SERM: A Recurrent Model for Next Location Prediction in Semantic Trajectories

Di Yao[1,3], Chao Zhang[2], Jianhui Huang[1,3], Jingping Bi[1,3]
[1]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]Computer Science Department, University of Illinois at Urbana-Champaign, Urbana, IL, USA
[3]University of Chinese Academy of Sciences, Beijing, China
[1,3]{yaodi, huangjianhui, bjp}@ict.ac.cn, [2]czhang82@illinois.edu

ABSTRACT

Predicting the next location a user tends to visit is an important task for applications like location-based advertising, traffic planning, and tour recommendation. We consider the next location prediction problem for semantic trajectory data, wherein each GPS record is attached with a text message that describes the user's activity. In semantic trajectories, the confluence of spatiotemporal transitions and textual messages indicates user intents at a fine granularity and has great potential in improving location prediction accuracies. Nevertheless, existing methods designed for GPS trajectories fall short in capturing latent user intents for such semantics-enriched trajectory data. We propose a method named semantics-enriched recurrent model (SERM). SERM jointly learns the embeddings of multiple factors (user, location, time, keyword) and the transition parameters of a recurrent neural network in a unified framework. Therefore, it effectively captures semantics-aware spatiotemporal transition regularities to improve location prediction accuracies. Our experiments on two real-life semantic trajectory datasets show that SERM achieves significant improvements over state-of-the-art methods.

KEYWORDS

Location Prediction, Semantic Trajectory, RNN

1 INTRODUCTION

Next location prediction, the task of inferring the location a user tends to visit next based on her preceding GPS trace, has been considered as an important building block for various mobile computing applications. For example, in ride sharing services, predicting the next locations for different users is vital to discovering groups of users that have destinations in close proximity. Such a functionality can enable service providers to design optimized scheduling strategies that reduce operating costs and energy consumptions.

While next location prediction has been extensively studied for GPS trajectories, recent years are witnessing a rapid growth of semantic trajectory data — wherein each record in the trajectory is associated with a text message that describes the user's activity. Driven by the prevalence of smartphones, hundreds of millions people leave semantics-rich digital traces on social media websites (*e.g.*, Twitter, Instagram, Snapchat) on a daily basis [8, 9, 11]. Meanwhile, raw GPS trajectories can be readily linked with external data sources (*e.g.*, land uses, social media) to enrich GPS records. In semantic trajectories, the attached text indicates user intents and serves as a useful signal in inferring the next activity the user tends to involve. There is an increasing need for location prediction methods tailored for semantic trajectory data.

Next location prediction for semantic trajectories, however, is not trivial. Compared with next location prediction for pure GPS trajectory data, this problem introduces two unique challenges. First, the semantics of user activities are usually expressed through short text messages. It is important yet challenging to address text sparsity and capture user intentions effectively. Second, to make rationale predictions, one has to jointly model multiple factors: 1) the spatiotemporal regularity — people make next visits based on their locations and the current time; 2) the activity semantics — the semantics of the user's current activity can play an important role in deciding the next location; and 3) the personal preferences — people choose different places to visit based on their own preferences. Due to the complicated interactions of these factors, it is difficult to combine them in a unified predictive model.

Related Work. Existing next location prediction techniques can be roughly divided into two categories: pattern-based and model-based. Pattern-based methods extract user mobility patterns from historic data and predict next location with these patterns. Various methods have been proposed for mining sequential patterns [10], periodic patterns [3], *etc.*, which have been shown to be useful for location prediction [6]. Nevertheless, pattern-based methods can only mine explicit patterns defined a-priori, and cannot capture all the movement regularities in the data.

Model-based methods learn statistical models to characterize user movement regularity and make predictions with these learned models. Different predictive models have been proposed, such as hidden Markov models (HMM) [5, 11], matrix factorization (MF) [2], periodic mobility models [1, 9], and recurrent neural networks (RNN) [4]. Unfortunately, these models fall short in capturing the semantics of user activities. They either do not consider textual information at all [2, 4, 5]; or assume the keywords of a latent state follows a multinomial distribution [11], which could suffer from text sparsity severely.

Contributions. We propose a semantics-enriched recurrent model (abbreviated as SERM onwards) for the next location prediction problem in semantic trajectories. Compared with previous methods, SERM models spatiotemporal regularities, activity semantics,

CIKM'17, November 6–10, 2017, Singapore.
 ISBN 978-1-4503-4918-5/17/11…$15.00
DOI: http://dx.doi.org/10.1145/3132847.3133056

and user preferences in a unified way. It achieves this by jointly learning: 1) the embeddings of multiple factors (user, location, time, keyword); and 2) the transition parameters of a recurrent neural network. It is thus capable of capturing semantics-aware spatiotemporal regularities underlying people's movements to predict next locations more accurately.

Our contributions are summarized as follows: (1) Different from next location prediction for GPS trajectories, we study the next location prediction problem for semantic trajectories, which introduces new challenges and opportunities. (2) We propose a semantics-enriched recurrent model that jointly learns the embeddings of different factors and the transition parameters of a recurrent neural network. (3) With experiments on two real-life semantic trajectory datasets, we demonstrate that SERM achieves significant improvements over existing methods.

2 PROBLEM DEFINITION

Consider a set of locations $\mathscr{L} = \{l_1, l_2, ..., l_M\}$. The locations could be either points-of-interests (POIs) or equally-sized spatial grids. Given a set of users $\mathscr{U} = \{u_1, u_2, ..., u_N\}$, we define the location sequence for each user $u_i \in \mathscr{U}$ as follows.

Definition 2.1 (Location Sequence). The location sequence for a user u_i is a time-ordered sequence $S(u_i) = \{r_1(u_i), r_2(u_i), ..., r_S(u_i)\}$. Each record $r_k(u_i) \in S(u_i)$ is a tuple (t_k, l_k, c_k), where: (1) t_k is the timestamp; (2) $l_k \in \mathscr{L}$ is the location of user u_i at time t_k; and (3) c_k is a text message describing the activity of user u_i.

Note that, in the raw location sequence $S(u_i)$, the time gap between two consecutive records could be large. As such, two records may have little correlation even if they are adjacent in the raw sequence. To address this problem, we impose a constraint on the time gap and define semantic trajectories below.

Definition 2.2 (Semantic Trajectory). A semantic trajectory for user u_i is a location sequence $T(u_i) = \{r_1(u_i), r_2(u_i), ..., r_K(u_i)\}$ *s.t.* $\forall 1 \le k < K, 0 < t_{k+1} - t_k \le \Delta t$, where $\Delta t > 0$ is a pre-defined time gap constraint.

With the time gap constraint Δt, we can segment every raw location sequence into a set of semantic trajectories. After the segmentation, the input location sequences are organized into a set of Z semantic trajectories $\mathscr{T} = \{T_1(u), T_2(u), \ldots, T_Z(u)\}$. Note that the trajectories in $\mathscr{T}$ can have different lengths.

We are now ready to formulate the next location prediction problem in semantic trajectories. Given a semantic trajectory $T_i = \{r_1(u_i), r_2(u_i), ..., r_K(u_i)\}$, the task of next location prediction is to predict the ground-truth location l_K from the location set $\mathscr{L}$, based on the preceding sequence $\{r_1(u_i), r_2(u_i), ..., r_{K-1}(u_i)\}$.

3 THE SERM MODEL

3.1 The Overall Architecture

SERM models the generation of the given semantic trajectories with recurrent neural networks. During the modeling process, we need to consider multiple factors for next location. First, a user's movement exhibits strong *spatiotemporal regularity* — the current location and time slot can have strong influence in deciding the next location. Second, the *semantics* of a user's current activity serves as a useful signal for next location prediction. Finally, the user's *personal preferences* often play an important role in next location prediction.

To encode the above observations, we design the architecture of SERM in Figure 1. At a high level, SERM models the trajectory generation process by integrating the embeddings of different factors into a many-to-many recurrent neural net.

As shown in Figure 1, SERM consists of three layers: the embedding layer, the recurrent layer, and the output layer. In the remainder of this section, we present the details of different layers and the learning procedure for the parameters.

3.2 The Embedding Layer

Consider the trajectory $T_v(u_i) = \{r_1(u_i), r_2(u_i), ..., r_K(u_i)\}$ for a given user u_i. The embedding layer is designed to capture the information of the dynamic factors in each record $r_k(u_i) = (t_k, l_k, c_k)$. As shown in the right part of Figure 1, we learn the embeddings for the location l_k, the timestamp t_k, and the content c_k, then concatenate them into a vector e_k to encode the information contained in $r_k(u_i)$. We detail the embedding of each factor in the following.

3.2.1 Timestamp Embedding. The original temporal information in each record is a real-value timestamp t_k. However, it is infeasible to embed every timestamp because time is continuous. We thus map one week into 48 slots (24 slots for weekdays and 24 slots for weekends) and consider each hour as a basic embedding unit. For any raw input timestamp t_k, we transform t_k into a one-hot 48-dimensional vector representation. The time embedding layer attempts to learn a $D_t \times 48$ transformation matrix E_t, where D_t is the dimension for the embedding. With the matrix E_t, we can transform any one-hot input time vector t_k into a D_t-dimensional vector e_{t_k} based on the following equation: $e_{t_k} = E_t \cdot t_k$.

3.2.2 Location Embedding. Each raw location l_k is represented as a M-dimensional one-hot vector, where the non-zero entry denotes the index for the corresponding location in $\mathscr{L} = \{l_1, l_2, ..., l_M\}$. The location embedding module aims to learn a $D_l \times M$ transformation matrix E_l, such that every location l_k can be transformed into a D_l-dimensional embedding vector e_{l_k} according to the following equation: $e_{l_k} = E_l \cdot l_k$.

3.2.3 Content Embedding. For each record $r_k(u_i)$, the content c_k is a V-dimensional vector representing a bag of keywords from a V-dimensional vocabulary. To embed the content, we define a $D_c \times V$ transformation matrix E_v. Then we transform the original content c_k into a D_c-dimensional vector e_{c_k} base on: $e_{c_k} = E_v \cdot c_k$. The transformation amounts to looking up the embeddings of all the keywords in c_k, and then take the sum of them to encode the semantics in the content. During training time, we use the pre-trained GloVe word vectors [7] to initialize the matrix E_v, and then fine-tune the parameters to adapt the embeddings for the next location prediction task.

3.3 The Recurrent Layer

For each record $r_k(u_i) = (t_k, l_k, c_k)$ in the input trajectory, the embedding modules are capable of generating vector representations for t_k, l_k, and c_k. By concatenating their embeddings, we obtain

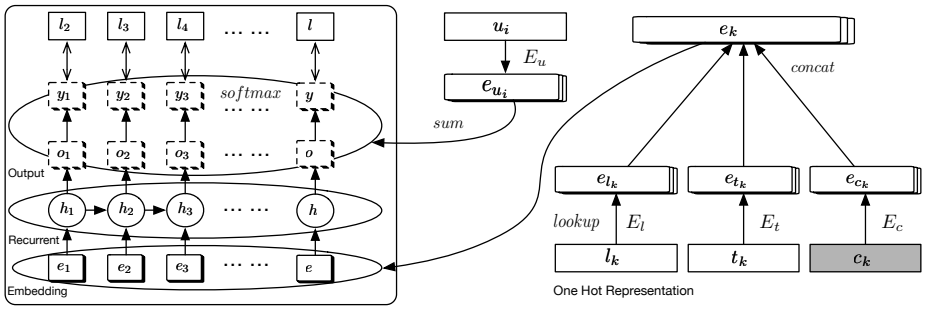

Figure 1: The architecture of the semantics-enriched recurrent model.

the vectorized representations $e_k \in R^{D_e}$ for the record (t_k, l_k, c_k) where $D_e = D_t + D_l + D_c$.

For a length-K semantic trajectory, the recurrent layer is comprised of K recurrent units. The k-th unit takes the embedding of the record (t_k, l_k, c_k) as its input, and then computes the hidden state as:

$$h_k = f(W \cdot h_{k-1} + G \cdot e_k + b).$$

In the above, $h_k \in R^{D_h}$ is a D_h-dimensional vector that represents the hidden state of the recurrent unit. It is computed by first combining the previous hidden state h_{k-1}, the embedding of the k-th record e_k, and the bias term b; and then performing a non-linear transformation $f(\cdot)$. The involved parameters are: the $D_h \times D_h$ matrix W, the $D_h \times D_e$ matrix G, the D_h-dimensional vector b, and the transformation function $f(\cdot)$. Here, $f(\cdot)$ is instantiated as the transformation function of *Long Short-Term Memory* (LSTM), which can easily handle different-length sequences by sharing the parameters of f at different time steps.

3.4 Output Layer

The hidden state h_k encodes the information observed until step k. Now we further integrate it with the users' personal preferences to output the prediction. To this end, we first transform h_k into a D_M-dimensional (M is the total number of locations) vector o_k, and then combine it with user embedding as follows:

$$o_k = H \cdot h_k + a \tag{1}$$

$$e_{u_i} = E_u \cdot u_i \tag{2}$$

$$o_k^{'} = e_{u_i} + o_k \tag{3}$$

The term H is a $M \times D_h$ matrix and the term a is a M-dimensional vector. They are introduced to map the original D_h-dimensional hidden state h_k into the M-dimensional vector o_k. Furthermore, with a $M \times N$ transformation matrix E_u, we also map the one-hot user representation u_i into a M-dimensional vector e_{u_i}. Finally, we sum the latent state representation o_k and the user embedding e_{u_i}, and derive the distribution over the M locations with the softmax function, namely: $y_k = \text{softmax}(o_k^{'})$.

It is worth mentioning that, we fuse the user personal preference with other dynamic factors in the output layer instead of treating one-hot user representations as input. This is because unlike location, time, and content, user preferences are static features that do not change with different steps. By including them at a later stage, we avoid repeating them in the input sequences. We have empirically compared those two different strategies in our experiments, and found that the late-fusion strategy leads to much better performance.

3.5 Parameter Learning

Based on the SERM model described above, we are now capable of generating a probability distribution over all the locations at each step. To train the SERM model and infer the parameters, we use cross entropy as the loss function for SERM. Given a training set with Z samples, we define the objective function as:

$$J = -\sum_{i=1}^{Z} \sum_{k=1}^{K_i - 1} l_{k+1} \log(y_k) + \frac{\lambda}{2} ||\Theta||^2$$

where $\Theta = \{E_t, E_l, E_c, E_u, W, G, H, b, a\}$ denotes all the parameters to be estimated, and λ is a pre-defined constant for the regularization term to avoid overfitting. To optimize the above objective function, we use Stochastic Gradient Descent and the *Back Propagation Through Time* algorithm to learn the parameter set Θ.

4 EXPERIMENT

4.1 Experimental Settings

4.1.1 Datasets. Our experiments are based on two semantic trajectory datasets in two cities: New York City and Los Angeles. The fist dataset [10], referred to as NY, consists of 0.3 million Foursquare check-ins from 2011-01 to 2012-01. The second dataset, referred to as LA, consist of 1.4 million tweets from 2014-08 to 2014-11 [11]. For LA, we discretize the whole city into 500m×500m grids and consider each grid as a location. For each data set, we first extract the location sequences for different users. Then we remove the users that have fewer than 50 records and segment the sequences into semantic trajectories with a time gap constraint $\Delta t = 10h$. Furthermore, we remove the trajectories with lengths smaller than three. After such preprocessing, we obtain 3103 trajectories from 235 users in NY and 7826 trajectories from 244 users in LA. [1]

4.1.2 Compared Methods. We compare SERM with the following methods: (1) **Nearest Locations (NL):** It chooses the nearest neighbors to the user's current location as predictions. (2) **MF:** [2] MF casts the next location prediction problem as a recommendation problem. Based on the observed user-location matrix, it learns low-dimensional vectors for users and locations, and recommend the most similar locations for the user's next visit. (3) **HMM:** [5] It learns a Hidden Markov Model to characterize the movement

[1] Code and data available at https://github.com/yaodi833/serm.git

regularities, and uses the learned model to choose the location with the largest probability for next location prediction. (4) **ST-RNN:** [4] It is the state-of-the-art method for next location prediction. It is also based on recurrent neural networks, but the focus is on modeling spatiotemporal transition matrices instead of learning the embeddings of different factors. (5) **SERM*:** This is a variant of SERM, which only models location, time, and user factors without using textual information.

4.1.3 Experimental Protocol. For each data set, we randomly select 80% trajectories as the training data, and use the remaining 20% for testing. We use two different metrics to evaluate the performance of different methods. The first is the hitting ratio @k, which examines whether the ground-truth location appears in the top-k result list. For the test trajectories, the hitting ratio measures the percentage of trajectories for which the ground-truth location is successfully recovered by the top-k result list. The second metric is the geographical distance, which computes the distance between the ground-truth location and the top-5 prediction.

4.1.4 Parameter Settings. The key parameters in SERM include: (1) the embedding dimension for time, location, and content, namely D_t, D_l, and D_c; (2) the dimension D_h for the hidden state. We have tuned all these parameters in the range $[10, 100]$. Due to the space limit, we omit the detailed results for different parameter settings. In general, the performance of SERM increases with them and gradually stabilizes when D_t, D_l, D_c and D_h are large enough. We finally set $D_t = D_l = D_c = D_h = 50$ when comparing SERM with other methods. For the compared methods, we tune their parameters to obtain the best performance on our data sets.

4.2 Performance Comparison

Table 1 reports the performance of different methods on the two data sets. All the experiments are repeated for five times and the average performance is reported.

Table 1: Performance comparison for different methods. HR is the hitting ratio; δ_d is the predictor error in distance.

Data	Method	HR@1	HR@5	HR@10	HR@20	δ_d/m
NY	NL	0.1630	0.2455	0.2998	0.4386	2903
	MF	0.1690	0.4326	0.5013	0.5358	1963
	HMM	0.1763	0.4298	0.5251	0.5518	1952
	ST-RNN	0.1942	0.4421	0.5381	0.6053	1602
	SERM*	0.2181	0.4398	0.5401	0.6107	1563
	SERM	**0.2535**	**0.4507**	**0.5433**	**0.6237**	**1457**
LA	NL	0.3745	0.4516	0.4704	0.4911	6061
	MF	0.3646	0.5810	0.6354	0.6877	2647
	HMM	0.3921	0.5935	0.6331	0.6732	2521
	ST-RNN	0.4311	0.6013	0.6521	0.6980	2384
	SERM*	0.4452	0.6147	0.6590	0.6973	2377
	SERM	**0.4625**	**0.6265**	**0.6670**	**0.7026**	**2177**

As shown, on both data sets, SERM and its variant SERM* outperform NL, MF, and HMM significantly. The comparisons with these methods are discussed as follows: (1) The NL method is an intuitive and straightforward baseline and works reasonably well. Nevertheless, people's movements are much more complex than the nearest-first principle. (2) The MF method does not achieve good performance for the next location prediction task. This phenomenon is expected. Although MF can successfully discover the locations a user is interested in, it fails to capture the sequential transition regularity, which is important for next location prediction. (3) For HMM, it is outperformed by RNN-based methods (ST-RNN, SERM*, SERM). The reason is two-fold. First, it relies on distribution assumptions for user behaviors. Second, it only models first-order dependency for people's movements, while RNN-based models can capture long-term dependencies.

ST-RNN turns out to be the strongest baseline, yet it is still inferior to SERM* and SERM. Compared to ST-RNN, the advantage of SERM mainly lies in its model design. To capture spatiotemporal dynamics, ST-RNN partitions the space and time into bins, and learns a transition matrix for every temporal and spatial bin. In contrast, SERM only involves one transition matrix and captures the spatiotemporal dynamics by embedding different locations and hours into low-dimensional vector spaces. Such a strategy largely reduces the parameters involved in SERM, and generates more reliable prediction models.

Finally, comparing the performance of SERM* and SERM, one can observe that SERM achieves considerable improvements over SERM*. By including textual information into the modeling process, SERM can capture the intentions of user activities more accurately, which enables SERM to perform semantics-aware next location prediction and achieve better performance.

5 CONCLUSIONS

We have proposed a novel recurrent model for semantics-aware next location prediction. By jointly learning the embeddings of multiple factors (user, location, time, keyword) and the transition parameters of a recurrent neural network, our proposed model can capture different contexts underlying user movements and enable semantics-aware location prediction. We have evaluated our model on two real-life datasets, and the results show our model achieves significant improvements over state-of-the-art methods.

REFERENCES

[1] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. In *KDD*, pages 1082–1090, 2011.

[2] N. Duong-Trung, N. Schilling, and L. Schmidt-Thieme. Near real-time geolocation prediction in twitter streams via matrix factorization based regression. In *CIKM*, pages 1973–1976, 2016.

[3] Z. Li, B. Ding, J. Han, R. Kays, and P. Nye. Mining periodic behaviors for moving objects. In *KDD*, pages 1099–1108, 2010.

[4] Q. Liu and *et al.* Predicting the next location: A recurrent model with spatial and temporal contexts. In *AAAI*, pages 194–200, 2016.

[5] W. Mathew, R. Raposo, and B. Martins. Predicting future locations with hidden markov models. In *UbiComp*, pages 911–918, 2012.

[6] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *KDD*, pages 637–646, 2009.

[7] J. Pennington and *et al.* Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543, 2014.

[8] J. Rao, H. He, H. Zhang, F. Ture, R. Sequiera, S. Mohammed, and J. Lin. Integrating lexical and temporal signals in neural ranking models for searching social media streams. In *Neu-IR*, 2017.

[9] Q. Yuan, W. Zhang, C. Zhang, X. Geng, G. Cong, and J. Han. PRED: periodic region detection for mobility modeling of social media users. In *WSDM*, pages 263–272, 2017.

[10] C. Zhang, J. Han, L. Shou, J. Lu, and T. F. L. Porta. Splitter: Mining fine-grained sequential patterns in semantic trajectories. *PVLDB*, 7(9):769–780, 2014.

[11] C. Zhang, K. Zhang, Q. Yuan, L. Zhang, T. Hanratty, and J. Han. Gmove: Group-level mobility modeling using geo-tagged social media. In *KDD*, pages 1305–1314. ACM, 2016.