Adversarial Learning

Daniel Lowd
Department 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

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 [2]. 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.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning—*Concept learning*; F.2 [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous

General Terms

Algorithms, Theory

Keywords

Adversarial classification, linear classifiers, spam

1. INTRODUCTION

Systems using machine learning have been successfully deployed for fighting spam, fraud, and other malicious activities. These systems typically consist of a classifier that flags certain instances as malicious based on a fixed set of features. For example, spam filters classify each incoming email message as spam or legitimate email by using a set of features such as which words are present.

Unfortunately, as classifiers become more widely deployed, the incentive for defeating them increases. In some domains, there is ample evidence that adversaries are actively modifying their behavior to avoid detection. For instance, senders of junk email often disguise their messages by adding unrelated words, sentences, or even paragraphs more indicative of legitimate email than spam.

Dalvi et al. explore the possibility of anticipating such attacks by computing the adversary's optimal strategy [2]. To do so, they make the unrealistic assumption that the adversary has perfect knowledge of the classifier. This is rarely true in practice: adversaries must learn about the classifier using some combination of prior knowledge, observation, and experimentation.

In this paper, we explore the role of *active* experimentation in adversarial attacks. In particular, we consider cases in which an adversary can send membership queries to the classifier to determine whether a specific instance is malicious or not. The adversary's goal is not to perfectly model the classifier, but rather to identify high-quality instances that are not labeled malicious with a reasonable (polynomial) number of queries. We define the adversarial classifier reverse engineering (ACRE) learning problem to formalize this problem. The ACRE learning problem differs significantly from both the probably approximately correct (PAC) model of learning [6] and active learning [1] in that (1) the goal is not to learn the entire decision surface, (2) there is no assumed distribution governing the instances and (3) success is measured relative to a cost model for the adversary.

While solving the problems of adversaries such as spammers may seem counterproductive, we believe that learning the vulnerabilities of current classifiers is the only way to fix them in the future.

The remainder of our paper is organized as follows. We first define when an ACRE problem is learnable in Section 2 and describe adversarial cost functions in Section 3. In Section 4, we present basic results regarding classifiers that are Boolean formulae. In Section 5, we prove efficient reverse engineering algorithms against linear classifiers. In particular, we show that, for some adversary cost functions, these classifiers are ACRE learnable. We present empirical results for the real-world domain of spam filtering in Section 5, and conclude in Section 6.

2. PROBLEM DEFINITION

We define a reverse engineering problem for classifiers 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 in 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

KDD'05 August 21–24, 2005, Chicago, Illinois USA

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*.

We assume that the adversary can issue *membership queries* to the classifier for arbitrary instances and has access to an *adversarial cost function* $a(\mathbf{x})$ that maps instances to non-negative real numbers. 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 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.

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.

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 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.

3. ADVERSARIAL COST FUNCTIONS

The hardness of an ACRE learning problem depends not only on the classifier, but also on the adversarial cost function. In the extreme case, in which the cost function is constant for all instances, the adversary need not issue a single membership query because the adversary is given a negative instance and every negative instance is optimal. While the hardness of an ACRE learning problem does depend on the adversarial cost function, the precise relationship between the hardness of ACRE learnability and the complexity of the adversarial cost function is subtle. For instance, if the adversary can efficiently learn the exact parameters of the classifier, then the problem would be ACRE learnable for any well-behaved adversarial cost functions because the adversary need not issue any membership queries to optimize the cost function. In this paper, we investigate a specific class of cost functions that is easy to analyze but still captures interesting adversarial problems.

A *linear cost function* is the weighted absolute difference between feature values in the base instance, $\mathbf{x}^\mathbf{a}$, and those in the target instance, $\mathbf{x}$:

$$a(\mathbf{x}) = \sum_i a_i |x_i - x_i^a|$$

By representing the cost $a(\mathbf{x})$ in terms of the base instance $\mathbf{x}^\mathbf{a}$, we capture the reasonable assumption that the instances most similar to $\mathbf{x}^\mathbf{a}$ are best. The scalars a_i represent the relative cost of changing each feature, allowing that some features may be more important or more expensive than others. We assume that all a_i are greater than zero.

In spam, for example, $\mathbf{x}^\mathbf{a}$ would be the email that makes the best sales pitch for the adversary's product. Each change to this email reduces its effectiveness, but some changes cost more than others. For instance, one might expect that the cost of removing the product's name would be larger than the cost of removing most other words and thus the scalar associated with the product name would be larger than most other words.

We also consider the restricted subset of *uniform linear cost functions*, where every a_i is one. In the spam domain, this could represent a spammer who simply wishes to add or obfuscate as few words as possible. While uniform linear cost functions are less expressive than linear cost functions, they make ACRE learning tractable for some problems.

4. 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.

5. LINEAR CLASSIFIERS

In this section, we demonstrate the ACRE k-learnability of linear classifiers under the adversarial cost functions described in Section 3. 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.

5.1 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 computing the relative weights of all other features using 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 obtain 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 $2^{-\mathsf{size}(c)}$. Thus, we can find exact weights when $\epsilon = \delta = 2^{-2\mathsf{size}(c)}$.

Theorem 5.1. *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. □

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

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

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 doing this is listed in Algorithm 2.

The inputs to our algorithm are a positive and a negative instance, $\mathbf{x}^+$ 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 5.2. *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. □

5.2 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 5.3. *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 $\mathbf{x}^\mathbf{a}$. 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 f1 ∈ C_y; f2 ∈ C_y; f3 ∉ C_y do
    toggle f1, f2, and f3 in y
    if c(y) = 1 then
      toggle f1, f2, and f3 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.

LEMMA 5.3.1. *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

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

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 5.4. *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 5.3.1 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 5.3.1 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 y (i.e., $c(y) = 0$) 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 $c(y) > 2c(x)$ 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 $c(x) = 0$ and $c(x^a) = 1$ 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 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 $c(y) > 2c(x)$. □

6. EMPIRICAL STUDY: SPAM FILTERING

In the previous sections, we defined the ACRE learning problem and provided efficient algorithms against continuous and Boolean linear classifiers under linear and uniform linear cost functions. In this section, we apply our theoretical results to the real-world domain of spam filtering.

In our experimental scenario, an adversary wishes to disguise some spam message to get it past a target spam filter. The adversary can explore the instance space by changing which words are present in an email. The adversary issues queries by sending messages to a test account protected by the target filter and observing which ones are blocked.

We simulate this scenario experimentally by training spam filters, reverse engineering them, and measuring the cost of disguising messages relative to the optimal cost (MAC). While basing our experimental methods on our theoretical framework, we also endeavor to keep the setup as realistic as possible. For example, we do not assume that the adversary knows the entire feature space, only an easily guessed subset.

6.1 Classifier Configuration

The training data used to configure our spam filters consisted of 500,000 Hotmail messages, voluntarily hand-labeled by the original recipients as "spam" or "legitimate". From these, we extracted almost 290,000 Boolean features.

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].

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 incorrectly classified as spam. We configured our spam filter thresholds so that each 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.

6.2 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.

6.3 ACRE Algorithm

The adversarial learning algorithm applied was 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.

6.4 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.

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

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 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.

7. CONCLUSION AND FUTURE WORK

> "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]

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.

8. ACKNOWLEDGEMENTS

We thank Mausam for helpful comments on a draft 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.

9. REFERENCES

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[2] 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.

[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] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

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