On the Hardness of Evading Combinations of Linear Classifiers

David Stevens
Dept. of Computer and Information Science
University of Oregon
Eugene, OR 97403
dstevens@cs.uoregon.edu

Daniel Lowd
Dept. of Computer and Information Science
University of Oregon
Eugene, OR 97403
lowd@cs.uoregon.edu

ABSTRACT

An increasing number of machine learning applications involve detecting the malicious behavior of an attacker who wishes to avoid detection. In such domains, attackers modify their behavior to evade the classifier while accomplishing their goals as efficiently as possible. The attackers typically do not know the exact classifier parameters, but they may be able to evade it by observing the classifier's behavior on test instances that they construct. For example, spammers may learn the most effective ways to modify their spams by sending test emails to accounts they control. This problem setting has been formally analyzed for linear classifiers with discrete features and convex-inducing classifiers with continuous features, but never for non-linear classifiers with discrete features. In this paper, we extend previous ACRE learning results to convex polytopes representing unions or intersections of linear classifiers. We prove that exponentially many queries are required in the worst case, but that when the features used by the component classifiers are disjoint, previous attacks on linear classifiers can be adapted to efficiently attack them. In experiments, we further analyze the cost and number of queries required to attack different types of classifiers. These results move us closer to a comprehensive understanding of the relative vulnerability of different types of classifiers to malicious adversaries.

Categories and Subject Descriptors

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

Keywords

Adversarial machine learning; spam

1. INTRODUCTION

In a growing number of adversarial domains, including many kinds of spam and fraud, machine learning is being used to detect malicious behavior [5, 6, 8, 18, 17, 14, 13, 3]. In such domains, criminals have a strong incentive to modify their behavior to evade detection. As criminals adapt, system designers respond by updating their classifiers, leading to a never-ending arms race. In order to get the upper hand, we need a better understanding of these adversarial dynamics so that we can understand the vulnerabilities of current approaches and design more robust methods in the future.

In this paper, we look at the theoretical and practical difficulty of evading certain types of non-linear classifiers with discrete feature spaces. To successfully evade a classifier, the attacker must find an instance that is classified as negative (innocent) and accomplishes the attacker's goal, such as an effective spam email that gets past a spam filter. In many cases, the attacker does not know the exact classifier, but can receive some feedback through interaction. For example, spammers may send "test emails" to accounts they control to see if their candidate spams would be blocked. Similarly, creators of comment spam can see if their postings are flagged or deleted, criminals on Twitter can see if their accounts are banned, and web spammers can query search engines searches to see the rankings of their own web pages. However, our focus is not on domain-specific heuristics but on general-purpose attacks which can be applied to a wide variety of domains.

We perform our analysis within the framework of adversarial classifier reverse engineering (ACRE) [10]. We assume that the adversary's preferences can be described by a cost function, where lower costs represent instances that are more desirable, such as more effective spam emails. A concept class is said to be ACRE learnable for a given cost function if an attacker can find the lowest-cost negatively-classified instance using only a polynomial number of membership queries. If an attacker can find an instance within a factor k of optimal with a polynomial number of queries, then the concept class is said to be ACRE k-learnable. Previous work has shown that convex-inducing classifiers are ACRE learnable (to arbitrary precision) in continuous feature spaces when the cost function is an L_1 distance [12], and linear classifiers are ACRE 2-learnable with binary-valued features when the cost function is Hamming distance relative to some "ideal" instance [10]. However, there have been no results on learning non-linear classifiers in discrete feature spaces, except for very limited Boolean formulae [10].

The discrete case is much harder than the continuous case because the adversary can only modify each dimension by discrete steps, and cannot perform arbitrary-precision line searches. This reduces the set of points that can be queried from all of R^n to merely the points on an n-dimension hypercube, assuming n binary-valued features. The discrete case is also arguably more realistic, because many classifiers use discrete attributes such as the presence or absence of a word or the number of times it appears, and these quantities cannot take on fractional values. Even truly continuous features, such as network latency, are often discretized by rounding to some precision, such as milliseconds.

In this initial work, we focus on non-linear classifiers where either the positive or negative class is given by a convex polytope. Since a convex polytope can be defined as an intersection of half-spaces, this is equivalent to an ensemble of linear classifiers where the aggregation function is AND (for a convex positive class) or OR (for a convex negative class). For example, consider a spam filter that has a sequence of individual linear classifiers that recognize different types of spam or recognize spam based on different types of features. Here, the negative class is a convex polytope, since such instances must be labeled as negative by every one of the component linear classifiers. In the limit as the number of component classifiers grows, these polytopes can approximate any convex set, making this a very general and powerful concept class. This class also has practical, real-world relevance: for example, Google uses a similar ensemble to label malicious advertisements [16].

We begin by showing hardness results that demonstrate that this set cannot be attacked efficiently in the worst case. We then consider the restriction where the linear classifiers operate on disjoint sets of features. Here we are able to demonstrate attacks that are as efficient as attacking a single linear classifier when the positive class is convex, and less efficient but still polynomial time when the negative class is convex. After proving these theoretical results, we demonstrate the efficiency of our methods on non-linear classifiers trained from spam data.

2. ACRE LEARNING

We begin by briefly reviewing the definitions of ACRE learning, defining the concept classes of interest, and presenting previous results on linear classifiers with binary-valued features. This will serve as the foundation for our new results on non-linear classifiers with binary-valued features. For further details on many of these topics, see Lowd and Meek [10].

The *instance space*, $\mathcal{X}$, is the set of all possible inputs to the classifier. For this paper, we assume that the instance space consists of n-dimensional binary-valued feature vectors, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. For example, in spam filtering, the ith feature could represent the presence ($x_i = 1$) or absence ($x_i = 0$) of the ith word used by the filter.

For convenience, we use the symbol $\oplus$ as a feature-wise XOR operator with the following notation:

$$(\mathbf{x} \oplus i) = (x_1, x_2, \ldots, \neg x_i, \ldots, x_n)$$

In other words, the ith feature value in the resulting instance has been "toggled" from 0 to 1 or 1 to 0, and all other values remain identical to those in $\mathbf{x}$. This operation can also be used with a set of feature indices, e.g., $(\mathbf{x} \oplus \{i, j, k\})$, which toggles all features in the set.

A *classifier*, $c(\mathbf{x})$, is a function from the instance space to the labels '$-$' and '$+$', indicating the negative/innocent and positive/malicious classes, respectively. A *concept class* is a set of classifiers $\mathcal{C}$. For this paper, one important concept class is *linear classifiers*, in which the label is determined by a weighted sum of the features:

$$c(\mathbf{x}) = \begin{cases} + & \text{if } \mathbf{w} \cdot \mathbf{x} \geq T \\ - & \text{otherwise} \end{cases}$$

where $\mathbf{w}$ is a vector of real-valued weights and T is a real-valued threshold.

An *adversarial cost function*, $a(\mathbf{x})$, is a function from the instance space $\mathcal{X}$ to the non-negative reals R^+ which defines the adversary's relative preferences over the instance space. While many different cost functions are possible, for this paper we assume that it is the Hamming distance relative to the adversary's "ideal" instance, $\mathbf{x^a}$:

$$a(\mathbf{x}) = \sum_{i=1}^{n} |x_i - x_i^a|$$

For a given instance $\mathbf{x}$, we refer to each feature index i for which $x_i \neq x_i^A$ as a "change." Assuming the cost function is clear from context, we use $C_{\mathbf{x}}$ to refer to the set of all such changes:

$$C_{\mathbf{x}} = \{i \mid x_i \neq x_i^a\}$$

Thus, we can equivalently define a Hamming distance cost function as the number of changes: $a(\mathbf{x}) = |C_{\mathbf{x}}|$.

As a concrete example, suppose a spammer wishes to send the most effective spam email that will get past a particular filter. The spammer's ideal spam message is $\mathbf{x^a}$, which has a cost of zero. Other messages have a cost equal to the number of words added or removed from $\mathbf{x^a}$, since emails that are more corrupted are likely to be less effective.

For a given classifier $c(\mathbf{x})$, the minimal adversarial cost (MAC) is the smallest cost of any negatively classified instance, and an instance of minimal adversarial cost (IMAC) is any negative instance with this cost. A k-IMAC is a negative instance with a cost of at most k times the MAC. In a *membership query*, the attacker requests the label $c(\mathbf{x})$ for a chosen instance $\mathbf{x}$. We assume that $\mathbf{x}$ may be any instance in $\mathcal{X}$ without restriction. The attacker may use this information to plan future queries and eventually find an IMAC or k-IMAC.

A concept class $\mathcal{C}$ is *ACRE learnable* for a set of cost functions if, for any $c \in \mathcal{C}$, an adversary can always find an IMAC given one negative instance $\mathbf{x}^-$ and a number of membership queries that is polynomial in the size of the problem representation. The class is *ACRE k-learnable* if the adversary can find a k-IMAC with a polynomial number of queries.

Lowd and Meek [10] show that linear classifiers with discrete features are ACRE 2-learnable with a Hamming distance cost function. They do this constructively by introducing an algorithm, FindBooleanIMAC, and proving its correctness. See Algorithm 1 for pseudocode. FindBoolean-IMAC works by iteratively reducing the cost of a negative instance. It begins with the provided negative example, $\mathbf{x}^-$, and uses two operations to reduce its cost: removing one change in the current instance and replacing two changes in the current instance with one not in the current instance. Recall that a "change" is simply a feature whose value is different in $\mathbf{x^a}$. Each operation reduces the number of changes,

Algorithm 1 FindBooleanIMAC$(\mathbf{x^a}, \mathbf{x^-})$
(adapted from [10])

```
$\mathbf{y} \leftarrow \mathbf{x}^-$
repeat
  $\mathbf{y}^{\mathbf{prev}} \leftarrow \mathbf{y}$
  for all $f \in C_y$ do
    if $c(\mathbf{y} \oplus f) = -$ then
      toggle $f$ in $\mathbf{y}$
    end if
  end for
  for all $f_1 \in C_y$; $f_2 \in C_y$; $f_3 \notin C_y$ do
    if $c(\mathbf{y} \oplus \{f_1, f_2, f_3\}) = -$ then
      toggle $f_1$, $f_2$, and $f_3$ in $\mathbf{y}$
    end if
  end for
until $\mathbf{y}^{\mathbf{prev}} = \mathbf{y}$
return $\mathbf{y}$
```

and thus the cost, by one. When neither operation can be performed without resulting in a positively-classified instance, the algorithm terminates. Lowd and Meek prove that this instance must have at most twice the optimal cost: if an instance with less than half the cost existed, then it would be possible to replace two changes in the suboptimal instance with one of the changes from the optimal instance.

3. EVADING CONVEX POLYTOPES

While previous results have demonstrated that linear classifiers are easy to evade, little is known about attacking non-linear classifiers in discrete feature spaces. Nelson et al. [12] prove that convex-inducing classifiers in a continuous feature space are ACRE learnable with an L_1 cost function. However, their results do not generalize to discrete feature spaces. Continuous spaces are easier than discrete spaces because the features can be adjusted by arbitrarily large or small amounts in order to explore the decision boundary of the classifier. In discrete feature spaces, it is NP-hard to even determine the sign of a feature weight [10].

We now define an interesting non-linear concept class and discuss the worst-case hardness of evading these classifiers in discrete feature spaces. Like Nelson et al. [12], we focus on classifiers where the set of positive or negative instances defines a convex set. However, we further restrict this set to be a convex polytope, which is equivalent to a conjunction or disjunction of linear classifiers when the positive label is interpreted as *True*. The following definitions specify this more precisely.

Definition A *disjunction (or union) of linear classifiers* is an ensemble of linear classifiers in which instances are labeled as positive if any component classifier labels it as positive:

$$c(\mathbf{x}) = \begin{cases} + & \text{if } c_i(\mathbf{x}) = + \text{ for any } i \in \{1, \ldots, C\} \\ - & \text{otherwise} \end{cases}$$

where C is the number of component classifiers.

Definition A *conjunction (or intersection) of linear classifier* is an ensemble of linear classifiers in which instances are labeled as positive if all component classifiers label it as positive:

$$c(\mathbf{x}) = \begin{cases} + & \text{if } c_i(\mathbf{x}) = + \text{ for all } i \in \{1, \ldots, C\} \\ - & \text{otherwise} \end{cases}$$

where C is the number of component classifiers.

To demonstrate the connection between these ensembles and convex polytopes, note that each component linear classifier defines two half-spaces, one for each class. In a disjunction of linear classifiers, the positive class is the *union* of the positive half-spaces from each component, and the negative class is the *intersection* of the negative half-spaces from each component. Thus, in a disjunction of linear classifiers, the negative instances are defined by a convex polytope, and in a conjunction of linear classifiers, the positive instances are defined by a convex polytope. Since these definitions are equivalent, we use them interchangeably.

Convex polytopes, or unions and intersections of half-spaces, are interesting for several reasons. In the limit, convex polytopes can approximate any convex set arbitrarily well. Thus, methods that work on convex polytopes may work on other convex-inducing classifiers, such as convex quadratic classifiers. Unions and intersections of half-spaces are also a practical way to build real-world classification systems, as demonstrated by Google's use of a combination of linear classifiers to detect malicious advertisements [16]. These types of classifiers are also easier to analyze than arbitrary convex-inducing classifiers.

Biggio et al. [2] have also analyzed the difficulty of evading combinations of linear classifiers, although they use a different measure of evasion difficulty. Their results suggest that combinations of classifiers are harder to attack, which is consistent with some of our theoretical results below. They also find that classifier combinations often have lower accuracy than a single linear classifier, which is consistent with our later experimental results.

3.1 Hardness Results for Convex Polytopes

Lowd and Meek [10] prove that even recovering the signs of features in a linear classifier is NP-Complete when the features are binary-valued. Since a linear classifier is a special case of a convex polytope, this results applies to these more complicated classifiers as well. However, while they show that a 2-approximation can be efficiently found in a linear classifier, we will prove that finding a constant factor approximation for convex polytopes requires an exponential number of queries in the worst case. We consider the cases of a convex negative class (disjunction of linear classifiers) and convex positive class (conjunction) separately.

Theorem 1. *With binary-valued features and Hamming distance costs, disjunctions of linear classifiers are not ACRE k-learnable for any constant k.*

Proof. We prove this constructively by defining a set of classifiers for which finding a k-IMAC requires exponentially many queries.

Partition the set of n features into three sets: F of size $\frac{n}{2}+1$, G of size $\frac{n}{2k}$, and H of size $\frac{n}{2} - \frac{n}{2k} - 1$. We construct a set of $\frac{n}{2k}$ classifiers. In the ith classifier, the weight for all features in F is $-\frac{1}{\frac{n}{2}+1}$, the weight for the ith feature in G is $-\frac{n}{k}$, the weight for all other features in G or H is $\left(\frac{n}{k}-1\right) / \left(\frac{n}{2k}-1\right)$, and the threshold is $-1 + \frac{1}{n}$.

Define $\mathbf{x^a}$ as $x_i^a = 0$ for all i, $\mathbf{x}^-$ as $x_i^- = 1$ for $i \in F$, and $\mathbf{x}'$ as $x_i' = 1$ for $i \in G$. Thus, $c(\mathbf{x}^-) = -$, as $-\frac{1}{\frac{n}{2}+1} \cdot \left(\frac{n}{2}+1\right) = -1$, which is less than the threshold of $-1 + \frac{1}{n}$ for each of the component classifiers. To show that $\mathbf{x}'$ is also negative, consider the ith component classifier. The ith element of G contributes a weight of $-\frac{n}{k}$ and the other $\frac{n}{2k}-1$ elements of G each contribute a weight of $\left(\frac{n}{k}-1\right) / \left(\frac{n}{2k}-1\right)$, for a total weight of $-\frac{n}{k} + \frac{n}{k} - 1 = -1$, which is less than the threshold. Since $a(\mathbf{x}^-) = \frac{n}{2}+1$ and $a(\mathbf{x}') = \frac{n}{2k}$, $\mathbf{x}^-$ is not a k-IMAC.

We can describe the negative instances more generally as follows: Any instance that has at least one feature from $G \cup H$ is positive, unless it includes every feature from G and none from H. This is because every feature in G or H has a large positive weight in every or almost every component classifier. This positive weight more than outweighs the total contribution of all features in F. Thus, once such a feature has been added, the only way to satisfy each of the classifiers is to include its "special" feature from G. If any such feature is omitted, then at least one of the classifiers will remain positive, leading to a positive class label. If any extra features from H are added, then there is no way to counteract this positive weight in all classifiers, so the instance will always be positive.

To find a negative instance with low cost, the attacker must determine which features are in G. However, each membership query provides very little information about G since only one non-empty subset of $G \cup H$ can ever result in a negative instance. Thus, G can only be determined by brute-force, which requires $\binom{n/2-1}{n/2k}$ queries. Basic properties and lower bounds of combinations allow us to conclude: $\binom{n/2-1}{n/2k} > \binom{n/3}{n/2k} \geq \left(\frac{2k}{3}\right)^{n/2k}$, which is exponential in n. □

Our proof relies on creating a classifier with $n/2k$ components where a feature that has a positive sign in one component may have a negative sign in another. Thus, an interesting question is: What hardness results can be achieved with only a small number of classifiers and where features have consistent signs? While this question remains open for convex negative instances, we can show such a result when the positive class is convex.

THEOREM 2. *With binary-valued features and Hamming distance costs, conjunctions of linear classifiers are not ACRE k-learnable for any constant k.*

PROOF. When the positive set is a conjunction, then the adversary needs to find an instance that is classified as negative side by just *one* of the component linear classifiers. Suppose that there are two components, c_1 and c_2. In c_1, the weight of the first $\frac{n}{2}$ features is -2, the weight of the next $\frac{n}{4k}$ features is $-4k$, the weight of the last $\frac{n}{2} - \frac{n}{4k}$ features is $+2n$, and the threshold is $-n+1$.[1] Thus, c_1 will classify an instance as negative if it has all of the $\frac{n}{2}$ low-weight features, all of the $\frac{n}{4k}$ higher-weight features, or an appropriate combination of the two sets. In c_2, the weight of the first $\frac{n}{2}$ features is -4, the weight of all other features is 0, and the threshold is $-n-1$. Thus, c_2 will classify an instance as negative if it has at least $\frac{n}{4}+1$ of the $\frac{n}{2}$ low-weight features. Let $x_i^a = 0$ for all i. Thus, the optimal evasion cost for c_1 is $\frac{n}{4k}$ but the optimal evasion cost for the c_2 is $\frac{n}{4}+1$. By choosing $\mathbf{x}^-$ so that $x_i^- = 1$ for $i = 1 \ldots \frac{n}{2}$, $\mathbf{x}^-$ is marked as negative by both components.

[1]If necessary, we can reorder all of the feature indices randomly to ensure that the adversary does not exploit any ordering properties of these feature indices.

To find an instance with a cost lower than $\frac{n}{4}+1$, the attacker must find the $\frac{n}{4k}$ features that have the higher magnitude weights in c_1. To determine that one of these features has a large weight in the c_1, the attacker must find at least one instance $\mathbf{y}$ such that $c_1(\mathbf{y}) = -$ and $c_2(\mathbf{y}) = +$. If all the instances queried are classified as negative by c_2, then the label from c_1 has no effect on the overall conjunction. If all instances queried are classified as positive by c_1, then there is no information to differentiate any of the features used by the c_1.

To be positive in c_2, such a $\mathbf{y}$ must have at most $\frac{n}{4}$ of the low-weight features. In c_1, the total weight of these is $-\frac{n}{2}$ or more. Therefore, to be classified as negative $\mathbf{y}$ must also have at least $\frac{n}{8k}$ of the higher-weight features and none of the positively-weighted features. Since the adversary does not know which are which, it must guess a set of at least $\frac{n}{8k}$ "good" features that includes none of the bad ones. There is no advantage to choosing a set of more than $\frac{n}{8k}$ features, since a larger fraction of these will contain at least one bad feature, leading to a positive instance. There are $\binom{n/2}{n/8k}$ ways to choose a candidate set of $\frac{n}{8k}$ features, and $\binom{n/4k}{n/8k}$ of those possible sets contain entirely "good" features. Using upper and lower bounds on numbers of combinations, the number of possible sets for every "good" set is at least $\left(\frac{2k}{e}\right)^{n/8k}$, which is exponential in n. Thus, the adversary must issue an exponential number of queries before it can be sure to find the features needed to construct a low-cost negative instance. □

4. DISJOINT FEATURES

Since convex polytope classifiers are hard to evade in the worst-case, we now consider a restricted subclass of convex polytopes and prove that this concept class is ACRE 2-learnable under Hamming distance cost functions.

Definition A *disjoint conjunction (or disjunction) of linear classifiers* is a conjunction (or disjunction) of linear classifiers in which the weight for the ith feature is non-zero in at most one of the component linear classifiers.

We use $\mathcal{F}_i$ to denote the set of all features that have non-zero weight in the ith component linear classifier.

This is a significant restriction, but not an implausible one. An email system may have one component classifier trained on plain-text subject and body content, one on HTML content, and perhaps another that looks at images. An email may have to pass one or all of these filters to avoid be labeled as spam. Another example is multi-factor biometric authentication, in which separate classifiers could analyze a person's fingerprint, face, and voice. In general, when combining information from classifiers trained to look at very different aspects of an object, these classifiers may often use entirely separate sets of features.

We now prove that these classes are vulnerable by extending FINDBOOLEANIMAC (Algorithm 1) to work both for a convex negative class and a convex positive class.

4.1 Evading Disjoint Disjunctions

In this case, the class label is a disjunction of the results of each linear classifier. Thus, to be labeled as negative, the adversary must create an instance that is labeled as negative by each linear classifier. We prove that this concept class is ACRE 2-learnable by showing that FindBooleanIMAC always finds a 2-IMAC without requiring any modifications.

Theorem 3. *With binary-valued features and Hamming distance costs, disjoint disjunctions of linear classifiers are ACRE 2-learnable.*

Proof. For a disjunction, the negative instances lie in the intersection of all negative half-spaces. Thus, the attacker must evade every component linear classifier simultaneously. Since their features are disjoint, no change used to evade one component has any affect on the other components. Therefore, the evasion subproblems are independent, and the minimum cost to evade the disjunction is the sum of the minimum cost to evade each component.

Let $\mathbf{y}$ be an instance returned from running FindBooleanIMAC on a disjoint disjunction of linear classifiers. It follows from the termination conditions of FindBooleanIMAC that no individual change can be removed from C_y and no pair of changes can be replaced with one not in C_y without resulting in a positive instance. Since $\mathcal{F}_i$ is a subset of the feature space, it also follows that no individual change in $C_y \cap \mathcal{F}_i$ can be removed and no pair of changes in $C_y \cap \mathcal{F}_i$ can be replaced by one in $\mathcal{F}_i - C_y$ without resulting in a positive instance according to the ith classifier c_i (which is the only classifier affected by these changes). Thus, the restriction of $\mathbf{y}$ to the features in $\mathcal{F}_i$ satisfies the termination conditions of running FindBooleanIMAC on c_i. Since FindBooleanIMAC is guaranteed to find a 2-IMAC for linear classifiers, $|C_y \cap \mathcal{F}_i|$ must be at most twice the minimal evasion cost for c_i. Since each feature subset of $\mathbf{y}$ evades the corresponding component with at most twice the minimum cost, the sum of these costs must be at most twice the total minimum cost. □

4.2 Evading Disjoint Conjunctions

In this case, the class label is a conjunction of the results of each linear classifier. In contrast to the previous case, this means the adversary need only create an instance that is labeled as negative by one component linear classifier. Therefore, the minimum cost to evade a conjunction is the minimum cost to evade any one of its component classifiers. However, in this case it is not enough to just find a negative instance and run FindBooleanIMAC. The reason for this is twofold. Firstly, a negative instance is only guaranteed to be negative in one of the linear classifiers. Thus, FindBooleanIMAC may not find the boundary of any of the classifiers that do not initially classify the instance as negative. Second, while FindBooleanIMAC will provide a 2-IMAC for some component, this component may have a much higher evasion cost than the lowest one. Therefore a more careful exploration is required and it is necessary to have enough negative instances so that each linear classifier classifies at least one instance as negative. The following algorithm takes an adversarial instance $\mathbf{x^a}$ and this minimal set of innocent examples X^- and, using a variant of FindBooleanIMAC as a subroutine, finds a collection of instances that evade every component classifier with at most twice the optimal cost.

Our method for efficiently evading disjoint conjunctions of linear classifiers is shown in Algorithm 2. FindBooleanConjunctionIMAC repeatedly calls a modified version of

Algorithm 2 FindBooleanConjunctionIMAC$(\mathbf{x^a}, X^-)$

```
$R \leftarrow \emptyset$; $S \leftarrow \emptyset$
for all $\mathbf{x}^- \in X^-$ do
    for all $f \in C_{\mathbf{x}^-} \cap R$ do
        toggle $f$ in $\mathbf{x}^-$
    end for
    while $c(\mathbf{x}^-) = -$ do
        $\mathbf{y} \leftarrow$ FindBooleanIMAC-R$(\mathbf{x^a}, \mathbf{x}^-, R)$
        add $\mathbf{y}$ to the solution set $S$
        add $C_y$ to the removed features set $R$
        for all $f \in C_{\mathbf{x}^-} \cap C_y$ do
            toggle $f$ in $\mathbf{x}^-$
        end for
    end while
end for
return the lowest-cost instance in $S$
```

FindBooleanIMAC to find negative instances that evade *some* component classifier with at most twice the optimal cost. The modified subroutine, FindBooleanIMAC-R, accepts a set of features R that must not be modified. As each negative instance is found, its features are added to R and removed from any initial negative instances in order to force the algorithm to find at least one instance that evades each component classifier. By taking the minimum cost of these, we obtain a cost that is at most twice optimal. We formalize these arguments in the proof of the following theorem.

Theorem 4. *With binary-valued features and Hamming distance costs, disjoint conjunctions of linear classifiers are ACRE 2-learnable when given a negative instance for each component.*

Proof. The minimum cost of evading the conjunction is the minimum cost of evading any one of its component classifiers. Thus, it suffices for the attacker to find a negative instance for each component with at most twice the minimum cost. The instance with smallest cost in this set must have at most twice the minimum overall cost.

We now prove that the set S constructed by FindBooleanConjunctionIMAC contains a 2-IMAC for each of the component classifiers. Consider the instance $\mathbf{y}$ returned from FindBooleanIMAC-R in one iteration of the while loop. Since the subroutine only returns negative instances, $\mathbf{y}$ must be negative in at least one classifier, c_i. Any changes in C_y that do not pertain to the ith classifier can be removed without affecting the label of $\mathbf{y}$. Since the subroutine removes all such changes before terminating, $C_y \subset \mathcal{F}_i$.

The only way to add features to R is for them to first be in some C_y. Since the features are disjoint, R will only contain features from $\mathcal{F}_i$ after finding some instance $\mathbf{y}$ that is negative in c_i and adding it to S.

If $c_i(\mathbf{y}) = -$ and $R \cap \mathcal{F}_i = \emptyset$, then FindBooleanIMAC-R considered all possible modifications relevant to c_i so $\mathbf{y}$ is a 2-IMAC for the linear classifier c_i. If $R \cap \mathcal{F}_i$ is non-empty, then at least one instance that is negative in c_i must already be in S. The first of these instances is a 2-IMAC, since $R \cap \mathcal{F}_i$ was empty when it was originally found.

It remains to be shown that we will find a 2-IMAC for every component. Since $\mathbf{x}^-$ has no changes in R and the subroutine is not allowed to add any change in R, C_y must have no change in R. C_y is non-empty and disjoint from R, so R

grows in each iteration. As long as some other classifier c_j remains for which there is no 2-IMAC in S, there must be some remaining $\mathbf{x}^- \in X^-$ such that $c_j(\mathbf{x}^-) = -$ and $R \cap \mathcal{F}_j$ remains empty. As R grows, the number of remaining features for other classifiers decreases until FindBooleanIMAC-R can no longer find negative instances for any other classifier and must therefore eventually find a 2-IMAC for c_j.

Therefore, FindBooleanConjunctionIMAC finds a 2-IMAC for every component, and thus returns a 2-IMAC for the overall conjunction. □

5. EXPERIMENTAL RESULTS

While we have shown that optimally evading disjoint conjunctions and disjunctions of classifiers can be approximated with a polynomial number of queries, we are also interested in how expensive this is in practice. Continuing the theme of email spam, we created classifiers trained on email data and had an adversary attempt to deceive these classifiers with minimal changes to their spam instances. The adversary may only interact with a classifier by querying it with an email instance to determine if it is classified as spam or non-spam. This could be done by creating two email accounts, sending messages from one to the other, and seeing how the message is classified when it is received. Rather than assuming that the adversary knows the entire feature space, we restrict the classifier to a plausibly guessable subset. We examine attacking both disjunction and conjunction classifiers, as well as baseline linear classifiers.

5.1 Classifier Configurations

All of our experiments were written in Python, using the Scikit-learn learning framework [15] to create and train classifiers. We trained Logistic Regression, Linear Support Vector Machine (SVM) and Naïve Bayes classifiers on the 2005 TREC public spam corpus [4], which is comprised of 92,189 labeled email messages. For simplicity, we restrict features considered to words that appear in at least ten documents and ignore common English stop words. To create disjoint linear classifiers to compose into our disjunction and conjunction classifiers, we split the feature space first three ways and then five ways to have classifiers composed of three or five linear classifiers. We achieve this split first by random partition and again by training a linear SVM classifier on all of the features and then using the value of the weights in the classifier's decision function to partition the features into sets ranging from the most "spammy" to most "hammy" features. This was partially inspired by Jorgensen et al. [9], who suggest using separate classifiers on the spammy, hammy, and neutral features in order to avoid "good word" attacks. These many combinations gave us a wide variety of classifiers for our experiments. For comparison purposes, we also train a linear classifier of each type using the entire feature set. Our methodology for training classifiers is intentionally simple, since our goal is to analyze the attack algorithms, not to develop the best spam filters.

To make the classifiers roughly comparable, we increase or decrease the bias of each component linear classifier by a constant value so that the false negative rate (fraction of spam classified as innocent) is 10%. This makes all of the classifiers equally effective at blocking spam, although their false positive rates (fraction of innocent email classified as spam) are different. Other previous work has used a fixed false positive rate [11, 10, 1]. We chose to fix the false negative rate rather than the false positive rate because we are interested in how attack difficulty depends on classifier type, not how it depends on the base effectiveness of each classifier.

	Best FP rate (%)			Mean FP rate (%)		
	LR	NB	SVM	LR	NB	SVM
Linear	0.04	0.18	0.07	0.04	0.18	0.07
Conj.	1.20	3.93	2.19	5.21	6.87	9.97
Disj.	2.90	5.41	2.84	4.85	12.90	6.75

Table 1: Best and geometric mean false positive rate for each base classifier type with each ensemble method.

See Table 1 for a basic comparison of the false positive rates of the different types of classifiers. Both of the ensemble methods performed worse than the simple linear classifier with all three classifier types, which suggests that these classifiers are not especially accurate on this domain. This is consistent with the results of Biggio et al. [2], who found that multiple classifier systems were usually less accurate than single classifiers. In our experiments, combinations of 3 classifiers were more accurate than 5, and partitioning features by their relative spamminess or hamminess was usually more accurate than the random partition.

5.2 Attack Configuration

Once our classifiers have been trained, we then run our adversarial algorithms as described above, as well as the FindBooleanIMAC algorithm. We perform the same optimizations used in [10] when performing the FindBooleanIMAC subroutine, except that we considered swapping all pairs of changes rather than only adjacent pairs. This is necessary with multiple classifiers because two consecutive changes in a list might pertain to different components. As an additional optimization for the conjunction attack, when we find an instance $\mathbf{y}$ that evades one of the component classifiers, we search for other features from the same component to add to R, which accelerates convergence.

Our adversarial and negative instance(s) are chosen randomly from the corpus. Because it is not realistic for an adversary to have complete knowledge of the classifier's feature space, we instead restrict the set of changeable features to one of three sets. The first is the set of all words in any of the ispell dictionaries (21,515 words). The other two are much smaller: the 1,000 most common words in the training data (excluding stop words) and 1,000 words selected randomly from the training data. While these technically require additional knowledge of the classifier, we believe these sets would be very easy to approximate using public email data or other text corpora. In addition to the features in the given set, the attacker is also allowed to include any feature present in the adversarial or negative instance(s).

In preliminary experiments, we found that the adversary's task was often much too easy: many spams could be disguised with a single change, and many others could be disguised with only two or three changes. When the optimal number of changes is some small constant k, then an optimal attack can be found by brute-force search in just $O(n^k)$ queries. It is only as the required number of changes grows larger that our polynomial-time algorithms are necessary. Therefore, to better analyze our algorithms, we artificially increase the "spamminess" of each email. We do this by changing 100 randomly-selected features, each of which makes the email look more like spam in one of the component classifiers.

We run the attack 100 times for each configuration of classifiers and features, resulting in around 9,200 runs overall.

5.3 Results

In this evaluation, we are primarily interested in three things: the vulnerability of each classifier, the cost of each attack attack relative to the optimal attack, and the overall complexity of the attacks.[2]

Vulnerability

Before discussing the performance of our attacks, we first examine how vulnerable these classification systems actually are. Figure 1 shows the optimal costs for defeating each classifier type. For example, Figure 1a shows the distribution of the optimal costs for defeating the classifiers composed of logistic regression classifiers, as well as the costs for defeating a single logistic regression classifier. As we may expect due to the similarity in the attack algorithm, the disjunction and linear attack tend to be similar in the size of the optimal disguise, with a very similar distribution in the Naïve Bayes case, and the linear classifier being easier to avoid in the other two cases. The actual distribution varies based on the type of linear classifier used, with Logistic Regression often not needing more than ten changes and Naïve Bayes needing 40 or more (and sometimes as many as 100). The SVM classifiers see the largest variance, and lie somewhere between the other two. These values are not surprising, as Logistic Regression and SVM tend to assign large weights to a few features, which the adversary can modify to more efficiently evade the classifiers. In sharp contrast to the Disjunction and Linear classifiers, the Conjunction classifier was defeatable with less than ten changes at least 90% of the time in every case. These results show while the disjunctive classifiers may sometimes be harder to defeat than the linear classifiers, the conjunctive ones tend to be much easier. Intuitively, this makes sense, as in the former case, the adversary must trick every linear classifier, while in the latter, a single defeat is sufficient. One of the variables under consideration was the way in which the features were partitioned between classifiers. However, both partitions we considered resulted in nearly identical distributions.

Approximation Quality

Our algorithms for attacking each classifier type is guaranteed to be within a factor of two of optimal. However, it is important to note that this is only if the exact feature space is known. If working with an approximated feature space, as we do in our experiments, the concept of "optimal" must be restricted to only consider the features available to the attacker. Figure 2 illustrates the relative costs of our attacks, both compared to the restricted vocabulary and the true optimal cost. As expected, we never go over a factor of two in the restricted case. More interestingly, over 90% of the time, the algorithms are able to get within a factor of two of the true optimal, despite only having an approximation of the feature space. It is sometimes as bad as four times optimal, but the frequency of costs decrease exponentially as they get higher.

[2]All figures in this section were created with the Matplotlib Python package [7].

To evaluate the dependence of the algorithm on the vocabulary used, we now split Figure 2b by vocabulary, as shown in in Figure 2c. As expected, the much larger dictionary set performs the best – almost always within a factor of 1.5. However, it is interesting to note that the random set is not much worse than the set of the top features, with all of the sets rarely going over a ratio of 2. This shows that a much smaller feature set can be selected with very little assumed information of the classifier without too much of a hit in approximation quality.

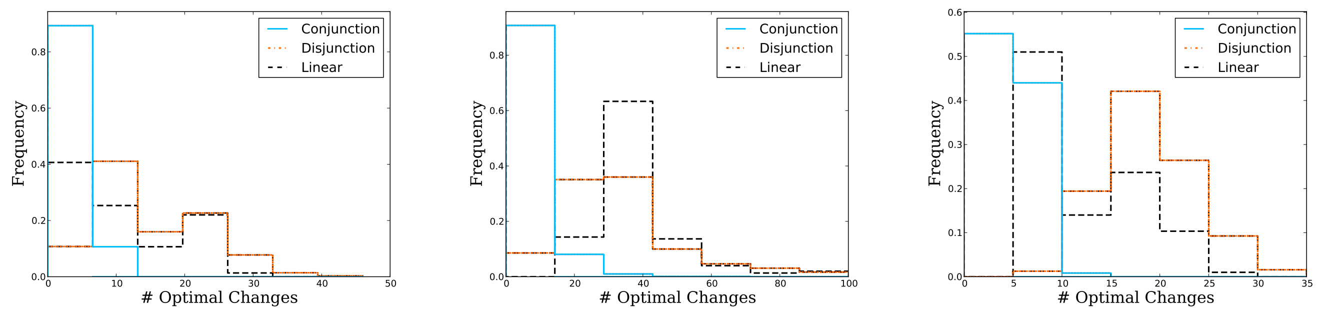
Complexity

The last major issue to evaluate is that of complexity. The bottleneck operation in most adversarial scenarios will be query time (e.g., the time it takes to send and email and have the mail server label it). As we just showed, a better approximation can be achieved by increasing the size of the feature set the adversary may consider. However, increasing this size will also lead to a larger number of queries. Therefore, we wish to evaluate the complexity of the attack by looking at the average number of queries as a function of the size of the considered feature set.

Figure 3 shows the number of queries as a function of the optimal number of changes (restricted to the given vocabulary). Again, we see that the conjunction classifier is easier to defeat than the others, while the disjunction behaves much like the linear classifier. Since these log-log plots display a nearly constant slope, this suggests that the number of queries is a polynomial function of the number of required changes. The dependence appears to be approximately linear for the conjunction and quadratic for the disjunction. Additionally, the full dictionary results only differ from that of the much smaller vocabularies by a constant factor of approximately 20. This suggests that the complexity only increases linearly with the size of the vocabulary used. Most of the time, using an entire dictionary will not be possible. However, in the previous section we showed that it is easy to guess a much smaller set that leads to very effective results. This linear relationship means that the attacker can easily find an attack that achieves the approximation quality they desire within an acceptable number of queries.

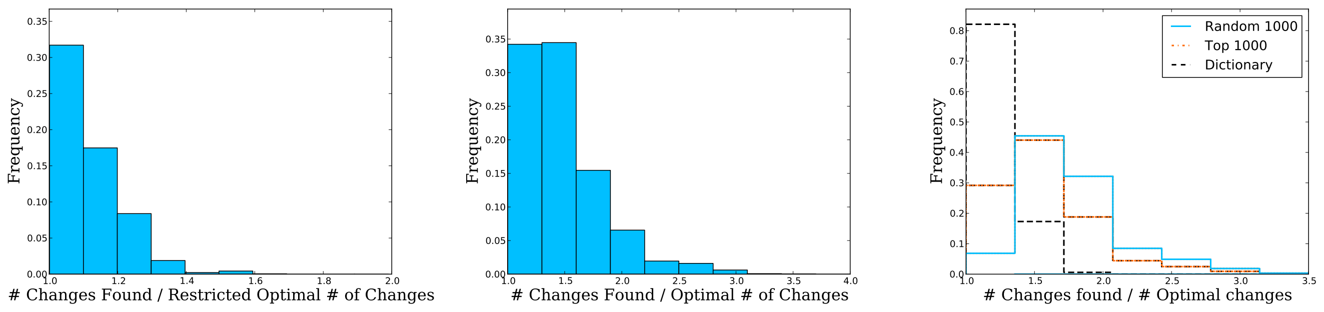
These relationships make sense based on the structure of the algorithm. Before converging, FindBooleanIMAC must verify that no possible change could replace any two existing changes in the current instance. The number of possible changes is approximately the number of features, and the number of existing changes is approximately the optimal cost, leading to a complexity of $O(nm^2)$ where m is the optimal cost.

The number of queries used might render these specific tasks impractical for a real-world adversary. However, our experimental procedure of adding extra spammy features to every email makes the problem artificially harder. On unmodified spam, the optimal cost is lower and so the number of queries is much smaller. Furthermore, these algorithms were designed to guarantee a nearly optimal attack against any classifier, which is a much more general problem than most adversaries face. In practice, an attacker is free to stop these algorithms early and use the best attack found so far. Many queries are spent in these algorithms *proving* that an attack is nearly optimal. Additionally, many queries might be spent with only slight improvement. Because of this, the number of queries can be significantly reduced if the algo-



(a) Logistic Regression (b) Naïve Bayes (c) SVM

Figure 1: Optimal cost of defeating each classifier type



(a) Optimal on Restricted Vocabulary (b) True Optimal (c) True Optimal, by Attack Vocabulary

Figure 2: Ratio of number changes found to optimal disguise.

rithm terminates as soon as an attack with an acceptable cost is found.

In Figure 4, we show the number of queries needed if the algorithm terminates as soon as an attack is found that is within a factor of two of the optimal achievable attack. Interestingly, the number of queries no longer exhibits a strong dependence on the number of optimal changes. Furthermore, the average number of queries is lowered by at least a factor of 100 in each case. This shows that the number of queries required to *find* a nearly optimal instance is just a small fraction of the queries required to *guarantee* a nearly optimal instance. Therefore, our algorithms are nearly as effective and vastly more efficient if stopped early, making these attacks much more practical.

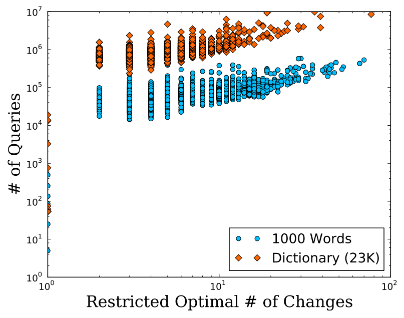
Finally, if the attacker has background knowledge about which features are likely to be spammy or innocent, then the order of the queries can be changed in order to find a good attack much faster. This background knowledge could come from educated guesses or from learning a similar model on publicly available data, as done by Biggio et al. [1]. Knowledge can also be gained by running our algorithms and observing which features are helpful or harmful; this could then be reused on future attacks against the same or similar classifiers. Lowd and Meek discuss even more efficient heuristics against linear classifiers [11], many of which could be adapted to combinations of linear classifiers. Any of these techniques could be used in conjunction with the early stopping method described above to construct very practical evasion attacks.

6. CONCLUSION

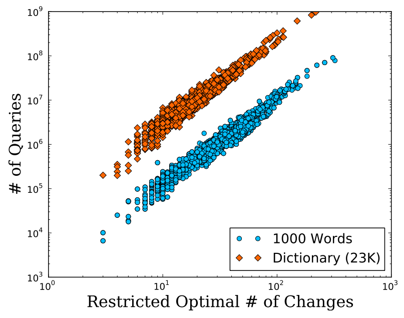
Understanding the theoretical hardness of attacking different types of classifiers is important for designing more robust systems in adversarial domains. In addition to measuring accuracy and efficiency, the hardness of evading a classifier can constitute another important dimension to consider. Previous results had shown that an attacker could find a negative instance to evade any linear classifier with at most twice the minimum Hamming distance using only a polynomial number of membership queries.

In this paper, we presented the first results for evading non-linear classifiers in discrete feature spaces. We found that arbitrary conjunctions and disjunctions of linear classifiers require exponentially many queries to even approximate minimum Hamming distance. This suggests that they may be more robust than individual linear classifiers. However, when the component classifiers operate on disjoint sets of features, we proved that these ensembles of linear classifiers can still be attacked relatively efficiently.

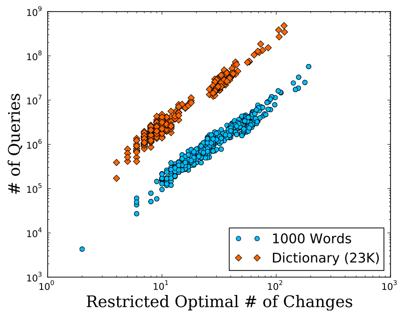
We also showed the relative practicality of these attacks using combinations of classifiers trained on email spam. The number of queries required depends linearly on the number of features considered and quadratically on the optimal cost. We found smaller vocabularies to be nearly as effective as larger vocabularies while requiring fewer queries. Additionally, simply stopping the algorithms early can find nearly optimal attacks while reducing the number of queries by two orders of magnitude.



(a) Conjunction

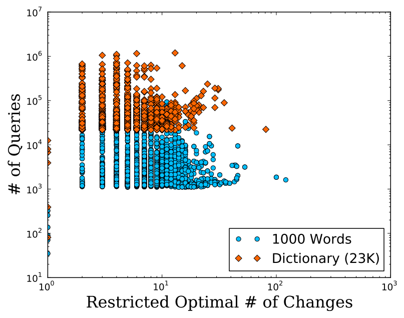


(b) Disjunction

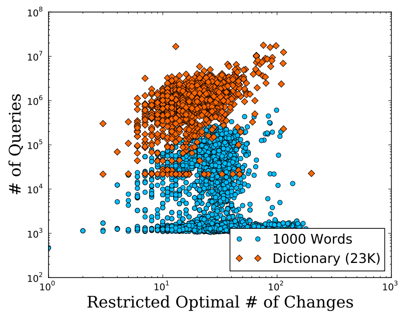


(c) Linear

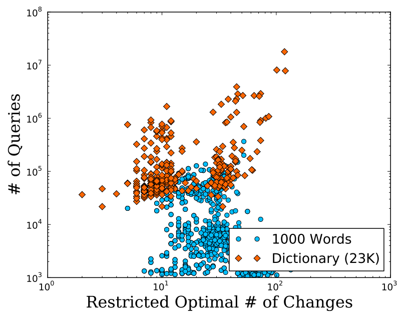
Figure 3: Number of queries as a function of the restricted optimal number of changes (log-log plot)



(a) Conjunction



(b) Disjunction



(c) Linear

Figure 4: Number of queries needed to be within a factor of 2 of the restricted optimal number of changes

It is important to remember that our results are only upper bounds on the difficulty of real-world classifiers. In practice, any given classifier may have properties that make it much easier to attack than the hardest classifier in the concept class. Furthermore, if the adversary has additional background knowledge or does not need to guarantee approximate optimality, much more efficient attacks may be possible. Thus, these results provide insight and guidance but do not constitute a complete evaluation of a classifier's security.

In ongoing work, we are investigating other criteria that may make conjunctions and disjunctions of linear classifiers easy to attack. For example, bounds on the number of components or the relative weights of the features may make efficient attacks possible even when the features overlap.

7. ACKNOWLEDGMENTS

This research was partly funded by ARO grant W911NF-08-1-0242. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ARO, NSF, or the U.S. Government.

8. REFERENCES

[1] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli. Evasion attacks against machine learning at test time. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*. Springer, 2013.

[2] B. Biggio, G. Fumera, and F. Roli. Multiple classifier systems for adversarial classification tasks. In *Proceedings of the 8th International Workshop on Multiple Classifier Systems*, pages 132–141. Springer, 2009.

[3] D. Chau, S. Pandit, and C. Faloutsos. Detecting fraudulent personalities in networks of online auctioneers. *Knowledge Discovery in Databases: PKDD 2006*, pages 103–114, 2006.

[4] G. Cormack and T. Lynam. Spam corpus creation for trec. In *Stanford University*, 2005.

[5] L. F. Cranor and B. A. LaMacchia. Spam! *Communications of the ACM*, 41(8):74–83, August 1998.

[6] I. Drost and T. Scheffer. Thwarting the nigritude ultramarine: Learning to identify link spam. In *Proceedings of the Sixteenth European Conference on Machine Learning*, pages 96–107. Springer, 2005.

[7] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[8] N. Jindal and B. Liu. Opinion spam and analysis. In *Proceedings of the International Conference on Web Search and Web Data Mining*, pages 219–230. ACM, 2008.

[9] Z. Jorgensen, Y. Zhou, and M. Inge. A multiple instance learning strategy for combating good word

attacks on spam filters. *J. Mach. Learn. Res.*, 9:1115–1146, June 2008.

[10] D. Lowd and C. Meek. Adversarial learning. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 641–647, New York, NY, USA, 2005. ACM.

[11] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *Proceedings of the Second Conference on Email and Anti-Spam (CEAS)*, pages 125–132, 2005.

[12] B. Nelson, B. I. P. Rubinstein, L. Huang, A. D. Joseph, S. J. Lee, S. Rao, and J. D. Tygar. Query strategies for evading convex-inducing classifiers. *CoRR*, abs/1007.0484, 2010.

[13] J. Neville, O. Şimşek, D. Jensen, J. Komoroske, K. Palmer, and H. Goldberg. Using relational knowledge discovery to prevent securities fraud. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 449–458. ACM, 2005.

[14] D. O'Callaghan, M. Harrigan, J. Carthy, and P. Cunningham. Network analysis of recurring YouTube spam campaigns. In *Proceedings of the Sixth International AAAI Conference on Weblogs and Social Media*. AAAI Press, 2012.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] D. Sculley, M. E. Otey, M. Pohl, B. Spitznagel, J. Hainsworth, and Y. Zhou. Detecting adversarial advertisements in the wild. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 274–282. ACM, 2011.

[17] G. Stringhini, C. Kruegel, and G. Vigna. Detecting spammers on social networks. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 1–9. ACM, 2010.

[18] S. Yardi, D. Romero, G. Grant, and d. boyd. Detecting spam in a Twitter network. *First Monday*, 15(1), 2010.