying a legitimate e-mail as spam, and gains 1 for correct classifications, taking into account the fact that in spam filtering false positive errors are much more costly than false negative ones. According to the above G_C, the SVM classifier used for determining the weights of the filtering rules was trained with imbalanced error costs (1 for false negative and 100 for false positive errors). We did not consider the cost the classifier faces for measuring features, given that it was a constant term in these experiments.

The expected utilites and the false positive and false negative error rates are reported in Table 5, together with the results of the default SpamAssassin configuration for reference. The second row shows values corresponding to the deterministic case (average and standard deviation over the 100 sets of weights), when the adversary does not modify malicious patterns. The third row shows the values corresponding to the deterministic case (averaged as explained above) and the non-deterministic case (average and standard deviation over the 100 sets of weights). It can be seen that, in the deterministic case, the expected utility of the adversary exhibits a very large increase attained by modifying malicious pattern

[5] It was possible to carry out an exhaustive search, as most SpamAssassin filtering modules have binary scores.

at operation phase. This is mirrored by a reduction in the expected utility of the classifier. The average false negative error rate gives a more clear picture: it raises from 0.97% to 19.55% (note that the false positive error rate does not change, since the adversary modifies only malicious patterns, and the classifier is never retrained). Instead, when the classifier randomises the decision function, the expected utility of the adversary drops to about half the value attained in the deterministic case, and the same happens to the false negative error rate, while the expected utility of the classifier increases to a value in the middle between the ones achieved in the deterministic case, with and without the adversary reaction. We point out that this are *pessimistic* results for the classifier, for two reasons. First, we assumed that the adversary knew the exact distribution of the decision function, $P(y_C|x,y)$, while in a real setting he can only estimate it. Second, we computed the adversary's optimal strategy on testing e-mails.

We can conclude that these experiments support the analytical results derived in Sect. 3, based on the analytical framework in [7], showing that hiding information to the adversary through the randomisation of the decision function can improve the hardness of evasion of a classifier. Our results also show that randomisation can be naturally implemented in a MCS architecture.

References

1. Ross, A.A., Nandakumar, K., Jain, A.K.: Handbook of Multibiometrics. Springer, Heidelberg (2006)
2. Giacinto, G., Roli, F., Didaci, L.: Fusion of multiple classifiers for intrusion detection in computer networks. Pattern Recognition Letters 24, 1795–1803 (2003)
3. Perdisci, R., Gu, G., Lee, W.: Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems. In: Proc. Int. Conf. Data Mining (ICDM), pp. 488–498. IEEE Computer Society, Los Alamitos (2006)
4. Sahami, M., Dumais, S., Heckerman, D., Horvitz, E.: A bayesian approach to filtering junk e-mail. AAAI Tech. Rep. WS-98-05, Madison, Wisconsin (1998)
5. Haindl, M., Kittler, J., Roli, F. (eds.): MCS 2007. LNCS, vol. 4472. Springer, Heidelberg (2007)
6. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification. Wiley, Chichester (2000)
7. Dalvi, N., Domingos, P., Mausam, Sanghai, S., Verma, D.: Adversarial classification. In: Proc. ACM Int. Conf. Knowledge Discovery Data Mining, pp. 99–108 (2004)
8. Globerson, A., Roweis, S.T.: Nightmare at test time: robust learning by feature deletion. In: Proc. Int. Conf. Mach. Learn., vol. 148, pp. 353–360. ACM, New York (2006)
9. Barreno, M., Nelson, B., Sears, R., Joseph, A.D., Tygar, J.D.: Can machine learning be secure? In: ASIACCS 2006: Proc. ACM Symp. Information, computer and communications security, pp. 16–25. ACM, New York (2006)
10. Lowd, D., Meek, C.: Adversarial learning. In: Proc. ACM Int. Conf. Knowledge Discovery Data Mining (KDD), pp. 641–647 (2005)
11. Kittler, J., Hatef, M., Duin, R.P., Matas, J.: On combining classifiers. IEEE Trans. Pattern Analysis and Machine Intelligence 20(3), 226–239 (1998)
12. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
13. Ho, T.K.: The random subspace method for constructing decision forests. IEEE Trans. Pattern Analysis and Machine Intelligence 20(8), 832–844 (1998)