Time-sensitive POI Recommendation by Tensor Completion with Side Information

Bo Hui
Auburn University
bohui@auburn.edu

Da Yan
University of Alabama at Birmingham
yanda@uab.edu

Haiquan Chen
California State University, Sacramento
haiquan.chen@csus.edu

Wei-Shinn Ku
Auburn University
weishinn@auburn.edu

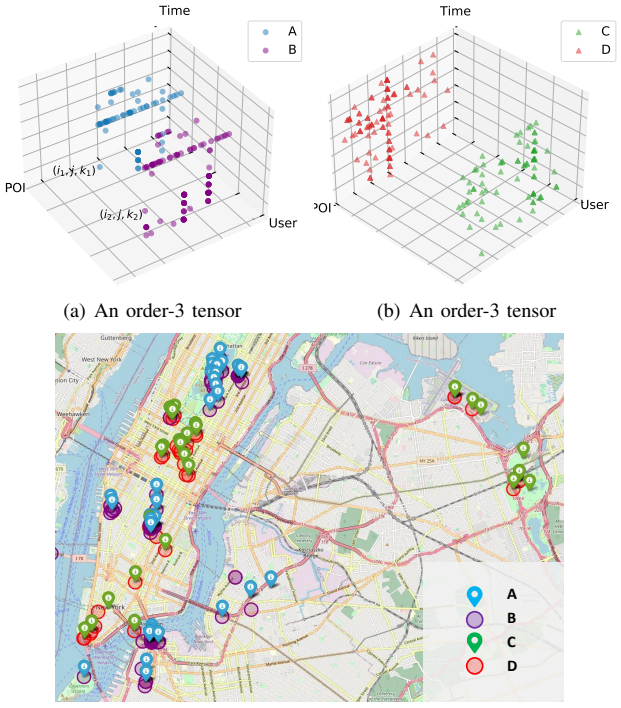
***Abstract*—Context has been recognized as an important factor to consider in personalized recommender systems. Particularly in location-based services (LBSs), a fundamental task is to recommend to a mobile user where he/she could be interested to visit next at the right time. Additionally, location-based social networks (LBSNs) allow users to share location-embedded information with friends who often co-occur in the same or nearby points-of-interest (POIs) or share similar POI visiting histories, due to the social homophily theory and Tobler's first law of geography. So, both the time information and LBSN friendship relations should be utilized for POI recommendation.**

Tensor completion has recently gained some attention in time-aware recommender systems. The problem decomposes a user-item-time tensor into low-rank embedding matrices of users, items and times using its observed entries, so that the underlying low-rank subspace structure can be tracked to fill the missing entries for time-aware recommendation. However, these tensor completion methods ignore the social-spatial context information available in LBSNs, which is important for POI recommendation since people tend to share their preferences with their friends, and near things are more related than distant things. In this paper, we utilize the side information of social networks and POI locations to enhance the tensor completion model paradigm for more effective time-aware POI recommendation. Specifically, we propose a regularization loss head based on a novel *social Hausdorff distance* function to optimize the reconstructed tensor. We also quantify the popularity of different POIs with location entropy to prevent very popular POIs from being over-represented hence suppressing the appearance of other more diverse POIs. To address the sensitivity of negative sampling, we train the model on the whole data by treating all unlabeled entries in the observed tensor as negative, and rewriting the loss function in a smart way to reduce the computational cost. Through extensive experiments on real datasets, we demonstrate the superiority of our model over state-of-the-art tensor completion methods.

***Index Terms*—POI, tensor completion, recommender system, social-spatial**

I. Introduction

Context has been recognized as an important factor to consider in personalized recommender systems [28]. With the prosperity of location-based social network (LBSN) services such as Foursquare and Yelp, a large number of check-ins have been collected by the LBSN companies and utilized for POI recommendations. The time dimension has always been a key context factor in such recommendations [32], since users' visiting preferences are highly time sensitive. For example, the season plays an important factor in where people visit; hot pot restaurants are the most crowded in the winter season, while holiday hotspots can transition from aquatics centers in summer to ski resorts in winter.



(a) An order-3 tensor (b) An order-3 tensor

(c) User friendship pairs (A-B and C-D) and their visited POIs

Fig. 1. Low-Rank Tensor Completion with Time and Social-Spatial Context

Since such time-awareness cannot be captured by a conventional collaborative filtering approach such as low-rank matrix completion over a low-rank user-item rating/purchase matrix, we adopt a three-dimensional user-POI-time tensor to effectively represent the check-in events. Figure 1(a) (resp. Figure 1(b)) shows the check-ins by User A and User B (resp. User C and User D) who are friends, in an order-3 user-POI-time tensor. The check-ins are obtained from the Gowalla dataset used in our experiments. We can observe that friends tend to visit similar POIs, such as the two markers at (i_1, j, k_1) and (i_2, j, k_2) which indicate that both A and B visited POI j albeit at different time slots k_1 and k_2, respectively.

Tensor completion is a popular approach to fill the missing entries of a partially observed tensor in recommender systems [42], based on the fact that tensors in real applications often exhibit a low-rank property. The approach reconstructs

a partially observed tensor through multilinear multiplication of latent factors, such as CP (CANDECOMP/PARAFAC) and Tucker factorizations [30]. However, real-world tensors in recommender systems often exhibit complex non-linear factor interactions, so recent works CoSTCo [32] and NTM [10] combine deep neural networks with tensor algebra in order to capture the nonlinear interactions among the tensor factors.

While these works can utilize the time dimension as a factor, they do not utilize the social-spatial information present in LBSNs, which is a very important factor when users decide where to visit. For example, we tend to turn to our friends for recommendations of nice restaurants, shopping stores and/or maintenance/repair services in our daily life, which aligns well with he social homophily theory [36]. Moreover, people tend to visit places that they are familiar with most of the time, so the visited POIs tend to form localized clusters which aligns well with Tobler's first law of geography [43]. As a result of these observations, we can conclude that two users tend to check in the same or nearby places if they are friends.

We have verified this observation using real data and we found it to be generally true and robust. As an illustration, Figure 1(c) plots the check-in locations of the two pairs of friends (A-B and C-D) on a map. We can see that the check-in locations of Users A and B (blue markers and purple circles) are highly overlapped, and so are the check-in locations of Users C and D (green markers and red circles). This shows that friends' check-ins tend to colocate more than non-friends' check-ins (e.g., those of Users A and C).

Social network information is proven to benefit the performance of recommendation [35] and the locations information can also improve the performance of POI recommendation [21]. In this paper, we aim to leverage the social-spatial information in LBSN as the side information to improve neural tensor completion based POI recommendation.

Our neural tensor completion model associates each user, POI and time unit with an embedding vector to learn, so that the value for each entry (i, j, t) in the tensor can be recovered from the embedding vectors of User i, POI j and time unit t. Most existing tensor completion models initialize the embeddings randomly or with one-hot encoding, especially when the content information of users and items (POI in our case) is not available due to privacy reasons. However, careful initialization is critical in various tensor problems, particularly for neural tensor models where gradient descent could get trapped in undesirable stationary points (e.g., a saddle point) if it starts from an arbitrary point. We, therefore, adopt a spectral method to obtain rough estimates of the initial user, POI and time embeddings for embedding initialization.

Our neural tensor completion model is trained to minimize the least-squares error between the reconstructed tensor and the original tensor. Generally, only positive data (check-ins) are observed in a POI recommender system. To train the model additionally with negative data, existing deep learning methods use the strategy of negative sampling to generate the negative samples, the performance of which is highly sensitive to the sampling strategy and the number of negative samples [8]. In particular, negative sampling is a biased approximation and often does not converge to the same loss as computed from all entries, making it difficult to converge to the optimal ranking performance regardless of how many update steps have been taken. Inspired by [9] which solves matrix completion without negative sampling, we train our neural tensor model with the whole data including all unlabeled data to combat the drawback of negative sampling; to address the high computational cost of directly computing a least-squares loss over all the tensor entries, we rewrite this loss function in a smart way to allow more efficient calculation and learning.

A key contribution of our model design is a backpropagatable formulation of the social Hausdorff distance function. The Hausdorff distance metric measures how far two sets of points are from each other. In our model design, we want to minimize the Hausdorff distance between two sets of check-in locations from each pair of friends to enforce the social homophily theory, but several challenges exist. Firstly, the inputs to the original Hausdorff distance metric are locations, not the outputs of our neural tensor model (i.e., the probability of each user-POI-time interaction). Secondly, even if we revise our social Hausdorff distance function to admit user-POI-time interaction probabilities, the minimization operator $min(\cdot)$ in the original Hausdorff distance function is not a smooth function with respect to its inputs, so it does not support backpropagation. We, therefore, need to modify this operator to support learning with backpropagation. The proposed social Hausdorff distance function enables our model to integrate both the social relations and POI locations to regularize the tensor completion formulation. To boost the diversity of recommendation, we leverage location entropy [13] to allow those less frequently visited POIs to be recommended, considering that they often better reveal the real user social strength than those places that everyone visits.

We hereby summarize our major contributions as follows:

- We present a backpropagatable formulation of the social Hausdorff distance-based loss function which enables our model to leverage both social graph and POI locations to regularize the tensor completion formulation.
- We integrate the social-spatial loss head with location entropy POI weights to improve recommendation diversity.
- We train our model with the whole data including all unlabeded data to combat the drawback of negative sampling in tensor completion, and rewrite the least-squares loss smartly to reduce the time complexity.
- We carefully initialize the latent user/POI/time embeddings with a spectral method to avoid being trapped in stationary points during training.

The rest of this paper is organized as follows. Section II reviews the related works on tensor completion. In Section III, we define our problem and overview our model framework. Section IV describes our technical details. In Section V, we present experiments that verify the superiority of our model. Finally, we conclude our paper in Section VI.

II. Related works

In this section, we review the formulations of tensor completion, and recent related works on neural tensor completion models and POI recommendation methods.

Tensor Factorization and Completion. Tensor as a data structure has been widely employed in the database and data mining community to represent multi-dimensional objects. In many application scenarios (e.g., recommender systems, knowledge base completion, and temporal health data analysis), only a sample of tensor entries are revealed to users, and the goal is to infer the values of all missing entries. A common prior knowledge leveraged for tensor completion is the low-rank tensor structure [4], which assumes that the unknown ground-truth tensor can be factorized into a few low-rank embedding matrices. CP (CANDECOMP/PARAFAC) [17] and Tucker decomposition [44] are two classical tensor factorization models, which can be considered as higher order generalizations of the matrix decomposition (e.g., singular value decomposition, or SVD). Without loss of generality, we consider 3D tensors. Formally, a rank-r CP model factorizes an order-3 tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ into three factor matrices: $\mathbf{U}^1 \in \mathbb{R}^{r\times I}$, $\mathbf{U}^2 \in \mathbb{R}^{r\times J}$ and $\mathbf{U}^3 \in \mathbb{R}^{r\times K}$, such that a tensor entry can be predicted as:

$$\hat{\mathcal{X}}_{i,j,k} = \sum_{t=1}^{r} {\mathbf{U}^1}_{t,i}{\mathbf{U}^2}_{t,j}{\mathbf{U}^3}_{t,k}. \quad (1)$$

In contrast, Tucker decomposition factorizes a tensor $\mathcal{X} \in \mathbb{R}^{I\times J\times K}$ into a core tensor $\mathcal{G} \in \mathbb{R}^{r_1\times r_2\times r_3}$ and three factor matrices such that:

$$\hat{\mathcal{X}}_{i,j,k} = \sum_{t_1=1}^{r_1}\sum_{t_2=1}^{r_2}\sum_{t_3=1}^{r_3} \mathcal{G}_{t_1,t_2,t_3}{\mathbf{U}^1}_{t_1,i}{\mathbf{U}^2}_{t_2,j}{\mathbf{U}^3}_{t_3,k}. \quad (2)$$

Based on CP and Tucker decomposition, many low-rank tensor factorization algorithms were developed. INDSCAL [6] is a special case of CP for three-way tensors that are symmetric in two modes. PARAFAC2 [16] is a variant of CP that can be applied to a collection of matrices. CANDELINC [7] is a CP variant with linear constraints. PARATUCK2 [18] groups the mode-1 objects and the mode-2 group into latent components.

Scalable Tensor Factorization and Completion. Among recent works, P-TUCKER [37] is a scalable Tucker factorization method for sparse tensors. SPALS [12] is a method to sample intermediate steps of alternating minimization algorithms for computing low-rank tensor CP decompositions. MAST [42] tracks the subspace of general incremental tensors for completion. GigaTensor [27] designs a scalable distributed algorithm for large-scale tensor decomposition. D-Tucker [26] compresses the tensor by computing randomized SVD.

To reconstruct a ground-truth tensor $\mathcal{X}$ based on a set, Ω, of observed entries at hand, the strategy of most existing works is to resort to the following least-squares formulation:

$$\text{minimize } f(\mathbf{U}^1, \mathbf{U}^2, \mathbf{U}^3) := \sum_{(i,j,k)\in\Omega} (\hat{\mathcal{X}}_{i,j,k} - \mathcal{X}_{i,j,k})^2. \quad (3)$$

Due to its highly nonconvex nature, however, solving this optimization problem exactly is computationally intractable. A line of polynomial-time approximation algorithms have been proposed, such as convex relaxation [41], sum of squares hierarchy [38], alternating minimization [25], vanilla gradient descent [4], and structure-aware proximal iterations [50].

Nonlinear and Neural Tensor Completion. While the above-mentioned works assume a multilinearity relationship between latent factors, recent works favor nonlinear factorization models to consider complex interactions in real-world. For example, in [19], the author introduces a kernel-based CP model to capture nonlinear relationships, and in [15], a Gaussian radial basis function is used for nonlinear tensor completion.

More recently, some works have proposed replacing the multilinear operations in tensor completion with multi-layer perceptrons (MLP), to utilize the non-linear activation layer of neural network models. CoSTCo [32] leverages the power of convolutional neural networks to model the interactions inside tensors. NTF [28] combines generalized CP and tensorized MLP to compute the tensor.

POI Recommendation. The problem of POI recommendation aims to predict where a user will visit next. The most common approach for recommendation is known as collaborative filtering [48]. Matrix completion [39] is used as a popular collaborative filter algorithm. These methods decompose the uer-POI matrix into two smaller matrices to discover the potential relationship between users and POIs. While the majority of matrix completion models apply an inner product on the latent factors, recent works tend to replace it with more complicated operation, such as neural network architecture [20]. Content-based filtering is another common approach for recommendation. Content-based recommender systems associate each user or POI with content information, such as location contexts [31] and spatial topics [21]. However, the performance of content-based methods relies on the quality of user and item features, and the content information is not always available due to privacy concerns. Graph neural networks [22], [24], [29] has also been used for recommendation [23]. TenInt [49] also formulates the recommendation as the tensor completion problem. There are several differences between TenInt and our model. First, TenInt does not integrate the spatial information (i.e., the distances between POIs) into tensor completion. It solves the problem by minimizing the squared loss regularized by the difference of user factors between each pair of friends. Thus, spatial information is not involved in their problem formulation. In contrast, our solution is able to leverage the spatial information. Instead of simply minimizing the difference of user factors between each pair of friends, we minimize the Hausdorff distance between check-ins of two users with friendship relations. Moreover, TenInt simply employs CP decomposition, while our model is a nonlinear factorization model that captures the complex interactions in the real world.

As a separate line of research different from the tensor completion formulation, many spatial-temporal models [52] have been developed to leverage both the location information and the temporal order of check-ins.

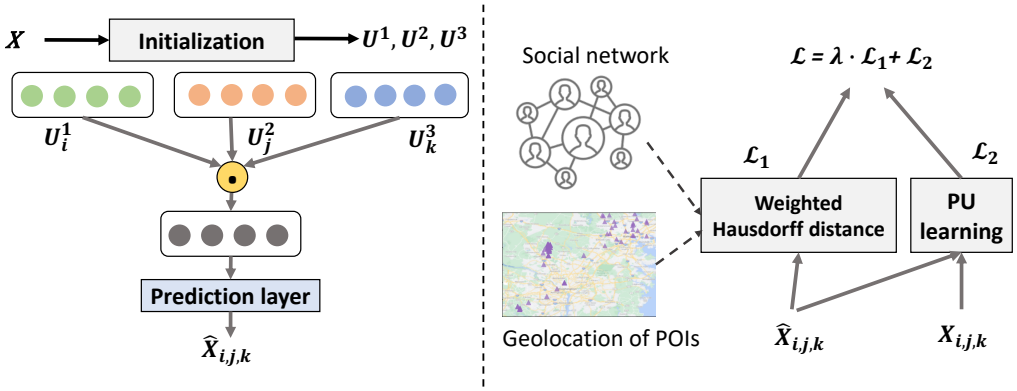


Fig. 2. The Model Framework

III. Preliminary

A. Problem Formulation

We use $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ to denote the check-in tensor of an LBSN, where I, J and K refer to the number of users, POIs, and time units (aka. intervals), respectively. The set $\Omega = \{(i,j,k) \,|\, \forall i \in \{1,\cdots,I\}, \forall j \in \{1,\cdots,J\}, \forall k \in \{1,\cdots,K\}\}$ contains indexes to all entries in $\mathcal{X}$. If i^{th} user checks in j^{th} POI at time interval k, then $\mathcal{X}_{i,j,k} = 1$ ($\mathcal{X}_{i,j,k}$ is an observed entry), otherwise $\mathcal{X}_{i,j,k} = 0$ (unlabeled). Let $\Omega_+ = \{(i,j,k) | \mathcal{X}_{i,j,k} = 1\}$ and $\Omega_- = \{(i,j,k) | \mathcal{X}_{i,j,k} = 0\}$ be the set of positive entries and unlabeled entries, respectively.

Given a social graph $G = (V, E)$, each node v_i corresponds to an LBSN user. If two users v_i and $v_{i'}$ are friends, there will be an edge $e_{i,i'}$ between v_i and $v_{i'}$. We use l_j to denote the location of j^{th} POI, which is a tuple of longitude and latitude. Then the problem is formulated as reconstructing tensor $\mathcal{X}$ as $\hat{\mathcal{X}}$ with a social graph $G = (V, E)$ and geolocations of POIs $\{l_j | \forall j \in \{1,\cdots,J\}\}$. The goal is to estimate the values of the unobserved entries $\hat{\mathcal{X}}_{i,j,k}$ as the scores among User i, POI j, and time interval k to recommend high-score entries.

B. Model Framework

Figure 2 overviews the framework of our model. Specifically, we first associate each user, POI and time interval with an embedding by a spectral initialization method. Then, the value of each entry is learned from corresponding embeddings by a neural network. The key novelty of our model is to use a hybrid loss function with two heads: (1) a weighted Hausdorff distance head to leverage the social-spatial information; and (2) the mean-squared error between the original tensor and reconstructed tensor on the whole data. We use the strategy of joint training to optimize the reconstructed tensor.

IV. Methodology

A. Embedding Initialization

Tensor factorization models usually use gradient descent (GD) as the optimization algorithm. However, GD could get trapped in undesirable stationary points (e.g., saddle point) if it starts from arbitrary points. Therefore, careful initialization is important in tensor factorization, and often necessary in order to achieve fast convergence [11]. In this paper, we use a spectral method to obtain rough estimates of the initial user, POI and time unit embeddings. Specifically, we first "unfold" the tensor $\mathcal{X}$ into three subspace matrices: the mode-1, mode-2 and mode-3 matricizations $\mathbf{A} \in \mathbb{R}^{I \times (JK)}, \mathbf{B} \in \mathbb{R}^{J \times (IK)}$ and $\mathbf{C} \in \mathbb{R}^{K \times (IJ)}$, respectively, where $\mathbf{A}_{i,(j-1)K+k} = \mathcal{X}_{i,j,k}$, $\mathbf{B}_{j,(i-1)K+k} = \mathcal{X}_{i,j,k}$, and $\mathbf{C}_{k,(i-1)I+j} = \mathcal{X}_{i,j,k}$. To estimate the rank-$r$ factors, a natural choice is to explore the principal subspace of $\mathbf{A}\mathbf{A}^T$, $\mathbf{B}\mathbf{B}^T$ and $\mathbf{C}\mathbf{C}^T$. Specifically, we zero out the diagonal entries of the three matrices, and then use the top-r eigenvectors of all off-diagonal matrices as the estimated factors in each subspace:

$$\begin{aligned} \mathbf{U}^1 &= eigen((\mathbf{A}\mathbf{A}^T)|_{\text{off-diag}}, r) \in \mathbb{R}^{I \times r} \\ \mathbf{U}^2 &= eigen((\mathbf{B}\mathbf{B}^T)|_{\text{off-diag}}, r) \in \mathbb{R}^{J \times r} \\ \mathbf{U}^3 &= eigen((\mathbf{C}\mathbf{C}^T)|_{\text{off-diag}}, r) \in \mathbb{R}^{K \times r}, \end{aligned} \tag{4}$$

where $|_{\text{off-diag}}$ extracts out the off-diagonal entries of a squared matrix, and $eigen(., r)$ extracts the top-r eigenvectors of the input matrix as the r columns of the output matrix. We zero out all diagonal entries because the diagonal entries bear too much influence on the principal directions and should be down-weighed [3]. Intuitively, we conduct PCA along one mode, by treating the values in the other two modes as features. The benefit of this initialization method is that the roughly estimated tensor factors can result in fast convergence, and we shall justify this in our experiments.

B. Tensor Factorization Formulation

We model the ternary interactions among users, POIs and time intervals, by computing the value of each entry $\mathcal{X}_{i,j,k}$ with the corresponding embedding vectors. Specifically, given the embedding vectors $\mathbf{U}_i^1$, $\mathbf{U}_j^2$ and $\mathbf{U}_k^3$ for User i, POI j and time interval k, we first compute a vector that equals their element-wise product:

$$\phi(\mathbf{U}_i^1, \mathbf{U}_j^2, \mathbf{U}_k^3) = \mathbf{U}_i^1 \odot \mathbf{U}_j^2 \odot \mathbf{U}_k^3, \tag{5}$$

where $\odot$ denotes the element-wise product of vectors. Note the similarity between Eq (5) and the CP formulation in Eq (1). To predict the value of $\mathcal{X}_{i,j,k}$, we pass the vector $\phi(\mathbf{U}_i^1, \mathbf{U}_j^2, \mathbf{U}_k^3)$ through a dense layer with parameter $\mathbf{h} \in \mathbb{R}^r$:

$$\hat{\mathcal{X}}_{i,j,k} = \mathbf{h}^T(\mathbf{U}_i^1 \odot \mathbf{U}_j^2 \odot \mathbf{U}_k^3) = \sum_{t=1}^{r} \mathbf{h}_t \mathbf{U}_{i,t}^1 \mathbf{U}_{j,t}^2 \mathbf{U}_{k,t}^3, \tag{6}$$

where $\mathbf{h}_t$ corresponds to the importance weight of the t^{th} factor dimension. Semantically, we treat $\hat{\mathcal{X}}_{i,j,k}$ as the probability that $X_{i,j,k} = 1$, i.e., User i visits POI j in time interval k.

We remark that the CP model is a special case of our formulation. Specifically, assume that $\mathbf{h}$ is a vector filled with all ones (i.e., $\mathbf{h} = [1, 1, ..., 1]^T \in \mathbb{R}^r$). Then Eq (6) becomes $\sum_{t=1}^{r} \mathbf{U}^1{}_{t,i}\mathbf{U}^2{}_{t,j}\mathbf{U}^3{}_{t,k}$, which is exactly Eq (1). Compared with CP, our model with learnable parameter $\mathbf{h}$ is more expressive and can capture more complicated interactions of multiple factors in recommender system.

To optimize the embeddings $\mathbf{U}^1$, $\mathbf{U}^2$, $\mathbf{U}^3$ and parameter $\mathbf{h}$, we may simply minimize the squared error $(\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2$ for observed entries. However this error head alone does not utilize the rich social-spatial information in an LBSN. We, therefore, design a hybrid loss function with two loss heads: one is the prediction error head computed over observed entries, and the other is a novel *social Hausdorff distance* function which measures how far friends' subsets of POIs are from each other, which we introduce next.

C. Social Hausdorff Distance

In our daily life, people frequently turn to their friends for recommendations of nice restaurants and other places to visit. According the social homophily theory, social networks tend to form clusters of nodes with similar properties or interests. In the meanwhile, Tobler's first law of geography states that near things are more related than distant things, implying that, for example, a user is more likely to have dinner in a restaurant visited and liked by his friends; moreover, if the user decides to go shopping after the dinner, he/she is more likely to check in at a nearby shopping mall.

Let us denote the i^{th} user's vertex in an LBSN by v_i, and we use i and v_i interchangeably in the following discussion. Formally, if a user $v_{i'}$ is a friend of another user v_i in an LBSN, then the set of $v_{i'}$'s check-ins tend to overlap with or at least close to v_i's check-ins. Therefore, it is natural to recommend a POI to a user if it is visited by his/her friends, or close to a POI visited by his/her friends. We thus add a regularization term to the loss function to enforce such a social-spatial cluster structure in our reconstructed tensor.

Hausdorff distance is metric to measure how far two point sets are from each other. Let us denote by $S(v_i)$ the potential POIs that will be visited by a user v_i, and denote by $\mathcal{N}(v_i)$ the set of POIs that were checked by v_i's friends:

$$S(v_i) = \{j \,|\, \exists k \in \{1, \cdots K\}, \hat{\mathcal{X}}_{i,j,k} > 0\}, \tag{7}$$

$$\mathcal{N}(v_i) = \{j \,|\, \forall (v_i, v_{i'}) \in E \colon \exists k \in \{1, \cdots K\}, \mathcal{X}_{i',j,k} = 1\} \tag{8}$$

Then, the average Hausdorff distance [1] (AHD) between $S(v_i)$ and $\mathcal{N}(v_i)$ is formulated as:

$$d_{AH}(S(v_i), \mathcal{N}(v_i)) = \frac{1}{|S(v_i)|} \sum_{j \in S(v_i)} \min_{j' \in \mathcal{N}(v_i)} d(j, j') + \frac{1}{|\mathcal{N}(v_i)|} \sum_{j' \in \mathcal{N}(v_i)} \min_{j \in S(v_i)} d(j, j'), \tag{9}$$

where $d(j, j')$ denotes the distance between POIs j and j'.

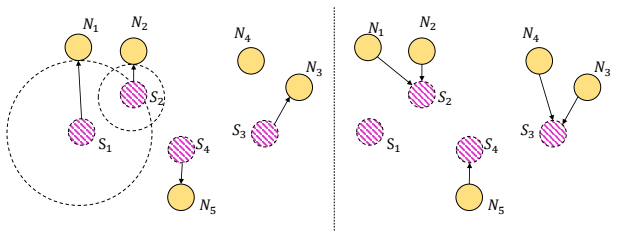


Fig. 3. Average Hausdorff Distance (Arrows Highlight Nearest Neighbors)

Intuitively, the AHD formulation above calibrates on average how far each point in $S(v_i)$ is to its closest point in $\mathcal{N}(v_i)$, and vice versa to make the distance measure symmetric.

Figure 3 illustrates AHD calculation. Specifically, the dashed pink circles represent those potential POIs that user v_i may be interested in and the yellow circles are the ground-truth POIs that are checked in by v_i's friends. For each dashed pink node $j \in S(v_i)$, $\min_{j' \in \mathcal{N}(v_i)} d(j, j')$ is the distance between j and its nearest neighbor in $\mathcal{N}(v_i)$. The first term in Eq (9) averages such distances for all $j \in S(v_i)$. Likewise, the second term averages such minimum distances for all $j' \in \mathcal{N}(v_i)$.

To regularize our loss function with social homophily and Tobler's first law of geography, we want to minimize the AHD between $S(v_i)$ and $\mathcal{N}(v_i)$. However, there are two challenges. First, recall from Eq (6) that our neural tensor model estimates the probability of each user-POI-time interaction, not the locations of POIs. The function must be learnable with respect to the output of the neural tensor model. The potential locations of $S(v_i)$ could be as large as the entire set of POIs in an LBSN, and we need to generalize the AHD definition to allow $S(v_i)$ to be set of locations with occurrence probabilities. Second, the minimization function $\min(\cdot)$ is not a smooth function with respect to its inputs, so it does not support backpropagation.

To address these two challenges, we devise a weighted Hausdorff loss function counterpart based on the idea of AHD:

$$d_{WH}(S(v_i), \mathcal{N}(v_i)) = \frac{1}{|A + \epsilon|} \sum_{j \in S(v_i)} p_{i,j} \min_{j' \in \mathcal{N}(v_i)} d(j, j') + \frac{1}{|\mathcal{N}(v_i)|} \sum_{j' \in N(v_i)} \underset{j \in S(v_i)}{M_\alpha} [p_{i,j} d(j, j') + (1 - p_{i,j}) d_{max}], \tag{10}$$

where $p_{i,j} = 1 - \prod_{k=0}^{K}(1 - \hat{\mathcal{X}}_{i,j,k})$ estimates the probability that User v_i will check in the j^{th} POI (in any of the K time intervals we consider). In the first term of Eq (10), $A = \sum_{j \in S(v_i)} p_{i,j}$ is the normalization factor for taking the weighted distance average (where weights are given by probabilities $p_{i,j}$), and the value of ϵ is set as 10^{-6} to avoid division by zero. The second term of Eq (10) approximates the second term of Eq (9), where $\min_{j \in S(v_i)} d(j, j')$ is now approximated by $\underset{j \in S(v_i)}{M_\alpha} [p_{i,j} d(j, j') + (1 - p_{i,j}) d_{max}]$ which allows the loss to backpropagate through $M_\alpha[.]$ into $p_{i,j}$ to tune the parameters in Eq (6), as we shall discuss in the next paragraph. In contrast, we do not apply $M_\alpha[\cdot]$ in the first term since $p_{i,j}$ (and hence $\hat{\mathcal{X}}_{i,j,k}$) is outside the $\min(.)$ function.

Here, $M_\alpha[x_1,\ldots,x_n] = (\frac{1}{n}\sum_{i=1}^{n} x_i^\alpha)^{\frac{1}{\alpha}}$ is the generalized mean, which equals $\min\{x_1,\ldots,x_n\}$ when $\alpha \to -\infty$. However, the more negative α is, the less smooth M_α becomes, and existing research found that $\alpha = -1$ is already sufficient to strike a good balance between its approximation quality to $\min(.)$ and the smoothness for effective backpropagation [40].

With the first term of Eq (10), we multiply the distance by $p_{i,j}$ to penalize the POI j with larger value of $p_{i,j}$ where there is no nearby POI checked by friends. In other words, if POI j is far from any of POIs that are checked by the friends of user v_i, $p_{i,j}$ tends to be 0.

Also in the second term of Eq (10), d_{max} corresponds to the maximum distance between any two POIs and it varies in different datasets. When we denote $f(j) = p_{i,j}d(j,j') + (1 - p_{i,j})d_{max}$, then we can see that the second term of Eq (10) approximates $\min_{j\in S(v_i)} d(j,j')$ with $\underset{j\in S(v_i)}{M_\alpha}[f(j)]$. Note that when $p_{i,j} \to 1$, $f(j) \to d(j,j')$ as desired; while when $p_{i,j} \to 0$, $f(j) \to d_{max}$ so the "soft" minimum $M_\alpha[.]$ tends to ignore j even if $d(j,j')$ is small, which makes sense since User v_i is unlikely to visit POI j. In other words, POIs with a low visit-probability by v_i near "friend" POIs $j' \in \mathcal{N}(v_i)$ will be penalized. While $f(.)$ is not the only function that enforces $f|_{p_{i,j}=1} = d(j,j')$ and $f|_{p_{i,j}=0} = d_{max}$, this linear function form is favored due to its numerical stability.

Note that in the deterministic scenario where $p_{i,j}$ is either 1 or 0, Eq (10) will become Eq (9) if we regard $M_\alpha[.]$ as $\min(.)$. Therefore, our loss function can be interpreted as average Hausdorff distance (AHD) extended with input uncertainty.

We remark that both terms in Eq (10) are necessary or we will get extreme results. Specifically, if the first term is removed, then $p_{i,j} = 1$ will always optimize $d_{WH}(S(v_i),\mathcal{N}(v_i))$ since $d(j,j') \leq d_{max}$. While if the second term is removed, $p_{i,j} = 0$ will always optimize $d_{WH}(S(v_i),\mathcal{N}(v_i))$ to be 0.

Diversifying Recommendation by Location Entropy. Recommending popular POIs already known by most users does not provide much additional information. Ideally, a user v_i would like to see recommended POIs that well match v_i's current interest but are not already visited by many people (but rather known by only a few who frequently pay visits). We measure the popularity of a POI using the concept of location entropy. Specifically, let $\Phi_{i,j} = \{\langle i,j,k\rangle \,|\mathcal{X}_{i,j,k} = 1\}$ be the set of all check-ins at POI j by a user i, and let $\Phi_j = \{\langle i,j,k\rangle \,|\mathcal{X}_{i,j,k} = 1\}$ be the set of check-ins at POI j by all users. Then the location entropy is defined as:

$$\mathcal{E}_j = -\sum_{i:\|\Phi_{i,j}|>0} \frac{|\Phi_{i,j}|}{|\Phi_j|} \log \frac{|\Phi_{i,j}|}{|\Phi_j|}. \tag{11}$$

A high value of $\mathcal{E}_j$ implies that POI j is visited by different users and thus known by many users, such as a local Costco wholesale warehouse. In this case, POI j plays a less important role in reflecting the social strength: two users who visit the same Costco warehouse are mostly likely not mutual friends; while in contrast, if two users show up in the same tennis court, they are more likely to be friends sharing the same hobby.

We thus further adjust $d(j,j')$ in the first (resp. second) term of Eq (9) by $e_j = exp(-\mathcal{E}_j)$ (resp. $e_{j'} = exp(-\mathcal{E}_{j'})$). Note that this weighting strategy naturally addresses the diversity of recommendation: a new French restaurant tends to have a higher weight in the learning process than Burger King, though it occurs sparsely in the check-in tensor. Combining the above location-entropy weighted distance with the input uncertainty, our final social Hausdorff distance for User v_i becomes:

$$d_{WH}(S(v_i),\mathcal{N}(v_i)) = \frac{1}{|A+\epsilon|}\sum_{j\in S(v_i)} p_{i,j}e_j \min_{j'\in\mathcal{N}(v_i)} d(j,j') + \frac{1}{|\mathcal{N}(v_i)|}\sum_{j'\in N(v_i)} e_{j'} \underset{j\in S(v_i)}{M_\alpha}[p_{i,j}d(j,j') + (1-p_{i,j})d_{max}], \tag{12}$$

Note that Eq (12) defines the social Hausdorff distance for just one user v_i. The final social Hausdorff loss function is defined as the sum of social Hausdorff distance for all users:

$$\mathcal{L}_1 = \sum_{v_i\in V} d_{WH}(S(v_i),\mathcal{N}(v_i)). \tag{13}$$

The next subsection describes how to combine this social-spatial loss with the least-squares regression problem in Eq (3).

D. Learning with Whole Data

Most existing tensor completion models are formulated to minimize the difference between the original data tensor $\mathcal{X}$ and the reconstructed tensor $\hat{\mathcal{X}}$. Since only the positive entries (check-in records in LBSNs) are observed, negative sampling is commonly used to generate negative entries for training. However, the performance of negative sampling is highly sensitive to the sampling strategy and the number of negative samples [8], [46]. In this paper, we propose to train our model on the whole data instead of by negative sampling. Then, the task is to minimize the mean-squared error of all entries:

$$\mathcal{L}_2 = \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K} w_{i,j,k}(\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2, \tag{14}$$

where $w_{i,j,k}$ is the entry weight for class balancing:

$$w_{i,j,k} = \begin{cases} w_+, & \text{if entry } (i,j,k)\in\Omega \text{ is positive} \\ w_-, & \text{if entry } (i,j,k)\in\Omega \text{ is unlabeled} \end{cases}$$

where we treat unlabeled entries simply as negative. Since the number of unlabeled entries is much larger than that of the positive entries, we set w_+ to be much larger than w_- so that a positive sample is as important as many unlabeled ones.

We remark that our method is a special case of negative sampling where all negative samples are sampled exactly once. Traditional negative sampling is a biased approximation and often does not converge to the same loss as computed from all entries [2], [47], making it difficult to converge to the optimal ranking due to the sensitivity to the sampling strategy. Our method is proposed to combat the drawback of negative sampling. Also, computing our new loss formulation naïvely as in the existing negative sampling scheme would lead to a prohibitive computational cost due to the large number of negative samples, and that is exactly what we address in this subsection. The time complexity of calculating Eq (14) is

$O(I \times J \times K)$. Considering that the numbers of users and POIs (i.e., I and J) are large in recommender systems, this time cost is intractable. We, therefore, rewrite this loss function as follows to make its evaluation affordable:

$$\begin{aligned}\mathcal{L}_2 &= \sum_{(i,j,k)\in\Omega_+} ((w_+ - w_-)\hat{\mathcal{X}}^2_{i,j,k} - 2\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}) \\ &+ w_- \sum_{r_1=1}^{r}\sum_{r_2=1}^{r} \mathbf{h}_{r_1}\mathbf{h}_{r_2}(\sum_{i=1}^{I}\mathbf{U}^1_{i,r_1}\mathbf{U}^1_{i,r_2})(\sum_{j=1}^{J}\mathbf{U}^2_{j,r_1}\mathbf{U}^2_{j,r_2}) \\ &(\sum_{k=1}^{K}\mathbf{U}^3_{k,r_1}U^3_{k,r_2}),\end{aligned} \tag{15}$$

where Ω_+ and Ω_- are the set of positives and unobserved entries in the train set, respectively. We next establish the equivalence of Eq (15) to Eq (14), and then explain how Eq (15) reduces the time cost.

Remark 1: Eq (15) is equivalent to Eq (14).

Proof: First, we factorize $(\mathcal{X}_{i,j,k} - \hat{\mathcal{X}}_{i,j,k})^2$ in Eq (14) and eliminate the constant term $\sum_{(i,j,k)\in\Omega}\mathcal{X}^2_{i,j,k}$:

$$\mathcal{L}_2 = \sum_{(i,j,k)\in\Omega} w_{i,j,k}\hat{\mathcal{X}}^2_{i,j,k} - 2\sum_{(i,j,k)\in\Omega} w_{i,j,k}\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k} \tag{16}$$

Note that the set of all entries Ω can be split into positive set Ω_+ and unlabeled set Ω_-, and $\mathcal{X}_{i,j,k}$ is 0 for entries in Ω_- since we treat unlabeled entries as negative as in negative sampling. Therefore, $\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}$ can also be eliminated for the unlabeled set Ω_- which gives:

$$\mathcal{L}_2 = \sum_{(i,j,k)\in\Omega} w_{i,j,k}\hat{\mathcal{X}}^2_{i,j,k} - 2\sum_{(i,j,k)\in\Omega_+} w_{i,j,k}\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k} \tag{17}$$

Now we replace $w_{i,j,k}$ with w_+ (resp. w_-) for entries in Ω_+ (resp. Ω_-):

$$\begin{aligned}\mathcal{L}_2 &= \sum_{(i,j,k)\in\Omega_+} w_+\hat{\mathcal{X}}^2_{i,j,k} + \sum_{(i,j,k)\in\Omega_-} w_-\hat{\mathcal{X}}^2_{i,j,k} \\ &\quad - 2\sum_{(i,j,k)\in\Omega_+} w_+\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k} \\ &= \sum_{(i,j,k)\in\Omega_+} ((w_+ - w_-)\hat{\mathcal{X}}^2_{i,j,k} - 2w_+\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}) \\ &\quad + \sum_{(i,j,k)\in\Omega_+} w_-\hat{\mathcal{X}}^2_{i,j,k} + \sum_{(i,j,k)\in\Omega_-} w_-\hat{\mathcal{X}}^2_{i,j,k} \\ &= \sum_{(i,j,k)\in\Omega_+} ((w_+ - w_-)\hat{\mathcal{X}}^2_{i,j,k} - 2w_+\mathcal{X}_{i,j,k}\hat{\mathcal{X}}_{i,j,k}) \\ &\quad + w_-\sum_{(i,j,k)\in\Omega}\hat{\mathcal{X}}^2_{i,j,k}\end{aligned} \tag{18}$$

In Eq (18), the first term and the second term can be regarded as the loss of the positive data and the whole data, respectively. Since the positive entries account for only a small portion of the whole data, the time complexity of the second term is the bottleneck of computation.

Recall from Eq (6) that $\hat{\mathcal{X}}_{i,j,k} = \sum_{t=1}^{r}\mathbf{h}_t\mathbf{U}^1_{i,t}\mathbf{U}^2_{j,t}\mathbf{U}^3_{k,t}$, so we rearrange $\sum_{(i,j,k)\in\Omega}\hat{\mathcal{X}}^2_{i,j,k}$ in the second term as:

$$\begin{aligned}&\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\Big((\sum_{r_1=1}^{r}\mathbf{h}_{r_1}\mathbf{U}^1_{i,r_1}\mathbf{U}^2_{j,r_1}\mathbf{U}^3_{k,r_1})(\sum_{r_2=1}^{r}\mathbf{h}_{r_2}\mathbf{U}^1_{i,r_2}\mathbf{U}^2_{j,r_2}\mathbf{U}^3_{k,r_2})\Big) \\ &= \sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}\Big(\sum_{r_1=1}^{r}\sum_{r_2=1}^{r}(\mathbf{h}_{r_1}\mathbf{U}^1_{i,r_1}\mathbf{U}^2_{j,r_1}\mathbf{U}^3_{k,r_1}\mathbf{h}_{r_2}\mathbf{U}^1_{i,r_2}\mathbf{U}^2_{j,r_2}\mathbf{U}^3_{k,r_2})\Big) \\ &= \sum_{r1=1}^{r}\sum_{r2=1}^{r}\mathbf{h}_{r_1}\mathbf{h}_{r_2}(\sum_{i=1}^{I}\mathbf{U}^1_{i,r_1}\mathbf{U}^1_{i,r_2})(\sum_{j=1}^{J}\mathbf{U}^2_{j,r_1}\mathbf{U}^2_{j,r_2})(\sum_{k=1}^{K}\mathbf{U}^3_{k,r_1}\mathbf{U}^3_{k,r_2}).\end{aligned} \tag{19}$$

This reduces the time cost of computing the second term from $O(I \times J \times K \times r)$ in Eq (14) to $O(r^2(I+J+K))$ here. Since $I, J, K \gg r$, the time cost of computing $\mathcal{L}_2$ now becomes much more tractable.

By replacing it into the second term in Eq (18), we obtain Eq (15). Note that this rewriting process is unique to our special case rather than general negative sampling, since the last term of Eq (18) sums over all entries in Ω, which translates to $\sum_{i=1}^{I}\sum_{j=1}^{J}\sum_{k=1}^{K}$ as in Eq (19), rather than a subset of sampled entries.

E. Loss Function

We combine the social-spatial loss head $\mathcal{L}_1$ and the least-squares loss head $\mathcal{L}_2$ into the final loss function $\mathcal{L}$ to train the model:

$$\mathcal{L} = \lambda\mathcal{L}_1 + \mathcal{L}_2 \tag{20}$$

where λ is a hyperparameter to adjust the importance of social Hausdorff distance loss. We use the strategy of joint training to optimize the parameters in our neural network model by directly minimizing the final loss $\mathcal{L}$. Note that a conventional tensor completion method only minimizes the difference between $\hat{\mathcal{X}}$ and $\mathcal{X}$, so it is not utilizing the social graph and POI locations.

V. Experiment

In this section, we report our comprehensive suite of experiments that answer the following questions regarding our model, named as TCSS (Tensor Completion with Social-Spatial regularization):

- Is it necessary to incorporate the time dimension for recommendation instead of considering just a user-POI interaction matrix?
- Can TCSS outperform existing state-of-the-art tensor completion models for time-aware recommendation?
- How does the time granularity along the time dimension influence the performance of recommendation?
- Is the performance consistent on different categories of POIs?
- Ablation study: how does training performance of TCSS compare with its counerparts using negative sampling and other initialization methods?
- How do the model hyperparameters influence the result quality?

A. Dataset

Three datasets are introduced in the experiment:

- **Gowalla**[1]**.** Gowalla was a worldwide location-based social network where users can check in at POIs. The dataset was collected from November 2010 to May 2011, including information such as users' friendship and check-in history. We only consider those users with at least 15 POI check-ins and at least one friend. We also filter out those POIs with fewer than 50 visitors. Locations in Gowalla are grouped into categories, and our preprocessing results in 6,392 shopping POIs, 5,667 entertainment POIs, 3,824 restaurant POIs and 2,272 outdoor

[1]https://www.yongliu.org/datasets/

POIs. The final preprocessed dataset contains 18,737 users and 1,666,455 check-ins in total.

- **Yelp**[2]**.** The original Yelp dataset contains 8,635,403 reviews for 160,585 worldwide businesses. We only consider those users that have at least 15 check-in records and 1 friendship relation. We filtered out those POIs with fewer than 50 visitors. The preprocessed data contains 718,214 check-ins from 17,534 users in 7,757 businesses.
- **Foursquare**[3]**.** This dataset includes global-scale check-in data collected from Foursquare during April 2012 to January 2014 (22 months). As before, we filtered out those users with fewer than 15 check-in POIs as well as those POIs with fewer than 50 visitors. The dataset also contains a user social network and we only consider those users with a least one friend. The preprocessed data contains 20,359 users, 5,777 POIs and 939,394 check-ins.
- **GMU-5K**[4]**.** This is a dense LBSN dataset generated by a simulator that simulates patterns of life. There are 11,189,377 check-ins at 8,901 POIs from 5,000 users. The density of the user-POI-time tensor is 3.21%.

B. Baselines

In order to verify the necessity of introducing the time dimension for POI recommendation, we implement two matrix-completion baselines that predict user-POI interactions:

- **MCCO**[5] **[5].** This work recovers the low-rank matrix from an incomplete set of entries. The matrix is recovered from multiplicative factors, and semidefinite programming is used to solve the convex relaxation of nuclear norm minimization.
- **PureSVD**[6] **[14].** The method treats all missing values as zeros, and performs conventional single value decomposition (SVD) to factorize a sparse user-item matrix as the product of a user orthonormal matrix, a diagonal matrix of singular values, and an item orthonormal matrix.

We further introduce 10 baselines, including 6 tensor completion methods, 3 spatiotemporal POI recommendation models that predict the user-POI-time interactions, and an algorithm that predicts user-POI interactions. We feed the historical check-ins as input to these baseline methods.

- **CP & Tucker**[7] **[30].** Eq (1) and Eq (2) define the tensor completion method of CP and Tucker decomposition, respectively. CP decomposes an order-3 tensor as a sum of three rank-one tensors, while the Tucker model factorizes an order-3 tensor into one core tensor along with three factor matrices.
- **NTM [10].** Nonlinear Tensor Machine also learns multi-aspect factors in recommender systems. It combines deep neural networks and tensor algebra to capture nonlinear interactions among multi-aspect factors.
- **NCF**[8] **[20].** Neural Collaborative Filtering is a neural network model that uses a multi-layer perceptron learn the nonlinear interaction relationship between the latent features. We follow NTM to feed the element-wise product of three MF vectors (user, POI, time) as the input of GMF Layer and concatenate three MLP vectors as the input of MLP Layer.
- **P-Tucker**[9] **[37]** is a scalable Tucker factorization method for sparse tensors. It performs alternating least squares with a row-wise update rule in a fully parallel way, which significantly reduces memory requirements for updating factor matrices.
- **CoSTCo**[10] **[32].** This work proposes a novel convolutional neural network (CNN) for tensor completion. It leverages the expressive power of CNN to model the complex interactions inside tensors and its parameter sharing scheme to preserve the desired low-rank structure.
- **STRNN [33].** This work extends recurrent neural networks (RNN) and proposes a Spatial Temporal RNN to model local temporal and spatial contexts for POI recommendation.
- **STAN [34].** This work proposes the Spatio-Temporal Attention Network (STAN) to exploit spatiotemporal information of all the checkins with self-attention layers along the trajectory.
- **STGN [52]** This work proposes the Spatio-Temporal Gated Network (STGN) by enhancing long-short term memory network, where spatio-temporal gates are introduced to capture the spatio-temporal relationships between successive checkins.
- **LFBCA [45]** This work proposes the location-friendship bookmark-coloring algorithm (LFBCA) to reconcile social interaction and location similarity in POI recommendation.

C. Performance Metrics

We adopt two widely used performance metrics that are designed to evaluate recommender systems: (1) hit ratio (Hit) [51] and (2) mean reciprocal rank (MRR) [51]. Given each entry (i, j, k) in the test set, we sample 100 random POIs $j_1, \ldots, j_{100}$ and predict the values of these entries (i, j_s, k), $s = 1, \ldots, 100$. We then sort the 100 values $\{\hat{\mathcal{X}}_{i,j_s,k}\}$ plus $\hat{\mathcal{X}}_{i,j,k}$ (i.e., 101 values in total) in non-increasing order.

Hit@10 counts the proportion of observed interactions (i, j, k) in the test set such that $\hat{\mathcal{X}}_{i,j,k}$ is within top-10 of the previous sorted list of length 101; while MRR averages the reciprocal ranks of $\hat{\mathcal{X}}_{i,j,k}$ in the sorted list over all interactions (i, j, k) in the test set, where the rank corresponds to the position of $\hat{\mathcal{X}}_{i,j,k}$ in the sorted list. We first average the reciprocal ranks of each user i along time dimension k, and then report their average over all users in the test set. On each dataset, we use 80% of check-ins as the observed tensor entries in $\mathcal{X}$ for training, and the remaining check-ins are used as the test set.

D. Model Configuration

In our experiments, we configure TCSS with the following default hyperparameters which were extensively tested and found to work consistently well. We set the weights w_+ and w_- as 0.99 and 0.01, respectively, by default. We use the default value $\lambda = 0.1$ as the weight of $\mathcal{L}_1$ in the loss computation. The length of embedding vectors is set as 10 by default, so we extract top-10 eigenvectors as the initialized embedding factors. For the time dimension, we set the granularity as month in a year (i.e., $k = 0, 1, \cdots, 11$). For example, if a check-in occurs in February, then $k = 1$. Recall Eq (12), where we set the default smoothing parameter α of the soft minimum function as -1, and set $\epsilon = 10^{-6}$ to avoid zero division. We use the Haversine formula[11] to calculate the distance between POIs considering that the POIs are distributed in a large area. All the parameters of baseline models are configured as described in the corresponding paper. We train our models using an Adam optimizer with a learning rate of 0.001 and a weight decay of 0.1. Our experiments were run on a machine equipped with a 2.20 GHz CPU, 26 GB RAM, and an NVIDIA Tesla P100 GPU. The source code of our model has been released at https://github.com/codingAndBS/tc-ss.git.

E. Model Comparison

Table I shows the performance comparison of our TCSS model with the selected representative baseline models. Note that we include not only various tensor completion approaches but also matrix completion methods. For matrix completion, we omit the time dimension and calculate Hit@10 and MRR based on the rank of entry (i, j) in

[2] https://www.yelp.com/dataset
[3] https://sites.google.com/site/yangdingqi/home/foursquare-dataset
[4] https://osf.io/e24th/wiki/home/
[5] https://github.com/JoonyoungYi/MCCO-numpy
[6] https://github.com/yoongi0428/RecSys_PyTorch
[7] http://tensorly.org/stable/user_guide/tensor_decomposition.html
[8] https://github.com/yihong-chen/neural-collaborative-filtering
[9] https://github.com/sejoonoh/P-Tucker
[10] https://github.com/USC-Melady/KDD19-CoSTCo
[11] https://pypi.org/project/haversine/

TABLE I
RESULTS COMPARISON

Model		Gowalla		Yelp		Foursquare		GMU-5K	
		Hit@10	MRR	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR
Matrix completion	MCCO	0.3309	0.1804	0.3951	0.1847	0.2880	0.1367	0.3321	0.2192
	PureSVD	0.3779	0.2107	0.4307	0.2087	0.5584	0.2856	0.6651	0.3772
POI recommendation	STRNN	0.3203	0.1908	0.3294	0.1363	0.2586	0.1288	0.4847	0.3481
	STAN	0.5239	0.3311	0.5112	0.3345	0.4723	0.3215	0.6328	0.2667
	STGN	0.5230	0.4127	0.4880	0.2485	0.5101	0.3185	0.4646	0.2810
	LFBCA	0.3513	0.2470	0.3945	0.1575	0.3541	0.1909	0.3828	0.1963
Tensor completion	CP	0.4544	0.2683	0.5522	0.2479	0.5156	0.2955	0.7072	0.4917
	Tucker	0.3742	0.2168	0.5902	0.2701	0.5585	0.3188	0.6989	0.4864
	P-Tucker	0.8162	0.3793	0.6255	0.2627	0.7843	0.3401	0.7473	0.3981
	NCF	0.7891	0.3453	0.6574	0.3057	0.8079	0.4279	0.8077	0.4629
	NTM	0.6181	0.3074	0.1719	0.0839	0.7699	0.3886	0.7897	0.3493
	CoSTCo	0.7571	0.2806	0.5565	0.2106	0.8336	0.3252	0.7465	0.4420
	TCSS	**0.9177**	**0.6206**	**0.7276**	**0.3408**	**0.9298**	**0.6133**	**0.9598**	**0.6376**

TABLE II
ABLATION STUDY

Model Variants	Gowalla		Yelp		Foursquare		GMU-5K	
	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR	Hit@10	MRR
Random initialization	0.8833	0.6107	0.6892	0.3275	0.9163	0.6095	0.9045	0.5196
One-hot initialization	0.8688	0.6036	0.6704	0.3061	0.8851	0.5613	0.8968	0.4926
Remove $\mathcal{L}_1$ ($\lambda = 0$)	0.8442	0.5763	0.6308	0.2896	0.8670	0.5115	0.8421	0.4854
Negative samping	0.8549	0.4098	0.5637	0.2218	0.8917	0.4348	0.9231	0.5550
Self-Hausdorff	0.8614	0.5858	0.6478	0.3029	0.8783	0.5354	0.8654	0.5071
Zero-out	0.8574	0.5571	0.6538	0.3064	0.8321	0.5248	0.8047	0.5012
Full-Fledged TCSS	**0.9177**	**0.6206**	**0.7276**	**0.3408**	**0.9298**	**0.6133**	**0.9598**	**0.6376**

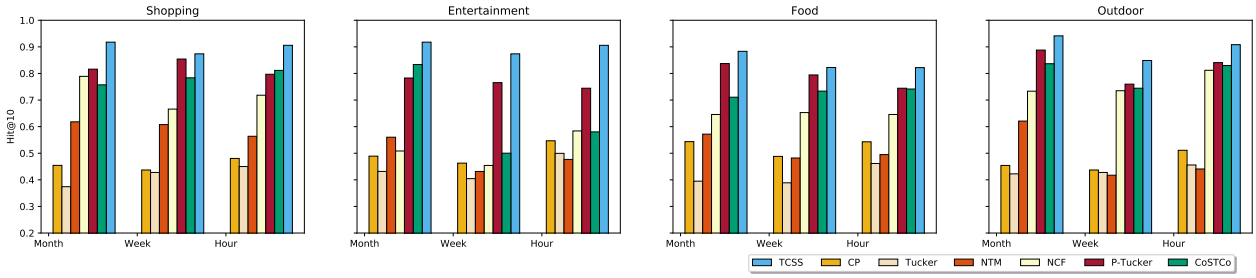


Fig. 4. Hit@10 on Different Categories

the test set. It is clear from Table I that tensor completion methods outperform matrix completion. The reason is that recommendation is time-sensitive, and most POIs have a peak period of visits in a year. For example, one may prefer to go to an aquatics center in summer instead of winter. It demonstrates the necessity of introducing the time dimension to conduct tensor completion for recommendation.

Our model achieves the best performance at around 92% Hit@10 and 0.62 MRR on Gowalla and Foursquare, outperforming the best-performing baseline, P-Tucker, by 0.1–0.2 in Hit@10 and doubles MRR. On Yelp, the performance drops since the sparsity of the data tensor in Yelp is lower than that in Gowalla and Foursquare. It implies that with more entries being observed, the recommendation performance can be further improved. We also observe that neural tensor networks perform much better than CP and Tucker decomposition. It verifies the effectiveness of using neural networks to solve the tensor completion problem. Compared with these tensor completion approaches, our model is able to utilize side information and whole data for model training. The improvements verify the effectiveness of our design. Also, those predictive baselines based on spatial-temporal neural networks (STRNN, STAN, STGN) or social networks (LFBCA) do not show advantage over the tensor completion formulation as adopted by us. Note that TCSS is able to model the spatial-temporal factors and simultaneously leverage the social relations. TCSS achieves the best performance on all datasets, which verifies the effectiveness of our social Hausdorff distance and PU learning strategy for recommendation.

F. Ablation Study

We also conduct ablation study to consider several variants of TCSS to validate the effectiveness of our techniques in model design.

Social Hausdorff distance. To verify the effectiveness of the proposed Social Hausdorff distance, we introduce two variants of our model: Self-Hausdorff and Zero-out. Specifically, (i) Self-Hausdorff replaces $\mathcal{N}(v_i)$ in Eq(13) with the set of POIs already visited by User v_i, to remove the social influence from the loss; (ii) Zero-out is trained with $\mathcal{L}_2$ only, and it disregards any POI that has a distance greater than a threshold σ to its nearest POI of User v_i, where σ is configured as 1% of the maximum distance between any two POIs.

In both the variants, those POIs that are far from the POIs checked by User v_i before are unlikely to be recommended to v_i. However, in a real scenario, v_i may turn to friends for POI recommendations, and some recommended POIs could be far from those POIs already visited by v_i. This scenario cannot be captured by these two variants, but is well captured by our social Hausdorff distance loss head $\mathcal{L}_1$. As shown in Table II, replacing our Social Hausdorff distance module with Self-Hausdorff or Zero-out leads to a performance degrade.

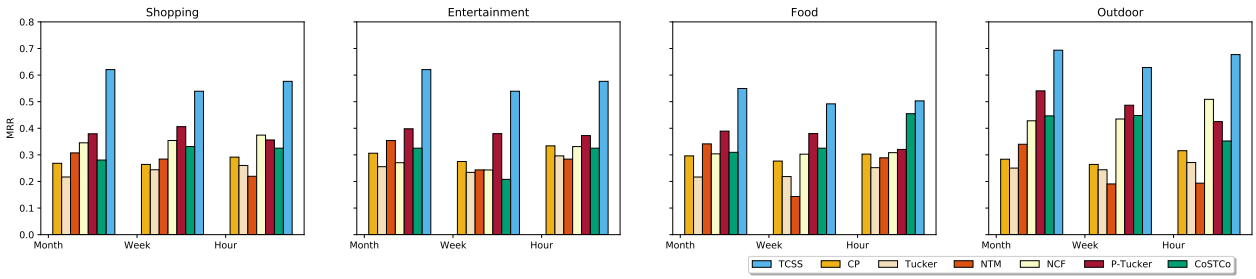
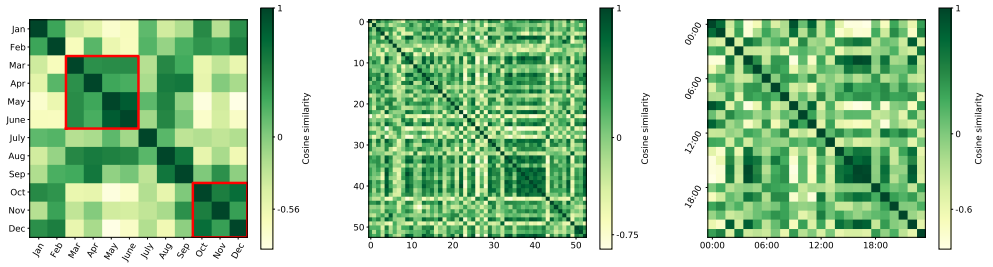
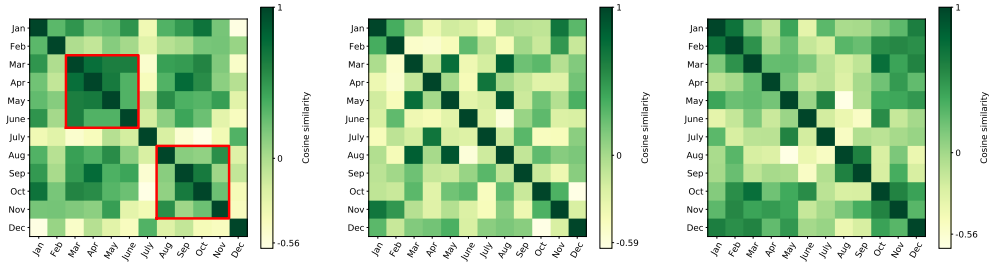


Fig. 5. MRR on Different POI Categories



(a) Month (Blocks Highlighted) (b) Week (c) Hour

Fig. 6. Heatmap of Similarity (Shopping)



(a) Entertainment (Blocks Highlighted) (b) Food (c) Outdoor

Fig. 7. Similarity on Other Categories

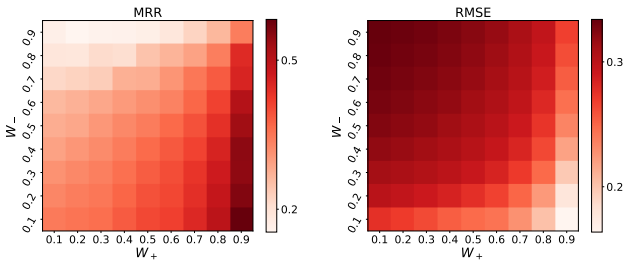
To further verify the effectiveness of our social Hausdorff distance loss head $\mathcal{L}_1$, we introduce another variant that only minimizes the least-squares error head (i.e., $\lambda = 0$). This method is marked by "Removing $\mathcal{L}_1$ ($\lambda = 0$)" in Table II, and we can see that the performance of this variant degrades significantly on all three datasets. This further verifies that our social Hausdorff loss head is important to improve the performance of timely recommendation.

Initialization. To verify the effectiveness of our spectral method for initializing latent factors, we introduce two variants which replace our initialization method with (i) naïve random initialization and (ii) one-hot encoding. In particular, the naïve random initialization associates $\mathbf{U}^1_i$, $\mathbf{U}^2_j$ and $\mathbf{U}^3_k$ with a random vector, which is also the initialization strategy of CP and Tucker. The variant of one-hot encoding uses the method of NCF [20], which indexes each user i (resp. POI j or time k) with its corresponding position being 1 and others being 0. This high-dimensional vector is then converted to embeddings using a learnable embedding layer. We use the same value for tensor rank in both model variants that adopt the baseline initialization methods. As Table II shows, neither of the two variants outperforms TCSS on any of the 3 datasets. Note that our spectral initializing method additionally enjoys fast convergence, which we will demonstrate later.

Learning on Whole Data. Different from existing approaches, we train TCSS with all entries (including all unlabeled entries) instead of sampling a subset of unlabeled entries as negative entries. Alternatively, we introduce a variant which adopts the strategy of negative sampling in [20] to randomly sample some negative entries, the number of which equals that of the observed entries. During training, we minimize the squared error between the original tensor and reconstructed tensor over all the observed entries and the sampled negative entries. Note that the loss head based on social Hausdorff distance remains in this variant and we only replace $\mathcal{L}_1$. This variant is marked as "Negative Sampling" in Table II. We observe that our training strategy on the whole data achieves better performance than negative sampling in terms of both metrics on all three datasets.

G. Analysis on POI Types and Time Granularity

Effect of POI Category. The Gowalla dataset assigns each POI a category, which provides us an opportunity to investigate how recommendation performance may vary depending on different types of POIs. Figures 4 and 5 show the results of TCSS and other baselines on four different POI categories: shopping, entertainment, food and outdoor. Note that we only consider one category in the training and test process, i.e., each tensor only involves one specific category of POIs. We observe that our model consistently outperforms all the baselines by a largin margin on all categories. Also interestingly, the performance on outdoor POIs is stronger than others. Intuitively, this is because outdoor POIs exhibit more seasonal characteristics (e.g., few people go to swimming in winter, so it is less likely to recommend an aquatics center to a user with a high score). Also surprisingly, the performance on the food category is relatively weak. There are two possible reasons. First, food POIs are less seasonal: people can go to a restaurant at anytime of the year. Second, people taste different types of food on a daily basis. Therefore, it is difficult



(a) MRR (b) RSME

Fig. 8. Effect of Different Weight Combinations (Gowalla)

TABLE III
PERFORMANCE WITH DIFFERENT (w_+, w_-)

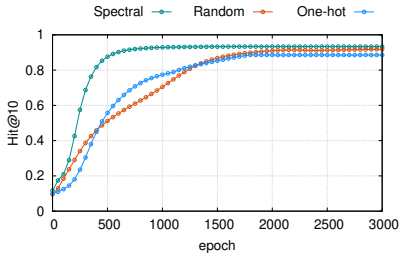
(w_+, w_-)	RMSE		Hit@10	MRR
	Positive	Negtive		
(0.9, 0.1)	0.4254	0.1627	0.9061	0.5875
(0.95, 0.05)	0.4197	0.1601	0.9077	0.6008
(0.99, 0.01)	0.4148	0.1577	0.9177	0.6206
(0.995, 0.005)	0.4163	0.1593	0.9198	0.6198
(0.999, 0.001)	0.4171	0.1602	0.9184	0.6039

to infer the potential check-in. In all four POI categories, TCSS outperforms the baselines by a large margin thanks to its leverage of social homophily that is naturally present in LBSNs.

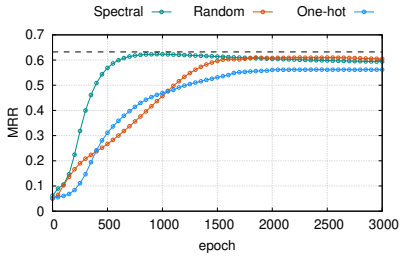
Effect of Time Granularity. In both Figures 4 and 5, we also show the model performance when the granularity of time is varying. Specifically, for "month", the length of the time dimension is 12 since there are 12 months in a year. For example, if a check-in occurs in August, the entry has $k = 7$ (k starts from 0). Likewise, "week" indicates during which week in a year a check-in occurs. Since there are 53 weeks in year, the length of time dimension is 53. Different from "month" and "week" using the index inside a year, "hour" uses the index of hour in a day. For example, if a user visits a bar at 22:00, the time index will be 21. Generally, few users will visit a bar during the daytime, so a recommender system would be able to utilize this time-unit specific knowledge when a proper time unit is chosen to build the data tensor. From Figures 4 and 5, we observe that recommendation in the granularity of "month" maintains a stronger predictive accuracy than "week", which justifies that our month-based time granularity in the previous subsections is a fair setting. Again, we see that TCSS significantly outperforms the baseline approaches in all the three granularity units being considered.

To further investigate the correlation of time units when using different time granularity, we calculate the cosine similarity between the latent factors (columns of $\mathbf{U}_3$) of two months, weeks and hours, respectively, the heatmaps of which are shown in Figure 6 where a darker color means that the factors of two time units are more similar (cosine similarity closer to 1). In Figure 6(a), we highlight two dark blocks in the heat map using red boxes, which suggest that the learned monthly temporal factors capture seasonal changes for recommendation. For example, one dark block in Figure 6(a) indicates that March, April, May and June are similar to each other. We also observe some blocks in Figures 6(b) and 6(c), but the seasonal changes revealed by weekly and hourly factors are weaker, partially explaining why their recommendation performance is not as good as the model using the "month" granularity in Figures 4 and 5.

Note that Figure 6 only shows the cosine similarity of different time units for the shopping POI category. Figure 7 further shows the cosine similarity between months for different POI categories, where we see that dark block patterns vary with the POI categories. Compared to other categories, there are fewer dark blocks on heatmap of "food", which further explains the poorer recommendation performance on the "food" category. To sum up, the more seasonality that is present in the time units adopted to re-construct the data tensor, the better TCSS prediction performance would be.



(a) Hit@10



(b) MRR

Fig. 9. Effectiveness of Initialization

TABLE IV
TRAINING TIME (ONE EPOCH)

Methods	Gowalla	Yelp	Foursquare
Original Loss: Eq (14)	2.29×10^5 s	2.49×10^5 s	4.51×10^5 s
Negative Sampling	30.12 s	11.03 s	30.61 s
Rewritten Loss: Eq (15)	0.13 s	0.11 s	0.17 s

H. Setting of weights

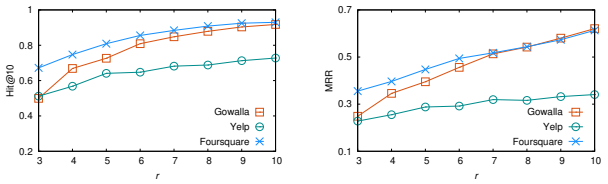
Since the number of negative entries is much larger than that of positive entries, we give more weight to positive entries to balance their importance during training. We have carefully tuned the setting of w_+ and w_- to obtain good default values. As shown in Figure 8, given a fixed w_-, the MRR will increase and the RMSE will decrease as w_+ increases. It verifies that setting the positive weight w_+ to be much larger than the negative weight w_- can improve the overall performance. Note that the weight scale matters. For example, $w_+ = w_- = 0.1$ is not equivalent to $w_+ = w_- = 0.9$, because only $\mathcal{L}_2$ in Eq (15) contains the weights so its relative importance w.r.t. $\mathcal{L}_1$ changes with weight scaling.

We also fine-tune (w_+, w_-) beyond $(0.9, 0.1)$. As Table III shows, as the ratio w_+/w_- increases, the performance improves till $(w_+, w_-) = (0.995, 0.005)$, beyond which the performance drops. Our default $(w_+, w_-) = (0.99, 0.01)$ achieves the highest Hits@10.

I. Training Efficiency

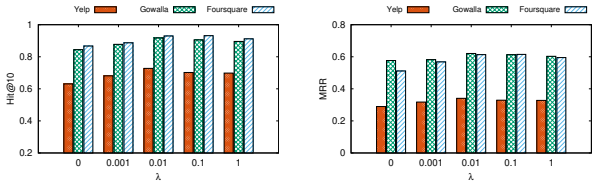
We next report the experiments to verify the training efficiency of our initialization method and the rewritten loss function. Recall that we introduced two TCSS variants that utilize different initialization methods: (i) random and (ii) one-hot. We compare the training process of our model to these two variants. Figure 9 shows the change of Hit@10 and MRR in the training process. Compared with random and one-hot vector initialization, our method can lead to a much faster convergence rate and a slight performance gain in both metrics. This is because our method initializes the factors with eigenvectors which are the estimation of the genuine factors. However, random or one-hot vector initialization may get trapped in undesirable stationary points when we use gradient descent to learn the factors.

We also compare our training time of each epoch with two other variants: (i) negative sampling and (ii) learning on the whole data



(a) Hit@10 (b) MRR

Fig. 10. Varying r



(a) Hit@10 (b) MRR

Fig. 11. Varying λ

directly by Eq (14) without rewriting. Note that we follow [20] to generate the negative samples. Since the number of samples is large, if we put all samples in a single batch, it will crash due to memory limit. Therefore, we fill each batch with 8096 samples instead of using a single large batch. Recall that the original loss function based on the whole data has a time complexity of $O(I \times J \times K \times r)$, while our rearrangement method reduces the complexity to $O((I+J+K) \times r^2)$.

Table IV shows a comparison of training time. Considering that the values of I and J are large in recommender systems (e.g., $I = 20{,}359$ in "Foursquare"), our rearranged loss can save a lot of computational resources. We observe in Table IV many orders of magnitude speedup in the training time of our method over the variant that directly computes Eq (14). While learning on the whole data combats the sensitivity of negative sampling, our rewritten loss function makes it truly practical to learn on large real-world data. Note that our training time is only around 1% of that of negative sampling (8096 entries per batch), while being more accurate as Table II has indicated.

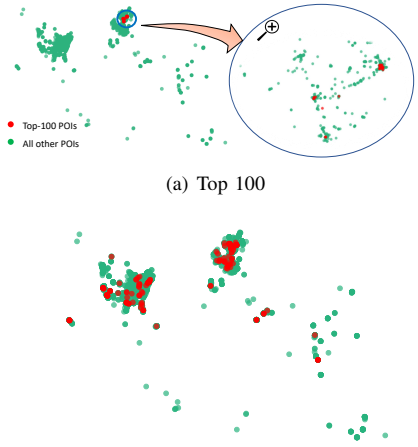
J. Parameter Sensitivity

The value of tensor rank r (i.e., the length of embeddings) plays a vital role in tensor completion. In Figure 10, we plot the performance of our model when the value of rank r varies. On all our three datasets, we see that a large value of r leads to significant gains in performance. Note that r is limited by the number of units along the time dimension (12 when month is used), K, which is much smaller than I and J. Due to the limitation of eigenvectors computation process, the maximum of r is 10 (less than $K - 1$). We can see that $r = 10$ gives the better performance than smaller values, since it allows TCSS to capture more factors.

The value of λ indicates the weight of the social Hausdorff distance in the loss function $\mathcal{L} = \lambda\mathcal{L}_1 + \mathcal{L}_2$. Figure 11 shows the effect of the regularization weight λ of the social-homophily loss head on the overall recommendation performance. We can see that as λ increases towards 0.01, the model performance improves in terms of all metrics on all three datasets; but the performance degrades as λ increases further to 1. It indicates that there exists a tradeoff in weighing the social Hausdorff distance loss head with the least-squares error.

K. A Case Study

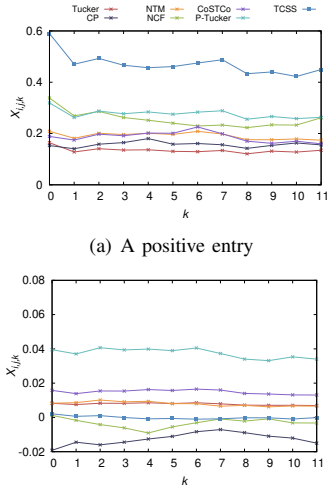
We next conduct a study case to investigate the recommendation scores from TCSS. Figure 12 shows the locations of all POIs in green. For a randomly selected user i, we sort his/her recommendation scores at a specific time k. The red points in Figure 12(a) highlight the top-100 scored POIs, and we can see that the POIs are clustered in small areas, which verifies the presence of Tobler's first law of geography in recommendation. As shown in Figure 12(b), the top-200 POIs are distributed in a much larger area, meaning that we can find a diverse set of recommended POIs as we move down the list.



(a) Top 100

(b) Top 200

Fig. 12. POIs with High Scores



(a) A positive entry

(b) A negtive entry

Fig. 13. Score Along the Time Dimension

We also investigate how the scores vary along the time dimension for different methods. As shown in Figure 13(a), for a randomly selected observed entry (i, j, k), we fix User i and POI j and compute the scores along the time dimension. Compared with baselines, our model leads higher scores though the score values vary over time. For a randomly selected negative (i.e., unobserved) entry, Figure 13(b) shows that the scores predicted by TCSS are near 0 as expected. In summary, our model associates a possible check-in with a higher score than baselines and assigns a negative entry with a low score in the meanwhile, which verifies its recommendation accuracy.

VI. Conclusion

In this paper, we studied the tensor completion problem for timely POI recommendation in an LBSN. We proposed a tensor completion model called TCSS to take advantage of the social-spatial side information to enforce social homophily and Tobler's first law of geography. TCSS adopts many techniques to improve recommendation quality, including a spectral based factor initialization, a novel social Hausdorff distance head to encode the social-spatial side information, and a smart way to reduce the computational cost of least-squares error over the whole data, the latter of which improves model stability compared with negative sampling. The effectiveness of these techniques has been empirically verified.

REFERENCES

[1] O. U. Aydin, A. A. Taha, A. Hilbert, A. A. Khalil, I. Galinovic, J. B. Fiebach, D. Frey, and V. I. Madai. On the usage of average hausdorff distance for segmentation performance assessment: Hidden bias when used for ranking, 2020.

[2] G. Blanc and S. Rendle. Adaptive sampled softmax with kernel based sampling. In *ICML*, 2018.

[3] C. Cai, G. Li, H. V. Poor, and Y. Chen. Nonconvex low-rank symmetric tensor completion from noisy data. *CoRR*, 2019.

[4] C. Cai, H. V. Poor, and Y. Chen. Uncertainty quantification for nonconvex tensor completion: Confidence intervals, heteroscedasticity and optimality. In *ICML*, 2020.

[5] E. J. Candès and B. Recht. Exact matrix completion via convex optimization. *Commun. ACM*, 55(6):111–119, 2012.

[6] J. D. Carroll and J.-J. Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 1970.

[7] J. D. Carroll, S. Pruzansky, and J. B. Kruskal. Candelinc: A general approach to multidimensional analysis of many-way arrays with linear constraints on parameters. *Psychometrika*, 1980.

[8] C. Chen, M. Zhang, C. Wang, W. Ma, M. Li, Y. Liu, and S. Ma. An efficient adaptive transfer neural network for social-aware recommendation. In *SIGIR*, 2019.

[9] C. Chen, M. Zhang, Y. Zhang, Y. Liu, and S. Ma. Efficient neural matrix factorization without sampling for recommendation. *ACM Trans. Inf. Syst.*, 38(2):14:1–14:28, 2020.

[10] H. Chen and J. Li. Neural tensor model for learning multi-aspect factors in recommender systems. In *IJCAI 2020*.

[11] Y. Chen, Y. Chi, J. Fan, and C. Ma. Gradient descent with random initialization: fast global convergence for nonconvex phase retrieval. *Math. Program.*, 2019.

[12] D. Cheng, R. Peng, Y. Liu, and I. Perros. SPALS: fast alternating least squares via implicit leverage scores sampling. In *NeurIPS*, 2016.

[13] J. Cranshaw, E. Toch, J. I. Hong, A. Kittur, and N. M. Sadeh. Bridging the gap between physical location and online social networks. In *UbiComp*, 2010.

[14] P. Cremonesi, Y. Koren, and R. Turrin. Performance of recommender algorithms on top-n recommendation tasks. In X. Amatriain, M. Torrens, P. Resnick, and M. Zanker, editors, *RecSys*, pages 39–46. ACM, 2010.

[15] X. Fang, R. Pan, G. Cao, X. He, and W. Dai. Personalized tag recommendation through nonlinear tensor factorization using gaussian kernel. In *AAAI*, 2015.

[16] R. A. Harshman. Parafac2: Mathematical and technical notes. *UCLA working papers in phonetics*, 1972.

[17] R. A. Harshman et al. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. 1970.

[18] R. A. Harshman and M. E. Lundy. Uniqueness proof for a family of models sharing features of tucker's three-mode factor analysis and parafac/candecomp. *Psychometrika*, 1996.

[19] L. He, C. Lu, G. Ma, S. Wang, L. Shen, P. S. Yu, and A. B. Ragin. Kernelized support tensor machines. In *ICML*, 2017.

[20] X. He, L. Liao, H. Zhang, L. Nie, X. Hu, and T. Chua. Neural collaborative filtering. In R. Barrett, R. Cummings, E. Agichtein, and E. Gabrilovich, editors, *WWW*, 2017.

[21] B. Hu and M. Ester. Spatial topic modeling in online social media for location recommendation. In Q. Yang, I. King, Q. Li, P. Pu, and G. Karypis, editors, *Seventh ACM Conference on Recommender Systems, RecSys '13, Hong Kong, China, October 12-16, 2013*, pages 25–32. ACM, 2013.

[22] B. Hui, H. Chen, D. Yan, and W. Ku. EDGE: entity-diffusion gaussian ensemble for interpretable tweet geolocation prediction. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, pages 1092–1103. IEEE, 2021.

[23] B. Hui, D. Yan, and W. Ku. Node-polysemy aware recommendation by matrix completion with side information. In *2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, December 15-18, 2021*, pages 636–642. IEEE, 2021.

[24] B. Hui, D. Yan, W. Ku, and W. Wang. Predicting economic growth by region embedding: A multigraph convolutional network approach. In M. d'Aquin, S. Dietze, C. Hauff, E. Curry, and P. Cudré-Mauroux, editors, *CIKM '20: The 29th ACM International Conference on Information and Knowledge Management, Virtual Event, Ireland, October 19-23, 2020*, pages 555–564. ACM, 2020.

[25] P. Jain and S. Oh. Provable tensor factorization with missing data. In *NeurIPS*, 2014.

[26] J. Jang and U. Kang. D-tucker: Fast and memory-efficient tucker decomposition for dense tensors. In *ICDE*, 2020.

[27] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos. Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries. In Q. Yang, D. Agarwal, and J. Pei, editors, *SIGKDD*, 2012.

[28] A. Karatzoglou, X. Amatriain, L. Baltrunas, and N. Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *RecSys*, 2010.

[29] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In *ICLR (Poster)*. OpenReview.net, 2017.

[30] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 2009.

[31] D. Lian, C. Zhao, X. Xie, G. Sun, E. Chen, and Y. Rui. Geomf: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *SIGKDD*, 2014.

[32] H. Liu, Y. Li, M. Tsang, and Y. Liu. Costco: A neural tensor completion model for sparse tensors. In *SIGKDD*, 2019.

[33] Q. Liu, S. Wu, L. Wang, and T. Tan. Predicting the next location: A recurrent model with spatial and temporal contexts. In *AAAI*, 2016.

[34] Y. Luo, Q. Liu, and Z. Liu. STAN: spatio-temporal attention network for next location recommendation. In *WWW*, 2021.

[35] S. Madisetty. Event recommendation using social media. In *ICDE*, 2019.

[36] P. R. Monge, N. S. Contractor, P. S. Contractor, R. Peter, S. Noshir, et al. *Theories of communication networks*. Oxford University Press, USA, 2003.

[37] S. Oh, N. Park, L. Sael, and U. Kang. Scalable tucker factorization for sparse tensors - algorithms and discoveries. In *ICDE*, 2018.

[38] A. Potechin and D. Steurer. Exact tensor completion with sum-of-squares. In *COLT*, 2017.

[39] S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. Factorizing personalized markov chains for next-basket recommendation. In *WWW*, 2010.

[40] J. Ribera, D. Guera, Y. Chen, and E. J. Delp. Locating objects without bounding boxes. In *CVPR*, pages 6479–6489. Computer Vision Foundation / IEEE, 2019.

[41] B. Romera-Paredes and M. Pontil. A new convex relaxation for tensor completion. In *NeurIPSs*, 2013.

[42] Q. Song, X. Huang, H. Ge, J. Caverlee, and X. Hu. Multi-aspect streaming tensor completion. In *SIGKDD*, 2017.

[43] W. R. Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970.

[44] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966.

[45] H. Wang, M. Terrovitis, and N. Mamoulis. Location recommendation in location-based social networks using user check-in data. In C. A. Knoblock, M. Schneider, P. Kröger, J. Krumm, and P. Widmayer, editors, *21st SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2013, Orlando, FL, USA, November 5-8, 2013*, pages 364–373. ACM, 2013.

[46] M. Wang, M. Gong, X. Zheng, and K. Zhang. Modeling dynamic missingness of implicit feedback for recommendation. In *NeurIPS*, 2018.

[47] X. Xin, F. Yuan, X. He, and J. M. Jose. Batch IS NOT heavy: Learning word representations from all samples. In *ACL*, 2018.

[48] C. Yang, L. Bai, C. Zhang, Q. Yuan, and J. Han. Bridging collaborative filtering and semi-supervised learning: A neural approach for POI recommendation. In *SIGKDD*, 2017.

[49] L. Yao, Q. Z. Sheng, Y. Qin, X. Wang, A. Shemshadi, and Q. He. Context-aware point-of-interest recommendation using tensor factorization with social regularization. In *SIGIR*, pages 1007–1010. ACM, 2015.

[50] Q. Yao, J. T. Kwok, and B. Han. Efficient nonconvex regularized tensor completion with structure-aware proximal iterations. In *ICML*, 2019.

[51] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983. ACM, 2018.

[52] P. Zhao, H. Zhu, Y. Liu, J. Xu, Z. Li, F. Zhuang, V. S. Sheng, and X. Zhou. Where to go next: A spatio-temporal gated network for next POI recommendation. In *AAAI*, 2019.