Automated Detection of Automated Traffic

Cormac Herley
Microsoft Research

Abstract

We describe a method to separate abuse from legitimate traffic when we have categorical features and no labels are available. Our approach hinges on the observation that, if we could locate them, unattacked bins of a categorical feature x would allow us to estimate the benign distribution of any feature that is independent of x. We give an algorithm that finds these unattacked bins (if they exist) and show how to build an overall classifier that is suitable for very large data volumes and high levels of abuse. The approach is one-sided: our only significant assumptions about abuse are the existence of unattacked bins, and that distributions of abuse traffic do not precisely match those of benign.

We evaluate on two datasets: 3 million requests from a web-server dataset and a collection of 5.1 million Twitter accounts crawled using the public API. The results confirm that the approach is successful at identifying clusters of automated behaviors. On both problems we easily outperform unsupervised methods such as Isolation Forests, and have comparable performance to Botometer on the Twitter dataset.

1 Introduction

Abuse is ubiquitous on the modern web. Services intended for human use must often contend with large volumes of bot traffic which consume resources and degrade the quality of experience delivered to legitimate users. This is especially true if there is a strategy that allows monetization of the abuse. Spam is an obvious example, but there are numerous others. Password-guessing, web-scraping, CAPTCHA-solving, automated signup requests, inauthentic social-media engagements and ranking- and click-fraud are examples of the large-scale automated abuse that online attackers engage in.

Service providers thus have a difficult task in maintaining good quality of service: they wish to block as much bot traffic as possible with minimum inconvenience to legitimate human users. None of the available solutions are without problems. CAPTCHAs, or tasks intended to be easy for humans but hard for bots [3, 10], are widely employed but have had mixed results. They cause considerable friction to users and there has been significant success in breaking them in an automated way [9] [8]. Attaching reputation to IP addresses that engage in abuse has achieved considerable success [35], but implicitly assumes that abuse comes from a limited pool of IP addresses and that this pool does not significantly overlap that used by legitimate users; further it assumes this lack of overlap will persist during deployment. Providers with tens or hundreds of millions of users attacked from botnets with millions of consumer IP addresses are unlikely to be able to rely on reputation for reliable decisions. Supervised machine learning approaches are often inapplicable since for many abuse problems there is no possibility of obtaining labels. Indeed, some authors who have tried supervised ML approaches reach pessimistic conclusions on its suitability for bot detection; e.g, Jan et al [28] write *"We argue that [a] rule-based system should be the first choice over machine learning for bot detection."* Anomaly detection approaches are often geared towards discovering outliers that are "few and isolated" [12]. By contrast, the abuse types listed above can sometimes be 10 - 90% of overall traffic. Thus, there is real difficulty applying anomaly detection techniques to this problem. Unsupervised learning approaches (e.g., one-class SVMs [32] and Isolation Forests [31]) usually assume numeric features, and often struggle when we one-hot encode categorical features.

In this paper we introduce a new method to estimate the likelihood that incoming traffic requests are from bots. The approach is designed for very noisy environments, e.g., where 50% or more of traffic is abuse. Our approach hinges on the observation that if we can identify buckets of traffic that contain little or no abuse then (under certain independence assumptions) we can estimate the clean distributions. For example, if none of the abuse traffic from Iowa uses Firefox then $P(x|\text{Iowa, Firefox})$ gives us the clean distribution of x for any feature that is independent of both state and browser family. Obviously, we must be careful about independence assumptions, and finding unattacked buckets is non-trivial, but if we can do these two things we will have the core of what we need.

Our contributions are as follows. We give a method to estimate the likelihood that a request is bot rather than human generated. The method is entirely unsupervised and requires no labels. It is one-sided, and attempts to estimate only the distributions of features in the benign[1] traffic. Hence the assumptions about attack traffic are mild; in particular we do not assume stationarity of attack traffic, we do not assume that abuse patterns seen in training will continue to be seen in deployment, and we do not assume a scarcity of any particular resource, such as IP addresses, available to the attacker. Our main attack assumptions are that at least some bins of at least some categorical features receive little or no attack traffic, and that at least some of the abuse distributions differ from the benign (e.g., the benign distribution are unknown or unachievable to the attacker). We show how these relatively unattacked bins can be identified, how they can be used to estimate feature distributions in the clean traffic, and hence calculate the odds of being bot for any request. This method is robust even if 90% or more of the traffic is abuse. The technique requires large amounts of data and may be more suitable to large rather than small or medium-scale services. We evaluate on a web-log dataset and a collection of 5.1 million Twitter accounts crawled between January and May 2021. We call the algorithm PROS after the central mechanism of its implementation: Pivot and seek Rank-One Sub-matrix.

2 Design principles

Our design goals and principles are as follows:

1. **Unsupervised:** for many abuse problems labels are unavailable and there is no way of establishing ground truth.

2. **One-sided:** We estimate only the clean distributions. We do not assume that patterns observed in attack behavior will persist, or that the attack traffic seen in training is representative of deployment.

3. **Categorical features:** much of the information collected from a browser is categorical making it difficult to use algorithms that assume real-valued features.

4. **Robust to base-rate uncertainty and variation:** the approach should work if the base-rate of malicious traffic is 10% of total, 90% of total, or anything in between. It should be robust to variations in the base-rate.

5. **Interpretable:** Ideally, human operators should be able to understand classification decisions.

[1]Note: we use the terms benign and malicious to refer to scripted and non-scripted traffic, even when scripted traffic presents no harm.

We next explain our reasoning in choosing these principles.

Unsupervised: Many abuse problems are inherently unsupervised. Web traffic that is a mixture of malicious and benign offers no real way to establish ground truth. This is not simply a question of cost or resources; even if we had unlimited budget for human-labelling, the information available at the time of the request is not enough for a human to distinguish bot from legitimate requests. Chio and Freeman summarize [15]:

> *Unlike spam, which can be given to a human to evaluate, there is no reasonable way to present an individual request to a reviewer and have that person label the request as bot or not.*

For many problems labelling even a small portion of the data is generally not possible. This property is common to a large number of abuse types (e.g., scraping, click-fraud, password guessing, credential stuffing). A recent paper by Xu et al [39] describing Facebook's abusive account detection acknowledges the absence of ground-truth labels and notes that it relies on "the results of rule-based heuristics as additional 'approximate labels."'

One-sided: Our detection method is a one-sided, or one-class classifier, meaning that we estimate only the clean distribution. We make no effort to identify or codify patterns of malicious behavior. It may be that one attacker sends a million requests per day using Chrome59.0.3451, or from a narrow block of IP addresses, or at a constant rate instead of the expected diurnal variation. We will not seek out these patterns; rather they will emerge naturally when distributions in the abuse traffic differs from those of the benign traffic. In fact exploiting any such patterns, if they exist, assumes that what is seen in training is representative of deployment. Avoiding this assumption means our approach is robust to changes in the malicious traffic: it is deviation from the estimated clean distribution that causes traffic to be marked as abusive rather than any heuristic as to what counts as suspicious or assumptions about the persistence of observed abusive behavior.

Categorical features: For many abuse problems the available features are categorical. This is certainly true for problems involving web-traffic (see Section 6), but also for many others. This restricts our options considerably since many ML and anomaly detection algorithms work with numeric features [12, 25]. Unsupervised ML methods often rely on the assumption that observations that are close in feature space are more likely to belong to the same class. When we have no natural idea of distance, as for categorical features, these methods are hard to apply. Mapping categorical features to numeric (e.g., using one-hot encoding) doesn't help much since, along each feature, observations are still either a distance of 0 or 1 apart. Other attempts to define a distance are also problematic. For example, traffic from 'Chrome77.0.3770' (a major release of the browser) might be predominantly benign while 'Chrome77.0.3771' (a non-released made-up version) might be 100% malicious; the closeness of the strings tells us nothing about likelihood of belonging to the same class.

Our method is close in spirit to many anomaly detection (AD) approaches; in AD it is also common to estimate the clean distribution and punish deviations. In spite of this similarity it is difficult to leverage existing AD techniques without modification. Many AD approaches rely on distance or density methods. The underlying assumption is that things close in feature space have high likelihood of being in the same class. This is true for K-means, K Nearest Neighbors, Isolation Forests [31], One-Class SVMs [32], etc. We review related anomaly detection work in Section 3.

Base-rate uncertainty: In abuse problems we seldom have a strong prior about the fraction of received traffic that is from bots. It's hard to say a priori whether sign-up abuse, or password guessing, or CAPTCHA-solving attempts are 5%, 50% or even 90% of total. For very large providers some forms of abuse presumably never entirely go away, but the amount might vary enormously over time as attackers gain or lose resources, change strategy, or alter their targets. Rules with fixed thresholds are likely to trade true for false positives at very different rates depending on the base-rate.

A related problem is that the base-rate can vary between training and deployment. Many supervised approaches assume that the ratio of benign to malicious seen in deployment is the same as seen in training [22] (and have performance that deteriorates rapidly if it is not [29]). Given the high-flux nature of abuse traffic this appears a problematic assumption. We seek an approach that works well across a wide range of base-rates.

Interpretability: We wish for the classification decisions to be explainable to a human operator. This is particularly important for unsupervised approaches where the absence of ground-truth labels means that traditional true and false positive metrics, Receiver Operator Characteristic (ROC) curves and Area Under Curve (AUC) metrics are unavailable. Human-interpretable decisions also clearly helps with maintainability and debugging.

Our approach outputs human-interpretable rules that can be very useful in identifying clusters of bot activity. If a bot sends significant traffic from Florida using IE9 then a rulset based on features state and browser will have a rule "Florida and IE9" near the top when sorted by bot likelihood.

3 Related work

Bot and scripted traffic have long posed problems for web services. Dwork and Naor [10] were perhaps the first to recognise the issue and propose a 'proof of work' to deter excessive use of free or unmetered services. CAPTCHAs are one approach [3] and most large web services use them in one form or other. Most CAPTCHAs have been shown to be breakable [8], and they do not appear to offer a durable robust defense. They also, of course, place a significant burden on the user population. Douceur first describes the problem that automated creation of pseudonymous account are for web services [19]; the magnitude of the problem has ballooned in the two decades since.

Sommer and Paxson [33] offer an excellent summary of the issues involved in applying ML to security problems. Among the issues that they highlight are lack of labels, and the absence of standardized datasets. Their emphasis is on intrusion detection, but both of these are certainly factors for abuse problems. Chio and Freeman [15] give a more recent overview of ML applications in security including abuse problems such as account creation and hijacking. They suggest clustering approaches which can detect large-scale coordination. They suggest mechanisms such as locality sensitive hashing as a measure of closeness for objects such as domain names which may be related but not identical.

An excellent survey of anomaly detection (AD) techniques is given by Chandola et al [12]. The common approach in anomaly detection is to estimate $P(\boldsymbol{x}|\overline{\text{bot}})$ in a 'bake-in' period. That is, it is expected that anomalies are rare, so that we can estimate the distribution using any available window and be confident that it approximates the true distribution. This often does not work for abuse problems since we cannot count on any particular time period being attack-free. Nonetheless our approach has much in common with AD techniques: unattacked bins play the role that a 'bake-in' period plays in many AD methods. Our approach is similar, in principle, to Herley and Schechter's method to detect password brute-forcing attacks [26]. They observe that the ratio of fails to successful logins acts as a proxy measure for the amount of attack traffic, and use this to identify unattacked subsets. Emmott et al [20] describe an approach to benchmarking AD algorithms, but the emphasis is on anomalies that are rare.

Important unsupervised learning approaches include Isolation Forests [31], One-Class SVMs [32] and clustering approaches, such as K-means and variants [27]. Each of these methods assumes numeric features and do not adapt uneasily to categorical features where there is no natural measure of distance between data points. That is, most unsupervised methods assume that closeness in feature space implies greater likelihood of belonging to the same class; this breaks down when we have categorical features (and thus no natural idea of distance). One-hot encoding is the common way of mapping categorical features to numeric; however, when the number of features is small we end up with only a limited number of discrete distances possible between samples; this greatly compromises performance.

There have been a number of approaches to detecting bot behavior and traffic. Ferrara et al [21] describe content, network and temporal characteristics of Twitter bots and give a review of detection mechanisms. Botometer (formerly Bot-or-not) [17] is a project at Indiana University that estimates bots based on account activity. When accounts have very little activity Botometer has no basis to make a decision. The public

API that they expose is our main baseline for comparison in Section 7.

Thomas et al [37] describe the thriving underground market for Twitter spam and abuse accounts. They use the accounts they discover being traded to train a supervised detection algorithm. Chavoshi et al [13] describe detecting Twitter clusters of bots by detecting highly synchronized activity. The idea is that managing a large collection of sybil accounts will often result in high temporal correlation in the activity of those accounts.

Kudugunta and Ferrara [30] describe a DNN approach to social bot detection. They use synthetic minority oversampling [14] which generates a large amount of synthetic labelled data from a smaller number of actual labels. Jan et al [28] also leverage a small amount of labelled data to synthesize quantities that can be used by neural networks. They echo some of findings of Sommer and Paxson [33] on the limitations of ML methods.

Given the importance of the problem there is a significant body of work on detecting social media bots. We should be clear that we do not claim to rival the performance of state-of-the-art detection: our testing on a Twitter dataset is driven by availability rather than an expectation that it is particularly suited to this domain. Grier et al do a large scale characterization of Twitter spam [24]. They find Twitter spammers achieve much higher clickthrough rates than on other platforms, and high-levels of use of URL shortening services. Benevenuto et al [5] describe detecting spammers on Twitter. A portion of accounts in a large dataset were manually labelled as spammer/not-spammer as a front-end to a supervised learning approach. Stringhini et al [34] describe a honeypot approach to the detection of spammers on social networks. By creating hundreds of honey accounts they were able to detect large volume of spam activity. Cao et al describe detecting clusters of malicious accounts on Facebook [11]. They observe that malicious accounts under a single attacker's control often perform actions that are synchronized or correlated in time. Xiao et al describe detecting clusters of fake accounts on LinkedIn [38]. The approach is supervised using manually generated labels on a sample of accounts. Stringhini et al [36] present a measurement study of the market for Twitter followers. The approach leverages the labels to find large clusters of accounts that share characteristics (e.g., naming patterns etc). Stringhini et al [35] describe how to detect clusters of related malicious accounts by observing that the contact points (or IP addresses) used to access them are often shared. Xu et al describe an approach to detecting abusive accounts at Facebook using a small number of human-labelled samples augmented by lower-precision automated labels [39].

4 Sketch of method

The traffic we observe is a mixture of malicious and benign. Let $\boldsymbol{x} = (x_0, x_1, x_2, \cdots, x_{M-1})$ be a vector of observed features. For example, these might be time, browser version, IP address, country, city, state, etc. We'll assume that the features of interest are categorical or ordinal (this is the case for most measurements available from a browser connection), so that feature x_j can take any of N_j different values. Time might be quantized to hour-of-day so that x_i takes one of 24 discrete values, browser version might have 60 or so buckets for the most common versions and one bucket for everything else.

The traffic observed, $\mathrm{O}(\boldsymbol{x})$, is a mixture of clean, $\mathrm{C}(\boldsymbol{x})$ and bot, $\mathrm{B}(\boldsymbol{x})$:

$$\mathrm{O}(\boldsymbol{x}) = \mathrm{C}(\boldsymbol{x}) + \mathrm{B}(\boldsymbol{x}).$$

We'll find it convenient to deal with distributions, which we derive by normalizing the histograms:

$$P(\boldsymbol{x}) = \frac{\mathrm{O}(\boldsymbol{x})}{|\mathrm{O}|}, \quad P(\boldsymbol{x}|\overline{\mathrm{bot}}) = \frac{\mathrm{C}(\boldsymbol{x})}{|\mathrm{C}|}, \quad P(\boldsymbol{x}|\mathrm{bot}) = \frac{\mathrm{B}(\boldsymbol{x})}{|\mathrm{B}|}.$$

The fraction of traffic that is clean is:

$$\alpha = \frac{|\mathrm{C}|}{|\mathrm{C}| + |\mathrm{B}|}. \tag{1}$$

Thus, the received distribution is:

$$P(\boldsymbol{x}) = \alpha \cdot P(\boldsymbol{x}|\overline{\mathrm{bot}}) + (1-\alpha) \cdot P(\boldsymbol{x}|\mathrm{bot}), \tag{2}$$

where $0 < \alpha \leq 1$. Obviously, $P(\boldsymbol{x}|\overline{\mathrm{bot}})$, $P(\boldsymbol{x}|\mathrm{bot})$ and α are all unknown.

Bot traffic might come from a single attacker, or from many. In general we should have

$$P(\boldsymbol{x}|\mathrm{bot}) = \sum_{q=0}^{Q-1} w_q \cdot P(\boldsymbol{x}|\mathrm{bot}_q), \tag{3}$$

where Q is the number of distinct attackers and the w_q are the weights of their contributions. If $Q = 1$ (i.e., there's only one attacker) then it might be plausible that the x_i are independent in the bot traffic. This might be the case, for example, if the attacker simply ranges over several possible bins for each feature using a `for`-loop. If $Q > 1$ (i.e., there are several attackers) it's extremely unlikely that the x_i are independent in the bot traffic. This is the case, since the sum of separable functions is not in general separable.

Suppose we assume that the x_i are independent in the benign traffic[2], i.e., $P(\boldsymbol{x}|\overline{\mathrm{bot}}) = \prod_i P(x_i|\overline{\mathrm{bot}})$. This simply says that, for example, browser version does not depend on time-of-day, etc. This gives that the overall observed distribution is:

$$P(\boldsymbol{x}) = \alpha \cdot \prod_i P(x_i|\overline{\mathrm{bot}}) + (1-\alpha) \cdot P(\boldsymbol{x}|\mathrm{bot}),$$

and the marginal observed distribution along x_j:

$$P(x_j) = \alpha \cdot P(x_j|\overline{\mathrm{bot}}) + (1-\alpha) \cdot P(x_j|\mathrm{bot}).$$

[2]Note: we will actually only assume conditional independence over subsets of the data, see Section 5. We do not assume the same thing for the malicious traffic. This is strictly weaker than the independence assumption made in many supervised algorithms.

We can also calculate the marginal distribution restricted to subsets of the data; for example restricting to a particular bucket of one feature $x_k = x_k^{'}$ we get:

$$P(x_j|x_k^{'}) = \alpha_k^{'} \cdot P(x_j|\overline{\text{bot}}, x_k^{'}) + (1-\alpha_k^{'}) \cdot P(x_j|\text{bot}, x_k^{'}),$$

where $\alpha_k^{'}$ is the (unknown) fraction of traffic in this bucket that is benign:

$$\alpha_k^{'} = \frac{|\mathrm{C}(\boldsymbol{x}_j|x_k^{'})|}{|\mathrm{C}(\boldsymbol{x}_j|x_k^{'})| + |\mathrm{B}(\boldsymbol{x}_j|x_k^{'})|}.$$

Note that (because of assumed independence) $P(x_j|\overline{\text{bot}}, x_k^{'}) = P(x_j|\overline{\text{bot}}, x_k^{''}) = P(x_j|\overline{\text{bot}})$.

Now suppose that this bucket $x_k = x_k^{'}$ receives no attack traffic (e.g., there's no attack traffic in the browser = 'Firefox67' or state = 'Iowa' buckets, etc). This gives us that $\alpha_k^{'} = 1$, $P(x_j|\text{bot}, x_k^{'}) = 0$ and hence:

$$P(x_j|x_k^{'}) = P(x_j|\overline{\text{bot}}), \quad j \neq k. \tag{4}$$

In words: any unattacked bucket $x_k = x_k^{'}$ of the k-th feature tells us the benign distribution for every other feature x_j (for $j \neq k$) that is independent of x_k in the benign traffic.

Armed with the clean distribution, the problem is now simple. The odds that an observation is malicious can be expressed:

$$\begin{aligned}\frac{P(\text{bot}|\boldsymbol{x})}{P(\overline{\text{bot}}|\boldsymbol{x})} &= \frac{P(\boldsymbol{x}) - \alpha \cdot P(\boldsymbol{x}|\overline{\text{bot}})}{\alpha \cdot P(\boldsymbol{x}|\overline{\text{bot}})} \\ &= \frac{P(\boldsymbol{x}) - \alpha \cdot \prod_i P(x_i|\overline{\text{bot}})}{\alpha \cdot \prod_i P(x_i|\overline{\text{bot}})}.\end{aligned} \tag{5}$$

From (4) we can get the clean distributions $P(x_i|\overline{\text{bot}})$ needed to calculate this odds if we can find unattacked buckets in at least two features.

This would leave α as the sole remaining unknown parameter. Note that (5) is monotonic in α, so that the order is not affected by α. To see this, denote as $\text{Odds}(\boldsymbol{x}, \alpha)$ the left-hand side of (5) and observe that if $\text{Odds}(\boldsymbol{x}, \alpha) > \text{Odds}(\boldsymbol{y}, \alpha)$ then

$$\frac{P(\boldsymbol{x})}{\alpha \cdot P(\boldsymbol{x}|\overline{\text{bot}})} - 1 > \frac{P(\boldsymbol{y})}{\alpha \cdot P(\boldsymbol{y}|\overline{\text{bot}})} - 1$$

which simplifies to

$$P(\boldsymbol{x}) \cdot P(\boldsymbol{y}|\overline{\text{bot}}) > P(\boldsymbol{y}) \cdot P(\boldsymbol{x}|\overline{\text{bot}}),$$

which is independent of α. Thus, if $\text{Odds}(\boldsymbol{x}, \alpha) > \text{Odds}(\boldsymbol{y}, \alpha)$ is true for any α it is true for all α. In other words if we fix α then (5) will give us a ranking of observations from most to least suspicious; if we're wrong about α we will still be right about the ordering.

4.1 Toy example

A toy example in words may help clarify the idea. Suppose, for US traffic, we examine only three features: browser version (e.g., 'Chrome77', 'Firefox69', 'Edge17', etc), state (i.e., 'Alabama', 'Alaska', etc) and local hour-of-day (i.e., 0 to 23). We might have 60 buckets for the most common browser versions (throwing everything else into a catchall 'isRest' bucket). Thus, there are $60 \times 50 \times 24$ possible 3-tuples and every request is quantized into one of them. The assumption of independence is that (in the benign traffic):

$$P(\text{br, st, hr}|\overline{\text{bot}}) = P(\text{br}|\overline{\text{bot}}) \cdot P(\text{st}|\overline{\text{bot}}) \cdot P(\text{hr}|\overline{\text{bot}}).$$

So that, for example,

$$P(\text{hr}|\overline{\text{bot}}, \text{'Chrome77'}) \approx P(\text{hr}|\overline{\text{bot}}, \text{'Firefox69'});$$

i.e., the time-of-day distribution doesn't differ by browser choice in the benign traffic. Equally,

$$P(\text{br}|\overline{\text{bot}}, \text{'Michigan'}) \approx P(\text{br}|\overline{\text{bot}}, \text{'Georgia'});$$

i.e., the benign browser distribution doesn't differ by location[3].

Now suppose that there is no attack traffic from, e.g., Iowa. From (4) this means that we can calculate

$$P(\text{br}|\overline{\text{bot}}) = P(\text{br}|\text{'Iowa'})$$

and

$$P(\text{hr}|\overline{\text{bot}}) = P(\text{hr}|\text{'Iowa'}).$$

Also suppose there is no attack traffic from 'IE9'; this gives

$$P(\text{st}|\overline{\text{bot}}) = P(\text{st}|\text{'IE9'})$$

and

$$P(\text{hr}|\overline{\text{bot}}) = P(\text{hr}|\text{'IE9'}).$$

This means that, given an observation $\boldsymbol{x} = (\text{br, st, hr})$, the odds of being malicious are (from (5)):

$$\frac{P(\text{br, st, hr})}{\alpha \cdot P(\text{br}|\text{'Iowa'}) \cdot P(\text{st}|\text{'IE9'}) \cdot P(\text{hr}|\text{'Iowa'})} - 1.$$

That is, except for α, we have everything we need to calculate the odds. Since we saw that the order of odds ratio was independent of α any fixed α gives a ranking from most to least likely to be malicious. All that we needed to assume was independence of the features in the benign traffic, and that we had unattacked buckets in some of the features.

[3]Note: we will relax any assumption of global independence in Section 5 below.

5 Detailed Description

Independence as described in Section 4 is an idealization. In practice we must expect many features will have some dependence on other features. Firefox might represent 15% of the market in France, but only 2% in Mexico, for example, indicating a dependence between browser family and country. It is unrealistic to expect complete mutual independence of all of the features that might be valuable in making a classification (although supervised approaches often assume this of both benign and malicious traffic). In addition to the three features used in the toy example of Section 4.1 we will want to consider others. Information available from the browser such as languages, fonts and plugins installed, protocols available, etc can all be valuable. Web services that involve user-provided input (e.g., form-filling) will have valuable textbox inputs; e.g., an account signup page will have a user-chosen username.

That independence is a problematic assumption is probably especially true of a global service. The larger the population of users the more diverse the traffic is likely to be. Browser-language is more likely to be independent of browser family at the web portal of a small regional bank than at Facebook. Independence means that the multivariate distribution is the product of 1-dimensional distributions. The larger and more diverse the traffic population the harder this is to guarantee. However, features that might not be globally independent might be locally independent subject to some condition. For example, browser version might not be independent of hour-of-day globally, but might be within particular countries.

There is formal support for the view that we can get conditional independence if we break a large dataset into smaller subsets. Formally, if $P(\boldsymbol{x}) = \prod_k P(x_k)$ then it is a rank-1 tensor (i.e., it is the product of 1-dimensional factors). However, every matrix or tensor can be written as a sum of rank-1 components [7, 23]. The rank of a tensor is the minimum number of rank-1 tensors required to express it in a sum. Instead of assuming global independence of features we will break into subsets (e.g., country, or browser family) over which it is more reasonable to assume conditional independence.

Let's break the data into subsets $\boldsymbol{S}_i$ whose union covers all traffic: $\cup_i \boldsymbol{S}_i = I$. The $\boldsymbol{S}_i$ might be countries, or blocks of IP addresses, or browser families, or intersections of these things. For example, the $\boldsymbol{S}_i$ might be the intersection of country and browser family, so that Mexico-Chrome, France-Firefox, Vietnam-Edge, etc are instances. We will use a set of features $\boldsymbol{x}$ that are mutually independent conditioned on $\boldsymbol{S}_i$:

$$P(x_j, x_k|\overline{\text{bot}}, \boldsymbol{S}_i) = P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i) \cdot P(x_k|\overline{\text{bot}}, \boldsymbol{S}_i) \quad \text{for } x_k \neq x_j.$$

Given a set of features $\boldsymbol{\lambda}(x_j)$, we'll say that x_j is independent of features in $\boldsymbol{\lambda}(x_j)$ conditioned on $\boldsymbol{S}_i$ (e.g., hour-of-day is independent of State given Mexico-Chrome) if for $x_k \in \boldsymbol{\lambda}(x_j)$:

$$P(x_j, x_k|\overline{\text{bot}}, \boldsymbol{S}_i) = P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i) \cdot P(x_k|\overline{\text{bot}}, \boldsymbol{S}_i).$$

	Description
x_k	Categorical feature
N_k	Cardinality of x_k
$\boldsymbol{x}$	Vector of features, e.g., $(x_0, x_1, x_2, \cdots, x_{M-1})$
$\boldsymbol{S}_i$	Subset of the data; e.g., a single country
$P(\boldsymbol{x})$	Observed distribution of $\boldsymbol{x}$
$P(\boldsymbol{x}\|\overline{\text{bot}})$	Distribution of $\boldsymbol{x}$ in benign traffic
$P(\boldsymbol{x}\|\text{bot})$	Distribution of $\boldsymbol{x}$ in malicious traffic
$\hat{P}(\boldsymbol{x}\|\overline{\text{bot}})$	Estimate of $P(\boldsymbol{x}\|\overline{\text{bot}})$
α	Fraction of traffic that is benign
$\boldsymbol{\lambda}(x_j)$	Set of features conditionally independent of $x_j\|\boldsymbol{S}_i$
$\boldsymbol{\mu}(\boldsymbol{\lambda}(x_j))$	Set of unattacked buckets of features in $\boldsymbol{\lambda}(x_j)$

Table 1: Explanation of variables used.

We'll denote this $x_j \perp \boldsymbol{\lambda}(x_j)|\boldsymbol{S}_i$. This does not imply that features in the set $\boldsymbol{\lambda}(x_j)$ are mutually independent; in general they are not.

Obviously, the received distribution is a mixture of benign and bot, and the benign distribution can be factored:

$$\begin{aligned} P(\boldsymbol{x}|\boldsymbol{S}_i) &= \alpha_i \cdot P(\boldsymbol{x}|\overline{\text{bot}}, \boldsymbol{S}_i) + (1-\alpha_i) \cdot P(\boldsymbol{x}|\text{bot}, \boldsymbol{S}_i) \\ &= \alpha_i \cdot \prod_{x_j \in \boldsymbol{x}} P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i) + (1-\alpha_i) \cdot P(\boldsymbol{x}|\text{bot}, \boldsymbol{S}_i). \end{aligned}$$

Here α_i is the fraction of traffic that is benign in $\boldsymbol{S}_i$, that is

$$\alpha_i = |C(\boldsymbol{x}|\boldsymbol{S}_i)|/(|C(\boldsymbol{x}|\boldsymbol{S}_i)| + |B(\boldsymbol{x}|\boldsymbol{S}_i)|)$$

The marginal distribution along x_j is:

$$P(x_j|\boldsymbol{S}_i) = \alpha_i \cdot P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i) + (1-\alpha_i) \cdot P(x_j|\text{bot}, \boldsymbol{S}_i). \quad (6)$$

Now denote as $\boldsymbol{\mu}(\boldsymbol{\lambda}(x_j))$ the (possibly empty) set of buckets of the features in $\boldsymbol{\lambda}(x_j)$ that receive no attack traffic within $\boldsymbol{S}_i$. This means that $P(x_j|\text{bot}, \boldsymbol{S}_i, \boldsymbol{\mu}(\boldsymbol{\lambda}(x_j))) = 0$, and hence:

$$P(x_j|\boldsymbol{S}_i, \boldsymbol{\mu}(\boldsymbol{\lambda}(x_j))) = P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i). \quad (7)$$

That is, over subset $\boldsymbol{S}_i$, we can estimate the clean distribution of feature x_j by restricting to the unattacked buckets $\boldsymbol{\mu}(\boldsymbol{\lambda}(x_j))$ of any feature conditionally independent of x_j; i.e. $x_k \in \boldsymbol{\lambda}(x_j)$. So, our odds, over $\boldsymbol{S}_i$, becomes:

$$\frac{P(\text{bot}|\boldsymbol{x}, \boldsymbol{S}_i)}{P(\overline{\text{bot}}|\boldsymbol{x}, \boldsymbol{S}_i)} = \frac{P(\boldsymbol{x}|\boldsymbol{S}_i)}{\alpha_i \cdot \prod_j P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i)} - 1. \quad (8)$$

We can now simply use (7) to get the values of $P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i)$.

5.1 Finding the clean distributions

We've seen, in (7), that unattacked buckets of a feature x_k lead us to the clean distribution of any features x_j that are conditionally independent of x_k. We next tackle the question of how to identify unattacked buckets.

Suppose that a set $\boldsymbol{b} = \{b_0, b_1, \cdots, b_{d-1}\}$ of d different buckets of features in $\boldsymbol{\lambda}(x_j)$, are unattacked. From (7) we get that:

$$P(x_j|\boldsymbol{S}_i, b_m) = P(x_j|\boldsymbol{S}_i, b_n) = P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i), \; \forall \; m, n \in \boldsymbol{b}. \quad (9)$$

That is, since all unattacked buckets of features in $\boldsymbol{\lambda}(x_j)$ give the clean distribution of x_j their marginal distributions will be equal. We next show that the converse is almost true. That is, if several buckets have the same marginal distribution then they are unattacked if we can assume that the abuse traffic does not precisely match the clean distribution (e.g., at least some of the benign distributions are unknown to or unachievable by the attacker).

Let's examine what happens when the marginal distributions of a set $\boldsymbol{b} = \{b_0, b_1, \cdots, b_{d-1}\}$ of the buckets of features in $\boldsymbol{\lambda}(x_j)$ are equal; i.e., $P(x_j|\boldsymbol{S}_i, b_m) = P(x_j|\boldsymbol{S}_i, b_n) = \boldsymbol{\phi} \; \forall \; m, n \in \boldsymbol{b}$ for some fixed $\boldsymbol{\phi}$ (but we don't know whether or not they are unattacked).

From (6) we get that $\forall \; b_m \in \boldsymbol{b}$:

$$\begin{aligned} \boldsymbol{\phi} &= \alpha_m P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i, b_m) + (1-\alpha_m) P(x_j|\text{bot}, \boldsymbol{S}_i, b_m) \\ &= \alpha_m P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i) + (1-\alpha_m) P(x_j|\text{bot}, \boldsymbol{S}_i, b_m). \quad (10) \end{aligned}$$

Recall that $P(x_j)$ is an N_j-dimensional unit vector. Thus, for a single b_m, the righthand side of (10) is a vector that lies on the line segment joining $P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i)$ and $P(x_j|\text{bot}, \boldsymbol{S}_i, b_m)$. When we consider all of the b_m, ϕ is the intersection of these d line segments. Since all of these line segments originate at $P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i)$ clearly $\phi = P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i)$ is one solution. This implies that all of the $\alpha_m = 1$, the b_m contain no abuse traffic; i.e., as desired. we have found the clean distribution.

Other solutions are possible. However, since ϕ is the intersection of line segments that already intersect at $P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i)$ this requires that all of the line segments are the same: i.e., $P(x_j|\text{bot}, \boldsymbol{S}_i, b_m) = P(x_j|\text{bot}, \boldsymbol{S}_i, b_n)$ for every $b_m, b_n \in \boldsymbol{b}$. This in turn requires that all of the α_m are equal.

Recall that:

$$\alpha_m = |C(\boldsymbol{x}|\boldsymbol{S}_i, b_m)| / (|C(\boldsymbol{x}|\boldsymbol{S}_i, b_m)| + |B(\boldsymbol{x}|\boldsymbol{S}_i, b_m)|).$$

Hence $\alpha_0 = \alpha_1 = \cdots = \alpha_{d-1}$ gives:

$$\begin{aligned} &(|C(\boldsymbol{x}|\boldsymbol{S}_i, b_0)|, |C(\boldsymbol{x}|\boldsymbol{S}_i, b_1)|, \cdots, |C(\boldsymbol{x}|\boldsymbol{S}_i, b_{d-1})|) \\ &\quad \propto (|B(\boldsymbol{x}|\boldsymbol{S}_i, b_0)|, |B(\boldsymbol{x}|\boldsymbol{S}_i, b_1)|, \cdots, |B(\boldsymbol{x}|\boldsymbol{S}_i, b_{d-1})|). \end{aligned}$$

This says that the ratio of abuse to clean traffic is identical in all d buckets. Since it's implausible that this happens by chance it implies that the attacker knows, and can precisely achieve, a d-dimensional sub-space of the clean distribution. In other words, we assume that achieving the same relative traffic strengths across d buckets is not possible if the attacker doesn't know the distribution.

A sole remaining exception is the corner case where $\alpha_0 = \cdots = \alpha_{d-1} = 0$; i.e., our d buckets contain no clean traffic at all. If an attacker sends identical traffic on, e.g., Chrome57, Chrome58, Chrome59 these buckets would have identical marginals for every x_j (assuming no clean traffic from those browsers). We easily rule this corner case out if at least one bucket in $\boldsymbol{b}$ is known to have at least some legitimate traffic. Buckets from features such as state, city, ISP, for example, should suffice: the mixture of benign-to-abuse will vary, but there should be no state or city or ISP contributing zero legitimate traffic.

A toy example may clarify the process. Suppose we seek the clean distribution of x_j and have that $\boldsymbol{\lambda}(x_j) = \{$browser, state, domain$\}$ (i.e., x_j is conditionally independent of those features in the clean traffic). Suppose we satisfy (9) with buckets $\boldsymbol{b} = \{$Chrome71, Chrome72, Edge84, Iowa, Oregon, Ohio, yahoo, hotmail$\}$. This tells us that those buckets are unattacked unless the ratio of abuse to clean traffic is the same in each of those buckets. This says that the attacker can precisely achieve a $d = 8$-dimensional subset of the benign distribution. The corner case where traffic from these buckets is all malicious instead of all benign is ruled out since it is implausible that, e.g., all traffic from Oregon or yahoo is malicious.

Notes: Exact equality of distributions is of course hard to achieve; for approximate equality of distributions we use the Kullback-Leibler (KL) divergence [16, 25]. Note also that we can be fairly tolerant of false negatives when searching for unattacked buckets: if 20 out of 50 buckets of a feature are unattacked it doesn't matter very much if we find only 5 of them.

5.2 Overall algorithm

We start with a collection of subsets $\boldsymbol{S}_i$ such that $\cup_i \boldsymbol{S}_i$ covers all of the data (using different countries as the subsets is our default). We seek classification rules for a set of features $\boldsymbol{x}$ that are mutually independent conditioned on $\boldsymbol{S}_i$. We need a collection of conditional independence relations:

$$x_j \perp \boldsymbol{\lambda}(x_j) | \boldsymbol{S}_i \;\; \text{for each } x_j \in \boldsymbol{x}. \quad (11)$$

In Section 6 we'll give the reasoning behind the independence relations we use in the case of web-logs.

We find an estimate for the clean distribution of feature x_j by searching (among features that are independent of x_j conditioned on $\boldsymbol{S}_i$) for clusters of buckets where $P(x_j|\boldsymbol{S}_i, b_m) \approx P(x_j|\boldsymbol{S}_i, b_n)$ for every b_m, b_n in the cluster.

A matrix interpretation of the approach in Section 5.1 aligns well with the implementation. Let $\boldsymbol{\Omega}, \boldsymbol{\Sigma}$, and $\boldsymbol{\Theta}$ be the observed, $\overline{\text{bot}}$ and bot matrices of marginal distributions of x_j over all the bins of all the features in $\boldsymbol{\lambda}(x_j)$. These have columns $P(x_j|\boldsymbol{S}_i, b_m), P(x_j|\overline{\text{bot}}, \boldsymbol{S}_i, b_m)$ and $P(x_j|\text{bot}, \boldsymbol{S}_i, b_m)$ for every bucket b_m of every feature $x_k \in \boldsymbol{\lambda}(x_j)$ respectively. Calculating $\boldsymbol{\Omega}$ is efficiently implemented as a matrix pivot operation. Clearly, we have:

$$\boldsymbol{\Omega} = \alpha \cdot \boldsymbol{\Sigma} + (1-\alpha) \cdot \boldsymbol{\Theta}.$$

Algorithm 1 PROS approach to find clean distributions. We iterate over the subsets for which conditional independence is defined and over each feature. Note that for features where the clean distribution is slowly-varying this training need not be re-run very often.

```
for $\boldsymbol{S}_i$ in $\{\boldsymbol{S}_i\}$ do
    for $x_j$ in $\boldsymbol{x}$ do
        $\boldsymbol{\Omega}(x_j) =$ data.pivot(index=$x_j$, columns =$\boldsymbol{\lambda}(x_j)$, aggfunc = sum())
        $\boldsymbol{\mu}(\boldsymbol{\lambda}(x_j)) =$ bins corresponding to largest cluster of co-linear columns of $\boldsymbol{\Omega}(x_j)$
        $\hat{P}(x_j|\boldsymbol{S}_i,\overline{\text{bot}}) \triangleq \text{Avg}\{P(x_j|\boldsymbol{S}_i,\boldsymbol{\mu}(\boldsymbol{\lambda}(x_j)))\}$
    end for
end for
```

Here $\boldsymbol{\Sigma}$ is rank-one, since $P(x_j|\overline{\text{bot}},\boldsymbol{S}_i)$ is independent of every $x_k \in \boldsymbol{\lambda}(x_j)$. However $\boldsymbol{\Theta}$ will in general be of full rank (since we are not assuming any independence relations in the abuse traffic). As the sum of rank-one and (in general) full-rank matrices, $\boldsymbol{\Omega}$ will also in general be full-rank.

To find the clean distribution of x_j we seek to form a rank-one matrix $\boldsymbol{\Omega}^{'}$ with a subset of the columns of $\boldsymbol{\Omega}$:

$$\boldsymbol{\Omega}^{'} = \alpha \cdot \boldsymbol{\Sigma}^{'} + (1-\alpha)\cdot \boldsymbol{\Theta}^{'}.$$

The columns of this rank-one matrix $\boldsymbol{\Omega}^{'}$ give the clean distribution. Algorithm 1 describes the approach.

Once we have estimates of the clean distributions we calculate the odds (i.e., compute (8)); this is done in Algorithm 2. This is simply an implementation of (8) with $\alpha_i = 0.5$ (recall from Section 4 that order is unaffected by α). The output of the algorithm is a set of rules that covers every possible observation. Every possible tuple of the features has a rule as generated by (8) and this is done for every subset $\boldsymbol{S}_i$. For example, if $\boldsymbol{x} = (x_0, x_1, x_2, x_3)$ is (browser, State, hour-of-day, organization) then a typical rule might be

(Chrome77.0.3770, Georgia, 13, Comcast),

together with the odds estimated in (8). Table 2 in Section 7 gives an example.

Algorithm 2 Calculate Rules: we use the clean distributions estimated in Algorithm 1. Note that updating rules can be re-run very often.

```
for $\boldsymbol{S}_i$ in $\{\boldsymbol{S}_i\}$ do
    $\text{Odds}(\boldsymbol{x}|\boldsymbol{S}_i) = P(\boldsymbol{x}|\boldsymbol{S}_i)/(0.5\cdot\prod_j \hat{P}(x_j|\overline{\text{bot}},\boldsymbol{S}_i)) - 1$
end for
```

5.3 Implementation notes

Conditional independence rules and subsets: We go through our reasoning for conditional independence relations for the concrete example of web-logs in Section 6. While we don't claim a procedure that is general for all applications and feature sets, this task is done only once, and we generally have a small number of features, so decisions on pairwise independence can be decided manually. Note that we can conjoin features: if x_j is independent of both `browser` and `week` it is independent of `browser-week`. This expands the number of bins (and can make it easier to find unattacked ones) at the cost of fewer data samples per bin.

Observe that PROS simply detects violations of our independence assumptions. If (11) holds then $\boldsymbol{\Sigma}$ is rank-one. If what we observe, $\boldsymbol{\Omega}$, is not rank-one then either the independence assumptions are wrong, there is a corrupting component $(1-\alpha)\cdot\boldsymbol{\Theta}$, or sampling effects are too great (precise independence is never achieved when we have finite number of samples). Thus, if we have a lot of data and high confidence in our independence assumptions, any deviations from independence in observed traffic indicates abuse.

PROS fails safely: if we are wrong about our independence assumptions, or there are no unattacked bins, then we will be unable to find a rank-one subset of the columns of $\boldsymbol{\Omega}$.

Estimating α: While the ordering of our odds estimates is independent of α it would still be very valuable to estimate it accurately. If we know $P(\boldsymbol{x})$ and $P(\boldsymbol{x}|\overline{\text{bot}})$ it can be seen that (2) still under-determines α and $P(\boldsymbol{x}|\text{bot})$; i.e., there is a range of possible solutions. The most conservative choice is to take the maximum value of α consistent with $P(\boldsymbol{x})$ and $P(\boldsymbol{x}|\overline{\text{bot}})$; this attributes as much traffic as possible to benign rather than malicious sources. This is found by taking the orthogonal projection of $P(\boldsymbol{x})$ onto $P(\boldsymbol{x}|\overline{\text{bot}})$:

$$\hat{\alpha} = \frac{\langle P(\boldsymbol{x}), P(\boldsymbol{x}|\overline{\text{bot}})\rangle}{\langle P(\boldsymbol{x}|\overline{\text{bot}}), P(\boldsymbol{x}|\overline{\text{bot}})\rangle}. \quad (12)$$

This is an upper bound. With this choice $\hat{P}(\boldsymbol{x}|\text{bot}) \triangleq [P(\boldsymbol{x}) - \hat{\alpha}\cdot P(\boldsymbol{x}|\overline{\text{bot}})]/(1-\hat{\alpha})$ will be orthogonal to $P(\boldsymbol{x}|\overline{\text{bot}})$, but will not necessarily have non-negative coefficients (and thus can't be a probability). This suggests our upper bound is loose. We can improve it by reducing $\hat{\alpha}$ until $P(\boldsymbol{x}) - \hat{\alpha}\cdot P(\boldsymbol{x}|\overline{\text{bot}})$ has only non-negative coefficients. This can be done by a simple one-dimensional search.

However, a word of caution on numerical robustness is necessary. It's easy to show that the upper bound in (12) is tight when class separation along $\boldsymbol{x}$ is good (i.e., $\langle P(\boldsymbol{x}|\text{bot}), P(\boldsymbol{x}|\overline{\text{bot}})\rangle \approx 0$), and loose when it is not. Seeking to improve a loose estimate by reducing $\hat{\alpha}$ as above can be sensitive to sampling effects and works best when the volume of data is very high.

Streaming data: When we run Algorithm 1 on streaming data it can increase robustness to update rather than calculate $\hat{P}(x_j|\boldsymbol{S}_i, \overline{\text{bot}})$ afresh over each block. The final estimate might be a weighted sum of the historical and current-block estimates; we might tilt the weights heavily toward historical estimates for slowly-varying features (e.g., state, family) and toward current-block for those that evolve rapidly (e.g., browser).

Concept drift: Note that our estimates of the clean distributions $P(x_j|\overline{\text{bot}})$ need not be retrained very often. Certain features, such as browser version might need to be re-estimated often, but others (such as geographic distribution) should be very-slowly varying. By contrast, we expect that the bot distribution to change very rapidly. Thus Algorithm 2 might be re-run very often.

Feature cardinality: categorical features often have high cardinality with many rare tokens. We run an encoding stage that maps the least-common tokens into a single 'other' bin. This procedure is sometimes known as backoff.

Efficiency: The clustering in the inner loop of Algorithm 1 can be a brute-force search if necessary. Since we deal with a small number of categorical features that have dozens or so buckets even a full N^2 search (where N is the sum of the cardinalities of features in $\boldsymbol{\lambda}(x_j)$) might not be unmanageable. We are also aided by the fact that we are tolerant of false negatives (i.e., unattacked bins that we don't find) in this search. So long as we find a few unattacked buckets we can estimate the benign distribution. Thus, if a full N^2 is too expensive we can do a search only over a sample of the buckets, and repeat several times if the sub-searches fail. Observe that Algorithm 1's only dependence on the size of the dataset comes from the matrix pivot operation; after that the complexity of finding unattacked bins is determined by the feature cardinalities. An interesting approach (which we have not implemented) to finding rank-one submatrices efficiently is described by Doan and Vavasis [18]. Running Algorithm 1 on the 3m row dataset of Section 6 took about 30 minutes on a 2.2GHz dual processor PC with 128G of RAM, and Algorithm 2 took 4 minutes. This does not include the time for parsing `userAgent` strings into its components.

Interpretability and Identification of clusters of bot activity: The rules output by Algorithm 2 are used to assign bot likelihood to each request, but just as importantly they help identify clusters of bot activity. For example, suppose a bot sends lots of traffic from Florida using IE9. In this case in a browser-state ruleset the rule "Florida and IE9" will appear near the top (when sorted by bot likelihood). That bot activity is often clustered in feature-space is exploited in many detection approaches [11, 13, 35, 36, 38, 39].

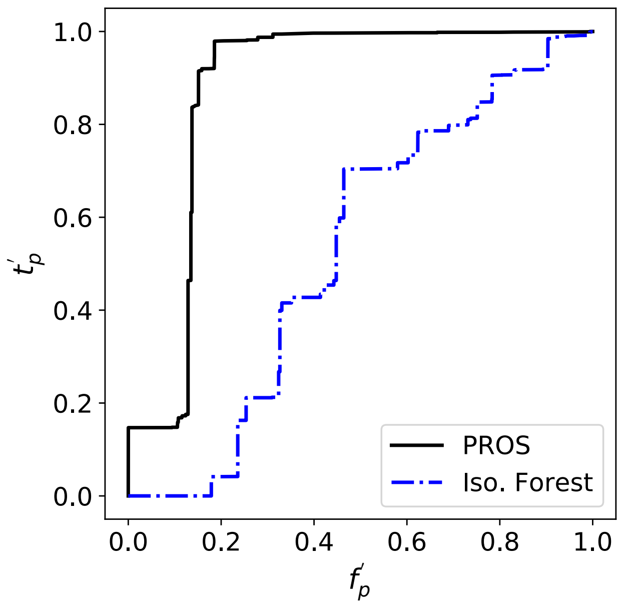

Figure 1: Lower bound for the ROC curves of PROS and Isolation Forests using features `browser, family, status` and `path`. Having labels for only a portion of the positive samples (and none for the negatives) the true ROC curve is, in each case, above and to the left of the curve shown. (a) Solid line: lower-bound ROC curve for PROS (AUC=0.877). (b) Dash dot line: lower-bound ROC curve for Isolation Forests (AUC=0.532).

6 Evaluation on web-log dataset

The site `www.secrepo.com` offers a repository of various security-relevant datasets. The daily logs of all web requests are made available under the Creative Commons Attribution 4.0 License. It receives approximately 15k requests per week, so that five years' worth of logs comprises about 3m rows in 2,000 separate files.[4] Our capture covers dates from January 2015 to August 2021.

The logs are stored in Apache Combined Log format [4]. The recorded fields are: `IP address, HTTP status, response size, referer, userAgent, timestamp, HTTP request`. From `IP address` we derive the features `city, state` and `country` using an IP geolocation service; e.g., `3.94.31.115` yields 'Ashburn', 'Virginia', 'United States'. From `userAgent` we derive `browser, family, OS, osFamily` using the Python library `user_agents`; e.g., 'Mozilla/5.0 (Windows NT 6.3; WOW64; rv:56.0) Gecko/20100101 Firefox/56.0' yields 'Firefox56.0', 'Firefox', 'Windows8.1', 'Windows'. From `timestamp` we derive `day, week, month` and `year`. From `HTTP request` we derive `method` and the `path` as the sub-

[4] We have no affiliation with the site, but did communicate with the site administrator who confirmed certain details below.

directory of the requested item; e.g., the request `GET /self.logs/access.log.2019-04-15.gz` yields `GET` and `/self.logs`.

Given these features the set of conditional independence relations we chose are as follows:

$$\begin{aligned}
\text{browser} &\perp \{\text{country}, \text{city}, \text{state}, \text{path}, \text{status}\} \mid \text{family}, \\
\text{status} &\perp \{\text{browser}, \text{week}, \text{family}, \text{osfamily}\} \mid \text{country}, \\
\text{path} &\perp \{\text{browser}, \text{week}, \text{family}, \text{osfamily}\} \mid \text{country}, \\
\text{family} &\perp \{\text{city}, \text{state}, \text{path}, \text{status}\} \mid \text{country}\}.
\end{aligned}$$

For example, the first line indicates that (within a browser family such as Chrome) we expect version to be independent of country, city, state, path and HTTP response status. Our reasoning is that version should be relatively independent of location since modern browser families regularly push automatic updates and regional user preferences have little chance to affect $P(\text{browser}|\text{family}, \overline{\text{bot}})$.[5] Similarly, we assume (within a browser family) that benign users of, say, Chrome82 are not more or less likely to visit particular paths or receive particular response codes than benign users of any other Chrome version.

The second line says that, within a country, we expect different browser versions, browser families, etc are equally likely to receive a particular response. Our reasoning is that, for example, human Firefox users should receive responses of 200 (i.e., success) and 404 (i.e., file not found) at the same rate as Chrome and Safari users.

As Chio and Freeman explain [15] there is little prospect of obtaining human-chosen labels for web-log requests. However, this logset contains three groups of data which we can label as automated traffic.

1. **Traffic identified as automated by the site**. The site responds to certain requests with HTTP response 418 "I'm a teapot." This response is often used by servers for [1] "requests they do not wish to handle, such as automated queries."

2. **Traffic requesting (non-existent) WordPress resources.** Malicious bots will often probe for WordPress resources looking for vulnerable configurations. The site administrator confirmed to us that the site has never had any WordPress installation. Thus, all requests to `wp-login, wp-admin, wp-includes` etc can be considered honeypot requests; e.g., the logset contains 90k requests to `wp-login`, the authentication entry point for the WordPress dashboard.

3. **Traffic requesting the daily logs.** The site contains five years of daily logs in 2,000 individual files with naming format `access.log.YYYY-MM-DD.gz`. Obviously, requests sampling these logs one-at-a-time will be dwarved by scripted requests. Inspection confirms that requests for these files are strongly dominated by repeated bursts (i.e., from the same userAgent and IP address) requesting a large number of consecutive files. Of course, while these bot requests are not harmful they are the type of automated activity that PROS attempts to identify.

[5]This does not imply distribution of browser family is independent of country (which we do not assume).

Thus, we label all requests with HTTP response status 418, or containing 'wp-' or 'access.log' in the request field as bot. Together they cover 39.1% of requests in the logset. Harnessing the output of a rules-based system as a source of labels is common practice (apparently including Facebook's abusive account detection system [39]).

Generating a Receiver Operator Characteristic (ROC) curve requires labels for all samples. Given labels for only a portion of the positive samples we can however bound the ROC curve. Since we have labels only for a portion of the positives (and count everything else as negative) we under-estimate the 'True' counts (i.e., TP and TN) and over-estimate the 'False' counts (i.e., FP and FN). We denote these available counts as $TP^{'}$ etc to distinguish them from the actual counts TP etc. Thus, a plot of

$$t_p^{'} = TP^{'}/(TP^{'} + FN^{'}) \text{ versus } f_p^{'} = FP^{'}/(FP^{'} + TN^{'})$$

will always be below and to the right of the ROC (since $t_p \geq t_p^{'}$ and $f_p \leq f_p^{'}$).

Using the conditional independence rules above we run Algorithm 1 to generate the clean distributions for `browser, family, path` and `status` and Algorithm 2 to generate rules. The dataset is too small to allow accurate per-country estimates for the features `status, path` and `family` or per-family estimates of browser except for Chrome and Firefox. To compensate we compute global estimates of the former (i.e., lump traffic from all countries together) and compute browser estimates only for Chrome and Firefox. Since we expect distributions of `family, path` and `status` to be relatively time-invariant we ran over the entire dataset. When seeking clean distributions of `status` and `path` we conjoined `browser` and `week`, so that Algorithm 1 sought unattacked bins over a much larger set. As for `browser`, since its clean distribution evolves we estimate its clean distribution, and generate rules, over one year blocks (with sufficient traffic it would be advisable to re-estimate browser distribution much more regularly, e.g., daily). Only Chrome and Firefox had sufficient traffic to allow us to estimate the distribution of versions. Thus, we calculated one ruleset using `browser, path` and `status` for Chrome and Firefox traffic and another ruleset using `family, path` and `status` for all other traffic. The calculated rules allow us to associate a likelihood of being bot with every request.

Figure 1 shows our results. The solid line is the lower bound for the ROC curve (i.e., on the available labels); the Area

Under Curve (AUC) is 0.877. For comparison we show the performance of an Isolation Forest classifier using `family`, `path` and `status` (using the `sklearn` implementation and one-hot encoding to map categorical variables to numeric). Clearly, Isolation Forests struggle to do better-than-random; the AUC is 0.532. As explained in Sections 2 and 3 Isolation Forests work well on numeric features, but struggle when categorical features are one-hot encoded to numeric.

Thus, PROS significantly outperforms Isolation Forests. However, at low "false positive" rates (e.g., $f_p^{'} < 0.05$) it appears to do poorly. A possible reason is that we have labels only for some of the positives: it's likely that there is scripted activity not captured by the three groupings we detailed above. Obviously it is difficult to achieve low "false positive" rates when an unknown fraction of positives are actually mis-labelled as negatives.

Sorting rulesets by bot likelihood allows us to identify clusters of bot activity. For example, if we sort the `browser` ruleset a clear pattern that appears is certain out-of-date browsers from major browser families being used for bot activity. For example, Firefox40.1 accounts for 34.7% of all Firefox traffic in the dataset even though it was superseded in September 2015; Chrome34.0.1847 accounts for 18.0% of all Chrome traffic in the dataset even though it was already six major versions out-of-date by the earliest date in our dataset.

While bots often seem to use legacy versions of major browsers it would be hasty to conclude we can block all such traffic. For example, Chrome49.0.2623 also accounts for significant traffic long after it was superseded, but does not feature as having high bot likelihood in our `browser` rulesets. Upon investigation it emerges this was the last version of Chrome supported on Windows XP; so this is likely legitimate traffic from old systems that can't update.

7 Evaluation on Twitter dataset

Twitter exposes an API that allows downloading certain information about individual accounts and about tweets [2]. It allows retrieval of information such as the screen_name, name, profile description, creation date, numbers of accounts followed, number of followers, number of tweets liked, most recent tweet, etc.

The API is throttled, so that the amount of data that can be downloaded is very restricted. The list of id's of followers of an account can be downloaded at a rate of about 5000 per minute; the details of those follower accounts can be downloaded at a rate of about 100 per minute.

To study an account we do not need a list of all of the followers, a sample probably suffices. Unfortunately the API does not provide a sampling mechanism. Hence, for a target account, we first download the entire list of follower id's, and then uniformly at random select a subset of its followers to download. Since the follower list for a single account with 10 million followers would take about 33 hours to download we restricted our attention to accounts that have 3 million followers and below. For example, to sample 20k followers of an account with 500k followers takes approximately $60 \times (500/5 + 20/0.1) = 300$ minutes or 5 hours.

In addition, the Botometer project at Indiana University [17] exposes an API that calculates their estimate that a Twitter account is bot. As the most accessible benchmark we compare our estimates against those from Botometer, as well as against Isolation Forests.

7.1 Dataset

Our dataset is a convenience sample acquired between January 10 and May 5 of 2021. It contains 159 `target` accounts; for each target we downloaded profile information on a uniform-at-random sample of its followers. In all we gathered the profile details of 5.1 million Twitter accounts; i.e., an average of 32k followers per target.

We chose `target` accounts by searching Twitter for keywords 'election fraud', 'brexit', 'vaccine choice', and 'climate hoax' and selecting the first accounts that matched our criteria of follower count between 20k and 3 million. For each of these chosen targets we downloaded the entire list of followers, selected at least 20k followers (uniformly at random) and downloaded the details of those accounts. We also added a small number of target accounts (e.g., @USENIXsecurity, @SuezDiggerGuy) to check that the clean distributions found are robust and not particular to the topics covered by the selected target seed accounts. We also added two `target` accounts acknowledged to have purchased bot followers (see Section 7.2.1). Our goal is simply that the dataset contain some non-trivial mixture of legitimate and abuse accounts. We emphasize that we make no claim that this dataset is in any way representative of the overall Twitter population.

We revisited all 159 of the `target` accounts on June 2, 2021. Of the 159 five had been suspended with a notice reading "Twitter suspends accounts which violate the Twitter Rules." This differs from the notice if an account is closed, and presumably indicates problematic behavior. Ranking the accounts from highest to lowest estimated bot follower-ship the five suspended target accounts were in positions #2, #23, #27, #41 and #61.

In addition to the retrieved features we derive from them certain others that are useful. In particular we will use `pattern`, `year`, `monthyear`, and `client`.

Pattern: a mapping of the account screen_name that maps lower-, upper-case, digits and special characters to 'l', 'U', 'd' and 's' respectively. With the exception of 'd', consecutive runs of the same character in the mapping are collapsed to a single instance. Thus, 'JohnDoe78' maps to 'UlUldd' and 'johndoe78' maps to 'ldd.' This mapping is a minor modification of the one used by Xiao et al [38].

Bins: we find it useful to define certain binary features. `noLocn`, `noDesc` and `noPic` indicate empty location and de-

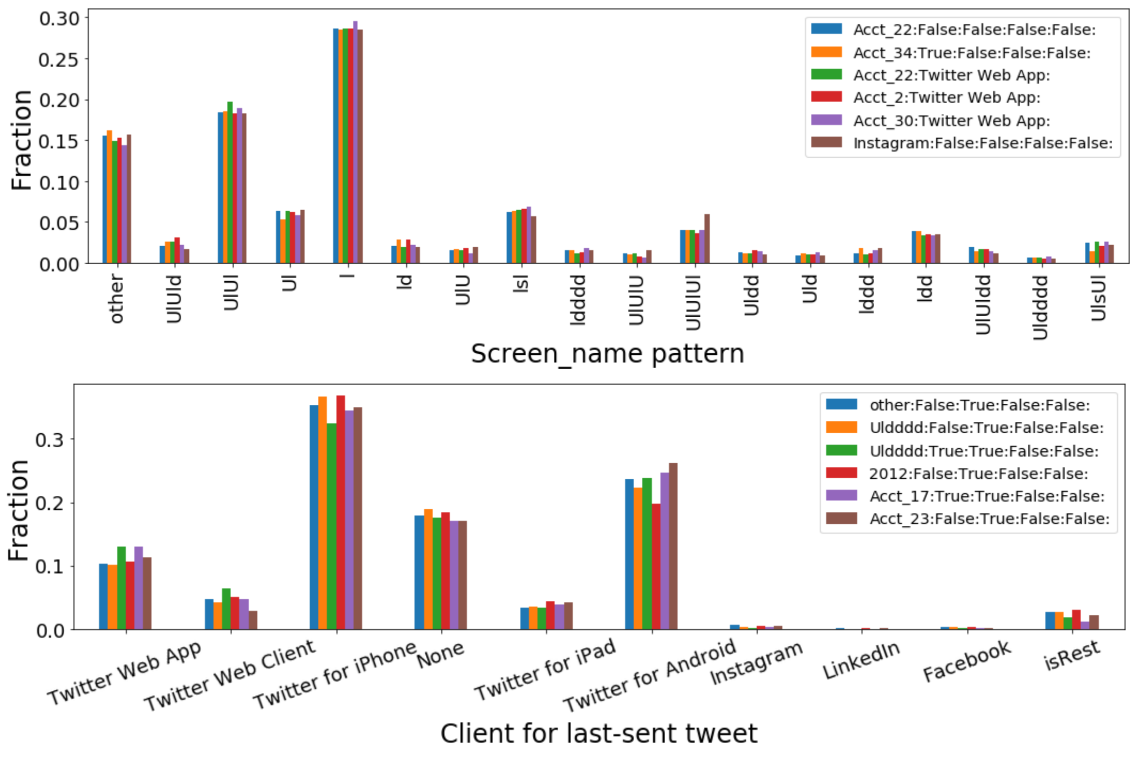

Figure 2: Clean distributions for `pattern` and `client` estimated by Algorithm 1. Observe that in each case we have $d = 6$ buckets giving very similar marginal distributions. That is, $P(x_j|b_m) \approx P(x_j|b_n)$ where $b_m, b_n \in \boldsymbol{b}$ and $\boldsymbol{b}$ is the set of buckets listed in the graph legend. Each marginal is $\alpha \cdot \text{Clean} + (1-\alpha) \cdot \text{Abuse}$. Section 5.1 explains why identical marginals implies that we have found the clean signal (i.e., $\alpha \approx 1$).

scription fields and if the account has a default profile picture. `noStat` indicates that the account has zero tweets. In addition we form a composite feature `Bins` by conjoining all four. For example, True, False, True, False would indicate empty location and default profile picture but a non-empty description and that the account has tweeted at least once. This Bins feature is conjoined with others in the set of conditional independence relations below.

The set of conditional independence relations we assume are as follows:

$$\begin{aligned}
\text{pattern} &\perp \{\text{targetBin}, \text{clientBin}, \text{yearBin}\},\\
\text{monthyear} &\perp \{\text{patternBin}, \text{clientBin}, \text{targetBin}\},\\
\text{client} &\perp \{\text{targetBin}, \text{patternBin}, \text{yearBin}\},\\
\text{year} &\perp \{\text{clientBin}, \text{patternBin}\}.
\end{aligned}$$

Note that `clientBin, patternBin, targetBin` etc refer to conjoining the `client` (resp. `pattern, target`) feature with the four binary features `noLocn, noDesc, noPic, noStat` which indicate, respectively, that the location, description, picture fields are empty or that the account has never tweeted.

Using these conditional independence relations we run Algorithm 1 to find the clean distributions of `pattern, monthyear, client, ratio` and `year`. Figure 2 shows the resulting clean distribution for `pattern` and `client`. Figure 2 (a) indicates that a cluster of $d = 6$ buckets among `targetBin` and `clientBin` were found to have co-linear marginals. Figure 2 (b) indicates that a cluster of $d = 6$ buckets among `patternBin, yearBin` and `targetBin` were found to have co-linear marginals.

7.2 Measurements

We present the following analysis of Twitter data to allow evaluation of the algorithm and to corroborate that it finds bot-like activity. We wish to emphasize, however, that this is not primarily a paper about Twitter or social-media bots. The rulesets we construct are limited by the features accessible through the API. The throttling of the API makes generating a larger dataset and/or including activity information about the accounts impractical. We wish to be clear that we do not claim to outperform algorithms that have access to graph, connectivity and activity information. Our goal is simply to demonstrate efficacy of the approach. Nonetheless, we achieve comparable performance to Botometer [17], even though our approach is generic rather than specific to social-media bots and we use a more limited set of features.

7.2.1 Accounts known to have bot followers

We obtained the Twitter handles of two accounts from individuals who acknowledged purchasing followers (see Section 7.3 for a discussion of ethical considerations). They did so in a discussion group (on a non-Twitter platform) dedicated to building social media engagement. After contacting them on a private messaging channel they were willing to share the Twitter handles with us, and the dates and size of the purchased blocks. They did not indicate whether they owned these accounts or were acting on behalf of others.

This gives us a source of labels to check efficacy. These are Acct_80 and Acct_159 in our dataset. Of Acct_80's followers $\approx 65\%$ were purchased; of Acct_159's $\approx 60\%$ were purchased. Since both accounts would have had organic growth at the same time as bot followers were being added we took our bot samples as the first 1k followers of the account (i.e., before purchases were a factor) and our bot samples as the first 1k followers beginning after the largest purchase event (e.g., Acct_80's bot followers were purchased in several different blocks). As a verification step, we manually examined 30 positive and 30 negative samples for both accounts; the positive samples exhibited considerably lower levels of engagement with the `target` account's content.

We created a ruleset with features `target`, `pattern`, `client` and `year`. In the creation of this ruleset Algorithm 2 would have used the clean distributions of `pattern` and `client` shown in Figure 2. Using the odds estimated by Algorithm 2 and the labels from our positive and negative samples we analyze Acct_80 and Acct_159.

Figure 3 shows ROC curves for these two accounts. In both cases we identify almost half of the true positives at a very low false positive rate. This indicates that half of the bots are easily separated from organic accounts, but things become progressively more difficult thereafter. The curves produced by Botometer [17] and Isolation Forests (using one-hot encoding) are also shown.

For Acct_80 the AUCs for PROS, Botometer and Isolation Forests were 0.795, 0.792, 0.657 respectively. While PROS and Botometer have similar AUCs note that our approach outperforms Botometer when the false positive rate is low. For consumer services, a low false positive rate is often considered mandatory: blocking legitimate users has a serious effect on user-experience, so operating at $f_p < 0.01$ may be required. On this basis PROS outperforms Botometer. Observe that Isolation Forests struggle to do better than random. For Acct_159 the AUCs for the three approaches were 0.811, 0.708 and 0.531. Again, observe that PROS does better than Botometer when f_p is low. In the interest of fairness we point out that, since it uses account-activity features, the comparison with Botometer is not like-to-like. When an account has no activity Botometer has no basis for judgement, and thus is at a disadvantage.

Note that Xu et al [39] describing abusive account detection at Facebook report achieving AUC of 0.89. While the details are very different and we do not claim to rival their system, their results offer good calibration on the state-of-the-art.

Target	Pattern	Client	Year	Count	Odds	Btmtr
Acct_8	ldd	Mobile Web (M2)	2013	404	692.85	0.93
Acct_8	other	Mobile Web (M2)	2013	381	391.03	0.88
Acct_74	l	Mobile Web	2009	189	165.61	0.81
Acct_74	other	Mobile Web	2009	110	110.31	0.86
Acct_74	l	Twitter Web Cl.	2009	3383	100.40	0.79
Acct_12	Ul	None	2012	1208	73.14	0.81
Acct_8	other	-1	2012	625	72.56	0.84
Acct_46	Uld	None	2016	842	68.57	0.88
Acct_74	ldd	Twitter Web Cl.	2009	547	64.17	0.80
Acct_46	UlUld	None	2016	1512	63.86	0.82

Table 2: Example ruleset using features `target, pattern, Client` and `year` sorted in descending order of bot-likelihood. Thus, followers of Acct_8 with screen_name patterns 'ldd' whose last tweet was sent using 'Mobile Web (M2)' and were created in 2013 are estimated to have a 692:1 odds of being bot and a fraction 0.93 of them are estimated to be bot by Botometer [17].

7.2.2 Account clusters with randomized screen_names

We again use the ruleset generated in Section 7.2.1 (i.e., with features `target, pattern, client` and `year`) over the whole dataset. We show a few rows of the output rules in Table 2. We also include a column that gives the median Botometer score for accounts covered by the rule. This can be thought of as the estimated fraction of accounts covered by the rule that Botometer estimates to be bot. Note that there is good agreement: where our method predicts high odds of being bot, Botometer largely concurs.

We inspected accounts covered by the rules with the highest odds. Many stood out as revealing additional structure indicative of automated creation. For example, the first line of Table 2 covers accounts with pattern 'ldd' following Acct_8 that were created in 2013 and whose last tweet used 'Mobile Web (M2)' as client. This rule covers a total of 4.7k of the followers of Acct_8 of which we had 404 in our sample dataset (i.e., the dataset contains 30k of Acct_8's 350k followers). Upon inspection all of these accounts appeared to have screen_names that were a last name followed by two apparently random lower-case letters and two apparently random digits. A sample of the last names is 'colwell', 'nichelson', 'fischbach', 'langston', 'whistler'; a sample of the suffixes is: 'un33', 'ot68', 'eq79', 'zo99', 'iq05.' To test the hypothesis that these suffixes are random we compared the empirical distribution of the last characters with a uniform distribution of digits. The observed frequency of digits 0-9 were $(38, 42, 36, 48, 35, 36, 38, 43, 42, 46)$ while under a uniform distribution we would expect 40.4 in each bin. Thus the

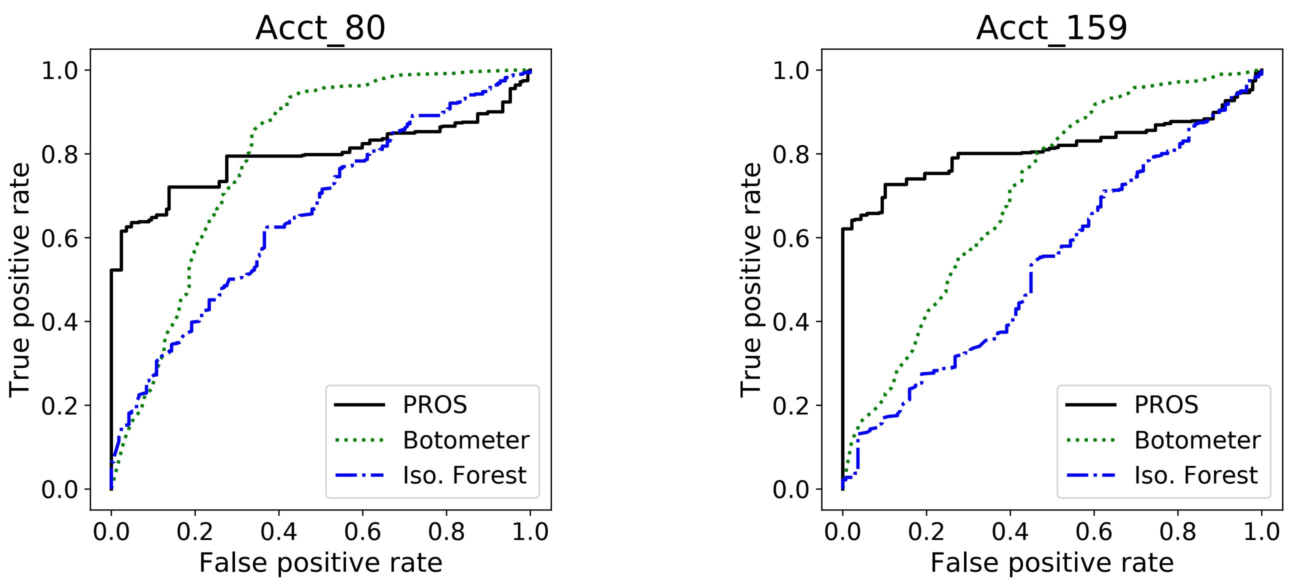

Figure 3: ROC curves for two accounts with acknowledged purchased bot-followers (see Section 7.3 for a discussion of ethical considerations). We took collections of 1k followers before and during the purchase periods as our negative and positive samples. Observe that our approach outperforms Botometer [17] at low false positive rates, but does worse thereafter, and easily beats Isolation Forests. Note that PROS and Isolation Forests are using only features available from account profiles (e.g., `target`, `pattern`, `client` and `year`) while Botometer uses account activity. Further, our approach is not specific to Twitter or social media bots: it is a generic unsupervised method applicable to a wide range of problems. For Acct_80 the AUCs for PROS, Botometer and Isolation Forests were 0.795, 0.792, 0.657 respectively and for Acct_159 were 0.811, 0.708 and 0.531.

goodness-of-fit to uniform is very good. A chi-square test confirms that the null-hypothesis (of uniform distribution) is not refuted. Similarly, we also compare the observed distribution of second-to-last character with a uniform distribution of digits, and compare the third- and fourth- to last characters with uniform distributions of the lowercase letters. In each case the fit to uniform is excellent and a chi-square test corroborates uniformity.

This effectively guarantees that accounts captured by this rule were generated by a script. It is well-known that distributions of digits chosen by humans (in passwords and usernames etc) are very far from uniform [6]. It is implausible that 404 humans acting independently choose `screen_name` suffixes drawn uniformly from the available characters.

Many other clusters of accounts captured by rules with high odds revealed structure. For example, accounts with pattern 'Ul' following Acct_12 whose last tweet had 'None' as client (i.e., line six of Table 2) appeared to have screen_names that were a first name followed by five apparently random lowercase letters. A sample of the first names is 'Eugene','Freida', 'Norma', 'Janette', 'Madeline'; a sample of the suffixes is 'gnbjs', 'xhdke', 'ffbej', 'cdfqa', 'jezdk'. Again, a chi-square test suggests that the last five characters of the 1208 screen_names covered by this rule are uniformly distributed. Again, we point out that this effectively guarantees that they were not produced independently by 1208 humans.

7.2.3 Abnormal follower creation patterns

We created a simple ruleset with features `target` and `monthyear`. This produces, for each target, a month-by-month likelihood that follower accounts created that month are bots. In sorting this list in descending order of odds we found that certain targets had extremely high odds in some months (suggesting the majority of their followers created in those months were bot). Figure 4 shows examples of accounts that showed high likelihood of having bot followers (as well as two normal accounts for reference). Note that the distribution of follower creation dates is very different from the distribution of dates on which followers started following.

For reference Figure 4(a) and (b) show the distribution of creation dates of followers of @USENIXsecurity (i.e., the Twitter account of the Usenix Security Symposium) and @SuezDiggerGuy (a parody account purporting to represent the operator of the mechanical digger attempting to free a ship stuck in the Suez canal in March 2021). While the topics covered by @USENIXsecurity and @SuezDiggerGuy have little in common it can be seen that the distribution of creation dates is broadly similar (and to the clean distribution of `monthyear` found by Algorithm 1 (not shown)). The distribution of creation dates for @SuezDiggerGuy shows that there is little support for the view that surges in interest in a topic translate into bursts in the distribution of follower creation dates. The @SuezDiggerGuy account was created in March 2021 and interest was concentrated in the seven day period of the blockage.

7.3 Ethical considerations

We first discuss the main Twitter dataset, used in Sections 7.2.2 and Section 7.2.3 (i.e., excluding the two targets studied in Section 7.2.1). Since all of the information on these 157 targets is publicly visible and we do not identify any individuals we did not seek Institutional Review Board approval. That is, all fields that we use are globally visible even to those who do not have Twitter accounts. To mitigate possible harm or embarrassment to individuals we do not identify any of the individual accounts involved directly. We replace every individual target Twitter handle with a unique identifier (e.g., 'Acct_27'). Exceptions are those belonging to organizations (e.g., @USENIXsecurity) or parody accounts (e.g., @SuezDiggerGuy) where there is no risk of revealing information that might be harmful to an individual.

Microsoft has detailed internal guidelines on how Personally Identifying Information (PII) should be handled. Even though all fields in our dataset are publicly visible we adhered to those PII guidelines. These include that data is encrypted at rest, is stored on a Bitlocker-enabled machine and that only those with need have access; as a result at no time did anyone other than the author have access.

We discuss the data from the two targets studied in Section 7.2.1 next. Purchasing bot-followers would offer a simple source of labelled data, but since doing so violates Twitter's Terms of Use we did not pursue this avenue. However, in a public forum that deals with building social media followership we encountered several individuals who openly discussed their experiences purchasing followers for Twitter, Instagram, Youtube and TikTok accounts. After contacting them, two of these individuals shared the Twitter handles of accounts for which they had purchased followers. These are the Acct_80 and Acct_159 that we used for evaluation.

We did not seek IRB approval before contacting these individuals. In retrospect, this was a mistake: seeking IRB approval or exemption might have identified risks. These risks include that the identity of the accounts might be established: either directly from data that we stored, or from what we publish or otherwise reveal. On the first of these risks we reiterate our compliance with the Microsoft PII handling guidelines, and emphasize that the dataset does not contain any profile information, tweets, retweets, likes, follows or other activity information about any of the target accounts: it contains information only about followers. Hence Twitter handles, names, account ID's, etc of Acct_80 and Acct_159 do not appear in the dataset. A file that maps target Twitter handle to Acct_XX strings is stored separately, also complying with PII guidelines; we have deleted the entries for Acct_80 and Acct_159 in this file. We committed not to publish the Twitter handles, or information that would allow them to be easily determined. Thus, we avoid giving details such as join date, nature of the account, etc; we don't include a chart of follower create dates (such as in Figure 2) which might act as a signature. Obviously we are obliged to refuse any queries that might de-anonymize these accounts.

Finally, we did not pay or offer to pay either individual. In contacting them we disclosed our identity and affiliation; we made clear that our intent was publication of our analysis and improvement of bot detection technologies. In each case the only information we obtained was the Twitter handle, the number of followers purchased, and the date.

	Followers	Description
Acct_94	2.4m	Fox News TV host
Acct_15	2.4m	Conservative TV and radio host
Acct_74	200k	Economic inequality activist
Acct_49	50k	Anti-vaccine account
Acct_48	300k	Fitness and alternative lifestyle
Acct_22	300k	Prominent anti-Brexit account
Acct_34	2.4m	Fox News TV host
Acct_2	300k	Writer for theAtlantic
Acct_128	350k	Writer for Washington Post
Acct_23	200k	Brexit/EU affairs journalist
Acct_12	280k	Conservative commentator
Acct_8	350k	Entrepreneur and 'thought leader'
Acct_29	280k	MSNBC host

Table 3: Brief description of some of the accounts mentioned.

8 Discussion

Identifying clusters: The human-interpretable rules that PROS outputs allow us to identify clusters of bot activity, which helps with evaluation. Individual requests from a legacy version like Chrome34.0.1847 might not seem strange, but accounting for 18.0% of Chrome traffic years after being superseded (as found in Section 6) seems too much for coincidence. A single user with a Twitter screen_name suffix with two random lower-case letters and two random digits is not remarkable, but 4.7k of them, all created in 2013, all using the same client, and all following the same account (as found in Section 7.2.2) is powerful evidence of co-ordinated activity. Similarly, when 82% of an account's 200k followers have creation dates in a single four-month period (as found in Section 7.2.3) it is hard to reconcile with the view that these are the results of independent actions. Identifying clusters with related properties or behavior has been exploited by many bot detection approaches [11, 13, 35, 36, 38, 39].

Limitations: We point out two limitations of PROS: a) it needs a lot of data, b) the base rate of abuse must be high enough to exceed the confidence intervals around our estimates of the clean distributions.

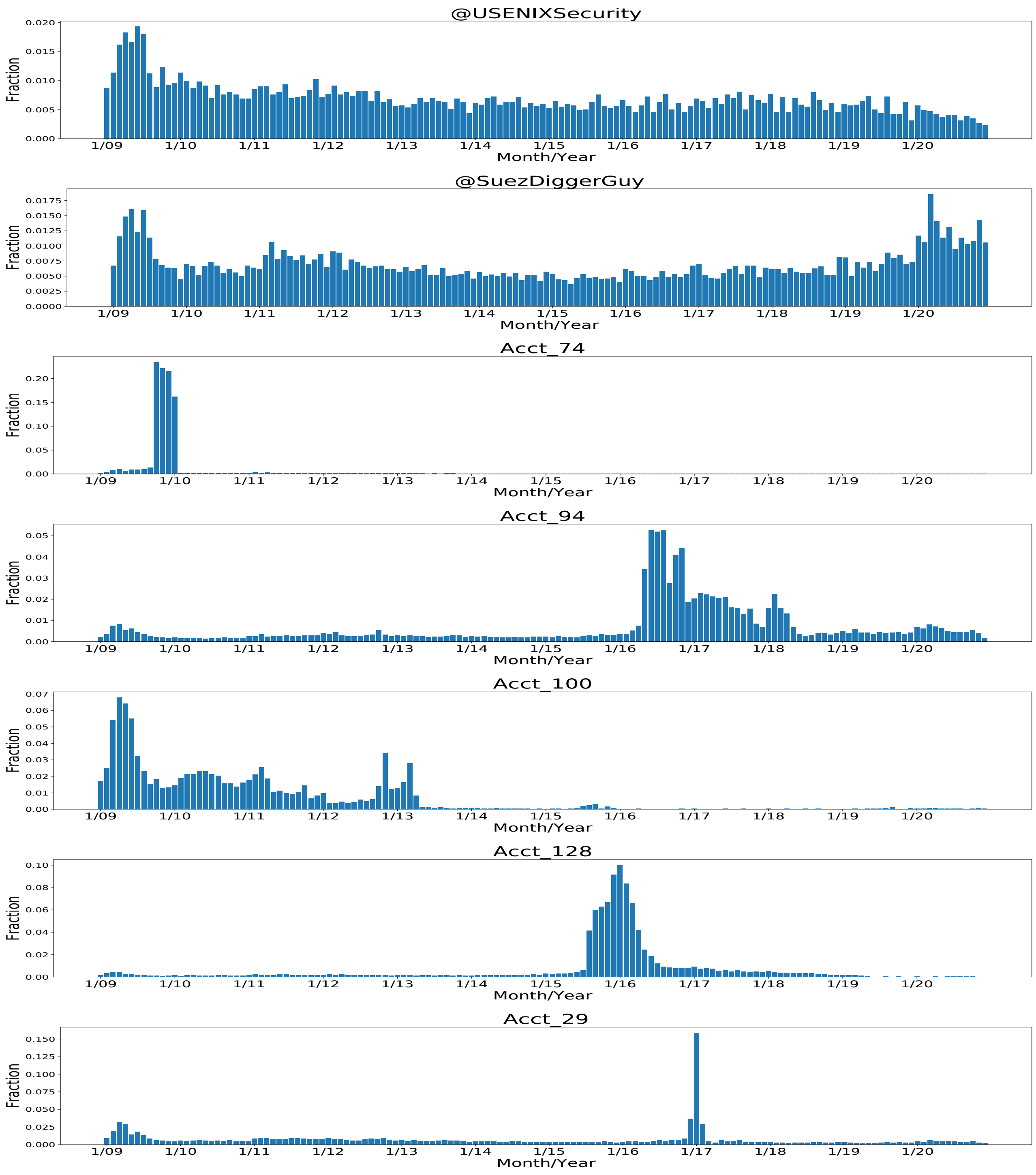

Figure 4: Two accounts with expected and five with unexpected distributions of follower creation dates. See Table 3 for a description of anonymized accounts. (a) Followers of @USENIXsecurity. (b) Followers of @SuezDiggerGuy. The fact that @SuezDiggerGuy and @USENIXsecurity have similar distributions suggests that surges of interest do not lead to surges in follower creation dates. (c) More than 82% of the 200k followers of Acct_74 joined in the four month period Oct. 2009 to Jan. 2010. (d) Near-identical volumes of followers in a three-month period in 2016 are 25× higher than the same months in 2015 for this Fox TV host. (e) Fewer than 4% of followers of this active anti-vaccine account were created later than June 2013. (f) More than 70% of followers of this Washington Post political writer were created between Aug. 2015 and May 2016. (g) More than 16% of followers of this MSNBC correspondent were created in a single month.

First, recall that we estimate the clean distribution of x_j using (7): that is we restrict to the unattacked buckets $\boldsymbol{\mu}(\boldsymbol{\lambda}(x_j))$ of features conditionally independent of x_j. We are thus using a small portion of the data to estimate clean. If x_j is browser version, one of the elements, x_k, of $\boldsymbol{\lambda}(x_j)$ might be state and if $\boldsymbol{\mu}(\boldsymbol{\lambda}(x_j)) = \{\text{Iowa, Georgia, Oregon}\}$ then the confidence intervals we can expect in our estimate of $P(x_j|\overline{\text{bot}})$ depends on the collective amount of data from those states.

Second, observe that we identify abuse traffic using (8). This punishes abuse traffic when the observed traffic (i.e., clean + abuse) exceeds what we expected (from our estimate of clean). This requires that the volume of abuse traffic is large enough to exceed the confidence intervals around our estimates of clean. The more data we have the tighter our confidence intervals will be; however very small amount of abuse will be hard to distinguish from random fluctuation.

Avoidance mechanisms: The significant assumptions of PROS are that our conditional independence relations hold, and that abuse traffic does not precisely match clean. Once clean distributions are found PROS punishes statistically significant deviations. The independence relations are properties of the benign traffic: nothing that attacker can do affects whether we chose correctly. Thus, the main avoidance mechanisms require sending abuse traffic that resembles clean as much as possible.

Bot detection as frontend to analytics: The utility of detecting bot traffic is not limited to making block/no-block decisions. Traffic analysis is important to understand how a service is used and to evaluate potential changes. Obviously, it is hard to have confidence in judgements if the human traffic we wish to analyze is contaminated by unknown, and wildly fluctuating, amounts of scripted traffic. Scenarios where we wish to improve the quality of data input may have very different expectations of a classifier than one that will block customer traffic. For example, an operating point of $(t_p, f_p) = (0.8, 0.05)$ would be unacceptable for most consumer services (i.e., falsely blocking 5% of benign customer requests) but delivers an excellent SNR boost prior to data analysis.

9 Conclusion

We have described a method to identify abuse in traffic. Our approach hinges on the observation that if we can identify bins of a feature x_j that receive no attack traffic then we can estimate the benign distribution of any feature that is independent of x_j. The key contribution is to show how these unattacked bins can be located simply and reliably.

Acknowledgements: The author thanks the anonymous reviewers for their comments; especially a particular reviewer whose very detailed remarks improved the paper greatly. He thanks Alexandra Savelieva, Fang Tu, Jay Pillai, and Kevin Qi for extensive feedback and discussions. He thanks Mike Sconzo for making the logs studied in Section 6 publicly available, and the two anonymous individuals who pointed him at Acct_80 and Acct_159.

References

[1] Mozilla HTTP 418 response. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/418.

[2] Twitter API. https://developer.twitter.com/en/docs/twitter-api.

[3] AHN, L., BLUM, M., HOPPER, N., AND LANGFORD, J. CAPTCHA: Using hard AI problems for security. In *Proceedings of the 22nd International Conference on Theory and Applications of Cryptographic Techniques* (2003), Springer-Verlag, pp. 294–311.

[4] APACHE. Combined Log Format. https://httpd.apache.org/docs/2.4/logs.html.

[5] BENEVENUTO, F., MAGNO, G., RODRIGUES, T., AND ALMEIDA, V. Detecting Spammers on Twitter. In *Collaboration, Electronic Messaging, Anti-abuse and Spam conference (CEAS)* (2010), vol. 6, p. 12.

[6] BONNEAU, J. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy* (2012), IEEE, pp. 538–552.

[7] BOURBAKI, N. *Elements of Mathematics: Algebra*. Springer, 2003.

[8] BURSZTEIN, E., AIGRAIN, J., MOSCICKI, A., AND MITCHELL, J. C. The end is nigh: Generic solving of text-based captchas. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)* (2014).

[9] BURSZTEIN, E., MARTIN, M., AND MITCHELL, J. Text-based CAPTCHA strengths and weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security* (2011), pp. 125–138.

[10] C. DWORK AND M. NAOR. Pricing via Processing or Combatting Junk Mail. *Crypto* (1992).

[11] CAO, Q., YANG, X., YU, J., AND PALOW, C. Uncovering Large Groups of Active Malicious Accounts in Online Social Networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), pp. 477–488.

[12] CHANDOLA, V., BANERJEE, A., AND KUMAR, V. Anomaly Detection: A Survey. *ACM Computing Surveys (CSUR) 41*, 3 (2009), 15.

[13] CHAVOSHI, N., HAMOONI, H., AND MUEEN, A. Debot: Twitter Bot Detection via Warped Correlation. In *ICDM* (2016), pp. 817–822.

[14] CHAWLA, N. V., BOWYER, K. W., HALL, L. O., AND KEGELMEYER, W. P. SMOTE: Synthetic Minority Over-Sampling Technique. *Journal of Artificial Intelligence Research 16* (2002), 321–357.

[15] Chio, C., and Freeman, D. *Machine Learning and Security*. O'Reilly Media, 2018.

[16] Cover, T., and Thomas, J. *Elements of Information Theory*. Wiley, 1991.

[17] Davis, C. A., Varol, O., Ferrara, E., Flammini, A., and Menczer, F. Botornot: A system to evaluate social bots. In *Proceedings of the 25th international conference companion on world wide web* (2016), pp. 273–274.

[18] Doan, X. V., and Vavasis, S. Finding Approximately Rank-One Submatrices with the Nuclear Norm and L_1-Norm. *SIAM Journal on Optimization 23*, 4 (2013), 2502–2540.

[19] Douceur, J. R. The Sybil Attack. In *International Workshop on Peer-to-peer Systems* (2002), Springer, pp. 251–260.

[20] Emmott, A. F., Das, S., Dietterich, T., Fern, A., and Wong, W.-K. Systematic Construction of Anomaly Detection Benchmarks from Real Data. In *Proceedings of the ACM SIGKDD workshop on Outlier Detection and Description* (2013), pp. 16–21.

[21] Ferrara, E., Varol, O., Davis, C., Menczer, F., and Flammini, A. The Rise of Social Bots. *Communications of the ACM 59*, 7 (2016), 96–104.

[22] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. A Survey on Concept Drift Adaptation. *ACM Computing Surveys (CSUR) 46*, 4 (2014), 44.

[23] Gantmacher, F. R. *The Theory of Matrices. 1 (1960)*. Chelsea, 1960.

[24] Grier, C., Thomas, K., Paxson, V., and Zhang, M. @ spam: the Underground on 140 Characters or Less. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), pp. 27–37.

[25] Hastie, T., Tibshirani, R., and Friedman, J. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc, 2001.

[26] Herley, C., and Schechter, S. Distinguishing attacks from legitimate authentication traffic at scale. *Network Distributed Systems Security Symp. (NDSS)* (2019).

[27] Huang, Z. Extensions to the k-means Algorithm for Clustering Large Data Sets with Categorical Values. *Data Mining and Knowledge Discovery 2*, 3 (1998), 283–304.

[28] Jan, S. T., Hao, Q., Hu, T., Pu, J., Oswal, S., Wang, G., and Viswanath, B. Throwing Darts in the Dark? Detecting Bots with Limited Data using Neural Data Augmentation. In *The 41st IEEE Symposium on Security and Privacy (IEEE SP)* (2020).

[29] Kelly, M. G., Hand, D. J., and Adams, N. M. The Impact of Changing Populations on Classifier Performance. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (1999), ACM, pp. 367–371.

[30] Kudugunta, S., and Ferrara, E. Deep Neural Networks for Bot Detection. *Information Sciences 467* (2018), 312–322.

[31] Liu, F. T., Ting, K. M., and Zhou, Z.-H. Isolation Forest. In *2008 Eighth IEEE International Conference on Data Mining* (2008), IEEE, pp. 413–422.

[32] Schölkopf, B., Williamson, R. C., Smola, A. J., Shawe-Taylor, J., Platt, J. C., et al. Support Vector Method for Novelty Detection. In *NIPS* (1999), vol. 12, Citeseer, pp. 582–588.

[33] Sommer, R., and Paxson, V. Outside the Closed World: On using Machine Learning for Network Intrusion Detection. In *Security and Privacy (SP), 2010 IEEE Symposium on* (2010), IEEE, pp. 305–316.

[34] Stringhini, G., Kruegel, C., and Vigna, G. Detecting Spammers on Social Networks. In *Proceedings of the 26th Annual Computer Security Applications Conference* (2010), pp. 1–9.

[35] Stringhini, G., Mourlanne, P., Jacob, G., Egele, M., Kruegel, C., and Vigna, G. {EVILCOHORT}: Detecting communities of malicious accounts on online services. In *24th {USENIX} Security Symposium ({USENIX} Security 15)* (2015), pp. 563–578.

[36] Stringhini, G., Wang, G., Egele, M., Kruegel, C., Vigna, G., Zheng, H., and Zhao, B. Y. Follow the Green: Growth and Dynamics in Twitter Follower Markets. In *Proceedings of the 2013 conference on Internet Measurement Conference* (2013), pp. 163–176.

[37] Thomas, K., Grier, C., Ma, J., Paxson, V., and Song, D. Design and Evaluation of a Real-time URL Spam Filtering Service. In *2011 IEEE Symposium on Security and Privacy* (2011), IEEE, pp. 447–462.

[38] Xiao, C., Freeman, D. M., and Hwa, T. Detecting clusters of fake accounts in online social networks. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security* (2015), pp. 91–101.

[39] Xu, T., Goossen, G., Cevahir, H. K., Khodeir, S., Jin, Y., Li, F., Shan, S., Patel, S., Freeman, D., and Pearce, P. Deep entity classification: Abusive account detection for online social networks. In *30th {USENIX} Security Symposium ({USENIX} Security 21)* (2021).