Detecting Clusters of Fake Accounts in Online Social Networks

Cao Xiao
University of Washington and LinkedIn Corporation
xiaoc@uw.edu

David Mandell Freeman
LinkedIn Corporation
dfreeman@linkedin.com

Theodore Hwa
LinkedIn Corporation
thwa@linkedin.com

ABSTRACT

Fake accounts are a preferred means for malicious users of online social networks to send spam, commit fraud, or otherwise abuse the system. A single malicious actor may create dozens to thousands of fake accounts in order to scale their operation to reach the maximum number of legitimate members. Detecting and taking action on these accounts as quickly as possible is imperative in order to protect legitimate members and maintain the trustworthiness of the network. However, any individual fake account may appear to be legitimate on first inspection, for example by having a real-sounding name or a believable profile.

In this work we describe a scalable approach to finding groups of fake accounts registered by the same actor. The main technique is a supervised machine learning pipeline for classifying *an entire cluster* of accounts as malicious or legitimate. The key features used in the model are statistics on fields of user-generated text such as name, email address, company or university; these include both frequencies of patterns *within* the cluster (e.g., do all of the emails share a common letter/digit pattern) and comparison of text frequencies across the *entire* user base (e.g., are all of the names rare?).

We apply our framework to analyze account data on LinkedIn grouped by registration IP address and registration date. Our model achieved AUC 0.98 on a held-out test set and AUC 0.95 on out-of-sample testing data. The model has been productionalized and has identified more than 250,000 fake accounts since deployment.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval - Spam; I.2.6 [**Artificial Intelligence**]: Learning

AISec'15, October 16, 2015, Denver, Colorado, USA.

DOI: http://dx.doi.org/10.1145/2808769.2808779.

Keywords

Spam detection, fake profiles, machine learning, data mining, clustering, classification

1. INTRODUCTION

Today people around the world rely on online social networks (OSNs) to share knowledge, opinions, and experiences; seek information and resources; and expand personal connections. However, the same features that make OSNs valuable to ordinary people also make them targets for a variety of forms of abuse. For example, the large audience on a single platform is a prime target for spammers and scammers, and the trustworthiness of the platform may make the targets more amenable to falling for scams [2]. The "gamification" aspects of a site (e.g., "Like" or "Follow" counters) lend to bots engaging in artificial actions to illegitimately promote products or services [33, 36]. Details about connections can be used to extract valuable business information [19]. And the large amount of member data available is enticing to scrapers who wish to bootstrap their own databases with information on real people [30]. According to statistics provided by the security firm Cloudmark, between 20% and 40% of Facebook accounts could be fake profiles [11]; Twitter and LinkedIn also face fake account problems to varying degrees [13, 21].

Regardless of the particular motivations for creating fake accounts, the existence of large numbers of fake accounts can undermine the value of online social networks for legitimate users. For example, they can weaken the credibility of the network if users start to doubt the authenticity of profile information [20]. They can also have negative impact on the networks' ad revenue, since advertisers might question the rates they pay to reach a certain number of users if many of them are not real people.

Yet fake accounts are hard to detect and stop. A large-scale OSN may have millions of active users and billions of user activities, of which the fake accounts comprise only a tiny percentage. Given this imbalance, false positive rates must be kept very low in order to avoid blocking many legitimate members. While some fake accounts may demonstrate clear patterns of automation, many are designed to be indistinguishable from real ones. Security measures such as CAPTCHAs and phone verification via SMS have been designed to interrogate suspicious accounts and hence raise the barrier to creating fake accounts. However, the OSN must still select a subset of accounts to challenge (since challenging all accounts would place undue friction on the experience of real users), and once faced with these challenges, spam-

mers can either solve the challenges using CAPTCHA farms or SIM card farms [7], or they can use the feedback to learn how to avoid the fake account classifier [16].

While there has been a good deal of research on detecting fake accounts (see Section 6), including some using machine learning algorithms, the literature still suffers from the following gaps:

1. None of the existing approaches perform fast detection of *clusters* of fake accounts. Most published fake account detection algorithms make a prediction for each account [1, 26, 31, 36]. Since a large-scale OSN may register hundreds of thousands of new accounts per day and bad actors try to create accounts at scale, it is more desirable to have a cluster-level detection algorithm that can perform fast, scalable detection and catch all accounts in a cluster at once.

2. None of the existing approaches are designed to detect and take action on fake accounts *before* they can connect with legitimate members, scrape, or spam. Existing algorithms for fake account detection are in general based on the analysis of user activities and/or social network connections [10,17,27,38,39], which means the fake accounts should be allowed to stay in the network for a while in order to develop connections and accumulate enough activity data. In practice we want to catch fake accounts as soon as possible after they are registered in order to prevent them from interacting with real users. This creates a challenge since we will only have some basic information provided during the registration flow. Hence an algorithm that can capture as many patterns as it can, based on very limited profile information, becomes an urgent need.

1.1 Our Contribution

In this work, we develop a scalable and time-sensitive machine learning approach to finding groups of fake accounts registered by the same actor. Our apprach solves the challenges described above as follows:

1. The first step in our pipeline is to group accounts into clusters, and our machine learning algorithms take as input *cluster-level* features exclusively. All of our features are engineered to describe the whole cluster rather than individual accounts, and the resulting classification is on entire clusters. Our approach is scalable to OSNs that have large amounts of daily account registrations.

2. Our algorithms use only features available at registration time or shortly thereafter. In particular, we do not require graph data or activity data. However, since the raw data available at registration time is limited, we must cleverly construct features that will enable us to distinguish good clusters from bad clusters. In Section 4 we describe three classes of features that allow us to achieve this goal. We also propose generic *pattern encoding algorithms* that allow us to collapse user- generated text into a small space on which we can compute statistical features.

We implemented our framework as an offline machine learning pipeline in Hadoop. The pipeline is comprised of three components: the *Cluster Builder*, which produces clusters of accounts to score; the *Profile Featurizer*, which extracts features for use in modeling; and the *Account Scorer*, which trains machine learning models and evaluates the models on new imput data. Details of our pipeline can be found in Section 3.

1.2 Experimental Results

We evaluated our approach on LinkedIn account data. For training data we sampled approximately 275,000 accounts registered over a six-month period, of which 55% had been labeled as fake or spam by the LinkedIn Security team.[1] We grouped account-level labels into cluster-level labels for training our classifiers.

We trained models using random forest, logistic regression, and support vector machine classifiers. We evaluated the classifiers' performance with 80-20 split in-sample testing and out-of-sample testing with a more recent data set. The latter test is a better approximation of real-life performance, since models are trained on data from the past and run on data from the present.

To measure the classifiers' performance, we computed AUC (area under the ROC curve) and recall at 95% precision. In practice the desired precision rates and thresholds for classification may be higher or lower depending on business needs and the relative cost of false positives and false negatives. We found that the random forest algorithm provided the best results for all metrics. On the held-out test set, the random forest model produced AUC 0.98 and recall 0.90 at 95% precision. When run on out-of-sample testing data the random forest model again performed best, with AUC 0.95 and recall 0.72 at 95% precision.

1.3 Organization of the Paper

The rest of the paper is organized as follows. In Section 2 we provide an overview of the supervised learning methods that we use in our study, along with metrics for model evaluation. In Section 3 we describe the machine learning pipeline used to implement our system, and in Section 4 we describe our approach to feature engineering. Next in Section 5 we provide results of our experiment on one embodiment of the proposed approach, describing the performance on testing data as well as results on live LinkedIn data. We discuss related work in Section 6, and we consider future directions in Section 7.

2. TRAINING METHODOLOGIES

2.1 Supervised Learning Methods

During model training, our goal is to construct and select subsets of features that are useful for building a good predictor. In our experiments, we considered the following three regression methods: logistic regression with L1 regularization [34], support vector machine with a radial basis function kernel [15], and random forest [4], a nonlinear tree-based ensemble learning method.

Logistic Regression. Given a set $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ of m training samples with $x^{(i)}$ as feature inputs and $y^{(i)} \in \{0, 1\}$

[1]Note that this sample is not representative of the LinkedIn member base, but rather is a sampling of accounts that had been flagged as suspicious for some reason.

as labels, logistic regression can be modeled as

$$p(y = 1|x, \theta) = \frac{1}{1 + \exp(-\theta^T x)}, \quad (1)$$

where $\theta \in \mathbb{R}^n$ are the model parameters.

Without regularization, logistic regression tries to find parameters using the maximum likelihood criterion, while with regularization, the goal is to control the tradeoff between fitting and having fewer variables being chosen in the model. In our study, we use L_1 penalization to regularize the logistic regression model. This technique maximizes the probability distribution of the class label y given a feature vector x, and also reduces the number of irrelevant features by using a penalty term to bound the coefficients θ in the L_1 norm. The model parameters $\theta \in \mathbb{R}^n$ are computed as

$$\arg\min_{\theta} \quad \sum_{i=1}^{m} -\log p(y^{(i)}|x^{(i)}, \theta) + \beta|\theta|_1. \quad (2)$$

In this formulation, β is the regularization parameter and will be optimally chosen using cross-validation.

Support Vector Machine. The second learning algorithm we consider is the support vector machine (SVM) [3,9,29,35]. The support vector machine algorithm looks for an optimal hyperplane as a decision function in a high-dimensional space.

Our training dataset again consists of pairs $(x^{(i)}, y^{(i)}) \in \mathbb{R}^n \times \{0, 1\}$. In our study, since we would like to use a nonlinear classifier, we used SVM with a radial basis function (RBF) kernel in training. The RBF kernel can be formulated as $k(x, x') = \exp(-r\|x - x'\|^2)$. The hyperparameter r is called the *kernel bandwidth* and is tuned based on results of cross-validation.

In principle, the SVM algorithm first maps x into a higher dimensional space via a function ψ, then finds a hyperplane H in the higher-dimensional space which maximizes the distance between the point set $\psi(x_i)$ and H. If this hyperplane is $\langle w, X\rangle = b$ (where X is in the higher- dimensional space), then the decision function is $f(x) = \langle w, \psi(x)\rangle - b$. The sign of $f(x)$ gives the class label of x. In practice, the function ψ is implicit and all calculations are done with the kernel k.

In our experiments, we adopt a probability model for classification using the R package "e1071" [25]. The decision values of the binary classifier are fitted to a logistic distribution using maximum likelihood to output numerical scores indicating probabilities. While we could just as easily use the raw SVM scores for classification, mapping the scores to probabilities allows us to compare SVM results with other models that output probability estimates.

Random Forest. The random forest algorithm [4] is an ensemble approach that combines many weak classifiers (decision trees) to form a strong classifier (random forest). For each decision tree, we first sample with replacement from the original training set to get a new training set of the same size. Then at each node of the decision tree, we choose m features at random, and split the decision tree according to the best possible split among those m features. The value of m needs to be chosen to balance the strength of individual trees (higher m is better) against the correlation between trees (lower m is better). Now given a new sample, the resulting model scores it by running the sample through all trees, and then combining the results; in the case of a binary classification problem like ours, the score is simply the percentage of trees that give a positive result on the sample.

2.2 Evaluation Metrics

All three of our classifiers output real number scores that can be used to order the samples in a test set. To measure the classifiers' performance, we calculate the AUC (area under the ROC curve), precision, and recall. We can calculate each metric either on the cluster level or on the account level, where each account is assigned the score output by the classifier for its parent cluster.

The area under the receiver operating characteristic curve (AUC) is commonly used in model comparison and can be interpreted as the probability that the classifier will assign a higher score to a randomly chosen positive example than to a randomly chosen negative example. A model with higher AUC is considered a better model. Advantages of AUC as a metric are that it doesn't require choosing a threshold for assigning labels to scores and that it is independent of class bias in the test set.

Precision and recall are well known metrics for binary classification. In our application, precision is the fraction of predicted fake accounts that are truly fake, while recall is the fraction of fake accounts in the wild that are caught by the model. For a classifier that outputs a score or probability, precision and recall can be calculated for each score threshold, giving a parametric curve. Since false positives in a fake account model are very costly, our metric of choice for model evaluation is recall rate at the threshold that produces 95% precision. (The 95% rate is merely for baselining; in practice we aim for much higher precision.)

3. MACHINE LEARNING PIPELINE

To make the proposed fake account detection system scalable, we designed and implemented a practical machine learning pipeline involving a sequence of data pre-processing, feature extraction, prediction and validation stages. The pipeline consists of three major components, which we describe below and illustrate in Figure 1.

3.1 Cluster Builder

The Cluster Builder, as its name implies, takes the raw list of accounts and builds clusters of accounts along with their raw features. The module takes user-specified parameters for (1) minimum and maximum cluster size; (2) time span of accounts registered (e.g. last 24 hours, last week), and (3) clustering criteria. The clustering criteria can be as simple as grouping all accounts that share a common characteristic such as IP address, or a more complex clustering algorithm such as k-means. Once the initial clusters are built, user-defined criteria can be added to filter out some of the clusters that are not likely to be suspicious or may introduce high false positives. For example, one may wish to filter out accounts registered from the OSN's corporate IP space, as these are likely to be test accounts that should not be restricted.

The Cluster Builder takes raw member profile tables as input and outputs a table of accounts with features that are needed for feature engineering, such as member's name, company, and education. Each row of the table represents one account and contains a "cluster identifier" unique to that account's cluster. This table is used as input to the Profile Featurizer.

In the training phase the Cluster Builder must also use account-level labels to label each cluster as real or fake. While most clusters have either all accounts or no accounts

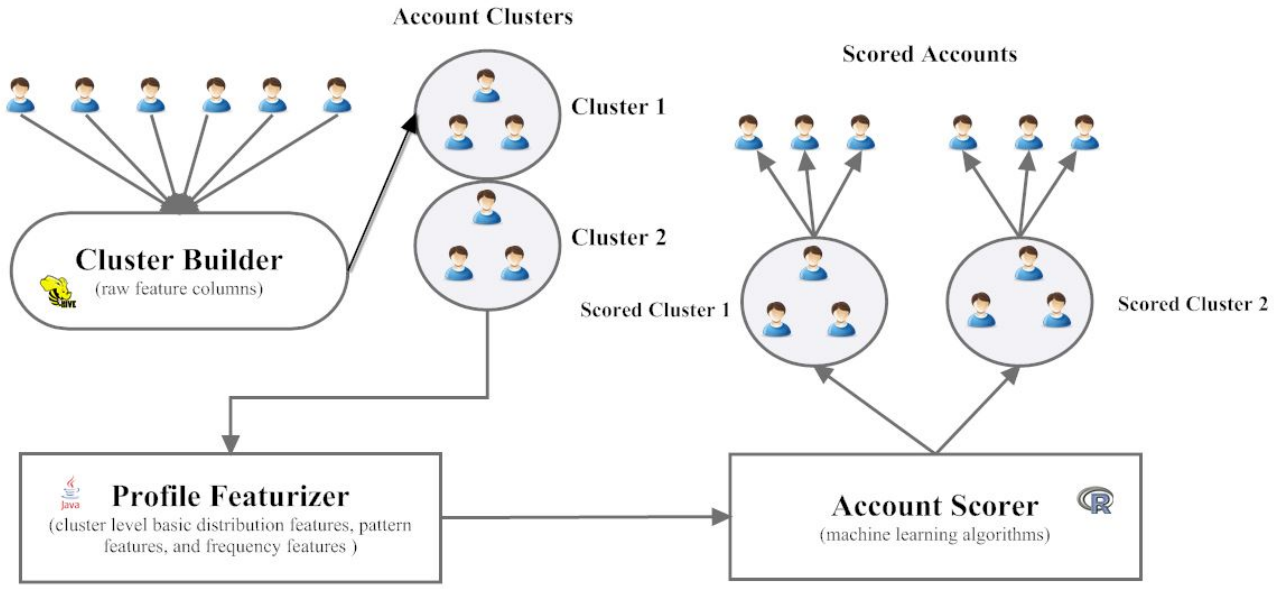

Figure 1: Our learning pipeline implementing the fake account clusters detection approach. We assemble accounts into clusters, extract features, train or evaluate the model, and assign scores to the accounts in each cluster.

labeled as fake, there will in general be a few clusters with some accounts in each group. Thus to compute cluster labels, we choose a threshold x such that the clusters with fewer than x percent fake accounts are labeled real and those with greater than x percent fake are labeled fake. The optimal choice of x depends on precision/recall tradeoffs (i.e., higher values of x increase precision at the expense of recall). However, as discussed in Section 5.2 below, in practice we find that the model is fairly insensitive to this choice.

3.2 Profile Featurizer

The Profile Featurizer is the key component of the pipeline. Its purpose is to convert the raw data for each cluster (i.e. the data for all of the individual accounts in the cluster) into a single numerical vector representing the cluster that can be used in a machine learning algorithm. It is implemented as a set of functions designed to capture as much information as possible from the raw features in order to discriminate clusters of fake accounts from clusters of legitimate accounts.

The extracted features can be broadly grouped into three categories, which we describe at a high level here; further details can be found in Section 4.

1. **Basic distribution features.** For each cluster, we take basic statistical measures of each column (e.g. company name). Examples include mean or quartiles for numerical features, or number of unique values for text features.

2. **Pattern features.** We have designed "pattern encoding algorithms" that map user-generated text to a smaller categorical space. We then take basic distribution features over these categorical variables. These features are designed to detect malicious users (especially bots) that are following a pattern in their account signups.

3. **Frequency features.** For each feature value, we compute the frequency of that value over the entire account database. We then compute basic distribution features over these frequencies. In general we expect clusters of legitimate accounts to have some high-frequency data and some low-frequency data, while bots or malicious users will show less variance in their data frequencies; e.g., using only common or only rare names.

3.3 Account Scorer

The Account Scorer's function is to train the models and evaluate them on previously unseen data. The Account Scorer takes as input the output of Profile Featurizer; i.e., one numerical factor for each cluster. The specific learning algorithm used is user-configurable; in our experiments we consider logistic regression, random forests, and support vector machines. In "training mode," the Account Scorer is given a labeled set of training data and outputs a model description as well as evaluation metrics that can be used to compare different models. In "evaluation mode," the Account Scorer is given a model description and an input vector of cluster features and outputs a score for that cluster indicating the likelihood of that cluster being composed of fake accounts.

Based on the cluster's score, accounts in that cluster can be selected for any of three actions: automatic restriction (if the probability of being fake is high), manual review (if the results are inconclusive), or no action (if the probability of being fake is low). The exact thresholds for selecting between the three actions are configured to minimize false positives and give human reviewers a mix of good and bad

accounts. The manually labeled accounts can later be used as training data in further iterations of the model.

4. FEATURE ENGINEERING

The quality of the numerical features output by the Profile Featurizer is the single most important factor in the effectiveness of our classifiers. We now describe this process in greater detail.

4.1 Basic Distribution Features

We began with a manual survey of clusters of fake accounts in the LinkedIn data set (see Section 5 for details) that had already been detected and labeled as fake. We find that accounts within a large cluster generally show patterns in their user-entered data such as name, company, or education. Sometimes they may be obvious; for example, all accounts may use identical text for the description of their current position. Such a pattern can be captured by what we term a *basic distribution feature*, in this case the feature being the number of unique position descriptions. The basic distribution features we consider include the following:

- For numerical features:
  - Min, max, and quartiles.
  - Mean and variance.
- For categorical features:
  - Number of distinct feature values in the cluster (both raw count and as a fraction of cluster size).
  - Percentage of null values (i.e. empty fields).
  - Percentage of values belonging to the mode.
  - Percentage of values belonging to the top two feature values.
  - Percentage of values that are unique.
  - Numerical features (see above) on the array of value counts.
  - Entropy, computed as $\sum_i -p_i \log(p_i)$, where i ranges over the feature values and

$$p_i = \frac{\text{number of instances of } i}{\text{number of distinct feature values}}.$$

For categorical features that take two values we can encode the values as 0/1 and compute the numerical features described above; we can also consider text fields as categorical and compute the corresponding distribution features.

4.2 Pattern features

We often find that when a single entity — whether bot or human — registers a cluster of fake accounts, the user-entered text in one or more columns always matches a certain pattern. For example, the email addresses on the accounts might appear as follows (this is a synthetic sample):

charlesgreen992@domain.com
josephbaker247@domain.com
thomasadams319@domain.com
chrisnelson211@domain.com
danielhill538@domain.com
paulwhite46@domain.com
markcampbell343@domain.com
donaldmitchell92@domain.com
georgeroberts964@domain.com
kennethcarter149@domain.com

Clearly all of these email addresses satisfy the regular expression `[a-z]+[0-9]+@domain\.com . We can apply this regular expression to the email`address to obtain a binary feature, upon which we can calculate the basic distribution features described above. In theory we could apply the techniques of Prasse et al. [28] to our training set to generate a list of spammy regular expressions and use each regular expression as binary features. However, this approach will generate a very sparse feature vector and will not generalize to unseen patterns.

Instead of relying on regular expressions, we have designed two "pattern encoding algorithms" that map arbitrary text to a smaller space. The first algorithm normalizes character classes: the universe of characters is divided into text classes such as uppercase, lowercase, digit, punctuation, etc., and each character is mapped to a representative character for the class, as described in Algorithm 1 below.

Algorithm 1 Pattern Encoding Algorithm (Length-Preserving)

```
Require: s.length > 0
procedure ENCODE(s)                          ▷ 'abc12' → 'LLLDD'
    i ← 0
    t ← ''
    while i < s.length do
        if isUpperCase(s[i]) then
            t ← t + 'U'
        else if isLowerCase(s[i]) then
            t ← t + 'L'
        else if isDigit(s[i]) then
            t ← t + 'D'
        else
            t ← t + 'O'
        end if
    end while
    return t
end procedure
```

This algorithm, which is length-preserving, would be able to detect email addresses that were all eight letters plus three digits at the same domain. However, it would not detect the list of email addresses above since the names and numbers are of varying length. To address this problem we use a length-independent variant that collapses consecutive instances of a class into a single representative, as described in Algorithm 2 below.

The output of this algorithm on the list of email usernames (i.e. the text before the @ sign) above would be "LD" in each case. Our experience shows that it is rare for a collection of legitimate users to all follow such a pattern, so this is a good feature for distinguishing clusters of accounts created by a single entity.

In addition to using the algorithm as described above, we can add new character classes such as punctuation or spaces, or we can collapse classes together (e.g. lowercase and uppercase letters). In this way simple metrics such as text length and word count can also be subsumed in the framework. Some of the patterns we used in our analysis are as follows:

- ENCODE() (Algorithm 1).
- SHORTENCODE() (Algorithm 2).

Algorithm 2 Pattern Encoding Algorithm (Length-Independent)

Require: $s.length > 0$

```
1: procedure ShortEncode(s)                ▷ 'abc12' → 'LD'
2:     i ← 0
3:     s ← Encode(s)
4:     curr ← ''
5:     t ← ''
6:     while i < s.length do
7:         if curr ≠ s[i] then
8:             t ← t + s[i]
9:             curr ← s[i]
10:        end if
11:        i ← i + 1
12:    end while
13:    return t
14: end procedure
```

- Len(Encode()) using a single character class (i.e. text length).
- Len(ShortEncode()) using two character classes, space and non-space (i.e. word count).
- Binary features checking the existence of each character class in Encode().
- Encode() on the first character of the text.

Once mapped to a smaller categorical space, we apply the basic distribution features described in Section 4.1 to compute numerical features.

4.3 Frequency features

Upon close examination of clusters of fake accounts, we often find patterns that are apparent to the trained eye but algorithmically hard to describe. For example, consider these two sets of names (again, a synthetic sample):

Cluster 1	Cluster 2
Charles Green	Shirely Lofgren
Joseph Baker	Tatiana Gehring
Thomas Adams	China Arzate
Chris Nelson	Marcelina Pettinato
Daniel Hill	Marilu Marusak
Paul White	Bonita Naef
Mark Campbell	Etta Scearce
Donald Mitchell	Paulita Kao
George Roberts	Alaine Propp
Kenneth Carter	Sellai Gauer

It's fairly apparent that these names are not randomly sampled from the population at large. The names in Cluster 1 are all common male names (in fact, they were generated by taking top-ranked first and last names from U.S. Census data) and the names in Cluster 2 are all exceedingly rare — it's possible there could be someone in the world named Bonita Naef, but the probability that she would register for a social network from the same IP address as Alaine Propp and all the others is quite low.

We quantify this intuition using data from the social network's entire member base. Specifically, for a given column of text (such as first name), we compute the frequency of that text among all of the social network's members. This gives a number between 0 and 1, on which we can compute basic distribution features as described in Section 4.1. We can do the same for the logarithms of the frequencies or the ranks of the features in the sorted list of frequencies, which can help to distinguish extremely rare entries from somewhat rare entries.

5. EXPERIMENTAL RESULTS

5.1 Data Acquisition

We evaluated our model on labeled LinkedIn account data, where the labels were provided by LinkedIn's Security and/or Trust and Safety teams. Our approach first requires us to choose a method of clustering accounts. For our study, we created clusters of LinkedIn accounts by grouping on registration IP address[2] and registration date (in Pacific Time). We chose this approach primarily because this is a grouping for which we were able to obtain a large amount of manually labeled data; in Section 7 we discuss other clustering approaches.

For our training set we collected labeled accounts from the 6-month period December 1, 2013 to May 31, 2014. During this time the accounts in all (IP, date) clusters satisfying an internal criterion for suspicious registration were sent to LinkedIn's Trust and Safety team for manual review and action. We extracted raw profile data for each account in these clusters and labeled the account as fake if it was restricted or as real if it was in good standing as of the survey time. The total number of labeled accounts was 260,644, of which 153,019 were fake accounts and 107,625 were legitimate.

In a similar fashion we obtained data from June 2014 to be used as "out-of-sample" testing data. This data included 30,550 accounts, of which 15,078 were fake accounts and 15,472 were legitimate.

5.2 Cluster Labeling

The labeled accounts in our data set fell into 20,559 distinct (IP, date) clusters. The median cluster size was 9; a histogram of the cluster sizes appears in Figure 2. For each cluster we calculated the percentage of accounts labeled as spam; a histogram of this data appears in Figure 3. We found that 89% of the clusters had either no fake accounts or all fake accounts, and only 3.8% of clusters had between 20% and 80% fake accounts.

To determine a threshold for labeling a cluster as fake (see Section 3.1), we ran our random forest classifier using cluster labels generated by setting three different thresholds: 20%, 50%, and 80%. The resulting AUC metrics at the account level were 0.9765, 0.9777, and 0.9776, respectively. We conclude that the relative ordering of scored accounts is insensitive to the cluster-labeling threshold, and we chose 50% as our threshold for further experiments. At this threshold, 10,456 of our training clusters were labeled as spam and 10,103 were labeled as legitimate. The out-of-sample test set fell into 2,705 clusters, out of which 1,227 were spam and 1,478 were legitimate.

5.3 Performance Analysis

We evaluated our proposed approach using the machine learning algorithms described in Section 2: logistic regression, SVM, and random forest. We chose these three algorithms to illustrate possible approaches; in principle any

[2]For accounts registered using IPv6 we grouped on the /56 subnet.

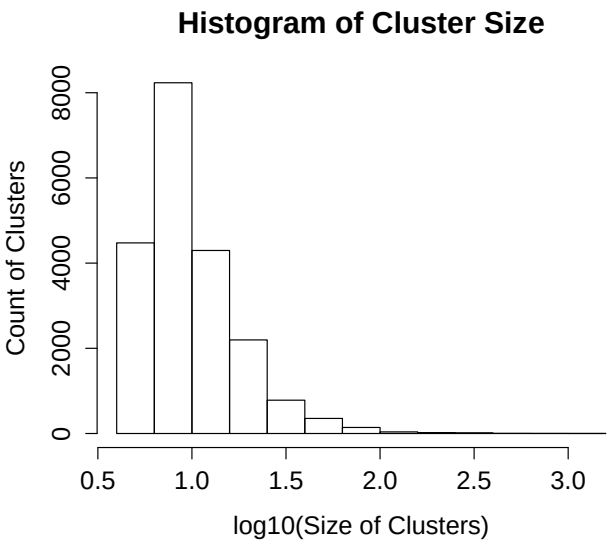

Figure 2: Distribution of cluster sizes for our training data.

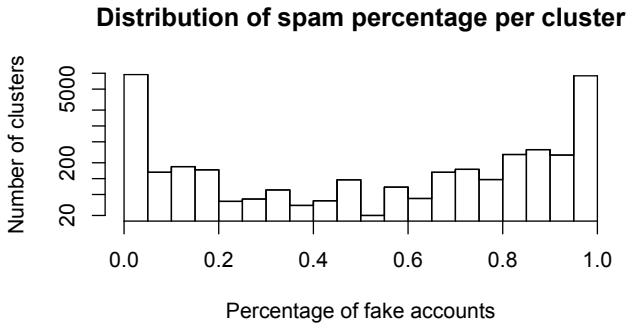

Figure 3: Distribution of spam percentage per cluster for our training data.

binary classification algorithm can be used, and the best algorithm may change depending on the domain area. To make a fair comparison and evaluation, the parameters of all supervised learning algorithms were determined through an 80-20 split cross-validation process. Specifically, 80% of the training data was used to construct the classifier and the remaining 20% of the data was used for "in-sample" performance testing. The optimal parameter setting is the setting that maximizes the in-sample testing AUC.

We ran the three aforementioned algorithms using R packages "glmnet" [14], "e1071" [25], and "randomForest" [23] on training data, respectively. Table 1 shows the in-sample prediction performance as measured by AUC and recall at 95% precision. The data show that random forest performs the best based on both metrics. The other nonlinear classifier, SVM with RBF kernel, also has good performance in terms of its AUC value. However, its recall at 95% precision is not as good as that of random forest, which indicates that for SVM, although we have high confidence in the fake accounts we catch, there are still many fake accounts not caught by the model. Among all models, logistic regression has the worst performance, because the nonlinearity in the true patterns cannot be well modeled using a linear classifier.

Table 1: 80-20 split testing performance (cluster level)

Algorithm	AUC	Recall@p95
Random forest	0.978	0.900
Logistic regression	0.936	0.657
SVM	0.963	0.837

Table 2 shows the testing AUC and recall at 95% precision for all three algorithms at the account level; that is, when each account is assigned the score computed for its cluster. The data show that our prediction for each algorithm is even more accurate for each account, and an examination of the data shows that this is caused by the classifiers being more accurate on larger clusters. (See Table 5 for further evidence.)

Table 2: 80-20 split testing performance (account level)

Algorithm	AUC	Recall@p95
Random forest	0.978	0.935
Logistic regression	0.951	0.821
SVM	0.961	0.889

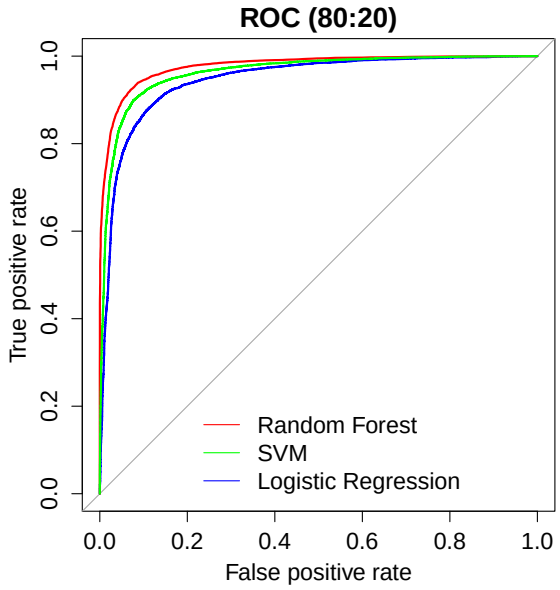

Figure 4: Comparison of ROC curves for different models on in-sample data.

We also tested our model on out-of-sample data from June, 2014. The motivation for doing out-of-sample testing is that spammers' patterns in their fake account data will vary over time as they experiment and learn from failure. Performing out-of-sample testing simulates this scenario in

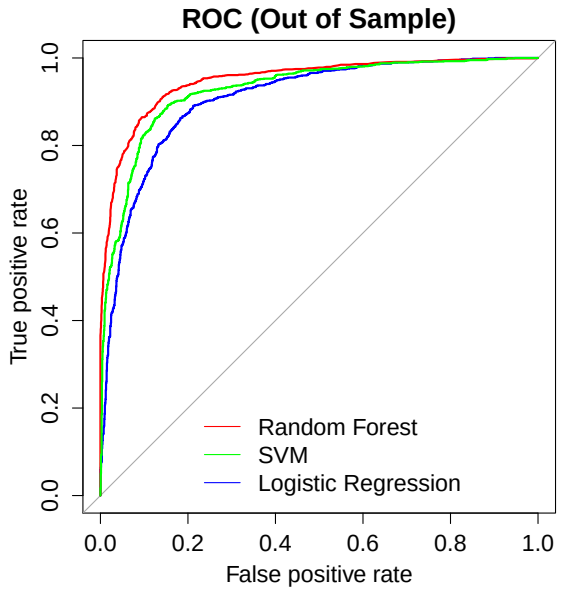

Figure 5: Comparison of ROC curves for different models on out-of-sample data.

production, and provides a practical and useful evaluation of the real-life performance of the model.

Tables 3 and 4 give the out-of-sample performance comparison of the three models trained from training data at the cluster level and account level, respectively. The data show that random forest still performs the best based on all metrics. The recall at 95% precision for all three algorithms decreases as compared with the cross-validation results, which confirms our assumption that given a certain level of precision (i.e., the fraction of predicted fake accounts that are truly fake), there are more fake accounts not being caught in the newer dataset. The results also indicate that we need to re-train our model regularly, so as to capture the newer patterns and increase the fraction of fake accounts caught.

Table 3: Out-of-sample testing performance (cluster level)

Algorithm	AUC	Recall@p95
Random forest	0.949	0.720
Logistic regression	0.906	0.127
SVM	0.928	0.522

Table 4: Out-of-sample testing performance (account level)

Algorithm	AUC	Recall@p95
Random forest	0.954	0.713
Logistic regression	0.917	0.456
SVM	0.922	0.311

One interesting finding here is that the performances of the SVM classifier *decreases* as we move from the cluster level to the account level. This result implies that unlike logistic regression and random forests, and SVM does better at classifying smaller clusters; it also suggests that an ensemble approach combining all of the classifiers may lead to still greater performance.

Analysis by Cluster Size. Table 5 shows the random forest results binned by cluster size. We see that as clusters grow bigger, the model performance improves. If a cluster has more than 30 accounts, which means on one day from an IP address there were more than 30 accounts signed up, we have almost perfect confidence to label this cluster and all accounts in the cluster. If a cluster has more than 100 accounts, then we can reach 100% accuracy level on all metrics for the cross-validation set.

Table 5: Random forest performance by cluster size

Cluster Size	AUC	Recall@p95
1 to 10	0.967	0.817
11 to 30	0.988	0.965
31 to 100	0.988	0.989
greater than 100	1.000	1.000

Analysis of Top Ranked Features. To gain more insight into the features in our study, we ranked them using the Gini importance index, which is calculated based on the Gini index [4]. In our model, the top features included average frequency counts of the two least common last or first names, as well as fraction of top patterns generated from our pattern encoding algorithms in name and email address.

5.4 False Positive and False Negative Analysis

We manually reviewed all accounts in our validation set and out-of-sample testing set that were predicted to be fake but were actually labeled as legitimate. We found that the majority of these were from organizational signups. A number of members all signed up from the same organization, which might all come from a single IP address, and some parts of their profile information may be similar. For example, their email addresses may follow a standard pattern (such as `<last name><first initial>@<organization>.org`). To address such false positives, we developed an organizational account detection model, and we configured our classifier in production so that organizational accounts that the model labels as fake are sent for manual review instead of automatically restricted. This approach helped resolve the false positive issue greatly.

We also manually reviewed all accounts in both datasets that were predicted to be legitimate but were labeled by humans as fake. In many cases, it turned out the prediction from our model was correct. Usually, if there is a legitimate mass signup (e.g., during a LinkedIn marketing event), the large number of signups will fall into a single cluster. This will likely trigger some previous rule-based model to label them as fake, and the human labeler might also label it as fake for the same reason. However, as the size of cluster grows, the account profile patterns within the cluster will become more and more diverse, which seems more normal from the model's perspective, so the model is able to correctly label the cluster as good. That also explains why as clusters grow bigger, the model becomes more and more accurate, as shown in Table 5. Where the model found errors

in the previous human label (as confirmed by our subsequent manual review), we reversed the previous decision, and relabeled those accounts as legitimate.

5.5 Running on Live Data

We implemented the system using Java, Hive, and R, and trained it on the dataset discussed in Section 5.1. Using Hadoop streaming, we ran the algorithm daily on new LinkedIn registrations. The highest-scoring accounts were automatically restricted. Scores in a "gray area" were sent to LinkedIn's Trust and Safety team for manual review and action. This process allows us to collect quality labeled data on borderline cases for training future models.

Since its rollout, the model has caught more than 15,000 clusters, comprising more than 250,000 fake LinkedIn accounts. The trend in the model's precision can be seen in Figure 6, which plots a 14-day moving average of the precision. The decrease in precision at one point is due to a large number of organizational signups that the model erroneously flagged; upon addition of the "organization detector" the precision returned to its previous levels.

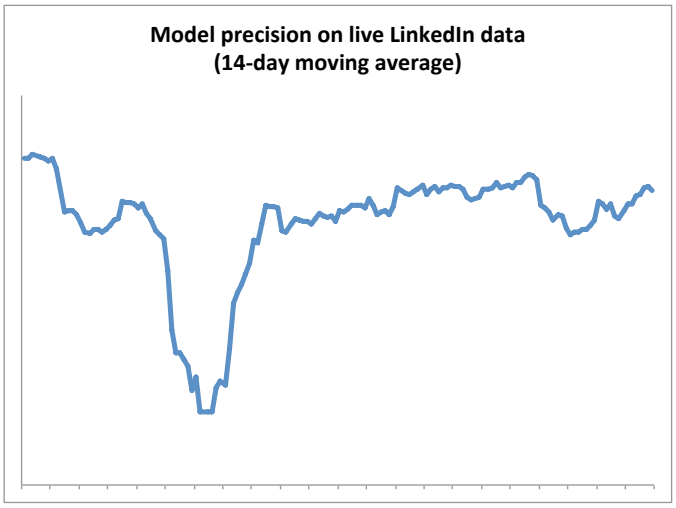

Figure 6: 14-day moving average of model precision (at the account level) since deployment.

Figure 7 shows a histogram of cluster sizes on live data and the precision (on the level of clusters) within each bucket. Most of the clusters detected were relatively small; the median cluster size was 11 accounts. Contrary to our experience with the training data, we found that precision did not in general increase with cluster size, except for the very largest clusters (of size greater than 100).

6. RELATED WORK

The problem of detecting fake accounts in online social networks has been approached from a number of different perspectives, including behavioral analysis, graph theory, machine learning, and system design.

Using a behavioral perspective, Malhotra et al. [24] develop features to detect malicious users who create fake accounts across different social networks. However, the features they propose are all account-level basic profile features. If the same spammer does not abuse different platforms us-

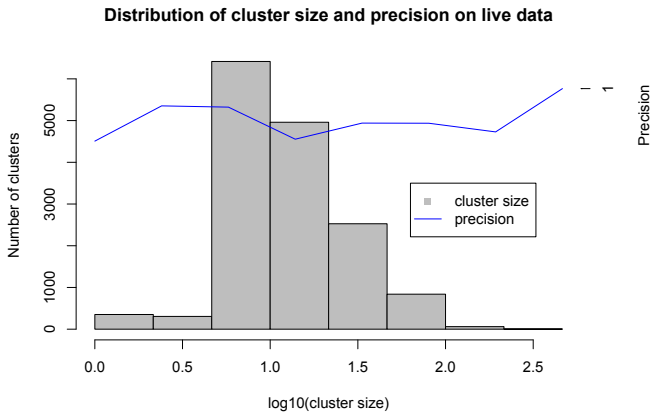

Figure 7: Distribution of cluster size and precision on live data.

ing some same basic profile information, the effect of such features would be reduced.

Much research has been done to analyze fake accounts in OSNs from a graph-theoretic perspective. Two relevant surveys are those of Yu et al. [38], who describe a number of specific sybil defense mechanisms, and Viswanath et al. [37], who point out that most existing Sybil defense schemes work by detecting local communities (i.e., clusters of nodes more tightly knit than the rest of the graph) around a trusted node.

In more recent graph-theoretic work, Jiang et al. [17] propose to detect fake accounts by constructing latent interaction graphs as models of user browsing behavior. They then compare these graphs' structural properties, evolution, community structure, and mixing times against those of both active interaction graphs and social graphs. Mohaisen et al. [27] detect Sybil nodes, which disrupt the fast mixing property of social networks, and thus propose several heuristics to improve the mixing of slow-mixing graphs using their topological structures. Conti et al. [8] analyze social network graphs from a dynamic point of view to detect adversaries who create fake profiles to impersonate real people and then interact with the real people's friends.

While the above graph-theoretic techniques may be applicable to the clusters of accounts we study in this work, our goal is to detect clusters *before* they can make the connections or engage in the behavior that produces the relevant graph structures. Thus our approach focuses on signals that are available at or shortly after registration time, which includes only a small amount of activity data and little to no connection data.

Many researchers have applied machine learning algorithms to the problem of spam detection on OSNs. Fire et al. [12] use topology anomalies, decision trees, and naive Bayes classifiers to identify spammers and fake profiles that are used in multiple social networks. Jin et al. [18] analyze the behavior of identity clone attacks and propose a detection framework. Cao et al. [5] develop a ranking algorithm to rank users in online services and detect fake accounts; this rank is calculated according to the degree-normalized probability of a short random walk in the non-Sybil region. Tan et al. [32] put the network spam detection problem in an unsupervised learn-

ing framework, deliberately removing non-spammers from the network and leveraging both the social graph and the user-link graph.

Rather than focus on detecting fake accounts after they penetrate the network, some researchers have focused on designing the systems themselves to prevent attacks in the first place. Lesniewski-Laas and Kaashoek [22] propose a novel routing protocol for distributed hash tables that is efficient and strongly resistant to Sybil attacks. Chiluka et al. [6] propose a new design point in the trade-off between network connectivity and attack resilience of social network-based Sybil defense schemes, where each node adds links to only a selective few of its 2-hop neighbors based on a minimum expansion contribution (MinEC) heuristic. Viswanath et al. [37] present a system that uses routing-based techniques to efficiently approximate credit payments over large networks.

While system-design techniques for preventing abuse can be effective, they are often not applicable in practice to a large-scale network that was originally designed to optimize for growth and engagement long before abuse became a significant issue.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a machine learning pipeline for detecting fake accounts in online social networks. Rather than making a prediction for each individual account, our system classifies clusters of fake accounts to determine whether they have been created by the same actor. Our evaluation on both in-sample and out-of-sample data showed strong performance, and we have used the system in production to find and restrict more than 250,000 accounts.

In this work we evaluated our framework on clusters created by simple grouping on registration date and registration IP address. In future work we expect to run our model on clusters created by grouping on other features, such as ISP or company, and other time periods, such as week or month. Another promising line of research is to use more sophisticated clustering algorithms such as k-means or hierarchical clustering. While these approaches may be fruitful, they present obstacles to operating at scale: k-means may require too many clusters (i.e., too large a value of k) to produce useful results, and hierarchical clustering may be too computationally intensive to classify millions of accounts.

From a modeling perspective, one important direction for future work is to apply feature sets used in other spam detection models, and hence to realize multi-model ensemble prediction. Another direction is to make the system robust against adversarial attacks, such as a botnet that diversifies all features, or an attacker that learns from failures. A final direction is to construct more language-insensitive pattern matching features; our features assume the text is written in an alphabet that can be mapped to a small number of character classes (e.g. uppercase or lowercase) and this does not readily adapt to pictographic languages such as Chinese.

8. REFERENCES

[1] S. Adikari and K. Dutta. Identifying fake profiles in LinkedIn. *Pacific Asia Conference on Information Systems Proceedings 2014*, 2014.

[2] J. Beall. Publisher uses fake LinkedIn identities to attract submissions. http://scholarlyoa.com/2015/02/10/publisher-uses-fake-linkedin-identities-to-attract-submissions.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152. ACM Press, 1992.

[4] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, Oct. 2001.

[5] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 15–15, Berkeley, CA, USA, 2012. USENIX Association.

[6] N. Chiluka, N. Andrade, J. Pouwelse, and H. Sips. Social networks meet distributed systems: Towards a robust sybil defense under churn. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 507–518, New York, NY, USA, 2015. ACM.

[7] D. B. Clark. The bot bubble: How click farms have inflated social media currency. *The New Republic*, April 20 2015. Available at http://www.newrepublic.com/article/121551/bot-bubble-click-farms-have-inflated-social-media-currency.

[8] M. Conti, R. Poovendran, and M. Secchiero. Fakebook: Detecting fake profiles in on-line social networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ASONAM '12, pages 1071–1078, Washington, DC, USA, 2012. IEEE Computer Society.

[9] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, New York, NY, USA, 2000.

[10] G. Danezis and P. Mittal. Sybilinfer: Detecting sybil nodes using social networks. Technical Report MSR-TR-2009-6, Microsoft, January 2009.

[11] Digital Trends Staff. 40 pct. fake profiles on Facebook? http://www.digitaltrends.com/computing/fake-profiles-facebook/.

[12] M. Fire, G. Katz, and Y. Elovici. Strangers intrusion detection - detecting spammers and fake profiles in social networks based on topology anomalies. *ASE Human Journal*, 1(1):26–39, Jan. 2012.

[13] D. M. Freeman. Using Naive Bayes to detect spammy names in social networks. In A. Sadeghi, B. Nelson, C. Dimitrakakis, and E. Shi, editors, *AISec'13, Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security, Co-located with CCS 2013, Berlin, Germany, November 4, 2013*, pages 3–12. ACM, 2013.

[14] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

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

[16] L. Huang, A. D. Joseph, B. Nelson, B. I. P. Rubinstein, and J. D. Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence, AISec 2011, Chicago, IL, USA, October 21, 2011*, pages 43–58, 2011.

[17] J. Jiang, C. Wilson, X. Wang, W. Sha, P. Huang, Y. Dai, and B. Y. Zhao. Understanding latent interactions in online social networks. *ACM Trans. Web*, 7(4):18:1–18:39, Nov. 2013.

[18] L. Jin, H. Takabi, and J. B. Joshi. Towards active detection of identity clone attacks on online social networks. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, CODASPY '11, pages 27–38, New York, NY, USA, 2011. ACM.

[19] P. Judge. Social klepto: Corporate espionage with fake social network accounts. https://www.rsaconference.com/writable/presentations/file_upload/br-r32.pdf.

[20] K. Lee. Fake profiles are killing LinkedIn's value. http://www.clickz.com/clickz/column/2379996/fake-profiles-are-killing-linkedin-s-value.

[21] K. Lee, B. D. Eoff, and J. Caverlee. Seven months with the devils: a long-term study of content polluters on Twitter. In *AAAI International Conference on Weblogs and Social Media (ICWSM)*, 2011.

[22] C. Lesniewski-Laas and M. F. Kaashoek. Whanau: A sybil-proof distributed hash table. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 8–8, Berkeley, CA, USA, 2010. USENIX Association.

[23] A. Liaw and M. Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[24] A. Malhotra, L. Totti, W. Meira Jr., P. Kumaraguru, and V. Almeida. Studying user footprints in different online social networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ASONAM '12, pages 1065–1070, Washington, DC, USA, 2012. IEEE Computer Society.

[25] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, and C. Chang. R package "e1071". 2014.

[26] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: Inferring user profiles in online social networks. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 251–260, New York, NY, USA, 2010. ACM.

[27] A. Mohaisen and S. Hollenbeck. Improving social network-based sybil defenses by rewiring and augmenting social graphs. In *Revised Selected Papers of the 14th International Workshop on Information Security Applications - Volume 8267*, WISA 2013, pages 65–80, New York, NY, USA, 2014. Springer-Verlag New York, Inc.

[28] P. Prasse, C. Sawade, N. Landwehr, and T. Scheffer. Learning to identify regular expressions that describe email campaigns. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[29] A. Rakotomamonjy. Variable selection using svm based criteria. *J. Mach. Learn. Res.*, 3:1357–1370, Mar. 2003.

[30] L. Ruff. Why do people create fake LinkedIn profiles? http://integratedalliances.com/blog/why-do-people-create-fake-linkedin-profiles.

[31] M. Singh, D. Bansal, and S. Sofat. Detecting malicious users in Twitter using classifiers. In *Proceedings of the 7th International Conference on Security of Information and Networks*, SIN '14, pages 247:247–247:253, New York, NY, USA, 2014. ACM.

[32] E. Tan, L. Guo, S. Chen, X. Zhang, and Y. Zhao. Unik: Unsupervised social network spam detection. In *Proceedings of the 22nd ACM International Conference on Conference on Information & Knowledge Management*, CIKM '13, pages 479–488, New York, NY, USA, 2013. ACM.

[33] K. Thomas, D. McCoy, C. Grier, A. Kolcz, and V. Paxson. Trafficking fraudulent accounts: The role of the underground market in Twitter spam and abuse. In *Proceedings of the 22nd USENIX Conference on Security*, SEC'13, pages 195–210, Berkeley, CA, USA, 2013. USENIX Association.

[34] R. Tibshirani. Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[35] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[36] B. Viswanath, M. A. Bashir, M. Crovella, S. Guha, K. P. Gummadi, B. Krishnamurthy, and A. Mislove. Towards detecting anomalous user behavior in online social networks. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 223–238, Berkeley, CA, USA, 2014. USENIX Association.

[37] B. Viswanath, M. Mondal, K. P. Gummadi, A. Mislove, and A. Post. Canal: Scaling social network-based sybil tolerance schemes. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 309–322, New York, NY, USA, 2012. ACM.

[38] H. Yu. Sybil defenses via social networks: A tutorial and survey. *SIGACT News*, 42(3):80–101, Oct. 2011.

[39] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. *IEEE/ACM Trans. Netw.*, 18(3):885–898, June 2010.