

Adversarial Classification

Nilesh Dalvi Pedro Domingos Mausam Sumit Sanghai Deepak Verma
 Department of Computer Science and Engineering
 University of Washington, Seattle
 Seattle, WA 98195-2350, U.S.A.

{nilesh,pedrod,mausam,sanghai,deepak}@cs.washington.edu

ABSTRACT

Essentially all data mining algorithms assume that the data-generating process is independent of the data miner’s activities. However, in many domains, including spam detection, intrusion detection, fraud detection, surveillance and counter-terrorism, this is far from the case: the data is actively manipulated by an adversary seeking to make the classifier produce false negatives. In these domains, the performance of a classifier can degrade rapidly after it is deployed, as the adversary learns to defeat it. Currently the only solution to this is repeated, manual, *ad hoc* reconstruction of the classifier. In this paper we develop a formal framework and algorithms for this problem. We view classification as a game between the classifier and the adversary, and produce a classifier that is optimal given the adversary’s optimal strategy. Experiments in a spam detection domain show that this approach can greatly outperform a classifier learned in the standard way, and (within the parameters of the problem) automatically adapt the classifier to the adversary’s evolving manipulations.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications—*data mining*; I.2.6 [Artificial Intelligence]: Learning—*concept learning, induction, parameter learning*; I.5.1 [Pattern Recognition]: Models—*statistical*; I.5.2 [Pattern Recognition]: Design Methodology—*classifier design and evaluation, feature evaluation and selection*; G.3 [Mathematics of Computing]: Probability and Statistics—*multivariate statistics*

General Terms

Algorithms

Keywords

Cost-sensitive learning, game theory, naive Bayes, spam detection, integer linear programming

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD’04, August 22–25, 2004, Seattle, Washington, USA.
 Copyright 2004 ACM 1-58113-888-1/04/0008 ...\$5.00.

1. INTRODUCTION

Many major applications of KDD share a characteristic that has so far received little attention from the research community: the presence of an adversary actively manipulating the data to defeat the data miner. In these domains, deployment of a KDD system causes the data to change so as to make the system ineffective. For example, in the domain of email spam detection, standard classifiers like naive Bayes were initially quite successful (e.g., [23]). Unfortunately, spammers soon learned to fool them by inserting “non-spam” words into emails, breaking up “spam” ones with spurious punctuation, etc. Once spam filters were modified to detect these tricks, spammers started using new ones [4]. Effectively, spammers and data miners are engaged in a never-ending game where data miners continually come up with new ways to detect spam, and spammers continually come up with new ways to avoid detection.

Similar arms races are found in many other domains: computer intrusion detection, where new attacks circumvent the defenses put in place against old ones [17]; fraud detection, where perpetrators learn to avoid the actions that previously gave them away [5, 25]; counter-terrorism, where terrorists disguise their identity and activities in ever-shifting ways [11]; aerial surveillance, where targets are camouflaged with increasing sophistication [22]; comparison shopping, where merchants continually change their Web sites to avoid wrapping by shopbots [3]; file sharing, where media companies try to detect and frustrate illegal copying, and users find ways to circumvent the obstacles [14]; Web search, where webmasters manipulate pages and links to inflate their rankings, and search engines reengineer their ranking functions to deflate them back again [9, 16]; etc.

In many of these domains, researchers have noted the presence of adaptive adversaries and the need to take them into account (e.g., [4, 5, 11]), but to our knowledge no systematic approach for this has so far been developed. The result is that the performance of deployed KDD systems in adversarial domains can degrade rapidly over time, and much human effort and cost is incurred in repeatedly bringing the systems back up to the desired performance level. This paper proposes a first step towards automating this process. While complete automation will never be possible, we believe our approach and its future extensions have the potential to significantly improve the speed and cost-effectiveness of keeping KDD systems up to date with their adversaries.

Notice that adversarial problems cannot simply be solved by learners that account for concept drift (e.g., [10]): while these learners allow the data-generating process to change

Research Track Paper

over time, they do not allow this change to be a function of the classifier itself.

We first formalize the problem as a game between a cost-sensitive classifier and a cost-sensitive adversary (Section 2). Focusing on the naive Bayes classifier (Section 3), we describe the optimal strategy for the adversary against a standard (adversary-unaware) classifier (Section 4), and the optimal strategy for a classifier playing against this strategy (Section 5). We provide efficient algorithms for computing or approximating these strategies. Experiments in a spam detection domain illustrate the sometimes very large utility gains that an adversary-aware classifier can yield, and its ability to co-evolve with the adversary (Section 6). We conclude with a discussion of future research directions (Section 7).

2. PROBLEM DEFINITION

Consider a vector variable $X = (X_1, \dots, X_i, \dots, X_n)$, where X_i is the i^{th} feature or attribute, and let the instance space \mathcal{X} be the set of possible values of X . An instance x is a vector where feature X_i has the value x_i . Instances can belong to one of two classes: positive (malicious) or negative (innocent). Innocent instances are generated i.i.d. (independent and identically distributed) from a distribution $P(X|-)$, and malicious ones likewise from $P(X|+)$. The global distribution is thus $P(X) = P(-)P(X|-) + P(+)P(X|+)$. Let the training set \mathcal{S} and test set \mathcal{T} be two sets of (x, y) pairs, where x is generated according to $P(X)$ and y is the “true” class of x . We define adversarial classification as a game between two players: CLASSIFIER, which attempts to learn from \mathcal{S} a function $y_C = \mathcal{C}(x)$ that will correctly predict the classes of instances in \mathcal{T} , and ADVERSARY, which attempts to make CLASSIFIER classify positive instances in \mathcal{T} as negative by modifying those instances from x to $x' = \mathcal{A}(x)$. (ADVERSARY cannot modify negative instances, and thus $\mathcal{A}(x) = x$ for all $x \in -$.) CLASSIFIER is characterized by a set of cost/utility parameters (see Table 1 for a summary of the notation used in this paper):

1. V_i : Cost of measuring X_i . Depending on their costs, CLASSIFIER may choose not to measure some features.
2. $U_C(y_C, y)$: Utility of classifying as y_C an instance with true class y . Typically, $U_C(+, -) < 0$ and $U_C(-, +) < 0$, denoting the cost of misclassifying an instance (costs being negative utilities), and $U_C(+, +) > 0$, $U_C(-, -) > 0$.

ADVERSARY has a corresponding set of parameters:

1. $W_i(x_i, x'_i)$: Cost of changing the i^{th} feature from x_i to x'_i . $W_i(x_i, x_i) = 0$ for all x_i . We will also use $W(x, x')$ to represent the cost of changing an instance x to x' (which is simply the sum of the costs of all the individual feature changes made).
2. $U_A(y_C, y)$: Utility accrued by ADVERSARY when CLASSIFIER classifies as y_C an instance of class y . Typically, $U_A(-, +) > 0$, $U_A(+, +) < 0$ and $U_A(-, -) = U_A(+, -) = 0$, and we will assume this henceforth.

The goal of CLASSIFIER is to build a classifier \mathcal{C} that will maximize its expected utility, taking into account that instances may have been modified by ADVERSARY:

$$U_C = \sum_{(x,y) \in \mathcal{X}\mathcal{Y}} P(x,y) \left[U_C(\mathcal{C}(\mathcal{A}(x)), y) - \sum_{X_i \in \mathcal{X}_C(x)} V_i \right] \quad (1)$$

where $\mathcal{Y} = \{+, -\}$ and $\mathcal{X}_C(x) \subseteq \{X_1, \dots, X_n\}$ is the set of features measured by \mathcal{C} , possibly dependent on x . We call \mathcal{C} the *optimal strategy* of CLASSIFIER.

The goal of ADVERSARY is to find a feature change strategy \mathcal{A} that will maximize its own expected utility:

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

We call \mathcal{A} the *optimal strategy* of ADVERSARY. Notice that ADVERSARY will not change instances if the cost of doing so exceeds the utility of fooling CLASSIFIER. For example, a spammer will not modify his emails to the point where they no longer help sell his product. In practice, U_C and U_A are estimated by averages over \mathcal{T} : $U_C = (1/|\mathcal{T}|) \sum_{(x,y) \in \mathcal{T}} [U_C(\mathcal{C}(\mathcal{A}(x)), y) - \sum_{X_i \in \mathcal{X}_C(x)} V_i]$, etc.

Given two players, the actions available to each, and the payoffs from each combination of actions, classical game theory is concerned with finding a combination of strategies such that neither player can gain by unilaterally changing its strategy. This combination is known as a Nash equilibrium [7]. In our case, the actions are classifiers \mathcal{C} and feature change strategies \mathcal{A} , and the payoffs are U_C and U_A . As the following theorem shows, some realizations of the adversarial classification game always have a Nash equilibrium.

THEOREM 2.1. *Consider a classification game with a binary cost model for ADVERSARY, i.e., given a pair of instances x and x' , ADVERSARY can either change x to x' (incurring a unit cost) or it cannot (the cost is infinite). This game always has a Nash equilibrium, which can be found in time polynomial in the number of instances.*

We omit the proof due to lack of space. Unfortunately, the calculation of the Nash equilibrium requires complete and perfect knowledge of the probabilities of all the instances, which in practice ADVERSARY and CLASSIFIER will not have. Computing Nash equilibria will generally be intractable. The chief difficulty is that even in finite domains the number of available actions is doubly exponential in the number of features n . The best known algorithms for finding Nash equilibria in general (nonzero) sum games have worst-case exponential time in the number of actions, making them triply exponential in our case. Even using the more general notion of correlated equilibria, for which polynomial algorithms exist, the computational cost is still doubly exponential. Recent years have seen substantial work on computationally tractable approaches to game theory, but they focus mainly on scaling up with the number of players, not the number of actions [12]. Further, equilibrium strategies, either mixed or pure, assume optimal play on the part of the opponent, which is highly unrealistic in our case. When this assumption is not met, standard game theory gives no guidance on how to play. (This, and computational intractability, have significantly limited its practical use.) We thus leave the general existence and form of Nash or other equilibria in adversarial classification as an open

Symbol	Meaning
$X = (X_1, X_2, \dots, X_n)$	Instance.
$P(x)$	Probability distribution of untainted data.
X_i	i^{th} feature (attribute).
$\mathcal{X}, \mathcal{X}_i$	Domain of X and X_i , respectively.
x, x_i	An instance and the i^{th} attribute of that instance.
\mathcal{S}, \mathcal{T}	Training and test set.
$y_C = \mathcal{C}(x)$	The CLASSIFIER function.
$x_A = \mathcal{A}(x)$	The ADVERSARY transformation.
V_i	Cost of measuring X_i .
$U_C(y_C, y)$	Utility for CLASSIFIER of classifying as y_C an instance of class y .
$W_i(x_i, x'_i), W(x, x')$	Cost of changing the i^{th} feature from x_i to x'_i and instance x to x' , respectively.
$U_A(y_C, y)$	Utility accrued by ADVERSARY when CLASSIFIER classifies as y_C an instance of class y .
$\mathcal{X}_C(x)$	Set of features measured by \mathcal{C} .
$LO_C(x_i)$	Log-odds or ‘‘contribution’’ of i^{th} attribute to naive Bayes classifier $\left(\ln \frac{P(X_i=x_i +)}{P(X_i=x_i -)}\right)$.
$gap(x)$	$gap(x) > 0 \iff$ NB classifies x as positive $\left(LO_C(x) - \frac{U_C(-,-)-U_C(+,-)}{U_C(+,-)-U_C(-,+)}\right)$.
ΔU_A	ADVERSARY’s utility gain from successfully camouflaging a positive instance $(U_A(-,+)-U_A(+,+))$.
$\Delta LO_{i,x'_i}$	‘‘Gain’’ towards making x negative by changing i^{th} feature to x'_i $(LO_C(x_i) - LO_C(x'_i))$.
$MCC(x)$	Nearest instance (costwise) to x which Naive Bayes classifies as negative.
$x_{[i=x'_i]}$	An instance identical to x except that i^{th} attribute is changed to $x'_i \in \mathcal{X}_i$.
$P_A(x)$	Probability distribution after ADVERSARY has modified the data.

Table 1: Summary of the notation used in this paper.

question, and propose instead to start from a set of assumptions that more closely resembles the way adversarial classification takes place in practice: CLASSIFIER initially operates assuming the data is untainted (i.e., $\mathcal{A}(x) = x$ for all x); ADVERSARY then deploys an optimal plan $\mathcal{A}(x)$ against this classifier; CLASSIFIER in turn deploys an optimal classifier $\mathcal{C}(\mathcal{A}(x))$ against this adversary, etc. This approach has some commonality with evolutionary game theory [26], but the latter makes a number of assumptions that are inappropriate in our case (infinite population of players repeatedly matched at random, symmetric payoff matrices, players having offspring proportional to average payoff, etc.).

In this paper, we focus mainly on the *single-shot* version of the adversarial classification game: one move by each of the players. We touch only briefly on the *repeated* version of the game, where players continue to make moves indefinitely. A number of learning approaches to repeated games have been proposed [6], but these are also intractable in large action spaces. Other learning approaches focus on games with sequential states (e.g., [15]), while classification is stateless.

We make the assumption, standard in game theory, that all parameters of both players are known to each other. Although this is unlikely to be the case in practice, it is generally plausible that each player will be able to make a rough guess of the other’s (and, indeed, its own) parameters. Classification with imprecisely known costs and other parameters has been well studied in KDD (e.g., [20]), and extending this to the adversarial case is an important item for future work.

3. COST-SENSITIVE LEARNING

In this paper, we will focus on naive Bayes as the classifier to be made adversary-aware [2]. Naive Bayes is attractive because of its simplicity, efficiency, and excellent performance in a wide range of applications, including adversarial ones like spam detection [23]. Naive Bayes estimates the

probability that an instance x belongs to class y as

$$P(y|x) = \frac{P(y)}{P(x)} P(x|y) = \frac{P(y)}{P(x)} \prod_{i=1}^n P(x_i|y) \quad (3)$$

and predicts the class with highest $P(y|x)$. The denominator $P(x)$ is independent of the class, and can be ignored. $P(x|y) = \prod_{i=1}^n P(x_i|y)$ is the naive Bayes assumption. The relevant probabilities are learned simply by counting the corresponding occurrences in the training set \mathcal{S} .

We begin by extending naive Bayes to incorporate the measurement costs V_i and classification utilities $U_C(y_C, y)$ defined in the previous section, and to maximize the expected utility U_C (Equation 1). For now, we assume that no adversary is present (i.e., $\mathcal{A}(x) = x$ for all x). We remove this restriction in the next sections.

Cost-sensitive learning has been the object of substantial study in the KDD literature [1, 27]. Given a classification utility matrix $U_C(y_C, y)$, the *Bayes optimal prediction* for an instance x is the class y_C that maximizes the conditional utility $U(y_C|x)$:

$$U(y_C|x) = \sum_{y \in \mathcal{Y}} P(y|x) U_C(y_C, y) \quad (4)$$

This is simply Equation 1 conditioned on a particular x , and ignoring the adversary and measurement costs V_i . In naive Bayes, $P(y|x)$ is computed using Equation 3.

Measurement costs are incorporated into the choice of which subset of features to measure, $\mathcal{X}_C \subseteq \{X_1, \dots, X_n\}$. Intuitively, we want to measure feature X_i only if this improves the expected utility U_C by more than V_i . Since a feature’s effect on U_C will in general depend on what other features are being measured, finding the optimal \mathcal{X}_C requires a potentially exponential search. In practice, \mathcal{X}_C can be found using standard feature selection algorithms with U_C as the evaluation function. We use greedy forward selection ([13])

Research Track Paper

in our experiments. (Feature selection can also be carried out online, but we do not pursue that approach here.)

4. ADVERSARY STRATEGY

In this section, we formalize the notion of an optimal strategy for ADVERSARY. We model it as a constrained optimization problem, which can be formulated as an integer linear program. We then propose a pseudo-linear time solution to the integer LP, based on dynamic programming. We make the following assumptions.

ASSUMPTION 1. Complete Information: Both CLASSIFIER and ADVERSARY know all the relevant parameters: V_i, U_C, W_i, U_A and the naive Bayes model learned by CLASSIFIER on \mathcal{S} (including $\mathcal{X}_C, P(y)$, and $P(x_i|y)$ for each feature and class).

ASSUMPTION 2. ADVERSARY assumes that CLASSIFIER is unaware of its presence (i.e., ADVERSARY assumes that $\mathcal{C}(x)$ is the naive Bayes model described in the previous section).

To defeat CLASSIFIER, ADVERSARY needs only to modify features in \mathcal{X}_C , since the others are not measured. From Equation 3:

$$\log \frac{P(+|x)}{P(-|x)} = \log \frac{P(+)}{P(-)} + \sum_{x_i \in \mathcal{X}_C} \log \frac{P(x_i|+)}{P(x_i|-)} \quad (5)$$

For brevity, we will use the notation $LOC(x) = \log \frac{P(+|x)}{P(-|x)}$ and $LOC(x_i) = \log \frac{P(x_i|+)}{P(x_i|-)}$, where LO is short for “log odds.” Naive Bayes classifies an instance x as positive if the expected utility of doing so exceeds that of classifying it as negative, i.e., if $U_C(+, +)P(+|x) + U_C(+, -)P(-|x) > U_C(-, +)P(+|x) + U_C(-, -)P(-|x)$, or

$$\frac{P(+|x)}{P(-|x)} > \frac{U_C(-, -) - U_C(+, -)}{U_C(+, +) - U_C(-, +)} \quad (6)$$

Let the log of the right hand side be $LT(U_C)$ (*log threshold*). Then naive Bayes classifies instance x as positive if $LOC(x) > LT(U_C)$, or equivalently if $gap(x) > 0$, where $gap(x) = LOC(x) - LT(U_C)$. If the instance is classified as negative, ADVERSARY does not need to do anything. Let us assume, then, that x is classified as positive, i.e., $gap(x) > 0$. The objective of ADVERSARY is to make some set of feature changes to x that will cause it to be classified as negative, while incurring the minimum possible cost. This causes ADVERSARY to gain a utility of $\Delta U_A = U_A(-, +) - U_A(+, +)$. Thus ADVERSARY will transform x as long as the total cost incurred is less than ΔU_A and not otherwise.

We formulate the problem of finding an optimal strategy for ADVERSARY as an integer linear program. Recall that \mathcal{X}_i is the domain of X_i . For $x'_i \in \mathcal{X}_i$, let δ_{i,x'_i} be an integer (binary) variable which takes the value one if the feature X_i is changed from x_i to x'_i , and zero otherwise. Let the new data item thus obtained be x' . The cost of transforming x to x' is $W(x, x') = \sum_i W_i(x_i, x'_i)$, and the resulting change in log odds is $LOC(x') - LOC(x) = \sum_i (LOC(x'_i) - LOC(x_i))$. Define $\Delta LO_{i,x'_i} = LOC(x_i) - LOC(x'_i)$. This is the *gain* in ADVERSARY’s objective of making the instance negative. Note that $\Delta LO_{i,x_i} = 0$; this represents the case where X_i has not been changed. To transform x so that the new instance is classified as negative, ADVERSARY needs to change the values of

some features such that the sum of their gains (decrease in log odds) is more than $gap(x)$.

Thus, to find the minimum cost changes required to transform this instance into a negative instance, we need to solve the following integer (binary) linear program:

$$\begin{aligned} \min \left\{ \sum_{x_i \in \mathcal{X}_C} \sum_{x'_i \in \mathcal{X}_i} W_i(x_i, x'_i) \delta_{i,x'_i} \right\} \quad \text{s.t.} \\ \sum_{x_i \in \mathcal{X}_C} \sum_{x'_i \in \mathcal{X}_i} \Delta LO_{i,x'_i} \delta_{i,x'_i} \geq gap(x) \\ \delta_{i,x'_i} \in \{0, 1\}, \quad \sum_{x'_i \in \mathcal{X}_i} \delta_{i,x'_i} \leq 1 \end{aligned}$$

The binary δ_{i,x'_i} values encode which features are changed to which values. The optimizing equation minimizes the cost incurred in this transformation. The first constraint makes sure that the new instance will be classified as negative. The second constraint encodes the requirement that a feature can only have a single value in an instance. We will call the transformed instance obtained by solving this integer linear program the *minimum cost camouflage (MCC)* of x . In other words, $MCC(x)$ is the nearest instance (costwise) to x which naive Bayes classifies as negative.

After solving this integer LP, ADVERSARY transforms the instance only if the minimum cost obtained is less than ΔU_A . Therefore, letting $NB(x)$ be the naive Bayes class prediction for x ,

$$A(x) = \begin{cases} MCC(x) & \text{if } NB(x) = +, W(x, MCC(x)) < \Delta U_A \\ x & \text{otherwise} \end{cases} \quad (7)$$

The above integer (binary) LP problem is NP-hard, as the 0-1 knapsack problem can be reduced to it [8]. However, a pseudo-linear time algorithm can be obtained by discretizing LOC , which allows dynamic programming to be used. Although the algorithm is approximate, it can compute the solution to arbitrary precision.

The procedure is shown in Algorithm 1. Function $FINDMCC(i, w)$ computes the minimum cost needed to change the log odds of x by w using only the first i features. It returns the pair $(MinCost, MinList)$ where $MinCost$ is the minimum cost and $MinList$ is a list of feature-value pairs denoting the changes to be made to x . (In each pair, i is the feature index and x'_i is the value it should be changed to.) To obtain the optimal adversary strategy, we need to compute $FINDMCC(n, W)$, where the integer W is $gap(x)$ after discretization and n is the number of features in \mathcal{X}_C . Note that $\Delta LO_{i,x'_i}$ is now a (non-negative) integer in the discretized log odds space.

The algorithm can be efficiently implemented using top-down recursion with memoization (so that no recursive call is computed more than once). Note that although the features can be considered in any order, some orderings may find solutions faster than the others. If, in the discretized space, instance x requires a gap of W to be filled by the transformation, then the algorithm runs in time at most $O(W \sum_i |\mathcal{X}_i|)$ (since the **for** loop in $FINDMCC$ is called at most W times and each time it takes $O(\sum_i |\mathcal{X}_i|)$ time). Hence it is pseudo-linear in the number of features. Pseudo-

linearity may be expensive for large values of W or for cases where features have large domains. We now present two pruning rules, one for use in the first situation, and one for the second.

Algorithm 1 FINDMCC(i, w)

```

if  $w \leq 0$  then
    return  $(0, \{\})$ 
end if
if  $i = 0$  then
    return  $(\infty, Undefined)$ 
end if
 $MinCost \leftarrow \infty$ 
 $MinList \leftarrow Undefined$ 
for  $x'_i \in \mathcal{X}_i$  do
    if  $\Delta LO_{i, x'_i} \geq 0$  then
         $(CurCost, CurList) \leftarrow \text{FINDMCC}(i-1, w - \Delta LO_{i, x'_i})$ 
         $CurCost \leftarrow CurCost + W_i(x_i, x'_i)$ 
         $CurList \leftarrow CurList + (i, x'_i)$ 
        if  $CurCost < MinCost$  then
             $MinCost \leftarrow CurCost$ 
             $MinList \leftarrow CurList$ 
        end if
    end if
end for
return  $(MinCost, MinList)$ 
    
```

Algorithm 2 $\mathcal{A}(x)$

```

 $W \leftarrow \text{gap}(x)$  (discretized).
 $(MinCost, MinList) \leftarrow \text{FINDMCC}(n, W)$ 
if  $NB(x) = +$  and  $MinCost < \Delta U_A$  then
     $newx \leftarrow x$ 
    for all  $(i, x'_i) \in MinList$  do
         $newx_i \leftarrow x'_i$ 
    end for
    return  $newx$ 
else
    return  $x$ 
end if
    
```

LEMMA 4.1. *If*

$$\max_{i, x'_i} \left(\frac{\Delta LO_{i, x'_i}}{W_i(x_i, x'_i)} \right) < \frac{\text{gap}(x)}{\Delta U_A}$$

then $\mathcal{A}(x) = x$.

This lemma is easy to prove and can be used to detect the instances for which $MinCost > \Delta U_A$. Instances which are positive by very large $\text{gap}(x)$ values can thus be pruned early on, and we need to run the algorithm only for more reasonable values of $\text{gap}(x)$.

Our second pruning strategy can be employed in situations where the cost metric is sufficiently coarsely discretized. We globally sort all the (i, x'_i) tuples in increasing order of $W_i(x_i, x'_i) \geq 0$. For identical values of $W_i(x_i, x'_i)$, we use decreasing order of $\Delta LO_{i, x'_i}$ as the secondary key. For a particular i , $W_i(x_i, x'_i)$ combination, over all i , we can remove all but the first entry in the list. This is valid because, if the X_i is changed in the optimal solution, then taking the

value x'_i with the highest $\Delta LO_{i, x'_i}$ will also yield the optimal solution. We can prune even further by only considering the first k tuples in each W such that $\sum_{j=1}^{i-1} \Delta LO_{j, x'_j} > \text{gap}(x)$ and $\sum_{j=1}^k \Delta LO_{j, x'_j} < \text{gap}(x)$. It is easy to see that this pruning does not affect the optimal solution.

Thus, if the feature-changing costs W_i are sufficiently coarsely discretized, we will never need to consider more than a few tuples for each integer value of W_i . Our algorithm will thus run efficiently even when the domains of features are large.

5. CLASSIFIER STRATEGY

We now describe how CLASSIFIER can adapt to the adversary strategy described in the previous section. We derive the optimal $\mathcal{C}(x)$ taking into account $\mathcal{A}(x)$, and give an efficient algorithm for computing it. We make the following additional assumptions.

ASSUMPTION 3. CLASSIFIER *assumes that* ADVERSARY *uses its optimal strategy to modify test instances (Algorithm 2).*

ASSUMPTION 4. *The training set* \mathcal{S} *used for learning the initial naive Bayes classifier is not tampered with by* ADVERSARY *(i.e.,* \mathcal{S} *is drawn from the real distribution of adversarial and non-adversarial data).*

ASSUMPTION 5. $\forall X_i \in \mathcal{X}$, $W_i(x_i, x'_i)$ *is a semi-metric, i.e., it has the following properties:*

1. $W_i(x_i, x'_i) \geq 0$ *and the equality holds iff* $x_i = x'_i$
2. $W_i(x_i, x''_i) \leq W_i(x_i, x'_i) + W_i(x'_i, x''_i)$

The above also implies that $W(x, x'') \leq W(x, x') + W(x', x'')$. The triangular inequality for cost holds in most real domains. This is because to change a feature from x_i to x''_i the adversary always has the option of changing it via x'_i , i.e., with x'_i as an intermediate value.

The goal of CLASSIFIER, as in Section 3, is to predict for each instance x' the class that maximizes its conditional utility (Equation 4). The difference is that now we want to take into account the fact that ADVERSARY has tampered with the data. Of all the probabilities used by CLASSIFIER (Equation 3), the only one that is changed by ADVERSARY is $P(x'|+)$; $P(+)$, $P(-)$ and $P(x'|-)$ remain unaltered. Let $P_A(x'|+)$ be the post-adversary version of $P(x'|+)$. Then

$$P_A(x'|+) = \sum_{x \in \mathcal{X}} P(x|+) P_A(x'|x, +) \quad (8)$$

In other words, the probability of observing an instance x' is the probability that the adversary generates some instance x and then modifies it into x' , summed over all x . Since $P_A(x'|x, +) = 1$ if $\mathcal{A}(x) = x'$ and $P_A(x'|x, +) = 0$ otherwise,

$$P_A(x'|+) = \sum_{x \in \mathcal{X}_A(x')} P(x|+) \quad (9)$$

where $\mathcal{X}_A(x') = \{x : x' = \mathcal{A}(x)\}$. There are two cases where ADVERSARY will leave an instance x untampered (i.e., $\mathcal{A}(x) = x$): when naive Bayes predicts it is negative, since then no action is necessary, and when there is no transformation of x whose cost is lower than the utility gained by making it appear negative. Thus

$$P_A(x'|+) = \left(\sum_{x \in \mathcal{X}'_A(x')} P(x|+) \right) + I(x')P(x'|+) \quad (10)$$

where $\mathcal{X}'_A(x') = \mathcal{X}_A(x') \setminus \{x'\}$, $I(x') = 1$ if $NB(x') = -$ or $W(x', MCC(x')) \geq \Delta U_A$, and $I(x') = 0$ otherwise (see Equation 7 and Algorithm 2). The untampered probabilities $P(x'|+)$ are estimated using the naive Bayes model (Equation 3): $P(x'|+) = \prod_{X_i \in \mathcal{X}_C} P(X_i = x'_i|+)$.¹ The optimal adversary-aware classification algorithm $\mathcal{C}(x')$ is shown below, with $\hat{P}()$ used to denote the probability $P()$ estimated from the training data \mathcal{S} . $\hat{P}_A(x'|+)$ is given by Equation 10 using the empirical estimates of $P(x'|+)$. The second term in Equation 10, $I(x')P(x'|+)$, is easy to compute given calls to $NB(x')$ and Algorithm 1 to determine if x has a feasible camouflage. The remainder of this section is devoted to efficiently computing the first term, $\sum_{x \in \mathcal{X}'_A(x')} P(x|+)$.

Algorithm 3 $\mathcal{C}(x')$

```

 $P_{x'}^- \leftarrow \hat{P}(-) \prod_i \hat{P}(X_i = x'_i|-)$ 
 $P_{x'}^+ \leftarrow \hat{P}(+) \hat{P}_A(x'|+)$ 
 $U(+|x') \leftarrow P_{x'}^+ U_C(+, +) + P_{x'}^- U_C(-, +)$ 
 $U(-|x') \leftarrow P_{x'}^+ U_C(-, +) + P_{x'}^- U_C(-, -)$ 
if  $U(+|x') > U(-|x')$  then
    return +
else
    return -
end if
    
```

One solution is to iterate through all possible positive examples and check if x' is their minimum cost camouflage. This is, of course, not feasible. We now study some theoretical properties of the MCC function which will later be used to prune this search. Recall that if $NB(x) = -$ then $gap(x) < 0$ and vice versa. We define $x_{[i=x'_i]}$ as a data instance which is identical to x except that its i th feature is changed to x'_i .

LEMMA 5.1. *Let x_A be any positive instance and let $x' = MCC(x_A)$. Then, $\forall i$,*

$$(x_A)_i \neq x'_i \Rightarrow gap(x') + LO_C((x_A)_i) - LO_C(x'_i) > 0$$

PROOF. Let $x'' = x'_{[i=(x_A)_i]}$. This implies that $W(x_A, x'') < W(x_A, x')$, since x'' differs from x_A on one less feature than x' . Also $gap(x'') = gap(x') + LO_C((x_A)_i) - LO_C(x'_i)$. Since x' is $MCC(x_A)$ and $W(x_A, x'') < W(x_A, x')$, $NB(x'')$ must be $+$, and therefore $gap(x'') > 0$, proving the result. \square

Given a negative instance x' , for each feature we compute all values v that satisfy Lemma 5.1. To compute $\mathcal{X}'_A(x')$, we only need to take combinations of these feature-value pairs and check if x' is their MCC . This can substantially reduce the number of positive instances in our search space. The search space can still potentially contain an exponential number of instances. However, after we employ the next theorem, we obtain a fast algorithm for estimating the set $\mathcal{X}'_A(x')$.

¹Notice that the optimal feature subset \mathcal{X}_C for the adversary-aware classifier may be different from the adversary-unaware one, but can be found using the same methods (see Section 3).

THEOREM 5.2. *Let x_A be a positive instance such that $x' = MCC(x_A)$. Let \mathcal{D} be the set of features that are changed in x_A to obtain x' . Let \mathcal{E} be a non-trivial subset of \mathcal{D} , and let x'_A be an instance that matches x' for all features in \mathcal{E} and x_A for all others, i.e., $(x'_A)_i = x'_i$ if $X_i \in \mathcal{E}$, $(x'_A)_i = (x_A)_i$ otherwise. Then $x' = MCC(x'_A)$.*

PROOF. By contradiction. Suppose $x'' = MCC(x'_A)$ and $x'' \neq x'$. Then $W(x'_A, x'') < W(x'_A, x')$. Also, since $\mathcal{E} \subset \mathcal{D}$, by definition of $W(x, y)$ we have $W(x_A, x') = W(x_A, x'_A) + W(x'_A, x')$. So by the triangle inequality

$$\begin{aligned} W(x_A, x'') &\leq W(x_A, x'_A) + W(x'_A, x'') \\ &< W(x_A, x'_A) + W(x'_A, x') \\ &= W(x_A, x') \end{aligned}$$

Thus $W(x_A, x'') < W(x_A, x')$, which gives a contradiction, since then $x' \neq MCC(x_A)$. This completes the proof. \square

The above theorem implies that if x_A is a positive instance such that $x' \neq MCC(x_A)$ then x' cannot be the MCC of any other instance x'_A such that the changed features from x_A to x' form a superset of the changed features from x_A to x' . We now use the following result to obtain bounds on $\mathcal{X}'_A(x')$.

COROLLARY 5.3. *Let FV be the set of feature-value pairs that satisfy Lemma 5.1. Let $GV \subset FV$ be such that $(i, x_i) \in GV$ if $x'_{[i=x_i]} \in \mathcal{X}'_A(x')$. Then for every $x_A \in \mathcal{X}'_A(x')$, the set of feature-value pairs where x_A and x' differ form a subset of GV .*

From the above corollary, after we compute GV , we only need to consider the combinations of feature-value pairs that are in GV and change those in the observed instance x' . Theorem 5.2 also implies that performing single changes from GV returns instances in $\mathcal{X}'_A(x')$. This gives us the following bounds on $\sum_{x \in \mathcal{X}'_A(x')} P(x|+)$.

THEOREM 5.4. *Let x' be any instance and let GV be the set defined in Corollary 5.3. Let $G = \{i | \exists x_i (i, x_i) \in GV\}$ and let $\mathcal{X}_i^G = \{x_i \in \mathcal{X}_i | (i, x_i) \in GV\}$. Then*

$$\begin{aligned} \sum_{(i, x_i) \in GV} P(x'_{[i=x_i]}|+) &\leq \sum_{x \in \mathcal{X}'_A(x')} P(x|+) \leq \\ &\left\{ \prod_{i \in G} \left[1 + \sum_{x_i \in \mathcal{X}_i^G} \frac{P(x'_{[i=x_i]}|+)}{P(x'|+)} \right] - 1 \right\} \end{aligned}$$

PROOF. The proof of the first inequality follows directly from Theorem 5.2, which states that changing any single feature of x' to any value in GV returns an instance from $\mathcal{X}'_A(x')$. To prove the second inequality, we observe that the expression on the right side, when expanded, gives the sum of probabilities of all possible changes in x' due to the set GV , and $\mathcal{X}'_A(x')$ is a subset of those instances. \square

Given these bounds, we can classify a test instance as follows. If plugging the lower bound into Algorithm 3 gives $U(+|x') > U(-|x')$, then the instance can be safely classified as positive. Similarly, if using the upper bound gives $U(+|x') < U(-|x')$, then the instance is negative. If GV is large, so is the lower bound on $P_A(\mathcal{X}'_A(x')|+)$. If GV is small, we can do an exhaustive search over the subsets of GV and check if each of the items considered belongs to $\mathcal{X}'_A(x')$. In our experiments we find that using the lower bound for making predictions works well in practice.

6. EXPERIMENTS

We implemented an adversarial classifier system for the spam filtering domain. Spam is an attractive testbed for our methods because of its practical importance, its rapidly evolving adversarial nature, the wide availability of data (in contrast to many other adversarial domains), the fact that naive Bayes is the *de facto* standard classifier in this area, and its richness as a challenge problem for KDD [4]. One disadvantage of spam as a testbed is that feature measurement costs are generally negligible, leaving this part of our framework untested. (In contrast, in a domain like counterterrorism feature measurements are a major issue, often requiring large numbers of personnel and expensive equipment, raising privacy issues, and imposing costs on millions of individuals and transactions.)

We used the following two datasets in our experiments:

Ling-Spam [24]: This corpus contains the legitimate discussions on a linguistics mailing list and the spam mails received by the list. There are 2412 non-spam messages and 481 spam ones. Thus, around 16.6% of the corpus is spam.

Email-Data [19]: This corpus consists of texts from 1431 emails, with 642 non-spam messages (conferences (370) and jobs (272)) and 789 spam ones.

Each of these datasets was divided into ten parts for ten-fold cross-validation. We defined three scenarios, as described below, and applied our implementation of naive Bayes (NB) and the adversary-aware classifier (AC) to each. We used `ifile` [21] for preprocessing emails.

6.1 Scenarios

The three spam filtering scenarios that we implemented differ in how the email is represented for the classifier, how the adversary can modify the features, and at what cost.

Add Words (AW): This the simplest scenario. The binomial model of text for classification is used [18]: there is one Boolean feature per word, denoting whether or not the word is present in the email. The only way to modify the email is by adding words which are not already present, and each word added incurs unit cost. This is akin to saying that the original mail has content that the spammer is not willing to change, and thus the spammer only adds unnecessary words to fool the spam detector. In this model, ADVERSARY’s strategy reduces to a greedy search where it adds words in decreasing order of LO_C .

Add Length (AL): This model is very similar to AW, except that the cost of adding a word is proportional to the number of characters in it. This corresponds to a hypothetical situation where ADVERSARY needs to pay a certain amount per bit of mail transmitted, and wants to minimize the number of characters sent.

Synonym (SYN): Generally, spammers want to avoid detection while preserving the semantic content of their messages. Thus, in this scenario we consider the case where ADVERSARY changes the mail by replacing the existing words by other semantically similar words. For example, a spammer attempting to sell a product would like to send emails claiming it to be cheap, but

	(+, +)	(+, -)	(-, +)	(-, -)
U_A	0	0	20	0
U_C	1	-10/-100/-1000	-1	1

Table 2: Utility matrices for Adversary and Classifier used in the experiments.

without the use of words like “free,” “sale”, etc. This is because the naive Bayes classifier uses the presence or absence of specific words with high LO_C to classify emails, independent of their actual meaning. Given the above intent, we define this scenario as follows. We use the multinomial model of text [18]: an email is viewed a sequence of word positions, with one feature per position, and the domain of each feature is the set of words in the vocabulary. In this case, the number of times a word occurs in an email is important. However, the word order is disregarded (i.e., the probability of word occurrence is assumed to be independent of location). For each word, we obtain a list of synonyms from the WordNet lexical database [28]. A word in an email can then be changed only to one of its synonyms, at unit cost.

It is easy to see that the costs used in all scenarios are metrics, so we can apply Lemma 5.1 and Theorem 5.2.

The U_A classification utility matrix for ADVERSARY we used is such that whenever a spam email is classified as non-spam the adversary receives a utility of 20, and all other entries are 0. Thus, in the SYN and AW scenarios 20 word replacements/additions are allowed. In the AL scenario, the cost of adding a character is set to 0.1, and as a result 200 character additions are allowed.

For CLASSIFIER, we ran the experiments with three different utility matrices (U_C). All matrices had a utility of 1 for correctly classifying an email and a penalty (negative utility) of 1 for incorrectly classifying a spam email as non-spam. The penalty for incorrectly classifying a non-spam email as spam was set to 10 in one matrix, 100 in another, and 1000 in the third. This reflects the fact that, in spam filtering, the critical and dominant cost is that of false positives: letting a single spam email get through to the user is a relatively insignificant event, but filtering out a non-spam email is highly undesirable (and potentially disastrous). The different $U_C(+, -)$ values correspond to the different values of the λ parameter in Sakkis et al [24]. Table 2 summarizes the utility parameters used in the experiments.

6.2 Results

The results of running the various algorithms on the Ling-Spam and Email-data datasets are shown in Figures 1 and 2 respectively. The figures show the average utilities obtained (with a maximum value of 1.0) by naive Bayes and the adversary-aware classifier under the different scenarios and different U_C matrices. The utility of naive Bayes on the original, untampered data (“PLAIN”) is represented by the black bar on the left. The remaining black bars represent the performance of naive Bayes on tainted data in the three scenarios, and the white bars the performance of the corresponding adversary-aware classifier. We observe that ADVERSARY significantly degrades the performance of naive Bayes in all three scenarios and with all three CLASSIFIER utility matrices. This effect is more pronounced in the Email-Data set because it has a higher percentage of spam

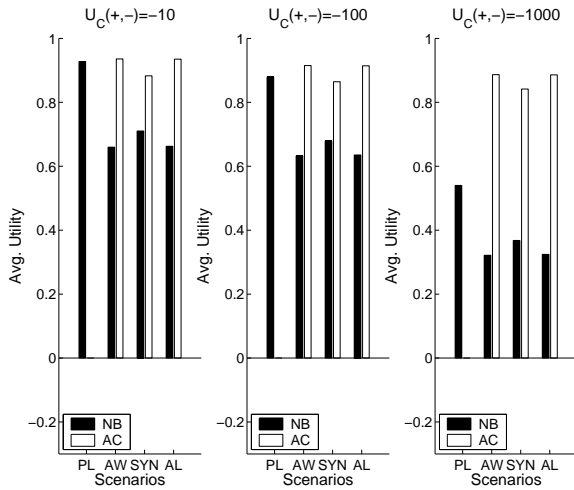


Figure 1: Utility results on the Ling-Spam dataset for different values of $U_C(+, -)$.

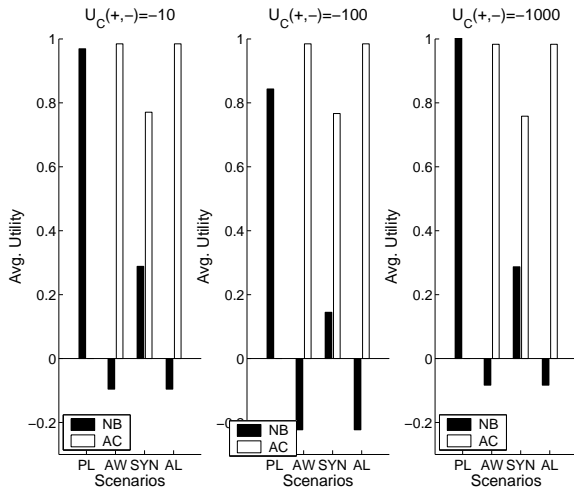


Figure 2: Utility results on the Email-Data set for different values of $U_C(+, -)$.

emails than Ling-Spam. For naive Bayes on Email-Data, the cost of misclassifying spam emails exceeds the utility of the correct predictions, causing the overall utility to be negative. In contrast, CLASSIFIER was able to correctly identify a large percentage of the spam emails in all cases, and its accuracy on non-spam emails was also quite high.

To help in interpreting these results, we report the numbers of false negatives and false positives for the Ling-Spam dataset in Table 3. We observe that as the misclassification penalty for non-spam increases, fewer non-spam emails are classified incorrectly, but naturally more spam emails are misclassified as non-spam. Notice that the adversarial classifier never produces false positives (except for the SYN scenario with $U_C(+, -) = 10$). As a result, its average utility stays approximately constant even when $U_C(+, -)$ changes by two orders of magnitude. An interesting observation is that ADVERSARY’s manipulations can actually help CLASSIFIER to reduce the number of false positives. This is because ADVERSARY is unlikely to send a spam mail unaltered, and as a result many non-spam emails which were previously

$U_C(+, -)$	10		100		1000	
	FN	FP	FN	FP	FN	FP
NB-PLAIN	94	2	124	1	165	1
NB-AW	481	2	481	1	481	1
AC-AW	93	0	123	0	164	0
NB-AL	477	2	477	1	477	1
AC-AL	94	0	124	0	165	0
NB-SYN	408	2	413	1	414	1
AC-SYN	164	1	196	0	229	0

Table 3: False positives and false negatives for naive Bayes and the adversary-aware classifier on the Ling-Spam dataset. The total number of positives in this dataset is 481, and the total number of negatives is 2412.

classified as positive are now classified as negative.

We also compared the running times of our algorithms for the three scenarios, for both the adversary and classifier strategies. For both AW and SYN models, the average running times were less than 5 ms per email. For AL, the average running time of the classifier strategy was less than 5 ms per mail while the running time of the adversary strategy was around 500 ms per email. The adversary running time for AW was small because one can use a simple greedy algorithm to implement the adversary strategy. In the SYN model, the search space is small because there are few synonyms per word. Hence the time taken by both algorithms is small. However, in the AL model, when the LOC of emails was high (> 50), the adversary took longer. On the other hand, the adversarial classifier, after using the pruning strategies, had to consider very few instances and these had small LOC . Hence its running time was quite small. From the experiments we can conclude that in practice we can use the pruning strategies for ADVERSARY and CLASSIFIER to reduce the search space and time without compromising accuracy.

To simulate the effects of non-adversarial concept drift (a reality in spam and many other adversarial domains), we also tried classifying the mails in the Email-data set using NB and AC trained on the Ling-Spam data. As the frequencies of spam and non-spam mails are different in the two datasets, we ran the classifiers without considering the class priors. For both algorithms, the results obtained were only marginally worse than the results obtained by training on the Email-data set itself, demonstrating the robustness of the algorithms.

6.3 Repeated Game

In Sections 4 and 5 we discussed one round of the game that goes on between ADVERSARY and CLASSIFIER. It consists of one ply of ADVERSARY in which it finds the best strategy to fool CLASSIFIER and then one ply of CLASSIFIER to adapt to it. Both parties can continue playing this game. However, CLASSIFIER is no longer using a simple naive Bayes algorithm. In these experiments, we make the simplifying assumption that ADVERSARY continues to model the classifier as Naive Bayes, and uses the techniques that we have developed for Naive Bayes.

At the end of each round, ADVERSARY learns a Naive Bayes classifier based on the outputs of the actual classifier that CLASSIFIER is using in that round. We denote the classifier used by CLASSIFIER in round $i - 1$ by C_{i-1} .

Let NB_{i-1} be the classifier that ADVERSARY learns from it. Then $\mathcal{A}_i(x)$ is defined as the optimal adversary strategy (as in Algorithm 2) to fool NB_{i-1} instead of the original NB learned on \mathcal{S} . The data coming from ADVERSARY in round i is \mathcal{A}_i applied to the original test data to produce \mathcal{T}_i , i.e., $\mathcal{T}_i = \mathcal{A}_i(\mathcal{T})$. CLASSIFIER uses Algorithm 3 based on NB_{i-1} to classify \mathcal{T}_i , i.e., $\mathcal{Y}_i = \mathcal{C}_i(\mathcal{T}_i)$. The key insight is that a new naive Bayes model NB_i can be trained on $(\mathcal{T}_i, \mathcal{Y}_i)$ and that can serve as the “CLASSIFIER” for the next round. We compared the performance of NB_i with that of \mathcal{C}_i and found them to be very similar, justifying our assumption, as ADVERSARY is not reacting to a “crippled” CLASSIFIER but to one which performs almost as well as the “optimal” CLASSIFIER. This procedure can then be repeated for an arbitrary number of rounds.

Figure 3 shows the results of this experiment on the Ling-Spam dataset for the AW scenario. The X-axis is round i of the game, and the Y-axis is the average utility obtained by \mathcal{Y}_i (the i^{th} adversary-aware classifier). The graphs also show the average utility obtained by $NB_{i-1}(\mathcal{T}_i)$, to demonstrate the effect of using an adversary-aware classifier at each round.

In all rounds of the game, CLASSIFIER using the adversary-aware strategy performs significantly better than the plain naive Bayes. As expected, the difference is highest when the penalty for misclassifying non-spam is 1000. Furthermore, in this scenario CLASSIFIER and ADVERSARY never reach an equilibrium, and utility alternates between two values. This is surprising at first glance, but a closer examination elucidates the reason. In the AW scenario, ADVERSARY can only add words. So the only way of tampering with an instance is to add “good” words with very low (negative) LO_C (based on NB_{i-1} in the i^{th} round). Let the top few “good” words be GW_{i-1} . These would have a high frequency of occurrence in spam mails of \mathcal{T}_i . When NB_i is learned on $(\mathcal{T}_i, \mathcal{Y}_i)$ these words no longer have a low LO_C and hence are not in GW_i . Thus, \mathcal{A}_{i+1} ignores these words, making them have a high LO_C in NB_{i+1} and be in GW_{i+1} ! This phenomenon causes LO_C values for a word to oscillate, giving rise to the periodic average utility in Fig. 3.

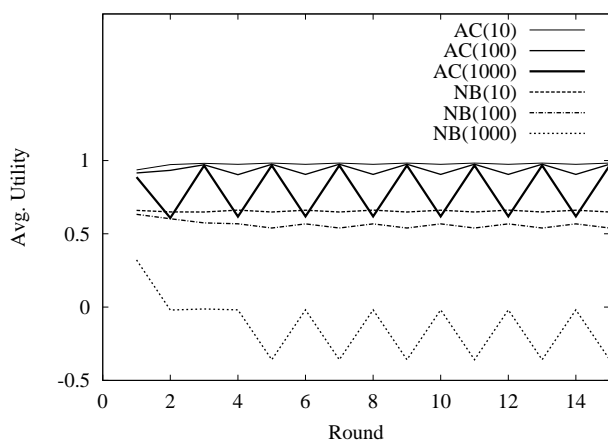


Figure 3: Utility of naive Bayes and the adversarial classifier for a repeated game in the AW scenario and Ling-Spam dataset. The number in parentheses is $U_C(+,-)$.

7. FUTURE WORK

This paper is only the first step in a potentially very rich research direction. The next steps include:

Repeated games. In reality, ADVERSARY and CLASSIFIER never cease to evolve against each other. Thus, we need to find the optimal strategy \mathcal{A}' for ADVERSARY taking into account that an adversarial classifier $\mathcal{C}(\mathcal{A}(x))$ is being used, then find the optimal strategy for CLASSIFIER taking \mathcal{A}' into account, and so on indefinitely. To what extent this can be done analytically is a key open question.

Theory. We would like to answer questions like: What are the most general conditions under which adversarial classification problems have Nash or correlated equilibria? If so, what form do they take, and are there cases where they can be computed efficiently? Under what conditions do repeated games converge to these equilibria? Etc.

Incomplete information. When CLASSIFIER and ADVERSARY do not know each other’s parameters, and ADVERSARY does not know the exact form of the classifier, additional learning needs to occur, and the optimal strategies need to be made robust to imprecise knowledge.

Approximately optimal strategies. When finding the optimal strategy is too computationally expensive, approximate solutions and weaker notions of optimality become necessary. Also, real-world adversaries will often act suboptimally, and it would be good to take this into account.

Generalization to other classifiers. We would like to extend the ideas in this paper to classifiers like decision trees, nearest neighbor, support vector machines, etc.

Interaction with humans. Because adversaries are resourceful and unpredictable, adversarial classifiers will always require regular human intervention. The goal is to make this as easy and productive as possible. For example, extending the framework to allow new features to be added at each round of the game could be a good way to combine human and automatic refinement of the classifier.

Multiple adversaries. Classification games are often played against more than one adversary at a time (e.g., multiple spammers, intruders, fraud perpetrators, terrorist groups, etc.). Handling this case is a natural but non-trivial extension of our framework.

Variants of the problem. Our problem definition does not fit all classification games, but it could be extended appropriately. For example, adversaries may produce innocent as well as malicious instances, they may deliberately seek to make the classifier produce false positives, detection of some malicious instances may deter them from producing more, etc.

Other domains and tasks. We would like to apply adversarial classifiers to computer intrusion detection, fraud detection, face recognition, etc., and to develop adversarial extensions to related data mining tasks (e.g., adversarial ranking for search engines).

8. CONCLUSION

In domains ranging from spam detection to counter-terrorism, classifiers have to contend with adversaries manipulating the data to produce false negatives. This paper formalizes the problem and extends the naive Bayes classifier to optimally detect and reclassify tainted instances, by taking into account the adversary's optimal feature-changing strategy. When applied to spam detection in a variety of scenarios, this approach consistently outperforms the standard naive Bayes, sometimes by a large margin. Research in this direction has the potential to produce KDD systems that are more robust to adversary manipulations and require less human intervention to keep up with them.

ACKNOWLEDGMENTS

We are grateful to Daniel Lowd, Foster Provost and Ted Senator for their insightful comments on a draft of this paper. This research was partly supported by a Sloan Fellowship awarded to the second author.

9. REFERENCES

- [1] P. Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 155–164, San Diego, CA, 1999. ACM Press.
- [2] P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.
- [3] R. B. Doorenbos, O. Etzioni, and D. S. Weld. A scalable comparison-shopping agent for the World-Wide Web. In *Proceedings of the First International Conference on Autonomous Agents*, pages 39–48, Marina del Rey, CA, 1997. ACM Press.
- [4] T. Fawcett. “In vivo” spam filtering: A challenge problem for KDD. *SIGKDD Explorations*, 5(2):140–148, 2003.
- [5] T. Fawcett and F. Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1(3):291–316, 1997.
- [6] D. Fudenberg and D. Levine. *The Theory of Learning in Games*. MIT Press, Cambridge, MA, 1999.
- [7] D. Fudenberg and J. Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.
- [8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, New York, NY, 1979.
- [9] L. Guernsey. Retailers rise in Google rankings as rivals cry foul. *New York Times*, November 20, 2003.
- [10] G. Hulthen, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106, San Francisco, CA, 2001. ACM Press.
- [11] D. Jensen, M. Rattigan, and H. Blau. Information awareness: A prospective technical assessment. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 378–387, Washington, DC, 2003. ACM Press.
- [12] M. Kearns. Computational game theory. Tutorial, Department of Computer and Information Sciences, University of Pennsylvania, Philadelphia, PA, 2002. <http://www.cis.upenn.edu/~mkearns/nips02tutorial/>.
- [13] R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.
- [14] B. Krebs. Online piracy spurs high-tech arms race. *Washington Post*, June 26, 2003.
- [15] M. L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163, New Brunswick, NJ, 1994. Morgan Kaufmann.
- [16] B. Lloyd. Been gazumped by Google? Trying to make sense of the “Florida” update. *Search Engine Guide*, November 25, 2003.
- [17] M. V. Mahoney and P. K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 376–385, Edmonton, Canada, 2002. ACM Press.
- [18] A. McCallum and K. Nigam. A comparison of event models for Naive Bayes text classification. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, Madison, WI, 1998. AAAI Press.
- [19] F. Nielsen. Email data, 2003. <http://www.imm.dtu.dk/~rem/datasets/>.
- [20] F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42:203–231, 2001.
- [21] J. Rennie. Ifile spam classifier, 2003. <http://www.nongnu.org/ifile/>.
- [22] P. Robertson and J. M. Brady. Adaptive image analysis for aerial surveillance. *IEEE Intelligent Systems*, 14(3):30–36, 1999.
- [23] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk e-mail. In *Proceedings of the AAAI-98 Workshop on Learning for Text Categorization*, Madison, WI, 1998. AAAI Press.
- [24] G. Sakkis, I. Androutopoulos, G. Paliouras, V. Karkaletsis, C.D. Spyropoulos, and P. Stamatopoulos. A memory-based approach to anti-spam filtering for mailing lists. In *Information Retrieval*, volume 6, pages 49–73. Kluwer, 2003.
- [25] T. Senator. Ongoing management and application of discovered knowledge in a large regulatory organization: A case study of the use and impact of NASD regulation's advanced detection system (ADS). In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 44–53, Boston, MA, 2000. ACM Press.
- [26] J. M. Smith. *Evolution and the Theory of Games*. Cambridge University Press, Cambridge, UK, 1982.
- [27] P. Turney. Cost-sensitive learning bibliography. Online bibliography, NRC Institute for Information Technology, Ottawa, Canada, 1998. <http://members.rogers.com/peter.turney/bibliographies/cost-sensitive.html>.
- [28] WordNet 2.0: A lexical database for the English language, 2003. <http://www.cogsci.princeton.edu/~wn/>.