Adversarial Learning

Daniel Lowd
Dept. of Computer Science and Engineering
University of Washington, Seattle
Seattle, WA 98195-2350
lowd@cs.washington.edu

Christopher Meek
Microsoft Research
Redmond, WA 98052
meek@microsoft.com

December 20, 2005

Abstract

Many classification tasks, such as spam filtering, intrusion detection, and terrorism detection, are complicated by an adversary who wishes to avoid detection. Previous work on adversarial classification has made the unrealistic assumption that the attacker has perfect knowledge of the classifier [1]. In this paper, we introduce the adversarial classifier reverse engineering (ACRE) learning problem, the task of learning sufficient information about a classifier to construct adversarial attacks. We present efficient algorithms for reverse engineering linear classifiers with either continuous or Boolean features and demonstrate their effectiveness using real data from the domain of spam filtering.

1 Introduction

> "If you know the enemy and know yourself, you need not fear the result of a hundred battles. If you know yourself but not the enemy, for every victory gained you will also suffer a defeat. If you know neither the enemy nor yourself, you will succumb in every battle."
>
> – Sun Tzu, The Art of War [5]

Machine learning algorithms have been successfully deployed in many scenarios, such as spam filtering [], web page ranking [], credit card fraud detection [], and has been proposed for choosing which securities brokers to audit [] and for detecting terrorists []. In each of these examples, a classifier is applied to human actions. The effect is that certain behaviors are punished whenever they are detected: emails that resemble spam are blocked; web pages that look irrelevant receive low rankings; and suspicious activity is audited. In each case, malicious adversaries have an incentive to fool the classifier, to adjust their actions slightly in order to remain undetected.

While there is growing interest in making classifiers robust to adversaries, little has been done to investigate the adversary's problem in detail. For example, Dalvi et al. unrealistically assume that the adversary acts with complete knowledge of the classifier but complete ignorance of any adversarial classification scheme [1]. As Tzu points out, understanding adversaries is key to defeating them.

In this work, we adopt the perspective of an adversary to investivate how best to defeat a classifier, both theoretically and practically. The adversary's goal is to find the best possible instance that is undetected by the classifier. Our notion of "best" comes from an adversarial cost function that assigns a non-negative real-valued number to each instance in the instance space. Depending on the domain, this could represent the fact that some spam emails are less effective than others, some web page manipulations are more time consuming to perform, or that some fraudulent transactions yield less money than others. When an adversary cannot find a minimal-cost undetected instance, we measure its performance relative to this optimal reward.

Strategies for finding good instances fall into two categories, passive and active attacks. In a passive attack, an adversary uses domain-specific background knowledge along with any available data to develop

an attack strategy. In an active attack, an adversary is allowed to query the classifier many times in an attempt to reverse engineer the classifier. Passive attacks make sense when the testing a hypothesis is expensive or costly. For example, in a fraud scenario, extensive experimentation could lead to jail time. In spam filtering, a spammer can often afford to send many test emails in order to learn more about the filter.

An active attack can be formulated as a learning task, similar to active learning with membership queries but with the goal of minimizing cost rather than learning the entire decision surface. Using this formulation, we can evaluate the hardness of reverse engineering different concept classes under different adversarial cost functions.

Since passive attacks tend to be domain-specific, it is difficult to say much about them theoretically. We empirically evaluate several simple heuristics in the spam domain and compare them to active attacks.

2 Problem Definition

We define adversarial problems over a fixed *instance space*, X, consisting of n-dimensional feature vectors. Each feature X_i may be real, integer, Boolean, etc. We refer to elements $\mathbf{x} \in X$ as *instances*. We use x_i to denote the value of the ith feature of the instance $\mathbf{x}$.

A *classifier*, c, is a function from instances $\mathbf{x} \in X$ to values in the set $\{0, 1\}$ (i.e., a Boolean classifier). We refer to instances $\mathbf{x}$ for which $c(\mathbf{x}) = 1$ as *positive instances* (i.e., those labeled as malicious), and those for which $c(\mathbf{x}) = 0$ as *negative instances*.

An *adversarial cost function*, $a(\mathbf{x})$ maps instances to non-negative real numbers. The adversarial cost function represents the increased cost (or decreased utility) of using some instances as compared to others. In the spam domain, for example, some spam messages are more effective at selling products. In credit card fraud, forgoing certain purchases may decrease the likelihood of detection, but it may also decreases the fraudster's reward.

In an *adversarial attack*, the adversary's goal is to find a negative instance $\mathbf{x}$ with low cost adversarial cost $a(\mathbf{x})$. In all attacks, it is presumed that the adversary knows the cost function $a(\mathbf{x})$ but not the classifier $c(\mathbf{x})$.

We consider two types of adversarial attacks in this paper. In *passive attacks*, the adversary has no direct access to $c(\mathbf{x})$ and must make educated guesses using any background knowledge or available data. In *active attacks*, the adversary can issue *membership queries* to the classifier for arbitrary instances. In both cases, the adversary is also provided with one positive instance, $\mathbf{x}^+$, and one negative instance, $\mathbf{x}^-$. For most domains, this is not an onerous assumption.

The remainder of this section deals with definitions for the analysis of active attacks. Because background knowledge and applicable heuristics will be domain-specific, passive attacks are much harder to analyze than active attacks. We compare both types of attacks empirically in later sections.

The *minimal adversarial cost* (MAC) of a classifier c and cost function a is the minimum cost $a(\mathbf{x})$ over all instances x classified negatively by c:

$$MAC(c, a) = \min_{\mathbf{x}: c(\mathbf{x})=0} a(\mathbf{x})$$

We also define *instances of minimal adversarial cost* (IMAC) as the set of all instances $\mathbf{x}$ classified negatively by c and with minimal cost:

$$\text{IMAC}(c, a) = \{\mathbf{x} \in X | a(\mathbf{x}) = \text{MAC}(c, a) \text{ and } c(\mathbf{x}) = 0\}$$

The *adversarial classifier reverse engineering (ACRE) learning problem* for classifier c and adversarial cost function a is to find an instance $\mathbf{x} \in \text{IMAC}(c, a)$; that is, an instance that is classified as non-malicious and has a minimum cost of all instances classified as non-malicious.

We say that a set of classifiers $\mathcal{C}$ is *ACRE learnable* under a set of cost functions $\mathcal{A}$ if an algorithm exists that, for any $c \in \mathcal{C}$ and any $a \in \mathcal{A}$, finds some $\mathbf{x} \in \text{IMAC}(c, a)$ using only polynomially many membership queries in: n, the number of features; $\mathsf{size}(c)$, the encoded size of c; and $\mathsf{size}(\mathbf{x}^+, \mathbf{x}^-)$, the encoded size of the positive and negative instances.

We assume that numerical parameters in c, $\mathbf{x}^+$, and $\mathbf{x}^-$ are encoded as strings of digits in a known, fixed base, so that excessively large or small values require longer encodings. With minimal changes, we can also handle encodings in scientific notation, e.g. 1.234×10^{5678}, in which the encoding size may be doubly logarithmic in the magnitude of the value.

For situations where finding an IMAC is intractable, we define k-IMAC, the set of all negative instances whose costs are within a constant factor k of the MAC:

$$k - \text{IMAC}(c, a) = \{\mathbf{x} \in X | a(\mathbf{x}) \leq k \cdot \text{MAC}(c, a) \text{ and } c(\mathbf{x}) = 0\}$$

Finally, we say that a set of classifiers $\mathcal{C}$ is is *ACRE k-learnable* under a set of cost functions $\mathcal{A}$ if, for any $c \in \mathcal{C}$ and any $a \in \mathcal{A}$, an algorithm exists that always finds $\mathbf{x} \in k - \text{IMAC}(c, a)$ using a polynomial number of queries as in ACRE learning.

The assumptions that the adversary can issue queries in instance space and that the adversary has a cost function over instances is an idealization. In many domains, an adversary can make educated guesses about the features that define an instance. In others, the mapping from object to instance may be difficult to determine, preventing the adversary from constructing arbitrary instances or defining a cost function over the instance space. Our goal is not to analyze this often domain specific problem of learning about feature representations but to focus on the ability for an adversary to reverse engineer a classifier assuming this knowledge.

3 Boolean Formulae

In this section, we assume that instances have only Boolean features and that each classifier c expresses the set of positive instances as a Boolean formula in terms of the literals X_i $(1 \leq i \leq n)$.

In general, this set of classifiers is not ACRE learnable under most interesting adversarial cost functions. Consider the classifier that only classifies two instances as negative, $\mathbf{x}'$ and $\mathbf{x}''$. This can easily be expressed as a Boolean formula with size $O(n)$. If $\mathbf{x}'$ is the unique IMAC according to the adversarial cost function a, but the provided negative instance $\mathbf{x}^- = \mathbf{x}''$, then the adversary can only find $\mathbf{x}'$ by a lucky guess or an exponential brute-force search.

Certain classes of Boolean formulae, however, are ACRE learnable. In particular, any class of Boolean formulae that can be learned using only a polynomial number of membership queries is ACRE learnable due to the finiteness of the instance space. For example, consider disjunctions of conjunctions of k unnegated literals (monotone k-DNF). We can learn this concept class in $O(n^k)$ queries by exhaustively testing possible k-DNF clauses and find an IMAC by brute-force searching through the negative instances.

In addition, because the adversary is given a positive instance, they can efficiently learn arbitrary conjunctions, e.g. $X_2 \wedge \neg X_4 \wedge X_5$. This can be accomplished by starting from the positive instance $\mathbf{x}^+$ and successively negating each feature to find the features in the conjunction—a feature is listed in the conjunction if, after negating the feature, the modified instance is classified as negative. We can therefore determine the exact conjunction in only n queries. Since the negation of a conjunction is a disjunction, we can analogously learn disjunctions of Boolean literals starting from the negative instance, $\mathbf{x}^-$.

Proving the learnability of less restricted Boolean formulas under different adversarial cost functions is a topic for future work.

4 Linear Classifiers

In this section, we demonstrate the ACRE k-learnability of linear classifiers under the adversarial cost functions described in We consider the case in which all of the features are either real-valued or Boolean.

Linear classifiers are one of the most popular types of classifier, due to their efficacy, simplicity and scalability. For example, many spam filters are naive Bayes models, a special class of linear classifiers.

Support vector machines with linear kernels and maximum entropy models are other examples of linear classifiers widely used for text classification.

A linear classifier consists of a set of n weights (one for each feature X_i), which we represent as a vector $\mathbf{w} \in R^n$, and a threshold, T. Thus, $\mathbf{x}$ is a positive instance if $\mathbf{w} \cdot \mathbf{x} > T$, and is otherwise a negative instance. We refer to $|\mathbf{w} \cdot \mathbf{x} - T|$ as $\mathsf{gap}(\mathbf{x})$. The gap of a negative (positive) instance is the weight that would have to be added (subtracted) to make it a positive (negative) instance.

A pair of instances can indicate the sign of a feature weight. Given a linear classifier c, a *sign witness* to a feature f is a pair of instances, $\mathbf{s}^+$ and $\mathbf{s}^-$ such that $c(\mathbf{s}^+) = 1$, $c(\mathbf{s}^-) = 0$, and $\forall i \neq f, s_i^+ = s_i^-$.

From the classifier definition, $\mathbf{w} \cdot \mathbf{s}^+ > T$ and $\mathbf{w} \cdot \mathbf{s}^- \leq T$, so it follows that $\mathbf{w} \cdot \mathbf{s}^+ - \mathbf{w} \cdot \mathbf{s}^- > 0$. Since $\mathbf{s}^+$ and $\mathbf{s}^-$ only differ in feature f, this reduces to $w_f \cdot (s_f^+ - s_f^-) > 0$. w_f is positive if and only if $s_f^+ > s_f^-$, so the sign witness proves the sign of feature f.

4.1 Continuous Features

We begin with the case in which all features are continuous. Our approach to demonstrate ACRE learnability is to first efficiently approximate feature weights and second to use these approximate weights to identify low cost instances.

We begin by describing FindWitness, a subroutine for finding a sign witness for some feature f, given one positive and one negative instance (e.g., $\mathbf{x}^+$ and $\mathbf{x}^-$). This procedure starts with $\mathbf{x}^+$ and changes feature values one at a time to match those of $\mathbf{x}^-$. At some point, the instance classification must change, and the most recently changed feature, f, must have non-zero weight. The previous value and the current value of the intermediate instance constitute the sign witness. This requires at most n membership queries.

Our algorithm FindContinuousWeights for learning the weights is provided in Algorithm 1. The algorithm proceeds by finding a single feature with non-zero weight, constructing an instance of known gap, and then computing the relative weights of all other features using simple line searches along each feature dimension. We next describe the algorithm in detail.

The inputs to the algorithm are positive and negative instances $\mathbf{x}^+$ and $\mathbf{x}^-$, an approximation threshold ϵ, and a lower bound on the magnitude of the ratio of any two non-zero weights δ. (Although the adversary may not know a good δ a priori, the ACRE learning algorithm we later present provides a value that still guarantees a good approximation.) The first step (i.e., FindWitness) is to find some feature f with non-zero weight and a sign witness $(\mathbf{s}^+, \mathbf{s}^-)$ for that feature. The instances in the sign witness have different values for the feature f and the same values for all other features. Since scaling the weights and threshold by a positive number has no effect on the decision boundary, we may assume that the weight of w_f is 1.0 or -1.0, depending on if its value is larger in the $\mathbf{s}^+$ or $\mathbf{s}^-$. Since w_f has unit magnitude and $\mathbf{s}^+$ and $\mathbf{s}^-$ differ only in feature f:

$$\begin{aligned} \mathsf{gap}(\mathbf{s}^+) + \mathsf{gap}(\mathbf{s}^-) &= |\mathbf{w} \cdot \mathbf{s}^+ - T| + |\mathbf{w} \cdot \mathbf{s}^- - T| \\ &= \mathbf{w} \cdot \mathbf{s}^+ - \mathbf{w} \cdot \mathbf{s}^- \\ &= \mathbf{w} \cdot (\mathbf{s}^+ - \mathbf{s}^-) \\ &= |\mathbf{s}_\mathbf{f}^+ - \mathbf{s}_\mathbf{f}^-| \end{aligned}$$

We refine the gap between our original sign witnesses using a binary search on the value of feature f to find a negative instance $\mathbf{x}$ with gap less than $\epsilon/4$. This requires $O(\log(1/\epsilon) + \mathsf{size}(\mathbf{s}^+, \mathbf{s}^-))$ queries. We increase or decrease x_f by 1.0 to arrive at an instance with gap between 1 and $1 + \epsilon/4$.

Finally, we compute the relative weight of each other feature using a line search. This consists of increasing or decreasing each x_i exponentially until the class of $\mathbf{x}$ changes, then bounding its exact value using a binary search. By finding feature values within $(1 + \epsilon/4)$, our total error is at most $(1 + \epsilon/4)^2 < 1 + \epsilon$, for $\epsilon < 8$. We ensure termination by testing the addition or subtraction of $1/\delta$ first; if the class remains unchanged, we assume $w_i = 0$. The number of queries per feature is logarithmic in $1/\epsilon$ and the ratio w_f/w_i. Under our assumed encoding, $\log(w_f/w_i)$ is $O(\mathsf{size}(c))$, so the total number of queries is polynomial.

Note that very large and very small weights require lengthier encodings. In fact, if we know the encoding length of the original classifier, $\mathsf{size}(c)$, then no parameter can have magnitude greater than $2^{\mathsf{size}(c)}$ and none can be less than $2^{-\mathsf{size}(c)}$. We can therefore find the exact weights by letting $\epsilon = \delta = 2^{-2\mathsf{size}(c)}$.

Algorithm 1 FindContinuousWeights(x^+, x^-, ϵ, δ)

$(\mathbf{s}^+, \mathbf{s}^-, f) \leftarrow$ FindWitness($\mathbf{x}^+, \mathbf{x}^-$)
$w_f \leftarrow 1.0 \cdot (s_f^+ - s_f^-)/|s_f^+ - s_f^-|$
Use $(\mathbf{s}^+, \mathbf{s}^-)$ to find negative instance $\mathbf{x}$ with $\mathsf{gap}(\mathbf{x}) < \epsilon/4$
$x_f \leftarrow x_f - w_f$
for each feature $i \neq f$ **do**
  Let $\hat{\mathbf{i}}$ be the unit vector along the ith dimension
  if $c(\mathbf{x} + \hat{\mathbf{i}}/\delta) = c(\mathbf{x} - \hat{\mathbf{i}}/\delta)$ **then**
    $w_i \leftarrow 0$
  else
    $w_i \leftarrow$ LineSearch($\mathbf{x}, i, \epsilon/4$)
  end if
end for

Theorem 4.1. *Let c be a continuous linear classifier with vector of weights $\mathbf{w}$, such that the magnitude of the ratio between two non-zero weights is never less than δ. Given positive and negative instances $\mathbf{x}^+$ and $\mathbf{x}^-$, we can find each weight within a factor of $1 + \epsilon$ using a polynomial number of queries.*

Proof. Follows from the correctness of the algorithm and the fact that each step uses at most polynomially many membership queries. □

We now describe how to use approximately learned feature weights to identify low-cost instances, relative to linear cost functions.

Recall that linear cost functions define the cost of an instance as a weighted sum of feature differences, relative to some base instance $\mathbf{x}^\mathbf{a}$. In a continuous linear classifier, we can arrive at an approximate IMAC instance by changing only one feature in $\mathbf{x}^\mathbf{a}$: the feature f with highest weight-to-cost ratio, $|w_f|/a_f$. Were we to use any other feature, we could always achieve the same benefit for cheaper by changing f instead.

This property of linear cost functions and linear classifiers enables us to efficiently approximate instances of minimal cost with arbitrary precision. Our algorithm FindContinuousIMAC for finding these instances is provided in Algorithm 2.

The inputs to our algorithm are a pair of positive and negative instances, $\mathbf{x}^+$ and $\mathbf{x}^-$, and an approximation threshold, ϵ.

The first step is to approximately learn all feature weights, within $1 + \epsilon/4$. Algorithm 1 depends on knowing the minimum absolute weight ratio, δ. Although the adversary may not know this value, we can use the minimum feature cost ratio, a_i/a_j, in its place. Since no feature with a smaller weight can have the largest cost ratio, we may safely approximate those feature weights as zero.

We then select the feature f with highest weight-to-cost ratio. Since our weights are learned approximately, this may not be the optimal feature, but it's close enough that our cost is within a fixed ratio of optimal. We find a $(1 + \epsilon)$-IMAC by doing a line search from $\mathbf{x}^\mathbf{a}$ along dimension f, to bound the change in x_f^a within a factor of $1 + \epsilon/4$. Our total error is therefore $(1 + \epsilon/4)^2 < 1 + \epsilon$. The number of queries required for this line search is $O(\log(1/\epsilon) + \log(\mathsf{gap}(\mathbf{x}^\mathbf{a})))$, but $\mathsf{gap}(\mathbf{x}^\mathbf{a})$ and $1/\epsilon$ are constants.

Algorithm 2 FindContinuousIMAC(x^+, x^-, ϵ)

$\delta \leftarrow \min_i a_i$
Run FindContinuousWeights($x^+, x^-, \epsilon/4, \delta$)
$f \leftarrow \mathsf{argmax}_i |w_i|/a_i$
$t \leftarrow$ LineSearch($\mathbf{x}^\mathbf{a}, f, \epsilon/4$)
Let $\hat{\mathbf{f}}$ be the unit vector along dimension f
return $\mathbf{x}^\mathbf{a} + t\hat{\mathbf{f}}$

Theorem 4.2. *Linear classifiers with continuous features are ACRE $(1+\epsilon)$-learnable under linear cost functions.*

Proof. Follows from the correctness of the algorithm and the fact that each step uses at most polynomially many membership queries. □

4.2 Boolean Features

We now consider the case in which all features are Boolean. In sharp contrast to the case in which all features are continuous, we show that learning even the sign of all features is NP-hard. Despite this hardness result, we demonstrate that, for uniform linear cost functions, the problem is ACRE 2-learnable. Unlike the previous analysis, the adversary succeeds in this reverse engineering problem by obtaining only partial knowledge of the classifier while identifying near optimal instances.

Evidence regarding the sign of a feature weight is provided by a sign witness. If a feature has no sign witness then the feature is irrelevant to the adversary because changing the feature in any instance never changes the class. The following theorem demonstrates that even determining which features are relevant can be very hard to do.

Theorem 4.3. *In a linear classifier with Boolean features, determining if a sign witness exists for a given feature is NP-complete.*

Proof. Clearly, the problem is in NP, because we can non-deterministically pick a witness, if one exists, and verify it with only 2 queries to the classifier.

We prove that the problem is NP-hard via a reduction from subset sum. In the subset sum problem, we are given a set of integers $S = \{s_1, s_2, \ldots, s_n\}$ and an integer t and want to determine if there exists a subset $S' \subseteq S$ such that $\sum_{s \in S'} s = t$.

We convert an instance of subset sum into a linear classifier with $n+1$ features where the ith feature weight $w_i = s_i$ for $1 \leq i \leq n$. The $n+1$st feature weight is set to some ϵ in the range $(0, 1)$. We further set the classifier's threshold to t.

If there exists a sign-witness $(\mathbf{x}^+, \mathbf{x}^-)$ for the $n+1$st feature, then $\mathbf{x}^-$ must have a gap less than ϵ, which is less than 1:

$$|\mathbf{w} \cdot \mathbf{x}^- - T| < 1$$

From our construction of $\mathbf{w}$ and T, this is equivalent to:

$$|(s_1, \ldots, s_n, \epsilon) \cdot \mathbf{x}^- - t| < 1$$

Because t and the s_i's are all integers, and because $x^-_{n+1} = 0$, the left-hand side reduces to a sum of integers. It must therefore evaluate to a non-negative integer less than 1, namely, 0:

$$(s_1, \ldots, s_n, \epsilon) \cdot \mathbf{x}^- - t = 0$$

By adding t to each side and letting $S' = \{s_i | x^-_i = 1\}$, this can be rewritten as:

$$\sum_{s \in S'} s = t$$

(ϵ never appears because $x^-_{n+1} = 0$.) Thus, the existence of a sign-witness implies a solution to the original subset sum problem. □

In spite of this hardness result, Boolean linear classifiers are still ACRE 2-learnable under a uniform linear cost function. We demonstrate this via an algorithm and a proof of its correctness and efficiency. Our algorithm FindBooleanIMAC for finding a low-cost instance is listed in Algorithm 3.

Because we are using a uniform linear cost function we have an ideal minimum-cost instance $\mathbf{x^a}$. For a feature vector $\mathbf{v}$ we use C_v to denote the set of features that have different values in $\mathbf{v}$ and $\mathbf{x^a}$. Again,

because we are using a uniform linear cost function, the cost of a feature vector $\mathbf{v}$ is $c(\mathbf{v}) = |C_v|$. The algorithm terminates due to the fact that we modify $\mathbf{y}$ to reduce the cost and terminate when $\mathbf{y}$ does not change.

The algorithm begins with the negative instance provided and repeatedly modifies it to find instances of lower and lower cost that are still classified as negative. The modifications we allow are removing individual changes (relative to $\mathbf{x^a}$) from the instance or replacing any pair of changes with a single other change. Each modification reduces the instance cost by one. The algorithm terminates when no modification can be made without producing a positive instance.

Algorithm 3 FINDBOOLEANIMAC($\mathbf{x^a}, \mathbf{x}^-$)

```
y ← x⁻
repeat
  y^prev ← y
  for all f ∈ C_y do
    toggle f in y
    if c(y) = 1 then
      toggle f in y
    end if
  end for
  for all f_1 ∈ C_y; f_2 ∈ C_y; f_3 ∉ C_y do
    toggle f_1, f_2, and f_3 in y
    if c(y) = 1 then
      toggle f_1, f_2, and f_3 in y
    end if
  end for
until y^prev = y
return y
```

Intuitively, this algorithm works for the following reason: if there exists another negative instance $\mathbf{x}'$ with fewer than half as many changes as $\mathbf{y}$, then the most helpful change in $\mathbf{x}'$ must be over twice as good as the two least helpful changes in $\mathbf{y}$. Since the algorithm considers all possible replacements of two changes with one change, it cannot terminate as long as such an instance exists. Some additional complexity is introduced by the fact that we can only add changes that are not already present in the current instance. However, we can make a similar argument considering only the disjoint changes. Our proof fully formalizes this.

We begin with a mathematical lemma, then use it to prove the algorithm's correctness.

Lemma 4.3.1. *For two sequences of non-positive real numbers $(s_1, \ldots, s_m)$ and $(t_1, \ldots, t_n)$, if the following conditions hold*

$$\textstyle\sum_i s_i \leq u \tag{1}$$

$$n > 2m \geq 2 \tag{2}$$

$$\textit{for all } j, \textstyle\sum_i t_i - t_j > u \tag{3}$$

then there exists j, k, l such that $l \neq k$ and $s_j - t_k - t_l < 0$.

Proof. Suppose that all conditions hold for the two sequences of non-positive numbers.

Aiming for a contradiction, we assume that for all k, l we have $t_k + t_l \leq u/m$. From condition 2 we can write

$$\sum_i t_i \quad = \quad \sum_{j=1}^{m} (t_{2j-1} + t_{2j}) + \sum_{k=2m+1}^{n} t_k. \tag{4}$$

Using our assumption and Equation 4, we obtain

$$\sum_i t_i - \sum_{k=2m+1}^{n} t_k \leq u.$$

Since there is at least one t_i in the second sumation we have a contradiction with Condition 3. Thus, there must exist k, l such that $t_k + t_l > u/m$. Using (1), we can upper bound the size of s_{min}, the smallest number in the s sequence as follows $s_{min} \leq u/m$. Thus, $t_k + t_l > u/m \geq s_{min}$ which proves the claim. □

Theorem 4.4. *Boolean linear classifiers are ACRE 2-learnable under uniform linear cost functions.*

Proof. We demonstrate that Algorithm 3 finds an appropriate instance.

Let $\mathbf{x}$ denote a minimum cost feature vector with $c(\mathbf{x}) = 0$. We use Lemma 4.3.1 to show that the second inner loop of Algorithm 3 will find a change to reduce the cost of $\mathbf{y}$ whenever $c(\mathbf{y}) > 2c(\mathbf{x})$, that is, whenever $\mathbf{y}$ is not a feature vector satisfying the theorem.

We assume that we have just completed the first loop of Algorithm 3 and assume that $c(\mathbf{y}) > 2c(\mathbf{x})$. We let $\mathbf{y}$ be our current feature vector with $c(\mathbf{y}) = 0$.

With each feature f we associate a real-valued quantity δ_f (defined below). We apply Lemma 4.3.1 to the sequences in which the s_i's are the δ's associated with features in $C_x \setminus C_y$ and the t_i's are the δ's assocated with features in $C_y \setminus C_x$.

We use $\mathsf{score}(\mathbf{v})$ to denote the dot product of the feature vector $\mathbf{v}$ and the feature weights $\mathbf{w}$ of our linear classifier: $\mathsf{score}(\mathbf{v}) = \mathbf{w} \cdot \mathbf{v}$.

We define $\delta_f = w_f(1 - 2x_f^a)$. Informally, this represents the change in instance score from adding change f. If $x_f^a = 0$, then changing feature f to 1 adds w_f to the score; otherwise, the change adds $-w_f$ to the score. The $(1 - 2x_f^a)$ term captures this sign change.

We now rewrite the definition of $\mathsf{score}(\mathbf{v})$ in terms of $\mathbf{x^a}$ and C_v using the δ_f values:

$$\mathsf{score}(\mathbf{v}) = \mathbf{w} \cdot \mathbf{x^a} + \sum_{f \in C_v} \delta_f. \tag{5}$$

We say that a feature f is *positive* with respect to a linear classifier and ideal instance $\mathbf{x^a}$ if changing the value of feature f in instance $\mathbf{x^a}$ makes the score of the instance larger; that is, $\delta_f > 0$.

There are no positive features in C_y because we have just completed first loop in Algorithm 3 and if there were any such features they would have been changed. There are also no positive features in C_x because $\mathbf{x}$ is the minimum cost feature vector and if there were any positive features they could be removed to create a feature vector with a lower cost and have the same classification. Therefore the sequences contain non-positive real numbers.

From the fact that $c(\mathbf{x}) = 0$ we know that $\mathsf{score}(\mathbf{x}) < T$. Which, using Equation 5 yields

$$\sum_{f \in C_x \setminus C_y} \delta_f < T - \mathbf{w} \cdot \mathbf{x^a} - \sum_{g \in C_x \cap C_y} \delta_g = u$$

satisfying Condition 1.

After the first loop of Algorithm 3 completes we know that there is no single change can be removed without changing the classification of $\mathbf{y}$ (i.e., $c(\mathbf{y}) = 0$) which implies that

$$\text{for all } f \text{ and } g \in C_y \setminus C_x, \sum_{f \in C_y \setminus C_x} \delta_f - \delta_g > u$$

satisfying Condition 3.

Finally, if $c(\mathbf{y}) > 2c(\mathbf{x})$ then $|C_y| > 2|C_x|$. This implies that $|C_y \setminus C_x| + |C_y \cap C_x| > 2|C_x \setminus C_y| + 2|C_y \cap C_x|$ which, in turn, implies that $|C_y \setminus C_x| > 2|C_x \setminus C_y|$. Because $c(\mathbf{x}) = 0$ and $c(\mathbf{x^a}) = 1$ it must be the case that $C_x \neq \emptyset$. Furthermore, $C_x \not\subset C_y$ otherwise the first loop in Algorithm 3 would remove all of the features in $C_y \setminus C_x$ and $\mathbf{y}$ would be optimal. We have demonstrated that the sequences satisfy Condition 2.

We have shown that all of the conditions of the lemma are satisfied which implies that the second loop will find a change if $c(\mathbf{y}) > 2c(\mathbf{x})$. □

5 Empirical Study: Spam Filtering

We now compare the active attacks discussed in the previous section to several passive attacks in the real-world domain of spam filtering.

We chose the domain of spam filtering for several reasons. First of all, spam is a significant nuisance for most internet users and incurs a significant cost on internet service providers. While spam messages were once infrequent annoyances, spam now makes up the majority of all email [2]. As bandwidth gets less expensive, the volume of spam could continue to increase.

Second, there is already evidence of adversarial behavior in this domain. Spammers frequently disguise their messages by obfuscating "spammy" words and adding unrelated words more indicative of legitimate email. In many cases, spammers can send many test messages to determine what will and will not be caught by a given spam filter.

Finally, spam is one of the easier adversarial domains to study, since the data is widely available and most spam filters are Boolean linear classifiers. These properties make spam a perfect application of our techniques.

In our experimental scenario, an adversary wishes to disguise a spam message in order to get it past a target spam filter. The adversary's control over the instance space consists of changing which words are present in an email. We assume unit cost for adding or removing/obfuscating a word.

For the active attacks, the adversary issues queries by sending messages to a test account protected by the target filter and observing which ones are blocked. Our implemented algorithms are as described earlier, but with a few performance optimizations.

We also consider several passive attack heuristics particular to the spam domain. Each uses publicly available data to create a ranked list of words believed to be helpful in disguising a message. A given spam message is disguised by appending words from this list until it is no long classified as spam. (In a purely passive attack, the number of words to append would have to be guessed as well, so our approach is only a lower bound.)

In the following subsections, we discuss how our classifiers were trained, details of how the attacks were executed, and the emprirical results.

5.1 Classifier Configuration

Our experimental data comes from the Hotmail feedback loop, a system in which users of Hotmail volunteer to label some of the messages they receive as spam or legitimate email. Because the number of labeled messages per user is relatively small, we trained global filters using all data rather personalized spam filters for each user. Our training set consists of 500,000 labeled messages from before May 1, 2004. This was split into tuning and validation sets of 490,000 and 10,000 examples, respectively. The test corpus consists of 10,000 messages received between May 1 and 8, 2004. Email in the test set was received by a different set of users than those of the training set in order to ensure generalization across users, not just across time. Our feature set consists of all tokens that appear in the subject or body of the email, as well as many non-textual features based on the message headers. After selecting only those features that occurred at least 10 times in the training data, just under 290,000 features remain.

From this data, we trained two linear classifiers, a naive Bayes model and a maximum entropy (maxent) model. Naive Bayes was first suggested for spam filtering in [4] and has since become a widely popular choice due to its simplicity and scalability. Maxent is more popular in the text classification community, but has also been applied to spam filtering [7].

Since both naive Bayes and maxent are linear classifiers, the difference between them is not how they are represented but how they are trained. Naive Bayes models are trained *generatively*, to maximize the likelihood of all training data. In contrast, maxent models are trained *discriminatively*, to maximize the likelihood of the class labels given the other features. Support vector machines are an alternate discriminative classifier that we do not explore here.

In a linear spam filter, such as naive Bayes or maxent, we can lower the filter threshold to increase the number of spam messages that are caught, but this also increases the number of legitimate messages

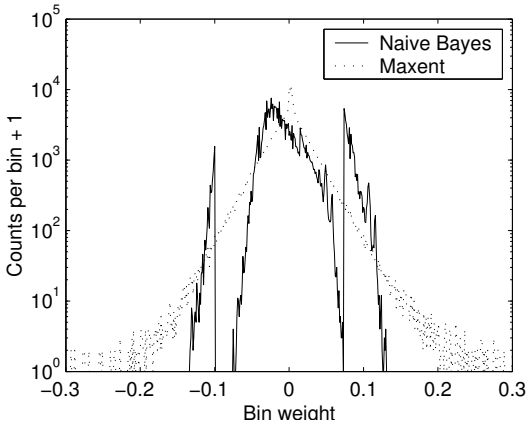

Figure 1: Histograms of normalized feature weights for naive Bayes and maxent filter. Bin width is 0.001. Each bin count was increased by one to permit plotting on a logarithmic scale.

incorrectly classified as spam. We configured our spam filter thresholds so that each filter classified 10% of the legitimate email in our test set as spam. While this may seem like a lot, our error rates are somewhat overestimated due to inconsistencies in how volunteers label their email.

We tuned each algorithm using 490,000 of the training examples, keeping the remaining 10,000 as a validation set. Naive Bayes performs best on the validation set with a Dirichlet prior where $\alpha_i = 0.1$. Maxent performs best with a Gaussian prior for the λ's with zero mean and variance σ^2 set to 0.01. We used these parameters for the remainder of our experiments.

With a fixed false positive rate of 10% on the test set, the maxent filter has 31% fewer false negatives than the naive Bayes filter. (Raw error rates are not meaningful due to labeling error in the test set.)

To better understand the differences between the two classifiers, we looked to their weight distributions. In order to make filter weights comparable, we scaled each filter's weights so that given a false positive rate of 10%, the median score of a true positive is 1 greater than the threshold. This provides a very reasonable interpretation of the weights: a weight of -0.01, for example, provides 1% of the weight necessary to get the median blocked spam past the filter. Put another way, a list of words whose weights sum to -1 is sufficient to get half of the currently blocked spam through the filter.

In spite of its lower error rate, the maxent filter also has large negative weights for many features, suggesting greater susceptibility to good word attacks. Figure 1 shows histograms of naive Bayes and maxent feature weights. The heavier tails of the maxent distribution represent features with relatively large positive or negative weights. The sharp spikes for naive Bayes correspond to features that appear a few times but only in spam or only in legitimate email. The locations of these spikes are determined by the prior weight tuning parameter.

These differences are interesting because the higher-weight features in maxent suggest greater potential vulnerability.

5.2 Passive Attacks

In passive attacks, attackers have no direct access to the spam filter. At best, they can make educated guesses as to which words are helpful and which are harmful. In our attacks, we focus on identifying helpful words via three different heuristics.

The first and simplest approach is to choose random words from some larger set of words in the hope that such words are more likely to be associated with legitimate email than with spam. The set of words could come from a dictionary or be chosen on the basis of the spammer's domain knowledge. We chose to

use dictionary words because these are available electronically. We refer to this type of attack as a *dictionary attack*.

To make a list of random dictionary words, we selected at random dictionary words that appeared in our filter. Approximately 23,000 of the 290,000 filter features are in our chosen dictionary, the first English ispell dictionary, english.0[1].

A more sophisticated approach would be to look at what words appear most often in legitimate email and use those. We assume that legitimate email is hard to come by for the average attacker and so use word frequencies from four different English corpora: Reuters news articles, written and spoken English, and USENET messages from 1992. We call this a *frequent word attack*. Wittel and Wu explore a similar attack using a single list of common English words with spammy words removed (2004).

In our implementation, we selected the 1,000 most frequent words according to each of the four English corpora. In the case of 1992 USENET, there are only 865 words, so we used them all.

Finally, since spam is easy to come by, we can easily consider word frequencies in spam. Words that appear frequently in legitimate email and never in spam are more likely to be effective than words that are common to both. We refer to this heuristic as a *frequency ratio attack*.

In this attack, we ranked each word by its frequency in each English corpora relative to its frequency in the spam corpus. Words that never appear in the spam corpus are excluded. (Few words are excluded for this reason, since the spam corpus features over 27,000 distinct dictionary words.) Since the 1992 USENET corpus started out with only 865 words, we selected the top 250 words; for the other corpora, we selected the top 1,000 words.

Note that the frequency ratio method is very similar to how naive Bayes computes weights. In fact, with the exact training data and configuration of the naive Bayes filter, one could discover its exact weights. Lacking those details, an attacker can still approximate the filters using these publicly available corpora.

5.2.1 Public Corpora for Passive Attacks

Our spam training data consists of all spam on spamarchive.org from the month of April, 2004[2]. Since our filters are trained using data up to May 1, we used spam from the same time period. To convert a set of emails into word frequencies, we selected the message bodies, normalized case, and tokenized them, splitting on non-letters. We further restricted the set of words to lower-case dictionary words.

The Reuters articles are from the Reuters-21578, Distribution 1.0 text categorization test collection, a set of 21,578 documents (but only 19,043 complete articles) from Reuters newswire in 1987[3]. We tokenized all article bodies (splitting on non-letters), normalized case, and generated frequency counts for all dictionary words. Uppercase words, such as proper nouns, are excluded from the list.

The written and spoken English corpora are based on the British National Corpus[4]. We used the word frequencies given but with a certain amount of regrouping. First, we removed number words, since words such as "one" or "five" are more likely to be written as "1" or "5" in an email. We also summed frequencies for words that would be indistinguishable when split on punctuation, (e.g., "can" and "can't"), and removed part-of-speech distinctions that resulted in repeated words (e.g., "no" appeared separately as a determiner and as an interjection).

The word frequencies for the 1,000 most common words on USENET in 1992 are publicly available as well[5]. As with written and spoken English, we split on punctuation and regrouped, ending up with 865 words after the regrouping. Most regroupings were due to distinctions of capitalization – "the" and "The" are treated as two distinct words in this corpus, but we combine their separate frequencies into one.

[1] English dictionaries are distributed with ispell, available from http://fmg-www.cs.ucla.edu/fmg-members/geoff/ispell.html

[2] An archive of current spam is available from ftp://spamarchive.org/pub/archives.

[3] The Reuters-21578 text categorization test collection, Distribution 1.0, is freely available for research use at http://www.daviddlewis.com/resources/testcollections/reuters21578.

[4] English word frequencies were obtained from the companion website for *Word Frequencies in Written and Spoken English: based on the British National Corpus*, http://www.comp.lancs.ac.uk/computing/research/ucrel/bncfreq/.

[5] 1992 USENET word frequencies are available from http://www.dcs.shef.ac.uk/research/ilash/Moby/.

5.3 Adversary Configuration

Because not all of the 290,000 features in the classifier are directly manipulable, and even fewer are easily guessed by adversaries, we restricted adversaries to a simple subset: 23,000 English words from the first ispell dictionary, english.0 that appear as features in our filter. We also experimented with two smaller word lists: the 1,000 English words most common in our training data, and 1,000 random English words appearing as features in our filter. We refer to these feature lists as Dict, Freq, and Rand, respectively.

The one exception to these word lists is when the adversary is searching for the first feature witness. In this search, we allow the adversary to change any token found in the body of the positive or negative instance. Our justification is that, given a pair of messages, it's easy to add or remove the few tokens present.

We used a single spam email from our test set to construct a uniform linear cost function. This corresponds to a unit cost per word obfuscated or added to the email.

5.4 Active Attacks

For our active attacks, we applied an optimized version of Algorithm 3. Our modified version maintains the theoretical guarantees of the simpler version, but uses many fewer queries than a naive implementation would.

Our first optimization is to skip unnecessary tests by remembering which changes will never be helpful. For example, any change that yields a positive instance need never be considered again. Additionally, if a change f was unable to take the place of two other changes at one stage in the algorithm, then we need never consider it in the future, since the changes in our current instance only get better on average as the algorithm progresses.

Our second optimization is to consider only $O(n)$ pairs of changes to remove, rather than all $O(n^2)$ combinations. Assume an even number of changes and group all changes into pairs. At least one of these $O(n)$ pairs must be average or worse. If there is an odd number of changes, then this might fail because one change remains unpaired. We compensate for this by constructing a second pairing in which a different change is left unpaired. If no pair of changes from either pairing is average or worse, then it's easy to show that the two left-out changes must together be worse than average. Since the algorithm's correctness only depends on finding a worse than average pair of changes, our theoretical guarantees remain.

5.5 Experimental Results

We ran our modified ACRE algorithm 1,000 times for each filter and word list combination. In each run, we started from a single legitimate email and a single spam email and compared the cost of the negative instance we found to the MAC, computed by greedily removing the largest-weight features and adding the smallest-weight features.

In 16.3% of the naive Bayes runs and 27.6% of the maxent runs, we never found a witness for a single non-zero feature. This happened because we only permitted the adversary to swap tokens in the body of the email; if other features determine the class too strongly, then we may never see a witness. In practice, an adversary could probably still come up with a witness in these cases by adjusting words in the subject and a few other "guessable" properties. At worst, the adversary need only find a different spam/legitimate email pair to start from, and the process will likely succeed.

Table 1 shows the medians and maximums for the adversary's instance cost, the adversary's cost relative the the MAC, and the number of queries used. Tests in which we failed to find a witness were excluded from these calculations.

Overall, our algorithm does quite well at finding low-cost instances: over half the time, it found instances within 17% of the optimal cost. Additionally, its instances only cost 50% more than optimal in the worst case, well below our theoretical bound of costing 100% more.

In terms of queries, our algorithms were reasonably efficient in the average case. Not suprisingly, fewer queries were required to sort through the smaller feature sets, Freq and Rand. More interestingly, naive Bayes models were significantly more difficult, especially in the worst case. This can be attributed to differences

Table 1: Empirical Results in Spam Domain

	med. cost	max cost	med. ratio	max ratio	med. queries	max queries
Dict NB	23	723	1.136	1.5	261k	6,472k
Dict ME	10	49	1.167	1.5	119k	646k
Freq NB	34	761	1.105	1.5	25k	656k
Freq ME	12	72	1.108	1.5	10k	95k
Rand NB	31	759	1.120	1.5	23k	755k
Rand ME	12	64	1.158	1.5	9k	78k

in the weight distributions of the two models. Our maxent model featured more large-magnitude feature weights than our naive Bayes model, leading to lower-cost negative instances and fewer changes to consider removing. This relationship is observable from the differences in the median and maximum adversarial cost for each scenario.

Our algorithms were designed to be generic and easy to analyze, not to be efficient in the number of queries. In practice, especially with domain knowledge, one can significantly reduce the number of queries. See [3] for an in-depth demonstration of this on the same dataset, along with more detailed analysis.

6 Conclusion and Future Work

ACRE learning is a theoretical framework for studying one's enemy and oneself, attacker and the defender, adversary and classifier. Much as PAC learning determines whether concepts can be learned efficiently relative to the natural distribution, ACRE learning determines whether an adversary can efficiently learn enough about a classifier to minimize the cost of defeating it.

But ACRE learning is more than just theory: in the domain of spam filtering, our ACRE learning algorithm performed quite well, easily exceeding the worst-case bounds. In practice, it may be possible to do much better using domain-specific heuristics.

Of course, the algorithms presented are not designed to be efficient in the number of queries but simple to analyze. In practice, especially with domain knowledge, one can significantly reduce the number of queries.

While our preliminary results only cover two types of linear classifiers, we hope that future work will cover additional types of classifiers, cost functions, and even learning scenarios. We have proven certain scenarios to be relatively easy; what scenarios are provably hard?

A number of framework questions remain as well. Under what conditions is ACRE learning robust to noisy classifiers? What can be learned from passive observation alone, for domains where issuing any test queries would be prohibitively expensive? If the adversary does not know which features make up the instance space, when can they be inferred? Can a similar framework be applied to relational problems, e.g. to reverse engineering collective classification? Moving beyond classification, under what circumstances can adversaries reverse engineer regression functions, such as car insurance rates?

Finally, how do such techniques fare against a changing classifier, such as a frequently retrained spam filter? Will the knowledge to defeat a classifier today be of any use tomorrow?

Years of research have led to good classification algorithms. Now that these classifiers have been deployed, adversaries are beginning to attack and defeat them. Common classifiers are fast becoming victims of their own success. One of our goals, although one not addressed in this paper, is to understand the susceptibility of different classifiers to adversarial attacks. Our adversarial learning framework measures the vulnerabilities of different classifiers to different adversaries, which is a first step. We hope that this line of research leads to classifiers that are provably difficult to reverse engineer for any adversary. At the very least, we hope that this framework will lead to useful descriptions of the relative vulnerability of different classifiers against different types of adversaries.

7 Acknowledgements

We thank Mausam for helpful comments on an earlier version of this paper. This work was partially done while the first author visited Microsoft Research, and was partiallly funded by an NSF Graduate Research Fellowship awarded to the first author.

References

[1] N. Dalvi, P. Domingos, Mausam, S. Sanghai, and D. Verma. Adversarial classification. In *KDD '04: Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 99–108. ACM Press, 2004.

[2] J. Graham-Cumming. How to beat an adaptive spam filter, 2004.

[3] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of the Second Conference on Email and Anti-Spam*, Palo Alto, CA, 2005.

[4] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A Bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[5] S. Tzu. The art of war, 500bc.

[6] G. L. Wittel and S. F. Wu. On attacking statistical spam filters. In *Proceedings of the First Conference on Email and Anti-Spam*, Mountain View, CA, 2004.

[7] L. Zhang and T. Yao. Filtering junk mail with a maximum entropy model. In *ICCPOL2003*, pages 446–453, ShenYang, China, 2003.