DETECT: Mobility Behavior Detection with Spatiotemporal Deep Embedded Neural Networks

Anonymous Author(s)

ABSTRACT

The advances in location-acquisition techniques have enabled the collection of massive trajectory data. Identifying mobility behaviors in these trajectory data is of great economic and social significance in various applications including urban planning, marketing and intelligence. Existing work for trajectory clustering typically relies on similarity measurements using raw spatial and/or temporal features of a trajectory and hence are not applicable to find similar moving behaviors at different spatial and temporal scales. Towards this end, we propose an unsupervised deep learning approach, Deep Embedded TrajEctory ClusTering network (DETECT), for mobility behavior detection. DETECT first identifies representative points in trajectories and then augments these points with geographical context information incorporating semantics relevant to mobility behaviors. Subsequently DETECT jointly learns both the fixed-length representations of augmented trajectories and their cluster assignments using deep neural networks, by first mapping the data to a low-dimensional space and then iteratively optimize a clustering objective. Our experimental results on two real-world datasets show that DETECT has achieved significant improvements over the state-of-the-art methods.

KEYWORDS

trajectory clustering, deep representation learning, geographical embedding

ACM Reference Format:

Anonymous Author(s). 2018. DETECT: Mobility Behavior Detection with Spatiotemporal Deep Embedded Neural Networks . In *Proceedings of (SIGMOD'19).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SIGMOD'19, ,

© 2018 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

With the rapid growth of mobile devices, enormous trajectory data have been collected through Location Based Services (LBS) and geo-tagging applications, such as Google Maps, Uber, Yelp, and Twitter. In these trajectories, massive human mobility behaviors are encoded. For example, commuting from home to work is a mobility behavior. Other examples include commuting for leisure travel, business travel, leisure lunch or dinner, business lunch or dinner, etc.

These mobility behaviors are of tremendous value in various applications. For instance, with the knowledge of users' mobility behaviors, better recommendation engines can be designed by leveraging common latent interests between users based on their similarity in mobility [33]. Likewise, merchants could target their potential users more effectively by mining mobility behaviors from trajectory data [11]. Besides, mobility behaviors in the trajectory data are also important for policy makers and law enforcement to better allocate public resources and detect abnormal or dangerous behaviors [29]. Due to the huge volume of the data, manual labeling relying on human experts is usually prohibitively expensive. Thus it is desirable to detect mobility behaviors with minimal amount of supervision, or preferably in an unsupervised manner. One possible approach is to use unsupervised clustering of trajectories following a manual labeling of clusters. While promising, this approach is challenging due to the following reasons.

First, the mobility behavior is more related to the geographical context of the trajectories than their absolute times and shapes. For instance, the "grocery shopping" behavior can have trajectories with dramatic different times and shapes for different choices of supermarkets. Thus in these cases, knowing the geographical context, i.e., the supermarket, tends to be more informative than the exact shapes and times of the trajectories. However, most traditional trajectory clustering algorithms (e.g., [12, 15, 27, 28]) group trajectories based on raw spatial and temporal distances while the geographical contexts are largely overlooked, which renders them inapplicable to the mobility behavior clustering problem.

In addition, trajectories with the same mobility behavior could exhibit quite different spatial and/or temporal scales. For example, a "commute to work" behavior could take from 10 minutes to up to 1 hour via different transportation modes or even with the same transportation modality but during

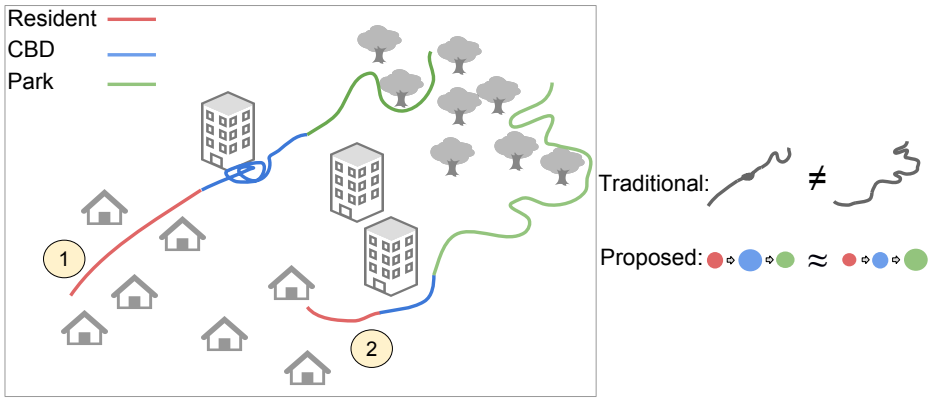


Figure 1: Augmenting trajectories with context information to facilitate mobility behavior based clustering. ***Trajectory 1 and Trajectory 2 both start from residential areas, stay in commercial areas (with different durations), and end in recreational areas. The two trajectories have quite different spatiotemporal scales (left). Raw trajectory data usually fails to provide enough information to capture the similarity between these two trajectories, while augmenting trajectories with context information makes it much easier for clustering algorithms to identify the same mobility behavior in these trajectories (right).***

different times of the day (e.g., rush hours), leading to diverse temporal scales in similar mobility behaviors. Meanwhile, spatial scales could also differ, e.g., from 1 mile to 20 miles, between trajectories of users living near or far away from their workplaces. In those cases, most existing trajectory clustering approaches would fail to accurately cluster mobility behaviors as they are built on distances between raw trajectories which are sensitive to scales. Hence these approaches will likely end up with clusters of different spatiotemporal scales, rather than clusters of different mobility behaviors.

Besides, the context of a trajectory is mainly captured by some representative points rather than those traveling points between stops [13]. Existing approaches, however, treat all points equally, and thus tend to reduce the attention on the critical parts of the trajectory.

Finally, the varying lengths of trajectories also bring more challenges to clustering problem. Since Euclidean distance is not compatible with length-variant and misaligned trajectories [14], many popular clustering approaches are not directly applicable to trajectory clustering. To solve this problem, existing approaches usually utilize special distance measurements that allow flexible alignment between trajectories, such as DTW, LCSS and the Fréchet distance. These alignment-based distance measurements usually allow little temporal disordering, possess great sensitivity to outliers, and incline to local similarities [28].

Therefore, to cluster mobility behaviors in trajectory data, we want to learn a representation with the following features: 1) The representation should include the geographical context of the trajectories. 2) The representation should capture various spatial and temporal scales. 3) The representation should learn useful properties without falling into localities from the variant-length trajectories. To learn such a representation, we propose the Deep Embedded TrajEctory ClusTering network (DETECT). DETECT first identifies representative locations, called "stay points", in a trajectory where best represents mobility behaviors. Subsequently, DETECT augments the trajectory with geographic contexts at these locations (e.g., using POIs from gazetteers). The augmented trajectories incorporate the essential semantics for identifying mobility behaviors (Figure 1). Finally, DETECT learns a fixed-length latent representation using a deep recurrent autoencoder which could capture global dynamics and learn salient properties. DETECT then jointly optimizes both the latent representation and cluster assignments without human intervention. Such joint optimization iteratively and smoothly updates the embedding towards a high confidence clustering distribution and consequently the embedding would result in better separability between mobility behaviors. Figure 2 demonstrates the workflow of how raw trajectories being clustered via different components of DETECT. Our experimental results on two real-world datasets show that DETECT significantly outperforms baseline approaches and aligns better with real-world observations.

In summary, our paper has the following major contributions:

- We propose a deep learning framework for all-scale trajectory clustering on mobility behaviors.

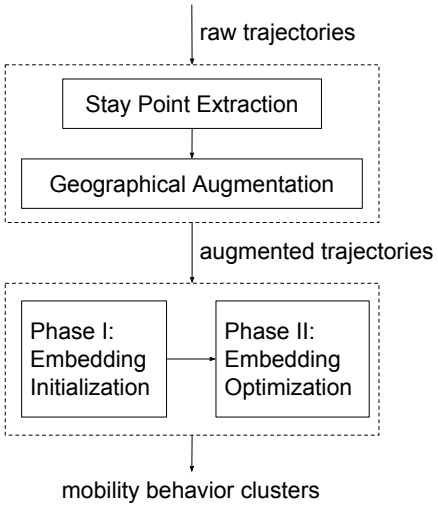


Figure 2: Workflow of the DETECT framework

- We propose a deep embedded neural network which learns a fixed-length representation of mobility behaviors from trajectories, and optimized jointly with clustering assignments inducing a better separability.
- We conducted extensive experiments on two real-world trajectory datasets, and the proposed approach significantly outperforms the state-of-the-art methods with regards to four classic clustering metrics.

The remainder of the paper is organized as follows. Section 2 presents the problem definition, notations, and other preliminaries. Section 3 describes the proposed methodology. Section 4 is a brief introduction of state-of-the-art solutions. Section 5 shows the experimental results including comparing DETECT with the existing methods and ablation study. Section 6 presents the related work followed by conclusions and future works in Section 7.

2 PRELIMINARIES

In this section, we introduce the notations and definitions used in the paper. Table 1 includes the descriptions of main symbols. Below we present the definitions of important terminologies.

Definition 1 (Trajectory). *A trajectory* $s = \{s^{(1)}, s^{(2)}, \ldots, s^{(T)}\}$ *is a time-ordered sequence of spatiotemporal points. Each spatiotemporal point* $s^{(t)}$ *consists of a pair of spatial coordinates, i.e., its latitude, longitude, and a time.*

Definition 2 (Staying Subtrajectory). *A staying subtrajectory* $s^{(i\to j)} \subset s, s^{(i\to j)} = \{s^{(i)}, s^{(i+1)}, \ldots, s^{(j)}\}$ *of trajectory* s *is a sub-sequence of* s*, such that within* $s^{(i\to j)}$*, the trajectory is limited to a range* ρ_s *in space and the duration of* $s^{(i\to j)}$*,* $s^{(j)}.time - s^{(i)}.time$ *is longer than a specific threshold* ρ_t*.*

Table 1: Notations

Symbol	Description
s_i	The i_{th} trajectory
$s_i^{(t)}$	The point at t_{th} timestamp of trajectory s_i
$\dot{s}_i$	The i_{th} stay point trajectory
$\dot{s}_i^{(t)}$	The point at t_{th} timestamp of stay point trajectory $\dot{s}_i$
ρ_s	Space threshold for stay point extraction.
ρ_t	Time threshold for stay point extraction.
x_i	The i_{th} augmented trajectory.
$x_i^{(t)}$	The geographical feature vector at t_{th} timestamp of augmented trajectory x_i
z_i	The latent embedding of the i_{th} trajectory
k	The number of clusters
μ_j	The centroid of the j_{th} cluster.
$Q(\cdot)$	Initial distribution of trajectory embedding
$P(\cdot)$	Auxiliary distribution of trajectory embedding

Definition 3 (Stay Point). *A stay point* $\dot{s}^{(t)}$ *of trajectory* s *is a spatiotemporal point, which is the geometric center of a longest sub-trajectory* $s^{(i\to j)} \subset s, s^{(i\to j)} = \{s^{(i)}, s^{(i+1)}, \ldots, s^{(j)}\}$*, such that* $s^{(i\to j)}$ *is a staying subtrajectory, and neither* $s^{(i-1\to j)}$ *nor* $s^{(i\to j+1)}$ *is a staying subtrajectory.*

Definition 4 (Point of Interest). *A point of interest (POI) is a spatial point with geographical semantics, such as a high school, a business office or a wholesale club. Each POI has a spatial coordinate and a major category, such as education, commerce and shopping.*

Definition 5 (Geographical Feature Vector). *A geographical feature vector* $x_i^{(t)} = \{x_{i,1}^{(t)}, x_{i,2}^{(t)}, \ldots, x_{i,M}^{(t)}\}$ *represents the POI histogram around point* $\dot{s}_i^{(t)}$*.* $x_{i,m}^{(t)}$ *denotes the weight of the* m_{th} *POI category.*

3 METHODOLOGY

In this section, we elaborate on the details of each module of the DETECT framework. We first describe the stay point detection module, in which we propose an efficient stay point detection, Fast-SPD. Then, we demonstrate the geographical augmentation module, in which we augment the result of Fast-SPD with geographical context. Finally, we present our deep embedded trajectory clustering network module, which first learns an embedding from the augmented trajectories and jointly optimize the embedding with clustering objectives.

3.1 Stay point detection

In order to learn an all-scale representation of trajectories, we first need to identify and preserve the essential locations from the trajectories with various spatiotemporal scales. The diversity of scales usually comes from different transportation modes and personal preferences during the traveling between the stops. Specifically, in a trajectory, many points are collected when the user is traveling between their stops either by car, by train, on foot, or by other transportation methods, through different routes. These points are not very helpful in identifying the context of the trajectory (i.e., the moving behavior). This is because they are less correlated with the context of the stops where most activities of the mobility behavior are conducted. Therefore, we develop a stay point detection approach, which typically extracts representative points from a trajectory. Li et al. [13] proposed a Stay Point Detection (SPD) algorithm. While SPD can detect meaningful stay points, the algorithm has an $O(T^2)$ (T is the length of the trajectory) computational complexity and hence does not scale to handle large numbers of lengthy trajectories. We propose Fast-SPD, which utilizes spatial buffers and pre-generated spatial indexes to extend SPD for scalable stay point detection. The spatial indexes (e.g., R-Tree) can help reduce the search space for identifying trajectory points close in space and time.

Algorithm 1: Fast-SPD(s, ρ_s, ρ_t)

Data: Raw trajectory s
Result: Stay point trajectory $\dot{s}$

```
1  begin
2      i ← 0
3      ṡ ← {}
4      while i < len(s) − 1 do
5          if dist(s^(i), s^(i+1)) > ρ_s then
6              i ← i + 1
7              continue
8          end
9          cands ← s^(i+1→T) ∩ b(s^(i), ρ_s)
10         neighbors ← CompareIdx(cands, s^(i+1→T))
11         if last(neighbors).time − s^(i).time > ρ_t then
12             neighbors ← neighbors ∪ {s^(i)}
13             ṡ ← s ∪ {average(neighbors)}
14             i ← i + len(neighbors)
15         end
16         i ← i + 1
17     end
18     ṡ ← {s^(0)} ∪ ṡ ∪ {s^(T)}
19     return ṡ
20 end
```

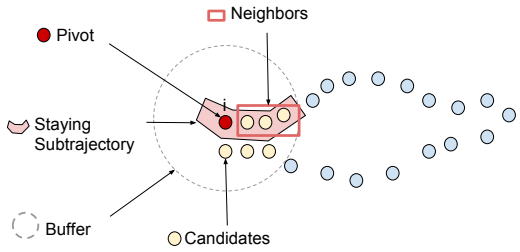


Figure 3: Accelerated Stay Point Detection: Fast-SPD

Figure 3 illustrates an example that we use to explain the procedure of Algorithm 1. In general, Fast-SPD loops through a trajectory, finds all staying subtrajectories, and then generates stay points from them. Specifically, Fast-SPD first scans the trajectory, and selects a pivot point $s^{(i)}$ (the red point) to start the search. In the meantime, a spatial buffer (dotted circle) is created with a radius ρ_s around the pivot $s^{(i)}$. Next, the algorithm uses R-tree to spatially join the buffer with the remaining points of the trajectory and finds candidate points (yellow points) within ρ_s (Line 9 in Algorithm 1). In the identified candidates, only those consecutive candidates are actual neighbors (yellow points in the red-border box), i.e., the following consecutive points within ρ_s. Only these neighbors can form a staying subtrajectory. The neighbors are identified by comparing the index between the candidates and the rest of trajectory (Line 10). If these neighbors cover a long enough period in time, i.e., $last(neighbors).time - s^{(i)}.time > \rho_t$, the neighbors with the pivot point $s^{(i)}$ together constitute a staying subtrajectory (the pink box) (Line 12). Subsequently, the algorithm extracts the geometric center of the staying subtrajectory as a stay point. Fast-SPD skips the neighbors and continues scanning until the end of the trajectory (Line 14). The combination of the spatial join (range query) and the neighborhood skipping reduces the time complexity by $O(c) - O(log(T))$ at each search of stay point comparing to SPD, where c (usually $c > log(T)$) is the length of the staying subtrajectories, and T is the length of the trajectory.

3.2 Geographical augmentation

To build a mobility behavior representation, we need to incorporate mobility semantics in the trajectories. After the stay points are extracted, DETECT augments the trajectory with geographical context at these stay points. The geographical context could be Point Of Interest (POI) data collected from different sources, such as GeoName and OpenStreetMap [8]. These POIs are often defined with categories based on their use. Besides, spatial buffers are effective in generating the geographical context for characterizing the environment of

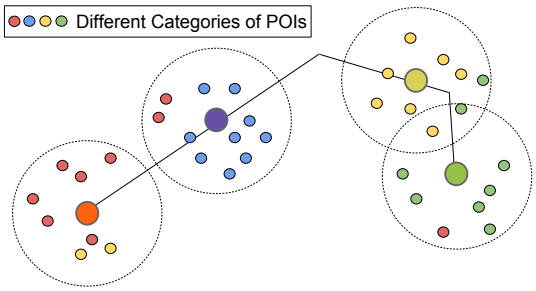


Figure 4: Augment stay points with geographical features

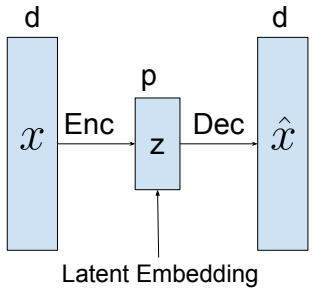


Figure 5: Example of Autoencoder structure: *The neural network is trained to reconstruct the input x , through a latent embedding z, to $\hat{x}$ with minimum distortion* $\sum_i^n (x_i - \hat{x}_i)^2$.

a location (e.g., in [16]). Similarly, DETECT builds the geographic context for stay points of a trajectory using spatial buffers and available geographic data. Not that is necessary to use a buffer rather than the closest POI, because the closest POI would be very sensitive to outliers, while the spatial buffer treats the POIs more loosely. In detail, as shown in Figure 4, for each stay point (large solid points), the algorithm creates a buffer (circles) centered at the stay point. Next, within each buffer, the algorithm counts different categories of POIs (small points in different colors) and normalizes by the global maximum count within the buffer. Thus each stay point is augmented with local geographic context.

Formally, suppose $\dot{s} = \{\dot{s}^{(t)}\}_{t=1}^{T}$ is a trajectory after stay point extraction, for each point $\dot{s}^{(t)}$, a buffer $b(\rho_{poi}, \dot{s}^{(t)})$ with radius ρ_{poi} centered at $\dot{s}^{(t)}$ is generated. All POIs annotated with M categories within the buffer are counted, and normalized to a vector $x_i^{(t)} = \{x_{i,1}^{(t)}, x_{i,2}^{(t)}, \ldots, x_{i,m}^{(t)}\}$. $x_{i,m}^{(t)}$ denotes the cardinality of the m_{th} POI category, which can be "Education", "Amusement", "Residence", etc. Therefore the augmented trajectories $\{x_i\}_{i=1}^{n}$ (n is the total number of trajectories) could encompass mobility contexts which are essential to clustering mobility behaviors.

3.3 Deep Embedded Trajectory Clustering Network

Now that the augmented trajectories incorporate geographical context and work well in various scales, our remaining task is to learn a representation encoding the mobility behaviors from the augmented trajectories. This section introduces the core module in DETECT which solves the remaining task. The input of this step is the augmented trajectories with various lengths. We propose a two-phase deep embedded trajectory clustering network, which maps the augmented trajectories to a low-dimensional mobility behavior space, thus produces an accurate clustering of mobility behaviors. In the rest of this section, we first introduce some fundamental knowledge of RNN and autoencoder, based on which we built DETECT Phase I. Then we describe the Phase I, which learns a fixed length latent embedding from the augmented trajectories through recurrent autoencoder. Finally we describe Phase II, an optimization step which jointly learns an embedding with clustering assignments for better separation of mobility behaviors.

3.3.1 Introduction to RNN and autoencoder. DETECT Phase I aims to represent the augmented trajectories by a fixed-length embedding. However, general autoencoder works on learning a fixed-length embedding from fixed-length vector data. Meanwhile Recurrent Neural Network (RNN) can capture the dynamics in variant-length sequences. Thus we adopt an RNN autoencoder in our model. Below we start with a brief introduction of autoencoder and RNN.

To learn a latent embedding, autoencoders are prevalently adopted in unsupervised learning with deep architectures [2]. The fundamental task of an autoencoder is to transform the input data to itself through low-dimensional feature space with minimum distortion. The structure of an autoencoders is depicted as in Figure 5. $x = \{x_1, x_2, \ldots x_n\}$, $\hat{x} = \{\hat{x}_1, \hat{x}_2, \ldots \hat{x}_n\}, x_i, \hat{x}_i \in \mathbb{R}^d$ are the input data and output data with d dimensions, and $z = \{z_1, z_2, \ldots z_n\}, z_i \in \mathbb{R}^p$ is the embedding data with dimension p (usually $p < d$). The task is to first encode x to embedding z through a class of functions named Encoder (Enc), then decode z to $\hat{x}$ through another class of functions named Decoder (Dec). And the goal is to minimize the difference between x and $\hat{x}$, i.e., $\min_{Enc,Dec} \sum_i^n (x_i - \hat{x}_i)^2$. Once the model is well trained, the latent embedding z will be used for unsupervised learning, e.g. clustering. However, the input of general autoencoder is usually fixed-length vectors, which is not applicable to our case. Thus we use RNN for encoder and decoder to learn the embedding from variant-length augmented trajectories.

Recent researches show that RNN has competitive advantages in capturing the dynamics of *variant-length* sequences

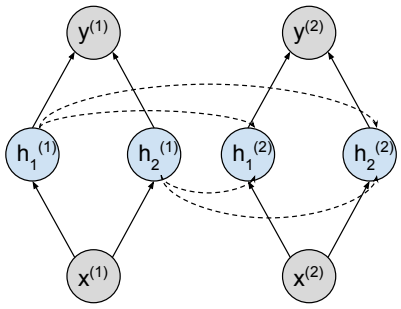


Figure 6: Example of an RNN structure with two units: *the hidden states* $h_1^{(2)}$ *and* $h_2^{(2)}$ *at timestamp* $t = 2$ *are updated by the previous hidden states* $h_1^{(1)}$ *and* $h_2^{(1)}$ *at timestamp* $t = 1$*, and the current input* $x^{(2)}$.

via cycles in the neural network [17]. RNN is a special feed-forward neural networks, which builds edges between different timestamps. Such connectivity brings RNN the notion of time. The structure of a simple RNN can be represented in Figure 6. The general target of an RNN is to predict output sequence $y = \{y^{(1)}, y^{(2)}, \dots y^{(T)}\}$, given a sequence $x = \{x^{(1)}, x^{(2)}, \dots x^{(T)}\}$. At each time stamp t, RNN updates the hidden node $h^{(t)}$ based on previous states $h^{(t-1)}$ and current value $x^{(t)}$, as in Equation (1). Here σ is an activation function. As shown in Figure 6, the hidden states $h_1^{(2)}$ and $h_2^{(2)}$ at timestamp $t = 2$ are updated by their previous hidden states $h_1^{(1)}$ and $h_2^{(1)}$ and the current input $x^{(2)}$.

$$h^{(t)} = \sigma(h^{(t-1)}, x^{(t)}) \tag{1}$$

Next, RNN predicts $\hat{y}^{(t)}$ from the current hidden state $h^{(t)}$:

$$\hat{y}^{(t)} = softmax(wh^{(t)} + b) \tag{2}$$

The prediction of $\hat{y}^{(t)}$ is actually affected by all previous timestamps $x^{(t-1)}, x^{(t-2)}, \dots$ by means of the recurrent connections. Long Short-Term Memory (LSTM) is a special RNN which has a better capability in learning the long-term dependencies [17], which is required in our scenario. Thus we use LSTM in our Phase I to better capture the dynamics.

3.3.2 Phase I - Recurrent autoencoder based clustering. In Phase I, we use the recurrent autoencoder to initialize a latent representation from variant-length augmented trajectories, and then we initialize the cluster assignments using kMeans based on the representation. Formally, the input of Phase I is a set of n augmented trajectories, $\{x_i\}_{i=1}^n$, where $x_i = \{x_i^{(t)}\}_{t=1}^{T_i}$ denotes the augmented trajectory with length T_i. The output is k clusters each represented by a centroid $\mu_j \in \{\mu_j\}_{j=1}^k$. The process includes two steps from the input trajectories to output clusters: 1) learning a fixed-length latent embedding using the recurrent autoencoder, and 2) applying kMeans on the embeddings of the trajectories. The recurrent autoencoder model consists of a recurrent encoder and a recurrent decoder. Similar to a general autoencoder, the task of the recurrent autoencoder is to encode the input trajectories to a latent embedding via the encoder and then reconstruct the trajectories based on the embedding. The model is trained to minimize the error of such reconstruction, and consequently the embedding could capture useful properties and dynamics of mobility semantics in the augmented trajectories. Next, we illustrate how the recurrent encoder and decoder work in the reconstruction process, as depicted in Figure 7. First, each augmented trajectory $x_i = \{x_i^{(t)}\}_{t=1}^{T_i}$ is fed to the recurrent encoder sequentially. The recurrent encoder will update the hidden state $h_{enc}^{(t)}$ and other parameters at each timestamp t, $h_{enc}^{(t)} = \sigma(h_{enc}^{(t-1)}, x^{(t)})$. After that, the decoder will reconstruct the trajectories with $h_{enc}^{(T_i)}$ as its initial state, $h_{dec}^{(1)} = h_{enc}^{(T_i)}$. With $h_{dec}^{(1)}$, the decoder will first generate $\hat{x}^{(1)}$. Then following hidden states and reconstructed trajectories are generated sequentially by: $h_{dec}^{(t)} = \sigma(h_{dec}^{(t-1)}, \hat{x}^{(t-1)}), \hat{x}^{(t)} = f_{dec}(h_{dec}^{(t)})$. Hence, the hidden state $h_{enc}^{(T_i)}$ connects the encoder and decoder, which are trained together to minimize the reconstruction error:

$$\ell = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{T_i}\sum_{t=1}^{T_i}(x_i^{(t)} - \hat{x}_i^{(t)})^2 \tag{3}$$

The last hidden state $h_{enc}^{(T_i)}$ of the recurrent encoder is designed to have all information about the entire augmented trajectory, since it's trained to faithfully reconstruct the augmented trajectory. Indeed, $h_{enc}^{(T_i)}$ can be the hidden embedding z of the trajectories. In phase I, we map the augmented trajectories $x_1, x_2, \dots, x_n$ to representative fix-length embeddings $z_1, z_2, \dots, z_n$ and then apply kMeans on the embedding space to create the initial cluster assignments. After we get the trained encoder along with the initial embedding and cluster assignments, we want to jointly learn the embedding and the cluster assignment further to get better separability of clusters.

3.3.3 Phase II - Optimization of clustering. In Phase I, a latent embedding is learned by minimizing the reconstruction error, however, the clustering objective is overlooked. Thus, the learned embeddings as well the as the generated clusters are not necessarily the best for clustering with regards to different mobility behaviors. Recent studies [9, 10, 25] show that deep neural network is able to improve clustering by optimizing an unsupervised objective. This is based on the assumption that in the initial clusters, points that are very close to the centroid are likely to be correctly predicted/clustered, i.e., high confidence prediction. Then the model improves

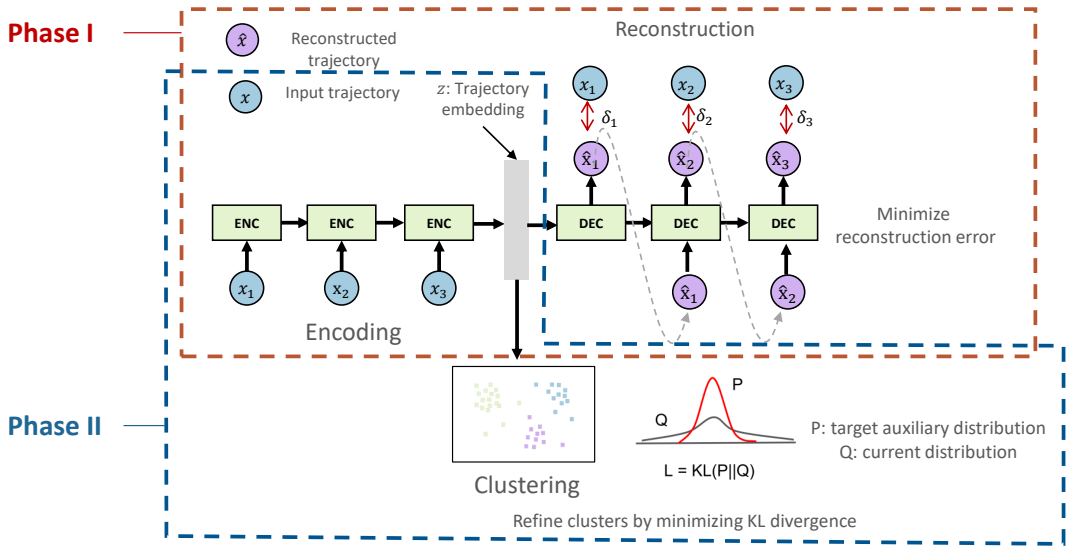


Figure 7: DETECT Phase I and Phase II: ***The augmented trajectories are fed to the recurrent autoencoder in Phase I. Phase I learns a hidden embedding z of the trajectories, and use z to generate initial cluster assignments by kMeans. Subsequently, from the initial embedding and cluster assignments, Phase II jointly learns the embedding (also the encoder) and the cluster assignments to a target auxiliary distribution P.***

the clustering iteratively by learning from high confidence predictions, which in turn helps to improve low confidence ones. Inspired by this idea, we further optimize the embeddings learned by recurrent autoencoder for better clustering cleanness in Phase II. Specifically, in Phase II, DETECT optimizes the learned representation, z, and cluster assignment by iterative minimizing the distance between the current cluster distribution Q and an auxiliary target cluster distribution P, i.e., the distribution derived from high confidence predictions. Details of such procedure are illustrated in the following paragraphs.

First, to calculate the current cluster distribution Q, we use the Student t-distribution which assigns higher confidence to points close to centroids. Suppose $z_i = f_\Theta(x_i)$ is the embedding of augmented trajectory x_i learned by the recurrent encoder in Phase I with parameters Θ, and μ_j denotes the j_{th} centroid in the current cluster. Equation 4 shows the equation to calculate the current distribution,

$$q_{ij} = \frac{(1 + ||z_i - \mu_j||^2)^{-1}}{\sum_{j'} (1 + ||z_i - \mu_{j'}||^2)^{-1}} \tag{4}$$

where q_{ij} as the probability of assigning z_i to centroid μ_j.

In the current cluster distribution, the low-confidence points might be assigned to the incorrect clusters since the embedding is not optimized for clustering. Thus, the cluster assignment and the embedding should be smoothly refined to have better clustering cleanness by learning from the distribution of high-confidence points. Therefore, it is necessary to define a target distribution, which should be able to (1) reinforce predictions (i.e., better cluster purity), (2) highly emphasis on data points assigned with high confidence, and (3) has a normalized loss contribution in case the embedding is highly affected by the sizes of clusters.

Equation (5) shows the calculation of the target auxiliary distribution P is where p_i is computed by first raising q_i to the second power and then normalizing by the frequency per cluster. Intuitively, the normalized quadratic distribution will put more emphasis on points with high probabilities.

$$p_{ij} = \frac{q_{ij}^2 / \sum_{i'} q_{i'j}}{\sum_{j'} (q_{ij'}^2 / \sum_{i'} q_{i'j'})} \tag{5}$$

To measure the distance between two distributions P and Q, we use KL divergence (Equation (6)) which is one of the most popular metric of distribution distances. Following the objective defined in Equation (6), we iteratively minimize the distance between the soft assignment Q and auxiliary distribution P by jointly learn the embedding, the cluster assignment, and correspondingly Θ of the recurrent encoder.

$$\ell = KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{6}$$

Accordingly, the cluster assignment and the recurrent autoencoder are jointly learned using the Stochastic Gradient Descent optimization minimizing the loss defined by K-L divergence. The loss will first propagate to z_i and μ_j through

gradients computed as in Equation(7). Then the gradients $\frac{\partial \ell}{\partial z_i}$ are propagated backward to the recurrent encoder and update Θ by $\frac{\partial \ell}{\partial \Theta}$

$$\begin{aligned}\frac{\partial \ell}{\partial z_i} &= 2\sum_{j=1}^{k}(z_i-\mu_j)(p_{ij}-q_{ij})(1+\left\|z_i-\mu_j\right\|^2)^{-1}\\ \frac{\partial \ell}{\partial \mu_i} &= 2\sum_{i=1}^{n}(z_i-\mu_j)(q_{ij}-p_{ij})(1+\left\|z_i-\mu_j\right\|^2)^{-1}\end{aligned} \tag{7}$$

The results from DETECT Phase II are updated trajectory cluster assignments along with the embeddings. Each cluster contains the trajectories with similar mobility behaviors. Now, the DETECT framework has learned a fix-length representation of trajectories for clustering mobility behaviors. The whole framework is fully unsupervised from augmenting the input trajectories to the final clustering results.

4 BASELINE APPROACHES

In this section, we introduce the state-of-the-art trajectory clustering approaches which we will compare with DETECT in our experiments. Traditional trajectory clustering approaches mostly adopt clustering methods used in sequence clustering, especially time series clustering [28]. These approaches mainly utilize a clustering technique with a distance measurement designed for sequential data. These clustering approaches include centroid-based approaches (e.g., kMeans, kMedoids), density-based approaches (e.g., DBSCAN), hierarchical approaches (e.g., Agglomerative Clustering), and spectral clustering approach. The distance measurements, however, are designed and adapted to different problems because the traditional pairwise distances such as Euclidean distance and Cosine distance are not applicable to variant-length sequences. Thus, researchers adopted measurements that could calculate distances between sequences with different lengths, e.g., Dynamic Time Warping (DTW) distance, Hausdorff distance, Longest Common SubSequence (LCSS) distance. In this section, we will briefly introduce some clustering approaches and distance measurements for sequences.

KMeans and DBSCAN are two popular clustering approaches. KMeans is a centroid-based clustering approach. The basic idea of kMeans is to find each instance the right cluster by assigning it to its nearest centroid and update these centroids. The algorithm assumes the number of clusters k is provided and initializes k random centroids. Then the algorithm employs an iterative clustering process, each iteration includes two steps: a) every instance is assigned to the closest centroid, b) all centroids are updated by averaging the instances in the same cluster. Such iterations continue until a criterion is reached, e.g., the assignment of clusters does not change. Unlike kMeans, DBSCAN is a density-based clustering approach which does not require the knowledge of the number of clusters. It relies on the user's preference on the size and tightness of clusters. The algorithm defines reachability to determine if the instances are close enough to each other. In detail, p is reachable to q if $dist(p,q) < \epsilon$. Such reachability is transitive, i.e., r is reachable to q if r is reachable to p and p is reachable to q. Thus the instances connected by the reachability form a cluster. Also, those instances not connected to any cluster will be marked as outliers. In this way, dense regions automatically generate clusters, and therefore the boundaries of clusters can be any shapes. This property makes DBSCAN sometimes preferable to kMeans since the boundaries of kMeans have to be convex shapes.

In distance measurements for sequences, DTW is one designed to capture the shape of a sequence, by allowing warping among different timestamps. The algorithm is based on dynamic programming, which finds the best alignment between two trajectories and minimizes their distance. Different from the Euclidean distance, which computes the distance between two points at the same slice, DTW allows one point in sequence s_1 to find the matching point in sequence s_2 at a different slice, while keeping the order of the mapping. Figure 8 shows the difference between the DTW and Euclidean distances, as well as the DTW alignment between two sequences. Under the hood, the DTW distance between s_1 and s_2 is calculated by the recursive equation as shown in Equation (8). An adaptive window (the blue area in Figure 8b) is often utilized to avoid over-distortion. We can observe in Figure 8a that the DTW distance is more capable than the Euclidean distance in capturing the shape and invariant to sequential shifts.

$$DTW(i,j) = dist(s_1^{(i)}, s_2^{(j)}) + min\begin{cases} DTW(i-1,j) \\ DTW(i,j-1) \\ DTW(i-1,j-1)\end{cases} \tag{8}$$

LCSS is a distance measurement originally used for natural language sequences, where the task is finding the maximum length of common substrings between two sentences. To extend LCSS to numeric sequential data, a threshold ϵ has to be determined to measure the similarity between points from each sequence. Specifically, given two sequences s_1, s_2, we seek for the longest sub-sequences $s_1' \subset s_1$, $s_2' \subset s_2$, such that $|s_1'| = |s_2'|$, and all corresponding pair-wise distances between s_1' and s_2' are less or equal to ϵ. The length of such subsequences is usually computed recursively as shown in Equation (9). The length is usually normalized to $\frac{LCSS(s_1,s_2)}{min(|s_1|,|s_2|)}$, so variant lengths can be mapped to $[0,1]$. In addition, the calculated result describes the similarity between two trajectories. Thus to define a distance measurement, people usually define the LCSS distance as $1 - \frac{LCSS(s_1,s_2)}{min(|s_1|,|s_2|)}$.

$$LCSS(s_1^{(i)}, s_2^{(j)}) = \begin{cases} 0 & i=0 \text{ or } j=0 \\ 1+LCSS(s_1^{(i-1)}, s_2^{(j-1)}) & dist(s_1^{(i)}, s_2^{(j)}) \le \epsilon \\ max(LCSS(s_1^{(i-1)}, s_2^{(j)}), LCSS(s_1^{(i)}, s_2^{(j-1)})) & otherwise\end{cases} \tag{9}$$

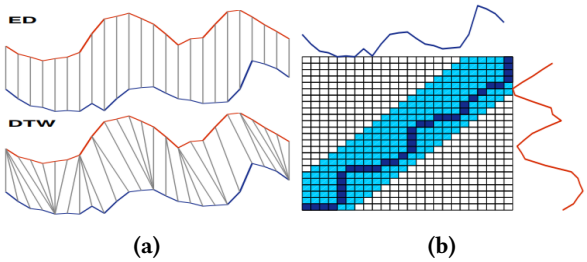


Figure 8: Alignment between two sequences: (a) difference between Euclidean and DTW (b) DTW alignment path [21]

5 EXPERIMENTAL RESULTS

In this section, we first present the datasets (Section 5.1), the baseline approaches with the settings of their parameters and the computing environments used in the experiments (Section 5.2). In Section 5.3, we introduce the metrics used to evaluate all approaches. In Section 5.4, we study the effect of stay point detection and geographical augmentation. Next, in Section 5.5, we compare DETECT with baseline approaches, based on four commonly used clustering metrics. Subsequently, we present various ablation studies to better understand the proposed model. Finally, we visualize both the embeddings learned by DETECT with t-SNE [18], and the raw trajectories on the real world map to provide more intuitions of the learned morel.

5.1 Datasets

Trajectory Data. Though there are plenty of open trajectory datasets, most of them, e.g., taxi and ride-sharing data [6], cannot be directly used in this experiment. This is because these datasets have limited information about passengers, while the identification of passenger is important in our problem [1]. Besides, moving object generators, e.g., [19], are not suitable either, because these generators are generally designed for synthesizing traffic in networks without considering the characteristics of individual routes. In other words, the synthetic data do not contain mobility behaviors.

After an exhaustive search, we find two datasets containing enough metadata for evaluating our proposed approach: the GeoLife dataset [30–32] and the DMCL dataset [20]. The GeoLife dataset consists of 17,621 trajectories generated by 182 users from April 2007 to August 2012. Some of them are labeled with "transportation mode", such as "Walk", "Bus". However, such labels are not useful in our mobility behavior problem, so we manually labeled a randomly selected subset of these datasets. In the GeoLife dataset, the subset contains 601 trajectories from 11 users while the DMCL dataset has 90 complete trajectories generated by two users over six months. In the GeoLife dataset, we identify six mobility behaviors: "campus activities", "hangouts", "dining activities", "healthcare activities", "working commutes", "studying commutes". While in the DMCL dataset, we identify four mobility behaviors: "studying commutes", "residential activities", "campus activities", "hangouts". The manual labels of the data are created by observing the trajectories on a map, before experiments and without any knowledge of the labels by any clustering approach. We assign the labels to the trajectories one after the other by 1) visualizing a trajectory on the map by Mapbox Gl[2], and 2) looking for where the trajectory stays and inferring what activity it involves by inspecting the use of surrounding buildings on the Google Maps. The readers may find and download our labeled data on the website [3].

Geographical Data. For the GeoLife dataset, we retrieve POI data from the PKU Open Research Data [4]. This dataset contains more than 14,000 POIs in Beijing, falling into 22 major categories including "education", "transportation", "company", "shopping", etc. For the DMCL dataset, we obtain POI data from OpenStreetMap (OSM) [8]. The POI dataset contains 30,401 POIs in Illinois, subject to 9 major categories such as "public", "accommodation".

5.2 Baseline approaches and environmental setup

kMeans+DTW. We choose the popular combination of kMeans clustering and the DTW distance as our first baseline approach. DBA [22] is adopted in calculating the centroids of clusters in each iteration. We set the number of cluster k equal to the number of manually labeled classes, i.e., $k = 4$ for the DMCL dataset, and $k = 6$ for the GeoLife dataset.

DBSCAN+LCSS. The second baseline is DBSCAN with LCSS as the distance metric. We tune the hyperparameters of DBSCAN+LCSS to generate the same number of clusters as used in kMeans. In experiments on the GeoLife dataset, We set $\epsilon_{LCSS} = 0.15$ as the common sequence threshold for LCSS, $\epsilon_{DBSCAN} = 0.03$ and $min\_samples = 18$ as the neighborhood thresholds. In experiments on the DMCL dataset, we set $\epsilon_{LCSS} = 1.5e{-}6$ and $\epsilon_{DBSCAN} = 1e{-}6$ and $min\_samples = 2$.

We implemente our approaches on a computer with an Intel Core i7-8850H CPU, a 16 GB RAM and an NVIDIA GeForce GTX 1080 GPU. We implement the KMeans+DTW using tslearn [23], and DBSCAN clustering using Sklearn

[1]For instance, three consecutive trips of a person will be separated into three trajectories in the taxi data, and we have no means to identify which three trajectories were generated by this person, much less to make up the entire mobility behavior.

[2]https://github.com/mapbox/mapboxgl-jupyter

[3]http://goo.gl/VkUjVP

with LCSS distance [4]. The proposed deep embedding network is built using Keras [5] with Tensorflow [1].

5.3 Evaluation metrics

The evaluation of clustering performance mainly consists of internal metrics and external metrics. The difference is the knowledge of ground truth cluster assignments. Internal metrics such as Silhouette coefficient and Davies-Bouldin index assume the true assignments are unknown and evaluate based on the characteristics inherent in the data set. These internal metrics usually depend on the distances between instances within and across clusters. However, in our experiment, the baseline approaches, kMeans+DTW and DBSCAN+LCSS, as well as our proposed deep embedding models have different definitions of pairwise distances. Thus it is not possible to unify these distances and therefore it is unfair to use these internal metrics in the comparison of the approaches. Hence we manually label the DMCL data and a subset of GeoLife data and then evaluate the models by four external metrics: Rand Index (RI), Mutual Information (MI), Purity, and Fowlkes-Mallows Index (FMI). The four metrics reflect the clustering accuracy of the clustering labels with regard to the ground truth classes.

Rand Index measures the simple accuracy, i.e., the percentage of correct prediction of clusters. Mutual Information measures the mutual dependency between the clustering result and the ground truth, i.e., how much information one can infer from the other. Zero mutual information indicates the clustering labels that are independent from the ground truth classes. Purity measures how pure are the clustering results, i.e., whether the trajectories in the same cluster belong to the same ground truth class. Fowlkes-Mallows Index measures the geometric mean of the pairwise precision and recall, which is robust to noises. We use all of these four metrics to evaluate how each clustering approach accurately and clearly divide the trajectories into clusters.

5.4 Effect of stay point detection and geographical augmentation

In this experiment, we show the effectiveness of our augmentation procedures. Four different types of preprocessed trajectories were fed to the baseline approach kMeans+DTW, and their clustering results are compared based on the aforementioned external metrics. The four types include: 1) *raw trajectories*, which are sequences of spatiotemporal points without any preprocessing 2) *stay points only trajectories*, which are sequences of spatiotemporal points extracted from *raw trajectories* by Fast-SPD 3) *geographical only trajectories*, which are sequences of geographical vectors generated at each point in *raw trajectories*, rather than at stay points 4) *fully augmented trajectories*, which are sequences of geographical vectors generated at each stay point extracted from the *raw trajectories*.

[4]https://github.com/maikol-solis/trajectory_distance

Table 2: Performance comparison with different types of trajectories

Data Type	RI	MI	Purity	FMI
Raw trajectory	0.3279	0.6351	0.5787	0.5773
Stay point only	0.3038	0.6753	0.5953	0.5675
Geographical only	0.4421	0.8471	0.6851	0.5889
Augmented trajectory	**0.5167**	**0.9339**	**0.7467**	**0.6292**

We apply the baseline approach kMeans+DTW to the four types of trajectories on GeoLife data. Table 2 shows the clustering performances. It is clear the fully augmented trajectories outperforms all other types of trajectories. The *stay points only trajectories* are better than the *raw trajectory* because the *raw trajectory* is sensitive to the spatial shapes, e.g., two working commutes with different shapes would have large distance using *raw trajectories*. Moreover, the *geographical only trajectories* have better accuracy than *raw trajectories* and *stay point only trajectories*. This is because geographical augmentation captures more mobility context of the trajectories. However, there will be many irrelevant geographical vectors generated by the intermediate periods without the stay point extraction. Therefore, it does not perform as well as *fully augmented trajectories*. Accordingly, both stay point detection and geographical augmentation are effective for preprocessing the trajectory data in the mobility behavior clustering problem.

Additionally, we compare the speeds of SPD and Fast-SPD. Figure 9 depicts the computation time of stay points with respect to the length of the trajectory. We observe that Fast-SPD has similar efficiency on short trajectories(~1k points) but performed much faster as the length of trajectory increases. The reason is that more spatial buffers are used in Fast-SPD as the length of trajectory grows. Specifically, Fast-SPD requires less time than SPD for refining the candidate points to verify and extract a stay point.

5.5 Clustering performance Comparison

We compare DETECT with the baseline approaches kMeans +DTW, DBSCAN+LCSS by four clustering metrics on two datasets. Since we have proved the effectiveness of the augmentation, the inputs of all approaches in this experiment are augmented trajectories. To study the impact of Phase II of our DETECT, we evaluate two variations of DETECT, the first one, termed "DETECT Phase I" only includes the first phase, while the second variant, termed "DETECT" includes both phases I and II. Figure 10 presents the results of each

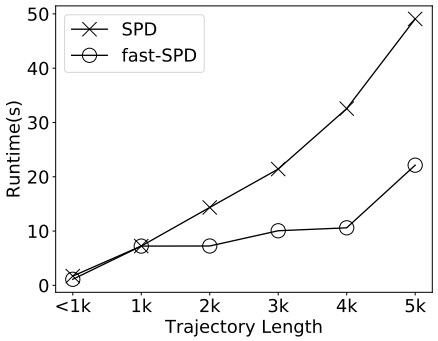


Figure 9: Speed comparison between Fast-SPD and SPD

metric, and the higher value represents better clustering performance. We observe that DETECT outperforms all other approaches on all metrics in both datasets.

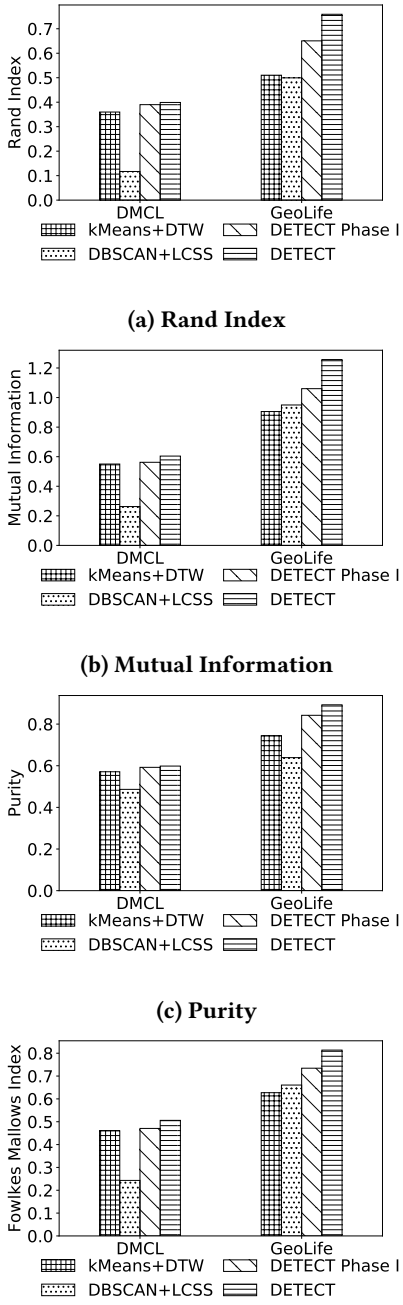
In particular, we observe that 1) DETECT has a more significant advantage over other approaches in GeoLife dataset, than in DMCL dataset. Because the larger dataset (GeoLife) allows more sufficient learning on the representation. 2) DBSCAN+LCSS perform quite differently in the two datasets because both DBSCAN and LCSS are very sensitive to their thresholds [28]. 3) kMeans+DTW is worse than DBSCAN+LCSS in MI and FMI on the GeoLife dataset. One possible explanation is that RI and Purity penalize less on mis-clustered cases than MI and FMI do. It is possible that DBSCAN's non-convex formation of clusters would gain better performance on large clusters, thus gains a higher score in MI and FMI. Therefore, the superiority of DETECT on all metrics across the datasets indicates its stable excellence.

In addition, we also include the intermediate results of DETECT, DETECT Phase I, into the comparison. The results of DETECT Phase I have not been improved by the joint optimization with clustering assignment. As expected, DETECT Phase I has worse performance than DETECT. Nevertheless, DETECT Phase I still outperforms baseline approaches, since it learns an expressive representation of the mobility semantics in the trajectories to reconstruct the augmented trajectories. However, baseline approaches rely on alignments between augmented trajectories and barely learn the dynamics in the data.

5.6 Ablation study

Below we discuss the effects of different parameter settings of DETECT on the reconstruction and clustering performance.

Effect of the latent embedding dimension. Table 3 demonstrates the effect of latent embedding dimension d on the reconstruction error in Phase I. We can observe either too small or too large dimensions have larger errors. With a



(a) Rand Index

(b) Mutual Information

(c) Purity

(d) Fowlkes Mallows Index

Figure 10: Comparison of different clustering approaches. DETECT achieves the best performance with all four metrics for both datasets.

Table 3: Mean and standard deviation of the reconstruction mean absolute error (MAE) of DETECT Phase I with different latent embedding dimension d

d	16	32	64	128
mean ($\times 10^{-3}$)	5.6	4.9	4.3	4.5
std ($\times 10^{-3}$)	0.61	0.2	0.06	0.23

Table 4: Mean and standard deviation of the reconstruction mean absolute error (MAE) of DETECT phase I with different epochs

epoch	100	500	1000	1500	2000
mean ($\times 10^{-3}$)	5.1	4.5	3.9	3.7	3.9
std ($\times 10^{-3}$)	0.17	0.23	0.06	0.06	0.38

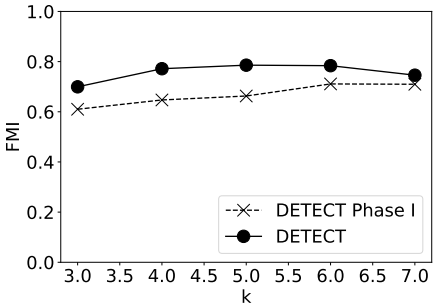
small number of hidden dimensions, the model is incapable to learn the representation. While if the number of dimensions grows too large, the model could easily overfit the training data and does not generalize well to the testing data.

Effect of the training epochs. Table 4 demonstrates the effect of training epoch. As depicted in the table, the standard deviation suddenly rises when epoch = 2000. This is because the model tends to overfit as epoch becomes very large.

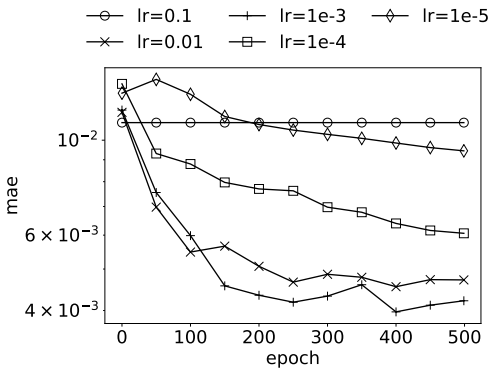
Effect of the number of clusters. We illustrate the effect of k on DETECT approaches in Figure 11a. The experiment is conducted on the GeoLife dataset, thus the number of ground truth classes is six. Consequently, we observe that both DETECT Phase I and DETECT reach their best FMI at around six. That conforms to the intuition and suggests that the DETECT model would perform better as the number of clusters approaches the number of ground truth classes.

Effect of the learning rate. In Figure 11b, we compare the learning curve with different learning rates. As the figure depicts, very large learning rate, i.e., $lr = 0.1$ the model would barely learn anything, which results in a large reconstruction error. In contrast, with learning rate that is too small, e.g., $lr = 1e-5$, the model will take too long to converge.

Effect of noisy input. To study the effect of noisy input, we corrupt the augmented trajectories with some Gaussian noises and train the model to learn to reconstruct the uncorrupted input. We generate the noises for different geographical columns separately with the corresponding variances, since different geographical features have different sensitivity to noises. Figure 12 shows the effects of adding different percentages of noises to the augmented trajectories.



(a) Effect of the number of clusters k



(b) Effect of the learning rate lr

Figure 11: Effect of DETECT parameters

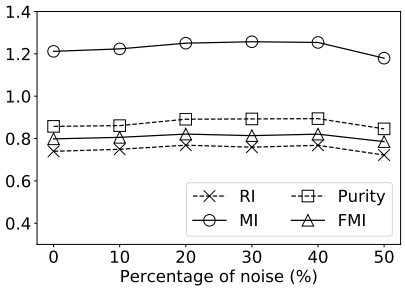
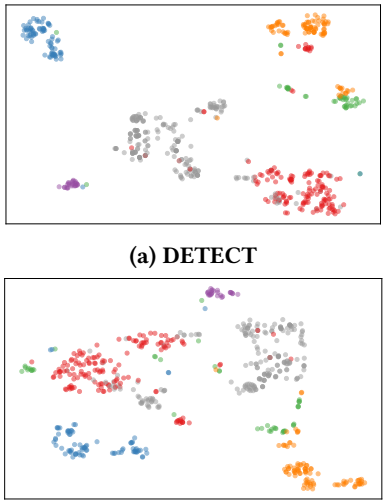


Figure 12: Effect of noisy input.

The clustering performance first increases slightly and then decreases. This is because adding a small amount of noise will encourage the model to learn a representation that is robust to noise. However, too much noise would prevent the model form learning useful representation.

5.7 Results visualization

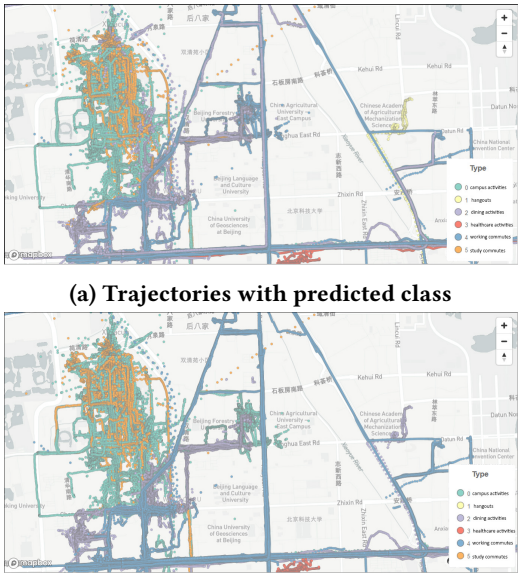
To further understand the learned representation of DETECT, we conduct a series of visualizations. Figure 13 shows the two-dimensional t-SNE [18] plot of DETECT embeddings in



(a) DETECT

(b) DETECT Phase I Only

Figure 13: Clustering results visualization of "DETECT" and "DETECT Phase I only" using t-SNE with perplexity 40



(a) Trajectories with predicted class

(b) Trajectories with ground truth class

Figure 14: Projecting raw trajectories clusters in the map. (a) Colored with predicted cluster labels (b) Colored with ground truth classes

the GeoLife dataset. Each point is colored using the corresponding ground truth class. We observe that 1) DETECT manages to generate well formed clusters (Figure 13a) with most points in the same class grouped into the same cluster and well separated from others, and 2) the clusters in Figure 13a is cleaner than the ones in Figure 13b, which justifies the effectiveness of the Phase II, which jointly optimizes the embedding and the clustering assignment.

Figure 14 visualizes the labeled trajectories in the GeoLife dataset on the real-world map colored with the predicted classes (Figure 14a) and the ground truth classes (Figure 14b). We aligned the predicted classes with the ground truth classes to make consistent coloring between these figures. We observe that predicted classes well match the ground truth, for trajectories with various shapes and lengths. This shows the effectiveness of the proposed model and justifies its capability of clustering trajectories in different spatial and temporal scales.

6 RELATED WORK

There are many studies in trajectory clustering. The previous work either use raw trajectory data (e.g., [27]) with a variety of distance/similarity measurements such as the classic Euclidean, Hausdorff, LCSS, DTW, Frechet or ad-hoc measurements for specific applications (e.g., [7, 15]). Moreover, since trajectories could have various lengths, some existing techniques focus on first finding local patterns in sub-trajectories. For example, Lee et al. [12] proposed to first partition trajectories into minimum description length (MDL) and then clusters the partitioned trajectories. These methods based on raw trajectory data are sensitive to the change in spatiotemporal scales [28].

Since raw spatial and temporal coordinates do not provide much semantic about a trajectory, other techniques use movement characteristics derived from the raw data for clustering. These movement characteristics include speed, acceleration, rotation, and stops, etc. In recent work [26], the speed, acceleration, and change of "rate of turn" (ROT) of each point in a trajectory are extracted as the input sequence for clustering. In another work, Wang et al. [24] annotate trajectories with movement labels such as "Enter", "Leave", "Stop", and "Move" and use these labels to detect events from trajectories. Similar to the approaches based on raw trajectory data, these methods do not work well for all-scale trajectories and require a careful design of the extraction of movement characteristics. Moreover, they barely capture any mobility behavior. A recent work in the domain of privacy also concerned about the semantics of users' traces [3], but their goal is to synthesize privacy-preserving traces according to the semantics which is not relevant to our problem.

Representation learning helps to overcome the need for manually curated trajectory features. Several existing works [9, 10, 25] proposed approaches on training neural networks for unsupervised learning. However, their inputs are mainly image and text, which are not applicable to our problem on clustering trajectories to mobility behaviors.

7 CONCLUSION

We proposed a novel deep learning framework, dubbed DETECT, for identifying mobility behaviors in trajectories. Specifically, we first built an efficient stay point extraction module and a geographical augmentation layer, which extract human mobility context from various scales of trajectories. Our experiments confirmed that such augmentation is effective. Subsequently, we further jointly learned a latent fixed-length trajectory representation as well as the clustering assignment in a fully unsupervised manner. When evaluated on two real-world trajectory datasets, DETECT achieved clearly better performance than the state-of-the-art baselines in all evaluated metrics.

The proposed framework is flexible to incorporate additional features. For future work, we intend to integrate more augmented features into the trajectories, such as temporal frequency, events and holidays. Besides, different types of clustering algorithms can be used in the initialization of DETECT Phase II.

REFERENCES

[1] Martín Abadi et al. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*.

[2] Pierre Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*. 37–49.

[3] Vincent Bindschaedler and Reza Shokri. 2016. Synthesizing plausible privacy-preserving location traces. In *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE, 546–563.

[4] State Information Center. 2017. Map POI (Point of Interest) data. http://dx.doi.org/10.18170/DVN/WSXCNM Peking University Open Research Data Platform.

[5] François Chollet et al. 2015. Keras. https://github.com/fchollet/keras.

[6] Didi Chuxing. 2018. DiDi Chuxing GAIA Open Data Initiative. https://gaia.didichuxing.com

[7] Nivan Ferreira, James T Klosowski, Carlos E Scheidegger, and Cláudio T Silva. 2013. Vector Field k-Means: Clustering Trajectories by Fitting Multiple Vector Fields. In *Computer Graphics Forum*, Vol. 32. Wiley Online Library, 201–210.

[8] OpenStreetMap Foundation. 2018. OpenStreetMap data. http://download.geofabrik.de/north-america.html

[9] Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. 2017. Improved deep embedded clustering with local structure preservation. In *International Joint Conference on Artificial Intelligence (IJCAI)*. 1753–1759.

[10] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. 2017. Deep clustering with convolutional autoencoders. In *International Conference on Neural Information Processing*. Springer, 373–382.

[11] Jing He, Xin Li, Lejian Liao, Dandan Song, and William K Cheung. 2016. Inferring a Personalized Next Point-of-Interest Recommendation Model with Latent Behavior Patterns.. In *AAAI*. 137–143.

[12] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*. 593–604.

[13] Quannan Li, Yu Zheng, Xing Xie, Yukun Chen, Wenyu Liu, and Wei-Ying Ma. 2008. Mining user similarity based on location history. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*. ACM, 34.

[14] T Warren Liao. 2005. Clustering of time series data – a survey. *Pattern recognition* 38, 11 (2005), 1857–1874.

[15] Bin Lin and Jianwen Su. 2008. One way distance: For shape based similarity search of moving object trajectories. *GeoInformatica* 12, 2 (2008), 117–142.

[16] Yijun Lin, Yao-Yi Chiang, Fan Pan, Dimitrios Stripelis, José Luis Ambite, Sandrah P Eckel, and Rima Habre. 2017. Mining public datasets for modeling intra-city PM2. 5 concentrations at a fine spatial resolution. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 25.

[17] Zachary C Lipton, John Berkowitz, and Charles Elkan. 2015. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019* (2015).

[18] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[19] Mohamed F Mokbel. 2010. Study and Implementation of Moving Object Data Generators for Road Networks. (2010).

[20] DMCL University of Illinois at Chicago. 2006. Real Trajectory Data. https://www.cs.uic.edu/~boxu/mp2p/gps_data.html

[21] John Paparrizos and Luis Gravano. 2015. k-shape: Efficient and accurate clustering of time series. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 1855–1870.

[22] François Petitjean, Alain Ketterlin, and Pierre Gançarski. 2011. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition* 44, 3 (2011), 678–693.

[23] Romain Tavenard. 2017. tslearn: A machine learning toolkit dedicated to time-series data. https://github.com/rtavenar/tslearn.

[24] Xiaofeng Wang, Gang Li, Guang Jiang, and Zhongzhi Shi. 2013. Semantic trajectory-based event detection and event pattern mining. *Knowledge and information systems* 37, 2 (2013), 305–329.

[25] Junyuan Xie, Ross Girshick, and Ali Farhadi. 2016. Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*. 478–487.

[26] Di Yao, Chao Zhang, Zhihua Zhu, Jianhui Huang, and Jingping Bi. 2017. Trajectory clustering via deep representation learning. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 3880–3887.

[27] Hyunjin Yoon and Cyrus Shahabi. 2008. Robust time-referenced segmentation of moving object trajectories. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE, 1121–1126.

[28] Guan Yuan, Penghui Sun, Jie Zhao, Daxing Li, and Canwei Wang. 2017. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review* 47, 1 (2017), 123–144.

[29] Yu Zheng. 2015. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)* 6, 3 (2015), 29.

[30] Yu Zheng, Quannan Li, Yukun Chen, Xing Xie, and Wei-Ying Ma. 2008. Understanding mobility based on GPS data. In *Proceedings of the 10th international conference on Ubiquitous computing*. ACM, 312–321.

[31] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.

[32] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th international conference on World wide web*. ACM, 791–800.

[33] K. Zhu, L. Zhang, and A. Pattavina. 2017. Learning Geographical and Mobility Factors for Mobile Application Recommendation. *IEEE Intelligent Systems* 32, 3 (May 2017), 36–44.