Unveiling Fake Accounts at the Time of Registration: An Unsupervised Approach

Xiao Liang[1*], Zheng Yang[2*], Binghui Wang[3,4*], Shaofeng Hu[1], Zijie Yang[2], Dong Yuan[2], Neil Zhenqiang Gong[3], Qi Li[2], Fang He[1]
[1]Tencent Inc. [2]INSC and BNRIST, Tsinghua University [3]Duke University [4]Illinois Institute of Technology

ABSTRACT

Online social networks (OSNs) are plagued by fake accounts. Existing fake account detection methods either require a manually labeled training set, which is time-consuming and costly, or rely on rich information of OSN accounts, e.g., content and behaviors, which incurs significant delay in detecting fake accounts. In this work, we propose *UFA* (*U*nveiling *F*ake *A*ccounts) to detect fake accounts immediately after they are registered in an unsupervised fashion. First, through a measurement study on the registration patterns on a real-world registration dataset, we observe that fake accounts tend to cluster on outlier registration patterns, e.g., IP and phone numbers. Then, we design an unsupervised learning algorithm to learn weights for all registration accounts and their features that reveal outlier registration patterns. Next, we construct a registration graph to capture the correlation between registration accounts, and utilize a community detection method to detect fake accounts via analyzing the registration graph structure. We evaluate *UFA* using real-world WeChat datasets. Our results demonstrate that *UFA* achieves a precision ~94% with a recall ~80%, while a supervised variant requires 600K manual labels to obtain the comparable performance. Moreover, *UFA* has been deployed by WeChat to detect fake accounts for more than one year. *UFA* detects 500*K* fake accounts per day with a precision ~93% on average, via manual verification by the WeChat security team.

CCS CONCEPTS

• **Security and privacy**; • **Computing methodologies** → **Machine learning**;

KEYWORDS

Fake account detection, unsupervised learning, graph

ACM Reference Format:
Xiao Liang, Zheng Yang, Binghui Wang, Shaofeng Hu, Zijie Yang, Dong Yuan, Neil Zhenqiang Gong, Qi Li, Fang He. 2021. Unveiling Fake Accounts at the Time of Registration: An Unsupervised Approach. In Proceedings of *the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '21), August 14–18, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3447548.3467094
*Equal Contribution

KDD '21, August 14–18, 2021, Virtual Event, Singapore

ACM ISBN 978-1-4503-8332-5/21/08…$15.00
https://doi.org/10.1145/3447548.3467094

1 INTRODUCTION

Online social networks (OSNs), e.g., Facebook, Twitter, and WeChat, are increasingly important and popular nowadays. People use those platforms to perform social activities including interacting with each other, sharing information and enjoying recreations. However, OSNs are known to be vulnerable to various attacks. In particular, an attacker registers massive fake accounts (also called Sybils) to perform various malicious activities, such as spreading spam, phishing URLs, malware, and disinformation [19] on OSNs. Therefore, it is crucial to detect such fake accounts in OSNs. Fake account detection has attracted continuing attention from multiple communities. Various methods have been developed [1, 4, 5, 7, 9–13, 16, 20, 21, 23–25, 27, 30–35] to detect fake accounts in OSNs. However, existing methods suffer from at least one of the following limitations. First, they require a labeled training set consisting of labeled benign accounts or/and labeled fake accounts. In practice, manually labeling a training set is time-consuming and costly. Alternatively, an OSN provider can outsource manual labeling to crowdsourcing services such as Amazon Mechanical Turk [29]. However, attackers could act as "turkers" to adversarially mislabel accounts. Thus, incorrect labels will be injected into the training set such that existing methods relying on accurate labels would have significant detection performance degeneration. Second, they detect fake accounts at a severe time delay. Specifically, these methods require fake accounts to generate rich account information such as content, behavior, and/or social graphs in advance for fake account detection. However, fake accounts may have already performed various malicious activities before being detected.

In this work, we design *UFA*, an unsupervised method, to detect fake accounts at the time of registration. *UFA* can overcome limitations of existing fake account detection methods. Specifically, *UFA* consists of four components: *feature extraction*, *unsupervised weight learning*, *registration graph construction*, and *fake account detection*. In order to effectively extract features for fake account detection, we first perform a measurement study on a real world registration dataset from WeChat, the largest OSN in China. In the dataset, each registration has a list of attributes such as the IP address, the phone number, the device ID, and the OS version. Our measurement results show that, compared with benign accounts, fake accounts tend to cluster on outlier registration patterns, e.g., fake accounts are likely to use the same IPs, use the same phone numbers from the same areas, be active at midnight, or use outdated WeChat version and OS version. Thus, we extract features that reveal outlier registration patterns for fake accounts.

In unsupervised weight learning, we first construct a registration-feature bigraph to capture the relationship between registration accounts and features. Specifically, we represent each account and each feature as a node. We add an edge between an account and a

feature if the account has the feature. Second, we design a statistical method to initialize the weight of each feature node, the weight of each registration account node based on the weights of feature nodes. Note that, our statistical method only uses the registration patterns and does not require labeled fake accounts or/and labeled benign accounts. In our method, the weight of node quantifies the node's anomaly. The initial weights do not consider the relationship between features and between registration accounts. To address this issue, we leverage a structure-based method, called linearized belief propagation [27], to propagate the initial weights of feature nodes and registration nodes among the bigraph and iteratively update each node's weight, where weight of a registration account indicates its probability of being fake.

One key challenge in unsupervised weight learning is that the correlation between registration accounts is not directly modeled, since there are no edges between registration accounts in the registration-feature bigraph. We further design a registration graph construction component to directly capture the correlation between registration accounts. In particular, we map the registration-feature bigraph into a weighted registration graph, where each node represents a registration account. An edge between two registration accounts is created only if they have high similarity in the registration-feature bigraph, and the similarity is also used to indicate the weight of the edge. Then, fake accounts are likely to be densely connected, while benign accounts are likely to be sparsely connected in the registration graph. Due to that fake accounts are densely connected, we develop a community detection algorithm to cluster the registration accounts into communities in the fake account detection component. We treat all accounts in a community as fake accounts if the community size is larger than a threshold.

We evaluate *UFA* using real-world WeChat registration datasets and demonstrate that *UFA* is effective. For instance, it can detect a large fraction of fake accounts (recall ~80%) with a high precision (~94%). We also extensively evaluate different unsupervised variants of *UFA* and demonstrate that the designed components in our method are necessary. For instance, without linearized belief propagation, the precision of *UFA* decreases from 94.37% to 86.42%. Moreover, we compare *UFA* with supervised methods. Our results show that the state-of-the-art supervised method needs to label 600K accounts in order to achieve comparable performance with *UFA* and the supervised method is sensitive to noisy labels. Furthermore, *UFA* has been deployed by WeChat for more than one year to detect fake accounts at the time of the account registration. The WeChat security team manually verifies that *UFA* detects 500K fake accounts per day and achieves a precision 93% on average.

The main contributions of the paper are summarized as follows:

- We perform a large-scale measurement study on real-world WeChat registration datasets and observe that fake accounts tend to cluster on outlier registration patterns.
- We propose an unsupervised *UFA* to detect fake accounts in WeChat using registration data.
- We systematically evaluate *UFA* and compare it with several unsupervised and supervised methods.
- We have deployed *UFA* on the WeChat platform for more than one year, showing that *UFA* achieves a positive impact in industry.

2 RELATED WORK

Traditional detection methods: These methods can be classified into feature-based methods and structure-based methods. Feature-based methods [5, 8, 9, 14, 16–18, 20, 22, 29] often model fake account detection as a binary classification problem and leverage machine learning techniques. Specifically, they first extract features from each account's content (e.g., URLs in tweets), behaviors (e.g., clickstreams and likes), registration information (e.g., IP address and agent). Then, they train a supervised classifier (e.g., logistic regression) based on these extracted features and a labeled training set (consisting of labeled fake accounts and labeled benign accounts), and use the trained classifier to detect fake accounts.

Structure-based methods [1, 3, 4, 7, 10–13, 21, 23, 24, 27, 28, 30–34] leverage the structure of the social graph to detect fake accounts and are often based on the observation that an account is likely to be fake if it is connected with other fake accounts. These methods often leverage graph-based machine learning techniques, e.g., random walk [4, 7, 12, 13, 30, 31, 33, 34] and belief propagation [10, 23, 24, 27], to analyze the structure of the social graphs between users.

Early detection methods: The key limitation of the traditional detection methods above is that they detect fake accounts at a severe time delay, which allows the methods to obtain enough information generated by fake accounts for analysis, e.g., rich content, behavior, and/or social graphs. Therefore, fake accounts may have already conducted various malicious activities when they are detected. *UFA* addresses these limitations by detecting fake accounts at the time of registration in an unsupervised fashion. Recently Yuan et al. [35] develop a detection method that is most related to our work. Specifically, they develop a supervised fake account detection method, called Ianus, using registration data from WeChat. Moreover, Ianus has been deployed by WeChat for a while to detect fake accounts and achieves a high detection accuracy. However, to maintain the high detection accuracy, Ianus needs to be frequently retrained and needs to be fed a large amount of new labeled registration accounts for each retraining. Moreover, WeChat security team revealed that it is very time-consuming and costly to manually label these registration accounts. *UFA* is designed to overcome the shortcomings of Ianus. Specifically, *UFA* does not require the labels and our evaluation results illustrate that *UFA* can achieve comparable performance with Ianus. Moreover, the WeChat security team has deployed *UFA* to detect fake accounts for more than one year.

3 MODEL DESIGN

3.1 Overview

UFA aims to detect fake accounts at the time of their registrations in an unsupervised fashion. Figure 1 overviews *UFA*. It consists of four key components, i.e., *feature extraction*, *unsupervised weight learning*, *registration graph construction*, and *fake account detection*. In feature extraction, inspired by the measurement study on registration patterns, we extract features that reveal outlier registration patterns of fake accounts. In unsupervised weight learning, we first construct a registration-feature bigraph to capture the relationship between registration accounts and features. Each node in the bigraph represents either a registration account or a feature, and each edge between a registration node and a feature node indicates that the registration account has the feature. Next, we design

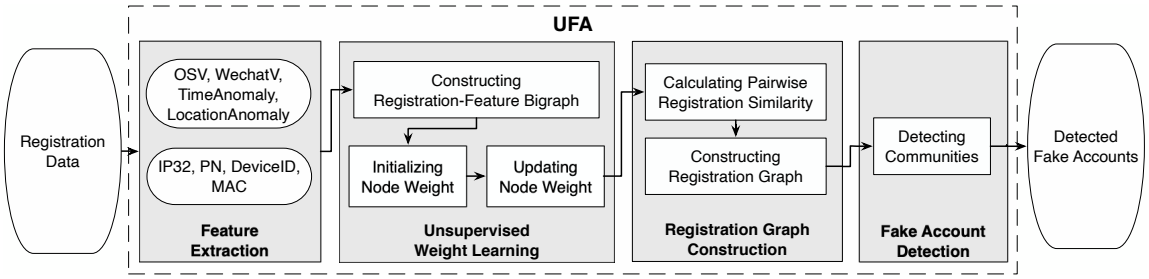


Figure 1: Overview of *UFA*.

Table 1: Registration Attributes

Attribute	Example
IP	10.xxx.xxx.10
Phone Number	+86-150-0516-xxxx
Timestamp	1499270558
WeChat Version	6.6.7
OS Version	iOS 10.3.2
Hashed WiFi MAC	5cee...f76a
Hashed Device ID	d6ad...fb63

a statistical method to initialize the weight of each node in the bigraph. The weight of node quantifies the node's anomaly. Our statistical method does not require labeled fake accounts or/and labeled benign accounts, and it is thus unsupervised. Here, the initial weights do not consider the relationship between features and between registration accounts. To address the issue, we adopt a state-of-the-art structure-based method, called linearized belief propagation [26], to propagate the initial weights of nodes among the registration-feature bigraph and iteratively update each node weight. The final weight of each registration account indicates the account's probability of being fake.

Unsupervised weight learning cannot learn the correlation between registration accounts since there are no edges between registration accounts in the registration-feature bigraph. Thus, we construct a registration graph to directly capture the correlation between registration accounts. Specifically, we map the registration-feature bigraph into a registration graph, where each node is a registration account and an edge is added between two registration accounts if their similarity is higher than a threshold. The similarity between two registration accounts is defined as the sum of weights of features shared by the two accounts. In the constructed registration graph, fake accounts are likely to be densely connected, while benign accounts are likely to be sparsely connected.

Finally, due to that fake accounts are densely connected in the registration graph, we utilize a community detection algorithm to cluster the registration accounts into communities in the fake account detection component. We treat all accounts in a community as fake accounts if the community size is larger than a threshold.

3.2 Feature Extraction

We first briefly present registration attributes collected by WeChat. Second, we perform a measurement study that reveals outlier registration patterns for fake accounts and observe similar results to the previous work performed on WeChat datasets [35]. For completeness, we include the detailed measurement results in Appendix.

Table 2: Feature Prefixes

FeaturePrefix	Description
IP32	hashed full IP address
PN	the phone number prefix
DeviceID	hashed device id
MAC	hashed mac
OSV	OS version
WeChatV	WeChat version
TimeAnomaly	whether an account is registered at late midnight, i.e., 2 a.m - 5 a.m
LocationAnomaly	whether an account is registered with different phone number-based location and IP-based location

Finally, we extract features that will be used for detection according to the registration attributes.

Registration attributes. When a user registers a WeChat account, WeChat collects several registration attributes about the account. We list the representative registration attributes in Table 1. For example, Phone Number is the number a user used to register an account. Each phone number can be used to register only one account and as account ID. WiFi MAC is the MAC address of the WiFi router that the phone used to register an account, and Device ID is the IMEI/Adsource of the phone used to register an account.

Observations from a measurement study. We have performed a measurement study on a real world registration dataset from WeChat[1]. Details of the measurement study are shown in the Appendix A. Comparing with benign accounts, fake accounts tend to have outlier registration patterns. In particular, fake accounts are likely to use the same IPs, use the same phone numbers from the same areas, be registered from the same devices, be active at midnight, and use rare and outdated WeChat and OS versions. A possible reason is that an attacker has limited resources (e.g., IPs, phone numbers, and devices) and can only automatically registers fake accounts using scripts.

Feature extraction. We extract features that show the outlier registration patterns for fake accounts. Specifically, we parse each registration's attribute and attribute value into a string with the format "%%FeaturePrefix%%_%%Value%%". All the feature prefixes are listed in Table 2. Each feature prefix is related to one registration attribute and represents a preprocessing function for that attribute; and each value is the result generated after preprocessing. For simplicity, we only show how we extract features with the feature prefix "TimeAnomaly", since the other features can be easily understood from their feature prefixes and the descriptions.

[1]The work was performed with an internship program with WeChat.

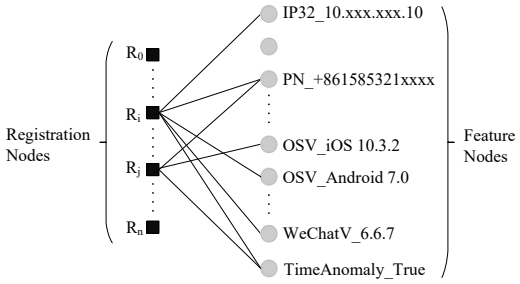


Figure 2: Registration-feature bigraph.

- **TimeAnomaly.** According to our measurement results (see Appendix A), we observe that accounts registered at late midnight are likely to be fake accounts, while benign accounts were mainly registered in daytime. Therefore, we define a feature prefix *TimeAnomaly*, which uses the timestamp attribute of a registration. TimeAnomaly is used to indicate whether an account's registration time is abnormal or not. That is, if the account is registered at daytime, the TimeAnomaly value is *False*, and the TimeAnomaly value is *True* if the account is registered at late midnight, i.e., between 2 AM and 5 AM in our work.

3.3 Unsupervised Weight Learning

In unsupervised weight learning, we aim to learn the weight for each extracted feature and each registration account. The weight quantifies the anomaly, with a range between 0 and 1. A higher weight indicates a larger probability of being abnormal. Specifically, we first construct a graph to capture the relationship between features and registration accounts, where each node represents a feature or a registration account and each edge between two nodes means the registration account has the feature. We design a statistical method to initialize a weight for each node in the constructed graph in an unsupervised fashion, and adopt a structure-based method to propagate the initial weights of nodes among the constructed graph and iteratively update the weight of each node. The final weight of each registration node can be regarded as the probability of being fake for that registration account.

Registration-feature bigraph construction. We construct a registration-feature bigraph to capture the relationship between the registration accounts and the extracted features, as shown in Figure 2. Specifically, we denote by $B(\mathcal{R}, \mathcal{F}, \mathcal{E})$ the registration-feature bigraph, where $\mathcal{R}$ represents a set of registration nodes and each registration node indicates a registration account, $\mathcal{F}$ represents a set of feature nodes and each feature node indicates a feature, and $\mathcal{E}$ represents an edge set, where an edge between a registration node and a feature node means the registration account has the feature. For example, in Figure 2, the registration node R_i connects with feature nodes such as "IP32_10.xxx.xxx.10", "PN_+861585321xxxx", "OSV_Android 7.0", which means that R_i uses the IP address "10.xxx.xxx.10", the phone number "+861585321xxxx" with the phone's OS version "Android 7.0" at the time of registration.

Feature/registration weight initialization. We design a statistical method to assign an initial weight for each feature and each registration account without requiring manual labels. Our statistical method relies on three concepts: *feature frequency*, *feature ratio*, and *mode feature* under a feature prefix.

Definition 3.1 (Feature frequency). A feature frequency is the number of registration accounts who have this feature.

Definition 3.2 (Feature ratio). Given a feature x with prefix **pre**, the feature ratio $ratio(x)$ is defined as the fraction of its feature frequency among all features with the same prefix **pre**, i.e.,

$$ratio(x) = \frac{freq(x)}{\sum\limits_{pre(s)=\textbf{pre}} freq(s)}, \tag{1}$$

where $freq(x)$ is x's feature frequency and $pre(x)$ is x's feature prefix.

Definition 3.3 (Mode feature). Given a feature prefix **pre**, the mode feature of **pre** is the feature x^* with this prefix having the maximal feature ratio, i.e.,

$$x^* = mode(\textbf{pre}) = \arg\max_{x \in \mathcal{F}, pre(x)=\textbf{pre}} ratio(x). \tag{2}$$

Now we can define the weight for each feature by utilizing the definitions. As aforementioned, we define the weight of a feature to quantify the feature anomaly. One natural choice is to relate the anomaly of the feature to its feature ratio. Specifically, given a feature, if it tends to be shared by benign accounts and has a small (or large) feature ratio, or it tends to be shared by fake accounts and has a large (or small) feature ratio, then the feature tends to be abnormal (or normal) and has a large (small) weight. According to this intuition, we first classify the feature prefixes of our extracted features into the following two categories: **Pre_A** = $\{OSV, WechatV, TimeAnomaly, LocationAnomaly\}$ and **Pre_B** = $\{IP32, PN, DeviceID, MAC\}$. For a feature whose prefix is in **Pre_A**, if its feature ratio is relatively larger, then the feature is less likely to be abnormal. The reason is that most of the features having large ratios with prefixes in **Pre_A** are shared by benign accounts, as demonstrated in our measurement study. In contrast, for a feature whose prefix is in **Pre_B**, if its feature ratio is relatively larger, the feature is more likely to be abnormal. The reason is that most of the features having large ratios with prefixes in **Pre_B** are shared by fake accounts, as illustrated in our measurement study.

Given the above analysis, one natural way to define the weight w_x for the feature x is as follows:

$$w_x = \begin{cases} 1 - ratio(x) & \text{if } pre(x) \in \textbf{Pre\_A}; \\ ratio(x) & \text{if } pre(x) \in \textbf{Pre\_B} \end{cases} \tag{3}$$

However, one key limitation is that the above defined feature weight cannot be used to directly compare feature ratios with different feature prefixes. For example, suppose we have a same ratio for two features under two different prefixes (but in the same prefix category). Based on Equation 3, we will assign the same weight for the two features. In practice, however, the feature ratio distributions are usually diversified across different feature prefixes, thus the same feature ratio under different prefixes could have different meanings. For instance, if we have two feature ratio distributions for two different prefixes. One feature ratio distribution is concentrated, i.e., its maximal feature ratio (the mode feature ratio) is very large, while the other feature ratio distribution is almost uniform, i.e., its maximal feature ratio is almost the average feature ratio. Thus, although two features under two different prefixes have the same

ratio, they should be assigned different weights. Unfortunately, the above defined weight does not have such a property.

To overcome this issue, we utilize a technique called *feature coupling* [15], which can make features with different prefixes comparable. Specifically, we define the weight for a feature by considering the feature ratio as well as the feature prefix's ratio. Formally, we define the weight w_x of a feature node $x \in \mathcal{F}$ as:

$$w_x = \frac{1}{2}\Big(dev(x) + base(x)\Big), \tag{4}$$

where

$$dev(x) = \begin{cases} 1 - \frac{ratio(x)}{ratio(mode(pre(x)))} & \text{if } pre(x) \in \textbf{Pre\_A}; \\ 1 - \frac{1-ratio(x)}{ratio(mode(pre(x)))} & \text{if } pre(x) \in \textbf{Pre\_B} \end{cases} \tag{5}$$

and

$$base(x) = \begin{cases} 1 - ratio(mode(pre(x))) & \text{if } pre(x) \in \textbf{Pre\_A}; \\ ratio(mode(pre(x))) & \text{if } pre(x) \in \textbf{Pre\_B} \end{cases} \tag{6}$$

Here, $\text{dev}(x)$ characterizes the anomaly of a feature x and $\text{base}(x)$ characterizes the anomaly of x's feature prefix. Specifically, suppose a feature x, where $\text{pre}(x) \in$ **Pre_A**. $\text{dev}(x)$ means that if the more the x's feature ratio deviates from the mode feature ratio, the more abnormal x is, and $\text{base}(x)$ means that if x's feature prefix has a larger mode feature ratio, then it is less likely to be abnormal. Note that, the definition of a feature weight in Equation 4 combines $\text{pre}(x)$ and $\text{base}(x)$, and thus can compare feature ratios across different feature prefixes. Moreover, since $\text{dev}(x) \in [0, 1]$ and $\text{base}(x) \in [0, 1]$, we have $w_x \in [0, 1]$.

Then, we initialize weights of feature nodes in the registration-feature bigraph using weights calculated from Equation 4. Note that, if a feature is unique, i.e., it is used by a single registration account, we set its weight to be 0.5, which means that we consider all unique features to be neutral.

Next, we assign the weight for each registration node in the registration-feature bigraph. Intuitively, a weight of a registration should consider all the weights of features the registration has. For simplicity, we initialize the weight of each registration node as the average weight of its connected feature nodes. Formally, for a registration node $r \in \mathcal{R}$, its weight w_r is defined as

$$w_r = \frac{\sum_{x \in \Gamma(r)} w_x}{|\Gamma(r)|}, \tag{7}$$

where $\Gamma(r)$ is the set of r's neighbors in the registration-feature bigraph and $|\Gamma(r)|$ is the size of $\Gamma(r)$.

Feature/registration weight update. The weight initialization does not consider the latent relationship between features and between registration accounts, since the weights are assigned once and fixed. To address the issue, we leverage a state-of-the-art structure-based method [26], called linearized belief propagation, to update each node weight in the registration-feature bigraph by considering the influence from its indirect neighbors. Specifically, for each node u in the registration-feature bigraph, we assign its prior probability of being fake as its initial weight w_u. Then, linearized belief propagation iteratively updates the weight of being fake for every node until it reaches a predefined number of maximum iterations, e.g., 10 iterations in our system. In particular, initially we have $p_u^{(0)} = w_u, \forall u \in \mathcal{F} \bigcup \mathcal{R}$. In the tth iteration, for each node u, we obtain its updated weight $p_u^{(t)}$ as follows:

$$p_u^{(t)} = w_u + 2 \sum_{v \in \Gamma(u)} (p_v^{(t-1)} - 0.5) \cdot (h_{vu} - 0.5), \tag{8}$$

where $h_{vu} \in [0, 1]$ is homophily strength of the edge $(v, u) \in \mathcal{E}$, which means the probability that two connected nodes u and v have the same label. In our system, we use a degree-normalized homophily strength for each edge [26], i.e., $h_{vu} = \frac{1}{2|\Gamma(u)|} + 0.5$. Moreover, we treat unique features to be neutral, i.e., we keep the weight of unique features to be 0.5 in all iterations.

After linearized belief propagation terminates, we treat the final updated weight of each account as its probability of being fake. Note that these final weights can be used for fake account detection. Specifically, we predict a registration account $r \in \mathcal{R}$ to be a fake account if its final weight $p_r > 0.5$, otherwise we predict r to be a benign account. However, we find that directly using these weights achieves suboptimal detection performance (see the experiment results in Section 4). This is because the correlation between registration accounts is not modeled directly, since there are no edges between registration accounts in the registration-feature bigraph.

3.4 Registration Graph Construction

We construct a weighted graph to directly capture the correlation between registration accounts. The constructed graph is mapped from the registration-feature bigraph. Specifically, for each pair of registration nodes $u, v \in \mathcal{R}$ in the registration-feature bigraph, we create an edge (u, v) between u and v if the similarity between u and v is larger than a threshold. In particular, we define the similarity $\text{sim}(u, v)$ between u and v as the sum of the final weight of u's and v's shared features. The intuition behind this is that if two registration nodes share many features and these features have large final weights (large probabilities of being abnormal), then the two registration accounts also have a large similarity and tend to be fake accounts. Formally,

$$\text{sim}(u, v) = \sum_{s \in \Gamma(u) \bigcap \Gamma(v)} p_s. \tag{9}$$

Moreover, we set the weight of an edge (u, v) in the new graph to be the similarity $\text{sim}(u, v)$. We call the constructed weighted graph *a registration graph* as all nodes in the graph are registration accounts. Fake accounts are more likely to be connected with each other with large edge weights, and benign accounts are more likely to be sparsely connected with small edge weights.

3.5 Fake Account Detection

In the registration graph, we note that fake accounts are densely connected and benign accounts are sparsely connected. To detect fake accounts, we need to identify dense subgraphs in the registration graph. A natural choice is to leverage community detection methods. We first adopt a community detection method, e.g., Louvain method [2], to cluster the nodes into different communities (i.e., dense subgraphs). Second, we predict all registration accounts in the communities whose sizes are larger than a threshold to be fake accounts.

Table 3: Dataset statistics.

Dataset	# benign accounts	# fake accounts
Day I	807K	929K
Day II	768K	1,017K
Day III	692k	1,073K
Day IV	720K	972K
Day V	770K	649K
Day VI	740K	610K
Day VII	782K	648K

Table 4: Results of *UFA* on the seven datasets.

Dataset	Precision	Recall	F-Score
Day I	89.49%	77.43%	83.03%
Day II	90.23%	80.48%	85.08%
Day III	92.00%	80.78%	86.23%
Day IV	90.66%	80.35%	85.19%
Day V	94.37%	80.05%	86.62%
Day VI	91.47%	80.63%	85.71%
Day VII	90.06%	79.11%	84.23%

4 EVALUATION

4.1 Experimental Setup

Datasets. We obtained seven datasets from WeChat. Each dataset includes registrations within one day. Table 3 shows the detail of the datasets. The labels are provided by the WeChat security team, who manually verified these labels and revealed that these labels have an accuracy larger than 95%. We will use these labels as ground truth for evaluation.

Evaluation metrics. We use three metrics, i.e., *Precision*, *Recall*, and *F-score*, to evaluate the performance. For a detection method, *Precision* is the fraction of its predicted fake accounts that are true fake accounts in the testing set, *Recall* is the fraction of true fake accounts in the testing set that are predicted as fake accounts by the method, and *F-Score* is the harmonic mean of *Precision* and *Recall*.

Compared methods. We compare *UFA* with several unsupervised variants including *UFA*-naive, *UFA*-noLBP, *UFA*-noRG and supervised methods including Ianus [35], XGBoost [6] and Deep Neural Network (DNN). Due to space limitation, the details of these methods are referred to Appendix B.

4.2 Experimental Results

***UFA* is effective.** Table 4 shows the results of *UFA* on the seven datasets. For instance, *UFA* can detect around 80% of fake accounts with *Precision* around 90% in all datasets. A key reason is that through unsupervised weight learning and registration graph construction, fake accounts are densely connected in the registration graph but benign accounts are sparsely connected. Then, the Louvain method can easily detect these dense connected fake accounts. For instance, in the registration graph constructed from registration data in Day V, on average, a fake account is connected with 22.32 other fake accounts, a fake account is connected with only 1.39 benign accounts, and a benign account is connected with only 2.04 other benign accounts. Note that, by studying the distribution of the registration data, we find that around 85% of fake accounts are clustered together and the remaining 15% of fake accounts are scattered, i.e., they do not share common features with others. Thus, the recall of ~80% achieved by *UFA* demonstrates its effectiveness in detecting fake accounts.

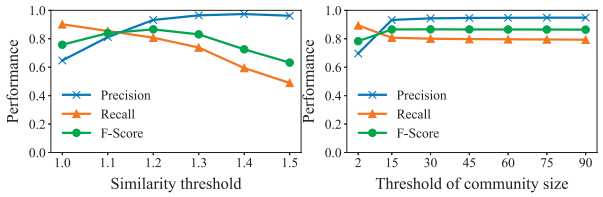


Figure 3: (a) Impact of the similarity threshold. (b) Impact of the community size.

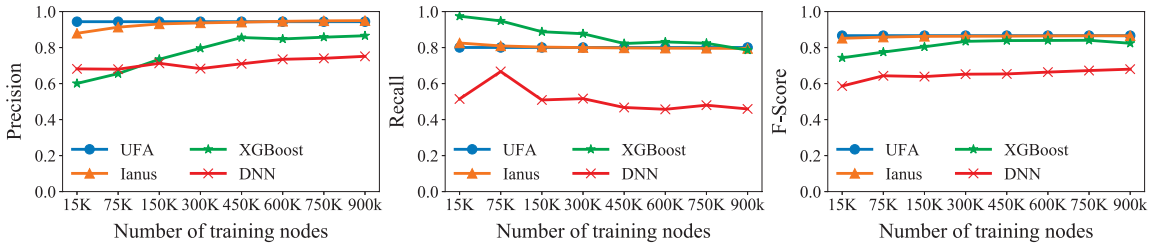
Table 5: Results of *UFA* vs. unsupervised baselines.

Model	Precision	Recall	F-Score
***UFA*-naive**	64.60%	94.33%	76.68%
***UFA*-noLBP**	86.42%	83.33%	84.85%
***UFA*-noRG**	69.91%	93.08%	79.85%
UFA	**94.37%**	**80.05%**	**86.62%**

Impact of the similarity threshold. Recall that *UFA* predefines a threshold of similarity between a pair of registration accounts to determine whether an edge is added between the two registration accounts. A natural question is how this similarity threshold impacts the *UFA*'s detection performance. Figure 3a shows the results of *UFA* vs. similarity threshold between 1.0 and 1.5. We observe that *Precision* increases and *Recall* decreases when the threshold increases, and *F-Score* first increases and then decreases. The reason is that a larger threshold makes it harder for a pair of registration accounts to be connected in our registration graph. When a higher threshold is used, the connected nodes in the registration graph are more likely to be fake accounts, so we can have a higher *Precision*. Meanwhile, fewer fake accounts can be connected with each other, thus *Recall* decreases. We also note that *F-Score* reaches the maximum when the similarity threshold is around 1.2. Therefore, we choose a default similarity threshold to be 1.2 in our experiments.

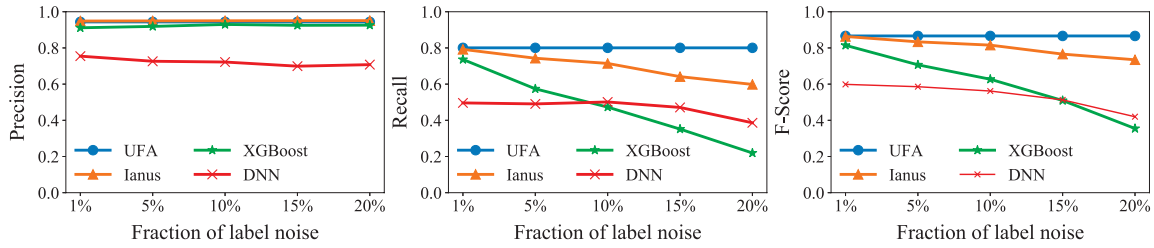
Impact of the community size. *UFA* uses the Louvain method to detect communities and predict registration accounts in the communities whose sizes are larger than a predefined threshold as fake accounts. We study the impact of different community sizes on *UFA*'s detection performance and show the results in Figure 3b. We observe that as the threshold of community size becomes larger, *Precision* and *F-Score* increase and *Recall* decreases at first. When the threshold is larger than 15, all the three metrics are stable. Therefore, we choose a default threshold of the community size to be 15 in our experiments.

***UFA* vs. *UFA*-naive.** The difference between *UFA* and *UFA*-naive is that *UFA* initializes weights for features based on Equation 4, while *UFA*-naive initializes weights for features based on the frequency ratio as shown in Equation 3. Table 5 shows the comparison results of *UFA* and *UFA*-naive. We observe that *Precision* and *F-Score* of *UFA* are significantly higher than those of *UFA*-naive. In particular, *UFA* obtains *Precision* that is about 30% higher than *UFA*-naive, and *UFA*'s *F-Score* is about 10% larger than *UFA*-naive's. This is because the frequency ratio based feature weight cannot directly



(a) Precision (b) Recall (c) F-Score

Figure 4: Comparing *UFA* with supervised methods vs different number of training nodes.



(a) Precision (b) Recall (c) F-Score

Figure 5: Comparing *UFA* with supervised methods vs different fraction of label noises.

compare features with different prefixes, while *UFA* introduces the mode feature for each feature prefix to overcome this issue. Thus, initializing nodes' weights based on the frequency ratio is inappropriate for detecting fake accounts.

***UFA* vs. *UFA*-noLBP.** The difference between *UFA* and *UFA*-noLBP is that *UFA* uses the linearized belief propagation method to iteratively update nodes' weights, while *UFA*-noLBP simply uses the initial nodes' weights. Table 5 shows the comparison result of *UFA* and *UFA*-noLBP. We observe that *Precision* and *F-Score* of *UFA* are higher than those of *UFA*-noLBP. This demonstrates that nodes' weights learned by the linearized belief propagation are more beneficial than the initial nodes' weights for fake account detection.

***UFA* vs. *UFA*-noRG.** The difference between *UFA* and *UFA*-noRG is that *UFA* uses the registration graph and the community detection method to detect fake accounts, while *UFA*-noRG does not construct the registration graph but uses the final weights outputted by linearized belief propagation for fake account detection. Table 5 shows the comparison results of *UFA* and *UFA*-noRG. We observe that *Precision* and *F-Score* of *UFA* are higher than those of *UFA*-noRG. This is because the registration graph in *UFA* directly captures the correlation between registration accounts, and thus can further promote the detection performance.

***UFA* vs. supervised methods.** Ianus [35] is the state-of-the-art supervised method using registration accounts. Ianus uses a labeled training set to train a logistic regression classifier and then applies the trained classifier to the testing set to predict and initialize nodes' weights. XGBoost and DNN are two popular supervised methods. Naturally, supervised methods can have better performance when more training nodes are used. One natural question is how many nodes need to be manually labeled in order for a supervised method to match *UFA*'s performance. Figure 4 shows the results of *UFA* and the compared supervised methods with different numbers of training nodes. We observe that Ianus requires manually labeling $600K$ nodes on the registration dataset in order to achieve *F-Score* comparable to *UFA*'s. Moreover, *UFA* outperforms XGBoost and DNN even when the two methods have $900K$ labeled accounts.

In practice, a training set might have label noises, i.e., some labeled benign accounts are mislabeled as fake accounts and some labeled fake accounts are mislabeled as benign accounts. Another question is how label noise in the training set impacts the detection performance of supervised methods. For a given level of noise $\tau\%$, we randomly sample $\tau\%$ of labeled benign accounts in the training set and mislabel them as fake accounts and sample $\tau\%$ of labeled fake accounts in the training set and mislabel them as benign accounts. Figure 5 shows the *F-Score* of *UFA* and the compared supervised methods with 900K labeled nodes. We observe that *F-Score* of all the compared supervised methods have a significant decrease when the fraction of label noise is above 5%. This indicates that all these supervised methods are sensitive to noisy labels.

Summary. *UFA* is effective to detect fake accounts at the time of their registration, outperforms the state-of-the-art supervised methods even when the available number of labeled nodes is insufficient, and is more practical.

5 REAL-WORLD DEPLOYMENT AT WECHAT

UFA has been deployed by WeChat to detect fake accounts for more than one year. It is implemented on Spark with Python and deployed on YARD, an internal cloud computing platform of WeChat. *UFA* works in a flow pattern, and the deployment diagram is shown in Figure 6. Specifically, *UFA* system has two stages: system initialization and system update.

System initialization. When initially deployed by WeChat, *UFA* collects a certain amount of registration accounts to extract features, build the registration-feature bigraph, learn weights for features

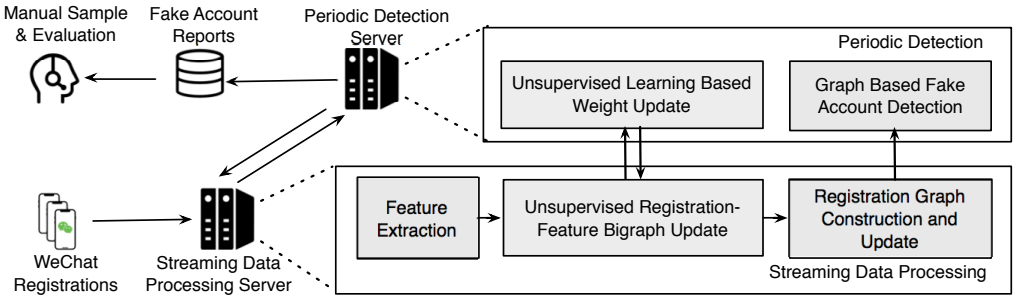


Figure 6: Real-world deployment of *UFA*.

and registration accounts, and construct the registration graph. WeChat used all registration accounts in the first week since *UFA* deployment to fulfill system initialization. After initialization, we have the weight of features and registration accounts. All steps in system initialization are executed on the streaming process server.

System update. After system initialization, when the WeChat server receives a new registration account, *UFA* first extracts the account features and then executes the following steps.

- **Step I: Create new nodes/edges in the registration-feature bigraph.** *UFA* first creates a new registration node and new feature nodes for those features not in the current registration-feature bigraph. Then, *UFA* creates new edges between the registration node and its features nodes. This step is run on the streaming process server.
- **Step II: Create new nodes/edges in the registration graph.** *UFA* retrieves existing registration accounts that are close to the new registration account in this step. Specifically, *UFA* finds all the existing registration accounts that have shared features with the new registration account. Then, *UFA* computes the similarities between the new registration account and the found registration accounts using the current weights of feature nodes. Next, in the registration graph, *UFA* creates a new node for the new registration account and creates an edge between the new registration account and each found registration account, if the similarity between two accounts is larger than a predefined threshold. This step is run on the streaming process server.
- **Step III: Periodically update feature/registration weights and detect fake accounts.** To mitigate the computational overhead, *UFA* updates the weight of feature/registration nodes and detects fake accounts hourly. Both the weight update and fake account detection are executed on the periodic detection server. Specifically, *UFA* on the streaming process server updates the registration-feature bigraph and registration graph for an hour and saves the graphs into a database. Then, the periodic detection server loads the graphs from the database and runs the unsupervised weight learning algorithm and the community detection algorithm to detect fake accounts. Next, *UFA* transfers the updated feature/registration weights and detected fake accounts from the periodic detection server to the streaming process server. The detected fake accounts are stored into a database. The streaming process server then uses the updated feature/registration weights for the next periodical system update.

Table 6: An example of registration information

ID	WeChat Version	OS Version	IP	Phone Number
u1	39.0.3.52	android-25	14.25.16.xxx	+861892551xxxx
u2	39.0.5.52	android-25	14.25.16.xxx	+861491129xxxx
u3	39.0.5.52	android-25	14.25.16.xxx	+861491100xxxx
u4	39.0.6.52	android-25	14.25.16.xxx	+861732987xxxx
u5	39.0.6.52	android-25	14.25.16.xxx	+861819880xxxx

***UFA*'s performance in real-world deployment.** WeChat suspends all detected fake accounts, where some of them might be benign accounts. If a benign account was suspended, WeChat security team would receive a complaint from the user who registered the account. WeChat uses the number of complaints to evaluate the performance of *UFA*. Specifically, WeChat security team has 6 security analysts to deal with users' complaints. Overall, less than 6% of suspended accounts applied to unlock their accounts.

Moreover, WeChat security team randomly samples 200 detected fake accounts by *UFA* to evaluate the performance of *UFA*. *UFA* achieves a *Precision* of 93%, which is consistent with the results measured according to the complaints. In summary, *UFA* detects 500K fake accounts per day on average and has detected 180 million fake accounts in total (out of 400 million registration accounts) since *UFA* deployment.

Limitation of Ianus in real-world deployment. Before *UFA*, WeChat has deployed Ianus [35] for around six months. Ianus is a supervised fake account detection method that also leverages accounts' registration information. However, Ianus exposed its weaknesses after being deployed for a time. First, manual labeling is time-consuming. WeChat security team has 6 security analysts whose duty is to label registration accounts and each analyst can label around 1000 registrations per day. To manually label, say 600K registrations, it will take around 100 days, which is time-consuming.

Second, accurate labels are hard to obtain. Another challenge is that many fake accounts resemble benign accounts when checking its registration information alone. Table 6 shows such an example in the dataset. If we only observe each user's registration information, we can hardly decide the user as a fake account. Only through observing the registration information of the total 5 users, we can decide that these users are fake as they all use the same IP, WeChat version, and OS version. Note that this situation is common in the

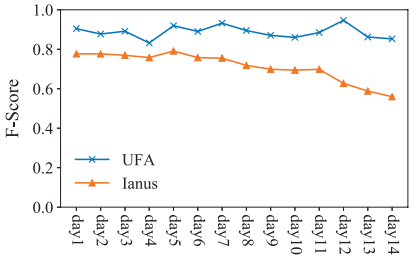


Figure 7: Performance of online systems.

real-world dataset, and thus it is hard to obtain accurate labels. With a certain amount of noisy labels, Ianus's performance will degrade as demonstrated in Section 4.2.

Third, Ianus needs to be retrained frequently to maintain high detection performance. Figure 7 shows Ianus's performance in a random consecutive 14 days with a fixed training set. For simplicity, we only show *F-Score* and we have similar observations on *Precision* and *Recall*. We see that Ianua's *F-Score* largely decreases, i.e., from 77.68% to 55.92%, within the 14 days. One possible reason is that attackers continuously change their registration patterns in order to evade Ianus's detection. Therefore, Ianus trained based on old registration patterns is insufficient to detect fake accounts with new registration patterns. To maintain the high detection performance, Ianus needs to be frequently retrained with accurate labels on the new registration patterns. However, as we mentioned above, it is time-consuming and hard to accurately label registration accounts. In contrast, *UFA* has a relatively stable *F-Score*, demonstrating that *UFA* outperforms Ianus in real-world deployment.

6 CONCLUSION

In this paper, we developed an unsupervised method *UFA* to detect fake accounts immediately after they are registered. We first extracted features revealing outlier registration patterns for fake accounts, inspired by a measurement study on a real-world registration dataset from Wechat. Then, we designed an unsupervised weight learning algorithm to learn weights for extracted features and the registration accounts. Furthermore, we constructed a registration graph to directly capture the correlation between registration accounts, such that fake accounts are densely connected while benign accounts are sparsely connected. Finally, we adopted a community detection algorithm to detect fake accounts via analyzing the registration graph structure. We evaluated *UFA* using real-world datasets from WeChat. Our results showed that *UFA* achieved a precision ~94% with a recall ~80%. *UFA* has been also deployed by WeChat to detect fake accounts for more than a year and achieved a precision of 93%.

ACKNOWLEDGEMENTS

This work is supported in part by BNRist under Grant BNR2020R-C01013. Qi Li is the corresponding author of this paper.

REFERENCES

[1] Lorenzo Alvisi, Allen Clement, Alessandro Epasto, and et al. 2013. Sok: The evolution of sybil defense via social networks. In *IEEE S & P*.
[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* (2008).
[3] Yazan Boshmaf, Dionysios Logothetis, Georgos Siganos, and et al. 2016. Íntegro: Leveraging Victim Prediction for Robust Fake Account Detection in Large Scale OSNs. *Computers & Security* (2016).
[4] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. 2012. Aiding the detection of fake accounts in large scale social online services. In *NSDI*.
[5] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. 2014. Uncovering large groups of active malicious accounts in online social networks. In *CCS*.
[6] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *KDD*.
[7] George Danezis and Prateek Mittal. 2009. Sybilinfer: Detecting sybil nodes using social networks.. In *NDSS*.
[8] Manuel Egele, Gianluca Stringhini, Christopher Kruegel, and Giovanni Vigna. 2015. Towards detecting compromised accounts on social networks. In *TDSC*.
[9] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y Zhao. 2010. Detecting and characterizing social spam campaigns. In *Internet measurement*.
[10] Peng Gao, Binghui Wang, Neil Zhenqiang Gong, Sanjeev R Kulkarni, Kurt Thomas, and Prateek Mittal. 2018. Sybilfuse: Combining local attributes with global structure to perform robust sybil detection. In *CNS*.
[11] Neil Zhenqiang Gong, Mario Frank, and Prateek Mittal. 2014. Sybilbelief: A semi-supervised learning approach for structure-based sybil detection. In *TIFS*.
[12] Jinyuan Jia, Binghui Wang, and Neil Zhenqiang Gong. 2017. Random walk based fake account detection in online social networks. In *DSN*.
[13] Abedelaziz Mohaisen, Nicholas Hopper, and Yongdae Kim. 2011. Keep your friends close: Incorporating trust into social network-based sybil defenses. In *INFOCOM*.
[14] Mainack Mondal, Bimal Viswanath, Allen Clement, Peter Druschel, Krishna P. Gummadi, Alan Mislove, and Ansley Post. 2012. Defending against Large-Scale Crawls in Online Social Networks. In *CoNEXT*.
[15] Guansong Pang, Longbing Cao, and Ling Chen. 2016. Outlier Detection in Complex Categorical Data by Modeling the Feature Value Couplings.. In *IJCAI*.
[16] Jonghyuk Song, Sangho Lee, and Jong Kim. 2011. Spam filtering in twitter using sender-receiver relationship. In *RAID*. Springer.
[17] Gianluca Stringhini, Wang Gang, Manuel Egele, and et al. 2013. Follow the green: growth and dynamics in twitter follower markets. In *IMC*.
[18] Gianluca Stringhini and Olivier Thonnard. 2015. That Ain't You: Blocking Spearphishing Through Behavioral Modelling. In *DIMVA*.
[19] Kurt Thomas, Chris Grier, Justin Ma, Vern Paxson, and Dawn Song. 2011. Design and evaluation of a real-time url spam filtering service. In *IEEE S & P*.
[20] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. 2013. Trafficking Fraudulent Accounts: The Role of the Underground Market in Twitter Spam and Abuse. In *Usenix Security*.
[21] Bimal Viswanath, Ansley Post, Krishna P Gummadi, and Alan Mislove. 2011. An analysis of social network-based sybil defenses. In *SIGCOMM*.
[22] Alex Hai Wang. 2010. Don't follow me: Spam detection in twitter. In *SECRYPT*.
[23] Binghui Wang, Neil Zhenqiang Gong, and Hao Fu. 2017. GANG: Detecting fraudulent users in online social networks via guilt-by-association on directed graphs. In *ICDM*.
[24] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. 2019. Graph-based security and privacy analytics via collective classification with joint weight learning and propagation. In *NDSS*.
[25] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. Semi-Supervised Node Classification on Graphs: Markov Random Fields vs. Graph Neural Networks. In *AAAI*.
[26] Binghui Wang, Jinyuan Jia, Le Zhang, and Neil Zhenqiang Gong. 2019. Structure-based sybil detection in social networks via local rule-based propagation. *IEEE TNSE* (2019).
[27] Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. 2017. SybilSCAR: Sybil detection in online social networks via local rule based propagation. In *INFOCOM*.
[28] Binghui Wang, Le Zhang, and Neil Zhenqiang Gong. 2018. Sybilblind: Detecting fake users in online social networks without manual labels. In *RAID*.
[29] Gang Wang, Manish Mohanlal, Christo Wilson, Xiao Wang, Miriam Metzger, Haitao Zheng, and Ben Y Zhao. 2013. Social turing tests: Crowdsourcing sybil detection. In *NDSS*.
[30] Jilong Xue, Zhi Yang, Xiaoyong Yang, Xiao Wang, Lijiang Chen, and Yafei Dai. 2013. Votetrust: Leveraging friend invitation graph to defend against social network sybils. In *INFOCOM*.
[31] Chao Yang, Robert Harkreader, Jialong Zhang, Seungwon Shin, and Guofei Gu. 2012. Analyzing spammers' social networks for fun and profit: a case study of cyber criminal ecosystem on twitter. In *WWW*.
[32] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y Zhao, and Yafei Dai. 2014. Uncovering social network sybils in the wild. *ACM TKDD* (2014).
[33] Haifeng Yu, Phillip B Gibbons, Michael Kaminsky, and Feng Xiao. 2008. Sybillimit: A near-optimal social network defense against sybil attacks. In *IEEE S & P*.
[34] Haifeng Yu, Michael Kaminsky, Phillip B Gibbons, and Abraham Flaxman. 2006. Sybilguard: defending against sybil attacks via social networks. In *SIGCOMM*.
[35] Dong Yuan, Yuanli Miao, Neil Zhenqiang Gong, Zheng Yang, Qi Li, Dawn Song, Qian Wang, and Xiao Liang. 2019. Detecting Fake Accounts in Online Social Networks at the Time of Registrations. In *CCS*.

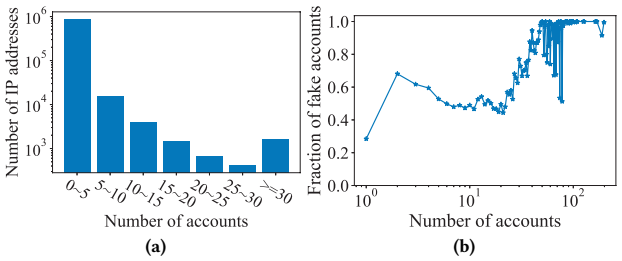


Figure 8: (a) The number of IP addresses that registered a given number of accounts. (b) The fraction of fake accounts among the accounts registered from the IP addresses that registered a given number of accounts.

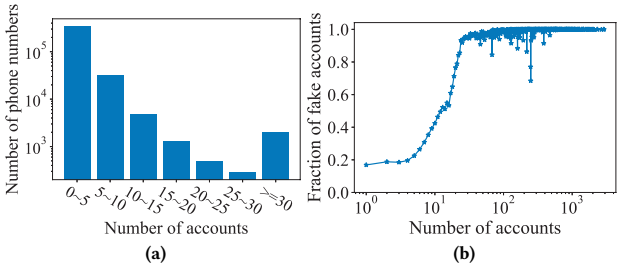


Figure 9: (a) The number of phone numbers that registered a given number of accounts. (b) The fraction of fake accounts among the accounts registered from the devices that registered a given number of accounts.

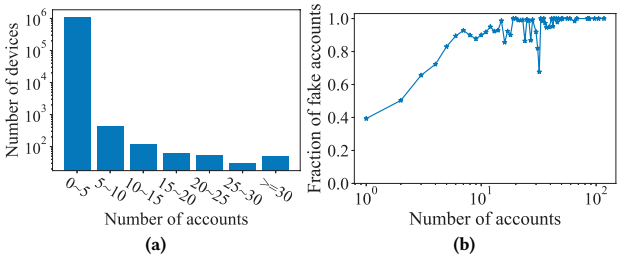


Figure 10: (a) The number of devices that registered a given number of accounts. (b) The fraction of fake accounts among the accounts registered from the devices that registered a given number of accounts.

A REGISTRATION DISTRIBUTION

A.1 IP

In total, we have 895,879 different IP addresses in our dataset. We group together all the accounts that were registered using a same IP address. Therefore, the size of a group indicates the number of accounts registered from the consistent IP address. Figure 8a shows the number of IP addresses that registered a given number of accounts. We observe an order of magnitude decline of the number of IP addresses when the given number of accounts grows from 0-5 to 5-10. This shows that a majority of IP addresses registered a small number of accounts (less than 5 accounts), while a small number of IP addresses registered a large number of accounts.

Moreover, fake accounts are more likely to be registered using the same IP addresses. Figure 8b shows the fraction of fake accounts among the accounts registered from the IP addresses that registered a given number of accounts. We observe that, when an IP address registered a small number of accounts (e.g., 10), it is hard to tell whether these accounts are fake accounts or not solely based on the fact that they share IP address. However, when a large number (e.g., 100) of accounts were registered from the same IP address, these accounts are more likely to be fake accounts.

A.2 Phone number

A user must provide a phone number when registering a WeChat account. Since we can only access the first 7 digits (phone service provider plus an area code) of phone numbers, we will study the difference between fake accounts and benign accounts using the prefix of the phone numbers. Figure 9a shows the number of phone number prefixes that registered a given number of accounts, while Figure 9b shows the fraction of fake accounts among the accounts registered from the phone number prefixes that registered a given number of accounts. Like IP addresses, we observe that a majority of phone prefixes have a small number of registered accounts, while a small number of phone prefixes have a large number of registered accounts. More specifically, if a phone prefix has more than 30 registered accounts, most of these accounts may be fake accounts.

A.3 Device

Figure 10a shows the number of devices (identified by IMEI) registering a given number of accounts, and Figure 10b shows the fraction of fake accounts among the accounts registered from the devices that registered a given number of accounts. We observe similar outlier patterns, i.e., fake accounts are likely to be registered from the same devices.

A.4 Registration time

Figure 11 shows the distribution of accounts registered within a day. We observe that most benign users register accounts from 8 a.m. to 24 p.m., very few users active at midnight. However, the number of fake accounts seems no changes during the whole day. Most accounts registered at midnight may be fake accounts.

A.5 Inconsistent geolocation

Both the IP address and the phone number can be mapped to a geolocation. Thus, we can analyze whether the two geolocations of each registration are the same. We observe that 65% of fake accounts have different IP-based location and phone number-based location. A possible reason is that attackers leverage cloud services and phone numbers obtained from local areas to register fake accounts. In addition, a user can also specify its location (e.g., country) in the profile when registering an account. We find that 96% of fake accounts specified countries that are inconsistent with the IP-based locations. A possible reason is that these fake accounts aim to attack benign accounts from particular locations.

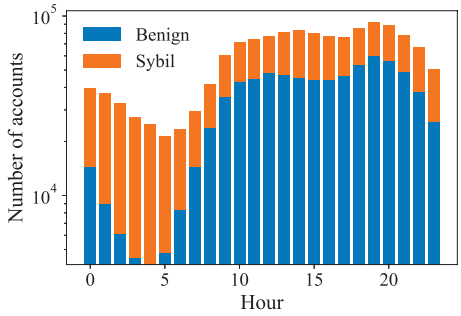


Figure 11: The number of accounts registered within a day.

A.6 Outdated WeChat and OS version

We analyzed the WeChat version and OS version and found that most accounts registered from outdated WeChat or/and OS versions are fake accounts. For example, only 2K accounts were registered from a certain outdated Android version and 96.5% of these accounts are fake. The same phenomenon occurs in iOS. For instance, we found that 99% of accounts registered from iOS 8 (an outdated OS version) are fake accounts. A possible reason is that attackers automatically register fake accounts using scripts, in which the WeChat and OS versions have not been updated.

A.7 Ethical and privacy considerations

WeChat has specified in the privacy policy that users' data would be collected when users register accounts. Moreover, all registration attributes have been anonymized to protect user privacy before submitted to WeChat's servers. For example, the customer code (i.e., the last four digits) of a phone number is removed. The IP address is hashed segment by segment, and WiFi MAC and Device ID are all hashed. In particular, we have an internship program with WeChat so that we can access the data above stored on WeChat's servers.

B EVALUATION DETAILS

B.1 Parameter setting

Our method *UFA* has the following two key parameters: the similarity threshold for a pair of registrations and the community size. By default, we set the threshold to be 1.2 and the community size to be 15. We will also study the impact of the two parameters. Moreover, we use the popular Louvain method [2] as the community detection method.

B.2 Compared method

We compare *UFA* with several unsupervised variants, including *UFA* using naive *frequency ratio* to initialize weights, *UFA* without linearized belief propagation, and *UFA* without registration graph construction. We denote these unsupervised variants as *UFA*-naive, *UFA*-noLBP, and *UFA*-noRG, respectively. Moreover, we also compare *UFA* with three supervised methods including state-of-the-art Ianus [35], XGBoost [6], and Deep Neural Network (DNN).

- **Unsupervised methods**
  - ***UFA*-naive.** *UFA*-naive does not use Equation 4 to calculate the initial weights of feature nodes in the registration-feature bigraph. Instead, it simply initializes the weights of feature nodes based on *frequency ratio* (i.e., Equation 3).
  - ***UFA*-noLBP.** *UFA*-noLBP does not use the linearized belief propagation method to iteratively update weights of nodes in the registration-feature bigraph. Instead, it simply uses the initial nodes' weights computed from Equation 4 and 7 and constructs the registration graph based on these initial nodes' weights.
  - ***UFA*-noRG.** *UFA*-noRG does not construct the registration graph. Instead, it simply uses the outputs of linearized belief propagation for fake account detection.
  - ***UFA*.** It uses all the four components.
- **Supervised methods**
  - **Ianus.** It is a supervised detection method. Specifically, Ianus uses the registration accounts in Day I as the training set, takes extracted features described in Yuan et al [35] as input, and trains a logistic regression classifier. Then, it predicts the weight of registration accounts in the testing set using the trained classifier, constructs the registration graph, and detects fake accounts via the weighted node degree method [35].
  - **XGBoost.** We first construct a feature vector for each registration account. Recall that each feature has the format "%%FeaturePrefix%%_%%Value%%". Specifically, we first count the number for each feature in the training set. Then, for each feature, we use its count as the value in the corresponding entry of the feature vector. For instance, if a registration account has an IP "10.212.212.10" and the IP appears 10 times in the training set, then the account's feature vector has a value 10 in the first entry. In our experiments, XGBoost uses the feature vector of each registration account as an input. Moreover, XGBoost uses 300 trees and each tree has a max depth of 8.
  - **DNN.** DNN uses the same feature vector as XGBoost. In our experiments, we use a DNN architecture with 3 hidden layers, with each layer having 32, 32, and 16 hidden units. We use ReLU as the nonlinear activation function and adopt Dropout with a dropout rate 0.5.

B.3 Training set and testing set

All unsupervised methods, i.e., *UFA*-naive, *UFA*-noLBP, *UFA*-noRG, and *UFA*, do not need a training set. They simply take registration data as input and detect fake registration accounts. We evaluate *UFA* on all the seven datasets and observe that the detect results are close, as shown in Table 4. Therefore, for simplicity, we will use the registration data in Day V as the testing set for all unsupervised and supervised methods.

Supervised methods require a labeled training set. To simulate real-world scenarios, we construct a training set from historical registrations and detect fake accounts in future registrations. Specifically, we randomly sample some registration accounts in Day I as a training set and test the three supervised methods on registration data in Day V.