Personalized Location Recommendation by Aggregating Multiple Recommenders in Diversity

Ziyu Lu
Dept. of Computer Science
The Univ. of Hong Kong
zylu@cs.hku.hk

Hao Wang
State Key Lab. for Novel Software Technology
Nanjing University
wanghao@nju.edu.cn

Nikos Mamoulis
Dept. of Computer Science
The Univ. of Hong Kong
nikos@cs.hku.hk

Wenting Tu
Dept. of Computer Science
The Univ. of Hong Kong
wttu@cs.hku.hk

David W. Cheung
Dept. of Computer Science
The Univ. of Hong Kong
dcheung@cs.hku.hk

ABSTRACT

Location recommendation is an important feature of social network applications and location-based services. Most existing studies focus on developing one single method or model for all users. By analyzing real location-based social networks, in this paper we reveal that the decisions of users on place visits depend on multiple factors, and different users may be affected differently by these factors. We design a location recommendation framework that combines results from various recommenders that consider various factors. Our framework estimates, for each individual user, the underlying influence of each factor to her. Based on the estimation, we aggregate suggestions from different recommenders to derive personalized recommendations. Experiments on Foursquare and Gowalla show that our proposed method outperforms the state-of-the-art methods on location recommendation.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Information Filtering

Keywords

Location recommendation, personalization, aggregation

1. INTRODUCTION

Due to the proliferation of mobile devices, location-based social networks (LBSNs) have emerged. Some of these social networks are dedicated to location sharing (e.g., Gowalla, Foursquare, etc.) while others provide location-based features together with other social networking services (e.g., Facebook, Twitter, etc.). In either case, users can share with their friends the experience of visiting locations (such as restaurants, view spots, etc.). Location recommendation is very important for these applications, because it provides better user experience and may thus contribute to business promotion in cyber-physical space.

This paper focuses on *check-in records* of LBSN users. An LBSN user may *check-in* at different locations from time to time, by explicit check-in operations via mobile applications or by implicit actions such as posting photos with location annotations. Therefore, over time, a user generates a sequence of check-in records. It is important to distinguish user-location check-ins from their analogues in classic recommender systems (e.g., user-item ratings [1]). As Hu et al. [12] have pointed out, a rating is *explicit*, indicating directly whether and how a user likes an item, whereas a check-in record is *implicit* with some unique characteristics:

1. There is no negative feedback in check-in records, i.e., check-ins do not indicate whether a user dislikes a location.
2. The check-in frequency, even after normalization, is not a reliable indicator of how much a user likes a location.

Despite any differences between ratings and check-ins, conventional *collaborative filtering* (CF) methods (e.g., [1]) are computationally applicable to check-in data. Recently, many new approaches have been proposed, making use of social, geographical, temporal, and semantic information of LBSN data (e.g., [7, 10, 18, 26] and others discussed later in Section 2). However, most existing methods take *unified* perspectives towards the recommendation problem, though some of them do consider more than one aspects of the check-in data. In other words, most exising methods focus on developing one *single* method / model for *all* users.

In this paper we argue, and reveal by analyzing real LBSN datasets, that the decisions of users on where to go depend on multiple factors, and different users may be affected differently by these factors. For example, some users are easily affected by their friends and do not care much about the places to visit, while some others are more independent with more concern of the places. Therefore, we aim to *personalize* location recommenders. That being the purpose, we propose a general framework to combine multiple recommenders that are potentially useful. A pair-based optimization problem is formulated and solved to identify the underlying preference of each user. The contributions of this paper are as follows:

- By analyzing two real LBSN datasets, Foursquare [10] and Gowalla [6], we reveal the diversity among recommendations made by different recommenders and the diversity of user preferences over recommenders.
- We propose a framework for location recommendation. Un-

LocalRec'15, September 19, 2015, Vienna, Austria.

der our framework we are able to personalize location recommenders to better serve the users.

- We evaluate our method using Foursquare and Gowalla data. The results show that our method outperforms the state-of-the-art location recommenders.

The rest of this paper is organized as follows. By analyzing Foursquare and Gowalla data, Section 2 shows the diversity in recommenders and user preferences. Motivated by certain observations, Section 3 describes LURA, our proposed framework, which is extensively evaluated in Section 4. Finally, Section 5 presents some related work and Section 6 concludes the paper.

2. DATA AND RECOMMENDERS

In this section we first introduce the datasets and formally define the location recommendation problem. Then, we select 11 representative recommenders which, to the best of our knowledge, cover all the factors that the state-of-the-art methods consider in location recommendation. By analyzing their performance on different users, we demonstrate the diversity of (i) recommendations from the representative recommenders and (ii) users' check-in behaviors.

2.1 Datasets & Recommendation Problem

We use Foursquare [10] and Gowalla [6] datasets. In both datasets, there are users ($\mathcal{U}$), locations ($\mathcal{L}$), and check-ins each in the form of a tuple (u, ℓ, t) recording the event that user u visited location ℓ at time t. Note that from the tuples we can infer the *check-in frequency* $c_{u,\ell}$ of user u to location ℓ. In many recommenders, the matrix $\mathbf{C} = (c_{u,\ell})_{u\in\mathcal{U},\ell\in\mathcal{L}}$ is the primary data source. Associated with each location there is a longitude and latitude coordinate. Friendships are represented as undirected, unweighted pairs of users. In addition, for each location we have collected its semantic information (i.e., category) from a Foursquare API[1].

We filtered the datasets to include only users that have at least 10 check-in records, and locations that are visited at least twice. The resulting Foursquare dataset has 11,326 users, 96,002 locations, and 2,199,782 check-in records over 364 days; Gowalla is a bit larger, containing 74,725 users, 767,936 locations, and 5,829,873 check-in records over a period of 626 days.

Top-N location recommendation. Given a user u, the *top-N location recommendation* problem is to suggest to u a list of N locations, previously unvisited by u, with the expectation that u will be intrigued to visit (some of) these locations.

2.2 Representative Recommenders

We select 11 representative recommenders that consider social, geographical, temporal, as well as semantic aspects of the data.

2.2.1 User-based CF Methods

User-based CF methods assume that similar users have similar preferences over locations, thus the score of location ℓ for user u, score(u, ℓ), is computed by the similarity-weighted average of other users' visits to ℓ:

$$\text{score}(u,\ell) = \frac{\sum_{v\in\mathcal{U}} w_{u,v}\cdot c_{v,\ell}}{\sum_{v\in\mathcal{U}} w_{u,v}}.$$

N locations with the largest scores are recommended to user u. Different weighing schemas (i.e., $w_{u,v}$) yield different recommenders (R_1-R_5).

R_1: User-based CF (UCF). $w_{u,v}$ could be the cosine similarity between users u and v, i.e., $w_{u,v} = \cos(\boldsymbol{c}_u, \boldsymbol{c}_v)$,where $\boldsymbol{c}_u$ and $\boldsymbol{c}_v$ are corresponding rows in the check-in matrix $\mathbf{C}$. This is the conventional user-based CF method [1].

R_2: Friend-based CF (FCF). Similarity between users could be reflected by their common friends. For friends u and v, let $w_{u,v} = \text{Jaccard}(F_u, F_v)$, where F_* refers to the set of friends of a user and $\text{Jaccard}(\cdot,\cdot)$ computes the Jaccard index. This recommender is proposed by Konstas et al. [14].

R_3: Friend-location CF (FLCF). In addition to common friends, commonly visited locations may as well reflect the closeness of two friends: $w_{u,v} = \eta\cdot\text{Jaccard}(F_u, F_v) + (1-\eta)\cdot\text{Jaccard}(L_u, L_v)$, where u and v are friends; L_* is the set of locations that a user has visited [14]. Parameter $\eta\in[0,1]$ is the weighing between common friends and common locations. Setting $\eta = 0.1$ achieves the best performance for this method on our data.

R_4: Geo-distance CF (GCF). The rationale of GCF is that nearby friends are more influential than faraway ones. This intuition is used by Ying et al. [28], where the weight $w_{u,v}$ is a rescale of the geographical distance between u and v:

$$w_{u,v} = 1 - \frac{\text{geodist}(u,v)}{\max_{w\in F_u}\text{geodist}(u,w)}.$$

The location of each user can be inferred from her most frequently visited locations.

R_5: Category CF (CCF). Consider users as keywords and location categories as documents; between user u and category C there can be a *relevance score*, $\text{rel}(u, C)$ (e.g., TF-IDF). To measure the similarity between two users u and v, Bao et al. [2] consider the sum of minimum relevance scores over all categories, i.e., $\text{sim}(u,v) = \sum_C \min\{\text{rel}(u,C), \text{rel}(v,C)\}$. This value is then penalized by the difference between users' *randomness* in preferences, thus the weight $w_{u,v}$ is

$$w_{u,v} = \frac{\text{sim}(u,v)}{1+|\text{ent}(u)-\text{ent}(v)|},$$

where $\text{ent}(\cdot)$ is the *entropy* of a user's preference over categories.

2.2.2 Item-based CF Methods

Item-based CF methods take a "transposed" view of the data, i.e., users are recommended to items instead of the other way round. When the weight $w_{\ell,\ell'}$ between two locations is properly defined, the score of user u to location ℓ, score(u,ℓ), can be computed as

$$\text{score}(u,\ell) = \frac{\sum_{\ell'\in\mathcal{L}} w_{\ell,\ell'}\cdot c_{u,\ell'}}{\sum_{\ell'\in\mathcal{L}} w_{\ell,\ell'}}.$$

R_6: Item-based CF (ICF). Similar to UCF, $w_{\ell,\ell'}$ could be the cosine similarity $\cos(\boldsymbol{c}_\ell, \boldsymbol{c}_{\ell'})$, where $\boldsymbol{c}_\ell$ and $\boldsymbol{c}_{\ell'}$ are corresponding columns in $\mathbf{C}$ the check-in matrix [1, 21].

R_7: Time-weighted CF (TCF). Recent check-in records are relatively more reliable if we consider any evolution of user preference. Ding and Li [7] use an exponential discounting function $f(t) = e^{-\alpha t}$ to describe temporal bias in $w_{\ell,\ell'}$. Namely, $w_{\ell,\ell'} = \cos(\boldsymbol{c}_\ell, \boldsymbol{c}_{\ell'})\, f(t_{u,\ell'})$ where $t_{u,\ell'}$ is the time of the user's check-in at ℓ'. α is the decreasing rate. We set $\alpha = \frac{1}{7}$ so that TCF achieves the best performance.

2.2.3 Probabilistic Methods

Besides the CF family, another methodology of making recommendations is to estimate the probability of user u visiting location ℓ, conditioned on the check-in history of u, i.e.,

$$\text{score}(u,\ell) = \Pr\{\ell\,|L_u\}.$$

The most probable N locations are recommended to the user. The next three recommenders (R_8-R_{10}) attempt to estimate $\Pr\{\ell\,|L_u\}$.

[1]https://developer.foursquare.com/

R_8: Power-law model (PLM). Ye et al. [26] study pairs of locations that have at least one common visitor, $P = \{(\ell, \ell') \,|U_\ell \cap U_{\ell'} \neq \emptyset\}$, and report that the set of all distance values, $\{\text{geodist}\,(\ell, \ell')\}_{(\ell,\ell')\in P}$, obeys a power-law distribution. Hence, $\Pr\{\ell\,|L_u\} = \prod_{\ell'\in L_u} a \cdot \text{geodist}\,(\ell, \ell')^b$, where parameters a and b are determined via a regression process.

R_9: Kernel density model (KDM). For a user u with past check-ins at locations $L_u = \{\ell_1, \ell_2, \cdots, \ell_n\}$, this factor estimates the probability of u visiting a new location ℓ using kernel techniques: $\Pr\{\ell\,|L_u\} = \frac{1}{n}\sum_{j=1}^{n} K_h\,(\text{geodist}\,(\ell, \ell_j))$, where $K_h(\cdot)$ is a *scaled kernel* trained on an individual basis. h is set to $n^{-\frac{1}{d+4}}$ where $d = 1$ is the dimensionality of the data (distance values). This model is used by Zhang and Chow [30].

R_{10}: Spatial kernel density model (SKDM). This is our improvement over R_9. Instead of $\text{geodist}\,(\ell, \ell')$ we directly consider distances between latitudes and longitudes and employ a 2D kernel density estimation, i.e., $\Pr\{\ell\,|L_u\} = \frac{1}{n}\sum_{j=1}^{n} K_h\,(\text{lat}(\ell) - \text{lat}(\ell_j), \text{lon}(\ell) - \text{lon}(\ell_j))$; here, $\text{lat}(\cdot)$ and $\text{lon}(\cdot)$ give the latitude and longitude of a location, and K_h is a 2D scaled kernel, with $h = n^{-\frac{1}{d+4}} = n^{-\frac{1}{6}}$.

2.2.4 *Matrix Factorization*

The last recommender belongs to the matrix factorization (MF) family. MF attempts to find latent structures of the check-in matrix $\mathbf{C}$. In particular, MF tries to find low-rank matrices (i.e., latent user and location profiles) $\mathbf{P}$ and $\mathbf{Q}$ such that $\hat{\mathbf{C}} = \mathbf{PQ}$ is a good approximation of $\mathbf{C}$. A zero-valued entry $c_{u,\ell}$ in $\mathbf{C}$ may thus have a good estimation $\hat{c}_{u,\ell}$ in $\hat{\mathbf{C}}$; recommendations can then be made.

R_{11}: Implicit matrix factorization (IMF). Proposed by Hu et al. [12], this is a modification of the conventional MF for implicit user feedbacks (such as check-ins).

2.3 Diversity in Recommendations

We now show that the 11 recommenders (R_1-R_{11}) generate very different recommendations. We use the first 300 days of Foursquare and the first 420 days of Gowalla to construct the recommenders, involving 3,942 and 1,307 users respectively. We compare the top-10 suggested locations by different recommenders.[2] For two such length-10 recommendation lists, Jaccard index can be used to measure their similarity. Therefore for each user, we measure the diversity of the 11 recommenders by pairwise aggregation [11]:

$$\text{diversity}(L_1, L_2 \cdots, L_n) = \frac{2 \cdot \sum_{i,j}(1 - \text{Jaccard}\,(L_i, L_j))}{n \cdot (n-1)}.$$

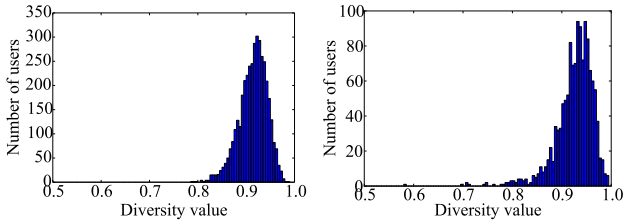
Intuitively, this value will be close to 0 if L_i's are all highly similar, and otherwise be close to 1 if they are pairwise different.

Figure 1 shows the distribution of the diversity values. In Foursquare, the diversity values are all between 0.79 and 0.99, with an average of 0.92, whereas in Gowalla the minimum, maximum, and average values are 0.58, 0.99, and 0.93 respectively. This indicates that the 11 recommenders generate very different results.

2.4 Diversity in User Preferences

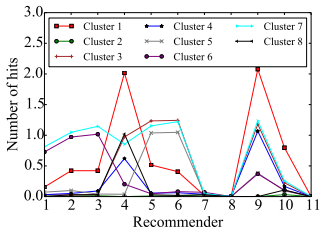
Users may also have different preferences over the aspects on which recommenders are based (i.e., friendship, geographical distance, etc.). To understand user preferences, we test the 11 recommenders using Foursquare (Days 1-300 for training and Days 301-360 for testing). For each user u, each recommender generates a list of 10 locations. The performance of the recommender is then measured by the number of *hits* (i.e., the number of recommended locations that are actually visited by u). Hence, a user u can be depicted by an 11-dimensional vector of which the jth element is the performance of recommender R_j.

[2]Note that sometimes a recommender may fail to generate a list of length 10. For example, FCF (R_2) requires that the target user has some friends but loners do exist in LBSNs. In such cases, we complement the length-10 list with the most popular locations.



(a) Foursquare (b) Gowalla

Figure 1: Distribution of diversity values.



(a) Centers of the clusters

Cluster	Coverage	Variance
1	1.65%	3.06
2	65.3%	0.12
3	5.71%	1.04
4	9.01%	0.82
5	7.52%	0.75
6	4.45%	1.55
7	3.71%	2.58
8	2.65%	0.25

(b) Statistics of the clusters

Figure 2: Diversity in user preferences.

To present the preferences of 3,942 Foursquare users, we group these users into 8 clusters using the K-means algorithm. (The number 8 is chosen to get the best modularity from the resulting clusters.) Figure 2(a) shows the centers of the 8 clusters in parallel coordinates, where an 11-dimensional vector is represented as a sequence of 11 values. Figure 2(b) shows some statistics of each cluster. For example, the entry "1, 1.65%, 3.06" means that, Cluster 1 contains 1.65% of the users, and the variance within the cluster is 3.06. We see that the 11 recommenders perform differently on the 8 clusters.

The above analysis supports our claims: (i) different recommenders make very different recommendations and (ii) different users have different behavioral patterns in visiting locations. This basically indicates that *a good recommender for one person is not necessarily good for another*. Combining different recommenders may produce better recommendations, since multiple recommenders together may cover a wider range of factors that potentially affect users' behaviors.

3. OUR METHOD

In this section we explain LURA, our framework for combining different location recommenders. (The name "LURA" comes from its execution cycle of Learn-Update-Recommend-Aggregate.) Algorithm 1 outlines the steps of LURA, which could be repeated every Δt time (e.g., every 3 days or 1 week, as long as there is necessity in providing new recommendations and there are sufficient data). LURA is based on two important elements: *recommenders* and *user preferences*; every time LURA runs, it keeps these elements up to date. Specifically, if invoked at time t, LURA first

learns the user's current preferences $\alpha_u^t(i)$ (i.e., weights) over different recommenders; this is done by testing the recommendations made at time $t - \Delta t$ against the actual check-ins during the period $(t - \Delta t, t]$ (Line 1). Then, LURA updates the recommenders to make use of all the data available (Line 2). After that, LURA makes recommendations to users, by aggregating the results of the component recommenders (Lines 3-4).

Algorithm 1 LURA$(G^t, \mathcal{R}^{t-\Delta t}, u)$

Input: G^t: the snapshot of LSBN up to the t-th (current) day; $\mathcal{R}^{t-\Delta t}$: a set of n recommenders, trained using $G^{t-\Delta t}$; u: a user
Output: N recommended locations for user u at time t

▷ *At time $t - \Delta t$ each recommender $R_i^{t-\Delta t}$ has recommended N locations to u, $R_i^{t-\Delta t}(u) = \left[\ell_{ij}^{t-\Delta t}\right]_{j=1,2,\cdots,N}$.*

1: **Learn** user u's current preferences, $\alpha_u^t(i)$, on each recommender $R_i^{t-\Delta t}$, based on recommendations $R_i^{t-\Delta t}(u)$ and u's check-in facts during the time period $(t - \Delta t, t]$
2: **Update** $\mathcal{R}^{t-\Delta t}$ to $\mathcal{R}^t$ (or rebuild from scratch) using G^t
3: **Recommend**: each recommender R_i^t recommends N locations to u, in the form of N location-score pairs, $R_i^t(u)$,
$$R_i^t(u) = \left[\ell_{ij}^t\right]_{j=1,2,\cdots,N}.$$
4: **Aggregate** recommendations $\boldsymbol{R}^t(u) = \left[R_i^t(u)\right]_{i=1,2,\cdots,n}$ using weights $\boldsymbol{\alpha}_u^t = \left[\alpha_u^t(i)\right]_{i=1,2,\cdots,n}$ to generate the final recommendation of N locations to u

Although maintaining and executing individual recommenders (Lines 2-3) is straightforward, LURA's novelty lies in the learning of user preferences $\boldsymbol{\alpha}_u^t$ (Section 3.1)[3] and in the strategies for aggregating the outputs $\boldsymbol{R}_u^t$ of different recommenders (Section 3.2). In general, the aggregation is a linear combination:

$$s^t(u, \ell; \boldsymbol{\alpha}_u^t, \varphi) = \sum_{i=1}^{n} \alpha_u^t(i) \cdot \varphi\left(R_i^t(u, \ell)\right). \quad (1)$$

Here $R_i^t(u, \ell)$ is the score of user-location pair (u, ℓ) estimated by recommender R_i^t; $\varphi(\cdot)$ is a strategy for handling individual scores. When there is no ambiguity on $\boldsymbol{\alpha}_u^t$ or φ in the context, we may omit one or both of them to simplify the notation.

3.1 Learning User Preferences

At time t, LURA has a set of recommenders $\mathcal{R}^{t-\Delta t} = \left\{R_1^{t-\Delta t}, R_2^{t-\Delta t}, \cdots, R_n^{t-\Delta t}\right\}$, and each $R_i^{t-\Delta t}$ has suggested N locations to user u at time $t - \Delta t$. New data during the period $(t - \Delta t, t]$ are used to evaluate the quality of each $R_i^{t-\Delta t}$ with regard to user u. This evaluation eventually results in weights $\alpha_u^t(i)$, which are then used to guide the aggregation process.

As user check-in data are implicit, we utilize a *pairwise* methodology for preference learning. Consider a user u and her visiting history up to time t, L_u^t; let $L_+^t = L_u^t \setminus L_u^{t-\Delta t}$ and $L_-^t = \{\ell | \ell \notin L_u^t\}$. For two locations $\ell \in L_+^t$ and $\ell' \in L_-^t$, to some extent it is reasonable to assume that u prefers ℓ to ℓ' (denoted as $\ell >_u \ell'$). Clearly, a good recommendation should be highly consistent with $>_u$. This means, for the linear aggregation of Formula 1, pairs $(\ell, \ell') \in P_u^t = L_+^t \times L_-^t$ can be used to tune the weights $\alpha_u^t(i)$. We consider maximizing the *likelihood* $\Pr\left\{P_u^t \middle| \boldsymbol{\alpha}_u^t\right\}$, which is the probability of observing P_u^t given the preference $\boldsymbol{\alpha}_u^t$. Assuming that pairs $(\ell, \ell') \in P_u^t$ are independent, then

$$\Pr\left\{P_u^t \middle| \boldsymbol{\alpha}_u^t\right\} = \prod_{(\ell,\ell') \in P_u^t} \Pr\left\{\ell >_u \ell' \middle| \boldsymbol{\alpha}_u^t\right\}.$$

Computing $\Pr\{\ell >_u \ell' | \boldsymbol{\alpha}_u\}$ is nontrivial, but an intuition is that this probability should be proportional to the difference between scores $s(u, \ell; \boldsymbol{\alpha}_u)$ and $s(u, \ell'; \boldsymbol{\alpha}_u)$: the larger the difference $s(u, \ell; \boldsymbol{\alpha}_u) - s(u, \ell'; \boldsymbol{\alpha}_u)$, the more confident we will be on concluding $\ell >_u \ell'$. Based on this idea, we set

$$\Pr\left\{\ell >_u \ell' \middle| \boldsymbol{\alpha}_u\right\} = \sigma\left(d_u^{\ell,\ell'}(\boldsymbol{\alpha}_u)\right),$$

where σ is the *logistic function* that can generate a probability distribution in the range of [0,1], $\sigma(x) = \frac{1}{1+e^{-x}}$, and $d_u^{\ell,\ell'}(\boldsymbol{\alpha}_u) = s(u, \ell; \boldsymbol{\alpha}_u) - s(u, \ell'; \boldsymbol{\alpha}_u)$. For the ease of algorithm design, the actual objective function being optimized is

$$\ln\left(\Pr\left\{P_u^t \middle| \boldsymbol{\alpha}_u\right\}\right) = \sum_{(\ell,\ell') \in P_u^t} \ln\left(\sigma\left(d_u^{\ell,\ell'}(\boldsymbol{\alpha}_u)\right)\right).$$

The learning approach (LEARNPREFERENCE) for finding $\boldsymbol{\alpha}_u^t$ based on $\boldsymbol{\alpha}_u^{t-\Delta t}$ uses a *stochastic gradient descent* technique, as shown in Algorithm 2. Since the training space P_u^t is large, samples are used for learning (Lines 3-5). The gradient descent update in Line 5 is done with the following formula:

Algorithm 2 LEARNPREFERENCE$(P_u^t, \boldsymbol{\alpha}_u^{t-\Delta t}, M, K)$

Input: $P_u^t = L_+^t \times L_-^t$; $\boldsymbol{\alpha}_u^{t-\Delta t}$; number of iterations M; number of samples K
Output: updated user preference $\boldsymbol{\alpha}_u^t$

1: **for** $j = 1, 2, \cdots, M$ **do**
2: $\quad \boldsymbol{\alpha}^{(j)} \leftarrow \boldsymbol{\alpha}_u^{t-\Delta t}$
3: $\quad$ Draw K sample pairs from the training space P_u^t
4: $\quad$ **for each** sample pair $p = (\ell, \ell')$ **do**
5: $\quad\quad \boldsymbol{\alpha}^{(j)} \leftarrow$ update with p and $\boldsymbol{\alpha}^{(j)}$ (gradient descent)
6: $\boldsymbol{\alpha}_u^t \leftarrow \frac{1}{M}\sum_{j=1}^{M} \boldsymbol{\alpha}^{(j)}$

$$\boldsymbol{\alpha}^{(j)} \leftarrow (1 - \gamma) \cdot \boldsymbol{\alpha}^{(j)} + \gamma \cdot \tau \cdot \left(\frac{\nabla_{\boldsymbol{\alpha}^{(j)}} d_u^{\ell,\ell'}\left(\boldsymbol{\alpha}^{(j)}\right)}{1 + e^{d_u^{\ell,\ell'}}}\right), \quad (2)$$

where $\gamma \in (0, 1)$ is the *learning rate*, τ the *step size*, and $\nabla_{\boldsymbol{\alpha}^{(j)}} d_u^{\ell,\ell'}$ the gradient:

$$\nabla_{\boldsymbol{\alpha}^{(j)}} d_u^{\ell,\ell'} = \begin{bmatrix} \varphi\left(R_1^{t-\Delta t}(u, \ell)\right) - \varphi\left(R_1^{t-\Delta t}(u, \ell')\right) \\ \varphi\left(R_2^{t-\Delta t}(u, \ell)\right) - \varphi\left(R_2^{t-\Delta t}(u, \ell')\right) \\ \vdots \\ \varphi\left(R_n^{t-\Delta t}(u, \ell)\right) - \varphi\left(R_n^{t-\Delta t}(u, \ell')\right) \end{bmatrix}. \quad (3)$$

After M independent iterations, Algorithm 2 terminates with a personalized weight $\boldsymbol{\alpha}_u^t$ (Line 6).

The quality of samples (Line 3) are of essential importance in Algorithm 2. Intuitively, if a sampled pair (ℓ, ℓ') is such that $\varphi\left(R_i^{t-\Delta t}(u, \ell)\right) \approx \varphi\left(R_i^{t-\Delta t}(u, \ell')\right)$ for all $R_i^{t-\Delta t}$ (i.e., no recommender has distinct preference over ℓ and ℓ'), then it does not provide much information for learning (i.e., $\nabla_{\boldsymbol{\alpha}^{(j)}} d_u^{\ell,\ell'} \approx \mathbf{0}$). Therefore, we use the strategies proposed by Rendle and Freudenthaler [19] to sample *informative* pairs. Since the size of L_+^t is typically small, the number of samples K is usually proportional to $|L_+^t|$, e.g., $K = K_0 \cdot |L_+^t|$ for some integer $K_0 > 1$. Therefore, sampling in L_+^t is to simply sample each $\ell \in L_+^t$ uniformly

[3] It is worth mentioning that a trivial implementation of LURA is to put equal weights on component recommenders. This, however, usually leads to very bad recommendations due to the diversities we studied in Section 2 (Figures 1-2). Indeed, the essence of learning is to identify those good recommenders out of a large population of bad ones, with regard to some individual user.

at random (i.e., K_0 times on average), and strategies for sampling informative pairs in fact focus on the set L^t_-.

Random sampling (RS). This is the basic strategy. Locations in L^t_- are selected uniformly at random, i.e.,

$$\Pr\left\{\ell' \,|u\right\} = \frac{1}{|L^t_-|}.$$

Static sampling (SS). This strategy favors popular locations, i.e., locations with many visitors have higher chances to be selected. Specifically,

$$\Pr\left\{\ell' \,|u\right\} \propto \exp\left(-\frac{\text{rank}(\ell')}{\lambda}\right), \lambda > 0,$$

where rank$(\cdot)$ is the "1234" ranking of L^t_- by check-in frequencies; smaller numbers are assigned to more popular locations.

Adaptive sampling (AS). When sampling, this strategy gives higher chances to those locations with higher scores (i.e., locations considered as promising by the recommender):

$$\Pr\left\{\ell' \,|u\right\} \propto \exp\left(-\frac{\text{rank}(u,\ell')}{\lambda}\right), \lambda > 0,$$

where rank(u, ℓ') is the "1234" ranking of ℓ' based on the score $s^{t-\Delta t}(u, \ell')$. Such promising yet unvisited locations may be more informative in identifying a user's preference.

3.2 Recommendation Aggregation

The last step in our LURA framework is to aggregate the recommendations from all the component recommenders, and then provide the user u with a final list of N items (Formula 1). We consider the following two different aggregation strategies.

Score-based aggregation (SA). This strategy is to use the scaled score of user-item pair (u, ℓ) estimated by each recommender (at time t). $\varphi\left(R^t_i(u,\ell)\right)$ is defined as

$$\varphi\left(R^t_i(u,\ell)\right) = \frac{R^t_i(u,\ell)}{\max_{\ell' \in L^t_-} R^t_i(u,\ell')}, \quad i = 1, 2, \cdots, n.$$

Rank-based aggregation (RA). This strategy considers the ranked position of a location ℓ. Given a ranked list of N locations, $\ell_1, \ell_2, \cdots, \ell_N$, this strategy assigns higher scores to top locations. In particular, $\varphi(\cdot)$ is defined as

$$\varphi\left(R^t_i(u,\ell)\right) = 1 - \frac{1}{N}\left(\text{rank}_i(u,\ell) - 1\right), \quad i = 1, 2, \cdots, n,$$

where rank$_i(u, \ell)$ is the ranked position of a location ℓ in the recommendation list provided to u by R^t_i. This strategy is a common variant of *Borda count* [8].

4. EXPERIMENTS

In this section, we evaluate LURA on Foursquare and Gowalla datasets. In Section 4.1 we explain the experiment setup as well as the evaluation metrics. Then, in Section 4.2 we study how different implementations, i.e., different sampling strategies (Section 3.1) and aggregation strategies (Section 3.2), affect the performance of LURA. After that, in Sections 4.3 and 4.4, we compare the best implementations of LURA with (i) their own component recommenders, and (ii) other advanced recommenders, respectively.

4.1 Setup

The experiments involve three time periods: (i) a *base* period $T^t_{\text{base}} = [1, t-\Delta t]$, in which the data are used for constructing component recommenders; (ii) a *learning* period $T^t_{\text{learn}} = (t - \Delta t, t]$, for learning user preferences, updating component recommenders, and constructing a LURA recommender; and (iii) a *testing* period $T^t_{\text{test}} = (t, t + \Delta t]$ for evaluating the recommenders. Competitor methods that do not require any learning of user preferences are built using all data in $T^t_{\text{base}} \cup T^t_{\text{learn}} = [1, t]$, so that any comparison between them and LURA is fair.

To evaluate the performance of a recommender R^t, for each user u we compare the top-N recommendation $R^t(u) = \{\ell_1, \ell_2, \cdots, \ell_N\}$ with $L^{(t,t+\Delta t]}_u$, the actual set of locations visited by u during the testing period T^t_{test}. In the following, we use $t = 300$ for Foursquare and $t = 420$ for Gowalla; Δt is set to 60 for both datasets. We omit the experiments on evolving t due to the lack of space. All the results at different t are similar, given sufficient data in period $(t - \Delta t, t]$.

Evaluation metrics. Two metrics are commonly used for location recommendation: Precision@N and Recall@N [26]. Let $H^t_u = \left|R^t(u) \cap L^{(t,t+\Delta t]}_u\right|$ be the number of correctly predicted locations with regard to user u (i.e., the number of successfully predicted locations). Then,

$$\text{Precision@}N = \frac{\sum_{u \in \mathcal{A}} H^t_u}{N \cdot |\mathcal{A}|},$$

$$\text{Recall@}N = \frac{\sum_{u \in \mathcal{A}} H^t_u}{\sum_{u \in \mathcal{A}} \left|L^{(t,t+\Delta t]}_u\right|}.$$

where $\mathcal{A}$ is the set of *active users*. For a user u to be active, she must (i) visit at least 5 new locations in the learning period T^t_{learn} so that LURA can infer her preference, and (ii) visit at least 1 new location in the testing period T^t_{test} so that the evaluation is nontrivial.

4.2 Different Implementations of LURA

We first study different implementations of LURA, i.e., we compare different sampling and aggregation strategies presented in Sections 3.1 and 3.2, and see how they affect the performance of LURA. We have 3 sampling strategies and 2 aggregation strategies, thus in total we have 6 different implementations of LURA.

Figure 3 shows the comparison result among these 6 implementations with respect to varying N. As we can see, the performance of different implementations are nearly the same, with adaptive sampling (AS) being slightly better than the other two sampling strategies in most of the cases. On the other hand, we observe that the performance of aggregation strategies is data-dependent: on Foursquare, rank-based aggregation (RA) is slightly better than score-based aggregation (SA) while on Gowalla it is the other way round. This could be due to the fact that Gowalla has relatively more data for learning user preferences (29,996 check-ins for 1,307 active users, comparing to 38,688 check-ins for 3,942 active users in Foursquare), and therefore the final scores are more accurate.

For both Foursquare and Gowalla, precision and recall values are all at the level of 1%-10%. Such performance of location recommendation is due to *data sparsity*, i.e., out of a huge collection of locations (96,002 in Foursquare and 767,936 in Gowalla) most people merely visit several to dozens of them.

4.3 Comparing with the 11 Component Recommenders

We compare LURA with its 11 component recommenders. We use the best sampling and aggregation strategies for LURA, i.e., we use LURA-ASSA and LURA-ASRA based on the findings in Section 4.2. The comparison results are shown in Figure 4.

Both LURA-ASSA and LURA-ASRA outperform all the other methods in all cases. This justifies that combining different recommenders may provide better recommendations. Among the 11

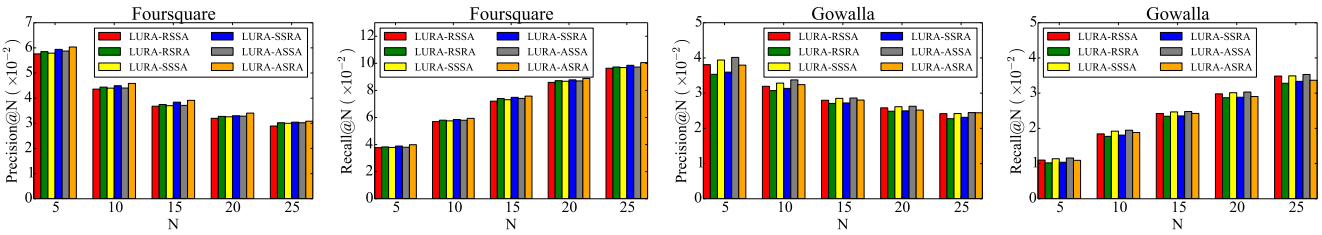


Figure 3: Comparing different implementations of LURA

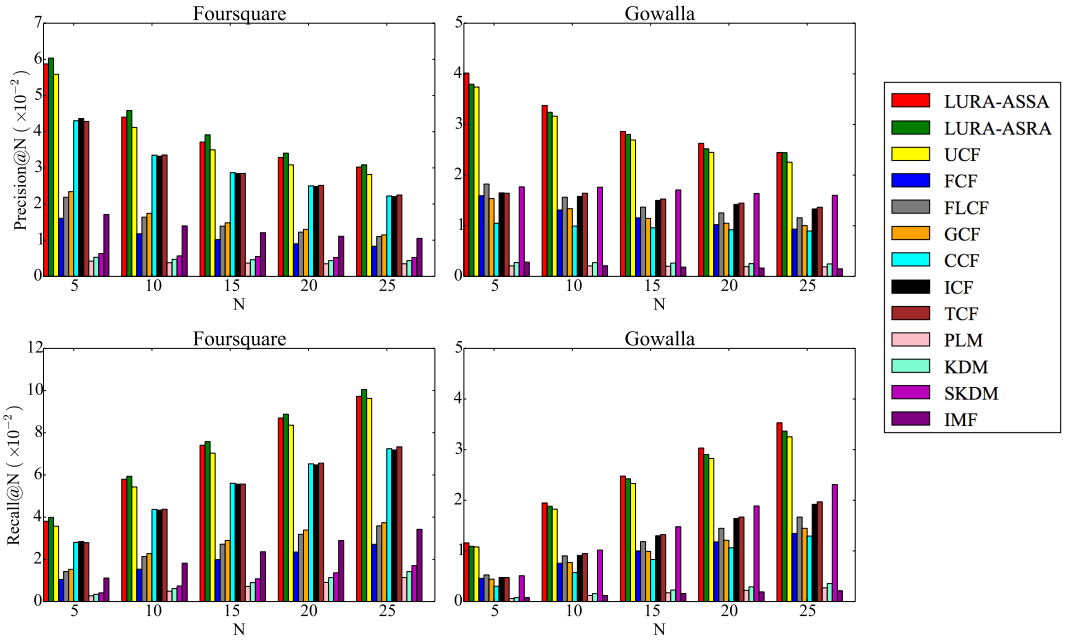


Figure 4: Comparing LURA with its 11 component recommenders

component recommenders of LURA, UCF always performs the best. This means that the behaviors of most users are best reflected by other similar users. Nonetheless, comparing to UCF, LURA can be better in Foursqure by up to 11.82% in precision and 11.72% in recall; in Gowalla these numbers are 8.53% and 8.52% respectively.

Considering the small absolute values of precision and recall, we also carry out tests of statistical significance. Specifically, we run a paired t-test between the numbers of hits of LURA and UCF, the best-performing component recommender. The p-values are both less than 0.01 (2.99×10^{-4} for Foursquare and 1.79×10^{-4} for Gowalla), indicating that the improvement of LURA over UCF is statistically significant.

4.4 Comparison with Other Methods

We also compare LURA with other existing methods that adopt similar ideas to what we use for LURA.

USG [26]. This is a linear combination of UCF, FLCF, and PLM, i.e., the component recommenders R_1, R_3, and R_8 of LURA.

$$\begin{aligned}\mathrm{USG}(u,\ell) \;=\; & (1-\alpha-\beta)\cdot\varphi\left(\mathrm{R}_1(u,\ell)\right)\\ & +\alpha\cdot\varphi\left(\mathrm{R}_3(u,\ell)\right)+\beta\cdot\varphi\left(\mathrm{R}_8(u,\ell)\right),\end{aligned}$$

where α and β are parameters, and $\varphi(\cdot)$ is a rescale of scores as what we use for LURA (Section 3.2). To construct USG at time t, we do as what [26] has proposed: we randomly select 70% of the data before time t to construct the three component recommenders: UCF, FLCF, and PLM, and then use the rest 30% to determine parameters α and β. The final parameters are $\alpha = \beta = 0.1$ for Foursquare, and $\alpha = 0.1$, $\beta = 0.2$ for Gowalla. These parameters are also consistent with what [26] has suggested. Note that the parameter settings are the same for all users, i.e., not personalized.

iGLSR [30]. This method combines GCF and KDM, i.e., the component recommenders R_4 and R_9 of LURA, having a clear focus on the geographical relationships between locations. Different from USG and LURA which aggregate scores linearly, iGLSR adopts the following combination:

$$\mathrm{iGLSR}(u,\ell) = \varphi\left(\mathrm{R}_4(u,\ell)\right)\cdot\varphi\left(\mathrm{R}_9(u,\ell)\right),$$

where $\varphi(\cdot)$ is as in USG and LURA.

RankBoost [9]. This is a method to combine a given set of *weak* recommenders. For a target user u, the training data for RankBoost is $P_u^t = L_+^t \times L_-^t$, as what is fed to LURA (Section 3.1). RankBoost takes K iterations to get a "boosted" recommender. During the k-th iteration, it aims at maximizing the discrimination between L_+^t and L_-^t,

$$Z_k = \sum_{(\ell,\ell')\in P_u^t} D_k(\ell,\ell') e^{\alpha_k\cdot\left(\varphi(h_k(u,\ell))-\varphi(h_k(u,\ell'))\right)},$$

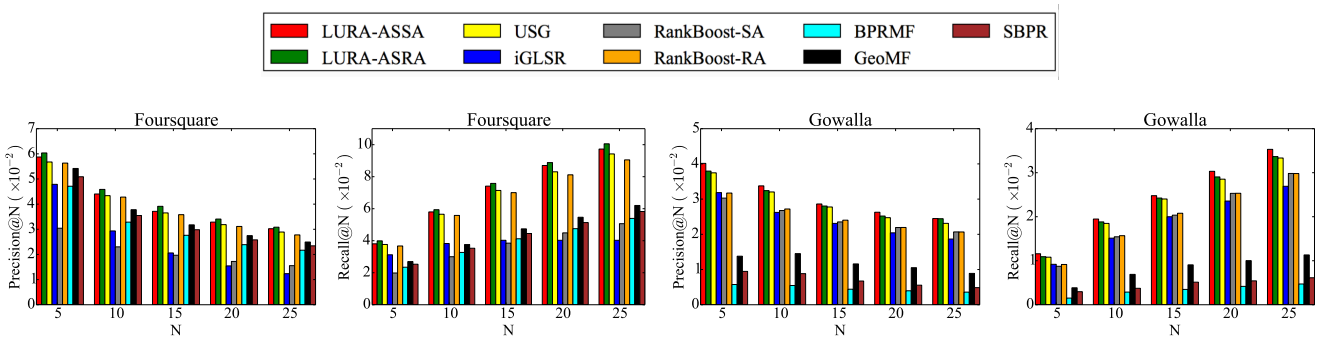


Figure 5: Comparing LURA with other existing methods

where D_k is a weight distribution over location pairs in P_u^t, measuring how important the pair (ℓ, ℓ') currently is; $\varphi(\cdot)$ is for scaling recommendation scores as introduced in Section 3.2; and h_k is the selected weak recommender during the k-th iteration. While pursuing this goal, RankBoost estimates a weight β_k for the selected recommender h_k and updates the distribution D_k. Eventually, recommendations are made based on the following scoring function:

$$\text{RankBoost}(u, \ell) = \sum_{k=1}^{K} \beta_k \cdot \varphi\left(h_k(u, \ell)\right).$$

Similar to LURA, RankBoost also has two options for $\varphi(\cdot)$: score-based (SA) and rank-based (RA). In our experiments we consider both options, i.e., we have RankBoost-SA and RankBoost-RA as two different implementations of RankBoost.

BPRMF [20]. This method also takes a pairwise view of locations, i.e., it considers $(\ell, \ell') \in P_u^t$ as training samples. Given the check-in frequency matrix at time t, $\boldsymbol{C}^t$, BPRMF manages to maximize the posterior probability $\Pr\left\{\Theta \middle| P_u^t\right\}$, where Θ is a *completion* of the matrix $\boldsymbol{C}^t$ (i.e., Θ has an estimation on every unknown (u, ℓ) entries in $\boldsymbol{C}^t$). BPRMF transforms this posterior probability using the Bayes' theorem, and optimizes $\Pr\left\{P_u^t | \Theta\right\} \Pr\{\Theta\}$ using a stochastic gradient descent method.

SBPR [31]. SBPR integrates social information into BPRMF. It divides items into three disjoint sets : positive items I^+, negative items I^-, and social items I_S. It assumes that items in I^+ are more preferable than those in I_S, while the ones I^- are the least preferable. Two partial orders are thus used in a BPR-like framework.

GeoMF [16]. GeoMF integrates geographical information into weighted MF methods [12]. Specifically, it assumes that a location may have a Gaussian impact in its neighboring area, which is considered as weights in an MF framework.

The comparison results are shown in Figure 5. USG and RankBoost consistently perform the best among methods other than LURA. LURA-ASSA and LURA-ASRA are clearly superior to USG and RankBoost. In Foursquare, LURA can outperform the best of USG and RankBoost by up to 7.16% in precision and 6.22% in recall; in Gowalla, these numbers are 7.35% and 8.12% respectively. Compared to USG which adopts unified parameters for all users, the performance improvement of LURA comes from its awareness of users' preferences over recommenders.

Similar to what we have done in Section 4.3, we also did a paired t-test for the numbers of hits of LURA and USG, the best-performing competitor. The p-values are both less than 0.05 (0.021 for Foursquare and 0.005 for Gowalla), implying that LURA's improvement over existing methods is also statistically significant.

5. RELATED WORK

5.1 Other POI Recommendation Methods

GPS data from mobile services are also used for location recommendation. Zhang et al. [33] analyzed GPS trajectories to discover points of interests for recommendation. Zhang et al. [32] studied *location-and-activity* recommendation using GPS data, where activities could be various human behaviors such as shopping, watching movies, etc. Leung et al. [15] proposed a co-clustering framework for location recommendation based on user-activity-location tripartite graphs. Some recent studies view location recommendation problem from a perspective of *topic modeling* (e.g., [17, 27]). In general, with additional information such as location contents, user-provided tips, etc., these methods build topic models for users and locations, based on which the likelihood of a user visiting a location can be estimated. In addition, Yuan et al. [29] considered *time-dependent* location recommendation, arguing that the locations recommended to a user at lunch time should be different from the ones recommended at leisure time.

5.2 Recommender Ensemble

Ensemble techniques have been used to combine several simple recommenders to form a stronger one. To name a few, Bar et al. [3] studied different ways to combine simple CF methods, including bagging, boosting, fusion, and randomness injection; Schclar et al. [22] considered AdaBoost regression on a neighbor-based CF method. Experiments on movie ratings data showed that these ensemble methods had improved performance. Besides, Tiemann and Pauws [24] considered ensembles of content-based methods, which succeeded in recommending music and news.

These ensemble methods attempt to minimize the *root mean square error* (RMSE) on ratings data. However, as we have emphasized earlier in this paper, check-in data are implicit, and thus pursuing numerical estimations of check-in frequencies will essentially be a distraction from the goal of location recommendation.

5.3 (Personalized) Learning to Rank

The problem of *learning to rank* is to determine a ranking among a set of items. Given some training data (e.g., observed user click-throughs that imply preference over search results), the problem is to find a ranking function that best agree with the training data, which can be done by machine learning techniques (e.g., [4, 5, 13]). Recently, Wang et al. [25] and Song et al. [23] studied *personalized* learning to rank, where personalized rankings were adapted from a global user-independent ranking.

The problem of (personalized) learning to rank is related to our problem in general, in that they both pursue the best ranking of

items. However, our problem is to infer user preference over different recommenders, which are themselves ranking functions over items (locations); the above-mentioned learning to rank techniques are thus not directly applicable to solve our problem.

6. CONCLUSION

In this paper we study the problem of location recommendation. We first investigate two real-life LBSN datasets, discovering diversities in (i) recommendations generated by representative recommenders, and (ii) user preferences over the recommenders. Based on these observations, we consider personalized location recommendation by inferring user preferences over multiple recommenders. We propose a LURA framework to achieve our goal and test it with extensive experiments. The results show that LURA achieves significant performance improvements over existing recommendation methods. In the future, we plan to investigate other possible ways to aggregate recommenders.

7. ACKNOWLEDGMENTS

The authors would like to acknowledge the support from Hong Kong RGC (Grant No. HKU715413E) and the National Natural Science Foundation of China (Grant No. 61432008).

8. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *TKDE*, 17(6):734–749, 2005.

[2] J. Bao, Y. Zheng, and M. F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *SIGSPATIAL GIS*, 2012.

[3] A. Bar, L. Rokach, G. Shani, B. Shapira, and A. Schclar. Improving simple collaborative filtering models using ensemble methods. In *MCS*, 2013.

[4] C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, 2007.

[5] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML*, 2005.

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

[7] Y. Ding and X. Li. Time weight collaborative filtering. In *CIKM*, 2005.

[8] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, 2001.

[9] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *JMLR*, 4:933–969, 2003.

[10] H. Gao, J. Tang, and H. Liu. gSCorr: Modeling geo-social correlations for new check-ins on location-based social networks. In *CIKM*, 2012.

[11] M. Halvey, P. Punitha, D. Hannah, R. Villa, F. Hopfgartner, A. Goyal, and J. M. Jose. Diversity, assortment, dissimilarity, variety: A study of diversity measures using low level features for video retrieval. In *ECIR*, 2009.

[12] Y. Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *ICDM*, 2008.

[13] T. Joachims. Optimizing search engines using clickthrough data. In *KDD*, 2002.

[14] I. Konstas, V. Stathopoulos, and J. M. Jose. On social networks and collaborative recommendation. In *SIGIR*, 2009.

[15] K. W.-T. Leung, D. L. Lee, and W.-C. Lee. CLR: A collaborative location recommendation framework based on co-clustering. In *SIGIR*, 2011.

[16] 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 *KDD*, 2014.

[17] B. Liu, Y. Fu, Z. Yao, and H. Xiong. Learning geographical preferences for point-of-interest recommendation. In *KDD*, 2013.

[18] X. Liu, Y. Liu, K. Aberer, and C. Miao. Personalized point-of-interest recommendation by mining users' preference transition. In *CIKM*, 2013.

[19] S. Rendle and C. Freudenthaler. Improving pairwise learning for item recommendation from implicit feedback. In *WSDM*, 2014.

[20] S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *UAI*, 2009.

[21] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, 2001.

[22] A. Schclar, A. Tsikinovsky, L. Rokach, A. Meisels, and L. Antwarg. Ensemble methods for improving the performance of neighborhood-based collaborative filtering. In *RecSys*, 2009.

[23] Y. Song, H. Wang, and X. He. Adapting deep RankNet for personalized search. In *WSDM*, 2014.

[24] M. Tiemann and S. Pauws. Towards ensemble learning for hybrid music recommendation. In *RecSys*, 2007.

[25] H. Wang, X. He, M.-W. Chang, Y. Song, R. W. White, and W. Chu. Personalized ranking model adaptation for web search. In *SIGIR*, 2013.

[26] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *SIGIR*, 2011.

[27] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen. LCARS: A location-content-aware recommender system. In *KDD*, 2013.

[28] J. J.-C. Ying, E. H.-C. Lu, W.-N. Kuo, and V. S. Tseng. Urban point-of-interest recommendation by mining user check-in behaviors. In *UrbComp*, 2012.

[29] Q. Yuan, G. Cong, Z. Ma, A. Sun, and N. M. Thalmann. Time-aware point-of-interest recommendation. In *SIGIR*, 2013.

[30] J.-D. Zhang and C.-Y. Chow. iGSLR: Personalized geo-social location recommendation - a kernel density estimation approach. In *SIGSPATIAL GIS*, 2013.

[31] T. Zhao, J. J. McAuley, and I. King. Leveraging social connections to improve personalized ranking for collaborative filtering. In *CIKM*, 2014.

[32] V. W. Zheng, Y. Zheng, X. Xie, and Q. Yang. Collaborative location and activity recommendations with GPS history data. In *WWW*, 2010.

[33] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *WWW*, 2009.