Trajectory-based Social Circle Inference

Qiang Gao
University of Electronic Science and Technology of China, Chengdu, China
qianggao@std.uestc.edu.cn

Goce Trajcevski
Iowa State University, Ames
gocet25@iastate.edu

Fan Zhou*
University of Electronic Science and Technology of China, Chengdu, China
fan.zhou@uestc.edu.cn

Kunpeng Zhang
University of Maryland, College park
kpzhang@umd.edu

Ting Zhong
University of Electronic Science and Technology of China, Chengdu, China
zhongting@uestc.edu.cn

Fengli Zhang
University of Electronic Science and Technology of China, Chengdu, China
fzhang@uestc.edu.cn

ABSTRACT

Learning explicit and implicit patterns in human trajectories plays an important role in many Location-Based Social Networks (LBSNs) applications, such as trajectory classification (e.g., walking, driving, etc.), trajectory-user linking, friend recommendation, etc. A particular problem that has attracted much attention recently – and is the focus of our work – is the Trajectory-based Social Circle Inference (TSCI), aiming at inferring user social circles (mainly social friendship) based on motion trajectories and without any explicit social networked information. Existing approaches addressing TSCI lack satisfactory results due to the challenges related to data sparsity, accessibility and model efficiency. Motivated by the recent success of machine learning in trajectory mining, in this paper we formulate TSCI as a novel multi-label classification problem and develop a Recurrent Neural Network (RNN)-based framework called *DeepTSCI* to use human mobility patterns for inferring corresponding social circles. We propose three methods to learn the latent representations of trajectories, based on: (1) bi-directional Long Short-Term Memory (LSTM); (2) Autoencoder; and (3) Variational autoencoder. Experiments conducted on real-world datasets demonstrate that our proposed methods perform well and achieve significant improvement in terms of macro-R, macro-F1 and accuracy when compared to baselines.

CCS CONCEPTS

• **Information systems** → **Location based services**;

KEYWORDS

trajectory mining, variational auto-encoder, social circle inference

ACM Reference Format:
Qiang Gao, Goce Trajcevski, Fan Zhou, Kunpeng Zhang, Ting Zhong, and Fengli Zhang. 2018. Trajectory-based Social Circle Inference. In *26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18), November 6–9, 2018, Seattle, WA, USA.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3274895.3274908

*Corresponding author: Fan Zhou (fan.zhou@uestc.edu.cn)

SIGSPATIAL'18, 2018, Washington,USA

ACM ISBN 978-1-4503-5889-7/18/11…$15.00
https://doi.org/10.1145/3274895.3274908

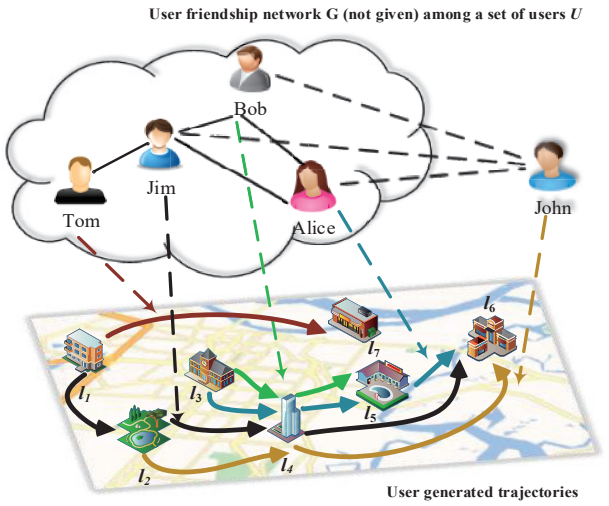


Figure 1: TSCI exemplified: The objective is to (1) reconstruct friendship among existing users and (2) identify friends for new users purely based on their trajectories.

1 INTRODUCTION

Location based social networks (LBSN) such as Wechat, Foursquare and Twitter, enable a generation of large amounts of social interaction data, part of which is based on users leaving "footprints" by checking in various locations – e.g., the places that they visited. These check-in based activities often form a sequence of spatial locations over time, thereby capturing users' moving trajectories which, in turn, can be analyzed and used for various LBSN applications such as: (1) trajectory classification (classifying user trajectories into different moving modes, e.g., driving, walking, etc.) [48]; (2) trajectory-user linking (finding the associated user who produced the specific trajectory [16]); and (3) social circles discovery (identifying user's friends) [40].

In this paper we study a related problem: *Trajectory-based Social Circle Inference* (TSCI) – however, we tackle a specific variant, different from the existing studied problems in that we do not have

any explicit information about social networks among users. TSCI usually involves two sub-tasks, as illustrated by the small-scale example in Figure 1: (1) Finding friends for users by learning latent patterns of trajectories generated by these users – e.g., Bob, Alice and Jim; and (2) Identifying potential friends from a set of existing users for a newly added user – e.g., John is a potential friend of Bob Alice and Jim.

We formally define TSCI as follows:
Given a Trajectory T generated by a user u, learn a model $\mathbb{M}$ to predict a set of users U who are in the same social circle as u or u's friends: $\mathbb{M}(T) \mapsto U$.

We believe that addressing this problem and generating effective methodology for its solution may lead to important insights in many real-world applications of extreme societal relevance. For example: – it can be utilized to identify a set of criminals/terrorists or find out their partners for a set of given anonymized trajectories; – it can help recommend friends for new visitors with mobility data; – it can also be used for friendship reconstruction, i.e., rebuilding the social circles according to their respective trajectories.

In this work, we formulate the TSCI problem from the perspective of multi-label classification. The challenges of TSCI problem stem mainly from the following three issues:

(1) *Data Sparsity* – the number of locations each user has visited is limited, given the existence of large amounts of unique locations.
(2) *Varying Sociability* – some users are gregarious and have more friends while others are not that socially active in making friends.
(3) *Feature exploiting* – identifying and extracting representative features from trajectories to measure the likelihood of friendship is difficult because there are no features (e.g., related to lifestyle, mobility behaviors) generic for every user.

The deep learning paradigm has achieved remarkable successes in problems from the fields of image processing and natural language processing. Specifically, the generative models such as Variational AutoEncoder (VAE) [23, 24] have shown promising performance in various tasks (e.g., text classification [39], machine translation [49], generating sentences [4]), which is attributed to its superior capabilities of capturing spatio-temporal features. Motivated by this, we propose a model called DeepTSCI based on adopting *recurrent neural networks* (RNN) and *VAE variants* to learn latent patterns of trajectories and consequently infer their social circles. To improve the performance, we incorporate unlabeled data into the model via pre-training using autoencoder.

To demonstrate the effectiveness of our proposed model on TSCI, we conducted experiments on real-world datasets, and the results demonstrate that our model achieves significant performance improvement as compared to baselines. The contributions of this paper are four-fold:

- We make the first attempt to address the *Trajectory-Social Circle Inference* problem using deep neural networks in the context of LBSN.
- We leverage both check-in embedding to a low dimensional space (capturing their latent features in an unsupervised manner) and trajectory embedding to improve the performance. We pre-train all sub-trajectories with autoencoder to enrich the model for capturing temporal patterns in trajectories.
- We use latent variables in VAE to approximate the distribution of trajectory friendship, and thus improve the performance of social circle inference.
- Our experiments, conducted on real datasets, demonstrate the effectiveness and robustness of our model in terms of macro-R, macro-F1 and accuracy, compared to the state-of-the-art baselines.

In the rest of this paper, we review the relevant related works in Section 2 and present the details of our model in Section 3. Experimental results are discussed in Section 4, followed by conclusions and directions for future work in Section 5.

2 RELATED WORK

Due to its impact in different LBSN applications contexts, various aspects of learning human mobility have been tackled from different research perspectives. We now review the relevant literature in (deep learning based) human mobility mining, friendship recommendation, and multi-label classification.

(Deep learning based) Human Mobility Mining: Traditional trajectory mining mainly focuses on detecting mobility similarity and capturing moving patterns [17, 18], along with exploiting trajectory semantics [44]. Most methods rely on finding the similarity of long trajectories, such as the Longest Common Sub-sequence, Edit distance and Co-visit [11, 29, 42, 45]. Chen et. al. proposed a learning-based approach for exploiting the sequence of POIs and transition patterns to recommend trajectories [6]. Liu et. al. integrated various factors to make POI recommendation including regional popularity, user mobility and user preferences, etc [28].

Deep learning technologies such as RNNs have been successfully applied in various domains, including next location prediction [30, 40], and trajectory classification [16]. Variational Autoencoder (VAE) [23, 24] also achieves good performance in sequence learning and classifying textual documents [39]; and learning effective representations via text hashing [5]. In this paper, we adopt VAE to capture implicit and explicit features in trajectories, and along with RNN we achieve significant improvement compared to baselines.

Friendship Recommendation and Social Structures: Friend recommendation in LBSN is mainly based on similarities of trajectories and other sources of data, for which paradigms like SVM and Decision Trees have been employed [2, 3]. Extracted features capturing local and global spatio-temporal characteristics of trajectories were used in [37] to check whether two users are similar enough to become friends. Matrix factorization, a well-established technique used in recommender systems, has also been successfully applied in link prediction and friend recommendation [3, 42], and inferring social network structures in ecology [32]. In contrast, the TSCI learns user mobility patterns to infer their friends, classifying whether or not a user is a friend of the generator for given (sub-)trajectories.

Multi-label Classification: Multi-label learning predicts multiple labels for unknown testing instances and has achieved successes in various applications, e.g., image tagging and text classification [38,

47]. Many methods have been proposed, such as ML-SVM, ML-KNN, etc [34, 46]; even using Multi-Label Learning as a special case of Multi-Instance Multi-Label (MIML) Learning [50, 51]. A novel method for optimizing multivariate performance measures that has shown superior performance is presented in [1], and Deep MIML for multi-label classification, which outperformed traditional methods in NLP (Natural Language Processing) and image classification was proposed in [14]. In this paper, we consider TSCI as a multi-label classification problem – and DeepTSCI achieves improved effectiveness and state-of-the-art performance for inferring users' social circles (friends).

3 METHODOLOGY

We now present the details of the DeepTSCI model and the proposed methodology, finalizing the section with a discussion of algorithmic details.

We denote the set of check-ins by $\mathcal{L}$, and $\Gamma = \{T_1, T_2, ..., T_n\}$ denote the set of all (sub)trajectories (each T_i is a sequence of check-ins and split based on some time interval), where n is the total number of trajectories. Let $T_i = (l_1, ..., l_t, ..., l_{N_i})$, where l_t is the element of $\mathcal{L}$ ($l_t \in \mathcal{L}$), and N_i is the length of (sub)trajectory T_i.

While the user generating T_i ($T_i \in \Gamma$) is anonymous, the goal of TSCI is to identify friends of T_i in the same social circle as this generator. Considering TSCI as a multi-label classification problem (similarly to [33, 43]), we define the friend label set as $U = (u_1, u_2, ..., u_F)$, and each (sub)trajectory T_i is associated with a subset U_i of U ($U_i \subseteq U$), where F is the total number of all members. The task of TSCI is to learn classifiers that link trajectories to members who are friends of their owner: $T_i (\in \Gamma) \mapsto U_i (\subseteq U)$. In addition to trajectories, we encode the relationship space as $\mathbf{Y}_i = (y_1, ..., y_k, ..., y_F)$ – i.e., if $y_k = 1 (1 \le k \le F)$, then u_k is T_i's friend; otherwise ($y_k = 0$), u_k is not a friend of T_i.

There are two focal subtasks of TSCI: (1) We use a subset of (sub)trajectories to learn latent patterns of relations among users and trajectories for friend prediction and then use the rest as the testing set to evaluate the prediction performance, denoted as RT-TSCI. (2) We predict friends of trajectories for new users whose trajectories never appear in the training set, denoted as RU-TSCI.

3.1 Trajectory Pre-processing

The trajectories are subject to pre-processing which consists of two main phases:

Trajectory Splitting: To reduce the computational overheads and capture a richer semantics of the moving patterns, we divide each trajectory into several consecutive (sub)trajectories with fixed time-interval – e.g., 6 hours. There exist many trajectory splitting methods, e.g., based on semantic meaning, shape of trajectories, etc [40]. In this paper, we adopt the simple method used in [16].

Check-in Embeddings: Inspired by [16, 31], we embed each check-in into a low dimensional vector instead of using traditional representation methods, such as one-hot. We obtain the check-in representation $\mathbf{v} \in \mathbb{R}^{|\mathcal{L}| \times d}$ by maximizing the probabilities of check-ins given their context in trajectories. Let $|\mathcal{L}|$ denote the number of check-ins in the dataset, and d be the dimensionality in the lower dimensional space. The check-ins in all trajectories can be easily embedded into the latter RNN model. More specifically, the embedding of a check-in l_t is to predict its probability given the context check-ins $l_{t-w} : l_{t+w}$. We define the context of check-in l_t as $C(l_t) = l_{t-w} : l_{t+w}$, where w is the size of sliding window of sub-trajectories.

In this paper, we consider to utilize the Continuous Bag-of-Words (CBOW) architecture, which is predicting a word given its context in natural language processing area, to embed our check-in into a low dimensional representation $\mathbf{v}(l_t) \in \mathbb{R}^d$. The probability $p(l_t|C(l_t))$ is defined by the *softmax* function as:

$$p(l_t|\mathbf{C}(l_t)) = \prod_{l' \in \mathbf{C}(l_t)} p(l_t|l') = \prod_{l' \in \mathbf{C}(l_t)} \frac{\exp(\mathbf{v}(l_t) \cdot \mathbf{v}(l'))}{\sum_{l'' \in \mathcal{L}} \exp(\mathbf{v}(l'') \cdot \mathbf{v}(l'))} \quad (1)$$

But it is expensive to compute the denominator of Eq.(1), because there is a need to enumerate each check-in $l'' \in \mathcal{L}$. Hence, we utilize the *hierarchical softmax* technique to alleviate this issue. The Huffman tree adopted in *hierarchical softmax* can obtain the better performance and higher efficiency for organizing these check-ins.

3.2 DeepTSCI Framework

We now describe the overall framework of our approach and present the details of the three proposed methods: LSTM-based (DeepTSCI-LSTM, Bi-DeepTSCI), autoencoder-based (DeepTSCI-AE), and variational autoencoder-based (DeepTSCI-VAE) TSCI. An illustration is provided in Figure 2. We note that: (a) LSTM has been widely studied and has demonstrated a superb performance in dealing with long sequences [20]; and (b) some alternative variant RNNs can also be adopted here, such as GRU [9, 16] and bidirectional RNNs [7, 25]. However, these are not the core parts of the current work.

3.2.1 *DeepTSCI-LSTM/Bi-DeepTSCI for TSCI*.

In this setting, we model trajectories using the LSTM (cf. [20]). Specifically, for each (sub)trajectory T_i, let h_{t-1} and h_t ($t \in \{1, 2, .., N_i\}$) denote the previous and current hidden state, respectively. The LSTM model used in DeepTSCI is implemented as follows:

$$\begin{aligned} i_t &= \sigma(W_p \mathbf{v}(l_t) + U_p h_{t-1} + V_p c_{t-1} + b_p) \\ f_t &= \sigma(W_f \mathbf{v}(l_t) + U_f h_{t-1} + V_f c_{t-1} + b_f) \\ o_t &= \sigma(W_o \mathbf{v}(l_t) + U_o h_{t-1} + V_o c_{t-1} + b_o) \end{aligned} \quad (2)$$

where i_t, f_t, o_t and b_* are respectively the input gate, forget gate, output gate and bias vector; σ is a logistic *sigmoid* function; matrices W_*, U_* and V_* ($\in \mathbb{R}^{d \times d'}$) are the different gate parameters, and d' is the hidden size in RNN module; $\mathbf{v}(l_t)$ is the embedding of the check-in location l_t. The memory cell c_t is updated by partially replacing the existing memory unit with a new cell c_t as:

$$c_t = f_t c_{t-1} + i_t \tanh(W_c \mathbf{v}(l_t) + U_c h_{t-1} + b_c) \quad (3)$$

where $tanh(\cdot)$ refers to the hyperbolic tangent function. The trajectory embedding is then updated by:

$$h_t = o_t \odot \tanh(c_t) \quad (4)$$

where $\odot$ is the element-wise product. We refer to this straightforward LSTM based TSCI solution as DeepTSCI-LSTM and its

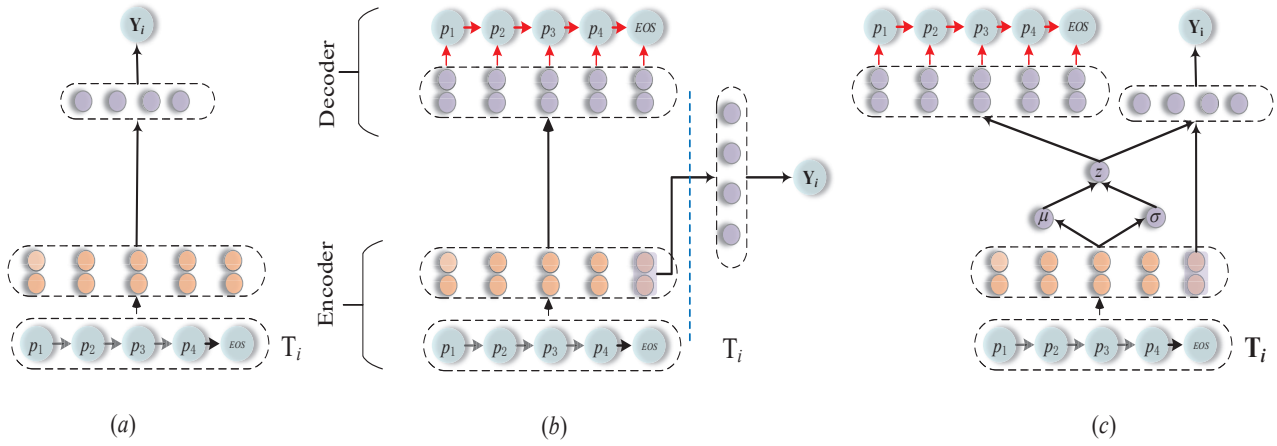


Figure 2: Overview of three proposed methods for the TSCI problem. The input consists of check-in embeddings of trajectories (low-dimension representation). The output is a list of possible friends with probabilities for each (sub)trajectory. (a) The architecture of DeepTSCI using LSTM or bi-directional LSTM to encode trajectories and make a multi-label classification, called DeepTSCI-LSTM and Bi-DeepTSCI. (b) DeepTSCI-AE leverages more data (trajectories in the testing set) via a pre-training process under an unsupervised component (e.g., encoder-decoder) to improve the performance. (c) uses variational autoencoder (VAE) to learn the implicit distributions capturing relationships between users and trajectories via a latent variable z. Combining with encoder, VAE can make a better classification. We define this method as DeepTSCI-VAE.

Bi-directional LSTM variant as Bi-DeepTSCI. As the input of classifier in this method, we define input feature as:

$$I = h_t \tag{5}$$

where I is the input of the classifier.

3.2.2 *DeepTSCI-AE for TSCI.*

Due to its advanced capabilities, autoencoder has been widely integrated into deep neural nets for extracting latent features of unlabeled data, especially in text classification problems [10, 36] – and RNN-based autoencoder is often used in the process of pre-training.

In the context of DeepTSCI, we apply autoencoder on all trajectories (including those in the testing set) to learn the latent characteristics. This pre-trained autoencoder will be used to decode friendship in the subsequent steps. The competitive gains of this autoencoder-based DeepTSCI are mainly based on leveraging a lot more unlabeled data. The LSTM also adopts the sequence autoencoder, but only in the initialization stage (cf. [10, 39]).

3.2.3 *DeepTSCI-VAE for TSCI.*

Variational autoencoder was proposed in [24, 35] and has been applied in many areas [22, 26, 39, 49]. It provides an efficient way to approximate the posterior of a distribution of latent variables. It derives a lower bound for the marginal likelihood of the observed data (sometimes denoted ELBO). In this paper, we assume a latent variable $\mathbf{z}$ for (sub)trajectories to capture their characteristics, whose true posterior distribution ($p(\mathbf{z}|\Gamma)$) is usually too complicated to have an analytical form. The typical solution is to find a distribution in an exponential family to approximate the true posterior: $q(\mathbf{z}) \sim p(\mathbf{z}|\Gamma)$. The Kullback-Leibler ($KL$) divergence is often used to measure the distance between two distributions (KL is always non negative). Thus, our objective is to minimize the KL divergence between $p(\mathbf{z}|\Gamma)$ and $q(\mathbf{z})$. Using the Bayesian theory, we have:

$$\begin{aligned} KL(q(\mathbf{z})||p(\mathbf{z}|\Gamma)) &= \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z}|\Gamma)} d\mathbf{z} \\ &= \int q(\mathbf{z})[\log q(\mathbf{z}) - \log \frac{p(\Gamma|\mathbf{z})p(\mathbf{z})}{p(\Gamma)}]d\mathbf{z} \\ &= \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z})} d\mathbf{z} - \int q(\mathbf{z}) \log p(\Gamma|\mathbf{z}) d(\mathbf{z}) + \log p(\Gamma) \end{aligned} \tag{6}$$

Here, since $\log p(\Gamma)$ does not depend on $\mathbf{z}$, the term can be considered as a constant for the purpose of the optimization task. Thus, we have:

$$\begin{aligned} &\log p(\Gamma) - KL(q(\mathbf{z})||p(\mathbf{z}|\Gamma)) \\ &= \int q(\mathbf{z}) \log p(\Gamma|\mathbf{z}) d\mathbf{z} - KL(q(\mathbf{z})||p(\mathbf{z})) \end{aligned} \tag{7}$$

In accordance to [12, 24], we choose a data dependent empirical distribution $q_\phi(\mathbf{z}|\Gamma)$ instead of an arbitrary $q(\mathbf{z})$ as the approximate posterior . Therefore, the lower bound of the marginal log likelihood of data (trajectories) becomes:

$$\log p_\theta(\Gamma) \geq \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\Gamma)}[\log p_\theta(\Gamma|\mathbf{z})] - KL(q_\phi(\mathbf{z}|\Gamma)||p(\mathbf{z})) \tag{8}$$

where we assume that $p(\mathbf{z}) \sim N(0, I)$, $q(\mathbf{z}|\Gamma) \sim N(\mu(\Gamma), \sigma(\Gamma)^2)$, and the parameters ϕ and θ will be learned via deep neural network models. The two terms on the right side of Equation (8) can be calculated separately: the first term can be considered as measuring the distance between the real Γ and the predicted $\widetilde{\Gamma}$ with cross entropy (cf. Equation (9)), while the second term denotes the

KL divergence between prior $p(\mathbf{z})$ and posterior $q_\phi(\mathbf{z}|\Gamma)$, defining *loss_KL* in Equation (10):

$$loss(\Gamma, \widetilde{\Gamma}) = \sum_{c'' \in \Gamma, \tilde{c}'' \in \tilde{\Gamma}} -c'' \cdot \log(\tilde{c}'') \tag{9}$$

$$loss\_KL = KL(q_\phi(\mathbf{z}|\Gamma)||p(\mathbf{z})) \tag{10}$$

where c'' and $\tilde{c}''$ denote the one-hot check-in distribution in Γ and the predicted check-in probability distribution in $\tilde{\Gamma}$, respectively. The objective of VAE is to minimize L_V:

$$L_V = loss(\Gamma, \widetilde{\Gamma}) + loss\_KL \tag{11}$$

The latent variable $\mathbf{z}$ is represented by a Gaussian variable:

$$\mathbf{z} = g_\theta(\epsilon, \Gamma) \tag{12}$$

where ϵ is an independent Gaussian noise variable: $\epsilon \sim \mathcal{N}(0, I)$. We note that the VAE introduces a reparameterization trick to reparameterize μ and σ^2:

$$g_\theta(\epsilon, \Gamma) = \mu + \sigma \odot \epsilon \tag{13}$$

where μ and σ^2 represent the respective mean and variance of the latent variables $\mathbf{z}$, both of which can be learned from the data via autoencoder; and $\odot$ denotes an element-wise product. In order to minimize the loss$(\Gamma, \widetilde{\Gamma})$, we choose the *softplus* (i.e, $f(x) = \ln(1+e^x)$) as the activation function to deal with the input of the decoder:

$$softplus(W_s\mathbf{z} + b_s) \tag{14}$$

where W_s and b_s denote the weight and bias, respectively.

Observe that in our VAE-based DeepTSCI, we improve Equation (5) by combining the latent variable $\mathbf{z}$ with the output in the last hidden states – as illustrated in Figure 2 (c). We also note that we can either use the mean $\bar{h}$ of all hidden states, or the hidden state at the last step h_t for friends prediction – and, in this paper, we choose the mean $\bar{h}$ for friend prediction.

$$I = W_v \begin{bmatrix} \bar{h} \\ \mathbf{z} \end{bmatrix} + b_v \tag{15}$$

where W_v and b_v are the weight and bias, respectively. Note that we use the VAE encoder which has been pre-trained to initialize parameters of the neural network classifier.

3.3 Multi-Label Classifier for TSCI

We next design a multi-label classifier, called Friend Decoder (FD), to decode the relationships among trajectories and labels. We define $FD(\Gamma)$, denoting the friendship set, as:

$$FD(\Gamma) = \begin{bmatrix} \mathbf{x}^0 \\ \mathbf{x}^1 \end{bmatrix} = W_\tau I + b_\tau \ , \mathbf{x}^0, \mathbf{x}^1 \in \mathbb{R}^F \tag{16}$$

where $\mathbf{x}^0$ means the probability set that this trajectory does not have the friendship with each label, and $\mathbf{x}^1$ means the probability set that this trajectory has the friendship with each label. Note that w_τ and b_τ denote the parameterized weight and bias, respectively. The final predicted multi-label is accordingly computed as:

$$\tilde{\mathbf{Y}} = (s([x_1^0, x_1^1]), \cdots, s([x_k^0, x_k^1]), \cdots, s([x_F^0, x_F^1])) \tag{17}$$

where $s(\cdot)$ is the $softmax$ function.

The list of predicted members/friends can be obtained from $\tilde{\mathbf{Y}}$ by comparing the probabilities of all (x_k^0, x_k^1) pairs – i.e., if $x_k^0 < x_k^1$, f_k is a member/friend of the trajectory Γ.

For RT-TSCI, we aim at minimizing the cost between the true $\mathbf{Y}$ and the empirical $\tilde{\mathbf{Y}}$. We train our model to maximize the log-likelihood with the parameter set κ. At each training step, stochastic gradient descent is used to estimate the parameter set κ as:

$$\tilde{\mathbf{Y}}(\Gamma, \kappa) \mapsto \mathbf{Y}(\Gamma)$$

To test RU-TSCI, we use the trained model to make friend prediction for new trajectories which are generated by new users – i.e., these users and their trajectories do not exist in the training set.

3.4 Implementation Aspects

We now turn the attention to a few typical issues intricately involved in our models: Dropout, KL cost annealing, Algorithmic Issues.

3.4.1 *Dropout*.

In order to alleviate the problem of overfitting inherent to RNN models, we apply a variational inference based on the dropout technique (cf. [15]) to our model for pre-training and learning. To ensure a well embedding of trajectories for the RNN model, we randomly drop some check-ins in the training. The dropout we use is defined similarly to [15].

3.4.2 *KL cost annealing*.

As demonstrated in [4], a dynamic weight w_{KL} ($0 \le w_{KL} \le 1$) added to the *KL* term in the VAE model can be very effective in training . At the start of VAE training, w_{KL} is zero, and it can make the model encode rich trajectory information to latent variables $\mathbf{z}$. The value of w_{KL} gradually increases as the training proceeds. Eventually, it will force the VAE to smooth out encodings and pack into the prior as we specified. Thus, the L_V now becomes:

$$L_V = loss(\Gamma, \widetilde{\Gamma}) + w_{KL} * loss\_KL$$

$$w_{KL} \leftarrow w_{KL} + step * \alpha$$

where *step* denotes the iterations of training, and α is the weight factor that can be set (or adjusted) by the users.

3.4.3 *Algorithmic Issues*.

With the sparsity of trajectories generated by various users, it may very well be the case that a number of relatively short trajectories have been generated in a given time-interval. In the interest of speeding up the training process and capturing more trajectory features, we sort all trajectories by their length. This, in turn, enables to organize each batch of trajectories with similar length as a bucket. This procedure provides two advantages for training and predicting: (1) It can speed up the process of the trajectory embedding, while still handling well various trajectories with different lengths. (2) It reduces the unnecessary padding for the respective batches of trajectories, which implicitly forces the RNN model to capture more trajectory features.

The process of DeepTSCI-VAE is summarized in Algorithm 1.

Algorithm 1: Training Algorithm of DeepTSCI-VAE.

Input: Trajectory:$T_i \in \Gamma$; User $U_i \in U$.
Embed Check-ins into vector $\mathbf{v}(l_t)$
/* Pre-training */
foreach $T_i \in \Gamma$ **do**
  Variational encode T_i to obtain latent variable $\mathbf{z}_i$ ($\in \mathbf{z}$);
  Decode latent variable $\mathbf{z}_i$ to obtain $\tilde{T}_i$;
  Minimize the cost L_V according to Eq.(11).
end
Output: Pre-training Model $\mathbb{M}_0$.
/* Training */
Training dataset $\mathcal{D} \leftarrow \emptyset$.
foreach $U_i \in U$ **do**
  $\mathcal{D} \leftarrow < T_i, U_i >$;
end
repeat
  foreach $< T_i, U_i > \in \mathcal{D}$ **do**
    Encode $U_i (\in U)$ to $\mathbf{Y}_i (\in \mathbf{Y})$;
    Variational encode T_i to obtain $\mathbf{z}_i$ and $\bar{h}_i (\in \bar{h})$;
    Decode $\mathbf{z}_i$ and $\bar{h}_i$ to obtain $\tilde{T}_i (\in \tilde{\Gamma})$ and empirical $\tilde{\mathbf{Y}}_i$ ($\in \tilde{\mathbf{Y}}$);
    Predict friendship $\tilde{\mathbf{Y}}_i$ via Eq.(15)-(17);
    Minimize L_V (Eq.(11)) and the cost between $\tilde{\mathbf{Y}}_i$ and $\mathbf{Y}_i$.
  end
until *converge*
Output: Training Model $\mathbb{M}$.

4 EXPERIMENTAL RESULTS

We now describe in detail the setup of our experiments and present the quantitative observations regarding the advantages of our approaches. We start with discussing the real-world datasets used in our experimental evaluations, followed by the parameters setting and metrics, as well as the list of the baselines used for comparisons in the experimental results. We note that the source code of the implementations used in our experiments is publicly available[1].

4.1 Datasets

To evaluate the performance of DeepTSCI in comparison with the baselines, we chose three popular and widely used datasets: Brightkite, Gowalla [8] and Foursquare [41, 42]. For Foursquare, we choose two most popular cities Tokyo and New York. All datasets include check-in and social network information. Specifically, each visit consists of user ID, location ID and a time stamp. Recall that this paper focuses more on learning mobility patterns to predict users' social circles – thus, chunking trajectories into sub-trajectories might improve the performance because they can capture more information about mobility patterns.

We conducted two different sets of experiments regarding predicting friend set for each trajectory: RT-TSCI and RU-TSCI. For RT-TSCI, we chose 50% of sub-trajectories of each user for training, and the rest for testing. For RU-TSCI, we infer social circle members for new trajectories generated by those who are not in the training set.

Brightkite and Gowalla: We first extracted user relationships to construct friends set as a multi-label set, and we concatenate all check-ins to form trajectories for each user. A long trajectory is further divided into sub-trajectories based on the time interval (e.g., 6 hours). Next, we link the friend set to each sub-trajectory for each user, and remove the user ID (for privacy preservation). For Brightkite and Gowalla, we randomly selected 201 and 92 users for RT-TSCI, respectively. The number of sub-trajectories is still large (in the order of thousands) in the training set and test set. In the RU-TSCI test, for the purpose of robustness, we chose new trajectories whose generators have at least 5 friends existing in the selected users for RT-TSCI. Thus, we effectively chose 514 users for Gowalla and 199 users for Brightkite.

Foursquare data in Tokyo and New York: We extracted users profile information to obtain their social connections (follower-followee) and applied the method proposed in [42] to construct respective social networks. The users and their corresponding trajectories are selected as follows. We first randomly chose 60 seeding users for Tokyo and 40 for New York. They also became our two global friend sets. We then explored the social network to find users who have at least 5 friends in the global sets. Among these users, we then randomly chose 200 and 150, with their corresponding trajectories, as the final dataset for Tokyo and New York, respectively. During selection, trajectories of users who have at least 5 friends but not in the RT-TSCI set are used for RU-TSCI test. The details of dataset description are shown in Table 1.

Table 1: Datasets description. F: the number of members in the global social circle set; U_{RT}: the number of users for RT-TSCI test; U_o: the number of users for RU-TSCI test; T_{train}/T_{test}: the number of sub-trajectories in training vs. the number of sub-trajectories in testing for RT-TSCI test; T_o: the number of sub-trajectories for RU-TSCI test.

Dataset	F	U_o	U_{RT}	T_{train}/T_{test}	T_o
Brightkite	92	199	92	10012/10061	37403
Gowalla	201	514	201	10104/10052	59514
Tokyo	60	248	200	9036/9153	22769
New York	40	62	150	6220/6295	5185

4.2 Parameter Settings

We now describe the parameters involved in the neural networks and the training for DeepTSCI. The check-in embedding leverages Skip-gram model with the window size of 5; and 10 in the negative samplings. We embed each check-in into a low dimensional latent space. The detailed settings are shown in Table 2. We tested on various parameter settings and did not find much variation – and the subsequent results are reported based on the fixed parameters in Table 2.

4.3 Metrics

Similar to [27, 34], we report macro-R, macro-F1 and accuracy to evaluate all experimental results for the TSCI problem. Let N denote the number of trajectories in the testing set. macro-R is the average proportion of predicted friends that are also in the ground truth. macro-F1 is defined as the harmonic mean of macro-P and macro-R.

[1] https://github.com/gcooq/TSCI

Table 2: Parameters used in all experiments. "window size" means how much contextual information we consider in the CBOW model for embedding; "embedding size" is the vector size of the embedded low dimension space; "hidden size" is the number of neuron units in the hidden layer; "initial lr": initial learning rate; "epoch lr" means that we will gradually decrease the value of learning rate in a fixed value; $|\mathbf{z}|$ denotes the dimensionality of latent variable $\mathbf{z}$; α is the the weight factor used in pre-training.

Parameters	Brightkite	Gowalla	Tokyo	New York
windows size	5	5	5	5
embedding size	250	250	200	200
hidden size	500	800	500	800
initial lr	0.001	0.001	0.001	0.001
epoch lr	8×10^{-6}	8×10^{-6}	2×10^{-6}	2×10^{-6}
dropout rate	0.5	0.5	0.5	0.5
batch size	16	16	16	16
$\|\mathbf{z}\|$	50	50	50	50
$\boldsymbol{\alpha}$	8×10^{-6}	8×10^{-6}	8×10^{-6}	8×10^{-6}

We also demonstrate the accuracy, as defined in [34], to report the performance of the models.

$$\text{macro-P} = \frac{1}{N} \sum \frac{\#\text{ correctly predicted friends}}{\#\text{ predicted friends}}$$

$$\text{macro-R} = \frac{1}{N} \sum \frac{\#\text{ correctly predicted friends}}{\#\text{ True friends}}$$

$$\text{macro-F1} = \frac{2 \times (\text{macro-P}) \times (\text{macro-R})}{(\text{macro-P}) + (\text{macro-R})}$$

$$\text{accuracy} = \frac{1}{N} \sum \frac{\#\text{ correctly predicted friends}}{\#\text{ True Friends} \cup \#\text{ predicted friends}}$$

4.4 Baselines

We now present the baselines used for comparison with our models.

• **Co-visit**: Longest Common Sub-sequence (LCS) has been widely used in computing trajectory similarity [19]. It cannot be directly applied to our TSCI because users generate very few check-ins in a fixed time interval. However, it is useful to find user common interests or locations [42]. For training data, we concatenate these sub-trajectories which have the same friendship label set. Each friend label associates with a concatenated trajectory. Then for a new testing trajectory, we identify their friends from similar trajectories in terms of common check-ins in the training data. The threshold for the number of co-visits can be manually set.

• **MF**: Matrix Factorization has been successfully applied in extracting latent characteristics of trajectories and social networks [3]. For our TSCI, we construct a friend check-in frequency matrix where each cell represents the number of times the corresponding check-in has been visited by that user in all trajectories. To obtain the friend list for a new testing trajectory, we calculate the similarity between its vector and every friend vector which are from the matrix factorization.

• **Decision Trees**: Decision Trees have been successfully used in friend recommendation [3]. We consider our TSCI as a multi-label classification problem. Traditional algorithms such as decision trees can then be applied to features extracted from trajectories. To capture trajectory features, we calculate the mean value of check-in vectors from trajectory.

• **SVM**: SVM algorithm has shown a better performance not only in text retrieval and mining area but also in trajectory classification. In accordance with [51], we train a linear kernel based SVM model for this multi-label classification problem in our experiments.

• **RandomForest**: Similarly to [21], we utilize the multi-label classification algorithm based on RandomForest to predict friends list for each trajectory.

• **MLP**: To show performance of basic deep learning in terms of capturing the temporal information, a multi-layer perceptron used in [14] is also compared.

• **TULER**: One of the most recent work on identifying human mobility model is proposed in [16] which leverages RNNs to capture the sequential patterns of human trajectories and to predict the generators of unknown trajectories. We train the "TULER" using a stacked GRU for the multi-label classification problem.

• **DeepMIML**: DeepMIML is a MIML learning method originally developed for sentence and image classification [14]. Since the corresponding codes have not been published yet, we re-implement it using AE to obtain the representation vector of trajectories and a 2D sub-concept layer proposed in DeepMIML to learn friends for a given trajectory.

We implemented the DeepTSCI model, TULER and Deep MIML [14] via Tensorflow, which is an open-source python library for deep learning. Most of traditional machine learning algorithms from the baselines are implemented based on the scikit-learn library. We also speed up DeepTSCI using one GTX1080 Ti GPU.

4.5 Performance Comparison

Table 3 and Table 4 show the performance comparison among proposed algorithms and baselines for RT-TSCI and RU-TSCI, respectively on four datasets, where the best performance is shown in bold, while the second best in underlined.

4.5.1 Performance for RT-TSCI.

The performance comparison of our proposed algorithms and baselines for RT-TSCI problem are shown in Table 3.

Our models vs. baseline models: As the result shows that we can infer user's social preference, and find out a majority of their social circle. Our methods achieve the best performance in all datasets in terms of macro-R, macro-F1 and accuracy except that DeepTSCI-LSTM performs a little worse compared to baselines in New York dataset. Proposed methods can well infer human social circle. For example, DeepTSCI-VAE outperforms other methods for all metrics, and gains up to 6.66%, 2.48% and 4.12% over RandomForest in Gawalla. Compared to TULER and DeepMIML, our proposed models can well capture trajectory moving patterns, and the results show that our model is well-suited to TSCI, under the framework of a multi-label classification problem.

Analysis of proposed model: In order to tackle with TSCI problem, we proposed four methods, including LSTM-based, Bidirectional LSTM-based, pre-training with Autoencoder, and Variational Autoencoder-based. Based on the results, we observe that Bi-DeepTSCI and DeepTSCI-VAE obtain the best performance. In this paper, we try to incorporate VAE to encode and decode trajectories, and embed trajectory into a lower dimensional latent vector $\mathbf{z}$. Overall, DeepTSCI-VAE achieves the best result in our proposed models.

Table 3: Performance comparison among different algorithms for RT-TSCI on four datasets.

Method	Brightkite			Gowalla			Tokyo			New York		
	macro-R	macro-F1	accuracy	macro-R	macro-F1	accuracy	macro-R	macro-F1	accuracy	macro-R	macro-F1	accuracy
Co-Visit[42]	0.5542	0.4207	0.3210	0.4144	0.3710	0.2941	0.4592	0.4463	0.3790	0.6209	0.5902	0.5339
MF[3]	0.5253	0.4121	0.2681	0.4600	0.2075	0.1122	0.2997	0.3119	0.1908	0.4608	0.4040	0.2774
Decision Trees[3]	0.5965	0.6369	0.4841	0.4493	0.5659	0.3861	0.3004	0.4100	0.2764	0.4199	0.5242	0.3791
SVM[51]	0.6348	0.6870	0.5404	0.4815	0.6218	0.4467	0.3442	0.4601	0.3246	0.3652	0.4893	0.3463
RandomForest[21]	0.6572	0.7099	0.5801	0.5473	0.6603	0.4997	0.5190	0.6043	0.5004	0.6404	0.7118	0.6199
MLP[14]	0.6524	0.6954	0.5547	0.5126	0.6288	0.4505	0.4766	0.5617	0.4401	0.4887	0.5903	0.4529
TULER[16]	0.6203	0.6896	0.5350	0.4697	0.6176	0.4358	0.5592	0.6282	0.5318	0.6487	0.7174	0.6300
DeepMIML[14]	0.5362	0.6371	0.4650	0.5239	0.5780	0.4022	0.5633	0.6010	0.5111	0.6480	0.7156	0.6273
DeepTSCI-LSTM	0.6746	0.7247	0.5866	0.5869	0.6799	0.5262	0.5797	0.6390	0.5479	0.6501	0.7103	0.6182
Bi-DeepTSCI	0.6829	0.7258	0.5955	0.6131	0.6838	0.5385	0.5886	0.6453	0.5579	0.6713	0.7203	0.6381
DeepTSCI-AE	0.6819	0.7289	0.5965	0.5636	0.6747	0.5113	0.5657	0.6373	0.5380	0.6696	**0.7249**	0.6405
DeepTSCI-VAE	**0.6999**	**0.7320**	**0.6061**	**0.6139**	**0.6851**	**0.5409**	**0.6022**	**0.6534**	**0.5704**	**0.6757**	0.7220	**0.6407**

Table 4: Performance comparison among different algorithms for RU-TSCI on four datasets.

Method	Brightkite			Gowalla			Tokyo			New York		
	macro-R	macro-F1	accuracy	macro-R	macro-F1	accuracy	macro-R	macro-F1	accuracy	macro-R	macro-F1	accuracy
Co-Visit[42]	0.1452	0.0542	0.0293	0.1010	0.0870	0.0420	0.1008	0.0894	0.0491	0.1172	0.0905	0.0522
MF[3]	0.1846	0.1006	0.0641	0.1114	0.0959	0.0511	0.1154	0.1130	0.0653	0.1051	0.0914	0.0558
Decision Trees[3]	0.1736	0.1962	0.1032	0.1150	0.1683	0.0874	0.0756	0.1089	0.0572	0.1787	0.2343	0.1320
SVM[51]	0.1736	0.1615	0.1032	0.1253	0.1878	0.0970	0.0901	0.1260	0.0643	0.1591	0.2271	0.1273
RandomForest[21]	0.1449	0.1615	0.0804	0.1352	0.1885	0.0900	0.0951	0.1251	0.0656	0.1961	0.2576	0.1442
MLP[14]	0.1584	0.1709	0.0855	0.1519	0.2101	0.1045	0.1183	0.1477	0.0778	0.2143	0.2707	0.1542
TULER[16]	0.3348	0.2525	0.1371	0.1377	0.2009	0.0999	0.1233	0.1554	0.0808	0.2489	0.2943	0.1661
DeepMIML[14]	0.2847	0.2482	0.1240	0.1524	0.1974	0.0989	0.1251	0.1579	0.0820	0.2433	0.2909	0.1620
DeepTSCI-LSTM	0.3441	0.2594	0.1362	0.1634	0.2222	0.1119	0.1345	0.1624	0.0859	0.2565	0.2996	0.1712
Bi-DeepTSCI	0.3250	0.2435	0.1297	0.1733	0.2246	0.1134	0.1342	0.1610	0.0806	0.2628	0.3036	0.1724
DeepTSCI-AE	0.3060	0.2572	0.1394	0.1568	0.2162	0.1086	0.1224	0.1552	0.0824	0.2442	0.2892	0.1625
DeepTSCI-VAE	**0.3541**	**0.2608**	**0.1404**	**0.1743**	**0.2271**	**0.1141**	**0.1386**	**0.1676**	**0.0902**	**0.2760**	**0.3101**	**0.1784**

4.5.2 Performance for RU-TSCI.

We now show the performance evaluation of RU-TSCI, cf. Table 4.

Effectiveness of RU-TSCI: Our proposed methods outperform baselines and DeepTSCI-VAE performs the best. But macro-R, macro-F1, and accuracy for all methods are relatively low. The possible reason is that we do not have any information about these new users except several short sub-trajectories. It is similar to a cold start problem in recommender systems. We use the learned model from RT-TSCI to predict social circles for these new users.

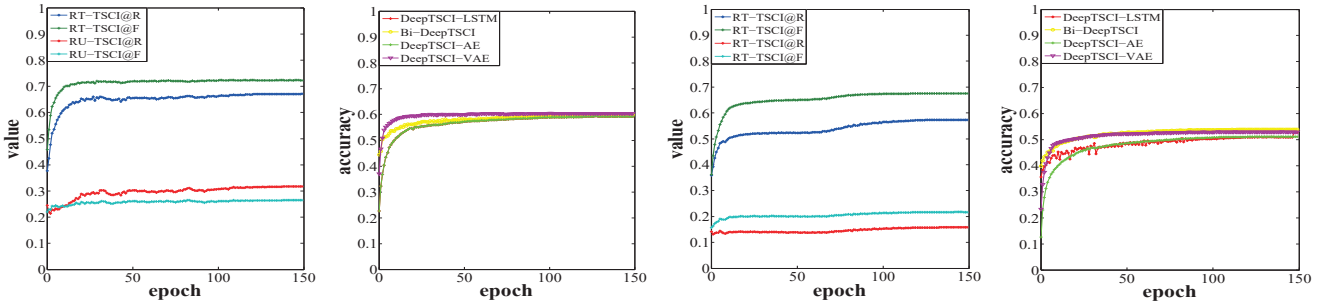
This is an important problem especially in security area. For example, a user enters into a city where he leaves some footprints within a short time period. Using our proposed model might help find his partners (related users).

Model Analysis: As the result shows in Table 4, our proposed model DeepTSCI-VAE still achieves the best performance compared to other methods. For example in New York dataset, it lifts up to 2.71%, 1.58% and 1.23% over TULER, and 1.32%, 0.65% and 0.60% comparing to Bi-DeepTSCI. For trajectory embeddings, we find out our proposed model can capture more semantic information comparing with TULER and DeepMIML methods, in which they also use RNN-based model to embed the trajectories.

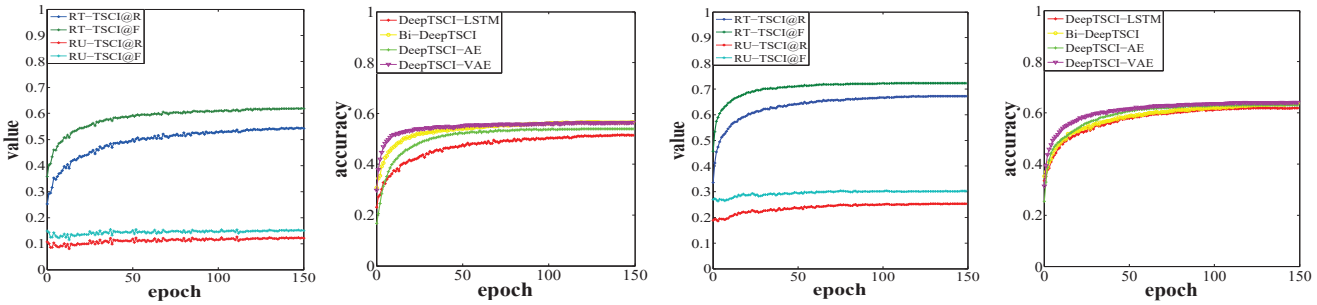
4.6 Model Robustness

In Figure 4, we depict a sub-trajectory (1068 → 32123 → 26273 → 3232 → 1371) generated within 6 hours by a randomly selected user in the New York dataset. The outcome of the predicted social fiends of this user is illustrated in the right part of Figure 4, which contains all the real friends of this user.

Next, to measure the sensitivity in terms of how the parameters involved in our models (e.g., learning rate, the number of iteration, etc.) affect the performance, we conducted experiments with various parameter settings. As shown in Figure 5, the accuracy of TSCI is proportional to the number of iterations (denoted by epoch). We



(a) DeepTSCI R-F on Brightkite. (b) RT-TSCI accuracy on Brightkite. (c) DeepTSCI R-F on Gowalla. (d) RT-TSCI accuracy on Gowalla.



(e) DeepTSCI F-R on Tokyo. (f) RT-TSCI accuracy on Tokyo. (g) DeepTSCI R-F on New York. (h) RT-TSCI accuracy on New York.

Figure 3: Results of Recall, F1 and accuracy on Gowalla, Brightkite, Tokyo and New York (@R and @F are abbreviations for macro-R and macro-F1).

decreased the learning rate in every epoch, but the accuracy did not exhibit too much variation as the learning rate changes after a certain epoch (in the sense of convergence).

Pre-training is an important component in the training process of our TSCI. In this work we used AE and VAE for pre-training. Figure 6(a) shows that the AE based pre-training can increase the accuracy, while the loss decreases. Our VAE based pre-training makes the convergence faster for both RT-TSCI and RU-TSCI tests, as shown in Figure 6(b). Once again we observe that the accuracy increases, while this time both the KL and the loss decrease. We re-iterate that we also achieved a good recall and F1 score for both RT-TSCI and RU-TSCI as the epoch increases (cf. Figure 3).

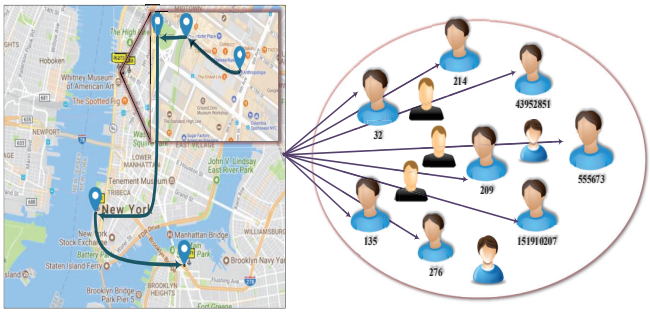
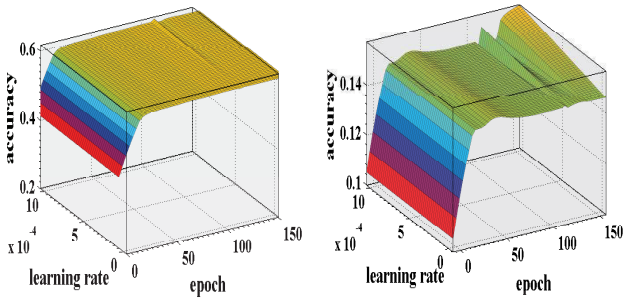
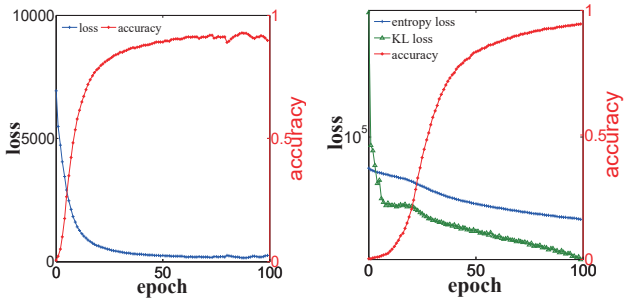


Figure 4: Visualization of the results on friend prediction for a sub-trajectory in New York dataset.



(a) RT-TSCI test on Brightkite. (b) RU-TSCI test on Brightkite.

Figure 5: Model robustness on Brightkite



(a) Pre-training AE on Gowalla. (b) Pre-training VAE on Gowalla.

Figure 6: Results of AE/VAE based pre-training on Gowalla

5 CONCLUSIONS AND FUTURE WORK

We proposed a framework – DeepTSCI – with deep learning based models to learn mobility patterns and infer users' social circles (friends) based on their trajectories. DeepTSCI integrates recurrent neural networks and autoencoder to understand latent characteristics of trajectories and does not require any extra data regrading users, such as user profile or their social connections. Experiments conducted on real-world datasets demonstrated that our proposed methods outperform baselines in terms of macro-R, macro-F1 and accuracy.

As part of our future work, we plan to incorporate the interplay of different semantic values of inter-relationships among friends (by way of corresponding labels). In addition, we plan to focus in a more detailed manner on potential improvements via supervised models like, for example, reinforcement learning methods, to better infer users' social circles in different contexts (e.g., age, gender, etc. – cf. [13]).

6 ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China (Grant No.61602097 and No.61472064), NSF grants III 1213038 and CNS 1646107, and ONR grant N00014-14-10215.

REFERENCES

[1] Apoorv Aggarwal, Sandip Ghoshal, Ankith M. S., Suhit Sinha, and Ganesh Ramakrishnan. 2017. Scalable Optimization of Multivariate Performance Measures in Multi-Instance Multi-label Learning. In *AAAI*.
[2] Mohammad Al Hasan, Vineet Chaoji, Saeed Salem, and Mohammed Zaki. 2006. Link prediction using supervised learning. In *SDM*.
[3] Basma Alharbi, Abdulhakim Ali Qahtan, and Xiangliang Zhang. 2016. Minimizing User Involvement for Learning Human Mobility Patterns from Location Traces. In *AAAI*.
[4] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. 2016. Generating sentences from a continuous space. In *CoNLL*.
[5] Suthee Chaidaroon and Yi Fang. 2017. Variational Deep Semantic Hashing for Text Documents. In *SIGIR*.
[6] Dawei Chen, Cheng Soon Ong, and Lexing Xie. 2016. Learning Points and Routes to Recommend Trajectories. In *CIKM*.
[7] Zheqian Chen, Ben Gao, Huimin Zhang, Zhou Zhao, Haifeng Liu, and Deng Cai. 2017. User Personalized Satisfaction Prediction via Multiple Instance Deep Learning. In *WWW*.
[8] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *KDD*.
[9] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* (2014).
[10] Andrew M Dai and Quoc V Le. 2015. Semi-supervised sequence learning. In *NIPS*.
[11] Hui Ding, Goce Trajcevski, Peter Scheuermann, Xiaoyue Wang, and Eamonn J Keogh. 2008. Querying and mining of time series data - experimental comparison of representations and distance measures. In *PVLDB*.
[12] Carl Doersch. 2016. Tutorial on Variational Autoencoders. *arXiv* (2016).
[13] Yuxiao Dong, Yang Yang, Jie Tang, Yang Yang, and Nitesh V. Chawla. 2014. Inferring user demographics and social strategies in mobile social networks. In *KDD*.
[14] Ji Feng and Zhi-Hua Zhou. 2017. Deep MIML Network. In *AAAI*.
[15] Yarin Gal and Zoubin Ghahramani. 2015. A Theoretically Grounded Application of Dropout in Recurrent Neural Networks. *Statistics* (2015), 285–290.
[16] Qiang Gao, Fan Zhou, Kunpeng Zhang, Goce Trajcevski, Xucheng Luo, and Fengli Zhang. 2017. Identifying Human Mobility via Trajectory Embeddings. In *IJCAI*.
[17] Fosca Giannotti, Mirco Nanni, Dino Pedreschi, Fabio Pinelli, Chiara Renso, Salvatore Rinzivillo, and Roberto Trasarti. 2011. Unveiling the complexity of human mobility by querying and mining massive trajectory data. *VLDB J.* 20, 5 (2011).
[18] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. 2007. Trajectory pattern mining. In *ACM SIGKDD*.
[19] Limin Guo, Guangyan Huang, Xu Gao, Jing He, Bin Wu, and Haoming Guo. 2015. DoSTra: discovering common behaviors of objects using the duration of staying on each location of trajectories. In *AAAI Workshop*.
[20] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.
[21] Hsun-Ping Hsieh and Cheng-Te Li. 2014. Inferring Social Relationships from Mobile Sensor Data. In *WWW Companion*.
[22] Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. 2017. Variational Deep Embedding: An Unsupervised and Generative Approach to Clustering. In *IJCAI*.
[23] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. 2014. Semi-supervised learning with deep generative models. In *NIPS*.
[24] Diederik P Kingma and Max Welling. 2014. Auto-Encoding Variational Bayes. In *ICLR*.
[25] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent Convolutional Neural Networks for Text Classification. In *AAAI*.
[26] Xiaopeng Li and James She. 2017. Collaborative Variational Autoencoder for Recommender Systems.. In *KDD*.
[27] Kwan Hui Lim, Jeffrey Chan, Christopher Leckie, and Shanika Karunasekera. 2015. Personalized tour recommendation based on user interests and points of interest visit durations. In *IJCAI*.
[28] Bin Liu, Yanjie Fu, Zijun Yao, and Hui Xiong. 2013. Learning geographical preferences for point-of-interest recommendation. In *KDD*.
[29] Hechen Liu and Markus Schneider. 2012. Similarity measurement of moving object trajectories. In *SIGSPATIAL*.
[30] Qiang Liu, Shu Wu, Liang Wang, and Tieniu Tan. 2016. Predicting the Next Location: A Recurrent Model with Spatial and Temporal Contexts. In *AAAI*.
[31] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *ICLR*.
[32] Ioannis Psorakis, Stephen J. Roberts, Iead Rezek, and Ben C. Sheldon. 2012. Inferring social network structure in ecological systems from spatio-temporal data streams. *Journal of The Royal Society Interface* 9, 76 (2012), 3055–3066.
[33] Jesse Read and Fernando Perezcruz. 2014. Deep Learning for Multi-label Classification. *Machine Learning* 85, 3 (2014), 333–359.
[34] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. 2011. Classifier chains for multi-label classification. *Machine Learning* 85, 3 (2011), 333.
[35] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. 2014. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *ICML*.
[36] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NIPS*.
[37] Hongjian Wang, Zhenhui Li, and Wang-Chien Lee. 2014. PGT: Measuring mobility relationship using personal, global and temporal factors. In *ICDM*.
[38] Yuhong Guo Xin Li. 2013. Active Learning with Multi-Label SVM Classification. In *IJCAI*.
[39] Weidi Xu, Haoze Sun, Chao Deng, and Ying Tan. 2017. Variational Autoencoder for Semi-Supervised Text Classification. In *AAAI*.
[40] Cheng Yang, Maosong Sun, Wayne Xin Zhao, Zhiyuan Liu, and Edward Y Chang. 2017. A Neural Network Approach to Jointly Modeling Social Networks and Mobile Trajectories. *TOIS* 35, 4 (2017), 36.
[41] Dingqi Yang, Daqing Zhang, Longbiao Chen, and Bingqing Qu. 2015. NationTelescope: Monitoring and visualizing large-scale collective behavior in LBSNs. *Journal of Network & Computer Applications* 55 (2015), 170–180.
[42] Guolei Yang and Andreas Züfle. 2017. Spatio-temporal Prediction of Social Connections. In *Proceedings of the Fourth International ACM Workshop on Managing and Mining Enriched Geo-Spatial Data (GeoRich '17)*. ACM, New York, NY, USA, 6:1–6:6.
[43] Chih Kuan Yeh, Wei Chieh Wu, Wei Jen Ko, and Yu Chiang Frank Wang. 2017. Learning Deep Latent Spaces for Multi-Label Classification. In *AAAI*.
[44] Josh Jia-Ching Ying, Wang-Chien Lee, and Vincent S. Tseng. 2013. Mining geographic-temporal-semantic patterns in trajectories for location prediction. *ACM TIST* 5, 1 (2013), 2:1–2:33.
[45] Josh Jia-Ching Ying, Eric Hsueh-Chan Lu, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S Tseng. 2010. Mining user similarity from semantic trajectories. In *SIGSPATIAL*.
[46] Min-Ling Zhang and Zhi-Hua Zhou. 2007. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition* 40, 7 (2007), 2038 – 2048.
[47] Shiquan Zhao, Jian Wu, Victor S. Sheng, Chen Ye, Pengpeng Zhao, and Zhiming Cui. 2015. Weak Labeled Multi-Label Active Learning for Image Classification. In *MM*.
[48] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *UbiComp*.
[49] Chunting Zhou and Graham Neubig. 2017. Multi-space Variational Encoder-Decoders for Semi-supervised Labeled Sequence Transduction. In *ACL*.
[50] Zhi-Hua Zhou and Min-Ling Zhang. 2017. *Multi-label Learning*. Springer US, Boston, MA, 875–881. DOI: http://dx.doi.org/10.1007/978-1-4899-7687-1_910
[51] Zhi-Hua Zhou, Min-Ling Zhang, Sheng-Jun Huang, and Yu-Feng Li. 2012. Multi-instance multi-label learning. *Artificial Intelligence* 176, 1 (2012), 2291 – 2320.