TweetScore: Scoring Tweets via Social Attribute Relationships for Twitter Spammer Detection

Yihe Zhang
University of Louisiana at Lafayette, LA, USA
yihe.zhang1@louisiana.edu

Hao Zhang
Oracle Corp., CA, USA
haozhang85@gmail.com

Xu Yuan
University of Louisiana at Lafayette, LA, USA
xu.yuan@louisiana.edu

Nian-Feng Tzeng
University of Louisiana at Lafayette, LA, USA
tzeng@louisiana.edu

ABSTRACT

The spammers have been grossly detrimental since the inception of Twitter social networks and keep polluting social environments by hiding themselves among a large amount of normal users. In this paper, we aim to address two challenges existing in the spammer detection problem: 1) monitoring tweets that have a higher probability of including spam messages; 2) providing an accurate solution for spam classification. To address these two challenges, we first propose a pseudo-honeypot framework for efficient tweets monitoring and collection. By taking advantage of users' diversity and selecting normal users as the parasitic body, the pseudo-honeypot can harness normal users with features having much more potentials of attracting spammers. This lets the pseudo-honeypot collect tweets that are far more likely to include spam messages. Furthermore, we design a novel spam classification solution called *TweetScore* by exploring both the intrinsic attributes' and users' relationships in social networks. *TweetScore* quantifies such relationships into a vector of numerical values to represent each tweet's score, reflecting the associated user's behaviors. The neural network is then employed to take these vectors as input to classify spams and spammers. Through extensive experiments, we demonstrate the efficiency of the pseudo-honeypot system on spam monitoring and the accuracy of *TweetScore* on spam classification. Specifically, the spam and spammer ratios collected by our pseudo-honeypot system are *four times* as much as those of a *non pseudo-honeypot* counterpart while the *TweetScore* can achieve, on an average, 93.5% accuracy, 93.71% precision, and 1.52% false positive in online spam classification. The experimental results also show that the proposed *TweetScore* exhibits significant performance improvement in terms of spammers detection, when compared to the existing solutions.

AsiaCCS '19, July 9–12, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6752-3/19/07…$15.00
https://doi.org/10.1145/3321705.3329836

CCS CONCEPTS

• **Information systems** → **Spam detection**; Collaborative filtering; • **Computing methodologies** → *Neural networks*; • **Human-centered computing** → Social networks; • **Theory of computation** → Random walks and Markov chains.

KEYWORDS

Spam detection; Social network

ACM Reference Format:
Yihe Zhang, Hao Zhang, Xu Yuan, and Nian-Feng Tzeng. 2019. TweetScore: Scoring Tweets via Social Attribute Relationships for Twitter Spammer Detection. In *ACM Asia Conference on Computer and Communications Security (AsiaCCS '19), July 9–12, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3321705.3329836

1 INTRODUCTION

The spammers have been the adversary to Twitter networks, persistently polluting social networking environments. By imitating normal user behaviors, spammers can create social relationships with other users and send unsolicited messages or requests, including the malware URLs, advertising, phishing, deceptive information, and others. Such harmful messages or requests can spread into the entire social network to target victims, thus significantly degrading the quality of user experience, stealing sensitive information, causing economic loss, and even changing victim's political opinions [17, 30]. Therefore, both the research community and social network providers are devoting considerable efforts to develop various solutions [5, 29, 31, 33, 34, 39] for spammer capturing so as to achieve clean and healthy social environments.

Spammers hide among a large amount of normal users, thereby difficult to be mined and classified. To capture spams or spammers, there are two challenges that need to be addressed. First, as there are billions of users exiting in Twitter networks and an average of 8,351 tweets are posting every second [26], it is unrealistic to process the entire dataset of Twitter users and their tweets. The most practical and efficient way is to collect a subset of tweets (or users) that include a large portion of spam messages (or spammers). But how to collect such a subset of the dataset, especially in a real-time manner, is vastly challenging. Second, as the spammers' behaviors and postings are hidden in a large number of benign tweets, how to classify spam messages and associated spammers efficiently from them while guaranteeing high accuracy remains an open problem. Especially nowadays, the spammers keep sending

non-spam messages or evolve with smart spammer techniques to hide themselves.

Many research efforts have been devoted to analyzing the large set of arbitrarily and blindly monitored tweets. These works [6, 8, 12, 20, 34] mainly focus on extracting features from users or tweets and perform classification through learning-based classifiers based on the disparity of attributes among spammers and normal users. This line of solutions can classify spam messages or spammers to some extent, but the efficiency and accuracy are limited. The reason is that they worked on blindly collected tweets, where the workload is extremely high and yet the ratios of captured spams or spammers are relatively low. More importantly, such analytical work extracts features from either users' or tweets' contents and then employs the machine learning classifiers to train attributes independently. However, the intrinsic relationships among users as well as their attributes, that have rich information to reflect spammer's behaviors, are not explored yet.

On the other hand, the honeypot-based spammer capturing methods [18, 19, 28, 36] have been promising for trapping spammers by manually deploying honeypots with specific features meeting spammer's taste. As a result, analytical work on collected users and tweets is significantly reduced. But such a method requires lots of efforts on manual setup, which leads to high deployment overhead. Moreover, some attributes are hard and even impossible to be equipped in artificially created accounts, e.g., creating a honeypot account with a 3-year history. Thus, honeypot solutions unavoidably have the essential drawbacks on deployment flexibility, feature diversity, and network scalability. Besides, their analytical work for spammer classification fails to consider intrinsic relationships among users as well as their attributes. Another line of works [4, 10, 13, 32, 35, 37, 38] have been proposed to analyze user relationships to explore spammer's (called *Sybil*) behaviors. Through modeling friend and follower relationships in social networks by a graph, they relied on graph-based techniques to characterize users' intrinsic relationships so as to classify each user as a *Sybil* node or not. However, these works only focus on exploring users' relationships but the attributes' relationships, which include much richer information to reflect spammer's behaviors, are still totally left out.

In this paper, we first propose the pseudo-honeypot as a novel system framework for users monitoring and tweets collection, with the aim to capture tweets that have much more potentials of including spam messages. The proposed pseudo-honeypot framework harnesses normal users as the parasitic body while taking advantage of those users' diversity and utilizing their associated features as the key resources to attract spammers. By selecting a set of users that have the features of meeting spammer's taste, the pseudo-honeypot can monitor these users' neighboring activities and collect tweets that are more likely to be spam messages. Although the pseudo-honeypot harnesses normal users, the monitoring activities can be controlled in a way that is completely transparent to them and other Twitter users, so as to comply with Twitter terms of the privacy policy. Obviously, the pseudo-honeypot can perform similarly as honeypot on trapping spammers while advancing it in multiple perspectives. It has the salient advantages on deployment flexibility, features variability, and network scalability. Besides, analyzing its collected users and their posting tweets can be significantly quicker while improving the captured spam or spammer ratios.

We then design a new solution, named *TweetScore*, to classify spam messages over collected tweets. Our solution aims to analyze both users and attributes relationships to reflect tweets' characteristics and then use these relationships to score tweets for spam classification. In particular, we extract the "mention" relationships among users and use them to model user's relationships for constructing an *Activity Graph*. At each user, we identify the associated attributes and leverage the *Activity Graph* to construct the attribute relationship graph, called an *Attribute Graph*. With both graphs, we develop a solution based on the "mention" frequency, *UV matrix decomposition*, and *PageRank* algorithm [16] to predict and score attribute relationships (i.e., edges) and attribute values (i.e., vertices). Meanwhile, we use the *Random Walk* algorithm [11] to extract the neighboring user relationships (i.e., walk l steps) and then represent these relationships as attribute scores to quantify each tweet's attributes. In parallel, we also use the *Random Walk* results to model the users' relationships and quantify such relationships into two vectors, i.e., sender vector and receiver vector. We consolidate the attribute scores vector, sender vector, and receiver vector to form a tweet's score vector. Such a vector can reflect both the relevant users' relationships and attributes' relationships of the associated tweet. The neural network is then employed, taking the tweet score vectors as the input, to further mine the deeper intrinsic attributes and users relationships for training before deployed to classify each tweet as a spam or non-spam.

The main contributions of our work are summarized as follows:

- We propose the pseudo-honeypot as a novel system framework and advocate it for efficient spams monitoring in Twitter social networks. The proposed pseudo-honeypot brings salient advantages on development flexibility, features availability, and network scalability when compared to the traditional honeypot-based solutions. By leveraging the pseudo-honeypot, we collect the set of tweets that have a much higher probability of including both spams and spammers. Experimental results confirm that spammer ratios captured by pseudo-honeypot are *four times* as much as those of using a *non pseudo-honeypot* counterpart.
- We design *TweetScore* as a novel spam classification solution by analyzing both neighboring users and attributes relationships to explore the unique characteristics of spams and spammers. With such relationships quantified by vectors, the neural network model is employed to train the relationship among neighboring users and attributes. This method outperforms previous solutions where users and their attribute's relationships are rarely analyzed, despite such intrinsic relationships likely provide much richer information that reflects the disparity of spammer and non-spammer behaviors.
- We implement the proposed pseudo-honeypot system and *TweetScore* solution for online spam detection in Twitter network. By conducting extensive experiments, we demonstrate that *TweetScore* can achieve, on an average, 93.5% accuracy, 93.71% precision, and 1.52% false positive in online spammer detection. Moreover, we show the advantages of *TweetScore* in terms of spammers detection over previous methods.

The remainder of this paper is organized as follows. In Section 2, we discuss our problem and give the necessary definitions. Section 3 presents the system design of pseudo-honeypot and Section 4 proposes a novel spam classification solution, i.e., *TweetScore*, and describes its detailed designs. In Section 5, we implement both the proposed pseudo-honeypot system and *TweetScore* solution in Twitter networks while conducting extensive experiments to demonstrate the performance of pseudo-honeypot network and the accuracy of *TweetScore* solution on spams collection and classification, respectively. We conclude our paper in Section 6.

2 PROBLEM STATEMENT AND DEFINITION

2.1 Problem Statement

This paper studies the spams gathering and spammers detection problem in Twitter social networks. We aim to propose a new system framework for monitoring and collecting tweets that have a higher probability of containing spam messages, and then design a novel solution to classify these tweets as spams or non-spams. There are two challenges to be solved in this problem. First, since the number of tweets postings in Twitter networks is enormous, it is unrealistic to filter all tweets for spam detection due to excessive workload. To address this challenge, we focus on the users that have more potentials of attracting spammer's interest and present a novel framework, called pseudo-honeypot, to harness these normal users while keeping transparency to them. This novel pseudo-honeypot framework enables us to take advantage of users' diversity (including user types, behaviors, features, and others) and leverage such diversity to attract spammers, without manually creating the artificial honeypots but performing similarly to the honeypot. The goal of pseudo-honeypot is to capture tweets that are more likely to include spam messages instead of blindly collecting tweets for classification.

On the other hand, since most of the tweets posted by spammers are benign messages (for the covert purpose), it is challenging for traditional attribute methods (i.e., extract independent attributes or text mining) to find them. In this paper, we focus on the "mention" activities included in each tweet and propose to analyze the relevant user and attribute relationships associated with it. The reason is that the "mention" activities include the most severe spammer behaviors, which thus provide much valuable information to garner spammers. The "mention" relationships from the tweets will be extracted to construct the *Activity Graph* and *Attribute Graph*, respectively. Based on these two graphs, we will develop new solutions to explore both tweet attributes' and users' relationships while quantifying such relationships into a vector of numerical values (i.e., score), so as to mine the discrepancy of spammer and normal users. We introduce a novel solution called *TweetScore* to implement such solution for spam classification.

As our solution depends on two types of graphs: *Activity Graph* and *Attribute Graph*, we give the definitions of these two types of graphs, which will be used later in our design.

2.2 Definition

2.2.1 User Activity Graph. The "mention" activities included in all tweets are extracted to construct the user activity graph. We use a directed graph $G = (V, E)$ to model mention activities among users,

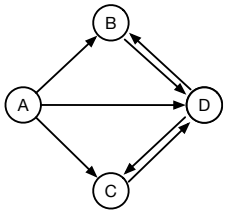

Figure 1: Example of *Activity Graph*. A node denotes a user and a directed edge denotes a mention behavior from sender.

where each vertex $i \in V$ in the graph represents one user and each edge $e(i, j) \in E$ represents the mention activity from users i to j, i.e., i mentions j. For example, if A mentions B, C, and D, while B and D, C and D mention each other, then the constructed activity graph is shown in Figure 1, where A, B, C and D indicate the users while the directed edges among them reflect the mention activities among four users.

2.2.2 User Attribute Graph. Each user has a set of attributes. We define the attribute graph as a directed graph $\bar{G} = (\bar{V}, \bar{E})$ to model the attribute relationships among users. In $\bar{G}$, each vertex represents one unique attribute and each edge represents the relationship between two attributes from two different users. The attribute relationships can be constructed based on the *Activity Graph* G. That is, for any two users i and j while each of them has a set of attributes, if i mentions j, then we assume any one attribute from user i has the directed relationships to any one attribute of user j. Such directed relationships are expressed as the directed edges in the graph $\bar{G}$. In a special case, if one user mentions the other user and both have one same attribute, there will be one directed edge from this attribute to itself. To better understand the *Attribute Graph*, we take the edge $A \rightarrow B$ in Figure 1 as an example. We assume user A has the attributes of *Friends 10, Followers 30*, and *Age 26* while user B has the attributes of *Friends 10, Followers 100,* and *Age 30*. Then, the constructed *Attribute Graph* is shown in Figure 2, where all attributes have directed edges based on $A \rightarrow B$ except the attribute *Friend 10* has directed edge to itself.

3 PSEUDO-HONEYPOT MONITORING SYSTEM

In this section, we present our proposed pseudo-honeypot system for efficient tweets monitoring. The goal of pseudo-honeypot is to collect tweets that are more likely to be spam messages. As discussed in Section 2, the pseudo-honeypots are constructed across normal users, that are more vulnerable to be targeted by spammers, to perform monitoring of activities or behaviors among these users and their neighbors. The main challenges here include: *i)* How to select such a set of users to meet the above criteria while obeying the ethical consideration; *ii)* How to guarantee the effectiveness of pseudo-honeypots.

3.1 Pseudo-honeypot Construction

User's diversity serves as the key resources to construct pseudo-honeypot nodes. Our goal is to select a set of features that have the higher potentials of attracting spammers' interests and then screen

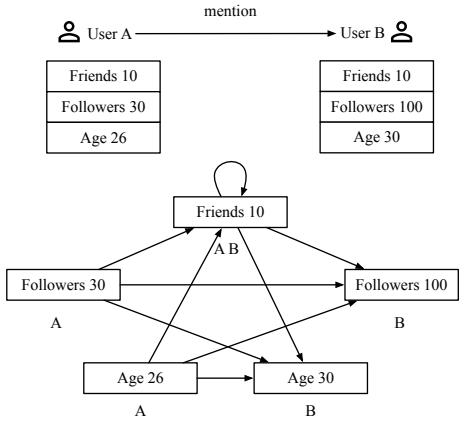

Figure 2: Example of *Attribute Graph*. Each node denotes an attribute of one user. Sender's attributes are fully connected to receiver's attributes.

a set of users accounts that possess these features as the pseudo-honeypot nodes. Obviously, these user accounts are more likely to serve as spammers' target because they meet spammers' taste. The feature identifying procedure is one of the key steps for building pseudo-honeypot systems. Our approach is to leverage the features that have been well explored in previous research [2, 22] with high efficiency in attracting spammers. We can identify a set of most effective ones and utilize them in the pseudo-honeypot system for screening users. After that, we take the reverse engineering strategy to refine the top ones that are more likely to attract spammers. These refined attributes guide our design of a highly effective pseudo-honeypot system.

After identifying a set of effective features, we screen the user accounts that possess these features and use them to serve as the pseudo-honeypot nodes, which surely can attract spammers with a higher likelihood. Note here, the selected users may also be spammers. By constructing pseudo-honeypot on this set of users, the pseudo-honeypot can monitor their activities from nearby users and collect their tweeting activities, thus yielding a much higher probability of including spammer's behaviors. Such a solution can significantly reduce the spammer detection workload and increase the probability of capturing spammers when comparing to the traditional spammer monitoring methods where spammer detection is performed on the blindly monitored massive tweets.

Notably, even the pseudo-honeypot network takes normal users as the parasitic body, its monitoring activities (i.e., data collection) should be completely transparent to the normal users. That is, it should be controlled to be unnoticeable to normal users and is not allowed to perform any social interaction or interference to Twitter users. Moreover, it is not allowed to dig the sensitive or secret information and can only fetch the public information so as to comply with the Twitter terms of the privacy policy. Such functions are supported by the Twitter *RESTful* and *streaming* APIs. By leveraging these APIs, pseudo-honeypot can monitor the users' activities while maintaining transparency to them.

3.2 Online Pseudo-honeypot Monitoring

Since users and their neighbor's activity patterns change over time, the selected users may not always stay attractive to spammers indefinitely. Besides, the spammers may keep changing their targets in order to attack more victims and prevent themselves from being detected. To improve the performance of pseudo-honeypot network, we enable pseudo-honeypot to drift across different sets of users, where on each set of users, pseudo-honeypot stays only for a short period of time and only when these users have adequate activities. To express a period of time, we define the *Prosperous Period* and *Recession Period* for selected users. The *Prosperous Period* represents the time period that users post new tweets and bring lots of mentions and reply activities in a certain time interval, while *Recession Period* denotes the time period that users either do not post new tweets, or post new tweets but bringing few mentions or replies in a certain time interval. We let the pseudo-honeypot to drift among different sets of users and always stay on the set of users in the *Prosperous Period* to perform tweets monitoring.

4 TWEETSCORE

In this section, we present our novel solution, called *TweetScore*, for efficient spam classification. At the core of *TweetScore* is to score tweets by considering the relevant users while quantifying users and their attribute relationships.

We start with an overview of *TweetScore*. Assume the pseudo-honeypot collects a set of tweets, denoted as $\mathcal{D}$. Our goal is to classify each tweet in $\mathcal{D}$ as spam or non-spam. We extract the "mention" activities among users as well as each user's attributes to construct the *Activity Graph* and *Attribute Graph*. For the *Attribute Graph*, we can score the attribute relationships (i.e., each edge) based on the mentioning frequency of the associated users. For attributes that have no relationship in *Attribute Graph*, we predict their values based on the scores of known attribute relationships. After that, we customize the *PageRank* algorithm [16] into our design to score the value of each attribute (i.e., vertex in the *Attribute Graph*). As a result, each attribute has an associated value to reflect both attribute and user relationships. On the other hand, we utilize the constructed *Activity Graph* to perform the *Random Walk* algorithm [11] for identifying the most relevant users (i.e., walk l steps) to this tweet. Then, we can quantify the relevant attribute scores of this tweet by using the start and end users (in the l steps) to evaluate attribute values (from *PageRank* results). In parallel, we also use the *Random Walk* results to model the users' relationships while quantifying such relationships into two vectors, sender vector and receiver vector. We concatenate the attribute scores vector, sender vector, and receiver vector into a single one to represent a tweet's score vector. Notably, such vector can reflect both the relevant users' relationships and attributes' relationships of the associated tweet. In the end, we label a subset of tweets as the ground truth and classify the remaining set of unlabeled tweets using the neural network model.

Figure 3 shows the flowchart of our design. In the following, we give the design details of each step in *TweetScore*.

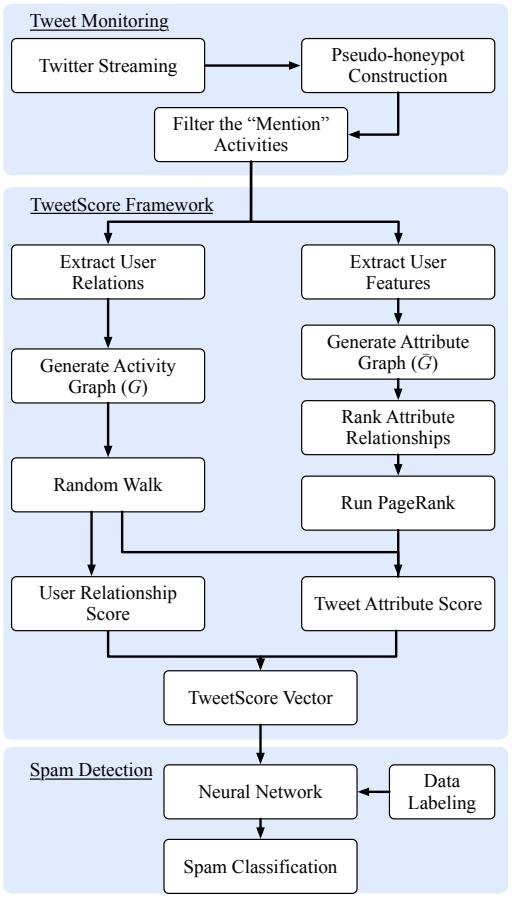

Figure 3: The flowchart of *TweetScore*.

4.1 Constructing Activity Graph

We extract user's *mentioning* activities from $\mathcal{D}$ to construct directed *Activity Graph* G as we discussed in Section 2.2. To model the mention frequency and similarity of two users (i.e., vertices), we define a *weighted Activity Graph* by giving a weight for each edge $\bar{e}(i,j) \in \bar{E}$, as follows:

$$w(i,j) = (1 + \eta \cdot Sim(i,j))M(i,j), \tag{1}$$

where $Sim(i,j)$ reflects the *consine similarity* [27] of activities between users i and j, η is an adjustment coefficient, and $M(i,j)$ counts the frequency of mention activities from i to j. To calculate the *cosine similarity*, we refer Figure 1 and take the vertices A and D as an example. We assume the mention frequency is 1 for each edge in this figure. Since A mentions B, C and D, we define $\vec{A} = [0,1,1,1]$ as the vector of mention frequency from A to A, B, C and D. Similarly, we define $\vec{D} = [0,1,1,0]$ as the vector of mention frequency from D to A, B, C and D. Then, the *cosine similarity* of A and D can be calculated as follows:

$$Sim(A,D) = \frac{\vec{A}\cdot\vec{D}}{\|\vec{A}\|\cdot\|\vec{D}\|} = \frac{<0,1,1,1> \cdot <0,1,1,0>}{\|<0,1,1,1>\|\|<0,1,1,0>\|} = 0.8165$$

4.2 Constructing Attribute Graph

We extract the attributes from user's profile. This set of attributes include **number of friends, number of followers, number of favorites, number of lists, account age, number of tweets, number of retweets, number of mentions, number of hashtags, number of URLs, number of characters in the tweet and number of digits in the tweet**. The previous research [19] has shown that the attribute value of a spammer varies much dramatically than a non-spammer, but it still stays within a relatively small range. Thus, for these attributes whose values vary with time, we define some continuous intervals for each attribute to capture such fluctuation so as to construct the stable vertices in the *Attribute Graph*. Thus, in the *Attribute Graph*, each vertex represents an interval of one attribute. For ease of expression, we abbreviate attribute interval as an *attribute* in the following steps, unless specified.

To construct the *Attribute Graph* $\bar{G}$, we take the user relationships (i.e., edges) from the *Activity Graph* to map the attribute relationships as we have described in Section 2.2. In $\bar{G}$, we quantify the attribute relationships (i.e., assign a weight for each edge in $\bar{G}$) with numerical values by counting the mentions frequency of the associated users. These numerical values represent the scores of attribute relationships. That is, if there is a total number of w_1 mentions from users having attributes x to users having attributes y, then the weight of this edge $\bar{e}(x,y)$ will be w_1, which represents the score of relationships between x and y. Obviously, we can only get the attribute relationships for these edges that are existing in $\bar{G}$, but for these attributes whose associated users have no mention activity, their relationships are unknown yet.

4.3 Scoring Attribute Relationships

We aim to score the relationships between any two attributes. This can be done by prediction based on the known ones in $\bar{G}$. We transform the scored $\bar{G}$ into a matrix expression and denote it as an N_a by N_a matrix $\mathbf{A}$, where both the row and column represent all vertices in $\bar{G}$ (i.e., attributes), N_a represents the total number of all attributes in $\bar{G}$, and the entries in A represent the scores of the attribute relationships. For these entries that do not have attribute relationships in $\bar{G}$, we leave it as blank. Our goal of this step is to predict these blank entries in $\mathbf{A}$.

Here, we employ the UV-Decomposition method, which has been widely used in recommendation system [14] and aims to find an N_a by m matrix $\mathbf{U}$ and an m by N_a matrix $\mathbf{V}$ such that $\mathbf{P} = \mathbf{UV}$ closely approximates to $\mathbf{A}$ for these non-blank entries, while giving predicted values to those blank entries. The general idea of such decomposition is shown below. To simplify our expression, we denote the entries of $\mathbf{U}$, $\mathbf{V}$, $\mathbf{P}$ and $\mathbf{A}$ in row i and column j as u_{ij}, v_{ij}, p_{ij}, and a_{ij}, respectively. The entries in $\mathbf{U}$ and $\mathbf{V}$ are initialized into all ones and then we vary each entry u_{ij} or v_{ij} to find a new value of u_{ij} or v_{ij} that minimizes the Root-Mean-Squarre-Error (RMSE) between $\mathbf{A}$ and $\mathbf{P}$, i.e., $\min(a_{ij} - p_{ij})$. Then, we can claim the new $\mathbf{P}$ approximates to $\mathbf{A}$. For example, we vary the entry u_{rs} (i.e., consider it as a variable) and want to find a new value for it, denoted as λ. This affects only the i-th row in $\mathbf{P}$, so each entry in the new r-th row in $\mathbf{P}$ will be: $p_{rj} = \sum_{k=1}^{m} u_{rk}v_{kj} = \sum_{k=1,k\neq s}^{m} u_{rk}v_{kj} + \lambda v_{sj}$ for each column j where all u_{rk} and v_{kj} are the current values in $\mathbf{U}$ and $\mathbf{V}$. Then, we employ the RMSE method to minimize the differentiation

of **A** and **P**, that is:

$$\min \sum_{j=1}^{j \notin Q} (a_{rj} - p_{rj}), \tag{2}$$

where Q represents the set of column numbers whose corresponding entries in **A** are blank.

To solve Eqn. (2), we take its derivative with respective to λ and set its formula equal to 0, then we can get the value of λ as follows:

$$\lambda = \frac{\sum_{j=1}^{m} v_{sj}(a_{rj} - \sum_{k=1,k\neq s}^{m} u_{rk} v_{kj})}{\sum_{j=1}^{m} v_{sj}^2}. \tag{3}$$

Similarly, we can set the entry of v_{rs} to a variable γ and take the same method to get a new value for v_{rs}, that is:

$$\gamma = \frac{\sum_{i=1}^{N_a} u_{ir}(a_{is} - \sum_{k=1,k\neq r}^{N_a} u_{ik} v_{ks})}{\sum_{i=1}^{N_a} u_{ir}^2}. \tag{4}$$

We iteratively execute above operations and update one entry in **U** or **V** each time until all u_{ij} and v_{ij} values are refreshed in both **U** and **V**. The final $\mathbf{P} = \mathbf{UV}$ will be the approximated matrix to **A** without blank entries. Until now, we have scored all attribute relationships while the *Attribute Graph* $\bar{G}$ can be reconstructed with all attribute relationships.

4.4 Scoring Attributes using PageRank

Now we quantify each attribute's score by using the attribute relationship scores. Our solution is based on the *PageRank* algorithm, which has been implemented by *Google* to analyze the link relationships and rank hyperlinks using numerical values. It has the property of ranking the important website with a higher value while ranking a less important website with a lower value. This property can be customized in our design to offer the attributes that *have different potentials of attracting spammer's interest with various scores*. In particular, if an attribute has more potentials of attracting spammer's interest, it will be assigned with a higher value while another attribute that has fewer potentials will be assigned with a lower value. Here, we customize the *PageRank* algorithm into our design and leverage it as the underlying algorithm to rank the attributes in the *Attribute Graph* based on the attribute relationships.

To employ *PageRank*, we require to have an irreducible matrix. The method proposed in [15] can be leveraged here to transform the **P** into an irreducible matrix $\hat{\mathbf{P}}$. That is, if there exists some rows having all entries with zero in **P**, we use the uniform vector $\frac{1}{N_a}\mathbf{e}^{\mathbf{T}}$ to replace each of these rows to get a new matrix $\bar{\mathbf{P}}$, where **e** is an $N_a \times 1$ vector with all ones and N_a is the total number of attributes, respectively. Next, we transform $\bar{\mathbf{P}}$ into an irreducible matrix $\hat{\mathbf{P}}$ as follows:

$$\hat{\mathbf{P}} = \kappa\bar{\mathbf{P}} + (1-\kappa)\frac{\mathbf{e}\mathbf{e}^T}{N_a}.$$

where κ is a parameter to adjust stochastic perturbation and is set to be 0.85 as suggested in [15].

With the irreducible matrix $\hat{\mathbf{P}}$, we can leverage the *PageRank* algorithm to calculate the scores of all attributes, which are denoted by an $1 \times N_a$ vector π. The vector π is initialized as $\pi^{(0)} = \frac{1}{N_a}\mathbf{e}^T$ and the Power Iteration Method [1] is employed to iteratively update this vector until it is converged. That is, we iteratively calculate $\pi^{(i+1)}$ by following:

$$\pi^{(i)}\hat{\mathbf{P}} = \pi^{(i+1)}, \tag{5}$$

where $\pi^{(i+1)}$ will be normalized by $\pi^{(i+1)}\mathbf{e} = 1$ in each iteration. Once Eqn. (5) is converged, each entry of final converged vector π is the rank of an attribute. In our context, the rank is also considered as the attribute's score. Such final *PageRank* vector is the same as dominant left eigenvector of matrix $\hat{\mathbf{P}}$. Note here, the convergence speed of this method depends on the subdominant eigenvalue which is between 0 and 1. It typically takes no more than 100 iterations.

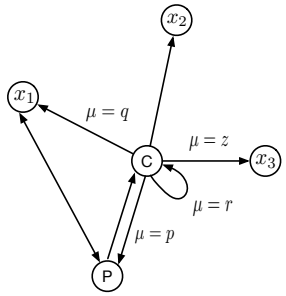

Figure 4: Illustration of random walk method on directed *Activity graph*. The selection of next node depends on the structure of both previous node (i.e., P) and current node (i.e., C).

4.5 Scoring Tweets

After we quantified each attribute using a score, we can map the score on the tweet to reflect its association to a user's behaviors. Here, we evaluate each tweet from most relevant user's attributes instead of only the one that posts this tweet, since the neighbor's behaviors are important factors to reflect a tweet's attribute. The *Random Walk* method is employed here to find the most relevant users by walking l steps. We can use their attribute scores to represent each tweet as an evaluation vector. Such a vector of attribute scores can reflect the trend of a tweet is spam or not to some extent.

We start from the users that are posting this tweet and perform l steps on *Activity Graph* G to find the most relevant users. The challenge here is how to decide which way to go at each step. We define $P_{i,j}$ as the probability of selecting the next node j from a node i as follows:

$$P_{i,j} = \begin{cases} \frac{\phi_{i,j}}{\sum_{X \in N_r(i)}(\phi(i,j))}, & k+1 < l; \\ 0, & \text{otherwise;} \end{cases} \tag{6}$$

where $\phi_{i,j}$ represents the unnormalized probability from i to j, $N_r(i)$ denotes all the neighborhoods of node j, and $\sum_{j \in N_r(i)}(\phi(i,j))$ is used to normalize the probability. The unnormalized transition probability of random walk is calculated as follows:

$$\phi_{i,j} = \mu_{i,j} \cdot w_{i,j}\,, \tag{7}$$

where $w_{i,j}$ is the edge weight in *Activity graph* G (in Eqn. (1)) and $\mu_{i,j}$ is a weighted value showing the types of path that we wish to go. As shown in Figure 4, at node C, there are four paths for selection and the weight of each path is indicated. In our design, we aim to find the next step users that have much more intimate relationships.

The assigned weights are used to measure the activity relationship of a user with its neighbor users. In this figure, p measures the mutual behaviors of two end users (between C and P). That is, the users C and P have the mutual "mention" activities with each other. Thus, these two users have close activity relationships with each other. q measures the inward behavior of a user (from C to x_1 but can also reach to P). In the figure, we can see P, C and x_1 stay in a small group, thus C and x_1 have a higher relevant relationship. z measures the outward behavior to another user (from C to x_3). This reflects the user C's proactive activity to its neighbor x_3. In social networks, most spammers' activities are outgoing edge, i.e., mentioning others. r measures the self-directed activity, thus it shows a weak relationship to its neighbors. Thus, we can assign the weight values in the following order, i.e., $p > q > z > r$, to represent the level of activity relationships of each type of link with its neighbor [1]. Notably, this graph traversal strategy can balance the extreme search strategies of *Depth-first Search* and *Breadth-first Search*.

After performing the *Random Walk*, we obtain a path for each tweet starting from the user that posts it and ending at a destination node. For a tweet, we assume the start and end users are i and k, respectively. We use the attribute scores of users i and k to score this tweet. We define an 1 by $2 * N_b$ vector $\mathbf{Sw}$ to represent the scores of each attribute associated to users i and k,where N_b represents the attributes extracted from user i. Note here the column indicates the attributes, not the attribute intervals. For user i or k, its attribute values may fall into a specific interval, so we take the score of this attribute interval as its attribute score. Then entries of $\mathbf{Sw}$ can be calculated as follows:

$$\mathrm{S}[j] = \frac{i_j + k_j}{2}, \tag{8}$$

$$\mathrm{S}[N_b + j] = i_j - k_j, \tag{9}$$

for columns j ($j \leq N_b$) and $N_b + j$, respectively.

As a result, each tweet can be represented by a vector of scores, which reflects the rich information of the associated behaviors.

4.6 Scoring Users' Dependence Relationships

In addition to score attribute relationships, the random walk paths are used to evaluate the user relationships in a network. For each user u_i, we define a high-dimensional vector variable $v_i \in \mathbb{R}^N$ to represent its activities, where all elements in vector v_i are independent to each other. Such a vector is simply a mapping from a user into a dense vector in $\mathbb{R}^N$. The advantage of such expansion is that the high-dimensional vector can represent more fine-grained and affluent features than a single value of a user [3]. We then quantify a user's activities with other users using a conditional probability model. That is, the probability (denoted as $p(u_j|u_i)$) of a user u_i has some activities with another user u_j can be defined as follows:

$$p(u_j|u_i) = \frac{e^{v_j v_i}}{\sum_{k \in U} e^{v_k v_i}} . \tag{10}$$

The goal here is to find the optimal value v of all users that can reflect neighboring users' activities. Thus, we study the optimization problem with the objective of maximizing the probability of all users activating with their neighbors. Here, the neighboring users represent the *Random Walk* results of each user in the activity graph. Then, this problem can be formulated as follows:

$$\max \sum_{u \in U} \log P_r(N_g(u)|u) , \tag{11}$$

where $N_g(u)$ represents the set of neighborhoods (the users in the random walk path from Section 4.5) for a user u, and P_r represents the conditional probability for a user u activating with her neighborhoods. As all elements in the high dimensional vector are independent to each other, we have:

$$P_r(N_g(u)|u) = \prod_{u_j \in N_g(u)} p(u_j|u) . \tag{12}$$

Put Equations (10), (11), and (12) together, we have:

$$\begin{aligned}
&\max \sum_{u \in \mathcal{U}} \log P_r(\mathcal{N}_g(u)|u) \\
&= \max \sum_{u \in \mathcal{U}} \log \prod_{u_j \in N_g(u)} p(u_j|u) \\
&= \max \sum_{u \in \mathcal{U}} \left[\sum_{u_j \in N_g(u)} \log e^{v_j v} - \sum_{u_j \in N_g(u)} \log \sum_{k \in \mathcal{U}} e^{v_k v} \right] \\
&= \max \sum_{u \in \mathcal{U}} \left[\sum_{u_j \in N_g(u)} v_j v - \sum_{u_j \in N_g(u)} \log \sum_{u_k \in \mathcal{U}} e^{v_k v} \right]
\end{aligned} \tag{13}$$

Note here, v, v_j and v_k are the high dimensional vectors of u, u_j, and u_k, respectively.

From Eqn. (13), we see the objective can be understood as maximizing the likelihoods of all users activating with their neighborhoods (first term in the right hand) while minimizing the likelihoods of users activating with their non-neighboring users (second term in the right hand) at the same time. Thus, Eqn (13) can be approximated and simplified by removing the $\sum$ in the second term, i.e.,

$$\mathrm{OPT}\ \max \sum_{u \in U} \left[\sum_{u_k \in N_g(u)} v_k v - \log \sum_{u_k \in U} e^{v_k v} \right] \tag{14}$$

It is extremely expensive to solve the above optimization problem for a large sized network as we need to consider all users in the networks. Since the neighboring users take a small portion while non-neighboring users take a large portion, we can employ the Negative sampling [21] by just identifying a small Negative sample set from a user μ's non-neighboring users. Denote $N_s(\mu)$ as the selected Negative sample set for a user μ, then OPT can be reformulated into the following approximate expression.

$$\max \sum_{u \in U} \left[\sum_{u_k \in N_g(u)} v_k v - \log \sum_{u_k \in N_s(u)} e^{v_k v} \right] \tag{15}$$

This optimization problem can be solved by using the Stochastic Gradient Ascend method [23]. Then, we can obtain the optimal solutions of the dependence relationship vectors v for all users.

[1] The optimal values of p, q, z and r can be identified through experimental test. Due to the space limitation, we omit its discussion here to conserve space.

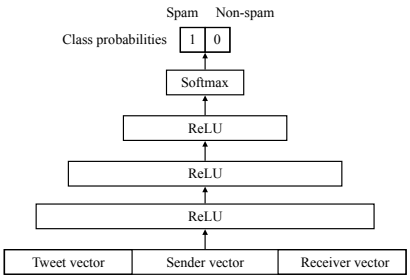

Figure 5: The layer structure of Neural Network model.

4.7 Neural Network Model

Till now, we obtain the score of each tweet **Sw**, its sender's dependent relationship vector v_s, and its receiver's dependent feature vector v_r. These values represent a tweet's characteristics and can be consolidated to a new tweet score vector **S**.

$$\mathbf{S} = \text{concatenate}(\mathbf{Sw}, v_s, v_r) \tag{16}$$

These three entries provide affluent information to reflect both users' and attributes' relationships associated to a tweet. After we obtain the consolidated tweet score vectors of all tweets, we can use a small set of collected data in $\mathcal{D}$ as the training dataset and label them as the ground truth. Then, we propose to employ the neural network to have deeper training of these relationships.

The structure of the neural network that we employed is shown as in Figure 5. The tweet score vector **S** will input to the first layer and then pass to the hidden layer for training, which includes three layers of fully connected Rectified Linear Units (ReLU). At the output layer, we leverage the standard *sigmoid* function which is commonly applied to the two-class logistic regression problem. In the training phase, a cross-entropy loss is minimized with gradient descent on the output of the *sigmoid* function. As we use the standard neural network model here, we omit the detailed description to conserve space.

5 EXPERIMENTS

This section presents the implementation of our advocated pseudo-honeypot system for tweet monitoring (in Section 3) and the use of our *TweetScore* solution on spam classification (in Section 4). Our goal is twofold. First, we show the effectiveness of the pseudo-honeypot system in spams collection. By comparing with its counterparts, i.e., non-pseudo-honeypot and honeypot systems, we illustrate the advantages of pseudo-honeypot system on gathering potential spams messages. Second, we evaluate the efficiency of *TweetScore* in spam classification for comparison with the existing works.

5.1 Implementation

In our implementation, we leverage the *hashtag-based* and *trending-based* features, that have been widely adopted in previous research [2, 22] and also have been demonstrated to effectively attract spammers. The selected features serve as the criteria to identify candidate users for our pseudo-honeypot nodes. Specifically, in *hashtag-based* features, we identify a set of features that have more potentials in attracting spammers, including *entertainment, business, tech, education, environment, social, astrology,* and *general*. Then, we select the top 10 hashtags (from [9]) under each hashtag-based category while for each hashtag, we screen 10 users that possess such a hashtag. For example, in the *entertainment* category, we obtain the top 10 popular hashtags from [9] and select 10 user accounts for each of them. Hence, under the *entertainment* category, we sample a total of 100 pseudo-honeypot nodes out of the top 10 hashtags. Likewise, we also take the top 10 popular hashtags for each of the other hashtag categories and then screen 100 users to serve as pseudo-honeypot nodes following the same way. There are a total of 800 pseudo-honeypot nodes that are constructed under *hashtag-based* features. In *trending-based* features, the following three types of tweets are considered: (1) The tweets with topics of trending up; (2) The tweet with topics of trending down; (3) The tweets with popular topics. For the sake of simplicity, these three types of tweets are abbreviated as *trending-up*, *trending-down*, and *trending-pop*, respectively. We take the top 10 topics under each type from [9] and sequentially screen 10 user accounts for each topic to serve as the pseudo-honeypot nodes. Thus, there is a total of 300 pseudo-honeypot nodes with the *trending-based* features.

In total, we have constructed a group of pseudo-honeypot network with 1, 100 nodes. The time to create one such sized pseudo-honeypot network is less than 1 minute. This significantly reduces the deployment cost when compared to the manual construction of traditional honeypots. Note here, the nodes in pseudo-honeypot network are not immutable and are to shift to another group of users after a specific time, ensuring the pseudo-honeypot network to involve only those users in *Prosperous Period*. This design improves the effectiveness of pseudo-honeypot network in terms of spam messages gathering, as we discussed in Section 3.2. We implement the pseudo-honeypot network in Python with Tweepy library, where the *streaming API* is employed to monitor users and retrieve tweets. The *TweetScore* solution is run on a workstation with dual Intel i5-6600K CPUs and 32 GB RAM.

We run our pseudo-honeypot system and perform spammer detection on gathered data for a total of 700 hours. The time for the pseudo-honeypot network to move to another new group of users is set to be one hour. The selection of the new group of users to serve as pseudo-honeypot nodes follows the same criteria as above. Within the 700-hour experiments, there is a total of 1, 694, 018 tweets collected with these tweets involving a total of 69, 3358 unique user accounts.

Neural Network Setting. As discussed in Section 4.7, the neural network model is employed to train the *TweetScore* vectors and then classify the spam messages and spammers. The neural network parameters are given in Table 1, where the first column represents the layer types that have been shown in Figure 5 and the second column represents the number of neurons that are used at each layer.

In this neural network model, the connection between any two neurons is controlled by a dropout layer with a dropout rate of 0.5. The binary Cross-Entropy is used to model the loss function as follows.

$$Loss = \frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)), \tag{17}$$

Table 1: The number of neurons at each layer in Neural Network model.

Layer Type	# of Neurons
Input	124
Fully Connected +ReLU	512
Fully Connected +ReLU	128
Fully Connected +ReLU	32
Softmax	2

where N denotes the total amount of input tweets, y denotes the label (i.e., 1 for spam and 0 for non-spam), and $p(y)$ is the predicted probability of spams in the N tweets. To train the neural network, we employ the Stochastic Gradient Descent method (SGD) as the optimizer. The learning rate R is initialized as the value of 0.1 and then is updated once for every 10 epochs. Thus, we have $R \leftarrow 0.5^{\lfloor(1+\frac{i}{10})\rfloor}$, where i denotes the training epochs. Moreover, we set the batch size as 100 and the total epoch count equal to 200, respectively.

Solution Comparison. To show the performance of *TweetScore*, we take the following two existing solutions for comparison.

- **SybilSCAR [32]:** SybilSCAR is a structure based method to detect Sybil (i.e., fake) accounts in the social network. In essence, it analyzes the graphic structure among users and classifies them into the benign and Sybil regions. The random walk and belief propagation methods are employed in the classification in SybilSCAR. SybilSCAR iteratively calculates the local rule:

$$\hat{p}^{(t)} = \hat{q} + 2\hat{w}A\hat{p}^{(t-1)}, \tag{18}$$

  where $\hat{p}$ and $\hat{q}$ represent the residual prior probability vector and posterior probability vector respectively. $\hat{w}$ is the parameter determs homophily (edge with same type nodes) strength between two nodes and A is an adjacency matrix of the social graph. According to $\hat{p}^{(t)}$, a threshold (e.g., 0.5) is set to distinguish the benign and Sybil users. This solution can work on our collected dataset to classify spammers and normal users, which are treated respectively as Sybil and benign. But the attributes relationships among users are not considered in this work.
- **Chen6M [7]:** Chen6M is a classification method, on extracting statistic information from the user profiles and tweets. It extracts a total of 12 features, listed in Table 2. After that, the traditional machine learning classifier, i.e., Random Forest, is employed for spam classification.

5.2 Accuracy of TweetScore

We take the first 100-hour data captured by pseudo-honeypot as the training dataset. To label a reliable ground truth dataset, we adopt the diversified approaches to ensure the labeled dataset covers a broad range of spams and spammers that can reflect their characteristics of diversity. That is, we first check suspended accounts to label a set of spams and spammers. Then, we employ the clustering method to group tweets, where Minhash [25] is used to check the similarity of tweets and then cluster them into different groups, The spammers and spams in each group are labeled via the following criteria: 1) if a user in one group is suspended by Twitter, all users in this group are labeled as spammers; 2) if a tweet in one group is labeled as spam, its users and all tweets in this group are labeled as spammers and spams, respectively. After such preprocessing, we can get a roughly labeled ground truth dataset. Lastly, we perform manual checking (by inspecting account and tweets text information) both in the labeled dataset and the remaining unlabeled dataset to refine a reliable ground truth dataset. In the first 100 hours data, we label a total of 15, 097 spammers and 219, 715 non-spams as our ground truth data.

Table 2: Extracted statistical features in [7].

No.	Notation	Description
1	Account age	The days of the account since created
2	follower count	The No. of followers of the account
3	following count	The No. of friends of the account
4	favourites count	The No, of favourites account received
5	lists count	The No. of list the account added
6	tweets count	The No. of tweets the account send
7	retweets count	The No. of retweets the accout send
8	hashtags count	The No. of hashtag in the tweet
9	mentions count	The No. of mentions in the tweet
10	urls count	The No. of urls in the tweet
11	char count	The No. of characters in the tweet
12	digits count	The No. of digits in the tweet.

Table 3: 10-fold cross-validation results.

Classifier	Precision	Accuracy	Recall	F1-macro
AB	0.855	0.872	0.797	0.831
GB	0.811	0.926	0.835	0.852
k-NN	0.760	0.901	0.722	0.820
SybilSCAR	0.661	0.454	0.436	0.445
Chen6M	0.976	0.955	0.852	0.927
TweetScore	0.989	0.967	0.914	0.946

10-**fold cross-validation.** We conduct the 10-fold cross validation on the labeled ground truth dataset to evaluate the performance of *TweetScore*, when compared to SybilSCAR Chen6M, and several traditional machine learning classifiers (i.e., *k Nearest Neighbors (kNN), Gradient Boosting (GB),* and *AdaBoost (AB)*). Table 3 shows the detailed accuracy, precision, recall, and F1-marco ratios of *TweetScore* and different counterpart solutions. From this table, it is evidenced that our *TweetScore* always outperforms other methods in all metrics. In particular, the precision, accuracy, recall, and F1-macro of *TweetScore* are 98.9%, 96.7%, 91.4% and 94.6% respectively. These results confirm the superior efficiency of *TweetScore* on spam classification.

Testing on new data. We take our 100-hour labeled ground truth dataset as the training data and pick a 10-hour data from the remaining 600-hour dataset for testing to show the performance of *TweetScore*, SybilSCAR, Chen6M, and Gradient Boosting (GB). We again adopt the diversified approaches to label this 10-hour data

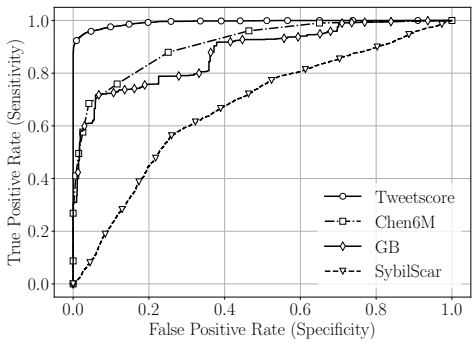

Figure 6: The ROC curves of *TweetScore* with different machine learning classifiers.

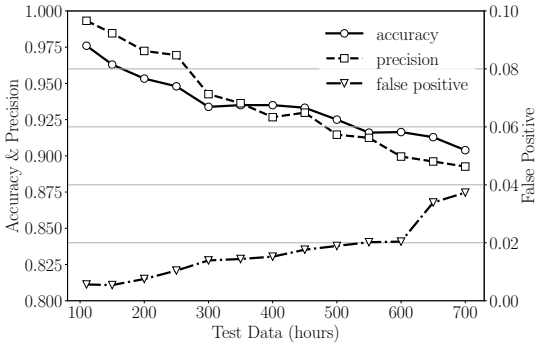

Figure 7: The accuracy, precision, and recall of *TweetScore* in the online spam classification (600 hours).

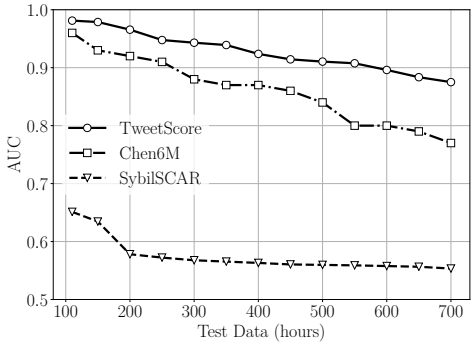

Figure 8: The AUC curves of TweetScore, SybilSCAR and Chen6M on the 600-hour data.

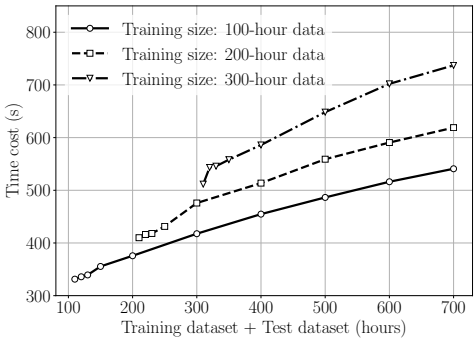

Figure 9: Time cost of *TweetScore* with various training and test dataset sizes.

for comparison. Figure 6 depicts the Receiver Operating Characteristic (ROC) curves of *TweetScore*, SybilSCAR, and Chen6M, and GB. The point on each curve represents the pair of TPR (true positive rate) and FPR (false positive rate) for a given decision threshold. A curve at the upper left represents that the solution has better performance than the one at the lower right. This result demonstrates the accuracy of *TweetScore* is highly competitive.

5.3 Online Learning and Testing Accuracy

We now illustrate the performance of *TweetScore* in online spam detection for a total of 700 hours. As it is technically hard to label all 700-hour dataset, we combine multiple state-of-the-art methods [12, 24, 36] to label most confident spams and keep tracking Twitter system to spot suspend accounts included in our test dataset. In our tweets collection procedure, we implement the *TweetScore*, SybilSCAR, and Chen6M complementing with the pseudo-honeypot to achieve online spam classification.

Accuracy of *TweetScore* in online spam classification. We let the pseudo-honeypot to report the collected tweets every 10 hours. In this experiment, we use the first 100 hours data as the training dataset and employ the *TweetScore* to classify the tweets reported later in every 10 hours. Figure 7 shows the accuracy, precision, and false positive of *TweetScore*. From this figure, the average ratios of accuracy, precision, and false positive of *TweetScore* are 93.50%, 93.71%, and 1.52%, respectively, within the 600 testing hours. These results demonstrate high efficiency and accuracy of *TweetScore* in online spam detection.

Comparison. We next compare the performance of *TweetScore*, SybilSCAR and Chen6M on spam classification within the 600 testing hours. The Area Under the Receiver Operating Characteristic Curve (AUC) is used here to evaluate the classification results. AUC can represent the probability that a classifier will rank randomly selected spam tweet higher than a randomly selected non-spam tweet. A higher AUC score means the better performance of spam detection classifier. Figure 8 exhibits the AUC for *TweetScore*, SybilSCAR and Chen6M, respectively. *TweetScore* is seen to outperform Chen6M and SybilSCAR. With the increasing of pseudo-honeypot running time, AUC values for all these three solutions keep decreasing. But the AUC curve of *TweetScore* decreases much slower than that from SybilSCAR and Chen6M. This demonstrated the advantage of *TweetScore* when compared to the existing solutions in online spam classification.

Scalability of *TweetScore*. This experiment aims to evaluate the scalability of *TweetScore* in terms of time cost by varying the sizes of training and test datasets. We vary the training dataset sizes from 100 hours, 200 hours, to 300 hours data, while the test dataset sizes increment by 10 hours data each time. Figure 9 shows the time cost of *TweetScore* with various training and test dataset sizes. With the 700-hour data, we take the first 100, 200, and 300 hours as the

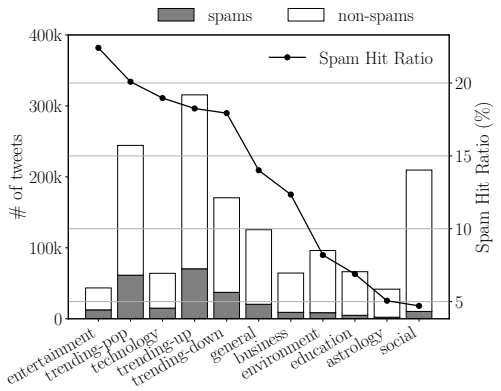

Figure 10: The number of spams and the hit ratios captured in the pseudo-honeypot network using different features.

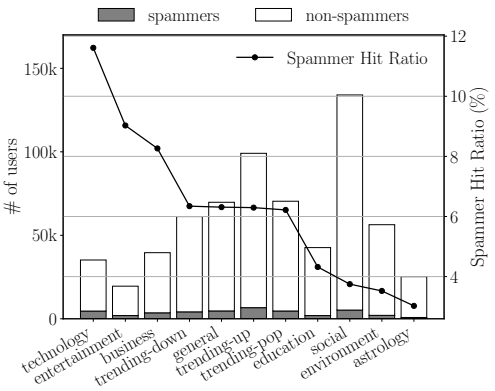

Figure 11: The number of spammers and the hit ratios captured in the pseudo-honeypot network using different features.

training datasets, so the testing process starts after 100, 200 and 300 hours, respectively, in the x-axis of Figure 9.

All three curves in Figure 9 are seen to rise nearly linearly. We find all these three curves are likely linearly increasing in Figure 9. This implies that our *TweetScore* has almost linear complexity with respect to the fixed training dataset size. From these three curves, the running times are found to increase when training dataset sizes grow (from 100 hours to 300 hours) even under the same test dataset size. The reason is that the dominating time cost of *TweetScore* lies in the neural network model training time, which requires longer time to train larger datasets.

5.4 Performance of Pseudo-honeypot Systems

We illustrate the details of captured spams and spammers under each of *hashtag-based* and *trending-based* features. To measure the effectiveness of pseudo-honeypot on spams and spammers collection, we refer to the *hit ratio* as the ratio of captured spam (or spammers) over collected tweets (or users accounts). Denote the hit ratio of spams and of spammers by H_r and $\hat{H}_r$, respectively. Figure 10 depicts the number of spams and its hit ratio captured by pseudo-honeypots under each feature. From this figure, we observe pseudo-honeypots captures more spams with *trending-based* features than with *hashtag-based* features. Specifically, we collect a total of 169, 121 spams using *trending-based* features and a total of 83, 158 spams using *hashtag-based* features. The hit ratios of the top three *hashtag-based* features are 22.41% (*entertainment*), 18.97% (*technology*), and 14.01% (*general*). On the other hand, ratios of *trending-pop, trending-up,* and *trending-down* are 20.09%, 19.97%, and 18.25%, respectively.

Figure 11 illustrates the number of spammers and the hit ratio captured by pseudo-honeypots under each feature. We can see *technology, entertainment, business, trending-down, general,* and *trending-up* to be the top 6 features that have the highest spammer hit ratios of 11.61%, 9.03%, 8.26%, 6.34%, 6.31% and 6.29%, respectively. The results from Figures 10 and 11 can guide us in the future design of pseudo-honeypots by selecting features with high hit ratios.

From Figures 10 and 11, we select the features of the top 5 spam *hit ratios* and the top 5 spammer *hit ratios*. These features are used to sample a more effective pseudo-honeypot system. That is, the

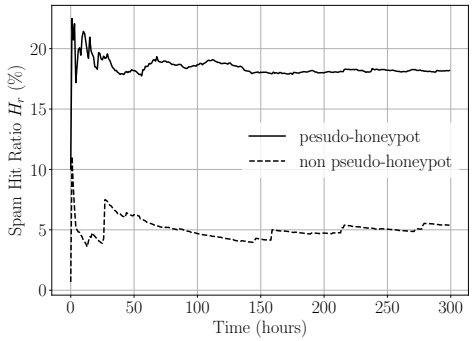

Figure 12: The comparison of spam hit ratio under pseudo-honeypot and *non pseudo-honeypot* solutions within 300 hours of experiments.

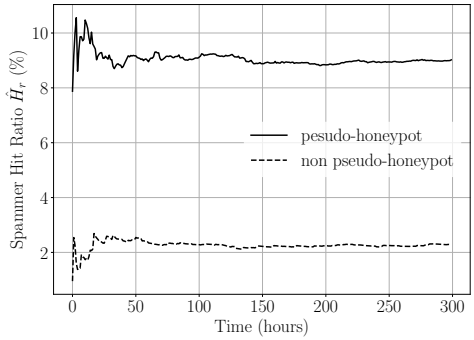

Figure 13: The comparison of spammer hit ratio under pseudo-honeypot and *non pseudo-honeypot* solutions within 300 hours of experiments.

pseudo-honeypot network randomly selects 300 users accounts, with each one possessing at least one of these features. The pseudo-honeypot network shift time is set to be 1 hour. For comparison, we arbitrarily select 300 user accounts every hour to perform tweets monitoring and execute this experiment with a total of 300 hours. This experiment is called as the *non pseudo-honeypots* counterpart.

Comparison to non pseudo-honeypot system. Figures 12 and 13 show the comparison of H_r and $\hat{H}_r$, respectively, for *non pseudo-honeypot* and pseudo-honeypot systems. According to Figures 12 and 13, both H_r and $\hat{H}_r$ of the pseudo-honeypot solution are almost four times more than those of *non pseudo-honeypot*. This demonstrates the advantages of pseudo-honeypot in capturing tweets (user accounts) that include a higher probability of spam messages (spammers).

Comparison to honeypot-based solutions. A large-sized honeypot systems is hard to be constructed, especially with some specific attributes. We select to compare the results of earlier prominent studies, i.e., Stringhini [28], Lee [18], and Yang [36]. We find that one pseudo-honeypot node can capture an average of 1.03 spammers per hour, whereas each honeypot node in Stringhini [28], Lee [18], and Yang [36] captures 0.0067, 0.12, 0.087, spammers per hour, respectively. This demonstrates the advantage of pseudo-honeypot system over the existing honeypot system in terms of spammers capture efficiency.

6 CONCLUSION

This paper has explored the novel pseudo-honeypot framework and *TweetScore* solution for efficient spam monitoring and classification in the Twitter network. The pseudo-honeypot network is constructed over users with features that have much more potentials of attracting spammers, thus significantly lifting the spam ratio included in the collected tweets when compared to collecting tweets blindly. Additionally, *TweetScore* allows us to explore the intrinsic attribute relationships among neighboring users of respective tweets. By scoring these relationships into a vector of numerical values, we profile the users' relationship and their tweets for better spam classification. We implemented the pseudo-honeypot system in Twitter network for tweet monitoring and collection, and employed *TweetScore* to perform the spam classification based on 700-hour collected data. The experiments have demonstrated that the pseudo-honeypot yields *four times* and *eleven times* spammer capture ratios when compared respectively with *non pseudo-honeypot* and the traditional honeypot systems. The results confirm that *TweetScore* achieves 93.50% accuracy, 93.71% precision, and 1.52% false positive in online spam detection.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their fruitful comments. This research was supported in part by Louisiana Board of Regents under Contract Number LEQSF(2018-21)-RD-A24.

REFERENCES

[1] Arasu, A., Novak, J., Tomkins, A., and Tomlin, J. Pagerank computation and the structure of the web: Experiments and algorithms. In *ACM International World Wide Web Conference (WWW), Poster Track* (2002), pp. 107–117.

[2] Benevenuto, F., Magno, G., Rodrigues, T., and Almeida, V. Detecting spammers on twitter. In *Collaboration, Electronic messaging, Anti-abuse and Spam conference (CEAS)* (2010).

[3] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of Machine Learning Research 3*, Feb (2003), 1137–1155.

[4] Boshmaf, Y., Logothetis, D., Siganos, G., Lería, J., Lorenzo, J., Ripeanu, M., and Beznosov, K. Integro: Leveraging victim prediction for robust fake account detection in osns. In *Network and Distributed System Security Symposium (NDSS)* (2015), pp. 8–11.

[5] Castillo, C., Donato, D., Gionis, A., Murdock, V., and Silvestri, F. Know your neighbors: Web spam detection using the web topology. In *ACM SIGIR Conference on Research and Development in Information Retrieval* (2007), pp. 423–430.

[6] Chen, C., Wang, Y., Zhang, J., Xiang, Y., Zhou, W., and Min, G. Statistical features-based real-time detection of drifted twitter spam. *IEEE Transactions on Information Forensics and Security 12*, 4 (2017), 914–925.

[7] Chen, C., Zhang, J., Chen, X., Xiang, Y., and Zhou, W. 6 million spam tweets: A large ground truth for timely twitter spam detection. In *IEEE International Conference on Communications (ICC)* (2015), IEEE, pp. 7065–7070.

[8] Chen, W., Yeo, C. K., Lau, C. T., and Lee, B. S. A study on real-time low-quality content detection on twitter from the users' perspective. *PloS One 12*, 8 (2017).

[9] Corporation, L. Hashtag analytics for your brand, business, product, service, event or blog. http://www.hashtags.org, 2018.

[10] Danezis, G., and Mittal, P. Sybilinfer: Detecting sybil nodes using social networks. In *Network and Distributed System Security Symposium (NDSS)* (2009), pp. 1–15.

[11] Grover, A., and Leskovec, J. node2vec: Scalable feature learning for networks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data mining* (2016), pp. 855–864.

[12] Herzallah, W., Faris, H., and Adwan, O. Feature engineering for detecting spammers on twitter: Modelling and analysis. *Journal of Information Science 44*, 2 (2018), 230–247.

[13] Jia, J., Wang, B., and Gong, N. Z. Random walk based fake account detection in online social networks. In *Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2017), pp. 273–284.

[14] Koren, Y., Bell, R., and Volinsky, C. Matrix factorization techniques for recommender systems. *Computer*, 8 (2009), 30–37.

[15] Langville, A. N., and Meyer, C. D. Deeper inside pagerank. *Internet Mathematics 1*, 3 (2004), 335–380.

[16] Langville, A. N., and Meyer, C. D. *Google's PageRank and beyond: The science of search engine rankings*. Princeton University Press, 2011.

[17] Larsson, A. O., and Moe, H. Studying political microblogging: Twitter users in the 2010 swedish election campaign. *New Media & Society 14*, 5 (2012), 729–747.

[18] Lee, K., Caverlee, J., and Webb, S. Uncovering social spammers: Social honeypots + machine learning. In *ACM SIGIR Conference on Research and Development in Information Retrieval* (2010), pp. 435–442.

[19] Lee, K., Eoff, B. D., and Caverlee, J. Seven months with the devils: A long-term study of content polluters on twitter. In *ICWSM* (2011).

[20] Mccord, M., and Chuah, M. Spam detection on twitter using traditional classifiers. In *International Conference on Autonomic and Trusted Computing* (2011), pp. 175–186.

[21] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems (NIPS)* (2013), pp. 3111–3119.

[22] Miller, Z., Dickinson, B., Deitrick, W., Hu, W., and Wang, A. H. Twitter spammer detection using data stream clustering. *Information Sciences 260* (2014), 64–73.

[23] Robbins, H., and Monro, S. A stochastic approximation method. In *Herbert Robbins Selected Papers*. Springer, 1985, pp. 102–109.

[24] Sedhai, S., and Sun, A. Semi-supervised spam detection in twitter stream. *IEEE Transactions on Computational Social Systems 5*, 1 (2018), 169–175.

[25] Shrivastava, A., and Li, P. In defense of minhash over simhash. In *Artificial Intelligence and Statistics* (2014), pp. 886–894.

[26] Site, W. W. W. A. Twitter usage statistics. http://www.internetlivestats.com/one-second/#tweets-band, 2019.

[27] Steinbach, M., Karypis, G., and Kumar, V. A comparison of document clustering techniques. In *KDD Workshop on Text Mining* (2000), pp. 525–526.

[28] Stringhini, G., Kruegel, C., and Vigna, G. Detecting spammers on social networks. In *Annual Computer Security Applications Conference (ACSAC)* (2010), pp. 1–9.

[29] Thomas, K., Grier, C., Song, D., and Paxson, V. Suspended accounts in retrospect: an analysis of twitter spam. In *ACM SIGCOMM conference on Internet measurement conference* (2011), pp. 243–258.

[30] Thomas, K., McCoy, D., Grier, C., Kolcz, A., and Paxson, V. Trafficking fraudulent accounts: The role of the underground market in twitter spam and abuse. In *USENIX Security Symposium* (2013), pp. 195–210.

[31] Wang, A. H. Don't follow me: Spam detection in twitter. In *IEEE International Conference on Security and Cryptography (SECRYPT)* (2010), pp. 1–10.

[32] Wang, B., Zhang, L., and Gong, N. Z. Sybilscar: Sybil detection in online social networks via local rule based propagation. In *IEEE International Conference on Computer Communications (INFOCOM)* (2017), pp. 1–9.

[33] Wang, G., Wang, T., Zheng, H., and Zhao, B. Y. Man vs. machine: Practical adversarial detection of malicious crowdsourcing workers. In *USENIX Security Symposium* (2014), pp. 239–254.

[34] Wu, T., Liu, S., Zhang, J., and Xiang, Y. Twitter spam detection based on deep learning. In *ACM Australasian Computer Science Week Multiconference* (2017), pp. 3:1–3:8.

[35] Yang, C., Harkreader, R., Zhang, J., Shin, S., and Gu, G. Analyzing spammers'

social networks for fun and profit: a case study of cyber criminal ecosystem on twitter. In *ACM International World Wide Web Conference (WWW)* (2012), pp. 71–80.

[36] Yang, C., Zhang, J., and Gu, G. A taste of tweets: reverse engineering twitter spammers. In *ACM Annual Computer Security Applications Conference (ACSAC)* (2014), pp. 86–95.

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

[38] Yu, H., Kaminsky, M., Gibbons, P. B., and Flaxman, A. Sybilguard: defending against sybil attacks via social networks. In *ACM SIGCOMM Computer Communication Review* (2006), vol. 36, pp. 267–278.

[39] Zhang, Y., Zhang, H., Yuan, X., and Tzeng, N.-F. Pseudo-honeypot: Toward efficient and scalable spam sniffer. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2019).