Scalable Optimization of Randomized Operational Decisions in Adversarial Classification Settings

Bo Li and Yevgeniy Vorobeychik
Electrical Engineering and Computer Science
Vanderbilt University
Nashville, TN
{bo.li.2,yevgeniy.vorobeychik}@vanderbilt.edu

Abstract

When learning, such as classification, is used in adversarial settings, such as intrusion detection, intelligent adversaries will attempt to evade the resulting policies. The literature on adversarial machine learning aims to develop learning algorithms which are robust to such adversarial evasion, but exhibits two significant limitations: a) failure to account for operational constraints and b) a restriction that decisions are deterministic. To overcome these limitations, we introduce a conceptual separation between learning, used to infer attacker *preferences*, and operational decisions, which account for adversarial evasion, enforce operational constraints, and naturally admit randomization. Our approach gives rise to an intractably large linear program. To overcome scalability limitations, we introduce a novel method for estimating a compact parity basis representation for the operational decision function. Additionally, we develop an iterative constraint generation approach which embeds adversary's best response calculation, to arrive at a scalable algorithm for computing near-optimal randomized operational decisions. Extensive experiments demonstrate the efficacy of our approach.

1 Introduction

Success of machine learning across a variety of domains has naturally led to its adoption as a tool in security settings, including intrusion detection, biometric identity recognition, and spam filtering. Unlike traditional uses of machine learning, however, these domains involve an *adversary*, who is likely to adapt to the use of such techniques, potentially reducing their effectiveness. Of particular interest in many such application domains is *adversarial classification*, or the task of determining whether a given input (email, system access, user behavior) is benign or "normal", or malicious (a phishing email or a system compromise). In such settings, we start with a training data set of labeled instances $\{(x_1, y_1), \ldots, (x_m, y_m)\}$, where x_i are feature vectors (e.g., whether or not specific spam/phish indicators are present in an email) and y_i are labels, which we can code as 0 corresponding to benign and 1 to malicious instances. This data set is used to train a classifier, h, that would presumably predict whether an arbitrary unseen instance x is malicious. The phenomenon of *adversarial evasion* puts a damper on this seemingly clean solution: if an adversary wishes, say, to send an email with features x, but $h(x)$ classifies it as malicious, an intelligent attacker would attempt to choose another email, corresponding to x', which would be classified as benign, and achieve the same, or nearly the same, ends.

The literature on *adversarial machine learning* tackles the problem of adversarial evasion in two ways: first, by trying to understand its feasibility and effectiveness [1, 2, 3, 4, 5], and second, by attempting to design machine learning algorithms which account for, and are robust to, evasion [1, 3, 6, 7, 8, 9, 10].

Past literature on algorithm design for adversarial classification suffers from two important limitations. First, previous approaches make no attempt to account for resource constraints involved in *operationalizing* the algorithms: in particular, it is the false positives, rather than false negatives, which are critical to adoption of intrusion detection systems, in large part because overabundance of "alerts" makes

Appearing in Proceedings of the 18^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2015, San Diego, CA, USA. JMLR: W&CP volume 38.

such a system operationally unusable [11]. Second, there is, to date, no principled way of embedding randomization into adversarial classification, even though stochasticity in defense is often highly effective in security [12, 13, 14]. Indeed, the use of randomization in adversarial classification has previously been suggested [15], but the proposed approach is ad hoc, simply adding "random noise" to the classifier output.

We address both of these limitations by rethinking the conceptual model of adversarial classification. Specifically, we separate the task of *learning*, which uses training data to *predict attack preferences*, and the task of *operational decisions*, which uses the resulting predictor, together with an evasion model, in computing optimal *randomized* operational policy that *explicitly abides by operational constraints*. The intuition for this separation is that the training data can be interpreted as *revealed preferences* of the attackers, in the sense that the attacks captured by it can be viewed as "ideal" attack vectors at that point in time. As an indirect consequence, our model enables one to use off-the-shelf machine learning packages, allowing progress in machine learning and adversarial decision making to be decoupled. On the technical side, we present a natural generalization of a commonly used evasion model (see, e.g., [4]) to randomized classification settings. We show that computing an optimal evasion is NP-Hard, but also exhibit an optimal branch-and-bound search method and two polynomial-time approximation algorithms, one with worst-case performance guarantees, and both shown to be "near-optimal" in experiments. On the operational side, we introduce a linear programming (LP) formulation for computing optimal randomized classification. While the baseline LP involves an exponential number of variables and constraints, we propose a collection of techniques which make use of a Fourier representation of Boolean functions [16], as well as constraint generation, to arrive at scalable approximation.

In all, we make the following contributions: 1) a general framework for optimizing operational decisions based on machine learning predictions; 2) a linear programming formulation to compute optimal randomized operational decisions under budget constraints; 3) an approach for scalable parity-basis approximation of operational decision function; 4) a model of attacker evasion and methods approaches for approximating optimal evasion; and 5) an extensive evaluation of our approach, which we show to significantly outperform the state of the art.

2 Model

We consider the adversarial binary classification problem over an input space $\mathcal{X}$, where each input feature vector $x \in \mathcal{X}$ can be categorized (labeled) as benign or malicious. The defender $\mathcal{D}$, collects a data set of labeled instances, $\mathcal{I} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, which we assume to accurately represent the current distribution of input instances and corresponding categories. $\mathcal{D}$ then applies an algorithm of choice, such as Naive Bayes, to obtain a probabilistic classifier which assigns to an arbitrary input x vector a probability $p(x)$ that it is generated by a malicious actor *assuming such an actor does not change their behavior*. In traditional applications, one would then use a threshold, θ, and classify an instance x as malicious if $p(x) \geq \theta$, and benign otherwise. This decision (and the choice of the threshold) are often motivated by overall tolerance for false positives, as well as operational considerations, for example, to ensure that the number of alerts does not exceed what can reasonably be inspected by security professionals. To consider operational decisions in general, as well as allow for randomization, we introduce a function $q(x, p(\cdot)) \in [0, 1]$ which prescribes a possibly randomized operational decision (e.g., the probability of filtering an email or manually investigating an observed network access pattern) for an instance x given a prediction $p(x)$. To simplify notation, we simply use $q(x)$ where $p(\cdot)$ is clear from context. Throughout, we assume that features are binary, a common case in adversarial classification settings (e.g., features could correspond to specific words or phrases being present in email, or specific sequences of system calls executed).

We model adversarial classification as a Stackelberg game between a defender and a population of attackers. In this game, the defender $\mathcal{D}$ moves first, choosing $q(\cdot)$. Next, the attackers learn $q(\cdot)$ (for example, through probing), and each attacker subsequently chooses an input vector x (e.g., a phishing email) to maximize their expected return (a combination of bypassing defensive countermeasures and achieving a desired outcome). Our assumption that the operational policy $q(\cdot)$ is known to attackers reflects threats that have significant time and/or resources to probe and respond to defensive measures, a feature characteristic of advanced cyber criminals [17].

2.1 Attacker Model

We interpret the data set $\mathcal{I}$ and the resulting predictions $p(x)$, as representing *revealed preferences* of a sample of attackers, that is, their preference for input vectors x. Our rationale is that if an attacker preferred some other input x', this attacker would have

chosen x' instead of x in $\mathcal{I}$. Consequently, $p(x)$ can be interpreted as an "ideal" attack, if only it were to succeed in bypassing defensive measures. If $q(x)$ is sufficiently close to 1, x is likely to fail, and the attacker will have an incentive to *evade* by choosing another instance x'. When decisions $q(x)$ are deterministic, a common approach in related literature is to assume that the attacker will find x' which is closest to x (in some distance metric, such as l_1 norm) of all alternatives classified as benign [18, 4, 7, 8, 9]. We now offer a natural generalization of this model to account for randomized $q(x)$. Specifically, if the attacker with a preference for x chooses an alternative attack vector x', we model his utility from successfully bypassing defenses as $V(x)Q(x,x')$, where $Q(x,x') = e^{-\delta||x-x'||}$, with $||\cdot||$ a norm (we use Hamming distance), $V(x)$ the value of the attack, and δ the importance of being close to the preferred x. The full utility function of an attacker with preference x for choosing another input x' when the defense strategy is q is then

$$\mu(x,x';q) = V(x)Q(x,x')(1-q(x')), \qquad (1)$$

since $1-q(\cdot)$ is the probability that the attacker successfully bypasses the defensive action.

While the above attacker model admits considerable generality, we assume that attackers fall into two classes: adaptive, as described above, and static, corresponding to the limiting case of $\delta \to \infty$. Let $v_t(x;q)$ be the value function of an attacker with type t and preference for x, when the defender chooses a policy q. $v_t(x;q)$ represents the maximum utility that the attacker with type t can achieve given q. For a static attacker, the value function is $v_S(x;q) = V(x)(1-q(x))$, that is, a static attacker always uses his preferred input x, and receives his corresponding value for it whenever the defender does not take action upon observing x. For an adaptive attacker, the value function is $v_A(x;q) = \max_{x'} \mu(x,x';q)$, that is, the maximum utility that the attacker obtains from using an *arbitrary* input x'. Finally, let P_A be the probability that an arbitrary malicious input was generated by an adaptive adversary; the probability that the adversary was static is then $P_S = 1 - P_A$.

2.2 Defender Model

A natural goal for the defender is to maximize expected value of benign traffic that is classified as benign, less the expected losses due to attacks that successfully bypass the operator. To formalize, we assume that the defender gains a positive value $G(x)$ from a benign input x only if it is not inspected. In the case of email traffic, this is certainly sensible if our action is to filter a suspected email. More generally, inspection can be a lengthy process, in which case we can interpret $G(x)$ as the value of time lost if x is, in fact, benign, but is carefully screened before it can have its beneficial impact. We define the defender's utility function $U_{\mathcal{D}}(q,p)$ as follows:

$$U_{\mathcal{D}}(q,p) = \mathbb{E}_x\left[(1-q(x))G(x)(1-p(x)) - p(x)(P_S v_S(x;q) + P_A v_A(x;q))\right].$$

To interpret the defender's utility function, let us first rewrite it for a special case when $V(x) = G(x) = 1$ and $P_S = 1$, reducing the utility function to $\mathbb{E}_x[(1-q(x))(1-p(x)) - p(x)(1-q(x))]$. Since $p(x)$ is constant, this is equivalent to minimizing

$$\mathbb{E}_x[q(x)(1-p(x)) + p(x)(1-q(x))],$$

which is just the *expected misclassification error*.

The final aspect of our model is a resource constraint on the defender. Sommer and Paxson [11] identify the cost of false positives and the gap between the output of machine learning algorithms and its use in operational decisions as two of the crucial gaps that prevent widespread use of machine learning in network intrusion detection. In our framework, $G(x)$ quantifies the loss of value due to false positives. We handle the hard constraint on defensive resources by introducing a budget constraint that our solution inspects at most a fraction c of events, on average.

2.3 Model Analysis

A natural sanity check that our formulation is reasonable is that the solution corresponds to intuition when there is no budget constraint or adaptive adversary. We now show that in this case, the policy $q(x)$ which uses a simple threshold on $p(x)$ (as commonly done) is, in fact optimal.

Proposition 2.1. *Suppose that $P_A = 0$ and $c = 1$ (i.e., no budget constraint). Then the optimal policy is*

$$q(x) = \begin{cases} 1 & \text{if } p(x) \geq \frac{G(x)}{G(x)+V(x)} \\ 0 & o.w. \end{cases}$$

Due to space restrictions, we leave detailed proofs in this paper to the full version. While traditional approaches threshold an odds ratio (or log-odds) rather than the probability $p(x)$, the two are, in fact equivalent. To see this, let us consider the generalized (cost-sensitive) threshold on odds ratio used by the Dalvi et al. [18] model. In their notation, $U_{\mathcal{C}}(+,+)$, $U_{\mathcal{C}}(+,-)$, $U_{\mathcal{C}}(-,+)$, and $U_{\mathcal{C}}(-,-)$ denote the utility of the defender (classifier) when he correctly identifies a malicious input, incorrectly identifies a benign input, incorrectly identifies a malicious input, and correctly identifies a benign input, respectively. In our setting, we have $U_{\mathcal{C}}(+,+) = 0$ (i.e., no loss), $U_{\mathcal{C}}(+,-) = 0$ (and capture the costs of false positives as operational constraints instead), $U_{\mathcal{C}}(-,+) = -V(x)$, and $U_{\mathcal{C}}(-,-) = G(x)$ (note that we augment the utility functions to depend on input vector x). The odds-ratio test used by Dalvi et al. therefore checks

$$\frac{p(x)}{1-p(x)} \geq \frac{U_{\mathcal{C}}(-,-) - U_{\mathcal{C}}(+,-)}{U_{\mathcal{C}}(+,+) - U_{\mathcal{C}}(-,+)} = \frac{G(x)}{V(x)}. \qquad (2)$$

and it is easy to verify that inequality 2 is equivalent to the threshold test in Proposition A.1.

Consider now a more general setting where $P_A = 0$, but now with a budget constraint. In this context, we now show that the optimal policy is to first set $q(x) = 0$ for all x with $p(x)$ below the threshold described in Proposition A.1, then rank the remainder in descending order of $p(x)$, and assign $q(x) = 1$ in this order until the budget is exhausted.

Proposition 2.2. *Suppose that $P_A = 0$. Then the optimal policy is to let $q(x) = 0$ for all x with*

$$p(x) < \frac{G(x)}{G(x) + V(x)}.$$

Rank the remaining x in descending order of $p(x)$ and set $q(x) = 1$ until the budget is exhausted, leaving the remaining budget to the next instance x, and setting $q(x) = 0$ for the rest.

In a nutshell, Proposition A.2 suggests that whenever the budget constraint binds, we should simply inspect the highest priority items. Therefore, randomization becomes important only when there is an adversary actively responding to our inspection efforts.[1]

3 Optimal Randomized Operational Use of Classification

Given the Stackelberg game model of strategic interactions between a defender armed with a classifier, and an attacker attempting to evade it we now develop an algorithmic approach for solving it. We begin by using a sample average approximation of the defender's utility function $U_{\mathcal{D}}$ (e.g., using instances in the training data), denoting it $\hat{U}_{\mathcal{D}}$. Using $\hat{U}_{\mathcal{D}}$ as the objective, we can maximize it using the following linear program (LP):

$$\begin{aligned}
\max_{q} \quad & \hat{U}_{\mathcal{D}}(q, p) && && \text{(3a)}\\
\text{s.t.}: \quad & 0 \le q(x) \le 1 && \forall\, x \in \mathcal{X} && \text{(3b)}\\
& v_A(x; q) \ge \mu(x, x'; q) && \forall\, x, x' \in \mathcal{X} && \text{(3c)}\\
& v_S(x; q) = V(x)(1 - q(x)) && \forall\, x \in \mathcal{X} && \text{(3d)}\\
& \mathbb{E}_x[q(x)] \le c, && && \text{(3e)}
\end{aligned}$$

where constraint 3c computes the attacker's best response (optimal evasion of q).

3.1 Scaling Up

The linear program 3 is not a practical solution approach for two reasons: a) $q(x)$ must be defined over the entire feature space $\mathcal{X}$, and b) the set of constraints is quadratic in $|\mathcal{X}|$. Since with n features $|\mathcal{X}| = 2^n$, this LP is a non-starter.

Our first step towards addressing the scalability issue is to represent $q(x)$ using a set of normalised basis functions, $\{\phi_j(x)\}$, where $q(x) = \sum_j \alpha_j \phi_j(x)$. This allows us to focus on optimizing α_j, a potentially tractable proposition if the set of basis functions is small. With this representation, the LP now takes the following form (to simplify exposition below, we assume that $P_A = 1$; generalization is direct):

$$\min_{\alpha \ge 0} \sum_j \alpha_j \mathbb{E}[G(x)\phi_j(x)(1 - p(x))] + \mathbb{E}[V(x)p(x)Q(x, \alpha)] \tag{4a}$$

$$\text{s.t}: Q(x, \alpha) \ge e^{-\delta||x - x'||}(1 - \sum_i \alpha_j \phi_j(x')) \quad \forall x, x' \in \mathcal{X} \tag{4b}$$

$$\sum_j \alpha_j \mathbb{E}[\phi_j(x)] \le c \tag{4c}$$

$$\sum_j \alpha_j \le 1. \tag{4d}$$

While we can reduce the number of variables in the optimization problem using a basis representation ϕ, we still retain the intractably large set of inequalities which compute the attacker's best response. To address this issue, suppose that we have an oracle $\mathcal{O}(x; q)$ which can efficiently compute a best response x' to a strategy q for an attacker with an ideal attack x. Armed with this oracle, we propose a constraint generation aproach, termed Adaptive Adversary based Scalable classification (AAS), to iteratively compute an (approximately) optimal operational decision function q (Algorithm 1 below).

Algorithm 1 AAS(X)

```
φ =ConstructBasis()
X̄ ← X
q ← MASTER(X̄)
while true do
  for x ∈ X_bad do
    x' = O(x; q)
    X̄ ← X̄ ∪ x'
  end for
  if All x' ∈ X̄ then
      // If no new x' generated
    return q
  end if
  q ← MASTER(X̄)
end while
```

The input to the AAS algorithm (Algorithm 1) is the feature matrix X in the training data, with X_{bad} denoting this feature matrix restricted to "bad" (malicious) instances. At the core of Algorithm 1 is

[1] Of course, we do not suggest that the policy in Proposition A.2 is easy to implement. Its purpose is entirely to understand the nature of our approach when applied to non-adversarial settings.

the MASTER linear program which computes an attacker's (approximate) best response using the modified LP 4, but using only a small subset of all feature vectors as alternative attacks, which we denote by $\bar{\mathcal{X}}$. The algorithm begins with $\bar{\mathcal{X}}$ initialized to only include feature vectors in the training data X. The first step is to compute an optimal solution, q, with adversarial evasion restricted to X. Then, iteratively, we compute each attacker's best response x' to the current solution, q, adding it to $\bar{\mathcal{X}}$ (the preferences of each attacker are parameterized by the attacks they executed in the original training data), rerun the MASTER linear program to recompute q, and repeat. The process is terminated when we cannot generate any new constraints (i.e., the available constraints already include best responses for all attackers). The following result is a direct consequence of a) finiteness of feature space, and b) the fact that at termination the attacker is playing an actual best response to the computed strategy q.

Theorem 3.1. *The AAS algorithm computes an optimal solution q given a fixed basis ϕ in finite time.*

The approach described so far in principle addresses the scalability issues, but leaves two key questions unanswered: 1) how do we construct the basis ϕ, a problem which is of critical importance to good quality approximation (the ConstructBasis() function in Algorithm 1), and 2) how do we compute the attacker's best response to q, represented above by an oracle $\mathcal{O}(x, q)$. We tackle these in turn.

3.1.1 Basis Construction

Our basis representation relies on harmonic (Fourier) analysis of Boolean functions [16, 19]. In particular, it is known that every Boolean function $f : \{0,1\}^n \to \mathbb{R}$ can be uniquely represented as $f(x) = \sum_{S \in B_S} \hat{f}_S \chi_S(x)$, where $\chi_S(x) = (-1)^{S^T x}$ is a parity function on a given basis $S \in \{0,1\}^n$, B_S is the set containing all the bais S, and the corresponding Fourier coefficients can be computed as $\hat{f}_S = \mathbb{E}_x[f(x)\chi_S(x)]$ [20, 19]. Our goal will be to approximate $q(x)$ using a Fourier basis. Our core task is to compute a set of basis functions to be subsequently used in optimizing $q(x)$. The first step is to uniformly randomly select K feature vectors $\vec{x_k}$. Then use a traditional learning algorithm, say Naive Bayes, to obtain the $p(x)$ vector and solve the linear program 3 to compute $q(x)$ restricted to these. We can now use the same set of feature vectors to approximate a Fourier coefficient of this $q(x)$ for an arbitrary basis S as $t = \frac{1}{m} \sum_{i=1}^{m} q(x^i)\chi_S(x^i)$. We can use this expression to compute a basis set S with the largest Fourier coefficient using the following integer linear program:

$$\max_S \quad \frac{1}{K} \sum_{k=1}^{K} q(x^k) r_S^k \tag{5a}$$

$$s.t.: \quad S^T x^k = 2y^k + h^k \tag{5b}$$

$$r_S^k = 1 - 2h^k \tag{5c}$$

$$y^k \in Z, h^k \in \{0,1\}, S \in \{0,1\}^n \tag{5d}$$

Our basis generation algorithm solves this program iteratively, each time adding a constraint that rules out a previously generated basis, until the optimal solution is zero. Each basis is optimized within limited time and then we collect the set of optimized basis functions B_S that are corresponding to the largest Fourier coefficients. To consider the largest negative Fourier coefficients, we simply change Program 5 to be minimization. We found, however, that negative Fourier coefficients were rare in our problem instances.

3.1.2 Computing Adversary's Best Response

The constraint generation algorithm AAS described above presumes the existance of an oracle $\mathcal{O}(x; q)$ which computes (or approximates) an optimal evasion of q (we call this a *best response* to q) for an attacker that would prefer to use a feature vector x. We now address this problem in detail. Note that since $V(x)$ is fixed in the attacker's evasion problem (because x is fixed), it can be ignored.

We begin by addressing the computational complexity of computing an optimal evasion. Informally, given an arbitrary set of bases ϕ and the adversary's preference feature vector x, the attacker wishes to modify as few features as possible to obtain a binary vector x' that minimizes $q(x')$. To make the analysis cleaner, we compute the bases ϕ_j as mapping to $\{0,1\}$, where $\phi_j(x) = \frac{1}{2}(\chi_{S_j}(x) + 1)$. A formal decision problem faced by the attacker is whether there exist a feature vector x' satisfying the following constraints:

$$\sum_j \alpha_j \phi_j(x') \leq \lambda \tag{6a}$$

$$\|x - x'\| \leq k, \tag{6b}$$

where λ and k are fixed given thresholds. This problem, which we call *EVASION*, can be shown to be computationally hard by reducing it from 3DM (details are in the extended version of the paper).

Theorem 3.2. *EVASION is NP-complete.*

Since adversarial evasion is NP-Hard, it is natural to develop an approximation algorithm to solve the following derived optimization problem:

$$\min_{x'} \sum_j \alpha_j \phi_j(x') \tag{7a}$$

$$\text{s.t.:} \quad ||x - x'|| \leq k \tag{7b}$$

Define $\Delta(x') = q(x) - q(x') = \sum_j \alpha_j(\phi_j(x) - \phi_j(x'))$, so that our objective can equivalently be stated as maximizing $\Delta(x')$ so that at most k features in x are

modified. Let Δ^* be the optimal solution to this problem. We present Algorithm 3 to compute x' which yields a provably near-optimal solution.

Algorithm 2 ApproxEvasion(F, q, k, ϵ)

$n \leftarrow |F|$
$D_0 \leftarrow \{(\emptyset, 0)\}$ // tuple $d_i = (feaSet, value) \in D$
$G = GenFeaGroup(F, S)$
$l \leftarrow 0$
for $i \leftarrow 1$ to $|G|$ **do**
  for $j \leftarrow 1$ to $|g_i|$ **do**
    $l \leftarrow l + 1$ // merge two tuple-lists by $d_i.value$
    $D_l \leftarrow MergeTuple(D_{l-1}, AddFea(D_{l-1}, f_{ij}, k))$
  end for
  $D_l \leftarrow Trim(D_l, \epsilon/2n)$
  remove elements from D_l that $d_l.value > q(x)$
end for
let d^* correspond to the maximum $d.value$ in D_n
return d^*

Theorem 3.3. *Suppose that the number of inputs in any basis is bounded by a constant c. Then* ApproxEvasion *(Algorithm 3) computes a solution x' to problem 7 which achieves $\hat{\Delta} \geq \frac{\Delta^*}{1+\epsilon}$, where $\hat{\Delta} = \Delta(x')$ in time poly$(n, \frac{1}{\epsilon}, 2^c)$.*

While the complete algorithm and proof are in the extended version, below we offer some intuition. The key issue in Algorithm 3 is that the length of D_i can be 2^i, making the merge algorithm take exponential time. To fix this, we employ a $Trim$ function to shorten the list length. The idea is that if some combinations of features have similar effect on $q(x)$, only one combination is considered. This means that with a trimming parameter δ, for any element d_i removed from D_i, there is an element d_j that approximates d_i, that is, $\frac{Retrieve(d_i)}{1+\delta} \leq Retrieve(d_j) \leq Retrieve(d_i)$. Notice that the $Trim$ action can only be done for features that have no common bases to avoid missing qualified feature combination. Therefore, *GenFeaGroup* algorithm is applied to group the features that need to be added as a whole before $Trim$. *AddFea* algorithm then helps to form different feature combinations and guarantees that at most k features are considered.

In addition to the approximation algorithm above, we consider two others: an optimal *branch-and-bound* search with worst-case exponential running time, and a greedy heuristic (*Greedy*). In the *branch-and-bound* scheme, we search in the space of feature changes to x. At any node with height l, we have thereby changed l features in x, and the utility of the attacker in this subtree is therefore bounded above by $e^{-\delta l}$ (since $1 - q(x') \leq 1$). This bound is used in pruning much of the search tree once a good solution using relatively few modifications is found. In the greedy heuristic, we start with x and iteratively flip features one at a time, flipping a feature that yields the greatest decrease in $q(x')$ each time.

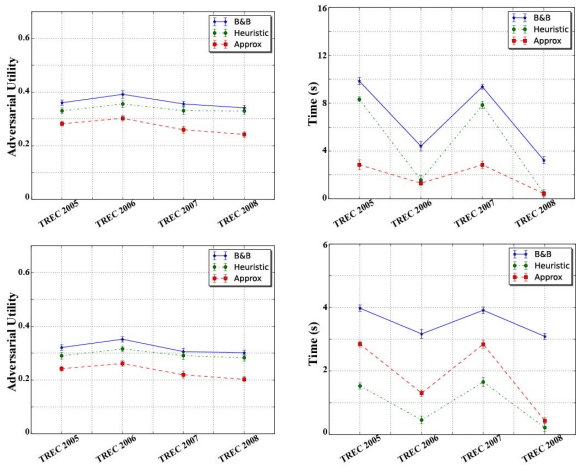


Figure 1: Comparision of expected adversary utility (left) and algorithm runtime (right), for the three adversrial evasion algorithms. Top: $\delta = 1$, $\epsilon = 0.01$. Bottom: $\delta = 3$, $\epsilon = 0.01$

We used TREC spam corpora to experimentally compare the three approaches to computing adversarial evasion: the ApproxEvasion algorithm,[2] branch-and-bound, and greedy heuristic. The results, shown in Figure 1, suggest (somewhat surprisingly) that the simple greedy heuristic offers a good balance between running time and quality: it is faster, usually quite significantly, than branch-and-bound, and loses less in solution quality than the approximation algorithm. Consequently, our implementation of AAS features the evasion oracle $\mathcal{O}$ which runs the greedy heuristic.

4 Experiments

To evaluate the efficacy of the proposed AAS algorithm for approximating optimal randomized operational decisions in adversarial classification settings, we compare the optimized utility of defender with the state of the art. The results below use 100 features, with additional results (using 500 features over the same domain) presented in the extended version of the paper.

In the experiments, we use the TREC spam corpora from 2005-2008.[3] First, we evaluate the performance

[2]Whie ApproxEvasion cannot be used directly, it can be adapted using a linear search in the space of thresholds k along with the same bound as used in branch-and-bound to truncate the search.

[3]Our choice of TREC corpora for this evaluation is due primarily to its longitudinal nature, which is key for a subset of our experiments.

of AAS as a spam filtering task to compare the classification accuracy with the state of the art alternatives [21, 8]. In this first set of experiments, which are used to test robustness to *naturally observed* spam evolution, we apply a fold of TREC 2005 data as training and evaluate and test on the test fold for the TREC 2005 and 2006-2008 corpora (in other words, we train on "current" data, and observe performance on "future" data). We compare our approach against using a static classifier it is based upon, where the pair of the form $\{C, AAS(C)\}$, consists of a static classifier C which is used to learn $p(x)$ for our model, and $AAS(C)$ corresponding to our AAS algorithm that utilizes C. Here we use the normalized utility as $U_D = 1 - \frac{w|X^+|+|X^-|}{w|X_{TN}|+|X_{TP}|}$, where $|X_{TN}|$ is the number of true negatives, while $|X_{TP}|$ the number of true positives. $|X^-| = \sum_x y(x)(1-q(x))$ represents the expected number of false negatives, while $|X^+| = \sum_x (1-y(x))q(x)$ the expected number of false positives; $w = \frac{G}{V}$.

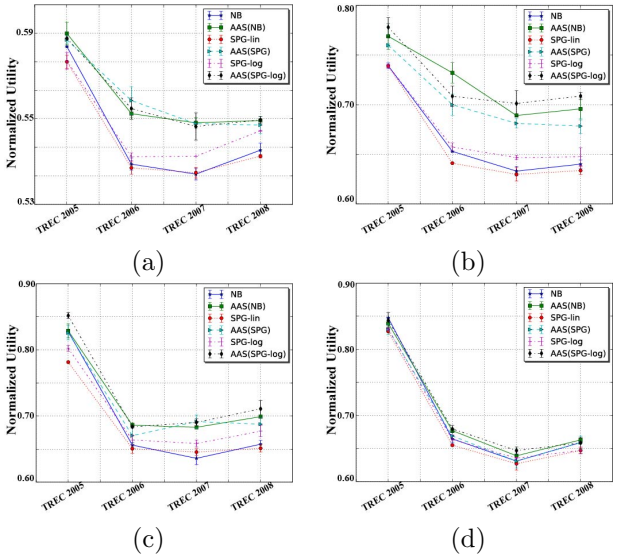


Figure 2: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 1$, $V(x) = G(x) = 1$, $P_A = 1$ (a) c=0.1; (b) c=0.3; (c) c=0.5; (d) c=0.9.

Figure 2 shows that when the budget constraint is tight, our approach significantly outperforms the baselines. From Figure 3 it can also be observed how the cost of adversary matters. When we fix $G(x) = 1$ and vary $V(x) = V$ (keeping it constant for all x), our approach still consistently outperforms alternatives.

In the next set of experiments, we simulated a counterfactual of sophisticated evasion attacks, deployed according to our model, drawing the same comparisons as above, but now treating each year in the TREC data as distinct (in other words, we consider each year as "current", and then simulate an evasion attack independently for each year). From Figure 4 and 5 we can see that our proposed method significantly outperforms the alternatives on different datasets across both alternative budget constraints and value models.

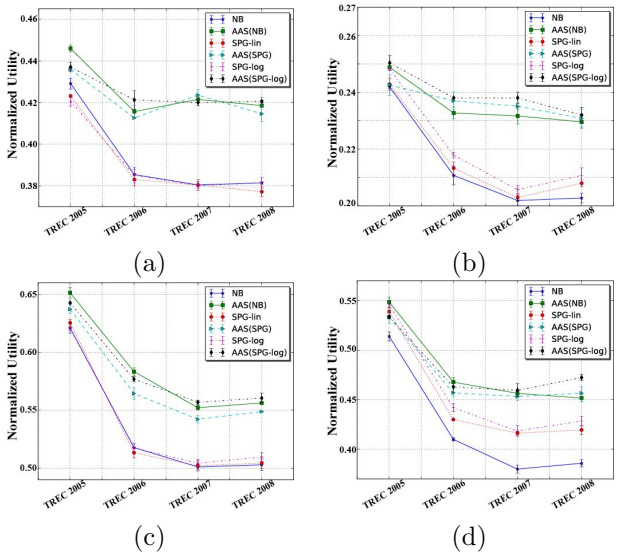


Figure 3: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 1$, $G(x) = 1$, $P_A = 1$ (a) $V(x) = 2$, c=0.1; (b) $V(x) = 10$, c=0.1; (c) $V(x) = 2$, c=0.3; (d) $V(x) = 10$, c=0.3.

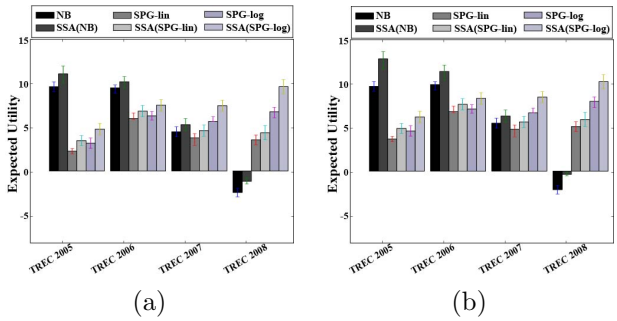
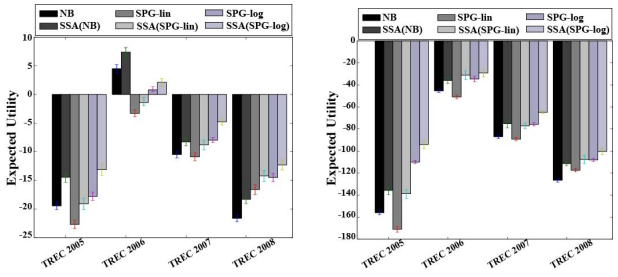


Figure 4: Comparison of the expected utility assuming $P_A = 1$, $V(x) = G(x) = 1$; (a) $c = 0.1$; (b) $c = 0.3$.

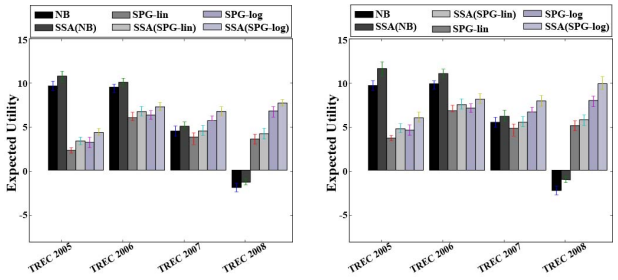
It is, of course, not surprising that our proposed approach outperforms alternative methods *in terms of the objective it tries to optimize*. A natural question, however, is whether this approach is robust to errors which would be inevitable in its practical deployment. To evaluate the robustness of our algorithm, we intro-



(a) (b)

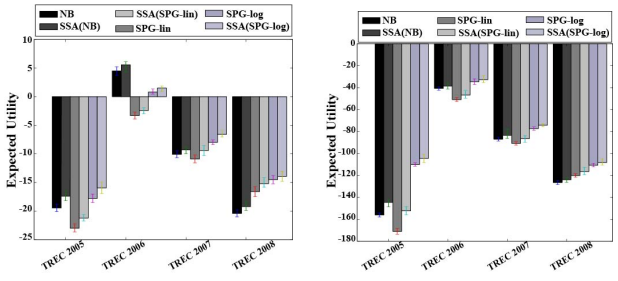
Figure 5: Comparison of the expected utility assuming $P_A = 1$; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.

duce errors into our attacker model. First, we introduce an error $\varphi = 0.2$ into the attacker model, so that $\hat{\delta} = \delta + \varphi$, where $\hat{\delta}$ is the "observed" and δ the actual model parameter. Figure 6 and 7 show that our approach still outperforms the state of the art alternatives even when harmed by very substantial inaccuracy in the model parameter estimates.



(a) (b)

Figure 6: Comparison of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $c = 0.1$; (b) $c = 0.3$.



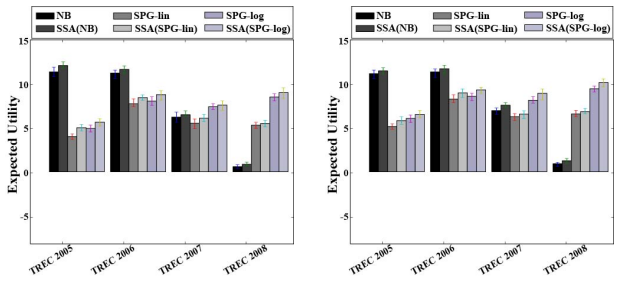
(a) (b)

Figure 7: Comparison of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.

Next, we consider robustness to a *qualitative* rather than *quantitative* error in adversarial model. To simulate this, we solve our model as before, but evaluate the solutions $q(x)$ by assuming an adversary's utility model actually decays polynomially as

$$Q_{poly}(x, x') = \frac{1}{1 + \delta \|x - x'\|}.$$

The results, shown in Figure 8, demonstrate that our model is robust even when the assumption about the adversary utility model is fundamentally incorrect.



(a) (b)

Figure 8: Comparison of the expected utility assuming $P_A = 1$, introducting adversarial model error; (a) $c = 0.1$; (b) $c = 0.3$.

5 Conclusions

We presented a general approach for computing optimal randomized decisions in adversarial classification settings. We solve the resulting intractably large problem by applying a finite set of basis functions and using constraint generation which leverages high-quality approximation of optimal adversarial classifier evasion. The proposed method outperforms than the state of the art alternatives on several metrics, is robust to errors (including qualitative mistakes in modeling assumptions) and its advantages are more apparent when operational decisions are costly. Moreover, by conceptually separating the problem of prediction (of adversary's preferences) and optimal operational decisions, the approach can both make use of off-the-shelf machine learning techniques, and naturally embed randomization. While the use of machine learning in adversarial settings, such as network intrusion detection, is still quite limited, our approach may pave the way for bridging the gap between algorithmic advances and operational deployment of such systems.

Acknowledgements

This work was supported in part by the National Science Foundation under Award CNS-1238959, by the Air Force Research Laboratory under Award FA8750-14-2-0180, and by Sandia National Laboratories.

References

[1] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM, 2004.

[2] Ling Huang, Anthony D. Joseph, Blaine Nelson, Benjamin I.P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, AISec '11, pages 43–58, New York, NY, USA, 2011. ACM.

[3] Bo Li and Yevgeniy Vorobeychik. Feature cross-substitution in adversarial classification. In *Neural Information Processing Systems*, 2014. to appear.

[4] Daniel Lowd and Christopher Meek. Adversarial learning. In *ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 641–647, 2005.

[5] B. Nelson, B. Rubinstein, L. Huang, A. Joseph, S. Lee, S. Rao, and J. D. Tygar. Query strategies for evading convex-inducing classifiers. *Journal of Machine Learning Research*, 13:1293–1332, 2012.

[6] MohamadAli Torkamani and Daniel Lowd. Convex adversarial collective classification. In *Proceedings of The 30th International Conference on Machine Learning*, pages 642–650, 2013.

[7] Michael Brückner and Tobias Scheffer. Nash equilibria of static prediction games. In *Advances in Neural Information Processing Systems*, pages 171–179, 2009.

[8] Michael Brückner and Tobias Scheffer. Stackelberg games for adversarial prediction problems. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 547–555. ACM, 2011.

[9] Richard Colbaugh and Kristin Glass. Predictive defense against evolving adversaries. In *IEEE International Conference on Intelligence and Security Informatics*, pages 18–23, 2012.

[10] Amir Globerson and Sam Roweis. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, New York, NY, USA.

[11] Robin Sommer and Vern Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, pages 305–316, 2010.

[12] P. Paruchuri, J. Pearce, Janus Marecki, and Milind Tambe. Playing games for security: An efficient exact algorithm for solving Bayesian Stackelberg games. In *Seventh International Conference on Autonomous Agents and Multiagent Systems*, pages 895–902, 2008.

[13] Manish Jain, Jason Tsai, James Pita, Christopher Kiekintveld, S. Rathi, Milind Tambe, and Fernando Ordonez. Software assistants for randomized patrol planning for the lax airport police and the federal air marshals service. *Interfacs*, 40:267–290, 2010.

[14] Ondrej Vanek, Zhengyu Yin, Manish Jain, Branislav Bosansky, Milind Tambe, and Michal Pechoucek. Game-theoretic resource allocation for malicious packet detection in computer networks. In *Eleventh International Conference on Autonomous Agents and Multiagent Systems*, 2012.

[15] Battista Biggio, Giorgio Fumera, and Fabio Roli. Adversarial pattern classification using multiple classifiers and randomisation. In *Lecture Notes in Computer Science*, pages 500–509, 2008.

[16] Jeff Kahn, Gil Kalai, and Nathan Linial. The influence of variables on boolean functions. In *Foundations of Computer Science, 1988., 29th Annual Symposium on*, pages 68–80. IEEE, 1988.

[17] Stas Filshtinskiy. Cybercrime, cyberweapons, cyber wars: Is there too much of it in the air? *Communications of the ACM*, 56(6):28–30, 2013.

[18] Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. Adversarial classification. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pages 99–108, New York, NY, USA, 2004. ACM.

[19] Ryan O'Donnell. Some topics in analysis of Boolean functions. In *ACM Symposium on Theory of Computing*, pages 569–578, 2008.

[20] Ronald De Wolf. A brief introduction to fourier analysis on the boolean cube. *Theory of Computing, Graduate Surveys*, 1:1–20, 2008.

[21] Vangelis Metsis, Ion Androutsopoulos, and Georgios Paliouras. Spam filtering with naive bayes-which naive bayes? In *CEAS*, pages 27–28, 2006.

Appendices

A Model Analysis

Proposition A.1. *Suppose that $P_A = 0$ and $c = 1$ (i.e., no budget constraint). Then the optimal policy is*

$$q(\vec{x}) = \begin{cases} 1 & \text{if } p(\vec{x}) \geq \frac{G(\vec{x})}{G(\vec{x})+V(\vec{x})} \\ 0 & o.w. \end{cases}$$

Proof. Since we consider only static adversaries and there is no budget constraint, the objective becomes

$$\max_{\vec{q}} \sum_{\vec{x}\in\mathcal{X}} \left[(1-q(\vec{x}))G(\vec{x})(1-p(\vec{x})) - p(\vec{x})v_S(\vec{x})\right],$$

and the only remaining constraint is that $q(\vec{x}) \in [0,1]$ for all $\vec{x}$. Since now the objective function is entirely decoupled for each $\vec{x}$, we can optimize each $q(\vec{x})$ in isolation for each $\vec{x} \in \mathcal{X}$. Rewriting, maximizing the objective for a given $\vec{x}$ is equivalent to minimizing $q(\vec{x})[G(\vec{x}) - p(\vec{x})(G(\vec{x})+V(\vec{x}))]$. Whenever the right multiplicand is negative, the quantity is minimized when $q(\vec{x}) = 1$, and when it is positive, the quantity is minimized when $q(\vec{x}) = 0$. Since

$p(\vec{x}) \geq \frac{G(\vec{x})}{G(\vec{x})+V(\vec{x})}$ implies that the right multiplicand is negative (more accurately, non-positive), the result follows. □

Proposition A.2. *Suppose that $P_A = 0$ and $c|\mathcal{X}|$ is an integer. Then the optimal policy is to let $q(\vec{x}) = 0$ for all $\vec{x}$ with*

$$p(\vec{x}) < \frac{G(\vec{x})}{G(\vec{x}) + V(\vec{x})}.$$

Rank the remaining $\vec{x}$ in descending order of $p(\vec{x})$ and set $q(\vec{x}) = 1$ for the top $c|\mathcal{X}|$ inputs, with $q(\vec{x}) = 0$ for the rest.

Proof. The LP can be rewritten so as to minimize

$$\sum_{\vec{x}} q(\vec{x})[G(\vec{x}) - p(\vec{x})(G(\vec{x}) + V(\vec{x}))]$$

subject to the budget constraint. By the same argument as above, whenever $p(\vec{x})$ is below the threshold, the optimal $q(\vec{x}) = 0$. Removing the corresponding $\vec{x}$ from the objective, we obtain a special knapsack problem in which the above greedy solution is optimal, since the coefficient on the budget constraint is 1. □

B Computing Adversary's Best Response

Theorem B.1. *EVASION is NP-complete.*

Proof. This adversary evasion problem is in NP, as we can non-deterministically pick $a \leq k$ features and verify if $q(\vec{x'}) \leq \lambda$.

We prove that the problem is NP-hard via a reduction from 3-dimensinal matching (3DM). For an arbitrary instance of 3DM, W, Y, and Z are finite, disjoint sets with the same number of d elements. T is a subset of $W \times Y \times Z$, which means T consists of triples (w, y, z) such that $w \in W, y \in Y$, and $z \in Z$. $M \subseteq T$ ($|M| = d$) is a 3-dimensional matching if for any two distinct triples $(w_1, y_1, z_1) \in M$ and $(w_2, y_2, z_2) \in M$, $w_1 \neq w_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$.

Each triple $(w_i, y_i, z_i) \in T$ corresponds to one feature, which controls a set of basis $(s_{w_i}, s_{d+y_i}, s_{2d+z_i})$. There are $n = |T|$ features and $m = |W| + |Y| + |Z| = 3d$ basises, which forms the basis matrix as the figure 9 below. Each elements within the matrix $b_{ji} = 1$ denotes that the jth basis is controlled by the ith feature; otherwise 0. As each feature controls exactly one basis from each part, we have for any feature $i (1 \leq i \leq n)$ and basis $j (1 \leq j \leq m)$, $\sum_{j=1}^{d} b_{ji} = 1$, $\sum_{d+1}^{2d} b_{ji} = 1$, $\sum_{2d+1}^{3d} b_{ji} = 1$, $(d = \frac{1}{3}m)$. Let $k = d$, $\lambda = q(x) - 3d/D$, $(D \geq 3d)$, $\Delta = q(\vec{x}) - q(\vec{x'})$. If $q(x') \leq \lambda$, we have $\Delta = q(\vec{x}) - q(\vec{xx'}) \geq 3d/D$. Let $\alpha_1 = \alpha_2 = ... = \alpha_m = \frac{1}{2D}$, and x is a vector with all 0. Therefore $\phi_j(x) = \alpha_j(-1)^{s_j x} = \frac{1}{2D}$ for $1 \leq j \leq m$. Consequentially, let $x^{l'}$ denotes the modified instance x', which only differs in feature l with x. If $b_{hl} = 1$, the corresponding basis function would flip the sigh, thus $\phi_h(x^{l'}) = \alpha_l(-1)^{s_h x^{l'}} = -\frac{1}{2D}$. Suppose there are J bases that have been flipped the sign,

$$\Delta = q(\vec{x}) - q(\vec{x'}) = \sum_{j=1}^{m} \alpha_j(-1)^{s_j x} - \sum_{j=1}^{m} \alpha_j(-1)^{s_j x'}$$

$$= \left(\sum_{j\in J} \alpha_j(-1)^{s_j x} + \sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x}\right) - \left(\sum_{j\in J} \alpha_j(-1)^{s_j x'} + \sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x'}\right).$$

As $\sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x} = \sum_{j\in S\setminus J} \alpha_j(-1)^{s_j x'}$, $\Delta = \frac{1}{2D}|J| - (-\frac{1}{2D})|J| = \frac{|J|}{D}$, which means the decrement of $q(x)$ equals to the number of basises that would flip the sign divided by D. It is easy to see how this construction can be accomplished in polynomial time. Therefore, suppose there

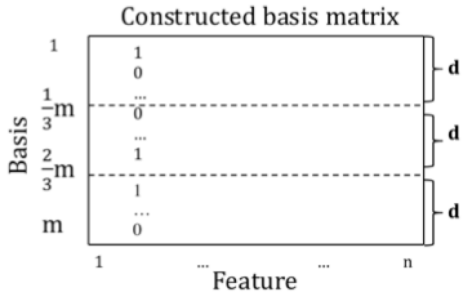


Figure 9: Illustration for the problem construction

are $a \leq k$ features that can be modified in x to satisfy that $q(\vec{x'}) \leq \lambda$. It follows that $\Delta = q(\vec{x}) - q(\vec{x'}) \geq 3d/D$. Additionally, as each feature only control 3 basises, the total number of basis that would flip the sign is $\Delta = q(\vec{x}) - q(\vec{x'}) \leq 3a/D \leq 3k/D = 3d/D$. It derives that $\Delta = 3d/D$, which means there is no overlap between selected basis. Accordingly, subset M ($|M| = d$) is chosen and each triple $(w_i, y_i, z_i) \in M$ corresponds to the set of controlled basises by feature i. Therefore the total number of elements within the selected subsets in M satisfies $|W| + |Y| + |Z| = \Delta \cdot D = 3d$. So any two selected distinct triples $(w_1, y_1, z_1) \in M$ and $(w_2, y_2, z_2) \in M$, $w_1 \neq w_2$, $y_1 \neq y_2$, and $z_1 \neq z_2$. This means if there is a solution for the adversary evasion problem, there exists a 3-dimensional matching.

Conversely, suppose M is a 3DM. The d selected exclusive triples correspond to $k = d$ specific feature, each of which controls 3 basis. As all the triples are non-overlapped, there are $3d$ different responding basises that would flip the sign, which means $q(\vec{x'}) = q(\vec{x}) - \Delta = q(\vec{x}) - 3d/D = \lambda$. Therefore, the adversary evasion problem can be solved if and only if a 3DM exists. □

Theorem B.2. *Suppose that the number of inputs in any basis is bounded by a constant c. Then* ApproxEvasion *(Algorithm 3) computes a solution x' to problem 6 which achieves $\hat{\Delta} \geq \frac{\Delta^*}{1+\epsilon}$, where $\hat{\Delta} = \Delta(x')$ in time poly$(n, \frac{1}{\epsilon}, 2^c)$.*

Proof. The operations of $Trim$ and removing from D_l every member that is greater than $q(\vec{x})$ maintain the property that every element of D_l meets our decreasing requirement. For every element d_i in D_i that the corresponding retrieved value is at most $q(x)$, there exists an element $d_k \in D_i$ such that $\frac{Retrieve(d_i)}{(1+\epsilon/2n)^i} \leq Retrieve(d_k) \leq Retrieve(d_i)$. This

must hold for the optimal Δ^*, therefore there exists an element $d \in D_n$ that $\frac{\Delta^*}{(1+\epsilon/2n)^n} \leq Retrieve(d) \leq \Delta^*$. Thus $\frac{\Delta^*}{Retrieve(d)} \leq (1+\frac{\epsilon}{2n})^n$. As this inequality must also hold for $\hat{\Delta}$, $\frac{\Delta^*}{\hat{\Delta}} \leq (1+\frac{\epsilon}{2n})^n$. Since $\lim_{n\to\infty}(1+\epsilon/2n)^n = e^{\epsilon/2}$ and $\frac{d}{dn}(1+\epsilon/2n)^n > 0$, the function $(1+\epsilon/2n)^n$ increases with n and we have $(1+\epsilon/2n)^n \leq e^{\epsilon/2} \leq 1+\epsilon/2+(\epsilon/2)^2 \leq 1+\epsilon$. Therefore, $\hat{\Delta} \geq \frac{\Delta^*}{1+\epsilon}$.

Next we show that it is a polynomial-time approximation scheme based on a restrictive feature group size c, which is the maximum size of each feature group obtained from Algorithm 4. To analyze the run time, we need to derive the bound on the length of D_i. After trimming between groups of features, successive elements d and d' of D_i must have the relationship $d'/d > 1+\epsilon/2n$. That is, they must differ by a factor of at least $1+\epsilon/2n$. Each list, therefore, constraints the value 0, possibly value $\delta > 0$, which is a small number less than the minimal α value, and up to $\lfloor \log_{1+\epsilon/2n} \frac{q(x)}{\delta} \rfloor$. Therefore we can derive that the number of elements in each list D_i is at most

$$2^c\left(\log_{1+\epsilon/2n}\frac{q(x)}{\delta}+2\right) = 2^c\left(\frac{\ln\frac{q(x)}{\delta}}{\ln(1+\epsilon/2n)}+2\right) \quad (8)$$

$$\leq 2^c\left(\frac{2n(1+\epsilon/2n)\ln\frac{q(x)}{\delta}}{\epsilon}+2\right) \quad (9)$$

$$< 2^c\left(\frac{3n\ln\frac{q(x)}{\delta}}{\epsilon}+2\right). \quad (10)$$

Therefore, this bound of the list length is polynomial in the size of the input n when $c \leq \log_2 n$. Since the running time of *ApproxAdversaryEvasion* is polynomial in the lengths of the D_i, we conclude that there is a polynomial-time approximation scheme ($O(n \cdot 2^c)$) with respect to the restricted feature group size as c ($c \leq \log_2 n$). □

Algorithm 3 returns the approximate solution of maximum Δ to obtain a minimal $q(\vec{x'})$ by modifying less or equal to k features.

Algorithm 3 *ApproxAdversaryEvasion*$(F, q(\vec{x}), \epsilon, k)$

```
n ← |F|
D_0 ← {(∅, 0)}   // tuple d_i = (feaSet, value) ∈ D
G = GenFeaGroup(F, S)
l ← 0
for i ← 1 to |G| do
  for j ← 1 to |g_i| do
    l ← l + 1   // merge two tuple-lists by d_i.value
    D_l ← MergeTuple(D_{l-1}, AddFea(D_{l-1}, f_ij, k))
  end for
  D_l ← Trim(D_l, ε/2n)
  remove elements from D_l that d_l.value > q(x⃗)
end for
let d* correspond to the maximum d.value in D_n
return d*
```

As the length of D_i can be 2^i, which makes the merge algorithm take exponential time, here we employ the Algorithm 6 to trim the list length. The idea is that if some combination of features make the decrease of $q(\vec{x})$ similar, then only one combination should be kept. This means that with a trimming parameter δ, for any element d_i removed from D_i, there is an element d_j that approximates d_i, that is,
$\frac{Retrieve(d_i)}{1+\delta} \leq Retrieve(d_j) \leq Retrieve(d_i)$.
However, this $Trim$ action can only be done for features that have no common basises to avoid missing qualified feature combination. Therefore, Algorithm 4 is applied to group the features that need to be added as a whole before $Trim$; and algorithm 5 helps to form different feature combinations and guarantee only less or equal to k features are considered.

Algorithm 4 *GenFeaGroup*(F, S)

```
G ← ∅
n ← |F|
m ← |S|
for j ← 1 to m do
  for i ← 1 to n do
    g_j ← ∅
    if s_ji = 1 then
      g_j ← g_j ∪ f_i
    end if
  end for
  G ← G ∪ g_j
end for
G ← DisjointSet(G)   // convert G to disjoint-sets
return G
```

Algorithm 5 *AddFea*(D, f, k)

```
m ← |D|
D' ← ∅
for i ← 1 to m do
  if size(i.set ∪ f) ≤ k then
    t'_i.set ← t_i.set ∪ f
    t'_i.value ← Retrieve(t'_i.set)
    insert t'_i into ordered D' by t'_i.value
  end if
end for
return D'
```

Algorithm 6 *Trim*(D, ϵ)

```
m ← |D|
D' ← d_1
last ← d_1.value
for i ← 2 to m do
  if d_i.value > last · (1 + ε) then
    append d_i onto the end of D'
    last ← d_i.value
  end if
end for
return D'
```

Finally, for each feature combination we would use the algorithm 7 to obtain the corresponding value based on the

chosen bases. Our goal is to find the feature combination that reduce the most from $q(\vec{x})$ by flipping fewer features, which means the strategy x' can have a higher chance to pass the classifier after less modification on the original "ideal" instance x.

Algorithm 7 *Retrieve*(d)

$w \leftarrow \emptyset$
for $f_i \in d$ **do**
  $w \leftarrow w \oplus w_{f_i}$ // w_{f_i} is basis set controled by f_i
end for
$v \leftarrow \sum_{s_j \in w} -2\alpha_{xj}$ // α_{xj} is the actual value in x
return v

C Experiments

Here we test the AAS scheme with the same set up of simulations on the feature space of 500, and similar results shown as below have demonstrated the consistency and robustness of our proposed approach.

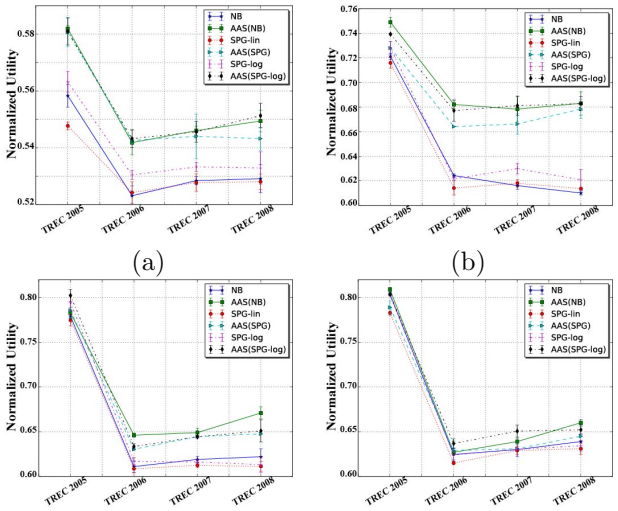


Figure 10: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 0.2$,$V(x) = G(x) = 1$, $P_A = 1$ (a) c=0.1; (b) c=0.3; (c) c=0.5; (d) c=0.9.

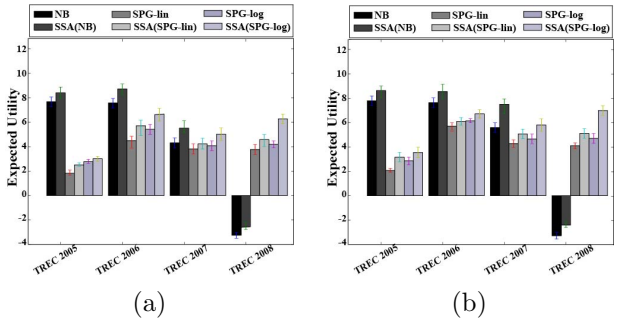


Figure 11: Comparison of the expected utility assuming $P_A = 1$, $V(x) = G(x) = 1$; (a) $c = 0.1$; (b) $c = 0.3$.

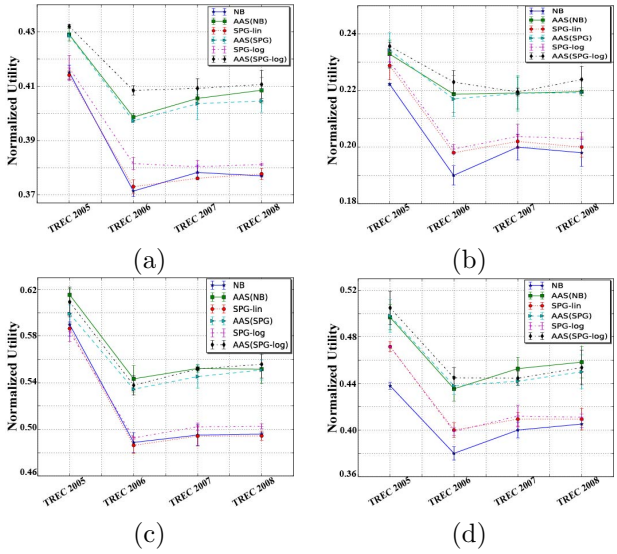


Figure 12: Comparison of normalized utility on TREC data, trained on year 2005, and tested on years 2005-2008. Our method is labeled as $AAS(\cdot)$, where the parameter is the classifier that serves to provide $p(x)$. The following parameters are used: $\delta = 0.2$,$G(x) = 1$, $P_A = 1$ (a) $V(x) = 2$, c=0.1; (b) $V(x) = 10$, c=0.1; (c) $V(x) = 2$, c=0.3; (d) $V(x) = 10$, c=0.3.

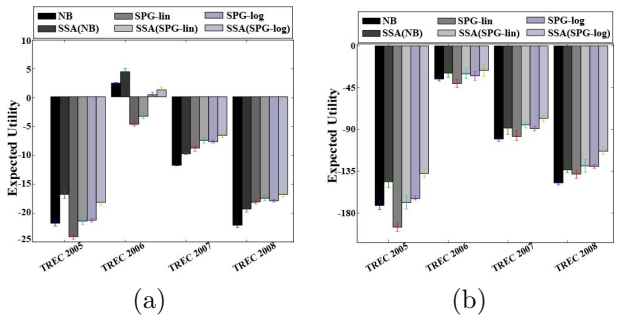
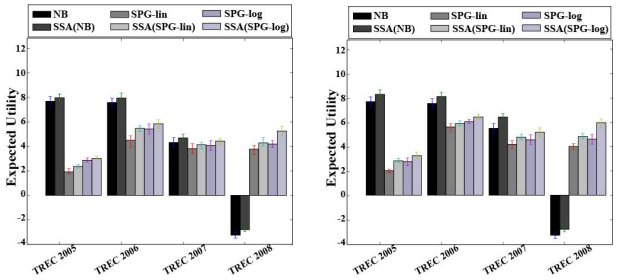
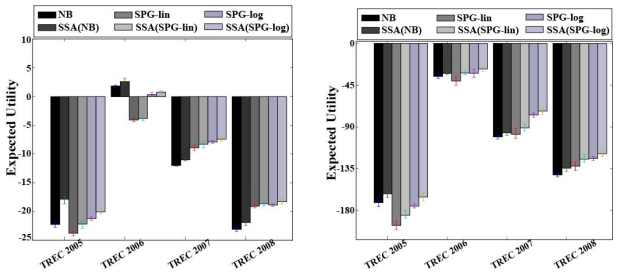


Figure 13: Comparison of the expected utility assuming $P_A = 1$; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.



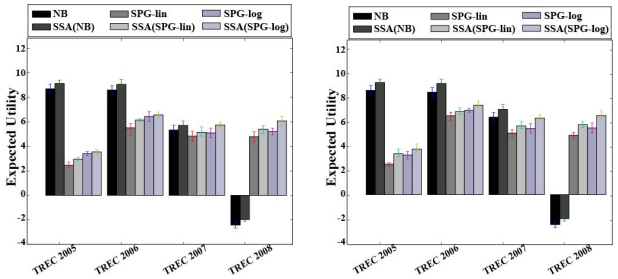
(a) (b)

Figure 14: Comparision of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $c = 0.1$; (b) $c = 0.3$.



(a) (b)

Figure 15: Comparison of the expected utility assuming $P_A = 1$, introducing parameter error with 0.2 for δ; (a) $V(x) = 2$; (b) $V(x) = 10$. $c = 0.3$.



(a) (b)

Figure 16: Comparison of the expected utility assuming $P_A = 1$, introducing adversarial model error; (a) $c = 0.1$; (b) $c = 0.3$.