SATAR: A Self-supervised Approach to Twitter Account Representation Learning and its Application in Bot Detection

# 3 PROBLEM DEFINITION

Let *𝑈* be a Twitter user, consisting of three aspects of user information: semantic *𝑇*, property *𝑃* and neighborhood *𝑁*. Let *𝑇* = {*𝑡𝑖* }*𝑖𝑀*=1 be a user’s semantic information of *𝑀* tweets. Each tweet *𝑡𝑖* =

{*𝑤*1*𝑖 ,* · · ·*,𝑤𝑄𝑖 𝑖* } contains *𝑄𝑖* words. Let *𝑃* = {*𝑝𝑖* }*𝑖𝑅*=1 be a user’s property information with a total of *𝑅* properties. Each property *𝑝𝑖* could be numerical such as follower count or categorical such as whether

the user is verified. Let *𝑁* = {*𝑁 𝑓 , 𝑁𝑡* }, where *𝑁 𝑓* = {*𝑁*1*𝑓 ,* · · ·*, 𝑁𝑢𝑓* } are *𝑢* followings of the user and *𝑁𝑡* = {*𝑁*1*𝑡,* · · ·*, 𝑁𝑣𝑡* } are *𝑣* followers. Similar to previous research [23, 40], we treat Twitter bot detection as a binary classification problem, where each user could either be human (*𝑦* = 0) or bot (*𝑦* = 1). Formally, we can define the Twitter bot detection task as follows:

# 4 SATAR METHODOLOGY

In this section, we present the details of the proposed Twitter user representation learning framework named as SATAR (Selfsupervised Approach to Twitter Account Representation learning).

## 4.1 Overview

Figure 1 illustrates the proposed framework SATAR. It consists of four major components: (1) a tweet-semantic sub-network, (2) a profile-property sub-network, (3) a following-follower sub-network and (4) a Co-Influence aggregator. Specifically, we use the Twitter API[[1]](#footnote-1) to obtain relevant data regarding a user’s semantic, property and neighborhood information. The tweet-semantic sub-network encodes a Twitter user’s textual information into *𝑟𝑠* with hierarchical RNNs of different depth accompanied by the attention mechanism. The profile-property sub-network encodes a Twitter user’s profile properties into *𝑟𝑝* with property data encoding and fully connected layers. The following-follower sub-network encodes a Twitter user’s neighborhood relationships into *𝑟𝑛* with neighborhood information extractor and fully connected layers. Finally, a non-linear Co-Influence aggregator takes the correlation between three aforementioned components into account, generating a representation vector that fully embodies the social status of a specific Twitter user. A softmax layer is then applied for user classification and enables model learning.

## 4.2 Tweet-Semantic Sub-Network

In this paper, we exploit user semantic information at two different levels, tweet-level and word-level, to capture the tweet content of users. Specifically, words in a user’s tweets could be fitted into two hierarchical structures. For tweet-level characterization, as defined in Section 3, *𝑤𝑖𝑗* denotes the *𝑖*-th word in the *𝑗*-th tweet of the user timeline, and *𝑡𝑗* represents the *𝑗*-th tweet of a specific user. We also concatenate temporally adjacent tweets: {*𝑤*1*,* · · ·*,𝑤𝐾* } = {*𝑤*11*,* · · ·*,𝑤𝑄*1 *,𝑤*12*,* · · ·*,𝑤𝑄𝑀𝑀* }, where the total word

1

count *𝐾* = Í*𝑖𝑀*=1 *𝑄𝑖*. Thus for word-level characterization, *𝑤𝑘* denotes the *𝑘*-th word in the user’s tweet history with temporally adjacent tweets concatenated to form a sequence. It is noteworthy that the underlying words are identical between tweet-level and word-level, but their annotations differ according to the user’s tweeting behaviors. To jointly leverage user tweet information on these two different levels, we propose tweet-level and word-level encoders of hierarchical RNNs to model tweet text sequences respectively and derive an overall semantic representation for Twitter users. The overall semantic representation for Twitter users are concatenated with results of tweet-level and word-level:

*𝑟𝑠* = *𝑐𝑜𝑛𝑐𝑎𝑡𝑒𝑛𝑎𝑡𝑖𝑜𝑛*(*𝑟𝑠𝑡* ;*𝑟𝑠𝑤*)*.* (1) where *𝑟𝑠𝑡* and *𝑟𝑠𝑤* are representations of tweets on tweet-level

and word-level.

Tweet-Level Encoder. The tweet-level encoder follows a bottomup approach. For the *𝑗*-th tweet of a specific user, we first embed words in it with an embedding layer:

*𝑥𝑖𝑗* = *𝑒𝑚𝑏*(*𝑤𝑖𝑗* )*,* 1 ⩽ *𝑖* ⩽ *𝑄𝑗,* 1 ⩽ *𝑗* ⩽ *𝑀,* (2) where *𝑄𝑗* is the length of the *𝑗*-th tweet, and we use Word2Vec [29] as the embedding layer *𝑒𝑚𝑏*(·). To encode the tweet, a bidirectional RNN processes the tweet in a forward pass and a backward pass. For the forward pass, a sequence of forward hidden states is generated for the *𝑗*-th tweet:

→−*𝑡* →−*𝑡* →−*𝑡* →−*𝑡*  *ℎ 𝑗* = *ℎ 𝑗,*1*, ℎ 𝑗,*2*,* · · ·*, ℎ 𝑗,𝑄𝑗 ,* (3)

where the hidden representation for each step is generated by

→−*𝑡* →−*𝑡 𝑗*

*ℎ 𝑗,𝑖* = *𝑅𝑁𝑁 ℎ 𝑗,𝑖*−1*,𝑥𝑖 .* (4)

Here we use LSTM [21] as *𝑅𝑁𝑁* (·), which is widely adopted to model long-term dependencies in a sequence. For the backward pass, a sequence of backward hidden states is generated similarly:

←−*𝑡* ←−*𝑡* ←−*𝑡* ←−*𝑡*  *ℎ 𝑗* = *ℎ 𝑗,*1*, ℎ 𝑗,*2*,* · · ·*, ℎ 𝑗,𝑄𝑗 .* (5)

We concatenate the forward and backward results to form a sequence of word representations in the *𝑗*-th tweet:

### 

*ℎ𝑡𝑗* = *ℎ𝑡𝑗,*1*,ℎ𝑡𝑗,*2*,* · · ·*,ℎ𝑡𝑗,𝑄𝑗 ,* (6)

→− ←−

where *ℎ𝑡𝑗,𝑖* = *ℎ 𝑡𝑗,𝑖*; *ℎ 𝑡𝑗,𝑖* . Since words in a tweet vary in their contribution to the tweet’s overall semantic meaning, the attention mechanism is adopted to aggregate word hidden representations into a tweet vector. Specifically,

*𝑒𝑥𝑝*(*𝑢𝑡* · *𝑣𝑡* )

*𝑗,𝑖*

*𝛼𝑡𝑗,𝑖* = *𝑡 𝑙 𝑡 ,* (7)

Í*𝑖*′ *𝑒𝑥𝑝*(*𝑢𝑗,𝑖*′ · *𝑣𝑙* )

where *𝑢𝑡𝑗,𝑖* = *𝑡𝑎𝑛ℎ*(*𝑊𝑙𝑡ℎ𝑡𝑗,𝑖* + *𝑏𝑙𝑡* ) transforms vectors for each word

and *𝑣𝑙𝑡* , *𝑊𝑙𝑡* and *𝑏𝑙𝑡* are learnable parameters. *𝛼𝑡𝑗,𝑖* represents the weight of the *𝑖*-th word in the *𝑗*-th tweet. Finally, the representation of the *𝑗*-th tweet can be obtained as follows:

*𝑡* ∑︁ *𝑡 𝑡*

*𝑣𝑗* = *𝛼𝑗,𝑖ℎ𝑗,𝑖.* (8)

*𝑖*

After deriving a vector for each tweet, the tweet-level encoder applies RNN similarly to tweet representations {*𝑣𝑡𝑗* }*𝑀𝑗*=1, generating a forward and a backward sequence. We concatenate the forward

and backward results to form a sequence of tweet representations:

*𝑡 𝑡 𝑡 𝑡*

*ℎ* = *ℎ*1*,ℎ*2*,* · · ·*,ℎ𝑀 ,* (9)

→− ←−

*𝑡 𝑡 𝑡*

where *ℎ𝑖* = *ℎ 𝑖* ; *ℎ 𝑖* . An attention layer is applied to model the influence each tweet has on the overall semantics of the user:

*𝑒𝑥𝑝*(*𝑢𝑡* · *𝑣𝑡* )

*𝑡 𝑖 ℎ*

*𝛼* = *,* (10)

*𝑖* Í ′ *𝑒𝑥𝑝*(*𝑢𝑡*′ · *𝑣𝑡* )

*𝑖 𝑖 ℎ*

where *𝑢𝑖𝑡* = *𝑡𝑎𝑛ℎ*(*𝑊ℎ𝑡ℎ𝑡𝑗* +*𝑏ℎ𝑡* ) transforms vectors for each tweet and

*𝑡 𝑡 𝑡 𝑡*

*𝑣ℎ*,*𝑊ℎ* and *𝑏ℎ* are learnable parameters. *𝛼𝑖* represents the weight of the*𝑖*-th tweet. Finally, the representation of a user’s tweet semantics from a tweet-oriented perspective can be obtained as follows:

*𝑡* ∑︁ *𝑡 𝑡*

*𝑟* = *𝛼𝑖 ℎ𝑖 .* (11)

*𝑠*

*𝑖*

Word-Level Encoder. The word-level encoder concatenates temporally adjacent tweets into a long sequence of words. For the *𝑖*-th word of the sequence, we first embed it with the embedding layer identical to the tweet-level encoder:

*𝑥𝑖* = *𝑒𝑚𝑏*(*𝑤𝑖*)*,* 1 ⩽ *𝑖* ⩽ *𝐾,* (12)

where *𝐾* is the total word count in the temporally concatenated tweets. A bidirectional RNN with attention is adopted to encode the concatenated sequence. For the forward pass, we have:

→−*𝑤* →−*𝑤* →−*𝑤* →−*𝑤*

*ℎ* = *ℎ* 1 *, ℎ* 2 *,* · · ·*, ℎ 𝐾 ,* (13)

→− →−

*𝑤 𝑤*

where *ℎ 𝑖* = *𝑅𝑁𝑁* ( *ℎ 𝑖*−1*,𝑥𝑖*) and LSTM is adopted for *𝑅𝑁𝑁* (·) regarding its particular length. For the backward pass, we have:

←−*𝑤* ←−*𝑤* ←−*𝑤* ←−*𝑤*

*ℎ* = *ℎ* 1 *, ℎ* 2 *,* · · ·*, ℎ 𝐾 ,* (14)

←− ←−

*𝑤 𝑤*

where *ℎ 𝑖* = *𝑅𝑁𝑁* ( *ℎ 𝑖*+1*,𝑥𝑖*). Then we concatenate the forward and backward results to form a sequence of word representations

in the user’s tweet history:

### 

*𝑤 𝑤 𝑤 𝑤*

*ℎ* = *ℎ*1 *,ℎ*2 *,* · · ·*,ℎ𝐾 ,* (15)

→− ←−

*𝑤 𝑤 𝑤*

where *ℎ𝑖* = *ℎ 𝑖* ; *ℎ 𝑖* . Then the attention mechanism is applied:

*𝑤 𝑤 𝑒𝑥𝑝*(*𝑢* · *𝑣* )

*𝑤 𝑖*

*𝛼𝑖* = Í ′ *𝑒𝑥𝑝*(*𝑢𝑤*′ · *𝑣𝑤*) *,* (16)

*𝑖 𝑖*

where *𝑢𝑖𝑤* = *𝑡𝑎𝑛ℎ*(*𝑊 𝑤ℎ𝑖𝑤* + *𝑏𝑤*), *𝑣𝑤*, *𝑊 𝑤* and *𝑏𝑤* are learnable parameters, *𝛼𝑖𝑤* represents the weight of the *𝑖*-th word in the concatenated sequence. Finally, the representation of a user’s tweet semantics from a word-oriented perspective is as follows:

*𝑤* ∑︁ *𝑤 𝑤*

*𝑟* = *𝛼𝑖 ℎ𝑖 .* (17)

*𝑠*

*𝑖*

## 4.3 Profile-Property Sub-Network

To avoid the undesirable bias incorporated in feature engineering, the profile-property sub-network utilizes profile properties that could be directly retrieved from the Twitter API. Different encoding strategies are adopted for different types of property data:

* There are 15 true-or-false property items in total. We use 1 for true and 0 for false. e.g. “profile uses background image”.
* There are 5 numerical property items in total. We apply z-score

normalization to numerical properties over the whole dataset. e.g. “favorites count”.

* There is one special property item: “location”. We divide locations geographically into different countries and apply one-hot encoding.

It is noteworthy that the follower count of a specific user would not be included in the property vector, which would be part of the self-supervised learning schema presented in Section 4.6.

The encoded property items are concatenated to form a raw property vector *𝑢𝑝*, which is then transformed to produce the Twitter user’s property representation *𝑟𝑝*:

*𝑟𝑝* = *𝑅𝑒𝐿𝑈* (*𝐹𝐶𝑝* (*𝑢𝑝*))*,* (18)

where *𝐹𝐶𝑝* (·) is a fully connected layer and *𝑅𝑒𝐿𝑈* (·) is a nonlinearity adopted as the activation function.

## 4.4 Following-Follower Sub-Network

For user followings, according to Twitter mechanism, their tweets will appear in the timeline and the following behaviors often demon-

*𝑓* strate interest in their tweet content. Thus we propose *𝑢𝑛* to model the following relationships:

*𝑓* 1 ∑︁

*𝑢𝑛* =  *𝑇𝐹* (*𝑢*)*𝑟𝑠* (*𝑢*)*,* (19)

Í*𝑢 𝑁 𝑓 𝑇𝐹* (*𝑢*)

∈ *𝑢*∈*𝑁 𝑓*

where *𝑁 𝑓* denotes the following set of a Twitter user,*𝑇𝐹* (*𝑢*) denotes the tweet frequency of user *𝑢* and *𝑟𝑠* (*𝑢*) is the semantic representation of user *𝑢* generated by the tweet-semantic sub-network. Tweet frequency *𝑇𝐹* is approximated by a user’s total tweet count divided by account active time, which is the time period between a user’s

# *𝑇𝐹* (*𝑢*)

registration and its last update. Note that Í*𝑢*′∈*𝑁𝑓 𝑇𝐹* (*𝑢*′) represents

*𝑓* the proportion that user *𝑢* appears in one’s timeline, thus *𝑢𝑛* serves as a weighted sum of followings’ semantics information according to their relative tweeting frequency.

For followers, as the average quality of followers of an account defines its social status and the quality could be evaluated by its properties, we propose to model the follower relationships as follows:

*𝑡* 1 ∑︁

*𝑢𝑛* = *𝑡 𝑟𝑝* (*𝑢*)*,* (20)

|*𝑁* |

*𝑢*∈*𝑁𝑡*

where *𝑁𝑡* denotes the follower set of a Twitter user, | · | denotes the cardinality of a set and *𝑟𝑝* (*𝑢*) is the property representation of user *𝑢* generated by the profile-property sub-network.

The following-follower sub-network then produces a raw hidden

*𝑓*

vector for neighborhood information *𝑢𝑛* = *𝑐𝑜𝑛𝑐𝑎𝑡𝑒𝑛𝑎𝑡𝑖𝑜𝑛*(*𝑢𝑛* ;*𝑢𝑛𝑡* ). The intermediate vector is then transformed to produce the Twitter user’s neighborhood representation *𝑟𝑛*:

*𝑟𝑛* = *𝑅𝑒𝐿𝑈* (*𝐹𝐶𝑛*(*𝑢𝑛*))*,* (21)

where *𝐹𝐶𝑛*(·) is a fully connected layer and *𝑅𝑒𝐿𝑈* (·) is the adopted activation function.

## 4.5 Co-Influence Aggregator

So far, we have obtained the representation vectors regarding three and all three aspects of a Twitter user, namely *𝑟𝑠*, *𝑟𝑝* and *𝑟𝑛* for tweet semantics, user property and follow relationships. A good bot detector should be comprehensive and robust to tamper. In other words, independently considering each aspect of user information would inevitably jeopardize the robustness of the bot detector. Co-attention has been a successful mechanism at handling correlation between two sequences, but it is not designed for mutual influence between multiple representation vectors. Thus we propose a Co-Influence aggregator to take the mutual correlation between tweet semantics, user property and follow relationships into consideration.

Firstly, the affinity index between a pair of aspects is derived:

*𝐹𝑠𝑝* = *𝑡𝑎𝑛ℎ*(*𝑟𝑠𝑇𝑊𝑠𝑝𝑟𝑝*)*,*

*𝐹𝑝𝑛* = *𝑡𝑎𝑛ℎ*(*𝑟𝑝𝑇𝑊𝑝𝑛𝑟𝑛*)*,* (22)

*𝐹𝑛𝑠* = *𝑡𝑎𝑛ℎ*(*𝑟𝑛𝑇𝑊𝑛𝑠𝑟𝑠*)*,*

where *𝑊𝑠𝑝*, *𝑊𝑝𝑛* and *𝑊𝑛𝑠* are learnable parameters of the aggregator. A hidden representation for each aspect which incorporates relevant information from the other two aspects are derived:

*ℎ𝑠* = *𝑡𝑎𝑛ℎ*(*𝑊𝑠𝑟𝑠* + *𝐹𝑠𝑝* (*𝑊𝑝𝑟𝑝*) + *𝐹𝑛𝑠* (*𝑊𝑛𝑟𝑛*))*,*

*ℎ𝑝* = *𝑡𝑎𝑛ℎ*(*𝑊𝑝𝑟𝑝* + *𝐹𝑠𝑝* (*𝑊𝑠𝑟𝑠*) + *𝐹𝑝𝑛*(*𝑊𝑛𝑟𝑛*))*,* (23)

*ℎ𝑛* = *𝑡𝑎𝑛ℎ*(*𝑊𝑛𝑟𝑛* + *𝐹𝑛𝑠* (*𝑊𝑠𝑟𝑠*) + *𝐹𝑝𝑛*(*𝑊𝑝𝑟𝑝*))*,*

where *𝑊𝑠*, *𝑊𝑝* and *𝑊𝑛* are learnable parameters of the aggregator. Finally, the proposed framework SATAR produces the Twitter user representation *𝑟* as follows:

*𝑟* = *𝑡𝑎𝑛ℎ*(*𝑊𝑉* · *𝑐𝑜𝑛𝑐𝑎𝑡𝑒𝑛𝑎𝑡𝑖𝑜𝑛*(*ℎ𝑠*;*ℎ𝑝*;*ℎ𝑛*))*,* (24) where *𝑊𝑉* is a learnable parameter of the aggregator.

## 4.6 Self-Supervised Learning and Optimization

Twitter user representation learning attempts to model a specific user with a distributed representation. We adopt follower count as the self-supervised signal for SATAR training. Specifically, a user’s follower count is separated into several categories based on its numerical scale and the overall follower count distribution. We train the representation learning framework SATAR to classify each user into such categories, obtaining user representation in the process. We believe that follower count would be an ideal self-supervised training signal due to the following reasons:

* Self-supervised training with follower count is task-agnostic. Whether it is bot detection, content recommendation or online campaign modeling, follower count relates to all tasks on social media without being specific to any of them.
* Follower count is most representative of a Twitter user. There is no better choice to describe a Twitter user more efficiently and accurately, especially when follower count also involves the evaluation of other users.
* Follower count is more robust to large-scale tamper. Although it is possible to purchase fake followers, according to Cresci *et al.* [7]’s investigation, an increase of 1,000 followers often costs from 13 to 19 U.S. dollars. As a result, it is costly to significantly alter the magnitude of a user’s follower count, let alone launch a campaign with many active bots.

Specifically, assuming that a user could be categorized into *𝐷* classes based on its follower count, a softmax layer is applied to the representation of the user *𝑟*:

*𝑦*ˆ = *𝑠𝑜𝑓 𝑡𝑚𝑎𝑥*(*𝑊𝑓 𝑟* + *𝑏𝑓* )*,* (25)

where *𝑦*ˆ = [*𝑦*ˆ1*,𝑦*ˆ2*,* · · ·*,𝑦*ˆ*𝐷*] is the predicted probability vector for each class,*𝑊𝑓* and *𝑏𝑓* are learnable parameters. *𝑦* = [*𝑦*1*,𝑦*2*,* · · ·*,𝑦𝐷*] denotes the self-supervised ground-truth for such classification in one-hot encoding. We minimize the cross-entropy loss function as follows:

### ∑︁

*𝐿*(*𝜃*) = − *𝑦𝑖𝑙𝑜𝑔*(*𝑦*ˆ*𝑖*)*,* (26)

1⩽*𝑖*⩽*𝐷*

where *𝜃* denotes the parameters in the proposed framework SATAR. Algorithm 1 presents the overall training schema of our proposed Twitter account representation learning framework SATAR.

1. <https://developer.twitter.com/en/products/twitter-api/early-access> [↑](#footnote-ref-1)