Detecting Malicious Accounts in Online Developer Communities Using Deep Learning

Qingyuan Gong[1,2], Jiayun Zhang[1,2], Yang Chen[1,2], Qi Li[3], Yu Xiao[4], Xin Wang[1,2], Pan Hui[5,6]
[1]School of Computer Science, Fudan University, China
[2]Shanghai Key Lab of Intelligent Information Processing, Fudan University, China
[3]Institute for Network Sciences and Cyberspace, Tsinghua University, China
[4]Department of Communications and Networking, Aalto University, Finland
[5]Department of Computer Science, University of Helsinki, Finland
[6]CSE Department, Hong Kong University of Science and Technology, Hong Kong
{gongqingyuan,jiayunzhang15,chenyang,xinw}@fudan.edu.cn,
qli01@tsinghua.edu.cn,yu.xiao@aalto.fi,panhui@cs.helsinki.fi

ABSTRACT

Online developer communities like GitHub provide services such as distributed version control and task management, which allow a massive number of developers to collaborate online. However, the openness of the communities makes themselves vulnerable to different types of malicious attacks, since the attackers can easily join and interact with legitimate users. In this work, we formulate the malicious account detection problem in online developer communities, and propose GitSec, a deep learning-based solution to detect malicious accounts. GitSec distinguishes malicious accounts from legitimate ones based on the account profiles as well as dynamic activity characteristics. On one hand, GitSec makes use of users' descriptive features from the profiles. On the other hand, GitSec processes users' dynamic behavioral data by constructing two user activity sequences and applying a parallel neural network design to deal with each of them, respectively. An attention mechanism is used to integrate the information generated by the parallel neural networks. The final judgement is made by a decision maker implemented by a supervised machine learning-based classifier. Based on the real-world data of GitHub users, our extensive evaluations show that GitSec is an accurate detection system, with an F1-score of 0.922 and an AUC value of 0.940.

CCS CONCEPTS

• **Security and privacy → Social aspects of security and privacy**.

KEYWORDS

Online Developer Community, Malicious Account Detection, Deep Learning, Social Networks

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM '19, November 3–7, 2019, Beijing, China
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357971

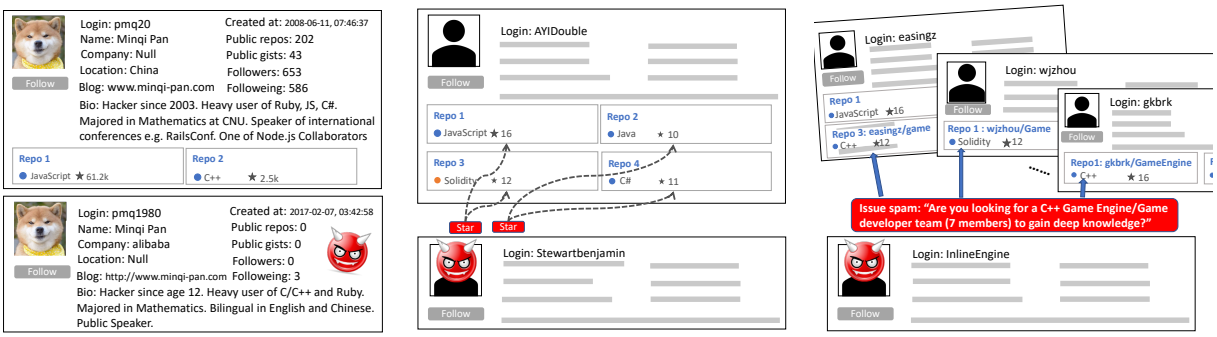
ACM Reference Format:
Qingyuan Gong[1,2], Jiayun Zhang[1,2], Yang Chen[1,2], Qi Li[3], Yu Xiao[4], Xin Wang[1,2], Pan Hui[5,6]. 2019. Detecting Malicious Accounts in Online Developer Communities Using Deep Learning. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357971

1 INTRODUCTION

Due to the flourishing software markets and the increasing development complexity, collaborative software development has become a trend. Online developer communities provide platforms for developers to collaborate in software development projects and to build social interactions. A number of online communities have been launched, such as GitHub and BitBucket, gathering millions of developers together. Such communities become a unique type of online social networks (OSNs) [16]. Different from generic OSNs like Facebook and Twitter, these online developer communities target a special group of users who conduct software development and code sharing. Thus, the activities within these communities are mostly related to collaborative software development. These communities also offer social networking functionalities. For example, a developer can follow other developers to receive notifications about the projects updates. For the users working on the same project, they can leave comments to each other, and can utilize the platforms to assign and manage programming tasks.

Online developer communities are generally open to anyone who would like to join. Similar with other OSNs, they are not exempted from threats of malicious activities, since malicious users can join these platforms conveniently. For example, GitHub is a representative developer community that had attracted more than 31 million users by Sep. 30, 2018[1]. However, nearly 21.50% of users were labeled as malicious users according to our measurement, which is a significant number. Malicious users are able to perform different types of harmful activities. We show three examples from real-world user data in Fig 1. In Fig. 1(a), an attacker creates a fake identity to impersonate an existing legitimate user, which is indistinguishable for visitors. This kind of identity impersonation attack [10] helps the attackers exploit the reputation of the victims. The attacker in Fig. 1(b) generates fake "stars" to make one user's

[1]https://octoverse.github.com/, accessed on Sep. 1, 2019.



(a) Identity Impersonation (b) Fake Stars to Repositories (c) Issue Spams to Repositories

Figure 1: Examples of Attackers in Online Developer Communities

repositories look popular. Fig. 1(c) shows that an attacker generates spams on GitHub automatically, sending a "Game developer" advertisement to as many game-related repositories as possible.

Detecting malicious users is an important and challenging problem in OSNs. There are several proposals [2, 4, 11, 12] on malicious account detection in OSNs. Behavioral patterns of users in online developer communities are quite different from generic OSNs. Developers mainly perform a series of development-centric activities such as creating repository, uploading/downloading source code, and sending pull request. The social interactions among developers are also highly related to software development. These unique characteristics introduce difficulties for applying existing solutions. To our knowledge, there lacks an approach which is known to be useful in distinguishing malicious accounts from legitimate ones in online developer communities.

The key to detect malicious users is to find the distinguishable factors from the user data. The data generated by OSN users can be divided into two categories. One is the descriptive data shown on each user's profile page, including the username, biography, company, and statistical indices such as the number of followers, followings or repositories. The other category is the activity data that logs users' activities. Online developer communities usually support a richer set of user activities (e.g. GitHub produces 42 types of events related to user activities). It is important to have an efficient framework for handling multimodal data and for solving the complexity of dynamic user activities.

In this paper, we conduct a data-driven study to explore the distinctive behavioral properties of legitimate and malicious users, and develop an algorithm to accurately detect malicious users based on these properties. We select GitHub for case study. We have made the following three key contributions. First, we formulate the problem of malicious account detection in online developer communities. We involve the idea of sequential analysis to study the development-centric and dynamic user activities and find the difference between legitimate and malicious users. Second, we design and implement GitSec, a new deep learning-based framework, which is able to make use of both the descriptive information and dynamic activity information to detect the malicious users. We integrate the Phased LSTM neural network [23] with the attention mechanism [33] to build a neural network-based architecture that can efficiently deal with the two related activity sequences. Third, we evaluate the prediction performance of GitSec using a dataset collected from GitHub. We compare GitSec with a series of existing solutions, and demonstrate the advantages of GitSec. According to our evaluation, GitSec can achieve a high detection performance, with an F1-score of 0.922 and an AUC value of 0.940.

2 BACKGROUND AND DATA COLLECTION

2.1 Background of GitHub

GitHub is a representative developer community, which has helped the online development of softwares and attracted more than 31 million developers around the world. GitHub regards each user activity as an *event*, such as the *create event* for a new repository or branch created. GitHub supports 42 types of events in total. Typical user activities include creating a new repository, cloning an existing repository, pulling the latest changes of a repository from GitHub, and committing and pushing locally made changes to the shared repository. GitHub hosts more than 96 million repositories, including popular open source projects like Linux Kernel, Python and TensorFlow. Through GitHub, developers are able to communicate with each other, assigning and claiming programming tasks through publishing issues under a repository. In addition, the conventional "following" function is also supported, allowing users to receive notifications on the status updates of any users on this platform. In these online communities, developers interact with each other with a main focus on collaborative development and code sharing, forming a special kind of social network.

2.2 GitHub Data Collection

Each GitHub user has a numeric user ID, which is assigned in an ascending order. The earlier a user signed up, the smaller user ID she has. In our work, we only consider the GitHub users which had been registered by Dec. 31, 2017. To obtain an unbiased user dataset, we use ID-based random sampling to implement the data crawling. Note that some numeric IDs do not have corresponding user accounts, and our crawler skipped these IDs. For each user, we use GitHub *users* API (https://api.github.com/user/ID) to access her descriptive information. We did the data crawling from Jun. 20, 2018 to Aug. 27, 2018.

We crawled the data of 10,667,583 randomly selected GitHub users including the demographic information, social connections and statistical indices of the activities shown on the user's profile page, such as the number of followings/followers/repositories. Besides the descriptive information shown on users' profile pages, GitHub stores the event data representing the dynamic activities of users. However, we are only allowed to access the latest 300 events of each users. Also, the events older than 90 days can no longer be accessed. Therefore, a user's historical events cannot be all collected through a single round of crawling. Luckily, the GHArchive project[2] records the public GitHub event timeline since February 2012 using periodical crawling. We further collect the event data of the crawled users from GHArchive.

GitHub has already annotated the malicious accounts, which can be adopted as the "ground-truth" to evaluate the prediction performance of our approach in malicious account detection. Profile pages of malicious users have been blocked by GitHub, whereas their basic information can still be accessed using GitHub API[3]. Therefore, our crawler is able to check if this user is a malicious user by further accessing the profile page https://www.github.com/username during the data collection. If the HTTP status code returned "404", showing that the profile page of this user has been blocked, this user will be labeled as a malicious one. If the returned HTTP status code is "200", this user will be annotated as a legitimate one. Among all users we have crawled, 78.50% of them are legitimate, and 21.50% of them are malicious. We take the obtained labels as the ground-truth to evaluate the prediction performance of our malicious account detection system. In particular, we focus on the users who have generated at least three events during the previous three years. Among these users, we randomly select 59,857 of them to form our dataset.

2.3 Ethical Issues

GitHub allows data crawling of users' public information for research purpose. Our data collection followed the "terms of service" of GitHub[4], and all information we collected was publicly accessible. We have consulted GitHub about our research and received their approval. Also, our study was reviewed and approved by the Research Department of Fudan university.

3 DYNAMIC USER ACTIVITY ANALYSIS

The public data of each GitHub user consists of a *descriptive part* and a *dynamic part*. The descriptive part mainly refers to the information about a user's profile and a set of statistical metrics of her activities. The dynamic part covers the fine-grained records of the activities users have generated. The descriptive part has been widely used to extract features to tell the difference between legitimate and malicious users in online communities [1, 8, 18, 34]. However, there exist attackers disguise themselves by creating profiles that look legitimate, such as the identity impersonation attack. Different from the demographic information, which is easy to manipulate, dynamic user activity information can give a detailed and informative view of user behavior in a long duration. Taking the fine-grained activity into consideration would lead to a higher chance to detect malicious behaviors.

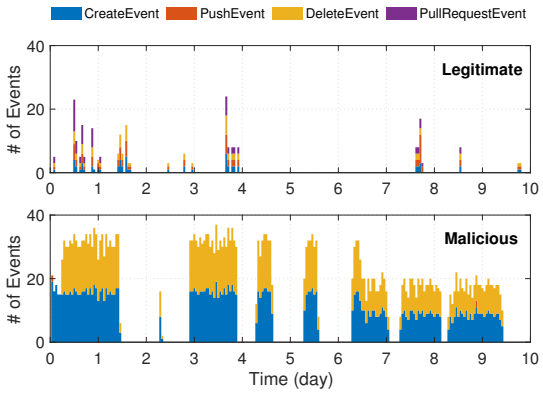


Figure 2: Examples of Dynamic Activities for GitHub Legitimate/Malicious Users

We plot the amount and types of events generated by one malicious user in our dataset, also the normal event logs of a legitimate user in Fig. 2. We can see this malicious user kept creating and deleting repositories or branches on GitHub with a high frequency in each hour. It generated a mass of *create* and *delete* events in the first 9 days. On the 10-th day, the user was blocked by GitHub. The legitimate user reveals different dynamic activity patterns. In this paper, we will make use of the users' dynamic behaviors to build a more reliable malicious account detector. We propose to use sequential analysis to deal with the dynamic activity patterns, applying the LSTM (long-short term memory) networks (Section 3.1), and its recent variation, PLSTM (Phased LSTM) networks (Section 3.2).

3.1 LSTM Structure

LSTM network [15] is a representative type of RNN (Recurrent Neural Network) that is able to deal with the long-distance dependencies between the elements in a sequence. An LSTM network is composed of an array of recurrently connected cells. The LSTM cells process one input element x_t at each time slot. It maintains a cell state variable c_t through the entire job process, memorizing the information left in this cell until the current time slot. The output of the cell is represented by the variable h_t, called a hidden state, which will be fed into the next cell in the $(t+1)$-th time slot with the input x_{t+1}. Each cell contains a number of neurons, which determines the dimension of the corresponding hidden state.

At the t-th time slot, the LSTM cell deals with the input x_t and h_{t-1} with three "gates", which is a striking feature of LSTM network. The gates control the fraction of information that is input into and output from the cell. There are input gate i_t, output gate o_t and forget gate f_t controlling the two inputs of h_{t-1} and x_t, using the sigmoid function σ to produce the following variables valued in the range $[0, 1]$.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad (1)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad (2)$$

[2]https://www.gharchive.org/, accessed on Sep. 1, 2019.
[3]Differently, self-deleted accounts cannot be accessed by the API.
[4]https://help.github.com/en/articles/github-terms-of-service, accessed on Sep. 1, 2019.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{3}$$

The cell state c_t will be updated in two steps. The cell first operates on the inputs h_{t-1} and x_t through a *tanh* function and produces an intermediate result $\tilde{c_t}$ valued in the range $[-1, 1]$

$$\tilde{c_t} = \tanh(W_c[h_{t-1}, x_t] + b_c). \tag{4}$$

W_f, W_i, W_o, W_c, and b_f, b_i, b_o, b_c are the weights and bias operators, respectively, which will be learned through back-propagation during the model training process.

Then the cell incorporates the information from the input and forget gates to update the state variable c_t as

$$c_t = f_t c_{t-1} + i_t \tilde{c_t}. \tag{5}$$

The cell status variable keeps updating until all the elements in the sequence have been processed and memorized through the entire process.

In addition, each cell generates a hidden state, which is regarded as the cell output of the corresponding time slot. The generation of the hidden state incorporates the status variable and the information from the output gate, which can be expressed as

$$h_t = o_t \tanh(c_t). \tag{6}$$

We adopt the cross entropy function to measure the classification loss and determine the parameters W_x and b_x involved in the model through back-propagation.

$$L_{neural} = -\sum_{i \in U} \hat{y}_i log(y_i) \tag{7}$$

where U denotes the user instances in the training dataset, while $\hat{y_i}$ and y_i are the prediction result about the user u_i indicated by the final output of the LSTM network, and the ground-truth label of her, respectively.

3.2 Phased LSTM Structure

The dynamic activities of different developers in online communities are often sparse and distributed in a wide time range. There exist some active developers who generate events frequently on GitHub, but also others who only produce very few events. It is hard to sample the events of all users at a same rate. However, LSTM networks regard the elements in the input sequence equally and update the cell state when processing each element. This results in a low efficiency if we construct the event sequence of each user to accommodate the very active developers, or a performance decrease if we just ignore the sampling irregularity of the sequence and treat each element equally. PLSTM [23] extends the standard LSTM network by adding an additional gate over the updates of the cell status. Instead of updating the cell state variables in each time slot, PLSTM network introduces a new gate k_t to control the updates of the status variables c_t and h_t, according to the sampling rate of the input sequence. The updating equations of c_t and h_t are changed into

$$c_t = k_t(f_t c_{t-1} + i_t \tilde{c_t}) + (1 - k_t)c_{t-1} \tag{8}$$

$$h_t = k_t(o_t \tanh(c_t)) + (1 - k_t)h_{t-1} \tag{9}$$

Comparing with the updating equations in LSTM, i.e., Eq. (5) and Eq. (6), the updates of the cell statuses in PLSTM work in the way that if the gate k_t is closed, the status variable c_{t-1} of the previous time slot will be maintained. If k_t is open, the status variable c_t of the current cell will be updated accordingly. Three neuron-specific parameters τ, $\mathcal{S}$, and γ_{on} are introduced to determine the value of k_t. τ represents the length of one entire period for a neuron, containing the open phase and the close phase of the gate. γ_{on} is the ratio of the open phase in the entire range τ. $\mathcal{S}$ controls the phase shift of the neurons. Neurons in each PLSTM cell share the same $\mathcal{S}$ and γ_{on}, but have a different value of τ. All parameters are learned in the training process.

With the help of an auxiliary variable ϕ_t, the value of k_t can be expressed as a step function below.

$$\phi_t = \frac{(t - \mathcal{S}) mod\ \tau}{\tau} \tag{10}$$

$$k_t = \begin{cases} \frac{2\phi_t}{\gamma_{on}}, & \text{if } \phi_t < \frac{1}{2}\gamma_{on} \\ 2 - \frac{2\phi_t}{\gamma_{on}}, & \text{if } \frac{1}{2}\gamma_{on} < \phi_t < \gamma_{on} \\ \alpha\phi_t, & \text{otherwise} \end{cases} \tag{11}$$

t corresponds to the sampling timestep and α are global parameters often set as 0.001. In each period, the "openness" of k_t rises from 0 to 1, then drops from 1 to 0, and then keeps closed. For each neuron, the value of ϕ_t reflects which phase is in for the current time. In the open phase, the status variables update at the degree controlled by k_t according to Eq. (8) and Eq. (9). The gate k_t keeps closed in the close phase when there is no valid input to neurons, but still allows important gradients pass through at the rate α, propagating useful gradients to the next cell. Under the control of the gate k_t, PLSTM is able to deal with asynchronously sampled sequences efficiently.

4 DESIGN OF THE MALICIOUS USER DETECTION SYSTEM

In this section, we propose the malicious user detection framework called GitSec. GitSec makes use of both the descriptive data and the fine-grained activity data to learn the difference between legitimate and malicious users in developer communities.

4.1 System Overview

GitSec is a two-stage detection system. The framework is shown in Fig. 3. In the first stage, GitSec leverages the *sequential analysis module* to deal with the dynamic data, including a pair of PLSTM networks and an attention layer. Users' activities on GitHub are formulated into two sequences, i.e., a sequence of time intervals between the successive events, and a sequence of event types. The two sequences are fed into two PLSTM networks, respectively. Each cell in a PLSTM network generates a hidden state. The attention layer over the two parallel PLSTM networks assigns weights to all hidden states involved. The hidden states which are not relevant to the classification will be diminished. The attention layer finally generates an output concatenating all the hidden states. In the second stage, GitSec incorporates the *descriptive features module* to extract features from users' static information. Aggregating these feature sets with the output from the first stage, GitSec exploits a

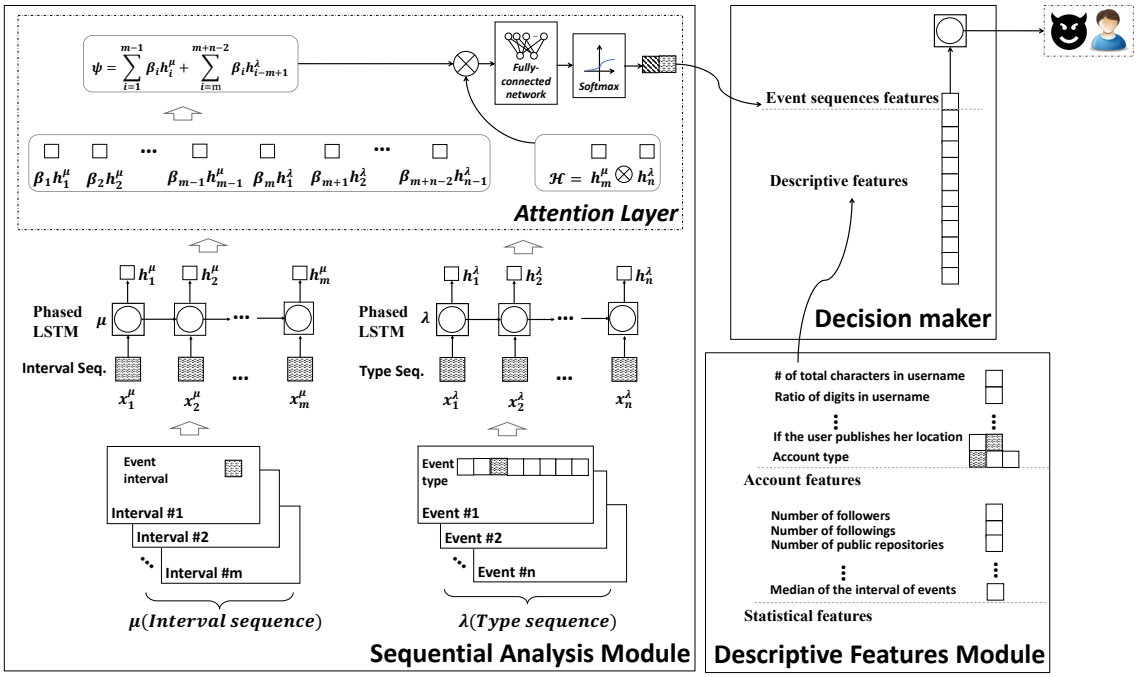


Figure 3: Illustration of GitSec Framework

decision maker to make the final judgement about whether the user is malicious or legitimate.

In the following subsections, we introduce the building blocks of GitSec and their contributions to the final decision. The parallel PLSTM networks to conduct the sequential analysis layer over users' dynamic activities, and the attention layer functioning over the hidden states of the PLSTM cells are shown in Section 4.2. The descriptive features layer and decision maker of GitSec are introduced in Section 4.3 and Section 4.4, respectively.

4.2 Sequential Analysis Module

Developers generate a variety of events following the software development procedures. The type of each activity expresses what the developers are doing, while the frequency of events can be used for measuring users' activeness.

In GitSec, we run two PLSTM networks in parallel. Each of them deals with one perspective of dynamic activities, i.e., intervals between successive events and the types of events. The intervals between successive events are normalized and form a sequence of length m, as the event interval elements from #1 to #m in Fig. 3. The types of the events are represented by one-hot encoding and form a sequence of length n, shown as the event type elements from #1 to #n in Fig. 3, $n = m + 1$.

The PLSTM networks μ (event interval sequence) and λ (event type sequence) process the two sequences independently, dealing with the elements one by one in the input sequence. The generated hidden states form the sequences of $\{h_1^\mu, h_2^\mu, ..., h_m^\mu\}$ and $\{h_1^\lambda, h_2^\lambda, ...h_n^\lambda\}$. h_t^μ or h_t^λ represents the information that has been learnt by processing the first t elements in the sequence. In PLSTM networks, each cell only takes the previous information into consideration for their operations, generating a hidden state to be fed into the next cell. Existing studies often take the final hidden state generated by the last cell as the output [12, 21, 27], memorizing the information generated by all cells. However, as discussed in [20, 33], each hidden state might record some useful information. In addition, each of the two PLSTM networks represents the dynamic activities from one aspect, and these two sequences are correlated to each other. The attention mechanism is a flexible approach to balance the weights of different hidden states according to their actual importance, instead of taking one hidden state as the output directly. Therefore, we apply an attention layer on top of the parallel PLSTM networks to further enhance the prediction by considering the hidden states of the two networks together. In specific, we apply the concatenation-based attention [20] to align the intermediate effect of the hidden states of the cells in the parallel LSTM networks.

In Fig. 3, $H^\mu = \{h_1^\mu, h_2^\mu, ..., h_m^\mu\}$ is the set of hidden states of the PLSTM network μ, while $H^\lambda = \{h_1^\lambda, h_2^\lambda, ..., h_n^\lambda\}$ is the set of hidden states generated by the cells in the PLSTM network λ. We first use a multi-layer perceptron (MLP) neural network to concatenate the last hidden states of the two PLSTM networks as $\mathcal{H} = W^\mu h_m^\mu + W^\lambda h_n^\lambda$. We introduce the intermediate variable v_i to evaluate the correlations between $\mathcal{H}$ and each hidden state

$$v_i = \begin{cases} \boldsymbol{v}^T \tanh\left(W_\mathcal{H}\mathcal{H} + W_h h_i^\mu\right), & i \in \{1, 2, ..., m-1\} \\ \boldsymbol{v}^T \tanh\left(W_\mathcal{H}\mathcal{H} + W_h h_{i-m+1}^\lambda\right), & i \in \{m, m+1..., m+n-2\} \end{cases} \tag{12}$$

where $W_{\mathcal{H}}, W_h \in R^{a\times d}$, $\boldsymbol{v}^T \in R^{1\times a}$. The value of a corresponds to the attention size parameter. d is the dimension of a hidden state, which is equal to the number of neurons in each cell.

The attention weight assigned to the hidden state h_i produced by the PLSTM network is expressed as the attention vector

$$\beta_i = \frac{exp(v_i)}{\sum_{q=1}^{m+n-2} exp(v_q)},\ i \in \{1, 2, ..., m+n-2\}. \tag{13}$$

The two individual PLSTM networks are connected through the attention layer. The hidden state of each cell is absorbed into the output by considering the joint effect of the two sequences.

Measuring the importance of each hidden state, we obtain the weighted hidden state vector

$$\psi = \sum_{i=1}^{m-1} \beta_i h_i^{\mu} + \sum_{i=m}^{m+n-2} \beta_i h_{i-m+1}^{\lambda}. \tag{14}$$

The vector ψ covers the former $m-1$ and $n-1$ hidden states. Concatenating with $\mathcal{H}$ which contains the m-th and n-th hidden states, it forms the joint output of the parallel PLSTM networks as $P = W_1\mathcal{H} + W_2\psi$, where W_1 and W_2 are the weight parameters for the concatenated output.

GitSec further employs a fully connected network to transform P into a two-dimensional vector $P' = W_P P + b_P$, where W_P and b_P are weight and bias parameters of the fully connected network that will be learned during the model training.

Then a softmax function transforms the elements of the two-dimensional vector into two probabilities p_1 and p_2, signifying whether the user is a legitimate or a malicious user from the view of event sequence.

$$p_i = \frac{exp(P_i')}{exp(P_1') + exp(P_2')},\ i \in \{1, 2\}. \tag{15}$$

The obtained values represent the likelihood of this user to be a malicious user concerning her fine-grained activities. These two probabilities become the output of the sequential analysis module, which will be regarded as two event features of the user and fed into the decision maker.

4.3 Descriptive Features Module

A user's descriptive data on her profile page is also informative. In Fig. 4(a)-Fig. 4(e), we compare the profiles between legitimate and malicious users according to several pre-selected metrics, i.e., average time interval between successive events, the number of characters in username, the number of followings, followers and repositories. We use Welch's t-test [32] to study the difference between the malicious and legitimate users concerning each of these five metrics. For each metric, we calculate the corresponding p-value. If the p-value is under 0.05, we can conclude that the malicious and legitimate users are significantly different in terms of the selected metric. For each of all the five metrics, we find that the p-value of the Welch's t-test is smaller than 0.001, indicating that these two types of users are quite different. In summary, malicious users generate successive events on GitHub with much shorter time intervals. Their usernames always contain more characters. Malicious users have fewer followings and followers than legitimate users. Also, they create much less number of repositories. The tendencies are consistent throughout different measurement indices. In addition, there are some optional information fields within each GitHub user's profile. In Fig. 4(f), we show the proportions of malicious users and legitimate users who have enabled each selected optional information field. We can see malicious users tend to fill out fewer information fields.

Table 1: Feature Subsets of GitSec (E: Event Features, A: Account Features, S: Statistical Features)

Set	Type	Illustration
E	float	· Probability of being legitimate by Sequential Analysis Module
	float	· Probability of being malicious by Sequential Analysis Module
A	integer	· Number of total characters in username
	float	· Ratio of digits in username
	integer	· Number of digits in username
	integer	· Number of symbols and special characters in username
	bool	· If the user fills the nickname
	bool	· If the user publishes a bio
	bool	· If the user publishes an authentic email
	bool	· If the user has blog
	bool	· If the user is looking for a job
	bool	· If the user publishes her company
	bool	· If the user publishes her location
	list	· Account type, developer: [0,0,1], organization: [0,1,0] and bot: [1,0,0]
S	integer	· Number of followers
	integer	· Number of followings
	float	· Ratio of the number of followers and followings
	integer	· Number of public repositories
	integer	· Number of public gists
	float	· Average of the interval of events
	float	· Variance of the interval of events
	float	· Minimum of the interval of events
	float	· Maximum of the interval of events
	float	· Median of the interval of events

After obtaining the event features in the first stage, GitSec further extracts the descriptive features from each user's profile page as the account features and statistical features, shown in Table 1. The account feature subset covers the username, nickname, their willingness for job search on GitHub, and the published personal information. The statistical feature subset includes the statistics of social connections, the number of her public repositories and gists, and the event related indices of her.

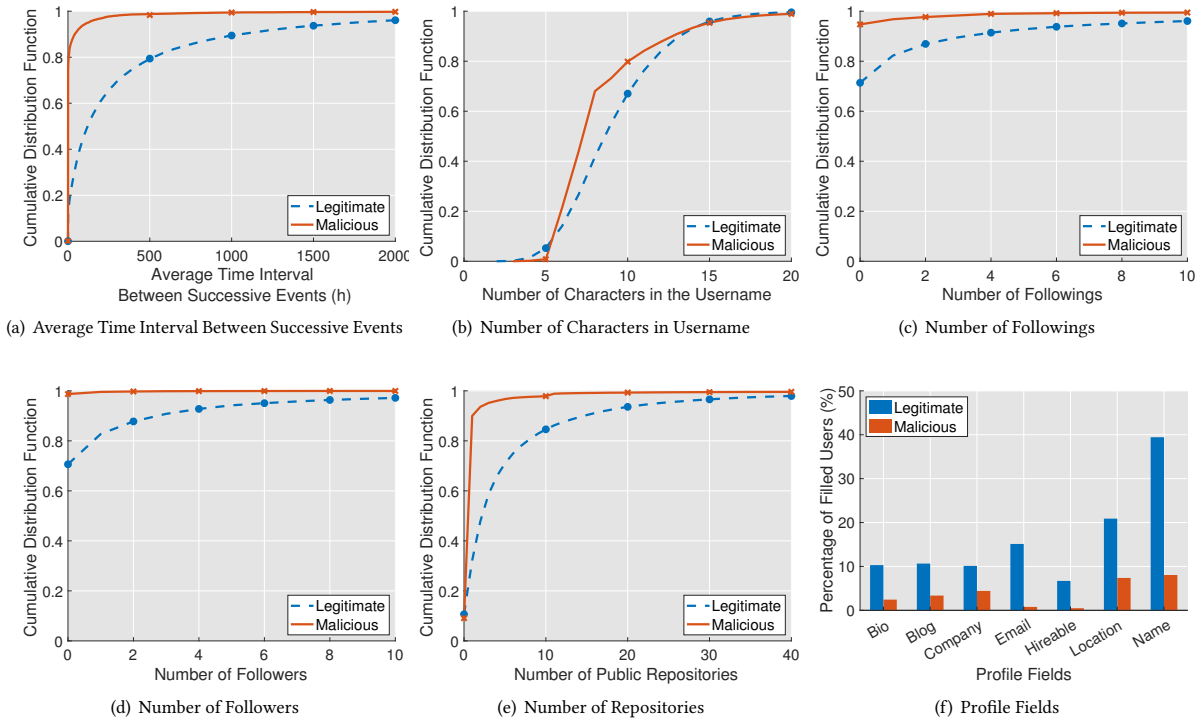
4.4 Decision Maker

GitHub employs a decision maker to conduct the final judgement. The decision maker could be the classifiers using supervised machine learning algorithms such as Random Forest [3], C4.5 decision tree [26], or recently proposed boosting systems such as XGBoost [6] and CatBoost [25]. The decision maker decides whether the user is legitimate or malicious. It outputs "0" or "1", showing whether this user is a legitimate or a malicious user.

We tune the parameters in the classifier by referring to the default log loss function. The parameters are tuned by sweeping a grid of possible values to optimize the log loss function

$$L = -\frac{\sum_{i=1}^{N}(y_i log(p_i) + (1-y_i) log(1-p_i))}{N}, \tag{16}$$

where N is the total number of the user instances examined, p_i is the prediction probability that the user i is classified correctly.



(a) Average Time Interval Between Successive Events (b) Number of Characters in Username (c) Number of Followings

(d) Number of Followers (e) Number of Repositories (f) Profile Fields

Figure 4: Behavioral Difference Between Legitimate and Malicious Users on GitHub

Once a parameter set is given, we can get an L value. We select the set of parameters that could help the decision maker achieve the lowest L value. Afterwards, given the data generated by the user u_i, the trained decision maker will be able to judge whether u_i is a malicious user.

5 IMPLEMENTATION AND EVALUATION

In this section, we implement GitSec in Section 5.1, and evaluate its detection performance using the real-world user data collected from GitHub in Section 5.2.

5.1 Implementation and Metrics

Our crawled dataset consists of 59,857 users in total, with 44,892 legitimate users and 14,965 malicious ones. We construct a training and validation dataset, and a test dataset in a ratio of 7:3. In both the datasets, the ratio of legitimate and malicious users keeps the same as in the entire dataset. We use 5-fold cross validation for the training and validation dataset.

To implement the sequential analysis module, we use TensorFlow, a widely used open source machine learning framework. Concerning the parameters of neural networks, we set the hidden size as 64, meaning that there are 64 neurons in each cell. The learning rate is set as 0.01. For the attention layer, the attention size of each mechanism is set as 50. To implement the decision maker, we use scikit-learn [24], a Python-based machine learning library.

We use the following classic metrics to evaluate the detection performance of GitSec.

- Precision: the fraction of detected malicious accounts which are really malicious.
- Recall: the fraction of malicious accounts who have been uncovered accurately.
- F1-score: the harmonic mean of precision and recall.
- AUC [9]: the probability that this classifier will rank a randomly chosen malicious user higher than a randomly selected legitimate user.

5.2 Experimental Results

There are several key designs in GitSec, including the parallel neural networks in the sequential analysis module, the attention layer to align the hidden states produced by the neural networks, and the classifiers used in the decision maker to conduct the final judgement. In the evaluation, we experiment different constructions of GitSec by involving different neural network models, attention mechanisms, and decision makers to evaluate the performance. Also, we examine the contribution of different features to the final decision, and compare GitSec with some state-of-the-art solutions.

5.2.1 Comparison of different neural network models. Ma et al. [21] proposed to use four types of recurrent neural networks to detect rumors in microblogs, i.e., standard RNN, one-layer LSTM, one-layer GRU[5] and two-layer GRU. In addition, Clockwork RNN (CW-RNN) [17] and Dilated RNN (D-RNN) [5] are two other recently

[5]Gated recurrent units (GRU) [7] is a gating approach in RNNs, which has an update gate and a reset gate for each gated recurrent unit. It is computationally more efficient than LSTM, but could achieve a comparable prediction performance.

Table 2: Evaluation on Different Attention Mechanisms

Models	Precision	Recall	F1-score	AUC
PLSTM with the combined time series seq.	0.915	0.825	0.868	0.900
Parallel PLSTM	0.928	0.883	0.905	0.930
Parallel PLSTM + AttentionLoc	0.928	0.887	0.906	0.931
Parallel PLSTM + AttentionConcat	0.924	0.892	0.907	0.934

emerging RNNs. For comparison, we measure the performance of these 6 networks on our dataset.

We run different neural network models on the event type sequence (Type seq.) and the event interval sequence (Interval seq.), respectively, and compare their performance in Fig. 5. From the results we find that PLSTM performs the best for both the sequences.

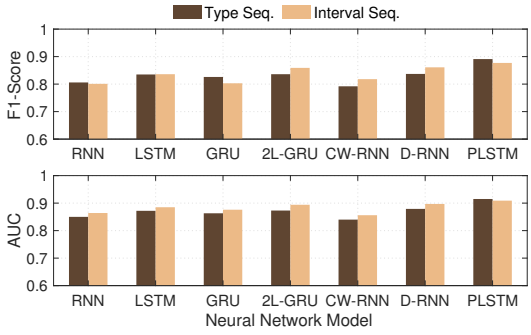


Figure 5: Evaluation on Different Neural Networks

5.2.2 Comparison of different attention mechanisms. Before evaluating the attention mechanisms, we compare the design of the parallel sequences and integrating the event type and event interval as a single sequence. To merge the two sequences, we check user's generated events on an hourly basis, and construct a new sequence that records activities in each hour. Considering the 14 types of events generated by the users in our dataset, each element in the sequence is a 32-dimensional vector describing the events generated by the user within that particular hour, expressing the total number of events within one hour, the largest number of events within one minute in this hour, the variance of the number of events in each minute, the mean of the time interval between successive events, the amounts of different types of events, and the ratios of different types of events. From Table 2 we see that the parallel design produces higher F1-score and AUC value than taking the combined time series sequence as the input. This verifies the necessity of using a parallel design to deal with the developer activities.

We further evaluate different attention methods using the parallel PLSTM networks. We apply the two attention methods mentioned in [20], i.e., the concatenation-based attention (AttentionConcat) and the location-based attention (AttentionLoc). We can see the performance gain of using both the attention methods in Table 2. With the two attention methods, AUC values increase by 0.001 and 0.004 for AttentionLoc and AttentionConcat respectively, when using parallel PLSTM networks to deal with the event interval and event type sequences. The advantage of the attention methods shows the importance of considering the relations between two event sequences.

Table 3: Evaluation on Different Decision Makers

Models	Parameters	Precision	Recall	F1-score	AUC
CatBoost (GitSec)	learning_rate=0.1, depth=9, iterations=135, l2_leaf_reg=8, border_count=122, one_hot_max_size=3	0.951	0.896	0.922	0.940
XGBoost	learning_rate=0.1, n_estimators=66, max_depth=4, min_child_weight=4, subsample=0.6, colsample_bytree=0.9, γ=0.1	0.956	0.886	0.920	0.936
RF	max_depth=7, n_estimators=130, max_feature='log2', min_sample_split=2, min_sample_leaf=8	0.950	0.883	0.915	0.934
SVM	C=256, gamma=0.1	0.942	0.886	0.913	0.934
LR	tol=10^{-4}, penalty='L1', C=256	0.934	0.892	0.912	0.935

5.2.3 Comparison of different decision makers. The decision maker in GitSec fits for various classifier algorithms. In our experiment, we examine the prediction performance of GitSec by applying different classifiers including tree boosting systems such as CatBoost and XGBoost, and classic machine learning algorithms including Random Forest (RF), support vector machine (SVM) and Linear Regression (LR). The resulted prediction performance of using each algorithm is shown in Table 3. We utilize McNemar's test [22] to calculate the statistical significance in order to test if any two classification algorithms are significantly different. We find that CatBoost is significantly different from any other classifiers (p-value < 0.05, McNemar's test). From the prediction performance concerning the four metrics we see that CatBoost achieves the highest scores considering both the F1-score and AUC metrics. In the following experiments, we employ CatBoost as the decision maker to implement GitSec.

5.2.4 Contribution of different feature subsets. The CatBoost classifier is fed with three feature subsets shown in Table 1. In this part of experiment, we test the contribution of difference feature subsets through comparing the performance of GitSec. In each experiment, we first subtract one subset of features at a time and examine the performance degradation of GitSec. It can be found in Table 4 that event features are the most important ones to the accuracy, as the classifier ignoring event features produces the lowest F1-score and AUC value. Second, we conduct experiments on GitSec starting from a random guess classifier. Then we add one feature subset at a time, to see how the performance increases. The results also show that event features are the most important ones for increasing performance, and statistical features are the second most important ones. From the results we see that only feeding the event features to the CatBoost, we can reach the F1-score of 0.911, while feeding the statistical features results in an F1-score of 0.883. Compared with event features and statistical features, account features play a less important role in the process of classification.

Table 4: Evaluation on Different Feature Subsets

Approach	Precision	Recall	F1-score	AUC
GitSec	0.951	0.896	0.922	0.940
- Event Features	0.928	0.854	0.889	0.916
- Account Features	0.947	0.890	0.918	0.937
- Statistical Features	0.939	0.892	0.915	0.936
Random Guess	0.248	0.495	0.330	0.497
+ Event Features	0.935	0.889	0.911	0.934
+ Account Features	0.594	0.678	0.633	0.762
+ Statistical Features	0.923	0.846	0.883	0.911

Table 5: Feature Importance (E: Event Features, S: Statistical Features, A: Account Features)

Rank	Importance	Feature	Category
1	29.31	Probability of Legitimate	E
2	12.30	Probability of Malicious	E
3	7.87	Maximum of the interval of events	S
4	5.23	Average of the interval of events	S
5	5.12	Variance of the interval of events	S
6	4.87	Median of the interval of events	S
7	4.39	Minimum of the interval of events	S
8	4.13	# of followers	S
9	4.09	# of public repositories	S
10	3.23	# of total characters in username	A

Table 6: Performance Comparison with Existing Solutions

Models	Precision	Recall	F1-score	AUC
GitSec	0.951	0.896	0.922	0.940
Gong et al. [12]	0.932	0.844	0.886	0.912
Al-Qurishi et al. [1]	0.870	0.804	0.836	0.882
Viswanath et al. [30]	0.479	0.937	0.634	0.799

5.2.5 Contribution of individual features. We further examine the importance of individual features in all feature subsets. CatBoost calculates the importance of the features according to their contribution to the prediction result. We list the importance of each feature for the classification in Table 5. The results show that the two event features contribute the most to the judgement of malicious users. The following seven of the top 10 important features are all from the statistical subset, consistent with the importance of the feature subsets. The number of characters in username is also an important feature in the account feature subset.

5.2.6 Comparison with state-of-the-art solutions. We compare the performance of GitSec with the following malicious user detection algorithms for online communities. We evaluate them with our collected GitHub developer dataset.

- Gong et al. [12]: We proposed the DeepScan algorithm, which could be used to detect malicious users in location-based social networks (LBSNs). We formulated the user activities into a single sequence and used a bidirectional LSTM network to deal with this sequence. Also, we used a XGBoost-based classifier to make the final detection.
- Al-Qurishi et al. [1]: They used a Random Forest classifier to identify malicious accounts in large-scale social networks. In particular, they applied principal component analysis (PCA) to process the selected user features, and fed the resulted principal components into the classifier.
- Viswanath et al. [30]: They utilized PCA to process typical user features, in order to separate the principle components of users' features into the normal space. The other components were taken into the residual space to identify malicious users.

Results of the comparisons are shown in Table 6. We can see that GitSec outperforms all these three approaches. DeepScan, another deep learning-based approach, ranks the second. Compared with DeepScan, GitSec increases the F1-score by 0.036 and the AUC value by 0.028.

6 RELATED WORK

6.1 User Behavior Analytics for Developers

Based on the rich publicly-visible information collected from online developer communities, such as GitHub, researchers made contributions on understanding the user behavior from different aspects. Li et al. [19] explored the prediction of the duration before a revision was accepted using the social graph data of GitHub. Tsay et al. [28] explored the influence of social and technical factors for the evaluation on contributions of pull requests. Vasilescu et al. [29] went deep into the continuous integration (CI) function, evaluating the effect of this function. Gousisos et al. [14] studied the contributors on GitHub in terms of the motivations, work habits and challenges. In this work, we study the malicious account detection problem in online developer communities for the first time. We formulate the problem and propose a deep learning-based approach to achieve an accurate detection.

6.2 Malicious Account Detection in OSNs

There are a number of literatures focusing on the detection of malicious users in OSNs [2, 4, 8, 11, 12]. Representative solutions to detect malicious accounts in OSNs include the social graph analysis, and machine learning-based classification using statistical features. The key to malicious user detection is to find the difference between malicious and legitimate users.

Some malicious account detection schemes rely on analyzing the social graph, based on the intuition that malicious accounts have limited social connections to legitimate users. Existing approaches include SybilRank [4], SybilBelief [11] and Integro [2]. This kind of approaches requires the knowledge of the entire social graph, which is very difficult for third-party application providers to obtain. Also, for some communities which are not social-oriented, using social connections will miss some discriminative signals revealed by other user activities.

There are also some machine learning-based approaches to identify malicious accounts. Kumar et al. [18] studied sockpuppetry across 9 online discussion communities. They applied a Random Forest classifier to identify sockpuppets. Yang et al. [34] introduced an SVM classifier to distinguish between legitimate users and sybil (fake) accounts. Similarly, Wang et al. [31] used the clickstream data of Renren to build an SVM classifier to uncover fake accounts. In this kind of approaches, feature selection is critical. However, most of the existing approaches have not considered the fine-grained dynamic activities of users. In our work, we demonstrate that the dynamic user activities play an important role in identifying malicious accounts.

6.3 Deep Learning-Based Approaches to Study Online User Behavior

Deep learning-based technologies have been used in studying online user behavior, in particular, in user classification according to different criteria. For example, Ma et al. [21] proposed to use recurrent neural networks to detect rumors from microblogs. They evaluated four different types of RNNs, i.e., basic tanh-RNN, one layer LSTM, one layer GRU and 2-layer GRU. Suhara et al. [27] proposed the DeepMood system, which is able to forecast depressed mood according to a user's self-reported histories, using the standard LSTM network. Differently, in this work, GitSec is designed for the online developer communities, which are totally different in terms of user composition and the basic functions. From the perspective of deep learning methodology, we adopt PLSTM, which is more suitable to deal with the dynamic software development activity data. Moreover, attention mechanism has been applied to further improve the detection performance.

7 CONCLUSION AND FUTURE WORK

In this paper, we formulate and explore the malicious account detection problem in online developer communities. In particular, we pick a representative platform, i.e., GitHub, for case study. We design and implement GitSec, a deep learning-based malicious account detection system, which takes the fine-grained dynamic activities generated by users into consideration. We adopt the latest advances of the recurrent neural networks to improve the prediction performance. According to our evaluation based on the activity data of 59,857 GitHub users, GitSec achieves an F1-score of 0.922. GitSec clearly outperforms the state-of-the-art approaches.

For future work, we plan to evaluate GitSec with the data of other online developer communities, since users tend to have accounts on multiple OSNs [13]. Also, we wish to collaborate with some developer communities to take back-end user activities into consideration, by integrating the clickstream information and the entire social graph into our solution.

ACKNOWLEDGEMENT

This work is sponsored by National Natural Science Foundation of China (No. 61602122, No. 71731004, No. 61572278 and No. U1736209), the Research Grants Council of Hong Kong (No.16214817) and the 5GEAR project from the Academy of Finland. Yang Chen is the corresponding author.

REFERENCES

[1] Muhammad Al-Qurishi, M. Shamim Hossain, Majed A. AlRubaian, Sk. Md. Mizanur Rahman, and Atif Alamri. 2018. Leveraging Analysis of User Behavior to Identify Malicious Activities in Large-Scale Social Networks. *IEEE Transactions on Industrial Informatics* 14, 2 (2018), 799–813.

[2] Yazan Boshmaf, Dionysios Logothetis, Georgos Siganos, Jorge Lería, José Lorenzo, Matei Ripeanu, and Konstantin Beznosov. 2015. Integro: Leveraging Victim Prediction for Robust Fake Account Detection in OSNs. In *Proc. of NDSS*.

[3] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.

[4] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. 2012. Aiding the Detection of Fake Accounts in Large Scale Social Online Services. In *Proc. of NSDI*.

[5] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael J. Witbrock, Mark A. Hasegawa-Johnson, and Thomas S. Huang. 2017. Dilated Recurrent Neural Networks. In *Proc. of NIPS*. 76–86.

[6] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. of ACM KDD*.

[7] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. of EMNLP*.

[8] Zi Chu, Steven Gianvecchio, Haining Wang, and Sushil Jajodia. 2012. Detecting Automation of Twitter Accounts: Are You a Human, Bot, or Cyborg? *IEEE Trans. Dependable Sec. Comput.* 9, 6 (2012), 811–824.

[9] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27, 8 (2006), 861–874.

[10] Oana Goga, Giridhari Venkatadri, and Krishna P. Gummadi. 2015. The Doppelgänger Bot Attack: Exploring Identity Impersonation in Online Social Networks. In *Proc. of ACM IMC*.

[11] Neil Zhenqiang Gong, Mario Frank, and Prateek Mittal. 2014. SybilBelief: A Semi-Supervised Learning Approach for Structure-Based Sybil Detection. *IEEE Transactions on Information Forensics and Security* 9, 6 (2014), 976–987.

[12] Qingyuan Gong, Yang Chen, Xinlei He, Zhou Zhuang, Tianyi Wang, Hong Huang, Xin Wang, and Xiaoming Fu. 2018. DeepScan: Exploiting Deep Learning for Malicious Account Detection in Location-Based Social Networks. *IEEE Communications Magazine* 56, 11 (2018), 21–27.

[13] Qingyuan Gong, Yang Chen, Jiyao Hu, Qiang Cao, Pan Hui, and Xin Wang. 2018. Understanding Cross-site Linking in Online Social Networks. *ACM Transactions on the Web* 12, 4 (2018), 25:1–25:29.

[14] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective. In *Proc. of ICSE*.

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (Nov. 1997), 1735–1780.

[16] Long Jin, Yang Chen, Tianyi Wang, Pan Hui, and Athanasios V. Vasilakos. 2013. Understanding User Behavior in Online Social Networks: A Survey. *Communications Magazine, IEEE* 51, 9 (2013), 144–150.

[17] Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. 2014. A Clockwork RNN. In *Proc. of ICML*.

[18] Srijan Kumar, Justin Cheng, Jure Leskovec, and V.S. Subrahmanian. 2017. An Army of Me: Sockpuppets in Online Discussion Communities. In *Proc. of WWW*.

[19] Libo Li, Frank Goethals, Bart Baesens, and Monique Snoeck. 2017. Predicting software revision outcomes on GitHub using structural holes theory. *Computer Networks* 114 (2017), 114–124.

[20] Fenglong Ma, Radha Chitta, Jing Zhou, Quanzeng You, Tong Sun, and Jing Gao. 2017. Dipole: Diagnosis Prediction in Healthcare via Attention-based Bidirectional Recurrent Neural Networks. In *Proc. of ACK KDD*.

[21] Jing Ma, Wei Gao, Prasenjit Mitra, Sejeong Kwon, Bernard J. Jansen, Kam-Fai Wong, and Meeyoung Cha. 2016. Detecting Rumors from Microblogs with Recurrent Neural Networks. In *Proc. of IJCAI*.

[22] Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 12, 2 (1947), 153–157.

[23] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. 2016. Phased LSTM: Accelerating Recurrent Network Training for Long or Event-based Sequences. In *Proc. of NIPS*.

[24] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, and et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[25] Liudmila Ostroumova Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *Proc. of NeurIPS*.

[26] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[27] Yoshihiko Suhara, Yinzhan Xu, and Alex 'Sandy' Pentland. 2017. DeepMood: Forecasting Depressed Mood Based on Self-Reported Histories via Recurrent Neural Networks. In *Proc. of WWW*.

[28] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of Social and Technical Factors for Evaluating Contribution in GitHub. In *Proc. of ICSE*.

[29] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and Productivity Outcomes Relating to Continuous Integration in GitHub. In *Proc. of ESEC/FSE*.

[30] Bimal Viswanath, Muhammad Ahmad Bashir, Mark Crovella, Saikat Guha, Krishna P. Gummadi, Balachander Krishnamurthy, and Alan Mislove. 2014. Towards Detecting Anomalous User Behavior in Online Social Networks. In *Proc. of USENIX Security*.

[31] Gang Wang, Tristan Konolige, Christo Wilson, Xiao Wang, Haitao Zheng, and Ben Y. Zhao. 2013. You are How You Click: Clickstream Analysis for Sybil Detection. In *Proc. of USENIX Security*.

[32] Bernard Lewis Welch. 1951. On the comparison of several mean values: an alternative approach. *Biometrika* 38, 3/4 (1951), 330–336.

[33] Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proc. of ICML*.

[34] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y. Zhao, and Yafei Dai. 2014. Uncovering Social Network Sybils in the Wild. *ACM Trans. Knowl. Discov. Data* 8, 1 (2014), 2:1–2:29.