Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

A Social Interaction Activity based Time-Varying User Vectorization Method for Online Social Networks

Tianyi Hao and **Longbo Huang**
Institute for Interdisciplinary Information Sciences, Tsinghua University
haoty14@mails.tsinghua.edu.cn, longbohuang@tsinghua.edu.cn

Abstract

In this paper, we consider the problem of user modeling in online social networks, and propose a social interaction activity based user vectorization framework, called the time-varying user vectorization (`Tuv`), to infer and make use of important user features. `Tuv` is designed based on a novel combination of `word2vec`, negative sampling and a smoothing technique for model training. It jointly handles multi-format user data and computes user representing vectors, by taking into consideration user feature variation, self-similarity and pairwise interactions among users. The framework enables us to extract hidden user properties and to produce user vectors. We conduct extensive experiments based on a real-world dataset, which show that `Tuv` significantly outperforms several state-of-the-art user vectorization methods.

1 Introduction

Due to the rapid recent development of Internet technologies, online social networks are becoming more and more popular with users. Important information associated with users can be inferred from online social networks, and various user-centric applications have been developed based on information from online social networks, including recommendations and advertisements. As a result, how to efficiently retrieve useful user information from online social networks has become one central question for both industry and academia.

Among the important topics about online social networks, in this paper, we focus on *user modeling* from an information retrieval perspective. User modeling has been receiving increasing attention in recent years, which aims at developing a deeper understanding about users, so that better analysis and prediction about online user behavior can be achieved. There have been many prior works proposing different approaches for user modeling, for instance, [Amir *et al.*, 2016; Tang *et al.*, 2015]. Different from these methods which are based on user connectivity, we instead focus on utilizing social interaction activities for characterizing online users.

Social interaction activities are widely available and valuable information. People on social media publish messages and photos describing events in life, exchanging ideas and sharing news. From these activities, one can observe many interesting properties of users. Compared to other information, e.g., connectivity, social interaction activities provide more detailed descriptions about users, and preserve time-dependent features. Thus, mining interaction activities should provide us with more accurate modeling for users.

However, efficiently utilizing social interaction data is challenging. First, interaction activities often contain various forms of data, e.g., connectivity and text, and one has to provide a unifying approach to properly fuse different information. Second, the proposed method should be computationally efficient and generate consistent results. Third, the model should respect the self-similar nature of user activities over time. This imposes additional constraints on the modeling solution and further increases the design complexity.

To tackle these challenges, in this work, we propose a novel framework for user modeling based on social interaction activities, called the *Time-varying User Vectorization* (`Tuv`). It is based on a novel combination of `word2vec` [Mikolov *et al.*, 2013a], negative sampling and a smoothing technique for model training. It jointly handles multi-format user data and computes user vectors by taking into consideration user feature variation, self-similarity and pairwise interactions among users. `Tuv` possesses unique advantages due to its vectorization nature. (i) Vectorization facilitates various analytical operations on users, e.g., similarity computation and clustering. (ii) Time-varying vectors better track evolving user properties, and enable a better user behavior prediction. These features can not only help develop a better understanding about users, but also be used to improve performance of Internet applications including recommendations and advertisements.

The method of user vectorization is not new, e.g., [Perozzi *et al.*, 2014; Tang *et al.*, 2015; Fu *et al.*, 2009]. Our work distinguishes itself from these works in two ways. First, while most works carry out vectorization based on social graphs, we instead focus on utilizing social interaction records, which gives us stronger information about user interactions. We also benefit from text data in interaction records, which contain important information about users not contained in connectivity data. Second, while most works focus on user modeling that is static in time, we handle the problem with **time-varying** user vectorization, which focuses on modeling user properties over different times. This way, we obtain a deeper understanding about how user properties change in time.

Our contributions are summarized as follows.

- We propose to utilize user interaction activities for time-varying user vectorization in online social networks. Different from most existing approaches, which often rely on static user information, e.g., connectivity graph, our idea enables a more detailed user characterization that keeps track of the dynamic patterns of user features.
- We design a novel framework based on user interaction activities called the *Time-varying User Vectorization* (`Tuv`). `Tuv` jointly handles multi-format data, including user labels, text messages and time information. Moreover, it is able to take into consideration self-similarity of user properties and time-varying relationships among users. Although `Tuv` is developed for user modeling, it also applies to other tasks with multi-sourced data processing and feature identification.
- We conduct extensive experiments based on a real-world dataset to compare `Tuv` with several state-of-the-art methods for user vectorization in three applications, i.e., user attention evaluation, time-varying user similarity estimation, and user occupation prediction. Our results show that `Tuv` is able to accurately capture hidden user properties and outperform existing methods.

2 Related Work

There have been prior works on social network modeling and user vectorization. [Ren *et al.*, 2008] uses SVD for network embedding to detect community structure. [Zhao *et al.*, 2010; 2011] provides a technique to embed social graph nodes into coordinate spaces to preserve node distances within graphs. DeepWalk [Perozzi *et al.*, 2014] generates random walk paths to vectorize social network nodes with the SkipGram model [Mikolov *et al.*, 2013a]. LINE [Tang *et al.*, 2015] defines proximities within a network, and utilizes these metrics for node vectorization. [Tu *et al.*, 2016; Grover and Leskovec, 2016; Liu *et al.*, 2016] are also node vectorization models. Different from our work which uses social interaction activities, all aforementioned works rely on social graphs.

Besides results on social graphs, there are also works utilizing other user information. [Zhang *et al.*, 2017] combines social graphs and user profile information. There are also vectorization models utilizing text information [Amir *et al.*, 2016; Yang *et al.*, 2016]. [Nallapati *et al.*, 2008; Cho *et al.*, 2016] combine network structures with text models, but the texts are associated to local network nodes instead of interaction among nodes. There have also been works on dynamic networks [Fu *et al.*, 2009; Rossi *et al.*, 2013], but most of them focus on network structures instead of social interaction records with multiple forms of data.

To jointly handle text information, we use the `word2vec` framework from [Mikolov *et al.*, 2013a; 2013b], which is an effective language model for word vectorization.

3 Problem Statement

In this section, we describe the setting of social interaction activities, and formally define the problem of time-varying user vectorization.

3.1 Representing Social Network Data

We describe how we represent social network data, including users and activities, in order to carry out our investigation.

(i) **User**: We denote the set of users in consideration by $U = \{u_1, u_2, \cdots, u_n\}$, where n is the total number of users.

(ii) **Activity**: We focus on two social interactions, publishing and forwarding. (a) *Publishing* is the activity that an author publishes a new message. We denote it as a tuple $A = \langle u, t, m \rangle$, where u is the author, t is the timestamp, and m is the text information. (b) *Forwarding* is the activity that a user reposts a message from another user. We denote it as a tuple $A = \langle u, t, m, u_a, t_a, u_f, t_f \rangle$. The notations u, t and m are similar to a publishing activity. u_a and t_a denote the author and timestamp of the original message. u_f is the user from whom the message is forwarded, and t_f is u_f's forwarding time. If the message is forwarded from the original author, we set u_f to be a null user $u_\varnothing$, and t_f to be a null time $t_\varnothing$. Finally, we denote the set of all activities by $\mathcal{A}$.

3.2 Time-varying User Vectorization

Our objective is to find, for each user $u \in U$, a vector representation, which can capture important characteristics of the user and can be used to analyze his relationship with others.

Specifically, we define the active time of a user to be the time period from his first activity to the last. Denote the start time of user u's activities to be T_u, and the end time to be $\tilde{T}_u$. Since user behavior does not change very frequently, in order to simplify analysis and computation, we divide time into discrete timeslots of size Δt. For each timeslot t, we define a vector $\vec{v}_u(t)$ for user u at time t. Our goal is to find vectors $\{\vec{v}_u(t)\}_{u,t}$ that satisfy the following requirements:

- The resulting vectors can reflect certain personal properties of users, such as occupations. Users with similar properties shall have similar vector representations.
- Relations among different user vectors can reflect relationship (e.g., closeness or similarity) among users.
- The vectors of a user should also not vary too much during consecutive time slots.

Prior works on user vectorization, e.g., [Perozzi *et al.*, 2014; Tang *et al.*, 2015; Ren *et al.*, 2008] often assume that each user vector $\vec{v}_u$ is a constant vector obtained from structure information of the social networks, e.g., connectivity. We instead focus on the problem of **time-varying** user vectorization based on user interaction activities, and aim to find a vector function $\vec{v}_u(t)$ whose value characterizes the time-varying nature of user properties. This problem is challenging, because social interaction is a complicated system with various data types, such as pairwise interactions, text messages and timestamps, and a unified framework needs to be designed in order to fuse multi-format data. In our work, we propose a novel framework which takes multi-format data into consideration, and captures the time-varying properties accurately.

4 Method for Vectorization

In this section, we present our time-varying user vectorization (`Tuv`) method. We start by describing the text data vectorization. Then, we define our objective function and present the algorithm for computing user vectors.

4.1 Vectorization for Text Messages

As we have described, there is a text message m in each activity record, which contains rich valuable information. Since text messages are often not in numerical form and cannot be directly used in vectorization algorithms, we first preprocess them and convert them into vectors.

For a text message m, we use the sum of `word2vec` vectors [Mikolov *et al.*, 2013a; 2013b] of the words in it to construct its vector representation $\vec{v}(m)$. This method has also been adopted in prior works, e.g., [Yu *et al.*, 2015; Arora *et al.*, 2015], and can often produce accurate results for short messages [Adi *et al.*, 2017]. Compared to more complicated models, e.g., [Le and Mikolov, 2014], it is computationally efficient for handling large-scale text messages.

4.2 Objective Function and Optimization

In this section, we explain the design and meaning of user vectors, and present our objective as well as how to solve the `Tuv` problem. Different from existing works based on graphs, e.g., [Perozzi *et al.*, 2014; Tang *et al.*, 2015], we focus on user interaction activities, since doing so enables the characterization of users' time-varying properties. This is a challenging task as it requires fusing multiple information types including user connectivity, text messages and timestamps.

Recall that we have two types of interactions: publishing $A = \langle u, t, m\rangle$ and forwarding $A = \langle u, t, m, u_a, t_a, u_f, t_f\rangle$. In order to jointly handle them, we unify both representations by changing $\langle u, t, m\rangle$ to $\langle u, t, m, u, t, u_\varnothing, t_\varnothing\rangle$, where $u_\varnothing$ is the null user and $t_\varnothing$ is the null time. Below, we regard every activity to be in this form.

As discussed above, for each user u, our goal is to find a vector $\vec{v}_u(t)$ for the user at each time t, which can reflect his properties. Below, one will see that the pattern of messages that a user usually publishes or forwards has a strong relationship with the properties of the user. Thus, we will make full use of the text messages when computing our user vectors.

To this end, we define the "likelihood" that a user u will publish or forward a message m at time t to be a function:

$$L_m(u, m, t) = S(\vec{v}_u(t) \cdot \vec{v}(m)).$$

Here $\vec{v}_u(t)$ is the vector for user u at time t, $\vec{v}(m)$ is the vector for text message m, and $S(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function. We use the dot product $\vec{v}_u(t) \cdot \vec{v}(m)$ because we want the user vector to be closer to the text message vectors they often publish or forward. The use of $S(x)$ ensures that the value of $L_m(u, m, t)$ is in the range $(0, 1)$. With this definition, $L_m(u, m, t)$ has a positive correlation with the probability that user u will publish or forward message m at time t, so that $\vec{v}_u(t)$ can reflect user properties related to text messages.

In addition to the user-message relationship, pairwise user relationship is also important. To utilize pairwise user relationship to obtain a more integrated model, we introduce two more vectors $\vec{r}_u(t)$ and $\vec{s}_u(t)$ for each user u at time t to describe the pairwise relationship among users. Based on these vectors, for user u in activity A, we define the following likelihoods to describe the relationship between pairs of users.

- Original author: The likelihood that user u at time t will forward a message authored by u_a at time t_a is defined as a function:

$$L_a(u, u_a, t, t_a) = S(\vec{v}_u(t) \cdot \vec{r}_{u_a}(t_a))$$

- Last forwarder: The likelihood that user u at time t will forward a message last forwarded by u_f at time t_f is:

$$L_f(u, u_f, t, t_f) = S(\vec{v}_u(t) \cdot \vec{s}_{u_f}(t_f)).$$

Intuitively, L_a denotes the likelihood that a user forward a particular user's original post, and L_f denotes the likelihood that a user forwards a friend's post. The reason to use $\vec{r}_u(t)$ and $\vec{s}_u(t)$ instead of $\vec{v}_u(t)$ is to allow the likelihood to be asymmetric. For a publishing activity, we similarly define the likelihoods as we have extended it to a forwarding one.

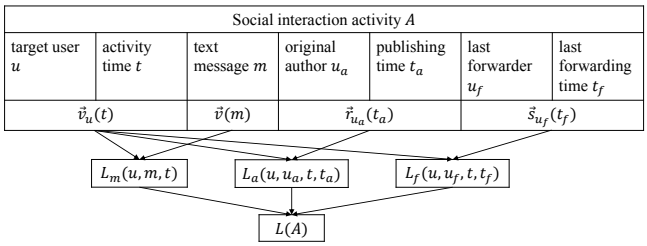

Figure 1: The procedure for generating the likelihood $L(A)$ from an activity record A. At first, for activity A, compute the partial likelihoods $L_m(u, m, t)$, $L_a(u, u_a, t, t_a)$ and $L_f(u, u_f, t, t_f)$. Then, multiply all above likelihoods to get the final output $L(A)$.

Based on the above definitions, we define the overall likelihood for user u to perform an activity A to be their product:

$$\begin{aligned} L(A) &= L_m(u, m, t) \cdot L_a(u, u_a, t, t_a) \cdot L_f(u, u_f, t, t_f) \\ &= S(\vec{v}_u(t) \cdot \vec{v}(m)) \cdot S(\vec{v}_u(t) \cdot \vec{r}_{u_a}(t_a)) \cdot S(\vec{v}_u(t) \cdot \vec{s}_{u_f}(t_f)) \end{aligned}$$

Here we remove the $S(\vec{v}_u(t) \cdot \vec{s}_{u_f}(t_f))$ term if $u_f = u_\varnothing$ and $t_f = t_\varnothing$. Figure 1 provides a graphical demonstration of this procedure for generating the likelihood function $L(A)$.

It remains to find appropriate vectors $\vec{v}_u(t)$, $\vec{r}_u(t)$ and $\vec{s}_u(t)$, so that the likelihood values best fit our dataset. To do so, we adopt the technique of negative sampling proposed in [Mikolov *et al.*, 2013b], which works as follows. For an activity $A = \langle u, t, m, u_a, t_a, u_f, t_f\rangle$, in addition to user u, we randomly choose a negative sample $\tilde{u} \neq u$, following the empirical distribution of users in the activity set $\mathcal{A}$. With the negative sample $\tilde{u}$, we define the negative likelihood $\tilde{L}(\tilde{A})$:

$$\begin{aligned} \tilde{L}(\tilde{A}) = (1 - S(\vec{v}_{\tilde{u}}(t) \cdot \vec{v}(m))) \cdot (1 - S(\vec{v}_{\tilde{u}}(t) \cdot \vec{r}_{u_a}(t_a))) \\ \cdot (1 - S(\vec{v}_{\tilde{u}}(t) \cdot \vec{s}_{u_f}(t_f))) \end{aligned}$$

where $\tilde{A}$ is attained by replacing user u with $\tilde{u}$ in A. Similarly we remove the $(1 - S(\vec{v}_{\tilde{u}}(t) \cdot \vec{s}_{u_f}(t_f)))$ term if $u_f = u_\varnothing$ and $t_f = t_\varnothing$. Then, given all activities $A \in \mathcal{A}$, our target likelihood function is given by:

$$L(\mathcal{A}) = \prod_{A \in \mathcal{A}} \left(L(A) \cdot \tilde{L}(\tilde{A}) \right)$$

Next, we want to also take into account the time-varying and self-similar property of users in the overall objective function. To do so, we assume that for each user u, the vector $\vec{v}_u(t + \Delta t)$ follows a Gaussian distribution, with its mean

Algorithm 1 Building the task list

Input: The set of user activities, $\mathcal{A}$, the set of all users, U
Output: The task list $\mathcal{T}$

Initialise $\mathcal{T}$ with empty list
for $A \in \mathcal{A}$ **do**
    Generate negative sample $\tilde{A}$, and add $\langle A, \tilde{A}\rangle$ to $\mathcal{T}$
for $u \in U$ **do**
    for $t \in \{T_u, T_u + \Delta t, \cdots, \tilde{T}_u - \Delta t\}$ **do**
        Add $\langle u, t, t + \Delta t\rangle$ to $\mathcal{T}$
return $\mathcal{T}$

being the vector $\vec{v}_u(t)$ in the last time slot, and the variance being σ^2. Then the probability distribution is given by:

$$f(\vec{v}_u(t+\Delta t) \mid \vec{v}_u(t)) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{|\vec{v}_u(t+\Delta t) - \vec{v}_u(t)|^2}{2\sigma^2})$$

This ensures that for the same user, his vectors in adjacent time slots are close to each other. The benefit of the Gaussian distribution is that we can obtain a quadratic term after taking a logarithm, making it suitable for a regularizer. Random processes with similar distributions have also been used in other models [Coulom, 2008; Wilson *et al.*, 2016].

Taking all above factors into account, our overall objective function is given by:

$$\mathcal{L}(\vec{v}, \vec{r}, \vec{s}) = \prod_{A\in\mathcal{A}} \Big(L(A) \cdot \tilde{L}(\tilde{A})\Big) \cdot \prod_{u\in U} \prod_{t=T_u}^{\tilde{T}_u - \Delta t} \exp(-\frac{|\vec{v}_u(t+\Delta t) - \vec{v}_u(t)|^2}{2\sigma^2})$$

Our goal is to find appropriate vectors $\vec{v}_u(t)$, $\vec{r}_u(t)$ and $\vec{s}_u(t)$, to maximize $\mathcal{L}(\vec{v}, \vec{r}, \vec{s})$. Taking a logarithm of it, we obtain

$$\log \mathcal{L}(\vec{v}, \vec{r}, \vec{s}) = \sum_{A\in\mathcal{A}} \Big(\log L(A) + \log \tilde{L}(\tilde{A})\Big) - \sum_{u\in U} \sum_{t=T_u}^{\tilde{T}_u - \Delta t} \frac{|\vec{v}_u(t+\Delta t) - \vec{v}_u(t)|^2}{2\sigma^2}$$

Similar to the optimization method in `word2vec` [Mikolov *et al.*, 2013a], we use stochastic gradient descent (SGD) [Bottou, 1991] to carry out the optimization. The main reason for this choice is that the global gradient of the objective function takes a long time to compute, especially for tens of millions of activity records. It is more practical to use the SGD algorithm to compute the gradient in an incremental way.

Our framework includes two steps. In the first step, we build a task list to contain data for our training. For each interaction record, we generate its negative sample, and add that to the task list. This is shown in Algorithm 1. In the second step, for each task in the list, we use SGD to update the vectors. We repeat this process for several iterations, decreasing the learning rate after each iteration. The detailed algorithm is shown in Algorithms 1 and 2 (the parameters $i_{\max}$, η_0, $\eta_{\min}$, λ will be specified later). The output vectors $\vec{v}_u(t)$, $\vec{r}_u(t)$ and $\vec{s}_u(t)$ are the desired results.

Algorithm 2 Optimization of the object function

Input: The task list, $\mathcal{T}$
Output: The vectors, $\vec{v}_u(t), \vec{r}_u(t), \vec{s}_u(t)$

1: Initialize $\vec{r}_u(t), \vec{s}_u(t)$ for all u, t with random vectors
2: $\eta = \eta_0$, $\vec{v}_u(t) = \vec{0}$ for all u, t
3: **for** each iteration $i \le i_{\max}$ **do**
4: Sort $\mathcal{T}$ with a uniform random order
5: **for** each task in $\mathcal{T}$ **do**
6: **if** the task is $\langle A, \tilde{A}\rangle$ **then**
7: $\vec{e} = \vec{0}$
8: **for** $a \in \{A, \tilde{A}\}$ **do**
 ▷ $a = \langle id, u, t, m, id_a, u_a, t_a, u_f, t_f\rangle$
9: $g_e = \eta(1(a = A) - S(\vec{v}_u(t) \cdot \vec{r}_{u_a}(t_a)))$
10: $\vec{e} = \vec{e} + g_e \vec{v}_u(t)$
11: $\vec{v}_u(t) = \vec{v}_u(t) + g_e \vec{r}_{u_a}(t_a)$
12: $\vec{r}_{u_a}(t_a) = \vec{r}_{u_a}(t_a) + \vec{e}$
13: Update $\vec{v}_u(t)$ and $\vec{s}_{u_f}(t_f)$ similar to line 7-12
14: **for** $a \in \{A, \tilde{A}\}$ **do**
15: $g_m = \eta(1(a = A) - S(\vec{v}_u(t) \cdot \vec{v}(m)))$
16: $\vec{v}_u(t) = \vec{v}_u(t) + g_m \vec{v}(m)$
17: **else if** the task is $\langle u, t, t + \Delta t\rangle$ **then**
18: $(\vec{v}_{\mathrm{m}}, \vec{v}_{\mathrm{d}}) = \frac{1}{2}(\vec{v}_u(t + \Delta t) \pm \vec{v}_u(t))$
19: $(\vec{v}_u(t + \Delta t), \vec{v}_u(t)) = \vec{v}_{\mathrm{m}} \pm e^{-\frac{2\eta}{\sigma^2}} \vec{v}_{\mathrm{d}}$
20: $\eta = \max(\lambda\eta, \eta_{\min})$
21: **return** $\vec{v}_u(t), \vec{r}_u(t), \vec{s}_u(t)$

5 Experiments and Applications

In this section, we describe experimental results of our `Tuv` model, obtained based on real-world social network data. We present three applications of our method and show that `Tuv` outperforms existing methods.

5.1 Data and Settings

We collect data from Sina Weibo (`http://weibo.com/`), a Chinese social media with over 100 million active users every day. We first identify and fix the user set, and then crawl their interaction records with Python scripts. We collected data for $7,370$ verified users (with IDs being verified by Sina Weibo) as our user set, with $252,560$ directed following edges among them. For interaction activities, there are $17,387,066$ records in total, including $9,829,429$ publishing records and $7,557,637$ forwarding records. The time duration is from August 2009 to June 2017.

After data collection, we begin to implement our framework. As described in Section 4.1, we compute the vectors of all the text messages in advance. Since most messages in Sina Weibo are in Chinese, we first use the THULAC tool [Sun *et al.*, 2016] to split the messages into words. After that, we compute the vector $\vec{v}(m)$ for each message m following Section 4.1. After obtaining the sentence vectors, following the algorithms in Section 4.2, we compute the vector representations $\vec{v}_u(t)$ and user relationship vectors $\vec{r}_u(t)$ and $\vec{s}_u(t)$. The detailed parameters for Δt being one year long are 100 dimensions for vectors, $i_{\max} = 30$, $\eta_0 = 0.1$, $\eta_{\min} = 1 \times 10^{-5}$, $\lambda = 0.518$, $\sigma^2 = 0.09$. For Δt being one season long, we re-

place them with $\lambda = 0.527$, $\sigma^2 = 0.00225$. Based on our experiments, the performance of our model is not very sensitive to the parameters, and the performance stays similar within a reasonable range of values. These values are chosen in this range to optimize the performance of the evaluation.

In the following, we present three applications of the resulting user vectors computed above, and compare them with other results obtained using other methods. We will see that Tuv achieves better performance.

5.2 User Attention Evaluation

Our first application is user attention evaluation, which is used to describe the level at which a user pays attention to another user. In an online social network, when user u_1 pays attention to user u_2, u_1 has a larger probability of reading and forwarding u_2's messages. User attention evaluation is an important problem and has a lot of applications including friend recommendation and message ranking.

To evaluate user attention, we conduct experiments on predicting whether a user will forward a message from another user at certain time. We use 80% of data to train the Tuv model and compute pairwise similarities for all users. Then, from the remaining 20% of data, we randomly choose 8000 records to generate 4000 positive samples and 4000 negative ones. Then we use 4000 samples among them to train an SVM [Cortes and Vapnik, 1995] (with input from Tuv model and output to be $+/-$ labels generated) and use 4000 samples to test. We then use the SVM accuracy to evaluate our model.

The positive and negative samples are generated in this way. For each record $A = \langle u, t, m, u_a, t_a, u_f, t_f \rangle$, we generate a positive sample $(+1, (u, t, u_a, t_a))$, meaning that the message is forwarded by u. Then, we randomly select another user $\tilde{u} \neq u$ to generate a negative sample $(-1, (\tilde{u}, t, u_a, t_a))$, meaning that the message is not forwarded by $\tilde{u}$.

We use the LIBSVM library [Chang and Lin, 2011] to do the classification. Each input vector contains $I_1 = \{\vec{v}_u(t), \vec{r}_u(t), \vec{s}_u(t)\}$, $I_2 = \{\vec{v}_{u_a}(t_a), \vec{r}_{u_a}(t_a), \vec{s}_{u_a}(t_a)\}$, together with the pairwise cosine similarities between vectors from these two sets $I_3 = \{\cos\langle \vec{v}_1, \vec{v}_2 \rangle | \vec{v}_1 \in I_1, \vec{v}_2 \in I_2\}$.

For comparison, we select several existing user vectorization methods. The input vectors are also user embedding vectors and their cosine similarities. The baselines include:

- DeepWalk [Perozzi *et al.*, 2014]: A method generating node vector representation from a social graph with random walks.
- LINE [Tang *et al.*, 2015]: A large-scale information network embedding scheme based on social graph.
- Weighted LINE with user activities (LINE-w): Instead of the social graph, use the social interaction records as the input to the LINE framework, and the edge weights are social interaction frequencies.
- LINE-w-time: Regard each user at each time slot as a node, and apply LINE-w.

We use accuracy, precision and recall as the metrics, and the results are shown in Table 1. We see that social interaction based methods outperform graph structure based methods. Among social interaction methods, the accuracy of Tuv-year outperforms than all other methods. For precision and recall, there is only one exception that static LINE-w has a better precision, but it has a much lower recall. The results demonstrate that Tuv achieves a good performance in user attention evaluation.

Method	attention (%)			similarities	
	acc.	pre.	rec.	RMSE	R^2
DeepWalk	72.6	78.2	62.7	0.0591	0.341
LINE	74.0	78.2	70.6	0.0567	0.393
LINE-w	83.4	**87.5**	78.0	**0.0494**	**0.539**
LINE-w-season	77.0	75.4	80.1	0.0649	0.206
LINE-w-year	82.8	81.9	84.4	0.0573	0.381
Tuv-season	82.3	80.8	84.7	0.0588	0.349
Tuv-year	**85.0**	82.5	**88.9**	0.0563	0.402

Table 1: Prediction result of user attention and user similarities.

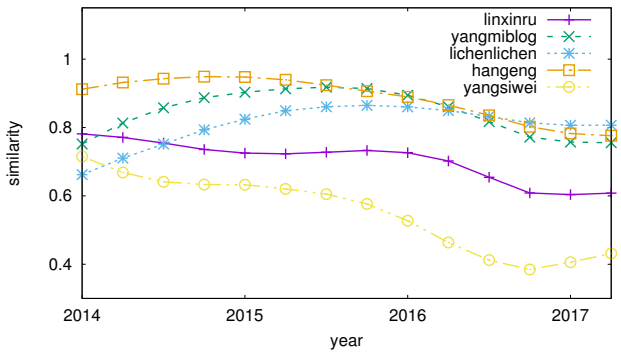

Figure 2: Similarities with fbb0916 for five users over seasons.

5.3 Time-varying User Similarities

Our second application is computing similarities between users. To carry out our evaluation, we use the cosine similarity of user vectors to denote the similarity between users. For u_1 and u_2 at time t, the similarity level is computed as:

$$\text{sim}(u_1, u_2, t) = \cos\langle \vec{v}_{u_1}(t), \vec{v}_{u_2}(t) \rangle.$$

Note that this is a feature not possessed by other methods without a time component, e.g., connectivity based similarity [Perozzi *et al.*, 2014; Tang *et al.*, 2015; Liu *et al.*, 2016].

With the user vectors we computed, for each user u at time t, we identify its closest neighbors with the largest similarity values. According to our observation, these neighbors are usually close friends, users who have deep working relations with the user, or users who have similar properties.

Take the blog ID fbb0916 (http://weibo.com/fbb0916) for example. We select five neighbors and plot the similarity values among them from 2014 to 2017 by seasons in Figure 2. It shows that our vectorization for users can efficiently capture time-varying user properties, including their relationship with other users in online social networks, and the pattern changes over time.

In order to evaluate the accuracy of our user similarities, we compute the message-based similarity among users as the ground truth. Specifically, it is defined as the Jaccard similar-

ity [Levandowsky and Winter, 1971], i.e.,

$$\mathrm{msim}(u_1, u_2, t) = \frac{|S(u_1, t) \cap S(u_2, t)|}{|S(u_1, t) \cup S(u_2, t)|},$$

where $S(u_1, t)$ and $S(u_2, t)$ is the set of messages that u_1 and u_2 published or forwarded at time t. We use 80% of all messages to train our Tuv model, and the remaining 20% to compute the Jaccard similarities. Then, we use linear regression to evaluate the correlation between the Tuv similarities $\mathrm{sim}(u_1, u_2, t)$ and the ground-truth $\mathrm{msim}(u_1, u_2, t)$, since a stronger correlation indicates a better accuracy of our Tuv method. We use $4,000$ pairs of users to train the linear regression model, and use $4,000$ pairs to test it. We also compare our method with the existing baselines.

The result is shown in Table 1. The metrics include the root-mean-squared error (RMSE) and the coefficient of determination (R^2). From the result, we see that except for LINE-w, our method Tuv-year reaches the best performance. However, the method LINE-w does not capture the time-based property. For the time-varying version of LINE-w, i.e., LINE-w-time, our method Tuv shows better results than LINE-w-time in both year-long and season-long timeslots.

5.4 Occupation Prediction

Our third application is occupation prediction. Occupations are important information related to user status. For verified users on the website, there is verification information, for instance, "music producer" and "writer", which we use to obtain their true occupations. Among the 7370 users, up to $5,990$ users take the eight most common occupations, such as music, actor and business. We use $3,990$ of them as training data, and the other $2,000$ as testing data. Assuming that vectors for user u at different time slots are $\vec{v}_u(T)$, $\vec{v}_u(T + \Delta t)$, $\cdots$, $\vec{v}_u(T + (k-1)\Delta t)$, we compute the general user vector $\vec{v}_u^*$ of the user as the average of all the above vectors.

Figure 3 shows the vector distribution of the users in three most common occupations, obtained by projecting the 100-dimension vectors onto a 2-d space by the PCA method [Pearson, 1901]. We can see that the user vectors for each occupation concentrate around a particular area, showing that our vectorization model captures hidden features of users.

After obtaining the average vector representation $\vec{v}_u^*$ of the users, we use SVM [Cortes and Vapnik, 1995] for training and predicting. For each user u, the input is the average vector $\vec{v}_u^*$. For comparison, except for the above methods, we also include the average text message vectors, i.e., the average of all text message vectors published or forwarded by a user, to represent the user. The prediction accuracies of Tuv and the baselines are shown in Table 2. For every scheme evaluated, we also created the "x+text" version of it, by concatenating the average text message vectors with the resulting vectors. Doing so leads to a better prediction accuracy for all schemes.

Note that the F_1 scores [Yang, 1999] are common accuracy measures for multi-class classification. In Table 2, "normal" refers to the algorithm with input being vectors generated by Tuv (or DeepWalk, LINE); "only text" refers to the case when the input are average sentence vectors, and "x+text" refers to the case when the input are vectors obtained by concatenating the Tuv vectors and average sentence vectors.

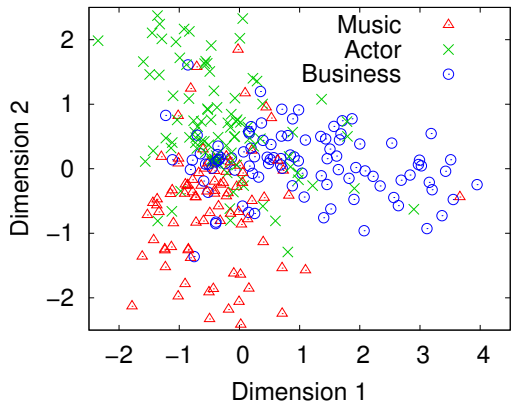

Figure 3: 2-d vector distribution of users in the three most common occupations under Tuv. Here we set $\Delta t = 1$ year.

Method	normal		x+text	
	F_1-mic	F_1-mac	F_1-mic	F_1-mac
Only text	-	-	0.668	0.520
DeepWalk	0.616	0.391	0.655	0.493
LINE	0.631	0.400	0.660	0.466
LINE-w	0.619	0.402	0.659	0.465
Tuv-season	0.650	0.495	0.662	0.528
Tuv-year	**0.668**	**0.517**	**0.681**	**0.543**

Table 2: Prediction result of occupations in F_1-micro and F_1-macro.

We see that Tuv outperforms the methods of DeepWalk, LINE and LINE-w for both the normal versions and the "x+text" versions. It is interesting to see that all "x+text" versions perform better, showing that text message vectors contain strong information for user occupations. It can also be observed that Tuv achieves a significant improvement after merging with the average text message vectors, while other baseline methods do not provide an obvious improvement.

6 Conclusion

In this paper, we propose a novel framework, called time-varying user vectorization (Tuv), for studying user relationship in online social networks. Tuv is different from previous works on user vectorization in that it focuses on utilizing user interaction activities, and can reflect changes of user properties over time. Our model takes into account text messages for all activities, original author and last forwarder for forwarding activities and time information. We have also considered the time-varying effect of user vectors. Our experimental results show that our Tuv scheme is efficient in characterizing user features and is suitable for various applications important to Internet business, including user attention evaluation, user similarity computing and occupation prediction.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China Grant 61672316 and Grant 61303195, the Tsinghua Initiative Research Grant, and the China Youth 1000-Talent Grant.

References

[Adi *et al.*, 2017] Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *ICLR*, 2017.

[Amir *et al.*, 2016] Silvio Amir, Byron C Wallace, Hao Lyu, and Paula Carvalho Mário J Silva. Modelling context with user embeddings for sarcasm detection in social media. In *CoNLL*, 2016.

[Arora *et al.*, 2015] Piyush Arora, Chris Hokamp, Jennifer Foster, and Gareth JF Jones. DCU: Using distributional semantics and domain adaptation for the semantic textual similarity SemEval-2015 Task 2. In *SemEval@NAACL-HLT*, pages 143–147, 2015.

[Bottou, 1991] Léon Bottou. Stochastic gradient learning in neural networks. *Neuro-Nımes*, 91(8):0, 1991.

[Chang and Lin, 2011] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: a library for support vector machines. *TIST*, 2(3):27, 2011.

[Cho *et al.*, 2016] Yoon-Sik Cho, Greg Ver Steeg, Emilio Ferrara, and Aram Galstyan. Latent space model for multimodal social data. In *WWW*, pages 447–458, 2016.

[Cortes and Vapnik, 1995] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[Coulom, 2008] Rémi Coulom. Whole-history rating: A Bayesian rating system for players of time-varying strength. In *International Conference on Computers and Games*, pages 113–124. Springer, 2008.

[Fu *et al.*, 2009] Wenjie Fu, Le Song, and Eric P Xing. Dynamic mixed membership blockmodel for evolving networks. In *ICML*, pages 329–336. ACM, 2009.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *KDD*, pages 855–864. ACM, 2016.

[Le and Mikolov, 2014] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *ICML*, pages 1188–1196, 2014.

[Levandowsky and Winter, 1971] Michael Levandowsky and David Winter. Distance between sets. *Nature*, 234(5323):34, 1971.

[Liu *et al.*, 2016] Li Liu, William K Cheung, Xin Li, and Lejian Liao. Aligning users across social networks using network embedding. In *IJCAI*, pages 1774–1780, 2016.

[Mikolov *et al.*, 2013a] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. 2013.

[Mikolov *et al.*, 2013b] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, pages 3111–3119, 2013.

[Nallapati *et al.*, 2008] Ramesh M Nallapati, Amr Ahmed, Eric P Xing, and William W Cohen. Joint latent topic models for text and citations. In *KDD*, pages 542–550. ACM, 2008.

[Pearson, 1901] Karl Pearson. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *KDD*, pages 701–710. ACM, 2014.

[Ren *et al.*, 2008] Wei Ren, Guiying Yan, Guohui Lin, Caifeng Du, and Xiaofeng Han. Detecting community structure by network vectorization. *Computing and Combinatorics*, pages 245–254, 2008.

[Rossi *et al.*, 2013] Ryan A Rossi, Brian Gallagher, Jennifer Neville, and Keith Henderson. Modeling dynamic behavior in large evolving graphs. In *WSDM*, pages 667–676. ACM, 2013.

[Sun *et al.*, 2016] Maosong Sun, Xinxiong Chen, Kaixu Zhang, Zhipeng Guo, and Zhiyuan Liu. THULAC: An efficient lexical analyzer for Chinese. 2016.

[Tang *et al.*, 2015] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. LINE: Large-scale information network embedding. In *WWW*, pages 1067–1077. ACM, 2015.

[Tu *et al.*, 2016] Cunchao Tu, Weicheng Zhang, Zhiyuan Liu, and Maosong Sun. Max-margin DeepWalk: Discriminative learning of network representation. In *IJCAI*, pages 3889–3895, 2016.

[Wilson *et al.*, 2016] Kevin H Wilson, Yan Karklin, Bojian Han, and Chaitanya Ekanadham. Back to the basics: Bayesian extensions of IRT outperform neural networks for proficiency estimation. In *EDM*, 2016.

[Yang *et al.*, 2016] Yi Yang, Ming-Wei Chang, and Jacob Eisenstein. Toward socially-infused information extraction: Embedding authors, mentions, and entities. *EMNLP*, 2016.

[Yang, 1999] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information retrieval*, 1(1-2):69–90, 1999.

[Yu *et al.*, 2015] Licheng Yu, Eunbyung Park, Alexander C Berg, and Tamara L Berg. Visual Madlibs: Fill in the blank description generation and question answering. In *ICCV*, pages 2461–2469, 2015.

[Zhang *et al.*, 2017] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. User profile preserving social network embedding. IJCAI, 2017.

[Zhao *et al.*, 2010] Xiaohan Zhao, Alessandra Sala, Christo Wilson, Haitao Zheng, and Ben Y Zhao. Orion: shortest path estimation for large social graphs. *Networks*, 1:5, 2010.

[Zhao *et al.*, 2011] Xiaohan Zhao, Alessandra Sala, Haitao Zheng, and Ben Y Zhao. Efficient shortest paths on massive social graphs. In *CollaborateCom*, pages 77–86. IEEE, 2011.