Localization Attacks Using Matrix and Tensor Factorization

Takao Murakami*, Member, IEEE, and Hajime Watanabe

Abstract—It is known that various types of location privacy attacks can be carried out using a personalized transition matrix that is learned for each target user, or a population transition matrix that is common to all target users. However, since many users disclose only a small amount of location information in their daily lives, the training data can be extremely sparse. The aim of this paper is to clarify the risk of location privacy attacks in this realistic situation. To achieve this aim, we propose a learning method that uses tensor factorization (or matrix factorization) to accurately estimate personalized transition matrices (or a population transition matrix) from a small amount of training data. To avoid the difficulty in directly factorizing the personalized transition matrices (or population transition matrix), our learning method first factorizes a transition count tensor (or matrix), whose elements are the number of transition counts that the user has made, and then normalizes counts to probabilities. We focus on a localization attack, which derives an actual location of a user at a given time instant from an obfuscated trace, and compare our learning method with the ML (Maximum Likelihood) estimation method in both the personalized matrix mode and the population matrix mode. The experimental results using four real datasets show that the ML estimation method performs only as well as a random guess in many cases, while our learning method significantly outperforms the ML estimation method in all of the four datasets.

Index Terms—location privacy, localization attack, Markov Chain, tensor factorization, matrix factorization.

I. INTRODUCTION

W

ith the widespread use of GPS-equipped devices such as smartphones and in-car navigation systems, people are increasingly using various types of LBS (Location-based Services) such as mapping, route finding, and POI (Point-of-Interest) search. Many people also disclose their current location using geosocial services such as location check-in, tagging, and nearby friends [2], [3]. A recent survey in the U.S. reports that 74% of smartphone owners use LBS to obtain real-time location-based information, and 18% of smartphone owners use geosocial services such as location check-in [4]. Nowadays, a great deal of information about users’ locations or traces (time-series location trails) is stored in a data center, which is called Spatial Big Data [5]. This data is expected to help identify routes that avoid congestion or reduce fuel consumption.

A preliminary version of this paper was presented at the First IEEE International Workshop on Big Data Security and Privacy (BDSP 2014), Washington DC, USA, October, 2014 [1].

T. Murakami is with Information Technology Research Institute (ITRI), National Institute of Advanced Industrial Science and Technology (AIST), Tokyo, 135-0064, Japan (e-mail: takao-murakami@aist.go.jp).

H. Watanabe is with Information Technology Research Institute (ITRI), National Institute of Advanced Industrial Science and Technology (AIST), Tsukuba, 305-8568, Japan (e-mail: h-watanabe@aist.go.jp).

It is frequently argued that while these services can provide significant benefits to users and society, the disclosure of location information raises serious privacy concerns. For example, the disclosure of a user’s location can lead to inference of sensitive locations such as homes and hospitals that he/she regularly visits. It can also expose other private information such as a user’s social relationship [6], [7], activities (e.g. working, shopping, sleeping) [8], [9], and long-term properties (e.g. age, gender, work role) [9], [10]. There is even a danger that a stalker exploits location information of the target user [11]. This kind of privacy is referred to as location privacy, and numerous studies have addressed this issue, mainly from the perspectives of attacks, defenses, and privacy metrics [12]–[14] (as described in detail in Section VII).

One of the most popular approaches to location privacy attacks is based on a Markov Chain model [15]–[21]. Some studies showed that this model works very well through comparison experiments [15], [18]. This approach first converts locations and time into discrete values by dividing an area, in which users can move, into M distinct regions $x_1, \cdots, x_M$ (or extracting M frequently visited POIs) (see the left side of Fig. 1) and partitioning time at a fixed interval (e.g. 20 minutes, 1 hour). It then assumes a Markov model for users’ behavior and computes an $M \times M$ transition matrix, which comprises the probability that a user moves from region $x_i$ to $x_j$ ($1 \leq i,j \leq M$). Based on this matrix, the adversary can carry out a location prediction attack [16], [17], which predicts a future location (i.e. region ID) of a target user from a past location that he/she disclosed. The adversary can also carry out a more general attack called a localization attack [21], which derives an actual location of a target user from his/her trace that is obfuscated (e.g. by adding noise, merging regions, deleting some locations).

The adversary can derive the actual location using either a personalized transition matrix that is learned for each target user or a population transition matrix that is common to all target users (see the middle/right side of Fig. 1). In this paper, we refer to the former case as a personalized matrix mode, and the latter as a population matrix mode. The personalized matrix mode has the advantage that the adversary can exploit unique features of each user’s behavior (e.g. favorite places or routes, walking speed) for deriving the actual location. It also enables a de-anonymization attack [18]–[21], which identifies

1\textsuperscript{1} It should be noted that location prediction can be used to provide useful information services (e.g. recommendation of POIs, targeted advertisements). Indeed, many papers studied location prediction to this end [15], [16]. However, they can cause privacy issues when they are used by a malicious adversary, as discussed in [16], [17].
Many users, however, disclose only a small amount of location information in their daily lives (e.g. a few times per day). Although the studies in [18]–[20] computed each personalized transition matrix using the ML (Maximum Likelihood) estimation method, the available training data can be extremely sparse in such cases\(^2\). We explain the problem arising from sparse training data using the example shown in Fig. 2. In this example, the training data of user \(u_1\) consists of only 3 traces, each of which is composed of 4 locations (\(M = 5\)). Since the transition from region \(x_1\) to \(x_2\) is not observed in this trace, the transition probabilities from region \(x_1\) or \(x_2\) cannot be estimated using the ML estimation method. We refer to the corresponding elements, which are marked with "?" in Fig. 2, as unobserved elements. Also, the remaining elements are overfitted to the training traces (e.g. the transition probabilities from region \(x_3\) are either 0 or 1), which will generally result in poor predictive performance (i.e. the overfitting problem [22]). We refer to such elements as overfitted elements. If there are many unobserved elements or overfitted elements, the adversary cannot accurately perform location privacy attacks.

The population matrix mode has an advantage over the personalized matrix mode in terms of the sparse data problem, since the adversary can use the training data of all target users to learn one matrix (i.e. the population transition matrix). However, even the population matrix mode might also suffer from the sparse data problem when the number of regions \(M\) is large, since the number of elements in the population matrix is \(M^2\) (i.e. it increases quadratically). Despite numerous studies in the field of location privacy, no studies have attempted to solve this sparse data problem, to the best of our knowledge.

The aim of this paper is to clarify the risk of location privacy attacks in a realistic situation where the amount of training data is limited. To achieve this aim, we propose a learning method that solves the sparse data problem for both the personalized matrix mode and the population matrix mode. We apply our learning method to the localization attack [21], and show its effectiveness using four real datasets: the Geolife dataset [23], the Gowalla dataset [24], the CRAWDAD (epfl/mobility) dataset [25], and the CRAWDAD (roma/taxi) dataset [26].

### A. Our Contributions

The main contributions of our work are as follows:

1) We propose a learning method that regards a set of personalized transition matrices as a 3rd-order tensor and uses tensor factorization [27]–[31] (we denote this “factorization-based” learning method by “FC”). Here, the summation of transition probabilities over destinations is always 1, and it is intractable to directly factorize the personalized transition matrices under this constraint. To avoid this problem, our learning method first factorizes a transition count tensor, whose elements are the number of transitions that the user has made, and then normalizes counts to probabilities. By doing so, the transition matrices can be accurately estimated from a small amount of training data (Section IV-A).

2) We apply our learning method to the localization attack [21], and provide a detail comparison with the ML (Maximum Likelihood) estimation method [18]–[20] as follows. We first extend our learning method explained above to the population matrix mode by using matrix factorization [30]–[32] (we denote this method by “FC\(*\)” (Section IV-B). We then compare our learning method with the ML estimation method in both the personalized matrix mode and the population matrix mode using four real datasets. The results show that the ML estimation method performs only as well as a random guess in many cases, while our learning method significantly outperforms the ML estimation method in all of the four datasets. This means that an attack based on the ML estimation method does not pose a threat in a “realistic” situation where the amount of training data is limited, while an attack based on our learning method is a threat even in this situation (Section V). We also show that FC can capture unique features of each user’s behavior by visualizing parameters in FC and the corresponding traces (Section VI).

This paper is an extension of a previously published conference paper [1]. Note that while the first contribution above is covered in [1], the second contribution is new. More specifically, in [1] we applied FC to the location prediction attack [16], [17], and showed that it outperformed the ML estimation method in the personalized matrix mode. In [33], we also compared these methods when applied to the de-anonymization attack [18]–[21]. However, since both [1] and

---

\(^2\)Although Shokri et al. [21] proposed a learning method that considers the case where some locations in training traces are missing, it provides almost the same performance as the ML estimation method when there is no missing locations, as described in Section II-C in details.
[33] evaluated these methods only in the personalized matrix mode (i.e. they did not evaluate them in the population matrix mode), it remained unclear whether our learning method is really effective, compared to the ML estimation method. Also, both [1] and [33] conducted experiments using only a single dataset. In this paper, we provide a detail comparison between our learning method and the ML estimation method in both the personalized matrix mode and the population matrix mode using four real datasets.

B. Paper Organization

The rest of this paper is organized as follows. In Section I-C, we define the basic notation used throughout this paper. In Section II, we describe the localization attack and the sparse data problem. In Section III, we explain matrix/tensor factorization. In Section IV, we propose a learning method using tensor factorization to solve the sparse data problem. We also extend this learning method to the population matrix mode by using matrix factorization. In Section V, we show the experimental results. In Section VI, we visualize parameters in tensor factorization and the corresponding traces. In Section VII, we survey the previous work related to ours. Finally, we conclude this paper in Section VIII.

C. Notation

Let $\mathcal{U} = \{u_1, \cdots, u_N\}$ be a set of $N$ targeted users, and $\mathcal{X} = \{x_1, \cdots, x_M\}$ be a set of $M$ regions (or POIs). We assume that time is discrete, and express time instants as integers (i.e. a set of time instants is $\mathbb{Z} = \{-2, -1, 0, 1, 2, \cdots\}$). Let $P_n$ be an $M \times M$ transition matrix of user $u_n \in \mathcal{U}$, and $p_{n,i,j}$ be its $(i,j)$-th element (i.e. the probability that user $u_n \in \mathcal{U}$ in region $x_i \in \mathcal{X}$ will move to region $x_j \in \mathcal{X}$ in the next time instant). Let further $P_g$ be an $M \times M$ population transition matrix, and $p_{g,i,j}$ be its $(i,j)$-th element (i.e. the probability that an average user in region $x_i \in \mathcal{X}$ will move to region $x_j \in \mathcal{X}$). We also denote a set of natural numbers less than or equal to $N$ by $[N] = \{1, 2, \cdots, N\}$.

In Sections II-A, II-B, II-C, and IV-A, we focus on the personalized matrix mode, in which the adversary uses personalized transition matrices $\{P_n|n \in [N]\}$. In Sections II-D and IV-B, we describe the population matrix mode, in which he/she uses a population transition matrix $P_g$.

II. LOCALIZATION ATTACKS AND SPARSE DATA PROBLEMS

In this section, we begin by describing a framework for localization attacks based on the location-privacy framework introduced by Shokri et al. [21] (Section II-A). We then describe an algorithm for localization attacks (Section II-B), and the sparse data problem (Section II-C). Since we focus on the personalized matrix mode from Sections II-A to II-C, we finally explain the case of the population matrix mode (Section II-D).

A. Framework for Localization Attacks

Fig. 3 shows a framework for localization attacks. This framework consists of the following two phases: training phase and localization phase. In the training phase, the adversary learns personalized transition matrices $\{P_n|n \in [N]\}$ using training traces. For training traces, we make a general assumption: the target user can disclose his/her own locations to the public using geosocial services (e.g. location check-in, tagging, nearby friends); the adversary may obtain the training traces of the target user by observing the user in person. However, the number of training traces can still be very small since many users disclose only a small amount of location information in their daily lives. It is also difficult for the adversary to observe the user in person, unless they are in a close relationship with each other.

In the localization phase, user $u_n \in \mathcal{U}$ produces a new trace that contains some sensitive locations (e.g. home, hospital). He/she (or a trusted third party [34]–[36]) obfuscates the trace and discloses it (either in realtime or after producing the actual trace) to use LBS while protecting his/her location privacy. Examples of obfuscation include perturbation [37]–[39], which adds noise to a location, location generalization (or region merging) [21], [40], [41], which combines multiple regions into one group, and location hiding [16], [21], [42], which deletes some locations from a trace. The adversary obtains the obfuscated trace, and derives the actual location of user $u_n$ at a given time instant using the personalized transition matrix $P_n$.

It should be noted that this attack is more general than the location prediction attack [16], [17], which predicts a future location of user $u_n$ from a past location that he/she disclosed. Suppose that user $u_n$ discloses his/her location at time instant 1. Then, the location prediction attack can be regarded as a special case of the localization attack where the actual trace is obfuscated by deleting all locations except for the location at time instant 1.
B. Algorithm for Localization Attacks

We briefly review the localization attack using the Forward-Backward algorithm [43], which was formalized in [21]. Let $\mathcal{X}'$ be a set of possible obfuscated regions. For example, we can use a power set of $\mathcal{X}$, which includes an obfuscated region produced by perturbation, location generalization (or region merging), and location hiding, as $\mathcal{X}'$ (i.e. $\mathcal{X}' = \mathcal{P}(\mathcal{X})$). Let further $o_n(t) \in \mathcal{X}'$ be an obfuscated region of user $u_n \in \mathcal{U}$ at time instant $t \in [T]$, and $o_n = (o_n(1), \ldots, o_n(T))$ be an obfuscated trace of user $u_n$ the adversary obtains.

Let $X_{n,t}$ be a random variable that represents a region user $u_n \in \mathcal{U}$ is in at time instant $t \in [T]$. Then, the posterior probability that the region user $u_n$ is in at time instant $t$ is $x_t \in \mathcal{X}$, given the obfuscated trace $o_n = (o_n(1), \ldots, o_n(T))$, is written as follows:

$$\Pr(X_{n,t} = x_t | o_n).$$

This probability can be estimated using the personalized transition matrix $P_n$ by the Forward-Backward algorithm [43].

In this paper, we do not explain the computation of (1) using the Forward-Backward algorithm (for details, see [21]), but explain how to derive the actual location of user $u_n$ at time instant $t$ using the estimated posterior probabilities $\{\Pr(X_{n,t} = x_t | o_n)| i \in [M]\}$. For example, the adversary can choose region $x_t$ whose posterior probability $\Pr(X_{n,t} = x_t | o_n)$ is the highest as an estimated region. This approach is known as the Bayes decision rule [44], and maximizes the attack success probability (i.e. the probability that the estimation is correct) if $\Pr(X_{n,t} = x_t | o_n)$ is accurately estimated. Another approach is to sort $M$ regions in descending order of the posterior probabilities and choose the top $L (\leq M)$ regions as candidates. This approach includes the Bayes decision rule as a special case where $L = 1$, and maximizes the probability that a correct answer is included in the $L$ candidates.

It should be noted that if the size of each region (or the number of candidates $L$) is large, it is difficult to estimate an exact location such as a hospital and home using only this attack. However, even in this case, the adversary may estimate the exact location by combining this attack with other auxiliary information. For example, suppose that a user disclosed the fact that he/she went to some hospital (but did not clarify which hospital) at a certain time by posting a message on a social networking site. By combining this fact with the localization attack, the adversary may narrow down the area and identify the hospital, which might specialize in a particular disease (e.g. cancer, mental illness), and therefore reveal the disease of the user.

C. Sparse Data Problems

The studies in [18]–[20] computed each transition matrix using the ML (Maximum Likelihood) estimation method. However, the ML estimation method suffers from the sparse data problem (i.e. there are many unobserved elements or overfitted elements in the personalized transition matrix $P_n$) when only a small number of training traces are available to the adversary, as described in Section I. If there are many unobserved elements or overfitted elements, the adversary cannot accurately estimate the posterior probability $\Pr(X_{n,t} = x_t | o_n)$ in (1), and therefore cannot carry out a localization attack with sufficient accuracy.

In [21], Shokri et al. proposed a learning method that considers the case where some locations in training traces are missing. Specifically, their method involves learning a distribution of a transition matrix $P_n$ by assuming a Dirichlet prior distribution for a vector $p_{n,i}$ and estimating missing locations using the Gibbs sampling method. However, since this method computes each transition matrix independently, it also suffers from the sparse data problem. More specifically, the mode of the estimated distribution of $P_n$ is equal to the ML estimate when there is no missing locations in training traces. Thus, in this case, the learning method in [21] provides almost the same performance as the ML estimation method (we also confirmed this in our experiments in Section V).

D. Population Matrix Mode

Both the ML estimation method [18]–[20] and the Gibbs sampling method [21] can also be applied to the population matrix mode, in which the adversary uses the population transition matrix $P_e$. In this case, the adversary assumes that the transition probability from region $x_t \in \mathcal{X}$ to $x_{t'} \in \mathcal{X}$ is the same for all target users, and is given by $p_{x_t,x_{t'}}$ (as defined in Section I-C). By substituting $P_e$ for $P_n$ in Section II-B, the adversary can carry out the localization attack in the population matrix mode.

Since the adversary can use training traces of all target users to train $P_e$, the adversary can use a larger amount of training data to estimate each element of the matrix (in other words, the number of elements that need to be estimated is reduced from $NM^2$ to $M^2$). However, as described in Section I, the population matrix mode has the following two disadvantages: (i) the adversary cannot exploit unique features of each user’s behavior (e.g. favorite places or routes, walking speed) for deriving a location; (ii) he/she cannot carry out the de-anonymization attack [18]–[21]. Furthermore, the adversary might still suffer from the sparse data problem in the case the number of regions $M$ is large (i.e. he/she wants to know an exact location of the user), since the number of elements in the population matrix is $M^2$ (i.e. it increases quadratically).

III. Matrix/Tensor Factorization

A tensor is a mathematical object that can be regarded as a multidimensional array [27]. It includes a vector as a 1st-order tensor, and a matrix as a 2nd-order tensor. In this paper, we regard a set of personalized transition matrices as a 3rd-order tensor that is composed of the “User” mode, “From Region” mode, and “To Region” mode, and use tensor factorization [27]–[31] (resp. matrix factorization) to overcome the sparse data problem in the personalized matrix mode (resp. population matrix mode). We also refer to this 3rd-order tensor as a transition probability tensor (see Fig. 4).

In this section, we briefly explain matrix factorization (Section III-A) and tensor factorization (Section III-B).
Matrix Factorization

Matrix factorization is one of the most popular approaches to item recommendation [32]. It factorizes a large matrix into two low-rank matrices to approximate its elements (including unobserved/overfitted elements) from a small amount of training data, which is called low-rank approximation.

Here we consider a matrix \( A \in \mathbb{R}^{M \times M} \) as an example (we consider a square matrix for ease of explanation). Matrix factorization approximates \( A \) as follows:

\[
\hat{A} = UV^T, \tag{2}
\]

where \( \hat{A} \) is the approximation of \( A \), and \( U \in \mathbb{R}^{M \times K} \) and \( V \in \mathbb{R}^{M \times K} \) are low-rank matrices called feature matrices (\( K \) is generally much smaller than \( N \) and \( M \)). That is, \( \hat{A} \) computes the interaction \( \hat{a}_{n,i} \) as the product of \( U \) and \( V^T \) (see the left side of Fig. 5). Let \( a_{n,i} \in \mathbb{R} \) be the \((n,i)\)-th element of \( A \), and \( \hat{a}_{n,i} \) be the approximation of \( a_{n,i} \). Let further \( u_n \) (resp. \( v_i \)) \( \in \mathbb{R}^K \) be the \(n\)-th row of \( U \) (resp. \( i\)-th row of \( V \)). The approximation \( \hat{a}_{n,i} \) is given by \( u_n \) and \( v_i \) as follows:

\[
\hat{a}_{n,i} = \langle u_n, v_i \rangle = \sum_{k=1}^{K} u_{n,k} v_{i,k}. \tag{3}
\]

\( u_n \) and \( v_i \) are called feature vectors, and \( u_{n,k} \) and \( v_{i,k} \) \( \in \mathbb{R} \) are called model parameters. By low-rank approximation, each element is influenced by similar feature vectors, and consequently elements (including unobserved ones) can be effectively estimated from a small amount of training data.

Tensor Factorization

Tensor factorization is a generalization of matrix factorization to a 3rd-order tensor. There are various methods to factorize a 3rd-order tensor: TD (Tucker decomposition) [27], CP (CD/PARAFAC) [27], PITF (Pairwise Interaction Tensor Factorization) [28], [29], etc. In this paper, we focus on PITF because of its high performance: it outperforms TD largely in runtime, and achieves much higher accuracy than CP [28].

We now explain PITF in detail. Here we consider a tensor \( A \in \mathbb{R}^{N \times M \times K} \) that is composed of the “User” mode, “From Region” mode, and “To Region” mode as an example. PITF factorizes a 3rd-order tensor: TD (Tucker decomposition) [27], which is one such example, factorizes a tensor composed of transition probabilities to recommend items (e.g. books, DVDs). However, they aimed at creating a personal ranking of items, and did not estimate transition probabilities themselves. On the other hand, the localization attack, which is the focus of this paper, uses the personalized transition matrix \( P_n \) (i.e. the transition probabilities themselves) to compute the posterior probability \( P(X_{n,t} = x_t | a_n) \) in (1). To the best of our knowledge, no studies have attempted to estimate the transition probabilities themselves using tensor factorization.

Fig. 4. Transition probability tensor.

\[
\begin{align*}
\text{From Region} & \rightarrow \text{To Region} \\
\text{User} & \begin{pmatrix}
0.8 & 0 & 0 & 0 & 0 \\
0 & 0.8 & 0 & 0 & 0 \\
0 & 0 & 0.8 & 0 & 0 \\
0 & 0 & 0 & 0.8 & 0 \\
0 & 0 & 0 & 0 & 0.8
\end{pmatrix} \\
\text{To Region} & \begin{pmatrix}
0.5 & 0 & 0 & 0 & 0 \\
0 & 0.5 & 0 & 0 & 0 \\
0 & 0 & 0.5 & 0 & 0 \\
0 & 0 & 0 & 0.5 & 0 \\
0 & 0 & 0 & 0 & 0.5
\end{pmatrix}
\end{align*}
\]

Fig. 5. Matrix factorization and PITF applied to a tensor composed of “User” mode, “From Region” mode, and “To Region” mode.

\[
\hat{a}_{n,i,j} = \langle u_n^{(a)}, v_i^{(a)} \rangle + \langle u_n^{(b)}, v_j^{(b)} \rangle + \langle u_i^{(c)}, v_j^{(c)} \rangle \tag{4}
\]

\[
= \sum_{k=1}^{K} u_{n,k}^{(a)} v_{i,k}^{(a)} + \sum_{k=1}^{K} u_{n,k}^{(b)} v_{j,k}^{(b)} + \sum_{k=1}^{K} u_{i,k}^{(c)} v_{j,k}^{(c)} \tag{5}
\]

where \( \hat{a}_{n,i,j} \) is the approximation of \( a_{n,i,j} \) by PITF, and \( u_n^{(a)}, u_n^{(b)}, v_i^{(a)}, v_j^{(b)}, u_i^{(c)}, v_j^{(c)} \in \mathbb{R}^K \) are \( K \)-dimensional feature vectors (\( K \) is generally much smaller than \( N \) and \( M \)). Let \( U^{(a)} \in \mathbb{R}^{N \times K}, V^{(a)} \in \mathbb{R}^{M \times K}, U^{(b)} \in \mathbb{R}^{N \times K}, V^{(b)} \in \mathbb{R}^{M \times K}, U^{(c)} \in \mathbb{R}^{M \times K}, \) and \( V^{(c)} \in \mathbb{R}^{M \times K} \) be the corresponding feature matrices (e.g. \( u_n^{(a)} \) is the \( n \)-th row of \( U^{(a)} \)). Then, we can see from (3) and (4) that PITF is a generalization of matrix factorization in that it models the interaction between “User” and “From Region” as \( U^{(a)} V^{(a)^T} \), the interaction between “User” and “To Region” as \( U^{(b)} V^{(b)^T} \), and the interaction between “From Region” and “To Region” as \( U^{(c)} V^{(c)^T} \). By factorizing a tensor this way, each element is influenced by similar feature vectors (i.e. similar users and similar regions), and can be effectively estimated from a small amount of training data.

IV. LEARNING TRANSITION MATRICES USING MATRIX/TENSOR FACTORIZATION

We now propose a learning method using tensor factorization to overcome the sparse data problem in the personalized matrix mode (Section IV-A). We also extend our learning method to the population matrix mode (Section IV-B).

A. Learning Personalized Transition Matrices

In Section III-B, we explained an example of factorizing a tensor composed of real numbers. However, very few studies have factorized a tensor composed of transition probabilities such as Fig. 4. The study of Rendle et al. [29], which is one such example, factorized a tensor composed of transition probabilities to recommend items (e.g. books, DVDs). However, they aimed at creating a personal ranking of items, and did not estimate transition probabilities themselves. On the other hand, the localization attack, which is the focus of this paper, uses the personalized transition matrix \( P_n \) (i.e. the transition probabilities themselves) to compute the posterior probability \( P(X_{n,t} = x_t | a_n) \) in (1). To the best of our knowledge, no studies have attempted to estimate the transition probabilities themselves using tensor factorization.
Thus we propose a learning method that estimates each element of a transition probability tensor using tensor factorization. Here it must be noted that the summation of transition probabilities over “To Region” is always 1 (i.e. \( \sum_{j=1}^{M} a_{n,i,j} = 1 \)), and that it is intractable to directly factorize a transition probability tensor under this constraint. To avoid this problem, our learning method first factorizes a transition count tensor, whose \((n,i,j)\)-th element is the number of transitions from region \(x_i \) to \(x_j \) that user \(u_n \in \mathcal{U} \) has made, and then computes a transition probability tensor by normalizing counts to probabilities. “?” is an element whose corresponding element in a transition probability tensor is unobserved.

Our learning method applies a transition count tensor to the tensor \( \mathcal{A} \) in Section III-B (i.e. \( a_{n,i,j} \) is the number of transitions from region \(x_i \) to \(x_j \) that user \(u_n \in \mathcal{U} \) has made). Let \( \Theta \) be a set of model parameters in PITF:

\[
\Theta = \{ \mathbf{U}^{(a)}, \mathbf{V}^{(a)}, \mathbf{U}^{(b)}, \mathbf{V}^{(b)}, \mathbf{U}^{(c)}, \mathbf{V}^{(c)} \}
\]

(6)

Our learning method computes a transition probability tensor as follows:

1. Compute a transition count tensor \( \mathcal{A} \) by counting each transition from the training data (in Fig. 6, an element in \( \mathcal{A} \), whose corresponding element in a transition probability tensor is unobserved, is marked with “?”).
2. Compute a set of model parameters \( \Theta \) from \( \mathcal{A} \), using the algorithm described above.
3. Compute \( \{ \hat{a}_{n,i,j} | n \in [N], i, j \in [M] \} \) from \( \Theta \) using (5) and (7).
4. Compute a transition probability tensor from \( \{ \hat{a}_{n,i,j} | n \in [N], i, j \in [M] \} \) by normalizing counts to probabilities.

It should be noted that although we use PITF as a factorization method in this paper, we can use other factorization methods such as TD and CP in our learning method.

We now explain how to compute a set of model parameters \( \Theta \) from \( \mathcal{A} \) (i.e. step 2) in detail. Here we note that the number of transitions \( a_{n,i,j} \) is not a probability but a non-negative value. Under this constraint, we compute a set of model parameters \( \hat{\Theta} \) that minimizes the following regularized squared error [30]–[32]:

\[
\hat{\Theta} = \arg \min_{\Theta \geq 0} \sum_{(n,i,j) \in \mathbb{D}} (a_{n,i,j} - \hat{a}_{n,i,j})^2 + \lambda \| \Theta \|^2_F, \tag{8}
\]

where \( \mathbb{D} = \{(n,i,j) | n \in [N], i, j \in [M], \sum_{j'=1}^{M} a_{n,i,j'} \geq 1 \} \).

The first term in (8) is the summation of squared errors over the set of elements \( \mathbb{D} \) whose corresponding elements in a transition probability tensor are observed (i.e. not marked with “?”). That is, we compute \( \hat{\Theta} \) based on observed elements only, and then estimate all elements including unobserved ones based on \( \hat{\Theta} \) (using (5) and (7)). The second term is introduced to avoid overfitting the observed elements, and is called a regularization term. \( \| \cdot \|^2_F \) is the Frobenius norm (i.e. square sum of all elements) and \( \lambda \) (\( \lambda > 0 \)) is called a regularization parameter. \( \lambda \) controls the extent of regularization, and is generally determined by cross-validation [32]. \( \Theta \geq 0 \) means that all model parameters are non-negative, which guarantees the non-negativity of the estimated transition count \( \hat{a}_{n,i,j} \) (see (5) and (7)).

Since (8) is not convex for all model parameters in \( \Theta \), it is intractable to find a closed-form solution to (8). Thus we use ANLS (Alternating Nonnegative Least Square) [30], [31], which is a popular approach to finding an approximate solution. ANLS solves an optimization problem for one model parameter \( \theta \in \Theta \) while fixing all other model parameters \( \Theta \setminus \{ \theta \} \), and iterates this process in a cyclic manner until convergence. By substituting (5) into (8) and solving (8) for one model parameter while fixing all other model parameters, we obtain the following update formulas:

\[
u_{n,k}^{(a)} \leftarrow \frac{\sum_{2\leq i \leq M} (a_{n,i,j} - \hat{a}_{n,i,j} + u_{n,k}^{(a)} v_{i,k}^{(a)}) v_{i,k}^{(a)}}{\sum_{2\leq i \leq M} (v_{i,k}^{(a)})^2 + \lambda} \tag{9}
\]

\[
u_{i,k}^{(a)} \leftarrow \frac{\sum_{2\leq n \leq N} (a_{n,i,j} - \hat{a}_{n,i,j} + u_{n,k}^{(a)} v_{i,k}^{(a)}) u_{n,k}^{(a)}}{\sum_{2\leq n \leq N} (u_{n,k}^{(a)})^2 + \lambda} \tag{10}
\]

\[
u_{j,k}^{(b)} \leftarrow \frac{\sum_{2\leq n \leq N} (a_{n,i,j} - \hat{a}_{n,i,j} + u_{n,k}^{(b)} v_{j,k}^{(b)}) u_{n,k}^{(b)}}{\sum_{2\leq n \leq N} (u_{n,k}^{(b)})^2 + \lambda} \tag{11}
\]

\[
u_{i,k}^{(c)} \leftarrow \frac{\sum_{2\leq n \leq N} (a_{n,i,j} - \hat{a}_{n,i,j} + u_{n,k}^{(c)} v_{i,k}^{(c)}) u_{n,k}^{(c)}}{\sum_{2\leq n \leq N} (u_{n,k}^{(c)})^2 + \lambda} \tag{12}
\]

\[
u_{j,k}^{(c)} \leftarrow \frac{\sum_{2\leq n \leq N} (a_{n,i,j} - \hat{a}_{n,i,j} + u_{n,k}^{(c)} v_{j,k}^{(c)}) u_{n,k}^{(c)}}{\sum_{2\leq n \leq N} (u_{n,k}^{(c)})^2 + \lambda} \tag{14}
\]

where \( \mathbb{D}_{1,n} = \{(i,j) | i, j \in [M], \sum_{i'=1}^{M} a_{n,i,j'} \geq 1 \} \),
\[\mathbb{D}_{2,i} = \{(n,j) | n \in [N], j \in [M], \sum_{j'=1}^{M} a_{n,i,j'} \geq 1 \} \),
\[\mathbb{D}_{3,j} = \{(n,i) | n \in [N], i \in [M], \sum_{i'=1}^{M} a_{n,i,j'} \geq 1 \} \),
and \( |x| = \max(x, e) \) (e is a small positive value, and typically \( e = 10^{-16} \) [30], [31]).
We first update \( U^{(a)}(= \{ u_{i,n}^{(a)} | n \in [N], k \in [K] \}) \) using (9), and then update \( V^{(a)}(= \{ v_{i,k}^{(a)} | i \in [M], k \in [K] \}) \) using (10). We update the other parameters (i.e. \( U^{(b)}, V^{(b)}, U^{(c)}, V^{(c)} \)) in the same way using (11)-(14), and iterate this update procedure until the values \( u_{i,k}^{(a)}, v_{i,k}^{(a)}, u_{n,k}^{(b)}, v_{j,k}^{(b)}, u_{i,k}^{(c)}, \) and \( v_{j,k}^{(c)} \) converge. As for initialization of model parameters, we should avoid setting all parameters to 0, because otherwise the denominator in (9) will become very small, and consequently parameters will oscillate and converge slowly. In our experiments in Section V, we adopted the widely-used random initialization method [45], which initializes each element as a random number between 0 and 1.

To sum up, our learning algorithm in the personalized matrix mode can be written as follows:

**Algorithm 1 (Learning a Transition Probability Tensor):**

1) Compute a transition count tensor \( \mathbf{A} \) from training data.
2) Compute model parameters \( \Theta \) from \( \mathbf{A} \) by iteratively updating parameters using (9)-(14) until convergence.
3) Compute \( \{ \hat{a}_{i,n,j} | n \in [N], i, j \in [M] \} \) from \( \Theta \) using (5) and (7).
4) Compute a transition probability tensor from \( \{ \hat{a}_{i,n,j} | n \in [N], i, j \in [M] \} \) by normalizing counts to probabilities.

Let \( Z \) be the total number of transitions in the training data, and \( \eta \) be the number of iterations in the step 2. Then, the time complexity of each step from 1 to 4 can be expressed as \( O(Z) \), \( O(\eta K |\mathcal{D}|) \), \( O(\eta NM^2) \) and \( O(\eta NM^2) \), respectively. Thus, the total time complexity is \( O(Z + \eta K |\mathcal{D}| + \eta NM^2) \). In comparison, the ML estimation method [18]–[20], which computes step 1 and 4 only, has the time complexity \( O(Z + NM^2) \).

**B. Learning a Population Transition Matrix**

Since a tensor includes a matrix as a second-order tensor, our learning method in Section IV-A can be easily extended to the population matrix mode. In this case, we factorize a transition count matrix, whose \((i,j)\)-th element is the number of transitions from region \( x_i \in \mathcal{X} \) to \( x_j \in \mathcal{X} \) in the training data of all target users, and then compute a transition probability matrix (i.e. population transition matrix) by normalizing counts to probabilities (see Fig. 7).

We apply a transition count matrix to the matrix \( \mathbf{A} \) in Section III-A (i.e. \( a_{i,j} \) is the number of transitions from region \( x_i \) to \( x_j \) in the training data of all target users), and compute a set of model parameters in matrix factorization. Let \( \Theta^* \) be the set of model parameters:

\[
\Theta^* = \{ U, V \} \quad \text{(15)}
\]

\[
= \{ u_{i,k}, v_{j,k} | i, j \in [M], k \in [K] \}. \quad \text{(16)}
\]

Using ANLS (Alternating Nonnegative Least Square) [30], [31], we can compute model parameters that approximates the ones that minimize the regularized squared error (in the same way as Section IV-A). The update formulas in this case can be written as follows:

\[
u_{i,k} \leftarrow \frac{\sum_{j=1}^{M} (a_{i,j} - \hat{a}_{i,j} + u_{i,k}v_{j,k})v_{j,k}}{\sum_{j=1}^{M} (v_{j,k})^2 + \lambda} \quad \text{(17)}
\]

\[
v_{j,k} \leftarrow \frac{\sum_{i=1}^{N} (a_{i,j} - \hat{a}_{i,j} + u_{i,k}v_{j,k})u_{i,k}}{\sum_{i=1}^{N} (u_{i,k})^2 + \lambda} \quad \text{(18)}
\]

where \( \mathcal{D}^*_{i,j} = \{ j | j \in [M], \sum_{j'=1}^{M} a_{i,j'} \geq 1 \} \) and \( \mathcal{D}^*_{j,i} = \{ i | i \in [M], \sum_{j'=1}^{M} a_{i,j'} \geq 1 \} \) (\( \lambda \) is a regularization parameter). We update \( \mathbf{U} (= \{ u_{i,k} | i \in [M], k \in [K] \}) \) using (17), and then update \( \mathbf{V} (= \{ v_{j,k} | j \in [M], k \in [K] \}) \) using (18). We iterate this update procedure until the values \( u_{i,k} \) and \( v_{j,k} \) converge.

Our learning algorithm in the population matrix mode can be written as follows:

**Algorithm 2 (Learning a Transition Probability Matrix):**

1) Compute a transition count matrix \( \mathbf{A} \) from training data.
2) Compute model parameters \( \Theta^* \) from \( \mathbf{A} \) by iteratively updating parameters using (17) and (18) until convergence.
3) Compute \( \{ \hat{a}_{i,j} | i, j \in [M] \} \) from \( \Theta^* \) using (3) and (16).
4) Compute a transition probability matrix from \( \{ \hat{a}_{i,j} | i, j \in [M] \} \) by normalizing counts to probabilities.

If \( K \) is much smaller than \( M \), the number of model parameters in the matrix factorization (= \( 2MK \)) is much smaller than the number of elements in the population transition matrix (= \( M^2 \)). It is also smaller than the number of model parameters in PITF (= \( 2NM^2 \)). Thus, our learning method in the population matrix mode has the fewest parameters, which is desirable when the training data is extremely sparse. However, it cannot exploit unique features of each user’s behavior (e.g. favorite places or routes), nor be applied to the de-anonymization attack [18]–[21], as described in Section II-D.

**V. EXPERIMENTAL EVALUATION**

**A. Experimental Set-up**

We carried out experiments to show the effectiveness of our learning method using four real datasets: the Geolife dataset [23], the Gowalla dataset [24], the CRAWDAD (epfl/mobility) dataset [25], and the CRAWDAD (roma/taxi) dataset [26].
- **Geolife dataset**: The Geolife dataset [23] was collected by Microsoft Research Asia. It contains GPS traces of 182 users in a period of over five years (from April 2007 to August 2012), and contains a variety of private movements such as going home, going to work, and shopping, mostly in Beijing. In our experiments, we used the traces in Beijing. We chose 80 users who had long traces (N = 80), and extracted, for each of them, 10 traces each of which comprises 10 locations and has a time interval of more than 30 minutes (we eliminated the remaining 102 users, since we had insufficient data to extract 10 such traces for these users).

- **Gowalla dataset**: The Gowalla dataset [24] contains 6442890 check-ins of 196591 users over a period of over two years (from February 2009 to October 2010). Although there are a large number of check-ins, they are scattered all over the world. In our experiments, we used the traces in New York. We chose 173 users who had long traces (N = 173), and extracted, for each of them, 10 traces each of which comprises 10 locations (a time interval is more than 30 minutes) in the same way as in the Geolife dataset.

- **CRAWDAD (epfl/mobility) dataset**: The CRAWDAD (epfl/mobility) dataset [25] contains GPS traces of 536 taxis collected over 30 days in the San Francisco Bay Area. We chose 529 taxis that had long traces (N = 529), and extracted, for each of them, 10 traces each of which comprises 10 locations (a time interval is more than 30 minutes) in the same way as in the above datasets.

- **CRAWDAD (roma/taxi) dataset**: The CRAWDAD (roma/taxi) dataset [26] contains GPS traces of 306 taxis collected over 30 days in Rome. We chose 306 taxis that had long traces (N = 306), and extracted, for each of them, 10 traces each of which comprises 10 locations (a time interval is more than 30 minutes) in the same way as in the above datasets.

We divided each of the four areas (Beijing, New York, the San Francisco Bay Area, and Rome) into 16 × 16 regions (M = 256), where the boundaries were determined so that all the traces were uniformly distributed along each axis (i.e. 1/16 = 6.25% per each bin). We used, for each user (or taxi), \( Y \in \{1, 3, 5, 7, 9\} \) trace(s) as training data, and the remaining 10 − \( Y \) trace(s) as testing data. Here we attempted 10 ways to randomly choose \( Y \) training trace(s) from 10 traces, and carried out, for each case, the following experiments.

Using the training data, we computed the personalized transition matrices \( \{P_n|n \in [N]\} \) (or a population transition matrix \( P^* \)). For comparison, we evaluated our learning method and the ML (Maximum Likelihood) estimation method [18]–[20] in both the personalized matrix mode and the population matrix mode:

- **ML**: The ML estimation method [18]–[20] in the personalized matrix mode (see Section II-C).
- **ML***: The ML estimation method [18]–[20] in the population matrix mode (see Section II-D).
- **FC**: Our learning method using tensor factorization in the personalized matrix mode (see Section IV-A).
- **FC***: Our learning method using matrix factorization in the population matrix mode (see Section IV-B).

In ML and ML*, we set a transition probability \( p_{n,i,j} \) (or \( p_{*,i,j} \)) for an unobserved element to 1/256 (i.e., we assumed a uniform distribution for these elements). In FC and FC*, we set the dimension \( K \) of feature vectors to \( K = 16 \) (since this provided high performance in [1]). We initialized the model parameters \( \Theta \) (or \( \Theta^* \)) by using the random initialization method [45], which initializes each parameter as a random number between 0 and 1. After the initialization, we computed \( \Theta \) (or \( \Theta^* \)) using our learning method. Here we set the regularization parameter \( \lambda \) to \( \lambda = 10^{-3} \) (we measured performance for \( \lambda \) values \( 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, \) and \( 10^{-5} \), and confirmed that the performance was almost the same in each case). After computing the transition matrix \( P_n \) (or \( P^* \)), we computed stationary probabilities (steady state probabilities) for user \( u_n \) (or all users), which are necessary in computing the posterior probability \( \Pr(X_{n,t} = x_i|\alpha_n) \) in (1) (for details, see [21]), by multiplying uniform probabilities by the transition matrix \( P_n \) (or \( P^* \)) repeatedly until convergence in the same way as [19].

Using the testing data, we evaluated the performance of the localization attack, which is described in Sections II-A and II-B. We first obfuscated each region in the testing traces using the location generalization and location hiding method [21] with parameters \( b \) and \( \phi \) (denoted by LGLH(\( b, \phi \))). This method generalizes (merges) a region by dropping lower \( b \) bit(s) for each of the x-coordinate and y-coordinate (expressed as a binary sequence), and hides (deletes) a region with probability \( \phi \). For example, LGLH(1, 0.1), on input region \( x_{34} \), outputs regions \( \{x_{33}, x_{34}, x_{49}, x_{50}\} \) with probability 0.9, and outputs \( \bot \) (\( \bot \) represents that the region is hidden) with probability 0.1 (see Fig 8). For parameters \( b \) and \( \phi \), we tested \( b \in \{0, 2\} \) and \( \phi \in \{0.5, 0.8\} \).

We then evaluated the performance in the case when the adversary estimated each region in the obfuscated traces by choosing \( L \) (\( 1 \leq L \leq 256 \)) candidates in descending order of posterior probabilities (see Section II-B). As a performance measure, we used the attack success rate, which is the ratio of the number of successful attacks (i.e., attacks in which the chosen \( L \) candidates include a correct answer) divided by the total number of attacks (i.e., \( N \times (10 − Y) \times 10 \)). We averaged

\[
\begin{bmatrix}
0000 & 0001 & 0010 & 0011 & \cdots & 1111 \\
0000 & x_1 & x_2 & x_3 & \cdots & x_{16} \\
0001 & x_17 & x_18 & x_19 & \cdots & x_{32} \\
0010 & x_{33} & x_{34} & x_{35} & \cdots & x_{48} \\
0011 & x_{49} & x_{50} & x_{51} & x_{52} & \cdots & x_{64} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1111 & x_{241} & x_{242} & x_{243} & x_{244} & \cdots & x_{256} \\
\end{bmatrix}
\]

Fig. 8. Generalized region (\( M = 256 \)). For example, LGLH(1, 0.1), on input region \( x_{34} \), outputs regions \( \{x_{33}, x_{34}, x_{49}, x_{50}\} \) (gray area) with probability 0.9, and outputs \( \bot \) that represents the region is hidden with probability 0.1.
the attack success rate over all the 10 ways to randomly choose training traces to obtain a stable performance.

### B. Experimental Results

We firstly evaluated the performance in the case when only one trace per user (or taxi) is used as training data ($Y = 1$). Fig. 9 shows the relationship between the number of candidates $L$ and the attack success rate. The dashed line represents the performance of a random guess, which randomly chooses $L$ candidates from a generalized region (or all regions when the region is hidden). For example, when we set $L = 2$, the attack success rate of a random guess is $50.4\%$ ($= 0.5 \times 1 + 0.5 \times 2 / 256$) and $3.1\%$ ($= 0.2 \times 2 / 16 + 0.8 \times 2 / 256$) for LGLH(0, 0.5) and LGLH(2, 0.8), respectively. We also show in Table I the attack success rate when we used LGLH(2, 0.5) and fixed $L = 1, 8, \text{or} 64$.

It can be seen from Fig. 9 that ML performs only as
TABLE I
THE ATTACK SUCCESS RATE [%] (Y = 1, LGLH(2, 0.5)). THE BEST PERFORMANCE IS HIGHLIGHTED IN BOLDFACE.

(i) Geolife dataset

<table>
<thead>
<tr>
<th>L</th>
<th>ML</th>
<th>ML*</th>
<th>FC</th>
<th>FC*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>11.93</td>
<td>10.64</td>
<td>14.54</td>
<td>10.97</td>
</tr>
<tr>
<td>8</td>
<td>36.00</td>
<td>40.14</td>
<td>50.47</td>
<td>49.07</td>
</tr>
<tr>
<td>64</td>
<td>61.31</td>
<td>60.98</td>
<td>81.05</td>
<td>79.46</td>
</tr>
</tbody>
</table>

(ii) Gowalla dataset

<table>
<thead>
<tr>
<th>L</th>
<th>ML</th>
<th>ML*</th>
<th>FC</th>
<th>FC*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20.41</td>
<td>13.91</td>
<td>21.39</td>
<td>14.35</td>
</tr>
<tr>
<td>8</td>
<td>42.24</td>
<td>49.73</td>
<td>58.94</td>
<td>53.35</td>
</tr>
<tr>
<td>64</td>
<td>62.38</td>
<td>74.82</td>
<td>87.09</td>
<td>85.52</td>
</tr>
</tbody>
</table>

(iii) CRAWDAD (epfl/mobility) dataset

<table>
<thead>
<tr>
<th>L</th>
<th>ML</th>
<th>ML*</th>
<th>FC</th>
<th>FC*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.54</td>
<td>9.79</td>
<td>9.30</td>
<td>10.97</td>
</tr>
<tr>
<td>8</td>
<td>27.32</td>
<td>40.88</td>
<td>41.76</td>
<td>45.50</td>
</tr>
<tr>
<td>64</td>
<td>60.71</td>
<td>69.24</td>
<td>74.92</td>
<td>77.72</td>
</tr>
</tbody>
</table>

(iv) CRAWDAD (roma/taxi) dataset

<table>
<thead>
<tr>
<th>L</th>
<th>ML</th>
<th>ML*</th>
<th>FC</th>
<th>FC*</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.56</td>
<td>9.47</td>
<td>13.93</td>
<td>13.25</td>
</tr>
<tr>
<td>8</td>
<td>28.37</td>
<td>34.08</td>
<td>48.99</td>
<td>47.96</td>
</tr>
<tr>
<td>64</td>
<td>58.56</td>
<td>58.15</td>
<td>78.78</td>
<td>79.61</td>
</tr>
</tbody>
</table>

well as a random guess in most cases. This is because the training data is extremely sparse; for each user (or taxi), the number of transitions in the training data is 9, while the number of elements in the personalized transition matrix is $M^2 = 65536$. It can also be seen that ML* outperforms ML in most of the cases, since much more training data (e.g. 529 times more in the CRAWDAD (epfl/mobility) dataset) are used to estimate each element. However, ML* still performs similar to a random guess in many cases (e.g. LGLH(0, 0.5), LGLH(2, 0.5)), since the number of elements is very large ($M^2 = 65536$).

On the other hand, FC and FC* significantly outperform the ML estimation method and a random guess in most cases, since they overcome the sparse data problem. From this result, we can say that the localization attack using our learning method is a threat even in a realistic situation where the amount of training data is limited. It can be seen from Fig. 9 and Table I that FC provides the best performance in the Geolife dataset and the Gowalla dataset, while FC* provides the best performance in the CRAWDAD (epfl/mobility) dataset. We believe this is because FC captures unique features of each user’s behavior in the Geolife dataset and the Gowalla dataset, while this is not the case in the CRAWDAD (epfl/mobility) dataset. In the CRAWDAD (epfl/mobility) dataset, each taxi does not have very distinct features (we confirmed this by visualizing the traces). It seems natural, since the destination of the taxi is often determined not by the driver but the customer. FC* outperforms FC in such cases, since the number of parameters is smaller (as described in Section IV-B). On the other hand, since the Geolife dataset and the Gowalla dataset contain a variety of private movements (e.g. going home, going to work), each user’s trace has distinctive features. FC effectively captures such features from a small amount of training data, and exploits it for the localization attack. In Section VI, we show that this is indeed the case by visualizing model parameters in FC and the corresponding traces.

We secondly evaluated the performance in the case when the number of training traces per user (or taxi) $Y$ is changed from 1 to 9. Here we used LGLH(2, 0.5) as an obfuscation mechanism. Fig. 10 shows the relationship between the number of training traces $Y$ and the attack success rate (for the number of candidates $L$, we tested three cases: $L = 1, 8, 64$). We also show in Table II the percentage of unobserved elements in ML and ML*.

It can be seen from Fig. 10 that in many cases, the performance of ML or ML* increases rapidly as the number of training traces $Y$ increases from 1 to 9. In particular, when $Y$ is large (e.g. $Y = 7$ or 9), ML outperforms FC and FC* in many cases in the Geolife dataset and the Gowalla dataset.
TABLE II
THE PERCENTAGE OF UNOBSERVED ELEMENTS [%].

<table>
<thead>
<tr>
<th></th>
<th>Y = 1</th>
<th>Y = 2</th>
<th>Y = 3</th>
<th>Y = 4</th>
<th>Y = 5</th>
<th>Y = 6</th>
<th>Y = 7</th>
<th>Y = 8</th>
<th>Y = 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) Geolife</td>
<td>97.82</td>
<td>94.86</td>
<td>92.79</td>
<td>91.05</td>
<td>90.15</td>
<td>89.48</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>42.81</td>
<td>20.35</td>
<td>12.15</td>
<td>8.20</td>
<td>5.55</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(ii) Gowalla</td>
<td>97.88</td>
<td>95.39</td>
<td>93.54</td>
<td>92.03</td>
<td>90.71</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>25.08</td>
<td>15.90</td>
<td>13.48</td>
<td>11.64</td>
<td>10.63</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iii) CRAWDAD (epfl/mobility)</td>
<td>96.81</td>
<td>91.06</td>
<td>86.13</td>
<td>81.54</td>
<td>77.59</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>9.26</td>
<td>5.86</td>
<td>4.92</td>
<td>4.57</td>
<td>4.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(iv) CRAWDAD (roma/taxi)</td>
<td>97.10</td>
<td>92.48</td>
<td>88.73</td>
<td>85.49</td>
<td>82.47</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ML</td>
<td>30.27</td>
<td>14.06</td>
<td>8.32</td>
<td>6.68</td>
<td>4.92</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Despite the fact that there are still many unobserved elements (as shown in Table II (i) and (iii)). This is because many users stay in the same region for a long time in these datasets. ML provides a good performance in such datasets when the amount of training data is large (on the other hand, ML provides the worst performance in the taxi datasets even when Y = 9, since each taxi moves very fast).

It should be noted again, however, that the available training data can be extremely sparse in reality, since many users disclose only a small amount of location information in their daily lives. For example, when Y = 1 (i.e. when more than 96% of the elements in the personalized transition matrices are unobserved, as shown in Table II), ML performs only as well as a random guess in most cases, as shown in Fig. 9. ML* also performs only as well as a random guess in many cases. On the other hand, our learning method derives the location of a target user with much higher accuracy than a random guess in all of the four datasets even when Y = 1.

VI. VISUALIZATION OF MODEL PARAMETERS AND TRACES

In Section V, we showed that our learning method in the personalized matrix mode (i.e. FC) outperformed the other learning methods (i.e. FC*, ML, ML*) in the Geolife dataset and the Gowalla dataset. To explain the reason for this, we show how well FC captures unique features of each user’s behavior from a small amount of training data in these datasets. Specifically, we show that each element of the personalized transition matrices in FC is indeed influenced by similar users and similar regions (as described in Section III-B) by visualizing model parameters and the corresponding traces.

In our analysis, we used all of the 80 users in the Geolife dataset, and chose 80 users from 173 users in the Gowalla dataset (to equalize the number of users). We chose, for each dataset, 2 traces per user (i.e. 80 × 2 × 10 locations) from the traces extracted in Section V-A, and used them as training data. For the purpose of simple visualization, we divided each of the two areas (Beijing and New York) into 4 × 4 regions at regular intervals, and set the dimension K of feature vectors to K = 4. We initialized the model parameters Θ in FC by using the random initialization method [45], determined the regularization parameter λ by 10-fold cross-validation, and trained Θ using our learning method. Then we examined the values of the following model parameters:

\[
\begin{align*}
\{U^{(a)}, V^{(a)}, U^{(b)}, V^{(b)}\} &= \{u_n^{(a)}, v_i^{(a)}, u_n^{(b)}, v_i^{(b)} \mid n \in [N], i \in [M]\} \\
&= \{u_{n,k}^{(a)}, v_{i,k}^{(a)}, u_{n,k}^{(b)}, v_{i,k}^{(b)} \mid n \in [N], i \in [M], k \in [K]\} \tag{19}
\end{align*}
\]

\[
\begin{align*}
\{U^{(a)}, U^{(b)}\} &= \{u_n^{(a)}, u_n^{(b)} \mid n \in [N]\} \\
&= \{u_{n,k}^{(a)}, u_{n,k}^{(b)} \mid n \in [N], k \in [K]\} \tag{20}
\end{align*}
\]

(i.e. the first and second terms in (5)). Here, we normalized these model parameters so that \(\sum_{n=1}^{N} u_{n,k}^{(a)} = 1\), \(\sum_{i=1}^{M} u_{i,k}^{(b)} = 1\), \(\sum_{n=1}^{N} u_{n,k}^{(b)} = 1\), and \(\sum_{i=1}^{M} v_{i,k}^{(b)} = 1\), for the purpose of visualization.

Fig. 11 and 12 show a visualization of \(\{U^{(a)}, U^{(b)}\}\) and \(\{V^{(a)}, V^{(b)}\}\), respectively. In Fig. 11 and 12, the elements of \(\{U^{(a)}, U^{(b)}\}\) and \(\{V^{(a)}, V^{(b)}\}\) whose corresponding model parameters are more than 0.025 and 0 are filled with gray, respectively (e.g. \(u_n^{(a)} > 0.025 \) and \(v_{i,k}^{(b)} = 0.12 > 0\) in the Geolife dataset). We now explain what these figures illustrate. Since our learning method approximates a transition count \(\alpha_{n,i,j}\) as \(\alpha_{n,i,j} = (u_n^{(a)}, v_i^{(a)}) + (u_n^{(b)}, v_i^{(b)}) + (u_{n,k}^{(a)}, v_{i,k}^{(a)})\) (see (4)), the user \(u_n \in \mathcal{U}\) who has a high model parameter \(u_{n,k}^{(a)}\) (or \(u_{n,k}^{(b)}\)) tends to go to the region \(x_i \in \mathcal{X}\) whose corresponding model parameter \(v_{i,k}^{(a)}\) (or \(v_{i,k}^{(b)}\)) is high. For example, it can be seen from Fig. 11(i)(H) and 12(i)(H), that users \(u_1, u_5, u_{10}, \ldots, u_80\) tend to go to regions \(x_4\) and \(x_7\) in the Gowalla dataset. In this sense, users \(u_4, u_5, u_{10}, \ldots, u_80\) are similar users, and regions \(x_4\) and \(x_7\) are similar regions. In other words, we can regard \(\{U^{(a)}, U^{(b)}\}\) (Fig. 11) as user profiles, and \(\{V^{(a)}, V^{(b)}\}\) (Fig. 12) as region profiles.

Fig. 13 shows a visualization of traces. The leftmost figure shows all of the traces (80 × 2 traces) used in our analysis. Each location is represented as a dot, and each trace is represented as a line (that has 10 dots). The other 8 figures (A)-(H) show the traces of the users whose model parameters \(v_{i,k}^{(a)}, v_{i,k}^{(b)}\) are more than 0.025. In the Geolife dataset, many traces are concentrated around the office of Microsoft Research Asia at the northwestern part of region \(x_{11}\). In the Gowalla dataset, many traces are around Manhattan, which is at the western part of region \(x_{7}\).

It can be seen that the model parameters really capture similar users and similar regions. For example, it can be seen from Fig. 11(i)(F), 12(i)(F), and 13(i)(F) that the users who have high \(u_{n,2}^{(b)}\) (i.e. \(u_3, u_7, \ldots, u_{70}\)) really stay in region \(x_{11}\) (i.e. around the office) in the Geolife dataset. It can also be seen from Fig. 11(i)(H), 12(i)(H), and 13(i)(H) that the users who have high \(u_{n,3}^{(b)}\) (i.e. \(u_4, u_5, \ldots, u_{80}\)) really go to \(x_7\) (i.e. the northeast of the office). Since one of them goes to \(x_4\), the others may also be interested in \(x_4\). A similar tendency can be observed in the Gowalla dataset (e.g. it can be seen from Fig. 11(ii)(F), 12(ii)(F), and 13(ii)(F) that users \(u_{23}\) and \(u_{20}\) go to region \(x_{8}\), and one of them also goes to \(x_{12}\). FC propagates such unique features among similar users and similar regions in this way, and exploits this to derive a location. We believe this is the reason FC provided the best performance in the Geolife dataset and the Gowalla dataset.
Fig. 11. Visualization of $\{U^{(a)}, U^{(b)}\} = \{u_{n,k}^{(a)}, v_{n,k}^{(b)}| n \in [N], k \in [K]\}$. The elements whose corresponding model parameters are more than 0.025 are filled with gray (e.g. $u_{8,1}^{(a)} > 0.025$ in the Geolife dataset).

Fig. 12. Visualization of $\{V^{(a)}, V^{(b)}\} = \{u_{i,k}^{(a)}, v_{i,k}^{(b)}| i \in [M], k \in [K]\}$. The value of each model parameter is written in the corresponding element. The elements whose corresponding model parameters are more than 0 are filled with gray (e.g. $v_{4,2}^{(a)} = 0.12 > 0$ in the Geolife dataset).

Fig. 13. Visualization of traces. The leftmost figure contains all of the traces ($80 \times 2$ traces). The other 8 figures ((A)-(H)) contain the traces of the users whose model parameters $u_{1,1}^{(a)}, u_{2,2}^{(a)}, \cdots, u_{10,1}^{(a)}$ or $v_{1,1}^{(b)}$ are more than 0.025. In the Geolife dataset, the office is located at the northwestern part of region $x_{11}$. In the Gowalla dataset, Manhattan is at the western part of region $x_{7}$.
VII. RELATED WORK

There are a number of studies in the field of location privacy; surveys can be found in [12]–[14]. The topics of these studies can be roughly classified into the following three categories: attacks, defenses, and privacy metrics.

Location privacy attacks based on a Markov chain model have been widely studied in the literature [15]–[21]. Based on this model, the adversary can carry out various kinds of attacks: location prediction attacks [16], [17], localization attacks [21], de-anonymization attacks [18]–[21], etc. Some studies showed that this model works very well. For example, Song et al. [15] compared several location predictors, and showed that the low-order Markov-based predictor outperformed the others. Shokri et al. [18] showed that an adversary who knows personalized transition matrices of the users de-anonymizes and de-obfuscates traces with higher accuracy than an adversary who only knows each user’s prior probabilities. Mulder et al. [20] de-anonymized 100 users with a success rate of 77% to 88% using traces in a period of one month as training data. Gamsb et al. [19] de-anonymized 59 users in the Geolife dataset [23] with a success rate of 45% using half of the dataset (over two years) as training data. However, the training data can be extremely sparse in reality, as we have noted repeatedly. Thus we proposed a learning method using tensor factorization (or matrix factorization) to address this problem.

There are also a number of defenses that have been proposed to protect location privacy. They can be broadly divided into two categories: anonymization and obfuscation. Well-known anonymization methods include k-anonymity based methods [34]–[36] and the mix-zone method [46], [47]. As for obfuscation, it can be further classified into the following four categories: perturbation (adding noise) [37]–[39], location generalization (merging regions) [21], [40], [41], location hiding (deleting) [16], [21], [42], and adding dummy locations [40], [48], [50]. In our experiments, we used not an anonymization method but an obfuscation method (i.e. location generalization and location hiding method [21]), since the localization attack is an attack against traces that are not anonymized but obfuscated.

To evaluate the effectiveness of location privacy attacks or defenses, several location privacy metrics have been proposed in the literature. Well-known metrics include k-anonymity based metrics [34], [35], entropy-based metrics [46], error-based metrics [21], [51], and differential privacy-based metrics [38], [39], [52]–[54]. Shokri et al. [55] pointed out that k-anonymity and entropy are not adequate to measure location privacy. They also showed in [21] that these metrics are not correlated with the attack success rate. In our experiments, we used the attack success rate as a metric of location privacy.

Finally, apart from location privacy, we note that tensor (or matrix) factorization (surveys can be found in [27], [30]) is widely studied in the field of machine learning. However, no studies have attempted to estimate transition probabilities using tensor (or matrix) factorization, to the best of our knowledge. Thus we proposed a learning method that factorizes a transition count tensor (or matrix), and then normalizes counts to probabilities.

VIII. CONCLUSION

In this paper, we proposed a learning method using tensor factorization to overcome the sparse data problem in the personalized matrix mode. We also extended it to the population matrix mode by using matrix factorization. We focused on the localization attack [21], and compared our learning method with the ML estimation method in both the personalized matrix mode and the population matrix mode (i.e. FC, FC*, ML, ML*). The experimental results using four real datasets showed that while the ML estimation method performs only as well as a random guess in many cases, our learning method significantly outperforms the ML estimation method in all of the four datasets, which strongly supports the effectiveness of our learning method. We also showed that our learning method in the personalized matrix mode captures unique features of each user’s behavior in the Geolife dataset and the Gowalla dataset by visualizing model parameters and traces.

Although we focus on the fact that the amount of training data can be very small, it can also happen in reality that the training traces contain missing locations [21]. As future work, we would like to extend our learning method to be able to compute model parameters in tensor factorization (or matrix factorization) while estimating missing locations in the training traces to further improve accuracy in such cases.

ACKNOWLEDGEMENT

The authors would like to thank Jacob Schuldt (AIST), Atsunori Kanemura (AIST), Shotaro Akaho (AIST), and Hideitsu Hino (University of Tsukuba) for technical comments on this paper. Also, this study was supported in part by Okawa Foundation Research Grant.

REFERENCES

Takao Murakami is a researcher of Information Technology Research Institute (ITRI), National Institute of Advanced Industrial Science and Technology (AIST). He received the BS degree, the MS degree, and Ph.D degree from the University of Tokyo in 2004, 2006, and 2014, respectively. He joined Hitachi, Ltd. in 2006, and AIST in 2014. His research interests include location privacy, biometrics, and information security. He received the Yamashita SIG Research Award from the Information Processing Society of Japan (IPSJ) in 2011, and the IEEE TrustCom’15 Best Paper Award in 2015. He also received the Dean’s Award of Graduate School of Information Science and Technology, the University of Tokyo, for his Ph.D thesis in 2014. He is a member of IEICE, IPSJ and IEEE.

Hajime Watanabe is a chief senior researcher of Information Technology Research Institute (ITRI), National Institute of Advanced Industrial Science and Technology (AIST). He received B.E., M.E. and Ph.D. degrees from Osaka University in 1991, 1993 and 1997, respectively. From 1994 to 1999, he was a research associate of NAIST, Japan. In 1999, he joined ETL, Japan (was reorganized into AIST in 2001). His research interests include cryptography and information security. He is a member of IEICE and IPSJ.