时空数据的表示学习

（申请清华大学工学博士学位论文）

培 养 单 位 ： 交叉信息研究院

学　　科 ： 计算机科学与技术

研　究　生 ： 曹 玮

指 导 教 师 ： 李 建 副 教 授

二〇一八年六月

On Learning Representations of Spatio-temporal Data

Dissertation Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Doctor of Philosophy

in

Computer Science and Technology

by

Wei Cao

Dissertation Supervisor : Associate Professor Jian Li

June, 2018

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容。

本人保证遵守上述规定。

（保密的论文在解密后应遵守此规定）

作者签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

摘 要

在现代社会中，基于位置的服务系统(LBS)为人们的日常生活带来了极大的便利。与此同时，LBS服务每天都会源源不断的产生出大量的，杂乱的数据。这些数据中即包含了时间信息，同时也包含了空间信息。因此，我们将这类由LBS服务产生的数据叫做**时空数据**（spatio-temporal data）。时空数据的挖掘为许多工业和商业应用带来了巨大的潜力，例如交通流分析[1–3]，旅游路径推荐[4–7]，以及基于地理位置的社交网络[8–11]。为了在时空数据的挖掘中得到有意义并且准确的结果，一个关键的步骤是从原始时空数据中抽取出有效**特征表示**（feature representations）。

在这篇论文中，我们研究如何从时空数据中学习出它的有效特征表示。由于时空数据固有的特征，时空数据的表示学习面临许多挑战。首先，时空数据中往往同时包含非常复杂的空间关联和时间依赖。同时，时空数据还受到诸多外部因素的影响，例如天气状况以及节假日等。一个有效的特征表示应该能够获取到数据中的这些关联关系。另外，在许多工业应用场景中，时空数据非常杂乱，伴随大量噪音，且数据规模巨大。这给时空数据的表示学习带来了非常大的困难。我们通过研究三个具体的问题，展示了如何解决上述挑战并有效学习到数据特征表示。

在第三章，我们考虑如何在数据的表征中捕获复杂的时空依赖性。我们以交通到达时间预估问题为例。在这个问题中，我们研究如何对任意给定的路线（通常表示为一个连通的路径序列），估计通过这条路径的驾驶时间。我们基于深度神经网络提出一个端到端的框架，叫做DeepTTE。在DeepTTE中，我们提出了一个基于地理卷积操作（geo-convolution）来捕获不同路径之间的空间依赖性。我们在地理卷积层上叠加了一个递归层（recurrent layer）来进一步捕捉不同路段之间的时空依赖性。上述结构可以直接从原始数据中学习到有效的特征表示。在模型最顶端，我们利用一个多任务学习模块（multi-task learning），基于之前得到的特征表示预测道路通达时间。

在第四章，我们研究当存在数据缺失时，如何进行时空数据的表示学

习。我们扩展了第三章中的结果，并提出一种全新的数据缺失补全的方法叫做BRITS（Bi-directional Recurrent Imputation Time Series）。我们的模型不对数据的生成过程做任何先验假设。我们利用双向的递归神经网络捕获数据中的依赖关系，并将学到的依赖关系表示为隐空间中的表示向量。基于表示向量，我们对缺失数据进行直接预测。在我们模型中，我们将预测得到的缺失数据作为RNN计算图的一部分。因此，补全误差能够完整回传以更新模型，进而使得补全结果更为准确。

在第五章，我们进一步研究在极端嘈杂的时空数据中如何进行表示学习。我们以多源轨迹数据（heterogeneous mobility dataset）中的用户识别问题为例。在这个问题中，我们重点关注时空数据的杂乱性与大规模性。我们将大规模轨迹数据中的用户识别问题进行的形式化定义。我们提出一个基于MapReduce的框架叫做Automatic User Identification（AUI）。我们的框架可以有效处理海量的轨迹数据。框架基于一个全新的轨迹相似度函数叫做Signal Based Similarity（SIG）。SIG可以有效测量不同采样率的高噪音轨迹数据之间的相似性。在SIG中，我们将用户轨迹进行抽取，转换为“共现”（co-occurrence）事件表示。我们在不同粒度下，分别基于共现事件表示计算轨迹间的相似性并进行汇总从而得到最终的相似性。我们将SIG和其他现有的轨迹相似性函数进行比较。实验结果表明我们的模型具有更强的鲁棒性和准确性。

关键词：时空数据；表示学习；深度神经网络

Abstract

The ubiquity of location-based services greatly benefits people's daily life. In consequence, a large volume of massive data records has been generated routinely in location-based services. Such data records contain both location information and time information, which we denote as *spatio-temporal data*. Mining spatio-temporal data shows great potentials in many industrial and commercial applications, such as traffic analysis[1–3], travel recommendation[4–7] and location-based social network[8–11]. In order to achieve meaningful and accurate mining results, an important step is to extract effective feature representations from the raw data.

In this thesis, we study learning representations of spatio-temporal data. Due to characteristics of spatio-temporal data, learning representations of such data faces many challenges. First, spatio-temporal data usually contains complex spatial and temporal correlations simultaneously, and it is also affected by various external factors such as the weather conditions and holidays. An effective feature representation should be able to capture such correlations in data. Furthermore, in many industrial scenarios, spatio-temporal data is massive, noisy, and in very large scales, which makes it hard to obtain accurate feature representations. We study three concrete problems to illustrate how to learn the spatio-temporal data representations while addressing such challenges.

In Chapter 3, we study capturing complex correlations in data representations. We use travel time estimation problem as an example. In this problem, we estimate the travel time of any given path (denoted by a sequence of connected roads). We propose an end-to-end framework based on deep neural networks called DeepTTE. Specifically, we present a geo-convolution operation for capturing the spatial correlations of different roads. We stack a recurrent layer on the geo-convolution layer to further capture the temporal dependencies of these roads. Such architecture enables us to extract effective feature representations from raw data. A multi-task learning component is given on the top of DeepTTE, that learns the travel time based on the extracted features.

In Chapter 4, we study handing missing values while learning spatio-temporal data representations. We extend the result in Chapter 3 and propose BRITS (Bi-directional Recurrent Imputation Time Series), a novel method for imputing missing values in spatio-temporal data. Our method does not impose any assumption about the data-generating process. Alternatively, we use a bi-directional recurrent neural network (RNN) to learn correlations in data and transform the learned correlations into several hidden space representations. We predict the missing values directly based on the hidden representations. Our model treats the missing values as variables of RNN graph. Thus, the imputation errors can be fully backpropagated which makes the imputation more accurate.

In Chapter 5, we further study learning representations for extremely massive spatio-temporal data. We use the user identification problem as an example, where we are supposed to identify users across heterogeneous mobility data sources. In this problem, we focus on the massiveness and scales of spatio-temporal data. We formulate the user identification problem over large-scale mobility datasets. We present a MapReduce-based framework called Automatic User Identification (AUI) which can scale to very large datasets. Our framework is based on a novel similarity measure called the *signal based similarity* (SIG) which measures the similarity of users' trajectories gathered from different data sources, typically with very different sampling rates and noise patterns. In signal based similarity, we transform users' trajectories to a series of "co-occurrence" events representations. We calculate the similarity scores based on such representations in multiple resolutions. We show that comparing with other existing similarity measures, SIG is much more accurate and robust for very massive mobility datasets.

Key words: Spatio-temporal data; Representation learning; Deep neural networks.

目 录

第 1 章 Introduction

1.1 Background

In recent years, the advances in communication technologies and the ubiquity of location-based services (LBS) greatly benefit people's daily life. For example, people use their GPS embedded devices to search the destinations, plan the routes or share their mobilities; institutes use the loop sensors to monitor the traffic or schedule the vehicles. In these scenarios, a large volume of massive data records are generated routinely. The data records contain both the location information and the time information, which we denote as the *spatio-temporal data*. Mining massive spatio-temporal data shows great potentials in various industrial and commercial applications, such as traffic analysis[1–3], travel recommendation[4–7], and location-based social network[8–13].

A typical way in mining spatio-temporal data contains two phases. In the first phase, we extract useful information from the raw data and represent such information as a fixed length feature vector. In the second phase, based on the extracted features, we train a machine learning model (e.g., logistical regression, probabilistic graphical model) to solve the mining task. In order to achieve meaningful and accurate results, it is important to extract effective feature representations of the raw data in the first stage. However, due to characteristics of spatio-temporal data, learning representations of such data is highly non-trivial and faces many challenges:

- **(Complex correlations)** The spatio-temporal data usually contains complex correlations. Due to its inherent characteristics, the spatial and temporal correlations both exist in the data simultaneously. For example, in a traffic system, the traffic condition in one road is correlated with its historical conditions, as well as its neighboring roads. Furthermore, the spatio-temporal patterns are also affected by various of external factors, such as weather conditions and holidays. Effective feature representations should be able to capture such complex correlations

in data.

- **(Sensitive to granularities)** The spatio-temporal data is sensitive to the granularities. The information loss caused by the coarse granularities can cause damages to the model performance. Comparing with vision or natural language tasks, for spatio-temporal data, it is important to retain the fine granularity information in the data when learning its representations. As an example, in image classifications, if we replace the original images with relatively coarse images, we can still identify the image classes (e.g., dogs, cats) with high probability. However, in the urban traffic, if we mix up two road segments, even if the road segments are very close, their properties (e.g., congestion level, traffic flow) can be extremely different.
- **(Missing values)** The spatio-temporal data is usually very noisy and irregular. Due to the communication error or unexpected device error, it is common that the data contains a large number of missing values. Existing methods often use interpolation or smoothing methods to impute the missing values[14–17]. However, since the correlations in spatio-temporal data are much more complex, simply adopting the smoothing values can lead to very inaccurate results[18].
- **(Massiveness)** In many industrial applications, the spatio-temporal data is very massive and in very large scales. For example, for Google Map Services, there are as many as one billion monthly active users and the patterns of different users in using the services can be very different. We should also consider the robustness of feature representations for some extreme cases and the efficiency for large-scale datasets.

In this thesis, we study learning representations of spatio-temporal data while addressing the above-mentioned challenges. Specifically, we introduce three concrete problems to illustrate how to handle such challenges in learning the spatio-temporal data representations. We present the descriptions of the problems and summarize our contributions in the following sections.

1.2 Capturing Complex Correlations

In Chapter 3, we study capturing complex correlations in data representations. The granularity sensitive issue is also considered in this chapter. Specifically, we use the travel time estimation problem as an example. Given a specific path, denoted by a sequence of connected road segments, the corresponding start time, and external factors such as weather conditions and driving habits, our objective is to estimate the time of traveling through this path. In this problem, the travel time is affected by diverse complex factors, including the spatial correlations, the temporal dependencies, and the external factors. Prior work[19–22] usually focuses on estimating the travel times of individual road segments or sub-paths and then summing up these times. Such methods did not consider the correlations between different road segments, which lead to inaccurate estimations.

Contributions: We propose an end-to-end *Deep* learning framework for *Travel Time Estimation* (called *DeepTTE*), that estimates the travel time of the whole path directly. We present a geo-convolution operation for capturing the spatial correlations of road segments in a fine granularity, by integrating the geographic information into the classical convolution. We stack a recurrent layer on the geo-convolution layer to further capture the temporal dependencies. With such architecture, we can directly learn effective feature representations from raw data. A multi-task learning component is given on the top of DeepTTE, that learns the travel time of both the entire path and each local path at the same time. Extensive experiments on two trajectory datasets show that our DeepTTE significantly outperforms the state-of-the-art methods.

1.3 Imputing Missing Values

In Chapter 4, we study handling the missing values while learning representations of spatio-temporal data. Given multiple correlated time series data, our objective is to predict their class labels and to fill in missing values at the same time. Existing imputation methods often impose strong assumptions of the underlying data generating process, such as linear dynamics in the state space[14,16,17,23]. However, since the spatio-

temporal data usually contains complex correlations, such assumptions may not hold.

Contributions: In this chapter, we propose BRITS (Bi-directional Recurrent Imputation Time Series), a novel method for imputing missing values. Our proposed method does not impose any assumption about the data-generating process. Instead, we use a bidirectional recurrent neural network to capture both the spatial and temporal correlations in the data and represents the captured correlations as several hidden states. We then predict missing values based on the obtained hidden states. We evaluate our model on one synthetic dataset and two real-world datasets. Experiments show that our model outperforms the state-of-the-art methods in both imputation and classification/regression accuracies.

1.4 Handling Massiveness

In Chapter 5, we further study the representation learning for extremely massive spatio-temporal data. We consider the user identification problem as an example. In this problem, we study identifying users across massive heterogeneous mobility data sources. Formally, given two large-scale mobility datasets which are generated from different data sources (e.g., geo-tagged check-in data, navigation points), our goal is to find the trajectory pairs across different datasets which are essentially generated by the same user. We face extremely complicated scenarios in finding such trajectory pairs. For example, (a) the sampling rates in different data sources can be very different; (b) for some sparse trajectories, it is hard to infer any movement of the users (c) for the users who work/live together, their mobilities have a significant overlap and it is easy to misidentify such users. There are many other cases or combinations of those cases, that are impossible to list exhaustively.

Contributions: We present a MapReduce-based framework called *Automatic User Identification* (AUI) which is easy to deploy and can scale to very large data sets. AUI is based on a novel similarity measure called the *signal based similarity* (SIG), which measures the similarity of users' trajectories, typically with very different sampling rates and noise patterns. In signal based similarity, we represent users' trajectories as a series of co-occurrence events and calculate the similarity scores according to the

events in multiple resolutions. Such similarity measure is very robust for the massive mobility datasets. We conduct extensive experimental evaluations, which show that our framework outperforms the existing methods significantly.

1.5 Organization

The rest parts of this thesis are organized as follows: In Chapter 2, we introduce some preliminaries on deep neural networks and map-reduce frameworks, where we use such techniques in learning spatio-temporal data representation. In Chapter 3, we propose our method DeepTTE on capturing data correlations in travel time estimation problem. This chapter is based on our prior work[24]. In Chapter 4, we propose our model BRITS to solve the missing value problem while learning spatio-temporal data representations. The chapter is based on our prior work[25]. In Chapter 5, we further present the representation learning of extremely massive spatio-temporal data. We present our framework AUI[26] to efficiently solve the user identification problem in very massive datasets. This chapter is based on our prior work[26]. Finally, we conclude this thesis and present several future directions in Chapter 6.

第 2 章 Preliminary

2.1 Deep Neural Networks

Deep neural networks are a family of machine learning algorithms based on the representation learning. It has been successfully applied in various of applications such as the computer vision, natural language processing and speech recognition, and achieves many break-through results[27,28].

A regular deep neural network consists of an input layer, an output layer, and multiple hidden layers. Each hidden layer can be regarded as a non-linear transformation of previous outputs. The output layer then predicts the classification/regression labels based on the obtained non-linear features. The stacked non-linear layers greatly enlarge the capacity of deep neural networks and make it possible to automatically discover high-order hierarchical features from raw data. To improve the efficiency and effectiveness of regular deep neural networks, multiple variations of deep architectures are further proposed. In this section, we introduce two commonly used deep architectures called convolutional neural networks and recurrent neural networks.

2.1.1 Convolutional Neural Networks 2D

The two-dimensional convolutional neural networks (2D-CNN) make explicit assumptions that the inputs are images. A typical 2D-CNN is comprised of multiple convolutional layers which take advantages of 2D input structures. Specifically, each convolutional layer contains several kernel filters. Each kernel filter is connected to a small region of previous layer's output and is replicated across the entire image. The replicated filters share the same weight parameters. Comparing with the regular deep neural networks, the convolutional neural network dramatically reduces the number of parameters and thus leads to a much more efficient learning procedure. See Fig. 2.1 for an illustration. The motivation of such design is based on the spatially local cor-

relation property of image data, i.e., each pixel is only correlated with the pixels in a local receptive field. Formally, for an input image $\mathbf{x}$, we denote the weight matrix of a kernel filter as $\mathbf{W}$ and the bias as b. Then, we have that the output h of such filter is

$$h_{ij} = \text{ReLU}((\mathbf{W} * \mathbf{x})_{ij} + b)$$

where $\text{ReLU}(x) = \max(0, x)$ and $*$ is the convolutional operator.

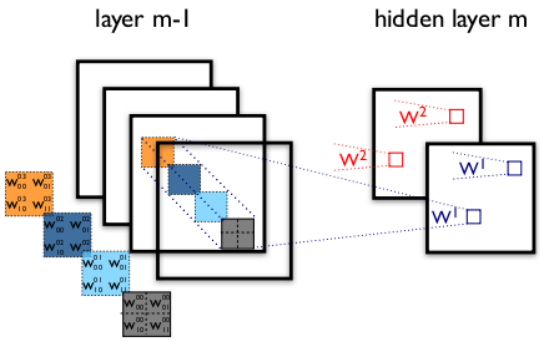


Figure 2.1 Example of 2D-CNN (figure from [29])

2.1.2 Convolutional Neural Networks 1D

Similar to the 2D-CNN, the one-dimensional convolutional neural networks (1D-CNN) use kernel filters to capture the local correlations. However, the input here is a one-dimensional sequential data, e.g., the sentences or audio signals. As an example, for the natural language, each kernel filter in 1D-CNN is applied on a small window of words, as shown in Fig. 2.2. In such case, 1D-CNN captures the correlations between adjacent words.

2.1.3 Recurrent Neural Networks

Recurrent neural networks (RNN) are also widely used to learn the temporal dependencies in the sequential data. Comparing with the feed-forward architectures, it has a "memory" cell which summarizes the information in the previous time steps. At each step, the recurrent neural network updates its memory according to the historical

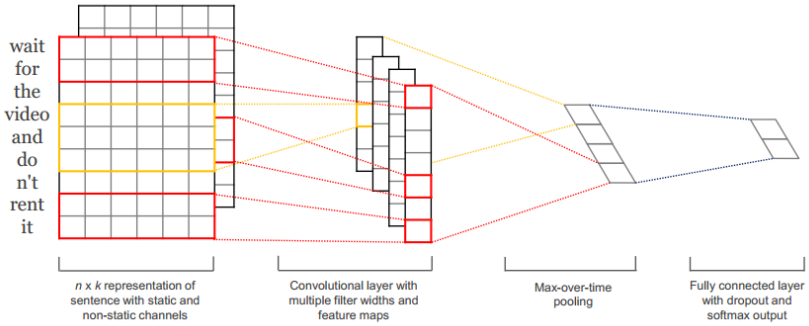


Figure 2.2 Example of 1D-CNN (figure from[30])

memory and the current input recursively. Formally, we use $\mathbf{h}_t$ to denote the memory vector in the t-th step and $\mathbf{x}_t$ to denote the input in this step. Then we have that

$$\mathbf{h}_{t+1} = \text{sigmoid}(\mathbf{W}_x \cdot \mathbf{x}_t + \mathbf{W}_h \cdot \mathbf{h}_t + \mathbf{b})$$

where $\mathbf{W}_x$, $\mathbf{W}_h$ and $\mathbf{b}$ are parameters to be learned. In practice, vanilla recurrent neural networks often face the gradient vanishing/exploding problem[27] which lead to training difficulties. To solve such issue, several improved architectures such as the Long Short Term Memory (LSTM), Gated Recurrent Unit (GRU) are proposed[31,32].

2.2 Map-Reduce Framework

Map-Reduce framework is a programming model proposed by Google for processing large-scale datasets[33]. Such framework provides a flexible way to implement the distributed algorithms and can be easily deployed on a cluster. A map-reduce framework contains two stages: the map stage and the reduce stage. In the map stage, users perform the filtering or other pre-processing operations, and emit a key-value pair $\langle k, v \rangle$. During the reduce stage, the values with the same key will be partitioned into the same reducer and the users implement the summarization operations of these values on the reducer.

To make it more concrete, we illustrate how we count the frequencies of different words in a large corpus with Map-Reduce framework. In the map stage, for each word w, we simply emit the key-value pair $\langle w, 1 \rangle$. In the reduce stage, each reducer receives

a word *w* and a list of 1's. Since the values with the same key are partitioned into the same reducer, we simply count the 1's for each word *w* and output its frequency.

第 3 章　Capturing Data Correlations in Travel Time Estaimation

In this chapter, we study the travel time estimation problem as an example of capturing complex correlations in spatio-temporal data representation. Estimating the travel time of any path (denoted by a sequence of connected road segments) in a city is of great importance to traffic monitoring, route planning, ridesharing, taxi/Uber dispatching, etc. However, it is a very challenging problem, affected by diverse complex factors, including spatial correlations, temporal dependencies, external conditions (e.g. weather, traffic lights). To effectively extract feature representations from input data, such correlations must be carefully considered. Prior work usually focuses on estimating the travel times of individual road segments or sub-paths and then summing up these times[19–22], which leads to an inaccurate estimation because such approaches do not consider road intersections/traffic lights, and local errors may accumulate. To address these issues, we propose an end-to-end *Deep* learning framework for *Travel Time Estimation* (called *DeepTTE*) that estimates the travel time of the whole path directly.

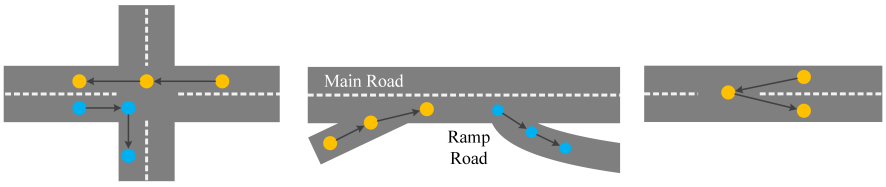
3.1 Introduction

Estimating the travel time for a given path, which is denoted by a sequence of connected sub-paths, is a fundamental problem in route planning, navigation, and traffic dispatching. When users are searching for candidate routes, accurate travel time estimations help them better planning routes and avoiding congested roads, which in turn helps to alleviate traffic congestion.

Although the problem has been widely studied in the past, providing an accurate travel time is still very challenging, affected by the following aspects:
1) *Individual vs. Collective*: There mainly exist two approaches to estimate the travel time of a path: a) *Individual TTE* that firstly splits a path into several road segments (or local paths), and then estimates the travel time for each local path, finally sums

over them to get the total travel time. b) *Collective TTE* that directly estimates the travel time of the entire path. Although individual TTE methods[19–22] can estimate accurate travel time for each road segment, it cannot model complex traffic conditions within the entire path, including road intersections, traffic lights, and direction turns. Besides, local errors may accumulate if there are many road segments in the given path. Collective TTE methods (e.g. Jenelius et al.[34]) are able to capture the aforementioned traffic conditions implicitly. However, as the length of a path increases, the number of trajectories traveling on the path decreases, which reduces the confidence of the travel time (derived from few drivers), pointing out that longer path is harder to estimate. Moreover, in many cases, there is no trajectory passing the entire path.

2) *Diverse complex factors*: the traffic is affected by spatial correlations, temporal dependencies, and external factors. Spatial correlations are various, even complex, as shown in Fig. 3.1. With three consecutive GPS points, it depicts different driving situation, showing the driver may go straight, turn right, turn around, drive into the main road or ramp road. Explicitly extracting these features are time-consuming, even infeasible because the driving situations are more complex in the real-world. We need to consider them implicitly in the method. In addition, these spatial correlations are time-varying. Taking Fig. 3.1 (b) as an example, in the evening rush hour, there are many vehicles the main road, drivers have to drive into the main road from the ramp road one after another slowly, but driving out of the main road may be very quick. But in the non-rush hour, driving into the main road is fast. Furthermore, traffic is affected by many external factors, like weather, driver habit, day of the week.



(a) Go straight (yellow) or turn right (blue)

(b) Drive into main road (yellow) or ramp road (blue)

(c) Turn around

Figure 3.1 Various driving situations

To address the above challenges, in this chapter, we propose an end-to-end frame-

work for *Travel Time Estimation*, called *DeepTTE*. The primary contributions of this chapter can be summarized as follows:

- We propose a spatio-temporal component to learn the spatial and temporal dependencies from the raw GPS sequence. In detail, the spatio-temporal component consists of two parts: a) *a geo-based convolutional layer* that transforms the raw GPS sequence to a series of feature maps, capable of capturing the local spatial correlations (like various driving situations in Fig. 3.1) from consecutive GPS points implicitly; b) *recurrent neural nets* (LSTMs) that learn the temporal dependencies of the obtained feature maps and embeddings from external factors. The spatio-temporal component directly learns effective feature representations of the GPS data.
- We propose a multi-task learning component that learns to estimate the travel time for each local path and the entire path simultaneously by a multi-task loss function, capable of balancing the tradeoff between the individual and collective estimations. For estimating the entire path accurately, we design a multi-factor attention mechanism to learn the weights for different local paths based on their hidden representations and the external factors.
- We present an attribute component that integrates external factors, including the weather condition, day of the week, distance of the path, and the driver habit. The learned latent representations are fed into several parts of the model to enhance the importance of these external factors.
- We conduct extensive experiments on two real-world large scale data sets which consists of GPS points generated by taxis in Chengdu and Beijing. The percentage errors on these two datasets are 11.89% and 10.92% respectively, which significantly outperforms the existing methods.

3.2 Preliminary

In this section, we first present several preliminaries and define our problem formally.

Definition 1 (Historical Trajectory)： We define a *historical trajectory* T as a sequence of consecutive historical GPS points, i.e., $T = \{p_1, \ldots, p_{|T|}\}$[①]. Each GPS point p_i contains: the latitude ($p_i.lat$), longitude ($p_i.lng$) and the timestamp ($p_i.ts$). Furthermore, for each trajectory we record its external factors such as the starting time (timeID), the day of the week (weekID), the weather condition (weather) and corresponding driver (driverID).

Definition 2 (Objective)： During the training phase, we learn how to estimate the travel time of the given path and the corresponding external factors, based on the spatio-temporal patterns extracted from the historical trajectories as we defined in Definition. 1. During the test phase, given the path P, our goal is to estimate the travel time from the source to the destination through P, with the corresponding external factors. We assume that the travel path P is specified by the user or generated by the route planing apps.

During the test phase, to make the testing data consistent with the training data, we convert a path P to a sequence of location points with equal distance gaps. Each location is represented as a pair of longitude and latitude.

Remark: In our experiment, to generate the test data, we remove the timestamps in the historical trajectories and resample each trajectory into a GPS location sequence with equal distance gaps. In this chapter, we do not consider how to optimize the path P.

3.3 Model Architecture

In this section, we describe the architecture of our proposed DeepTTE, as shown in Fig. 3.2. DeepTTE is comprised of three components: an attribute component, a spatio-temporal learning component, and a multi-task learning component. The attribute component is used to processes the external factors (e.g. weather) and the basic information of the given path (e.g. start time). Its output is fed to the other components

① The GPS devices usually generate one record for every fixed length time gap. This can cause our model to learn a trival pattern (e.g., simply counts the number of GPS records). To avoid such case, we resample each historical trajectory such that the distance gap between two consecutive points are around 200 to 400 meters.

as a part of their inputs. The spatio-temporal learning component is the main building component that learns the spatial correlations and temporal dependencies from the raw GPS location sequences. Finally, the multi-task learning component estimates the travel time of the given path based on the previous two components, capable of balancing the tradeoff between individual estimation and collective estimation.

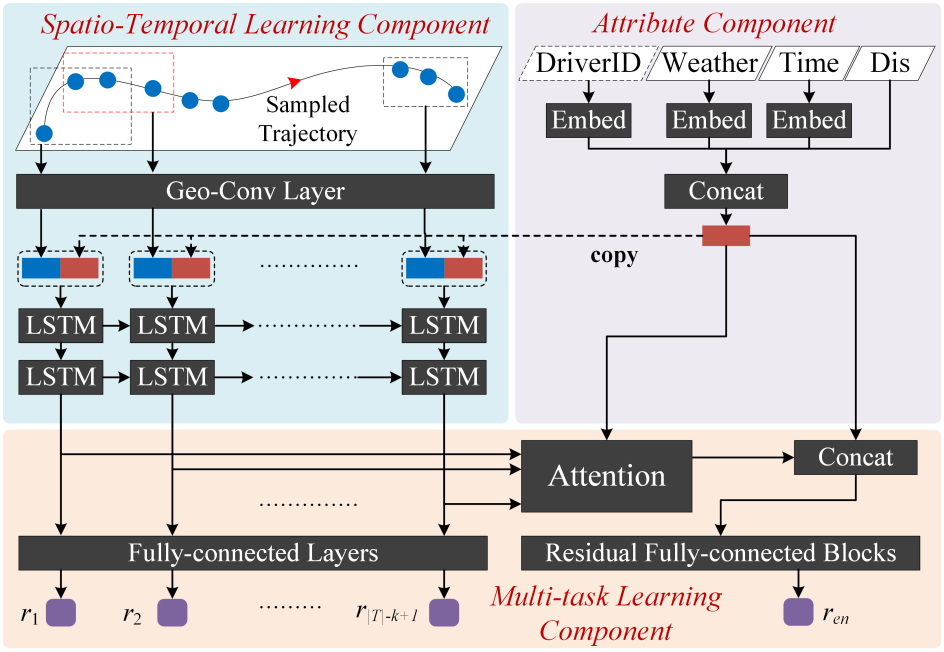


Figure 3.2 DeepTTE Architecture. Dis: distance; concat: concatenate.

3.3.1 Attribute Component

As we mentioned, the travel time of a path is affected by many complex factors, such as the start time, the day of week, the weather condition and also the driving habits. We design a simple yet effective component to incorporate such factors into our model, where we call it the attribute component. We can also easily incorporate more factors into the model.

As an example in Fig. 3.2, we incorporate the attributes of the weather condition (rainy/sunny/windy etc.) , the driver ID, the time information (day of the week and

timeslot of travel start①). We use weatherID, driverID, weekID and timeID to denote these attributes respectively. Note that these factors are categorical values which cannot be fed to the neural network directly. In our model, we use the embedding method[35] to transform each categorical attribute into a low-dimensional real vector. Specifically, the embedding method maps each categorical value $v \in [V]$ to a real space $\mathbb{R}^{E\times 1}$ (we refer to such space as the embedding space) by multiplying a parameter matrix $W \in \mathbb{R}^{V\times E}$. Here V represents the vocabulary size of the original categorical value and E represents the dimension of embedding space. Usually, we have that $E \ll V$. Comparing with the one-hot encoding[35], the embedding method mainly has two advantages. First, since the vocabulary size of the categorical values can be very large (e.g., there are 14863 drivers in our dataset), the embedding method effectively reduces the input dimension and thus it is more computationally efficient. Furthermore, it has been shown that the categorical values with similar semantic meaning are usually embedded to the close positions[35]. Thus, the embedding method helps find and share similar patterns among different trajectories.

Besides the embedded attributes, we further incorporate another important attribute, the *travel distance*. Formally, we use $\Delta d_{p_a\to p_b}$ to denote the total distance of traveling from GPS point p_a to p_b along the path, i.e., $\Delta d_{p_a\to p_b} = \sum_{i=a}^{b-1} \mathsf{Dis}(p_i, p_{i+1})$ where Dis is the geographic distance between two GPS points. Then, we concatenate the obtained embedded vectors together with the travel distance $\Delta d_{p_1\to p_{|T|}}$. The concatenation is used as the output of the attribute component. We denote such output vector as *attr*.

3.3.2 Spatio-Temporal Component

In this section, we propose the spatio-temporal component. The spatio-temporal component consists of two parts. The first part is a geo-convolutional neural network which transforms the raw GPS sequence to a series of feature maps. Such component captures the local spatial correlation between consecutive GPS points. The second part is the recurrent neural network which learns the temporal correlations of the obtained

① We divide one day into 1440 timeslots. Each timeslot corresponds to one minute.

feature maps.

3.3.2.1 Geo-Conv Layer

We first present the Geo-Conv layer. Recall that a historical trajectory T is a sequence of GPS location points $\{p_1, \ldots, p_{|T|}\}$ where each p_i contains the corresponding longitude/latitude(See Definition 1). As we mentioned in the introduction part, capturing the spatial dependencies in the GPS sequence is critical to travel time estimation. A standard technique to capture the spatial dependencies is the *convolutional neural network* (CNN), which is widely used in the image classification, object tracking and video processing etc[36,37]. A typical convolutional layer consists of several convolutional filters. For a multi-channel input image, a filter learns the spatial dependencies in the input by applying the convolution operation on each of the local patches. We refer to such convolutional layer as 2D-CNN. Zhang et al.[38] used the 2D-CNN to predict the citywide crowd flow. In their work, they first partitioned the city into a $I \times J$ grid and then mapped each GPS coordinate into a grid cell. However, in our case, directly mapping the GPS coordinates into grid cells is not accurate enough to represent the original spatial information in the data. For example, we can not distinguish the turnings if the related locations are mapped into the same cell. Thus, our task requires a much finer granularity. Inspired by this, we proposed a *Geo-Conv layer* which is able to capture the spatial dependency in the geo-location sequence while retains the information in a fine granularity.

The architecture of Geo-Conv layer is shown in Fig. 3.3. For each GPS point p_i in the sequence, we first use a non-linear mapping

$$loc_i = \tanh(W_{loc} \cdot [p_i.lat \circ p_i.lng]) \tag{3-1}$$

to map the i-th GPS coordinates into vector $loc_i \in \mathbf{R}^{16}$, where $\circ$ indicates the concatenate operation. Thus, the output sequence $loc \in \mathbf{R}^{16\times|T|}$ represents the non-linearly mapped locations. Note that such sequence can be seen as a 16-channel input. Each channel describes the geographical features of the original GPS sequence. We introduce a convolutional filter, with parameter matrix $W_{conv} \in \mathbf{R}^{k\times16}$. It applies the con-

volution operation on the sequence *loc*, along with a one-dimensional sliding window. We use $*$ to denote the convolutional operation. The i-th dimension of its output is denoted as,

$$loc_i^{conv} = \sigma_{cnn}(W_{conv} * loc_{i:i+k-1} + b) \tag{3-2}$$

where b is the bias term, $loc_{i:i+k-1}$ is the subsequence in loc from index i to index $i+k-1$ and σ_{cnn} is the corresponding activation function.

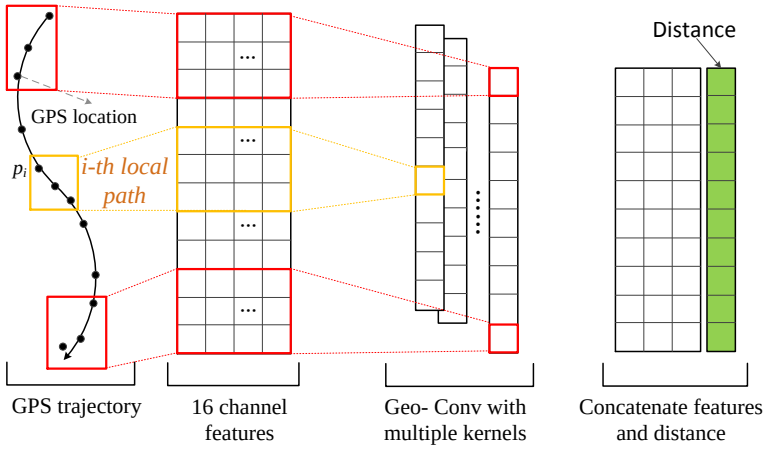


Figure 3.3 Geo-Conv Layer.

Definition 3 (Local Path)： We refer to the sub-sequence from point p_i to point p_{i+k-1} as the *i-th local path*.

Thus, loc_i^{conv} essentially captures the spatial feature of the i-th local path. By concatenating the outputs of c filters, we obtain a feature map of local paths with shape $\mathbf{R}^{c\times(|T|-k+1)}$. In the literature of natural language processing, such architecture is so called 1D-CNN[30].

Nevertheless, in our task, the travel time is highly related with the total distance of the path. It is hard for 1D-CNN to extract the geometric distance directly from the raw longitudes/latitudes. Therefore, in the Geo-Conv layer, we further append a column to the previous obtained feature map. The i-th element of the new appended column (green part in Fig. 3.3) is $\sum_{j=i+1}^{i+k-1} \mathrm{Dis}(p_{j-1}, p_j)$, i.e., the distance of the i-th local path.

Thus, we obtain the final feature map of shape $\mathbf{R}^{(c+1)\times(|T|-k+1)}$ by our Geo-Conv layer. We denote this feature map as loc^f.

3.3.2.2 Recurrent Layer

The feature map loc^f captures the spatial dependencies of all the local paths. To further capture the temporal dependencies among these local paths, we introduce a recurrent layers in our model. The recurrent neural network (RNN) is an artificial neural network which is widely used for capturing the temporal dependency in sequential learning, such as natural language processing and speech recognition[37,39]. The recurrent neural network is able to "memorize" the history in the processed sequence. When processing the current time step in the sequence, it updates its memory according to the current input and the previous memories. The output of the recurrent neural network is the memory sequence at all the time steps in the sequence.

In our model, the input sequence of the recurrent neural network is the feature map loc^f outputted by Geo-Conv layer. The feature map loc^f can be regarded as a sequence of spatial features with length $|T| - k + 1$. Moreover, we find that incorporating the attributes information is helpful to further enhance the capacity of the recurrent layers (recall that we have obtained the attributes representation vector $attr$ in the attribute component). Thus, in simple terms, the updating rule of our recurrent layer can be expressed as

$$h_i = \sigma_{rnn}(W_x \cdot loc_i^f + W_h \cdot h_{i-1} + W_a \cdot attr) \tag{3-3}$$

where h_i is the memory after we processed the i-th local path and σ_{RNN} is a non-linear activation function. In practice, the Eq.(3-3) usually fails in processing the long sequence due to vanishing gradient and exploding gradient problems[31]. To overcome such issue, we use two stacked Long Short-Term Memory (LSTM) layers instead. LSTM was first developed by Hochreiter et al[31]. It introduces an input gate and a forget gate to control the in/out information flow. Such gate mechanism enables LSTM to forget some unimportant information and effectively alleviate the gradient vanishing/exploding problem. Furthermore, it has been shown that a stacked LSTM is more

efficient to increase the model capacity compared with a single layer LSTM[40].

Now, by utilizing the Geo-Conv layer and the recurrent layer, we obtain the sequence $\{h_1, h_2, \ldots, h_{|T|-k+1}\}$ which represents the spatio-temporal features of the raw GPS sequence.

3.3.3 Multi-task Learning Component

We finally introduce a multi-task learning component which combines the previous components and estimates the travel time of input path. Prior work in estimating the travel time can be divided into two types, the *individual estimation* and the *collective estimation*. The individual estimation estimates the travel time of each local path and sum them up. The collective estimation instead estimates the travel time of the entire path directly. However, as we mentioned in the challenge part, the local errors may accumulate if we adopt the individual estimation since such method does not consider the spatio-temporal dependencies among the local paths. In the mean time, if we use the collective estimation, we usually face the data sparsity problem since only a few trajectories traveled through the entire path or the longer sub-paths.

In our model, we combine these two methods with our multi-task learning component. The multi-task learning component estimates the travel time of both entire path and each local path simultaneously. As during the training phase, the travel times of the local paths are available. The local path estimation utilizes this information for stabilizing the training procedure and enhancing the accuracy.

3.3.3.1 Estimate the local paths

Recall that we use the spatio-temporal component to obtain a sequence $\{h_1, h_2, \ldots, h_{|T|-k+1}\}$. Here each h_i corresponds to the spatio-temporal feature of local path $p_i \rightarrow p_{i+1} \rightarrow \ldots \rightarrow p_{i+k-1}$. We simply use two stacked fully-connected layers with size 64 and 1 to map each h_i to a scalar r_i. Here r_i represents the travel time of the i-th local path. The stacked fully-connected layers for different h_i share the same weight parameters, as shown in Fig. 3.2.

3.3.3.2 Estimate the entire path

The estimation of the entire path is relatively more complex. Recall that the length of spatio-temporal feature sequence $\{h_i\}$ is also variable. To estimate the travel time of the entire path directly, we first need to transform the feature sequence into a fixed length vector. A simple method to achieve this is to use *mean pooling*, i.e., $h_{mean} = \frac{1}{|T|-k+1} \sum_{i=1}^{|T|-k+1} h_i$. Such mechanism is simple yet effective. It demonstrates a good performance for most paths as shown in the experiment part. However, the mean pooling method treats all the spatio-temporal features h_i equally. In fact, the uncertainty of accurately estimating the travel time is usually caused by several critical local paths. For example, if the path contains multiple road intersections, traffic lights, or road segments which can be extremely congested, we should pay more attention on such parts since they are more difficult to estimate. Inspired by this, in our model, we adopt the *attention mechanism* instead of the mean pooling. The attention mechanism is essentially the weighted sum of sequence $\{h_i\}$ where the weights are parameters learned by the model. Formally, we have that

$$h_{att} = \sum_{i=1}^{|T|-k+1} \alpha_i \cdot h_i \tag{3-4}$$

where α_i is the weight for the i-th local path, and the summation of all α_i equals 1. To learn the weight parameter α, we consider the spatial information of the local paths, as well as the external factors such as the start time, the day of week and the weather condition. For example, if the starting time of the path is a weekend evening, we would be careful with the road segments near entertainment district since they are usually very congested at that time. In our model, the vector *attr* outputted by the attribute component captures the effect of external factors, and the feature sequence $\{h_i\}$ captures the spatio-temporal information of local paths. Thus, we devise our attention mechanism based on *attr* and $\{h_i\}$:

$$\begin{aligned} z_i &= \langle \sigma_{ATT}(attr), h_i \rangle \\ \alpha_i &= \frac{e^{z_i}}{\sum_j e^{z_j}} \end{aligned} \tag{3-5}$$

where σ_{ATT} is a non-linear mapping which maps *attr* to a vector with the same length as h_i. Substituting Eq. (3-5) into Eq. (3-4), we obtain the vector h_{att}.

Finally, we pass h_{att} to several fully-connected layers with equal size (we use the size of 64 in our experiment). The fully-connected layers are connected with *residual connections* which is a technique to train a very deep neural network[41]. The residual connection adds "shortcuts" between different layers (dashed line in Fig. 3.2). Thus, previous information flow can skip one or more non-linear layers through the shortcut and the skipped layers just need to learn the "residual" of the non-linear mapping. In our model, we use σ_{f_i} to denote the i-th fully-connected layer. For the first fully-connected layer, the output of this layer is $\sigma_{f_1}(h_{attr})$. For the rest of the fully-connected layers, suppose the output of the i-th layer is x. Then, the output of the $(i+1)$-th layer can be represented as $\sigma_{f_{i+1}}(x) \oplus x$ where $\oplus$ is the element-wise add operation. It has been shown that training the neural networks with residual connections is easier and more robust[41]. At last, we use a single neuron to obtain the estimation of the entire path, which we denote as r_{en}.

3.3.4 Model Training

We finally present the training procedure of our model. Our model is trained end to end. During the training phase, we use the *mean absolute percentage error (MAPE)* as our objective function. Since MAPE is the relative error, we can enforce our model to provide accurate results for both the short paths and the long paths. However, we use multiple criterions to evaluate our model, including the rooted mean squared error (RMSE) and the mean absolute error (MAE).

Recall that during the training phase, we estimate the travel time of all the local paths and the entire path simultaneously. For the local paths, we define the corresponding loss as the average of losses in each local path, i.e.,

$$L_{local} = \frac{1}{|T| - k + 1} \sum_{i=1}^{|T|-k+1} \left| \frac{r_i - (p_{i+k-1}.ts - p_i.ts)}{p_{i+k-1}.ts - p_i.ts + \epsilon} \right| \tag{3-6}$$

where ϵ is a small constant to prevent the exploded loss value when the denominator is

close to 0. For the entire path, we define the corresponding loss as

$$L_{en} = |r_{en} - (p_{|T|}.ts - p_1.ts)|/(p_{|T|}.ts - p_1.ts + \epsilon). \tag{3-7}$$

Our model is trained to minimize the weighted combination of two loss terms

$$\beta \cdot L_{local} + (1 - \beta) \cdot L_{en} \tag{3-8}$$

where β is the combination coefficient that linearly balances the tradeoff between L_{local} and L_{en}. By default, during the test phase, we use the travel time estimation of the entire path r_{en} as our final estimation.

3.4 Experiment

In this section, we report our experimental results on two large scale real-world datasets. We first compare our model with several baseline methods, including the state-of-the-art collective estimation method TEMP[42]. We then present the effectiveness of our model by a set of controlled experiments① .

3.4.1 Experiment Setting

3.4.1.1 Data Description

We evaluate our model on two large scale datasets:

- **Chengdu Dataset**: Chengdu Dataset consists of 9, 737, 557 trajectories (1.4 billion GPS records) of 14864 taxis in August 2014 in Chengdu, China. The shortest trajectory contains only 11 GPS records (2km) and the longest trajectory contains 128 GPS records (41km).
- **Beijing Dataset**: Beijing Dataset consists of 3, 149, 023 trajectories (0.45 billion GPS records) of 20442 taxis in April 2015 in Beijing, China. The shortest trajectory contains 15 GPS records (3.5km) and the longest trajectory contains 128 GPS records (50km).

① The code and the sample data can be downloaded at https://github.com/UrbComp/DeepTTE

The trajectories in both datasets are associated with the corresponding weekID, timeID and driverID. For Beijing Dataset, we further collected the corresponding weather conditions (16 types including sunny, rainy, cloudy etc) as well as the road ID of each GPS point.

3.4.1.2 Parameter Setting

The parameters we used in our experiment are described as follows:

- In the attribute component, we embed weekID to $\mathbf{R}^3$, timeID to $\mathbf{R}^{16}$, driverID to $\mathbf{R}^8$ and the corresponding weather type to $\mathbf{R}^3$.
- In the geo-conv layer, we fix the number of filters $c = 32$ and we use ELU function[43] as the activation σ_{cnn} in Eq.(3-2). The ELU is defined as $\mathsf{ELU}(x) = e^x - 1$ for $x \leq 0$ and $\mathsf{ELU}(x) = x$ for $x > 0$. For the kernel size k, we evaluate our model under different values of k.
- In the recurrent layer, we use tanh as the activation σ_{rnn} in Equ. (3-3). We fix the size of the hidden vector h_i as 128.
- In the multi-task learning component, we first use a fully-connected layer with tanh activation as σ_{att} in Eq. (3-5). We use $\mathsf{ReLU}(x) = \max(0, x)$[44] as the activation of residual fully-connected layers. We fix the number of the residual fully-connected layers as 4 and the size of each layer as 128. Furthermore, we evaluate our model for different combination coefficient β in Eq. (3-8) from 0.0 to 0.99.

For each dataset, we use the trajectories generated in the last 7 days as the test set and the rest of trajectories as the training set. We adopt Adam optimization algorithm[45] to train the parameters. The learning rate of Adam is 0.001 and the batch size during training is 400. We train the model for 100 epochs and select the best models by 5-fold cross-validation.

Our model is implemented with PyTorch 2.0, a widely used Deep Learning Python library.

3.4.2 Performance Comparison

To demonstrate the strength of our model. We first compare our model with several baseline methods, including:

- **AVG:** We simply calculate the average speed in the city during a specific time interval (e.g. 13:00-14:00 PM on Monday). We estimate the travel time of given trajectory based on its starting time and the historical average speed.
- **TEMP:** TEMP[42] is the state-of-the-art collective estimation method. It estimates the travel time of the given path based on the "neighbor" trajectories, i.e., the trajectories which have the closed starting and destination as the query path. This work outperforms the most outstanding individual TTE[22] as well as Bing/Baidu Map API in their experiment. However, there are about 10% paths that the original TEMP method can not estimate due to the lack of neighbor trajectories. For those paths, we enlarge the neighborhood dynamically until we can find enough neighbor trajectories (10 trajectories in our experiment). We refer to such implementation as *dynamic TEMP* (D-TEMP).
- **GBDT:** Gradient Boosting Decision Tree (GBDT) is a powerful ensemble method[46] and widely used in practice. In our problem, the input of GBDT is as same as the input of DeepTTE, including all the inputs in the attribute component and the raw GPS sequence. Note that since the length of GPS sequence is variable, GBDT can not handle such sequence directly. Here, we uniformly sample (with replacement) each GPS sequence to a fixed length of 128.
- **MlpTTE:** We use a 5-layer perceptron with ReLU activation to estimate the travel time. The input of MlpTTE is almost the same as GBDT, except that the categorical values are properly embedded to real vectors. The size of hidden layers in MlpTTE is fixed as 128.
- **RnnTTE:** RnnTTE is also a simplified model of DeepTTE. We use a vanilla RNN and mean pooling to process the raw GPS sequence into a 128-dimensional feature vector. We concatenate such feature vector and the output of the attribute component. The concatenation then is passed to the residual fully-connected layers to obtain the estimation.

Table 3.1 Performance Comparison on Chengdu Dataset

	MAPE (%)	RMSE (sec)	MAE (sec)
AVG	28.1	533.57	403.71
D-TEMP	22.82	441.50	323.37
GBDT	19.32 ± 0.04	357.09 ± 2.44	266.15 ± 2.24
MlpTTE	16.90 ± 0.06	379.39 ± 1.94	265.47 ± 1.53
RnnTTE	15.65 ± 0.06	358.74 ± 2.02	246.52 ± 1.65
DeepTTE	**11.89 ± 0.04**	**282.55 ± 1.32**	**186.93 ± 1.01**

Table 3.2 Performance Comparison on Beijing Dataset

	MAPE (%)	RMSE (sec)	MAE (sec)
AVG	24.78	703.17	501.23
D-TEMP	19.63	606.76	402.50
GBDT	19.98 ± 0.02	512.96 ± 3.96	393.98 ± 2.99
MlpTTE	23.73 ± 0.14	701.61 ± 1.82	489.54 ± 1.61
RnnTTE	13.73 ± 0.05	408.33 ± 1.83	275.07 ± 1.48
DeepTTE	**10.92 ± 0.06**	**329.65 ± 2.17**	**218.29 ± 1.63**

For our model DeepTTE, we fix the kernel size as 3 and combination coefficient β as 0.3. We repeat each experiment for 5 times. We report the mean and the standard variation of the different runs. The experiment result is shown in Table 3.1 and 3.2.

As we can see, simply using the average speed leads to a very inaccurate result. In contrast, the collective method D-TEMP and the ensemble method GBDT are much better than AVG. MlpTTE shows a good performance on Chengdu Dataset. However, it does not consider the spatio-temporal dependencies in the data. For the dataset which contains more complex traffic conditions (e.g., Beijing Dataset), we find that MlpTTE tends to overfit the training data and thus leads to a bad result. RnnTTE use the recurrent layers to capture the temporal dependencies in the data. It achieves 15.65% and 13.73% on two datasets which are much better than above mentioned methods. Our model DeepTTE further significantly outperforms RnnTTE. The error rates in two datasets are only 11.89% and 10.92%. We use Paired T-Test[46] to further test the significance of our model. The p-value for all the test pairs are less than 10^{-8} which

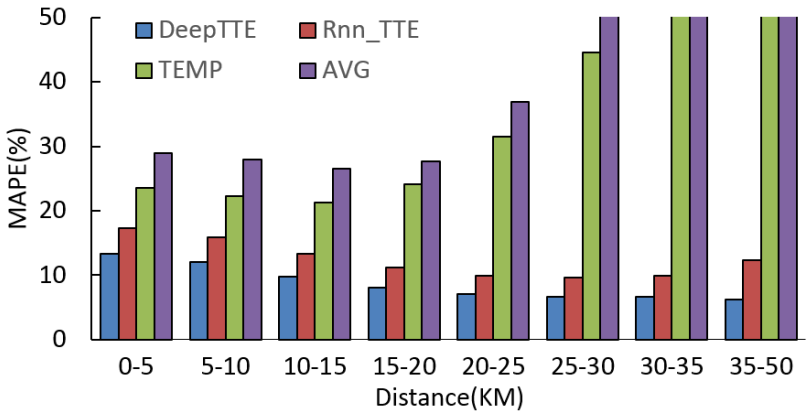


Figure 3.4 Error rates for trajectories with different lengths.

demonstrates that our model is significantly better than baselines. In Fig. 3.4, we compare the model performances for the trajectories with different lengths. When the path length goes larger, AVG and TEMP methods face a serious individual-collective trade-off problem. As a consequence, the error rates increase sharply. RnnTTE shows a better result for the trajectories less than 35 km. However, it fails to handle the long paths (with length greater than 35 km). In contrast, our model demonstrates remarkable results for longer paths. Finally, recall that there are about 10% paths that the original TEMP can not estimate. For the rest of paths in two datasets, the average error rate is 19.96% for the original TEMP but only 11.21% for our model.

3.4.3 Effect of Attribute Component

We show the effectiveness of different attributes, including the weatherID, driverID and weekID. We devise a set of controlled experiments on Beijing Dataset. For each experiment, we eliminate exactly one attribute.We find that weatherID and weekID affect the estimation significantly. Eliminating such two attributes causes an error growth of 1.09% and 0.77% respectively. This also conforms to our intuitive sense, i.e., we usually spend much more time for traveling the same path under bad weather conditions. Eliminating the driver information causes an error increment of 0.30% which seems not significant. However, we stress the trajectories in our dataset are generated by taxi drivers. Most of the taxi drivers are very experienced and have

similar driving habits. The driver information might be more useful for estimating the travel time of normal people. We leave it as an intriguing direction for future work.

3.4.4 Effect of Geo-Conv

We first evaluate our model under the absence of Geo-Conv layer. To achieve this, we eliminate the geo-conv layer and directly pass the sequence of $\{p_i.lat \circ p_i.lng\}$ to the following LSTM layers. The MAPE in Chengdu/Beijing Dataset under this setting is 13.14% and 12.68% (comparing with 11.89% and 10.92%).

We further test our model if we increase the kernel size k in Equ. (3-2) from 3 to 4, 5 and 6. We find that the performance of our model in Beijing Dataset decreases when the kernel size goes larger. The percentage errors for $k = 4, 5, 6$ are 11.18%, 11.31% and 11.38%. The reason might be that when the kernel size $k = 3$, a local path consists of two consecutive road segments which can exactly represent the turnings or road intersections (see Fig. 3.1). Thus, it is more easily for our model to capture such spatial features in the local paths.

3.4.5 Effect of Multi-task Learning

To show the effectiveness of the multi-task learning component, we first evaluate our model under different combination coefficient β from 0.0 to 0.99. The result is shown in Fig 3.5. We find that our model is pretty robust for a wide range of β. The high error rates are only found in two ends, i.e., when $\beta = 0$ or when β goes to 1. We then replace the attention mechanism in the multi-task learning component with the meaning pooling method. The error rates in Chengdu/Beijing Dataset increase to 12.09% and 11.11% respectively.

3.4.6 Incorporate Road Information

Recall that in the introduction part, we stress that our model does not rely on any map-matching algorithm but directly handle the raw GPS sequence. Nevertheless, it is very easy to incorporate the road information into our model when it is available. We collect the corresponding road ID of each GPS point in Beijing Dataset (there are

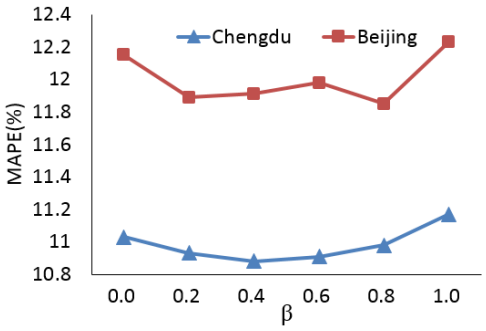


Figure 3.5 Error rates for different β.

189727 road segments in total). We embed each road ID to a 32-dimensional vector. For each GPS point, we use $p_i.rid$ to denote the embedded road ID of p_i. To utilize such road information, we can simply add $p_i.rid$ into Eq. (3-1), i.e., we have that

$$loc_i = \tanh(W_{loc} \cdot [p_i.lat \circ p_i.lng \circ p_i.rid])$$

in Geo-Conv layer. The MAPE of DeepTTE after incorporating the road information decreases from 10.92% to 10.59%.

3.4.7 Predicting Time

Despite the training time of DeepTTE is longer, the prediction is very efficient. During the test phase, estimating every 1000 paths takes DeepTTE 0.037s, RnnTTE 0.031s, MlpTTE 0.029s on GPU (including I/O time), GBDT 0.017s, and TEMP 0.47s. TEMP is slower for searching neighbors.

3.5 Related work

There is a large body of literature on the estimation of travel time; we only mention a few closely related ones.

3.5.1 Road Segment-Based Travel Time Estimation

Estimating travel time has been studied extensively[21,47,48]. However, these works estimated the travel time of individual road segment without considering the correlations between the roads. Yang et al.[19] used a spatial-temporal Hidden Markov Model

to formalize the relationships among the adjacent roads. Wang et al.[49] improved this work through an ensemble model based on two observed useful correlations in the traffic condition time series. Wang et al.[20] proposed an error-feedback recurrent Convolutional neural network called eRCNN for estimating the traffic speed on each individual road. These studies considered the correlation between different roads. However, they focused on accurately estimating the travel time or speed of individual road segment. As we mentioned in the Section 3.1, the travel time of a path is affected by various factors, such as the number of road intersections and the traffic lights in the path. Simply summing up the travel time of the road segments in the path does not lead to an accurate result[34].

3.5.2 Path-Based Travel Time Estimation

Rahmani et al.[50] estimated the travel time of a path based on the historical data of the path. However, the historical average based model may lead to a poor accuracy. Moreover, as new queried path may be not included in the historical data, it suffers from the data sparse problem. Yuan et al.[51] built a landmark graph based on the historical trajectories of taxis, where each landmark represents a single road. They estimate the travel time distribution of a path based on the landmark graph. However, as the landmarks are selected from the top-k frequently traversed road, the roads with few traveled records can not be estimated accurately. Furthermore, Wang et al.[22] estimated the travel time of the path, based on the sub-trajectories in the historical data. They used the tensor decomposition to complete the unseen sub-trajectory and such method enhance the accuracy effectively. Nevertheless, it still suffers from the data sparsity problem since there are many sub-trajectories which were visited by very few drivers. Yang et al.[52,53] used a hybrid graph to maintain the path weights and thus summarized the travel time distribution of different paths. They focused on estimating the distribution of travel time (instead of a real value), which is different from our setting.

3.5.3 Deep Learning in Spatial Temporal Data

Recently, the deep learning techniques demonstrate the strength on spatio-temporal data mining problems. Song et al.[54] built an intelligent system called Deep-Transport, for simulating the human mobility and transportation mode at a citywide level. Zhang et al.[55] proposed a deep spatio-temporal residual network for predicting the crowd flows. Dong et al.[56] studied characterizing the driving style of different drivers by a stacked recurrent neural network. To best of knowledge, no prior work studies estimating the travel time of the whole path based on the deep learning approach.

3.6 Conclusion

In this chapter, we study estimating the travel time for any given path. We propose an end-to-end framework based on deep neural networks. Our model can capture the spatial and temporal correlations in the given path, and transform the captured correlations into an effective feature representation. Our model also considers various factors which may affect the travel time such as the driver habit, the day of the week. We conduct extensive experiments on two very large scale real-world datasets. The results show that our model achieve a high estimation accuracy and outperforms the other off-the-shell methods significantly.

第 4 章　Missing Value Imputation with Recurrent Dynamics

In this chapter, we study handling missing values while learning spatio-temporal data representations. The proposed methods in this chapter enable our spatio-temporal data representation learning algorithms to further handle the missing values. Given multiple (spatially) correlated time series data, how to fill in missing values and to predict their class labels? Existing imputation methods often impose strong assumptions of the underlying data generating process, such as linear dynamics in the state space. In this chapter, we propose BRITS, a novel method based on recurrent neural networks for missing value imputation in time series data. Our proposed method directly learns the missing values in a bidirectional recurrent dynamical system, without any specific assumption. The imputed values are treated as variables of RNN graph and can be effectively updated during the backpropagation. BRITS has three advantages: (a) it can handle multiple correlated missing values in time series; (b) it generalizes to time series with nonlinear dynamics underlying; (c) it provides a data-driven imputation procedure and applies to general settings with missing data. We evaluate our model on three real-world datasets, including an air quality dataset, a health-care data, and a localization data for human activity. Experiments show that our model outperforms the state-of-the-art methods in both imputation and classification/regression accuracies.

4.1 Introduction

Multivariate time series data are abundant in many application areas, such as financial marketing [57,58], health-care[59,60], meteorology[61,62], and traffic engineering[55,63]. Time series are widely used as signals for classification/regression in various applications of corresponding areas. However, missing values in time series are very common, due to unexpected accidents, such as equipment damage or communication error, and may significantly harm the performance of downstream applications.

Much prior work proposed to fix the missing data problem with statistics and

machine learning approaches. However most of them require fairly strong assumptions on missing values. We can fill the missing values using classical statistical time series models such as ARMA or ARIMA (e.g., Ansley et al.[23]). But these models are essentially linear (after differencing). Kreindler et al.[14] assume that the data are smoothable, i.e., there is no sudden wave in the periods of missing values, hence imputing missing values can be done by smoothing over nearby values. Matrix completion has also been used to address missing value problems[15,64]. But it typically applies to only static data and requires strong assumptions such as low-rankness. We can also predict missing values by fitting a parametric data-generating model with the observations[16,17], which assumes that data of time series follow the distribution of hypothetical models. These assumptions make corresponding imputation algorithms less general, and the performance less desirable when the assumptions do not hold.

In this chapter, we propose BRITS, a novel method for filling the missing values for multiple correlated time series. Internally, BRITS adapts recurrent neural networks (RNN)[31] for imputing missing values, without any specific assumption over the data. Much prior work uses non-linear dynamical systems for time series prediction[65–67]. In our method, we instantiate the dynamical system as a bidirectional RNN, i.e., imputing missing values with bidirectional recurrent dynamics. In particular, we make the following technical contributions:

- We design a bidirectional RNN model for imputing missing values. We directly use RNN for predicting missing values, instead of tuning weights for smoothing as in the method proposed by Che et al.[59]. Our method does not impose specific assumption, hence works more generally than previous methods.
- We regard missing values as variables of the bidirectional RNN graph, which are involved in the backpropagation process. In such case, missing values get delayed gradients in both forward and backward directions with consistency constraints, which makes the estimation of missing values more accurate.
- We simultaneously perform missing value imputation and classification/regression of applications jointly in one neural graph. This alleviates the error propagation problem from imputation to classification/regression.

Additionally, the supervision of classification/regression makes the estimation of missing values more accurate.

- We evaluate our model on three real-world datasets, including an air quality dataset, a health-care dataset and a localization dataset of human activities. Experimental results show that our model outperforms the state-of-the-art models for both imputation and classification/regression accuracies.

4.2 Preliminary

We first present the problem formulation and some necessary preliminaries.

Definition 4 (Multivariate Time Series)： We denote a multivariate time series $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$ as a sequence of T observations. The t-th observation $\mathbf{x}_t \in \mathbb{R}^D$ consists of D features $\{x_t^1, x_t^2, \ldots, x_t^D\}$, and was observed at timestamp s_t (the time gap between different timestamps may be not same). In reality, due to unexpected accidents, such as equipment damage or communication error, $\mathbf{x}_t$ may have the missing values (e.g., in Fig. 4.1, x_1^3 in $\mathbf{x}_1$ is missing). To represent the missing values in $\mathbf{x}_t$, we introduce a *masking vector* $\mathbf{m}_t$ where,

$$\mathbf{m}_t^d = \begin{cases} 0 & \textit{if } x_t^d \textit{ is not observed} \\ 1 & \textit{otherwise} \end{cases}.$$

In many cases, some features can be missing for consecutive timestamps (e.g., blue blocks in Fig. 4.1). We define δ_t^d to be the time gap from the last observation to the current timestamp s_t, i.e.,

$$\delta_t^d = \begin{cases} s_t - s_{t-1} + \delta_{t-1}^d & \textit{if } t > 1, \mathbf{m}_{t-1}^d = 0 \\ s_t - s_{t-1} & \textit{if } t > 1, \mathbf{m}_{t-1}^d = 1 \\ 0 & \textit{if } t = 1 \end{cases}.$$

See Fig. 4.1 for an illustration.

In this chapter, we study a general setting for time series classification/regression problems with missing values. We use $\mathbf{y}$ to denote the label of corresponding classifi-

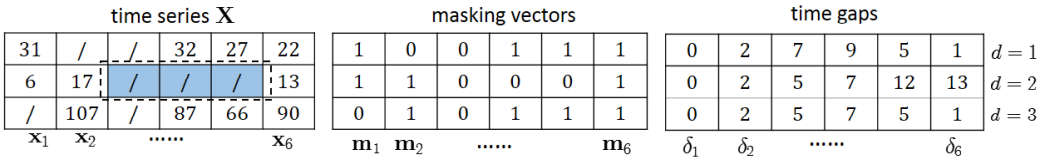


Figure 4.1 An example of multivariate time series with missing values. $\mathbf{x}_1$ to $\mathbf{x}_6$ are observed at $s_{1...6} = 0, 2, 7, 9, 14, 15$ respectively. Considering the 2nd feature in $\mathbf{x}_6$, the last observation of the 2nd feature took place at $s_2 = 2$, and we have that $\delta_6^2 = s_6 - s_2 = 13$.

cation/regression task. In general, $\mathbf{y}$ can be either a scalar or a vector. Our goal is to predict $\mathbf{y}$ based on the given time series $\mathbf{X}$. In the meantime, we impute the missing values in $\mathbf{X}$ as accurate as possible. In another word, we aim to design an effective multi-task learning algorithm for both classification/regression and imputation.

4.3 BRTIS

In this section, we describe the BRITS. Differing from the prior work which uses RNN to impute missing values in a smooth fashion[59], we learn the missing values directly in a recurrent dynamical system[68,69] based on the observed data. The missing values are thus imputed according to the recurrent dynamics, which significantly boosts both the imputation accuracy and the final classification/regression accuracy. We start the description with the simplest case: the variables observed at the same time are mutually uncorrelated. For such case, we propose algorithms for imputation with *unidirectional recurrent dynamics* and *bidirectional recurrent dynamics*, respectively. We further propose an algorithm for correlated multivariate time series in Section 4.3.3.

4.3.1 Unidirectional Uncorrelated Recurrent Imputation

For the simplest case, we assume that for the t-th step, x_t^i and x_t^j are uncorrelated if $i \neq j$ (but x_t^i may be correlated with some $x_{t' \neq t}^j$). We first propose an imputation algorithm by unidirectional recurrent dynamics, denoted as **RITS-I**.

In a *unidirectional recurrent dynamical system*, each value in the time series can be derived by its predecessors with a fixed *arbitrary* function[65–67]. Thus, we iteratively impute all the variables in the time series according to the recurrent dynamics. For

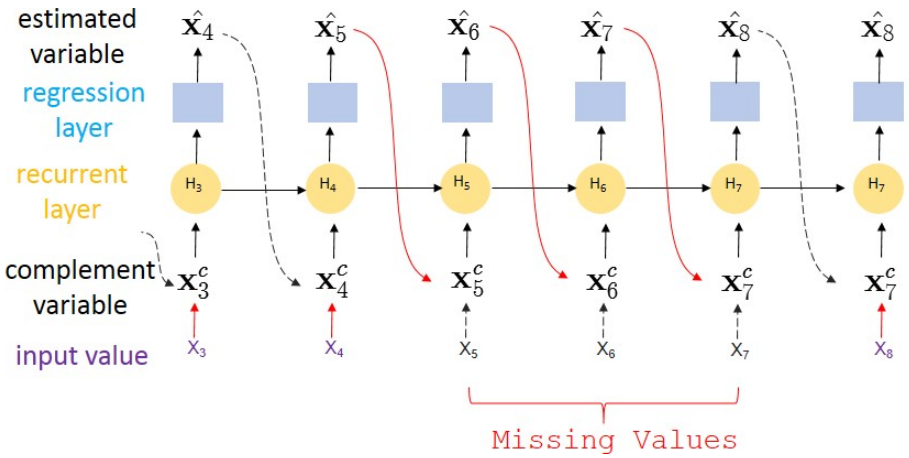


Figure 4.2 Imputation with unidirectional dynamics.

the t-th step, if $\mathbf{x}_t$ is actually observed, we use it to validate our imputation and pass $\mathbf{x}_t$ to the next recurrent steps. Otherwise, since the future observations are correlated with the current value, we replace $\mathbf{x}_t$ with the obtained imputation, and validate it by the future observations. To be more concrete, let us consider an example.

Example 1: Suppose we are given a time series $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{10}\}$, where $\mathbf{x}_5, \mathbf{x}_6$ and $\mathbf{x}_7$ are missing. According to the recurrent dynamics, at each time step t, we can obtain an estimation $\hat{\mathbf{x}}_t$ based on the previous $t-1$ steps. In the first 4 steps, the estimation error can be obtained immediately by calculating the estimation loss function $\mathcal{L}_e(\hat{\mathbf{x}}_t, \mathbf{x}_t)$ for $t = 1, \ldots, 4$. However, when $t = 5, 6, 7$, we cannot calculate the error immediately since the true values are missing. Nevertheless, note that at the 8-th step, $\hat{\mathbf{x}}_8$ depends on $\hat{\mathbf{x}}_5$ to $\hat{\mathbf{x}}_7$. We thus obtain a "delayed" error for $\hat{\mathbf{x}}_{t=5,6,7}$ at the 8-th step.

4.3.1.1 Algorithm

We introduce a recurrent component and a regression component for imputation. The recurrent component is achieved by a recurrent neural network and the regression component is achieved by a fully-connected network. A standard recurrent network[70]

can be represented as

$$\mathbf{h}_t = \sigma(\mathbf{W}_h\mathbf{h}_{t-1} + \mathbf{U}_h\mathbf{x}_t + \mathbf{b}_h),$$

where σ is the sigmoid function, $\mathbf{W}_h$, $\mathbf{U}_h$ and $\mathbf{b}_h$ are parameters, and $\mathbf{h}_t$ is the hidden state of previous time steps.

In our case, since $\mathbf{x}_t$ may have missing values, we cannot use $\mathbf{x}_t$ as the input directly as in the above equation. Instead, we use a "complement" input $\mathbf{x}_t^c$ derived by our algorithm when $\mathbf{x}_t$ is missing. Formally, we initialize the initial hidden state $\mathbf{h}_0$ as an all-zero vector and then update the model by:

$$\hat{\mathbf{x}}_t = \mathbf{W}_x\mathbf{h}_{t-1} + \mathbf{b}_x, \tag{4-1}$$

$$\mathbf{x}_t^c = \mathbf{m}_t \odot \mathbf{x}_t + (1 - \mathbf{m}_t) \odot \hat{\mathbf{x}}_t, \tag{4-2}$$

$$\gamma_t = \exp\{-\max(0, \mathbf{W}_\gamma\delta_t + \mathbf{b}_\gamma)\}, \tag{4-3}$$

$$\mathbf{h}_t = \sigma(\mathbf{W}_h[\mathbf{h}_{t-1} \odot \gamma_t] + \mathbf{U}_h[\mathbf{x}_t^c \circ \mathbf{m}_t] + \mathbf{b}_h), \tag{4-4}$$

$$\ell_t = \langle \mathbf{m}_t, \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{x}}_t) \rangle . \tag{4-5}$$

Eq. (4-1) is the regression component which transfers the hidden state $\mathbf{h}_{t-1}$ to the estimated vector $\hat{\mathbf{x}}_t$. In Eq. (4-2), we replace missing values in $\mathbf{x}_t$ with corresponding values in $\hat{\mathbf{x}}_t$, and obtain the complement vector $\mathbf{x}_t^c$. Besides, since the time series may be irregularly sampled, in Eq. (4-3), we further introduce a *temporal decay factor* γ_t to decay the hidden vector $\mathbf{h}_{t-1}$. Intuitively, if δ_t is large (i.e., the values are missing for a long time), we expect a small γ_t to decay the hidden state. Such factor also represents the missing patterns in the time series which is critical to imputation[59]. In Eq. (4-4), based on the decayed hidden state, we predict the next state $\mathbf{h}_t$. Here, $\circ$ indicates the concatenate operation. In the mean time, we calculate the estimation error by the estimation loss function $\mathcal{L}_e$ in Eq. (4-5). In our experiment, we use the *mean absolute error* for $\mathcal{L}_e$. Finally, we predict the task label $\mathbf{y}$ as

$$\hat{\mathbf{y}} = f_{out}(\sum_{i=1}^{T} \alpha_i\mathbf{h}_i),$$

where f_{out} can be either a fully-connected layer or a softmax layer depending on the

specific task, and α_i is the weight for different hidden state which can be derived by the *attention mechanism*① or the *mean pooling mechanism*, i.e., $\alpha_i = \frac{1}{T}$. We calculate the output loss by $\mathcal{L}_{out}(\mathbf{y}, \hat{\mathbf{y}})$. Our model is then updated by minimizing the accumulated loss $\frac{1}{T}\sum_{t=1}^{T} \ell_t + \mathcal{L}_{out}(\mathbf{y}, \hat{\mathbf{y}})$.

4.3.1.2 Practical Issues

In practice, we use LSTM as the recurrent component in Eq. (4-4) to prevent the gradient vanishing/exploding problems in vanilla RNN[70]. Standard RNN models including LSTM treat $\hat{\mathbf{x}}_t$ as a constant. During backpropagation, gradients are cut at $\hat{\mathbf{x}}_t$. This makes the estimation errors backpropagate insufficiently. For example, in Example 1, the estimation errors of $\hat{\mathbf{x}}_5$ to $\hat{\mathbf{x}}_7$ are obtained at the 8-th step as the delayed errors. However, if we treat $\hat{\mathbf{x}}_5$ to $\hat{\mathbf{x}}_7$ as constants, such delayed error cannot be fully backpropagated. To overcome such issue, we treat $\hat{\mathbf{x}}_t$ as a variable of RNN graph. We let the estimation error passes through $\hat{\mathbf{x}}_t$ during the backpropagation. Fig. 4.2 shows how RITS-I method works in Example 1. In this example, the gradients are backpropagated through the opposite direction of solid lines. Thus, the delayed error ℓ_8 is passed to steps 5, 6 and 7. In the experiment, we find that our models are unstable during training if we treat $\hat{\mathbf{x}}_t$ as a constant.

4.3.2 Bidirectional Uncorrelated Recurrent Imputation

In the RITS-I , errors of estimated missing values are delayed until the presence of the next observation. For example, in Example 1, the error of $\hat{\mathbf{x}}_5$ is delayed until $\mathbf{x}_8$ is observed. Such error delay makes the model converge slowly and in turn leads to inefficiency in training. In the meantime, it also leads to the *bias exploding* problem[71], i.e., the mistakes made early in the sequential prediction are fed as input to the model and may be quickly amplified. In this section, we propose an improved version called **BRITS-I** . The algorithm alleviates the above-mentioned issues by utilizing the *bidirectional recurrent dynamics* on the given time series, i.e., besides the forward di-

① The design of attention mechanism is out of this chapter's scope.

rection, each value in time series can be also derived from the backward direction by another fixed arbitrary function.

To illustrate the intuition of BRITS-I algorithm, again, we consider Example 1. Consider the backward direction of the time series. In bidirectional recurrent dynamics, the estimation $\hat{\mathbf{x}}_4$ reversely depends on $\hat{\mathbf{x}}_5$ to $\hat{\mathbf{x}}_7$. Thus, the error in the 5-th step is back-propagated from not only the 8-th step in the forward direction (which is far from the current position), but also the 4-th step in the backward direction (which is closer). Formally, the BRITS-I algorithm performs the RITS-I as shown in Eq. (4-1) to Eq. (4-5) in forward and backward directions, respectively. In the forward direction, we obtain the estimation sequence $\{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_T\}$ and the loss sequence $\{\ell_1, \ell_2, \ldots, \ell_T\}$. Similarly, in the backward direction, we obtain another estimation sequence $\{\hat{\mathbf{x}}'_1, \hat{\mathbf{x}}'_2, \ldots, \hat{\mathbf{x}}'_T\}$ and another loss sequence $\{\ell_1', \ell_2', \ldots, \ell_T'\}$. We enforce the prediction in each step to be consistent in both directions by introducing the "*consistency loss*":

$$\ell_t^{cons} = \text{Discrepancy}(\hat{\mathbf{x}}_t, \hat{\mathbf{x}}'_t) \tag{4-6}$$

where we use the *mean squared error* as the discrepancy in our experiment. The final estimation loss is obtained by accumulating the forward loss ℓ_t, the backward loss ℓ'_t and the consistency loss ℓ_t^{cons}. The final estimation in the t-th step is the mean of $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{x}}_t'$.

4.3.3 Correlated Recurrent Imputation

The previously proposed algorithms RITS-I and BRITS-I assume that features observed at the same time are mutually uncorrelated. This may be not true in some applications. For example, in the air quality data[72], each feature represents one measurement in a monitoring station. Obviously, the observed measurements are spatially correlated. In general, the measurement in one monitoring station is close to those values observed in its neighboring stations. In this case, we can estimate a missing measurement according to both its historical data, and the measurements in its neighbors.

In this section, we propose an algorithm, which utilizes the feature correlations in

the unidirectional recurrent dynamical system. We refer to such algorithm as **RITS**. The feature correlated algorithm for bidirectional case (i.e., **BRITS**) can be derived in the same way. Note that in Section 4.3.1, the estimation $\hat{\mathbf{x}}_t$ is only correlated with the historical observations, but irrelevant with the the current observation. We refer to $\hat{\mathbf{x}}_t$ as a "history-based estimation". In this section, we derive another "feature-based estimation" for each x_t^d, based on the other features at time s_t. Specifically, at the t-th step, we first obtain the complement observation $\mathbf{x}_t^c$ by Eq. (4-1) and Eq. (4-2). Then, we define our feature-based estimation as $\hat{\mathbf{z}}_t$ where

$$\hat{\mathbf{z}}_t = \mathbf{W}_z \mathbf{x}_t^c + \mathbf{b}_z, \tag{4-7}$$

$\mathbf{W}_z$, $\mathbf{b}_z$ are corresponding parameters. We restrict the diagonal of parameter matrix $\mathbf{W}_z$ to be all zeros. Thus, the d-th element in $\hat{\mathbf{z}}_t$ is exactly the estimation of x_t^d, based on the other features. We further combine the historical-based estimation $\hat{\mathbf{x}}_t$ and the feature-based estimation $\hat{\mathbf{z}}_t$. We denote the combined vector as $\hat{\mathbf{c}}_t$, and we have that

$$\beta_t = \sigma(\mathbf{W}_\beta[\gamma_t \circ \mathbf{m}_t] + \mathbf{b}_\beta) \tag{4-8}$$

$$\hat{\mathbf{c}}_t = \beta_\mathbf{t} \odot \hat{\mathbf{z}}_t + (1 - \beta_\mathbf{t}) \odot \hat{\mathbf{x}}_t. \tag{4-9}$$

Here we use $\beta_t \in [0, 1]^D$ as the weight of combining the history-based estimation $\hat{\mathbf{x}}_t$ and the feature-based estimation $\hat{\mathbf{z}}_t$. Note that $\hat{\mathbf{z}}_t$ is derived from $\mathbf{x}_t^c$ by Eq. (4-7). The elements of $\mathbf{x}_t^c$ can be history-based estimations or truly observed values, depending on the presence of the observations. Thus, we learn the weight β_t by considering both the temporal decay γ_t and the masking vector $\mathbf{m}_t$ as shown in Eq. (4-8). The rest parts are similar as the feature uncorrelated case. We first replace missing values in $\mathbf{x}_t$ with the corresponding values in $\hat{\mathbf{c}}_t$. The obtained vector is then fed to the next recurrent step to predict memory $\mathbf{h}_t$:

$$\mathbf{c}_t^c = \mathbf{m}_t \odot \mathbf{x}_t + (1 - \mathbf{m}_t) \odot \hat{\mathbf{c}}_t \tag{4-10}$$

$$\mathbf{h}_t = \sigma(\mathbf{W}_h[\mathbf{h}_{t-1} \odot \gamma_t] + \mathbf{U}_h[\mathbf{c}_t^c \circ \mathbf{m}_t] + \mathbf{b}_h). \tag{4-11}$$

However, the estimation loss is slightly different with the feature uncorrelated case. We find that simply using $\ell_t = \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{c}}_t)$ leads to a very slow convergence speed. Instead,

we accumulate all the estimation errors of $\hat{\mathbf{x}}_t$, $\hat{\mathbf{z}}_t$ and $\hat{\mathbf{c}}_t$:

$$\ell_t = \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{x}}_t) + \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{z}}_t) + \mathcal{L}_e(\mathbf{x}_t, \hat{\mathbf{c}}_t).$$

4.4 Experiment

Our proposed methods are applicable to a wide variety of applications. We evaluate our methods on three different real-world datasets. The download links of the datasets, as well as the implementation codes can be found in the GitHub page① .

4.4.1 Dataset Description

4.4.1.1 Air Quality Data

We evaluate our models on the air quality dataset, which consists of PM2.5 measurements from 36 monitoring stations in Beijing. The measurements are hourly collected from 2014/05/01 to 2015/04/30. Overall, there are 13.3% values are missing. For this dataset, we do pure imputation task. We use the exactly same train/test setting as in prior work[72], i.e., we use the 3rd, 6th, 9th and 12th months as the test data and the other months as the training data. To train our model, we randomly select every 36 consecutive steps as one time series.

4.4.1.2 Health-care Data

We evaluate our models on health-care data in *PhysioNet Challenge 2012*[73], which consists of 4000 multivariate clinical time series from intensive care unit (ICU). Each time series contains 35 measurements such as Albumin, heart-rate etc., which are irregularly sampled at the first 48 hours after the patient's admission to ICU. We stress that this dataset is extremely sparse. There are up to 78% missing values in total. For this dataset, we do both the imputation task and the classification task. To evaluate the imputation performance, we randomly eliminate 10% of observed measurements from data and use them as the ground-truth. At the same time, we predict the in-hospital

① https://github.com/NIPS-BRITS/BRITS

death of each patient as a binary classification task. Note that the eliminated measurements are only used for validating the imputation, and they are never visible to the model.

4.4.1.3 Localization for Human Activity Data

The UCI localization data for human activity[74] contains records of five people performing different activities such as walking, falling, sitting down etc (there are 11 activities). Each person wore four sensors on her/his left/right ankle, chest, and belt. Each sensor recorded a 3-dimensional coordinates for about 20 to 40 millisecond. We randomly select 40 consecutive steps as one time series, and there are $30,917$ time series in total. For this dataset, we do both imputation and classification tasks. Similarly, we randomly eliminate 10% observed data as the imputation ground-truth. We further predict the corresponding activity of observed time series (i.e., walking, sitting, etc).

4.4.2 Experiment Setting

4.4.2.1 Model Implementations

We fix the dimension of hidden state $\mathbf{h}_t$ in RNN to be 64. We train our model by an Adam optimizer with learning rate 0.001 and batch size 64. For all the tasks, we normalize the numerical values to have zero mean and unit variance for stable training.

We use different early stopping strategies for pure imputation task and classification tasks. For the imputation tasks, we randomly select 10% of non-missing values as the validation data. The early stopping is thus performed based on the validation error. For the classification tasks, we first pre-train the model as an imputation task. Then we use 5-fold cross validation to further optimize both the imputation and classification losses simultaneously.

We evaluate the imputation performance in terms of *mean absolute error* (`MAE`) and *mean relative error* (`MRE`). Suppose that $\texttt{label}_i$ is the ground-truth of the i-th item, $\texttt{pred}_i$ is the output of the i-th item, and there are N items in total. Then, `MAE` and

MRE are defined as

$$\texttt{MAE} = \frac{\sum_i |\texttt{pred}_i - \texttt{label}_i|}{N}, \quad \texttt{MRE} = \frac{\sum_i |\texttt{pred}_i - \texttt{label}_i|}{\sum_i |\texttt{label}_i|}.$$

For air quality data, the evaluation is performed on its original data. For heath-care data and activity data, since the numerical values are not in the same scale, we evaluate the performances on their normalized data. To further evaluate the classification performances, we use *area under ROC curve* (AUC)[75] for health-care data, since such data is highly unbalanced (there are 10% patients who died in hospital). We then use standard accuracy for the activity data, since different activities are relatively balanced.

4.4.2.2 Baseline Methods

We compare our model with both RNN-based methods and non-RNN based methods. The non-RNN based imputation methods include:

- **Mean:** We simply replace the missing values with corresponding global mean.
- **KNN:** We use k-nearest neighbor[46] to find the similar samples, and impute the missing values with weighted average of its neighbors.
- **Matrix Factorization (MF):** We factorize the data matrix into two low-rank matrices, and fill the missing values by matrix completion[46].
- **MICE**: We use Multiple Imputation by Chained Equations (MICE), a widely used imputation method, to fill the missing values. MICE creates multiple imputations with chained equations[17].
- **ImputeTS**: We use ImputeTS package in R to impute the missing values. ImputeTS is a widely used package for missing value imputation, which utilizes the state space model and kalman smoothing[76].
- **STMVL**: Specifically, we use STMVL for the air quality data imputation. STMVL is the state-of-the-art method for air quality data imputation. It further utilizes the geo-locations when imputing missing values[72].

We implement KNN, MF and MICE based on the python package fancyimpute① . In recent studies, RNN-based models in missing value imputations achieve remarkable

① https://github.com/iskandr/fancyimpute

Table 4.1 Performance Comparison for Imputation Tasks (in MAE(MRE%))

Method		Air Quality	Health-care	Human Activity
Non-RNN	Mean	55.51 (77.97%)	0.720 (100.00%)	0.767 (96.43%)
	KNN	29.79 (41.85%)	0.732 (101.66%)	0.479 (58.54%)
	MF	27.94 (39.25%)	0.622 (87.68%)	0.879 (110.44%)
	MICE	27.42 (38.52%)	0.634 (89.17%)	0.477 (57.94%)
	ImputeTS	19.58 (27.51%)	0.390 (54.2%)	0.363 (45.65%)
	STMVL	12.12 (17.40%)	/	/
RNN	GRU-D	/	0.559 (77.58%)	0.558 (70.05%)
	M-RNN	14.24 (20.43%)	0.451 (62.65%)	0.248 (31.19%)
Ours	RITS-I	12.73 (18.32%)	0.395 (54.80%)	0.240 (30.10%)
	BRITS-I	11.58 (16.66%)	0.361 (50.01%)	0.220 (27.61%)
	RITS	12.19 (17.54%)	0.300 (41.89%)	0.248 (31.21%)
	BRITS	**11.56** (**16.65**%)	**0.281** (**39.14**%)	**0.219** (**27.59**%)

performances[59,77–79]. We also compare our model with existing RNN-based imputation methods, including:

- **GRU-D:** GRU-D is proposed for handling missing values in health-care data. It imputes each missing value by the weighted combination of its last observation, and the global mean, together with a recurrent component[59].
- **M-RNN:** M-RNN also uses bi-directional RNN. It imputes the missing values according to hidden states in both directions in RNN. M-RNN treats the imputed values as constants. It does not consider the correlations among different missing values[79].

We compare the baseline methods with our four models: RITS-I (Section 4.3.1), RITS (Section 4.3.2), BRITS-I (Section 4.3.3) and BRITS (Section 4.3.3). We implement all the RNN-based models with PyTorch, a widely used package for deep learning. All models are trained with GPU GTX 1080.

4.4.3 Experiment Results

Table 4.1 shows the imputation results. As we can see, simply applying naıve mean imputation is very inaccurate. Alternatively, KNN, MF, and MICE perform much better than mean imputation. ImputeTS achieves the best performance among all the non-RNN methods, especially for the heath-care data (which is smooth and contains few sudden waves). STMVL performs well on the air quality data. However, it is specifically designed for geographical data, and cannot be applied in the other datasets. Most of RNN-based methods, except GRU-D, demonstrates significantly better performances in imputation tasks. We stress that GRU-D does not impute missing value explicitly. It actually performs well on classification tasks. M-RNN uses explicitly imputation procedure, and achieves remarkable imputation results. Our model BRITS outperforms all the baseline models significantly. According to the performances of our four models, we also find that both bidirectional recurrent dynamics, and the feature correlations are helpful to enhance the model performance.

We also compare the accuracies of all RNN-based models in classification tasks. Table 4.2 shows the performances of different RNN-based models on the classification tasks. As we can see, our model BRITS outperforms other baseline methods for classifications. Comparing with Table 4.1, GRU-D performs well for classification tasks. Furthermore, we find that it is very important for GRU-D to carefully apply the dropout techniques in order to prevent the overfitting (we use $p = 0.25$ dropout layer on the top classification layer). However, our models further utilize the imputation errors as the supervised signals. It seems that dropout is not necessary for our models during training.

To better understand our model, we generate a set of univariate synthetic time series. Speficically, we randomly generate a time series with length $T = 36$, using the state-space representation[80]:

$$\begin{aligned} x_t &= \mu_t + \theta_t + \epsilon_t, \\ \mu_t &= \mu_{t-1} + \lambda_{t-1} + \xi_t, \\ \lambda_t &= \lambda_{t-1} + \zeta_t, \end{aligned}$$

Table 4.2　Performance Comparison for Classification Tasks

Method	Health-care (AUC)	Human Activity (Accuracy)
GRU-D	0.828 ± 0.004	0.939 ± 0.010
M-RNN	0.800 ± 0.003	0.938 ± 0.010
RITS-I	0.821 ± 0.007	0.934 ± 0.008
BRITS-I	0.831 ± 0.003	0.940 ± 0.012
RITS	0.840 ± 0.004	0.968 ± 0.010
BRITS	**0.850 ± 0.002**	**0.969 ± 0.008**

$$\theta_t = \sum_{j=1}^{s-1} -\theta_{t-j} + \omega_t,$$

where x_t is the t-th value in time series. The residual terms ϵ_t, ξ_t, ζ_t and ω_t are randomly drawn from a normal distribution $\mathrm{N}(0, 0.3)$. We eliminate about 22% values from the generated series, and compare our model BRITS-I (note the data is univariate) and ImputeTS. We show three examples in Fig. 4.3.

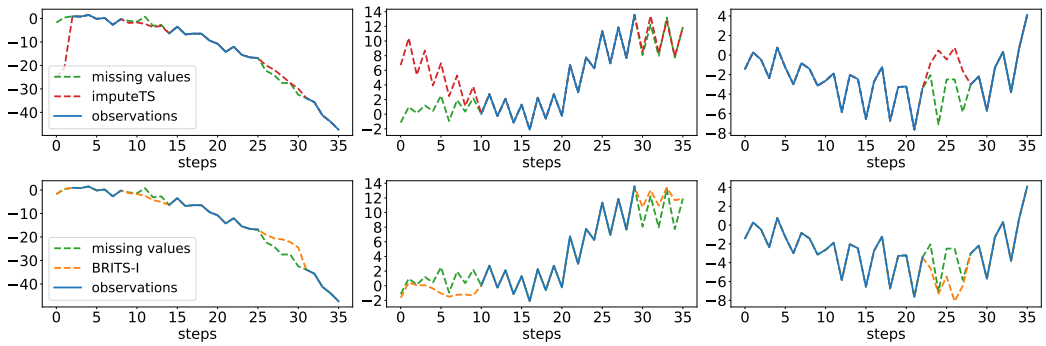


Figure 4.3　Example for synthetic time series imputation. Each column corresponds to one time series. The figures in the first row are imputations of ImputeTS algorithm, and the figures in the second row are imputations by BRITS-I .

The first row corresponds to the imputations of ImputeTS and the second row corresponds to our model BRITS. As we can see, our model demonstrates better performance than ImputeTS. Especially, ImputeTS fails when the start part of time series is missing. However, for our model, the imputation errors are backpropagated to pre-

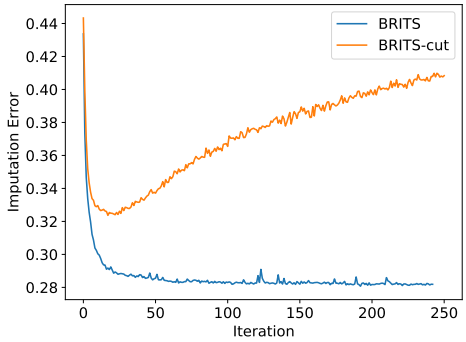


Figure 4.4　Validation errors for BRITS and BRITS-cut during training

vious time steps. Thus, our model can adjust the imputations in the start part with delayed gradient, which leads to much more accurate results.

Discussion As we claimed in Section 4.3.1, we regard the missing values as variables of RNN graph. During the backpropagation, the imputed values can be further updated sufficiently. In the experiment, we find that if we cut the gradient of $\hat{\mathbf{x}}_t$ (i.e., regard it as constant), the models are unstable and easy to overfit during the training. We refer to the model with non-differentiate $\hat{\mathbf{x}}_t$ as BRITS-cut. Fig. 4.4 shows the validation errors of BRITS and BRITS-cut during training for the health-care data imputation. At the first 20 iterations, the validation error of BRITS-cut decreases fast. However, it soon fails due to the overfitting.

4.5　Related Work

There is a large body of literature on the imputation of missing values in time series. We only mention a few closely related ones. The *interpolate method* tries to fit a "smooth curve" to the observations and thus reconstruct the missing values by the local interpolations[14,16]. Such method discards any relationships between the variables over time. The *autoregressive method*, including ARIMA, SARIMA etc., eliminates the non-stationary parts in the time series data and fit a parametrized stationary model. The *state space model* further combines the ARIMA and the Kalman Filter[16], which provides more accurate results. These methods usually apply to the univariate time series imputations. For multivariate time series imputation, *Multivariate Imputation by*

Chained Equations (MICE)[17] first initialize the missing values arbitrarily and then estimates each missing variable by Gibbs sampling. The graphical model[81] introduces a latent variable for each missing value, and finds the latent variables by learning their transition matrix. There are also some data-driven methods for missing value imputation. Yi et al.[72] imputes the missing values in air quality data with geographical features. Wang et al.[15] imputes the missing values in recommendation system with collaborative filtering.

Recently, some researchers attempted to impute the missing value with recurrent neural networks[18,77,78]. The recurrent components are trained together with the classification/regression component, which significantly boosts the accuracy. However, they assumed that the missing values can be represented as the combination of global mean and the most recent observations. They used RNN to impute the missing data in a smooth fashion but ignores the relationships in the recurrent dynamics. To the best of our knowledge, all of the existing methods treated the imputed values as static constants and have limited utilization of future information. Differ from the existing methods, our model directly approximates the recurrent dynamical system and demonstrates superior results.

4.6 Conclusion

In this chapter, we proposed BRITS, a novel method to use recurrent dynamics to effectively impute the missing values in multivariate time series. Instead of imposing assumptions over the data-generating process, our model learns the representations of data correlation in a bidirectional recurrent dynamical system, and imputes missing values based on the learned representations directly. Our model treats missing values as variables of the bidirectional RNN graph. Thus, we get the delayed gradients for missing values in both forward and backward directions, which makes the imputation of missing values more accurate. We performed the missing value imputation and classification/regression simultaneously within a joint neural network. Experiment results show that our model demonstrates more accurate results for both imputation and clas-

sification/regression than state-of-the-art methods.

第 5 章 Handling Massiveness in User Identification

We further study the representation learning for extremely massive spatio-temporal data in this chapter. Specifically, we use the user identification problem as an example. In this problem, we investigate efficient ways of identifying users across heterogeneous data sources, such as different GPS-embedded devices, mobile apps or location-based service providers. We present a MapReduce-based framework called *Automatic User Identification* (AUI) which is easy to deploy and can scale to a very large data set. Our framework is based on a novel similarity measure called the *signal based similarity* (SIG) which measures the similarity of users' trajectories gathered from different data sources, typically with very different sampling rates and noise patterns. In signal based similarity, we adopt a co-occurrence representation of users' trajectories in multiple resolutions, and calculate the similarity scores based on the obtained representations. Such similarity measure is very robust for the massive mobility datasets. We conduct extensive experimental evaluations, which show that our framework outperforms the existing methods significantly.

5.1 Introduction

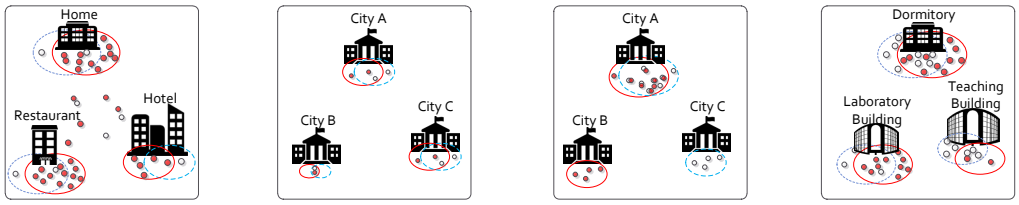
Ubiquitous location based services and applications have enabled people to use GPS-embedded devices for navigation, travel planning and geolocation information sharing in their daily life. Such mobility data is now collected routinely at a very large scale. The large volume of mobility data gives rise to new opportunities for discovering patterns and characteristics of human mobility behaviors. An increasing number of researches empowered by mobility data has emerged recently. Meanwhile, mining of such mobility data also shows great potentials in various industrial and commercial applications, including traffic analysis[1–3], travel recommendation[4–7,82–85], location-based social network[8–13], geographical searching[86,87] etc.

In real applications, mobility data is usually generated from heterogeneous data

sources, such as different GPS-embedded devices, mobile apps, or LBS providers etc. In this chapter, we aim to study the efficient approach of identifying users from mobility datasets collected from heterogeneous sources. A closely related topic is *user similarity search* where the goal is to retrieve a subset of users with similar spatio-temporal patterns. Essentially, we can think of the user identification problem as a special case of user similarity search problem. For each trajectory, we retrieve the most similar ones from another data source and identify whether they belong to the same person.

Despite the fact that the user similarity search has been studied quite extensively[83,88–91], not much work has investigated the mobility data collected from heterogeneous data sources, which is much more complicated. In our experiments, the data is collected from various mobile apps, including the navigation data, map queries, geo-tagged records in social platforms etc (see Section 5.6 for details). To make our exposition more concrete, we first illustrate some distinct features of our data and our objective.

- The most distinctive feature we have found about the trajectories is the sheer variety of the sampling rates among different sources. For example, trajectories of GPS navigation data are usually sampled at a very high rate whereas the geo-tags or map queries are sampled with an extremely low rate. Even in the same data source, the utilization frequencies could vary drastically during different time intervals. On the other hand, most prior work uses approximately uniformly (and often densely) sampled data[83,88,89,92].
- For some sparse trajectories, it is almost impossible to infer any information of user movement. For example, for the geo-tagged check-in data in our data set, each user only generated one GPS record every 2.63 days on average. Most of prior work measures the trajectory similarities based on the movement behaviors of the users such as the speed, move direction, spatial-temporal closeness (i.e., two trajectories are similar only if they appeared in approximately the same place at approximately the same time)[88,92–96]. However, such features are not available in our data due to the extreme sparsity.
- The trajecotries of the users with close relationship usually have a significant



(a) Trajectories of the same person which are sampled at very different rates.

(b) Trajectories of the same person which co-occurred in several places far apart from each other.

(c) Trajectories of the same person which are disjoint in several cities.

(d) Trajectories of two school mates which have significant overlap.

Figure 5.1　Four typical cases observed in the real dataset

overlap. For example, we investigate the mobility data of several students who study in University T. Most of their trajectories lie on their department buildings and dormitories. Such overlap renders it difficult to distinguish them. However, few prior work investigates user identification problem under such case.

- For different data sources, the trajectories can be temporally disjoint. For example, two data sets of the same group of anonymized users with inconsisten user id, one is collected at January and another is collected at February. Especially, for the businessmen, they may go to several different cities during several months which makes it difficult to measure their similarities.

To provide some intuitions for the readers and to illustrate the challenges, we show an example in Example 2.

Example 2： *In Figure 5.1, we show four typical cases observed in the real datasets (there are many other cases or combinations of those cases, that are impossible to list exhaustively).*

Each of (a)(b)(c) represents two trajectories (from different data sources) that belong to the same person and (d) shows the trajectories that belong to two schoolmates.

- *In (a), the white trajectory is sampled at a much lower rate than the red trajectory but they both occurred in several fixed places frequently. Such co-occurrences*

are significant for identifying the same person, especially when they take place far apart from each other, such as several different cities, as shown in (b).

- *However, in (c), we observed from the data that the trajectories of the same person can be also disjoint in several cities. Such case often happens when the trajectories are temporally disjoint as well. However, as they co-occurred significantly in one city (city A), we can still identify that they belong to the same person.*
- *In (d), as the trajectories in the same campus have significant overlap, it is hard to distinguish the users from their schoolmates. Nevertheless, the red trajectories occurred more in the laboratory building while the white trajectory tends to go to the teaching building more. Such pattern enables us to identify them uniquely.* □

The above features make our user identification problem very different from the previous trajectory similarity problems[83,88–90,92,95–98] in that the trajectories we deal with are sampled at very different rates, and extremely noisy. To address the challenge, we propose a MapReduce-based framework, called *Automatic User Identification* (AUI), which is based on a novel trajectory similarity measure. AUI is easy to deploy and can scale to very large data set. We summarize our technical contributions below:

- We formulate the user identification problem over large scale heterogeneous mobility datasets and we present a MapReduce-based frame called AUI.
- We design an effective filtering strategy based on the MapReduce-based framework. With the filtering strategy, for each trajectory we only need to compare it with a small number of candidates. The filtering strategy is the foundation of that AUI can scale to very large data sets.
- We design a novel similarity measure called the *signal based similarity* (SIG) by considering the frequencies of the co-occurrences and the locations where they took place. Since our data is collected from many different data sources, we do not assume any property of the mobility data (e.g., sample rate, time span). We show that compared with the existing measures, our measure can handle the ex-

tremely noisy cases effectively. The experiment result shows that our measure is more robust and accurate for our user identification problem with heterogeneous data sources.

- We adopt a rejection strategy in order to reduce the mis-identification cases. Many application scenarios are highly sensitive to misidentification, i.e. a few erroneous cases could lead to serious consequences. Our strategy enhances the accuracy of the framework significantly.
- We evaluate our framework by 6 experiments of different cases. For the easiest case (31511 users in China), we achieve an accuracy of 99.94%. For the hardest case (14115 college students who study in the same campus), we achieve an accuracy of 90.09% whereas the best existing method only achieves an accuracy of 61.38% in this case.

5.2 Formulation and Overview

The problem studies the user identification across heterogeneous data sources. We first present several useful definitions.

Definition 5 (Trajectory)： A trajectory $T = \{p_1, p_2, \ldots, p_{|T|}\}$ is a temporally ordered sequence of spatio-temporal points. Each point p_i is associated with three attributes x, y, t where the pair (x, y) ① indicates the coordinate of p_i and t indicates the timestamp when p_i was recorded.

Here we stress that the trajectories in our data could be extremely sparse, i.e., there may be less than one spatio-temporal point per day in average.

Definition 6 (Mobility data set)： A mobility data set D is defined as a collection of trajectories. Each trajectory $T \in D$ is associated with an id $T.id$.

Definition 7 (Matching trajectory)： Suppose two trajectories T_A, T_B are collected from different mobility data sets. If T_A and T_B are generated from the same user, then we call T_B a matching trajectory of T_A.

① We use the mercator coordinate in our experiment.

Our problem is defined as follow. Given two mobility data sets D_A and D_B (usually collected from two different data sources), for each $T_A \in D_A$, our goal is to identify whether there exists $T_B \in D_B$ which is the matching trajectory of T_A. We do not assume any property of the mobility data set. Thus, the sampling rates of the trajectories could be very high or extremely low. Furthermore, D_A and D_B could be temporally disjoint, i.e., they are collected at different time intervals.

We explain the reason why we can achieve user identification across heterogeneous mobility data sources. Montjoye et al.[99] showed that the human mobilities are highly unique. Each individual has her/his own mobility pattern. People tend to visit the places where they often visited in the past. Even for the users with very close relationship, they still have noticeable different mobility patterns as we illustrated in Fig 5.1(d). Such uniqueness of human mobility allows us to identify trajectories that belong to the same user from different sources.

To handle the extremely sparse trajectories (which makes it hard to infer any mobility pattern from the daily data), we accumulate the trajectories during a long time interval (for example, the trajectories during 3 months). Thus, by accumulating the historical mobility data, it is possible to infer the mobility pattern of a user such as the places he/she tends to visit. Fig. 5.2 shows an example in our data set where the trajectory is collected from February to May in 2015 with 5 points per day in average. By accumulating all these points, we can clearly see that this user usually stay at home and the workplace. Moreover, he/she had visited Wangfujing and Beijing Botanical Garden one day in the past 3 months. Motivated by this, we consider both the frequency of

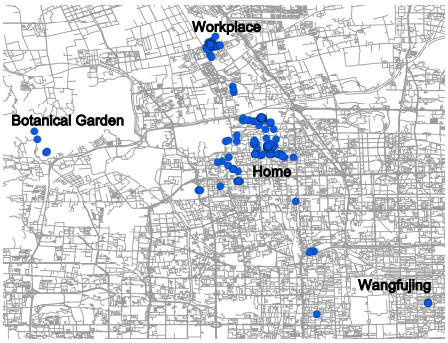


Figure 5.2　A trajectory in our data.

spatial co-occurrences (i.e., two trajectories co-occurred at approximately the same location) under different granularities and the locations where they took place. We define our trajectory similarity mainly based on these factors.

We first present an overview of AUI. Our framework consists of three stages: the pre-processing stage, the multi-resolution filtering stage and the verification stage. All of the stages are implemented on the MapReduce frame. In the pre-processing stage, we first perform data compression by transforming each trajectory into consecutive transitions between a set of *stay points*, i.e., the locations where the user stays for a while rather than just passing by. Since for large scale data sets, pairwise comparison is rather expensive, in the multi-resolution filtering stage, we partition the map into multiple grids with multi-resolution. For each trajectory $T_A \in D_A$, we select a small subset of trajectories in D_B which co-occurred frequently in multiple cells with T_A as the candidate set of T_A. Finally, as the trajectories we deal with are sampled at very different rates and extremely noisy, in the verification stage, we evaluate each candidate with the signal based similarity which can handle such cases effectively and we carefully select the matching trajectory. We further extend our similarity measure and adopt an effective rejection strategy in the verification stage to reduce the misidentification cases (Section 5.5).

5.3 Pre-Processing

The pre-processing stage transforms each trajectory into a set of stay points, i.e., the location points that the user stay for a while rather than passing by. Thus stay points usually carry the particular semantic meanings such as the building they live or the park they went[89]. The pre-processing stage consists of map-only jobs. For each trajectory $T \in D_A/D_B$, the map function takes $T.id$ as the key and T as the value. We use a similar method as in[89] to pre-process the trajectories. For each trajectory in a mobility data set, we split it into consecutive segments. A segment is defined as a series of continuous location points sharing the same mobility status, such as stay, move and pass-by. For example, a user drove for 2 hours to a shopping mall and spent 3 hours in the mall, then drove to another place. Such trajectory can be split into 3 segments: driving to the mall,

Algorithm 1: Pre-Processing

```
1  Map(⟨T.id, T⟩)
2  S ← {}, P ← {p_1};
3  for i = 2 ... |T| do
4      p_b ← P[1] ;
5      t ← p_i.t − p_b.t;
6      d ← Dis(p_i, p_b);
7      if d ≥ Δ_d or t ≥ Δ_t^u then
8          if t ≥ Δ_t^l then
9              sp.loc ← MeanCoordinate(P) ; // stay point
10             sp.cnt ← |P|; // number of around points
11             S.add(sp) ; // add the stay point into S
12         P ← {p_i}; // start a new segment
13     else
14         P.add(p_i); // add p_i into current segment
15 emit(⟨T.id, S⟩)
```

staying at the mall and driving to another place. For each segment under stay status, we use the geometric center of the segment as the corresponding stay point. We then use a series of stay points as the compressed trajectory. A location point is considered as under stay status if the user spent more than Δ_t^l minutes but less than Δ_t^u minutes around this point within a distance of Δ_d meters. Comparing with the method proposed by Li et al[89], we set an additional upper bound Δ_t^u here (typically 12 hours). The reason is that in some sparse trajectories the time gap between two consecutive location points can exceed several days, which renders it difficult to infer the user's mobility status during the gap. For such cases, we regard the latter location point as the start point of a new segment. Furthermore, we stress that the pre-processing stage mainly effect on the non-sparse trajectories. For the extremely sparse trajectories, almost a single spatio-temporal point can represent a stay point. See Algorithm 1 for the pseudo-code.

Algorithm 2: Multi Resolution Filtering

```
   // first phase
1  Map(⟨T.id, S⟩)
2  for i = 1 ... N do
3      sz ← the cell size of G_i;
4      for sp ∈ S do
5          cx, cy ← the cell coordinate that sp lies in G_i;
6          c ← (cx, cy, sz);
7          emit(⟨c, ⟨T.id, sp.cnt⟩⟩);
8  Reduce(⟨c, list(⟨T.id, T.cnt⟩)⟩)
9  if len(list(⟨T.id, T.cnt⟩)) ≤ m_c then
10     l_A ← items of list(⟨T.id, T.cnt⟩) where T.id ∈ D_A;
11     l_B ← items of list(⟨T.id, T.cnt⟩) where T.id ∈ D_B;
12     for ⟨T_A.id, T_A.cnt⟩ ∈ l_A do
13         for ⟨T_B.id, T_B.cnt⟩ ∈ l_B do
14             o ← min{T_A.cnt, T_B.cnt} ; // co-occurrences
15             output(T_A.id, ⟨T_B.id, c, o⟩);
16
   // second phase
17 Map(⟨T_A.id, list(⟨T_B.id, c, o⟩)⟩)
18 for ⟨T_B.id, c, o⟩ ∈ list(⟨T_B.id, c, o⟩) do
19     sz ← the size of c;
20     Increase the ranking score of T_B.id by (r_sz · o);
21 I ← top Q trajectory ids with largest ranking scores;
22 for T_B.id ∈ I do
23     Merge the items of T_B.id into one key-value pair
         ⟨T_A.id, ⟨T_B.id, list(⟨c, o⟩)⟩⟩;
24     output(⟨T_A.id, ⟨T_B.id, list(⟨c, o⟩)⟩⟩);
```

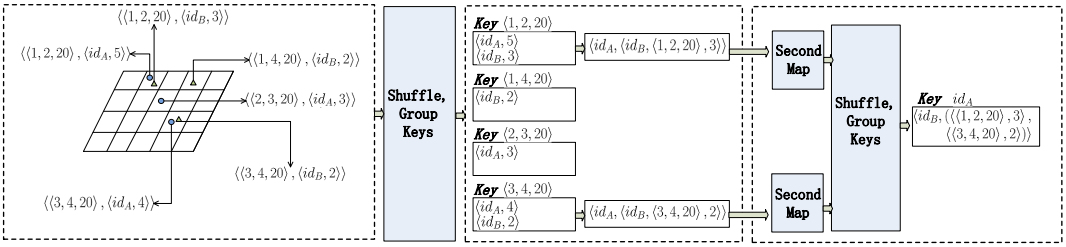


Figure 5.3　Running example of the multi-processing stage

5.4　Multi-Resolution Filtering

Since the pair-wise comparison is expensive, especially for the large data sets, in the multi-resolution filtering stage, for each $T_A \in D_A$, we only select a small subset

from D_B as its candidates. The multi-resolution filtering stage contains two phases. In the first phase, for each $T_A \in D_A$, we gather the "significant" co-occurrences with T_A. In the second phase, we evaluate the gathered co-occurrences and select a small subset of trajectories as the candidate set of T_A.

Formally, in the first phase, we partition the map into N grids with different granularities, i.e., the side length of cells in the grid. We use $G_1, \ldots, G_N$ to denote these grids. The map function takes the trajectory id $T.id$ as the key and the corresponding stay points S as the value. For each G_i, we enumerate the stay point $sp \in S$ and emit the key-value pairs $\langle c, \langle T.id, sp.cnt \rangle \rangle$, which indicates that $T.id$ occurred in a cell c for $sp.cnt$ times. Here c represents a tuple $\langle cx, cy, sz \rangle$ where $\langle cx, cy \rangle$ is the coordinate of the cell in the grid and sz is the corresponding side length. Note that since different grids are partitioned into different granularities, the tuple $\langle cx, cy, sz \rangle$ indicates a specific cell uniquely. The trajectory ids occurred in the same cell thus must be shuffled into the same reduce task.

In the reduce stage, each reduce function takes a key value pair $\langle c, list(\langle T.id, T.cnt \rangle) \rangle$ as input. If the length of $list(\langle T.id, T.cnt \rangle)$ exceeds m_c, we simply drop the cell c. Here m_c is a parameter to be specified (see Section 5.6 for details). Such cells usually correspond to the "common places" of the users such as the subway stations, which are not significant for the identification. Dropping such cells on one hand accelerates the algorithm. On the other hand, in Section 5.6, we show that it enhances the performance of our algorithm. For the rest of the cells, the reduce function splits $list(T.id)$ into two groups, the ids from D_A and the ids from D_B, denoting as $list(T_A.id)$ and $list(T_B.id)$ (to ensure the source of trajectory id is distinguishable, we add different special marks to the trajectory ids in different data sources.) Each pair $(T_A.id, T_B.id)$ from the two lists is a candidate pair. For each candidate pair, we emit a key-value pair $\langle T_A.id, \langle T_B.id, c, o \rangle \rangle$ which indicates that $T_A.id$ and $T_B.id$ co-occurred in the cell c for o times. Here the co-occurred frequency o is obtained by $\min\{T_A.cnt, T_B.cnt\}$.

In the second phase, the map function simply emits its input $\langle T_A.id\,, \langle T_B.id, c, o \rangle \rangle$. Thus, the key-value pairs with the same $T_A.id$ are shuffled into the same reduce task and each reduce task takes the key-value pair $\langle T_A.id, list\,(\langle T_B.id, c, o \rangle) \rangle$ as the input.

We evaluate each $T_B.id$ in $list(\langle T_B.id, c, o\rangle)$ with a ranking score. Formally, for each co-occurrence with $T_B.id$ in a cell with side length sz, we increase the ranking score of $T_B.id$ by a parameter r_{sz}. The finer granularity they co-occurred, the larger ranking score we increase. We select the top Q ids with the largest ranking scores as the candidate set of T_A. For each candidate $T_B.id$, we merge the related co-occurrences into one key-value pair $\langle T_A.id, \langle T_B.id, list(\langle c, o\rangle)\rangle\rangle$. See Algorithm 2 for the pseudo code.

We show a running example in Fig. 5.3. The grid in Fig. 5.3 is partitioned into small cells. The side length of each cell is 20 meters. The circle points are the stay points of user id_A and the triangle points are the stay points of user id_B. In the map stage of the first phase, for each cell we emit the users who occurred in this cell with corresponding frequency. By shuffling and grouping the keys, in the reduce stage we output the co-occurrences of id_A. Finally, in the second phase of the multi-resolution stage, we select the candidates of id_A and merge the related co-occurrences.

5.5 Verification Stage

In the verification stage, we evaluate each candidate of T_A and carefully select the matching trajectory. We first present the signal based similarity in Section 5.5.1 which is the foundation of the verification. Next, we present the algorithm of verification in Section 5.5.2.

5.5.1 Signal Based Similarity

Intuitively, the signal based similarity takes a pair of trajectories as input and observes the co-occurrences sequentially. Each co-occurrence is regarded as a "signal" which indicates that two trajectories might belong to the same user. The similarity is the final signal when the whole sequence is processed. The stronger the final signal is, the more similar the two trajectories are. We distinguish two kinds of signals, the *observed signal* and the *stimulus signal*. The observed signal is directly calculated by the co-occurrences in each cell. The strength of the observed signal increases as the frequency of co-occurrences increases. However, such co-occurrences may be affected by some

"kernel places". For example, in Figure 5.4, the two trajectories co-occurred frequently at the company, as a result, they are also frequently observed co-occurring at the nearby bus station. We refer to the company in this example as a kernel place. To capture such spatial correlation feature, we introduce the stimulus signal. We assume that there exist several *kernel cells* initially. Each kernel cell emits a positive stimulus signal. Each stimulus signal spreads out spatially from the kernel cell with an attenuation factor $\alpha < 1$. We consider the observed signal as the superposition of the decaying stimulus signals. The signal based similarity extracts the kernel cells and recovers the strengths of the stimulus signals from the observed signals. The final signal is calculated by considering both the strength of stimulus signals and the distances between the kernel cells.

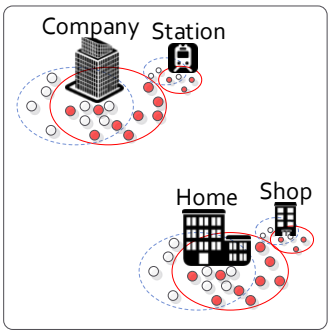


Figure 5.4　The company and the home represent two kernel places.

Formally, suppose we are given two trajectories $T_A \in D_A$ and $T_B \in D_B$. We consider each grid G_i separately. We use $(c_1, o_1), \ldots, (c_m, o_m)$ to denote the observed co-occurrences in G_i. Here c_k represents the k-th co-occurred cell and o_k represents its corresponding frequency. For simplicity, we do not require a specific order here. We first calculate the observed signal in each cell. Note that the frequency of co-occurrences has the following diminishing marginal utility property: when many co-occurrences at a specific cell have been observed, further co-occurrences at the same cell can only contribute limited information. To capture such property, we use a sigmoid function

$f_s(o_k; \eta, \gamma)$ to calculate the observed signal in c_k:

$$\mathsf{ob}(c_k) = f_s(o_k) = \frac{\eta}{1 + e^{-\gamma o_k}} - \frac{\eta}{2} \tag{5-1}$$

where η, γ are two parameters to be specified (see Section 5.6 for details).

To recover the stimulus signals efficiently, we simply process the observed signals sequentially to approximate the stimulus signals. For the cell c_k, we assume that it is only affected by the cells $c_1, \ldots, c_{k-1}$. Note that the stimulus signals we obtained may be different under different orders of cells. However, in the experiments, we find a specific order rarely affects the accuracy for user identification. Specifically, we use $\mathsf{st}(c_k)$ to denote the stimulus signal in cell c_k. Initially, we have that $\mathsf{st}(c_1) = \mathsf{ob}(c_1)$. For any $k > 1$, we have

$$\mathsf{st}(c_k) = \max \begin{cases} \mathsf{ob}(c_k) - \sum_{l<k} \mathsf{st}(c_l) \cdot \alpha^{\mathrm{Dis}_{\mathrm{grid}}(c_k, c_l)} \\ 0 \end{cases} . \tag{5-2}$$

where $\mathrm{Dis}_{\mathrm{grid}}$ is the distance metric defined on the grid. In our algorithm, we use Euclidean distance of cell centers scaled by the side length of the cell as $\mathrm{Dis}_{\mathrm{grid}}$. Thus, the kernel cells are the cells with non zero stimulus signal. We use K to denote the indices of kernel cells, i.e., $K = \{k : \mathsf{st}(c_k) > 0\}$.

Next, we take the distances between the kernel cells into consideration. We use $\mathsf{md}(c_k)$ to denote the minimum distance from cell c_k to the previous kernel cells, i.e.,

$$\mathsf{md}(c_k) = \min_{(l<k) \wedge (l \in K)} \mathrm{Dis}_{\mathrm{grid}}(c_k, c_l).$$

Similarly, we define a sigmoid function $f_d(\mathsf{md}(c_k); \lambda, \mu)$ which has the same form as f_s with parameters λ and μ to be specified. Then, the signal in the grid G_i is:

$$\mathsf{sig}_i = \mathsf{st}(c_1) + \sum_{k \in K \setminus \{1\}} \mathsf{st}(c_k) \cdot (1 + f_d(\mathsf{md}(c_k))) \tag{5-3}$$

The equation 5-3 essentially captures the feature that we showed in Fig. A.4(c), i.e., it is significant to observe the co-occurrences taking far apart from each other.

Algorithm 3: Signal Based Similarity for T_A and T_B

for $i = 1 \ldots N$ **do**
  $K \leftarrow \{\}$;
  for $k = 1 \ldots m$ **do**
    $o \leftarrow$ co-occurrences in cell $c_k \in C$;
    $\mathsf{ob}(c_k) = f_s(o)$;
    if $k == 1$ **then**
      $\mathsf{st}(c_1) \leftarrow \mathsf{ob}(c_1)$;
    else
      $\mathsf{st}(c_k) \leftarrow \max\{0, \mathsf{ob}(c_k) - \sum_{l \in K} \mathsf{st}(c_l) \cdot \alpha^{\mathrm{Dis}_{\mathrm{grid}}(c_k, c_l)}\}$;
    if $\mathsf{st}(c_k) > 0$ **then**
      K.add(k);
  $\mathsf{sig}_i = \mathsf{st}(c_1) + \sum_{k \in K \setminus \{1\}} \mathsf{st}(c_k) \cdot (1 + f_d(\mathsf{md}(c_k)))$;
$\mathsf{SIG} = \sum_{i=1}^{N} \beta_i \cdot \mathsf{sig}_i$;

Algorithm 4: Verification

$\mathsf{Map}(\langle T_A.id, \langle T_B.id, list(\langle c, o \rangle) \rangle \rangle)$
calculate the signal based similarity SIG;
calculate the weighted jaccard similarity WJS;
if $\mathsf{SIG} \geq \theta_{\mathsf{SIG}}$ *and* $\mathsf{WJS} \geq \theta_{\mathsf{WJS}}$ **then**
  $\mathrm{emit}(\langle T_A.id, \langle T_B.id, \mathsf{SIG}, \mathsf{WJS} \rangle \rangle)$;

$\mathsf{Reduce}(\langle T_A.id, list(\langle T_B.id, , \mathsf{SIG}, \mathsf{WJS} \rangle) \rangle)$
if $\exists T_B^*.id \in list(T_B.id)$ *dominate all the others* **then**
  $\mathrm{output}(\langle T_A.id, T_B^*.id \rangle)$;

Finally, we sum the signals of all the grids. We set a weight parameter β_i for the G_i. Again, the finer granularity G_i is, the larger weight parameter β_i becomes. Thus, the signal based similarity is defined as:

$$\mathsf{SIG} = \sum_i \beta_i \cdot \mathsf{sig}_i \tag{5-4}$$

See Algorithm 3 for the pseudo code.

5.5.2 Verification

In the verification stage, each map function takes a key-value pair $\langle T_A.id, \langle T_B.id, list(\langle c, o\rangle)\rangle\rangle$. Note that $list(\langle c, o\rangle)$ contains all the co-occurrences in all grids. Thus, we can directly calculate the signal based similarity from $list(\langle c, o\rangle)$.

To verify the candidates, a feasible way is to measure each candidate of T_A with the signal based similarity and select the most similar one as the matching trajectory. In the experiment section, we show that such strategy achieves a reasonable performance. However, many application scenarios are highly sensitive to misidentification, i.e. a few erroneous cases could cause serious consequence. It is necessary to adopt a rejection strategy, i.e., to refuse to identify the trajectories that may lead to mistakes. In Section 5.6, we show that it is difficult and inaccuracy to obtain a rejection strategy with only the signal based similarity. Motived by this, we adopt an effective rejection strategy with another similarity measure called the weighted Jaccard similarity which is applied in combination with the signal based similarity. The weighted Jaccard similarity was first proposed by Ioffe et al.[100]. It measures the similarity between two weighted set. For our problem, we regard each trajectory as a set of cells it has visited. The weight of each cell is the corresponding visiting frequency. Thus, for a given pair of trajectories, it measures the similarity of the places they visited and the corresponding frequencies. For a specific grid G_i , we use v_c^A (v_c^B resp.) to denote the frequency of T_A (T_B resp.) being observed at cell c. The similarity of T_A and T_B on grid G_i is defined as:

$$\mathsf{wjs}_i = \frac{\sum_c \min\{v_c^A, v_c^B\}}{\sum_c \max\{v_c^A, v_c^B\}}. \tag{5-5}$$

We show an example in Fig. 5.5. The points in the same shape (circle or triangle) indicate a specific id. We use the red color to indicate that the user occurred in this cell for 5 times. Similarly, we use yellow color to represent the frequency of 3 and the white color to represent the frequency of 1. Thus, the weighted Jaccard similarity in the left figure is $17/27 = 0.63$ and the weighted Jaccard similarity in the right figure is $17/49 = 0.35$.

Note that the term $\min\{v_c^A, v_c^B\}$ is exactly the frequency of co-occurrence of T_A and

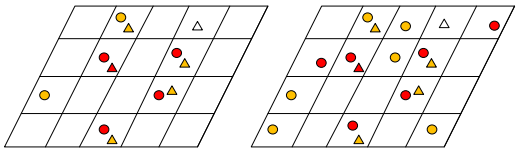


Figure 5.5 Example of the weighted Jaccard similarity.

T_B in cell c which is already contained in $list(\langle c, o\rangle)$, all we need to do is calculate the term $\max\{v_c^A, v_c^B\}$. Re-writing the denominator as $\sum_c v_c^A + \sum_c v_c^B - \sum \min\{v_c^A, v_c^B\}$, we find that the first two terms are exactly the accumulated frequency of all the stay points which can be obtained easily at the beginning. Thus, the weighted Jaccard similarity on G_i also can be calculated from $list(\langle c, o\rangle)$. Similarly, we use $\mathsf{WJS} = \sum_i \beta_i \cdot \mathsf{wjs}_i$ as the weighted Jaccard similarity of T_A and T_B.

We first calculate the signal based similarity and the weighted Jaccard similarity for all the candidates. Next, we set two *small* thresholds θ_{SIG} and θ_{WJS} to filter out the dissimilar candidates. If a candidate $T_B.id$ has its signal based similarity smaller than θ_{SIG} or its weighted Jaccard similarity smaller than θ_{WJS}, we filter it out from the candidate set. After the filtering, if there exists a unique maxima in the candidate set, we return it as the matching trajectory. Otherwise, we reject the identification. A candidate is called a maxima if there do not exist another candidate with both of the two similarities greater than it. For example, suppose we have 3 candidates with $(\mathsf{SIG}, \mathsf{WJS})$ equal to $(12.7, 0.4), (13.1, 0.3), (10, 0.35)$ respectively. Then the first two candidates are both maxima. See Algorithm 4 for the pseudo code.

We stress that when the mobility data sets are sampled in very different rates, the weighted Jaccard similarity becomes very low which leads to a great error. However, in the experiment section, we show that by combining with the signal based similarity and setting an extremely small threshold θ_{WJS}, AUI achieves a great performance.

5.6 Experiment Evaluation

In this section, we conduct extensive experiments on real mobility data sets to demonstrate the performance of the proposed algorithm. We first describe our experiment setting in Section 5.6.1 then we continue by presenting the effects of the param-

Table 5.1 Description of the data sets

Data set	#Trajectories	Average points per trajectory	Average points per trajectory after compression	Time span (Year-Month-Day)
CN-Sparse	6396	81.31	20.47	14-08-01 to 15-02-28
CN-Dense	31511	2303.12	196.65	14-08-01 to 15-02-28
CN-Part1	27019	1063.70	92.78	14-08-01 to 14-11-30
CN-Part2	30853	1441.28	117.65	14-12-01 to 15-02-28
CB-Sparse	888	101.11	29.09	15-03-01 to 15-05-31
CB-Dense	4323	1012.71	170.07	15-03-01 to 15-05-31
CB-Part1	3014	512.58	84.02	15-03-01 to 15-04-15
CB-Part2	3770	714.07	120.37	15-04-16 to 15-05-31
UT-Sparse	1992	101.61	27.06	15-03-01 to 15-05-31
UT-Dense	14115	776.04	166.52	15-03-01 to 15-05-31
UT-Part1	9695	374.66	79.53	15-03-01 to 15-04-15
UT-Part2	12497	544.49	116.10	15-04-16 to 15-05-31

eters in Section 5.6.2. We further compare our algorithm with the existing methods in Section 5.6.3. Finally, in Section 5.6.4 we give a discussion of our experimental results.

5.6.1 Experiment Setting

Our data set is collected from users who shared location data using different mobile apps of Baidu Inc. The data sources include the navigation data, map queries, geo-tagged records in social platforms etc. During our experiment, all the user id were anonymized by hashing. The trajectories of the same user have the consistent hashing id which we use it as the ground truth.

We distinguish two kinds of data sets, *the dense mobility data set* and *the sparse mobility data set*. The sources of dense mobility data sets contains the navigation data and the GPS location data. The sources of sparse mobility data sets contains map queries, geo-tagged check-in data etc. Thus, the trajectories in the dense mobility data

are sampled at a very high rates whereas the trajectories in the sparse mobility data are sampled at a low rates.

We randomly select 31511 users in China and extract the corresponding mobility data from the dense mobility data set, denoting as *CN-Dense*. The time span of CN-Dense is from August, 2014 to February 2015. Among these users, 6396 users can be found in the sparse mobility set. We extract the mobility data of these 6396 users from the sparse mobility set, denoting as *CN-Sparse*. For the first experiment, we use CN-Sparse and CN-Dense as D_A and D_B respectively as we described in Section 5.2.

Next, to evaluate our algorithm in the case that the trajectories are temporally disjoint, we split CN-Dense into two parts. The first part is from August 2014 to November 2014, denoting as *CN-Part1* and the second part is from December 2014 to February 2015, denoting as *CN-Part2*. Note that some trajectories only appeared during one of the time intervals. Thus, after splitting, the number of trajectories of each data set can be less than 31511. Furthermore, only part of the trajectories in D_A have the matching trajectories.

We further select 4323 company employees who work in Company B and 14115 college students who study in University T. We design another 4 experiments with the same method we used in our first two experiment settings.

To pre-process the data, we set $\Delta_d = 100\text{m}$ and $[\Delta_t^l, \Delta_t^u] = [0.5\text{h}, 12\text{h}]$ (Algorithm 1). Thus, a location point is considered as a stay point if the user stays around the point within 100 meters for more than half an hour but less than 12 hours. We compare the average number of location points each trajectory contains before and after the pre-processing. The details as shown in Table 5.1.

We stress that the first two experiments (CN-Sparse vs. CN-Dense and CN-Part1 vs. CN-Part2) form the easiest two cases, since these users are randomly chosen from a large population distributed on a vast spatial domain, which renders it easy to uniquely identify them even under coarse granularity. The last two experiments (UT-Sparse vs. UT-Dense and UT-Part1 vs. UT-Part2), form the hardest two cases since these students all study and live in the same campus and it is difficult to distinguish any of them from their schoolmates.

All the algorithms are implemented in Python and ran under streaming mode of Hadoop system (Release 2.6.0). All the experiments are conducted on a Hadoop Cluster with 10 nodes. Each node corresponds to a computer with a Intel Xeon E312 CPU of 2 cores (2.1GHz for each core) and a 8G memory.

5.6.2 Effects of Parameters

Before we show the experiment results, we first give two useful definitions.

Definition 8 (hitting rate)： Given a specific trajectory $T_A \in D_A$ and its candidate set, if the matching trajectory of T_A is contained in its candidate set, we say we "hit" T_A. Suppose there are H trajectories in D_A which is hit. Then the *hitting rate* of D_A is defined as $H/|D_A|$.

Definition 9 (coverage, accuracy)： Suppose we use a single similarity measure S to identify the users. For each $T_A \in D_A$, denote its candidate set as C. If

$$\max_{T_B.id \in C} \mathrm{S}(T_A, T_B) \leq \theta$$

where θ is a given threshold, then we refuse to identify T_A. Otherwise, we select the candidate with the largest similarity as the matching trajectory. Suppose there are Rej trajectories which we refuse to identify and there are Cor trajectories which are correctly identified. Then the *coverage* is defined as $1 - \frac{Rej}{|D_A|}$ and the *accuracy* is defined as $\frac{Cor}{|D_A| - Rej}$.

Note that Definition 9 defines the coverage of a single similarity measure. For AUI, since it utilizes a pair of similarity measures and selects the matching trajectory only if it is the unique maxima. The coverage of AUI is essentially associated with the "cohesion" of the data sets and can be determined automatically, i.e., the coverage would be much lower if users within the data sets are spatially correlated or exhibit a high degree of social homophily.

Our default parameter setting is presented in Table 5.2. To illustrate the effects of different parameters, each time *we only change one parameter and keep the others unchanged*.

Table 5.2 Default Parameter Setting

variable	value	source
N	2 (with side lengths = $\{20m, 200m\}$)	Algorithm 2
r_{20}, r_{200}	0.625, 0.375	Algorithm 2
m_c	2000	Algorithm 2
Q	20	Algorithm 2
η	16	Equ. (5-1)
γ	0.2	Equ. (5-1)
α	0.4	Equ. (5-2)
λ	50	Equ. (5-3)
μ	1/4000	Equ. (5-3)
β_1, β_2	0.8, 0.2	Equ (5-3)

5.6.2.1 Effects of Q

Recall that in the multi-resolution filtering stage, for each $T_A \in D_A$, we select a candidate set of size Q (Algorithm 2). If Q is too small, it is easy to miss the matching trajectories in their candidate set. However, the results show that only choosing $Q = 20$ is enough to achieve the hitting rates from 91.61% to 99.66%. A higher hitting rate can be obtained by choosing larger value of Q, which would increase the time cost of the verification stage at the same time, as shown in Table 5.3.

5.6.2.2 Effects of N

In the multi-resolution filtering stage, we partition the map into N grids with different granularities (Algorithm 2). To illustrate the effects of N, we use 3 experiments with different parameter settings. In Setting A, we only have one grid. The side length of each cell is 20m. In Setting B, we use the default setting as shown in Table 5.2. In Setting C, we partition the map into 4 grids. The side lengths equal to 20m, 50m, 100m, 200m respectively and we set the weight parameters β as $\{0.6, 0.4, 0.2, 0.1\}$ from the finest granularity to the coarsest granularity. The results are shown in Table 5.4. Each item in Table 5.4 corresponds to a pair *(coverage, accuracy)*.

From the experimental results, it is easy to see that with multiple granularities,

Table 5.3 Hitting rate of each experiment

Experiment	$Q = 3$	$Q = 20$	$Q = 50$
CN-Sparse vs. CN-Dense	96.17%	97.62%	97.62%
CN-Part1 vs. CN-Part2	98.0%	98.82%	98.74%
CB-Sparse vs. CB-Dense	90.50%	99.66%	99.66%
CB-Part1 vs. CB-Part2	83.14%	91.61%	93.50%
UT-Sparse vs. UT-Dense	85.14%	99.10%	99.10%
UT-Part1 vs. UT-Part2	78.01%	91.88%	94.22%

Table 5.4 Effects of N

Experiment	Setting A (20m)	Setting B (20m, 200m)	Setting C (20m, 50m, 100m, 200m)
CN-Sparse vs. CN-Dense	(28.92, 100.00)	(59.72, 99.94)	(92.01, 99.74)
CN-Part1 vs. CN-Part2	(59.46, 99.95)	(88.35, 99.80)	(92.01, 99.78)
CB-Sparse vs. CB-Dense	(52.70, 99.78)	(73.31, 97.39)	(81.98, 95.60)
CB-Part1 vs. CB-Part2	(57.84, 89.32)	(70.66, 91.36)	(70.22, 92.01)
UT-Sparse vs. UT-Dense	(56.32, 97.77)	(71.18, 90.20)	(62.60, 94.47)
UT-Part1 vs. UT-Part2	(49.88, 88.25)	(60.81, 90.09)	(58.40, 92.51)

our algorithm achieves a much better performance. The reason is that the finer granularity only captures the significant co-occurrences of two trajectories, such as the co-occurrences in the same building. However, due to the noise of the mobility data and the location error of GPS devices, it is hard to identify the users with such limited information. As we can see, in Setting A, the algorithm rejects the most of the identifications for all of the experiments. Especially, for the experiment CN-Sparse vs. CN-Dense, the coverage is only 28.92%. However, by utilizing multiple granularities with proper weight parameters, we successfully identify 92.01% of the trajectories with the accuracy of 99.74% for the experiment CN-Sparse vs. CN-Dense in Setting

Table 5.5 Effects of m_c

m_c	coverage (%)	accuracy (%)	time (s)
50	47.08	88.45	33
200	56.48	88.27	66
800	57.75	88.19	164
2000	60.81	90.09	521

C, which is much better than the other settings.

5.6.2.3 Effects of m_c

As we explained in Section 5.4, the cells with a large population usually correspond to some "common places". Eliminating such cells (more than m_c users being observed in this cell) on one hand accelerates the algorithm, on the other hand enhances the performance of the algorithm.

We evaluate the effects of m_c on the hardest experiment UT-Part1 vs. UT-Part2 (the other experiments are not sensitive to m_c since the trajectories are distributed on a vast spatial domain). We compare the coverage, the corresponding accuracy and the running time under different values of m_c. The results are shown in Table. 5.5.

From Table 5.5 we can see that, the coverage increases with the increase of m_c. If m_c is too small (e.g. $m_c = 50$ in our experiment), the algorithm drops the most of the cells and only retain the sparsely-populated cells. Of course, observing the co-occurrences in those sparsely-populated cells is very significant for user identification. However, the coverage in such case is extremely low. Furthermore, we find that the accuracy of each experiment does not vary much when m_c varies. Such property echoes that AUI determines the coverage automatically according to the cohesion of the data set.

It is notable that the complexity of the reduce stage in the first phase of the multi-resolution filtering is $O(\min\{m_c, |T_c|\}^2)$ where T_c is the trajectories ids occurred in the current cell. Thus, a large value of m_c leads to a high time complexity as well.

5.6.2.4 Effects of α

As we presented in Section 5.5.1, in the signal based similarity, each kernel cell emits a stimulus signal which spreads out spatially with an attenuation factor α. Since α is only related with the signal based similarity, to show the effects of α, we evaluate the algorithm performance by only utilizing the signal based similarity under the coverage equal to 90%. We set α to be 0 and $1 - 10^{-30}$ respectively. Such two values correspond to that the stimulus signal does not spread out at all and the stimulus signal does not decrease almost respectively. The results are shown in Table 5.7.

Such results essentially reflect the effects of the "kernel cells". When $\alpha = 0$, the cells are independent (we do not distinguish the kernel cells in such case). As we can see by taking the relations of the cells into consideration (setting a proper $\alpha > 0$), the accuracy increases a lot in the hard experiments (UT-Sparse vs. UT-Dense, UT-Part1 vs. UT-Part2).

5.6.2.5 Effects of Distance Values between Kernel Cells

As we mentioned in Section 5.5, the distance values between the kernel cells play an important role in the signal based similarity. To illustrate the effects of taking the distance into consideration, we evaluate our algorithm with 3 experiments with different settings. Recall that the signal based similarity in Equ.(5-3) is calculated by

$$\mathsf{sig}_i = \mathsf{st}(c_1) + \sum_{k \in K \setminus \{1\}} \mathsf{st}(c_k) \cdot (1 + f_d(\mathsf{md}(c_k))).$$

In Setting A, we set $f_d(x) = 0$, i.e., the signal based similarity is unrelated with the distance values between the cells. We use the default setting as in Table 5.2 as Setting B. In Setting C, we replace the sigmoid function with a linear function $f_d(x) = x$ to eliminate the diminishing marginal utility property. Again, we compare the accuracies under the coverage equal to 90%. The results are shown in Table 5.7

From Table 5.7, we can see that for the two easiest experiments, the distances between the cells do not effect on the accuracy much. However, for the hardest two

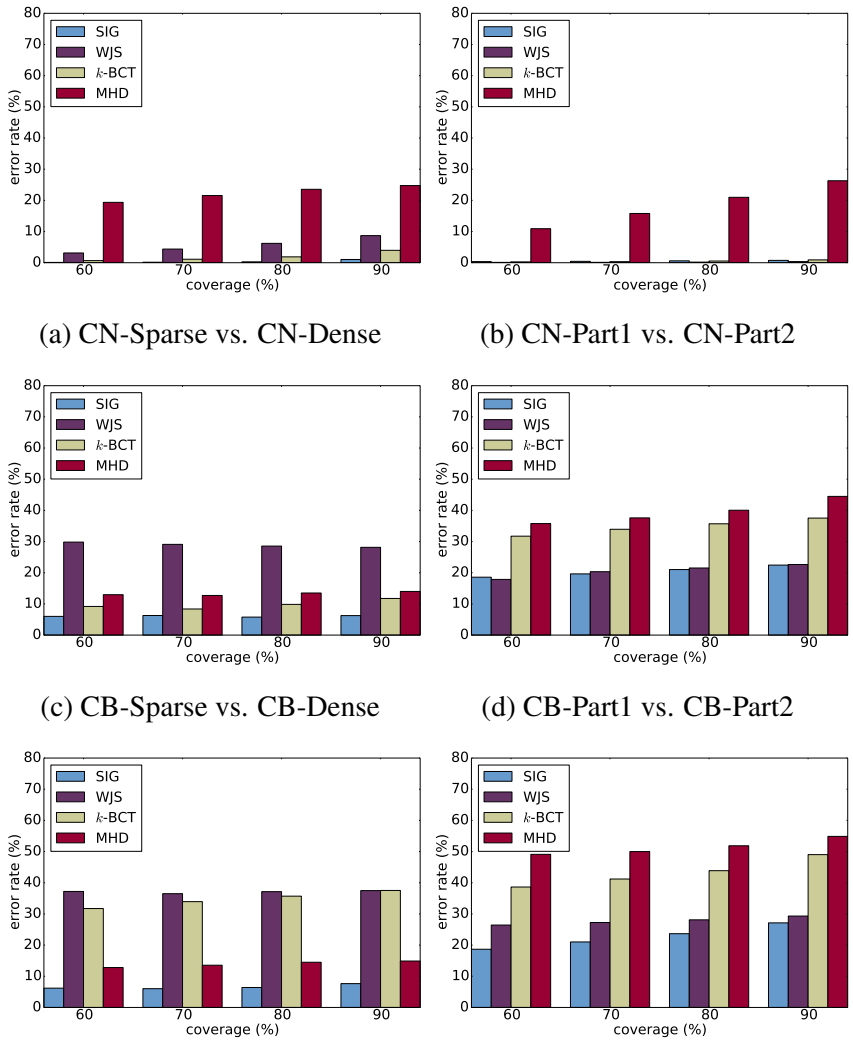
Table 5.6　Effects of α

Experiment	accuracy(%) $\alpha = 0$	accuracy(%) $\alpha = 0.4$	accuracy(%) $\alpha \approx 1$
CN-Sparse vs. CN-Dense	98.66	98.64	69.43
CN-Part1 vs. CN-Part2	99.58	99.59	98.68
CB-Sparse vs. CB-Dense	94.25	94.37	76.50
CB-Part1 vs. CB-Part2	79.63	80.08	73.00
UT-Sparse vs. UT-Dense	87.06	93.15	69.47
UT-Part1 vs. UT-Part2	69.18	72.34	47.50

Table 5.7　Effects of Distances

Experiment	Setting A	Setting B	Setting C
CN-Sparse vs. CN-Dense	98.32	98.64	98.52
CN-Part1 vs. CN-Part2	99.48	99.59	98.86
CB-Sparse vs. CB-Dense	87.23	94.37	86.36
CB-Part1 vs. CB-Part2	79.94	80.08	63.05
UT-Sparse vs. UT-Dense	76.00	93.15	86.14
UT-Part1 vs. UT-Part2	68.53	72.34	57.79

experiments, the accuracy increases dramatically when the distance values and the diminishing marginal utility property is taking into consideration.



(a) CN-Sparse vs. CN-Dense　(b) CN-Part1 vs. CN-Part2

(c) CB-Sparse vs. CB-Dense　(d) CB-Part1 vs. CB-Part2

(e) UT-Sparse vs. UT-Dense　(f) UT-Part1 vs. UT-Part2

Figure 5.6　Error rates of different similarity measures under the same coverage

5.6.3　Comparisons with Other Algorithms

In this section, we use the default setting as shown in Section 5.6.2 and compare our algorithm with the other existing methods.

We first compare the performance of the signal based similarity with the existing measures. As we mentioned in Section 5.1, since our trajectories we deal with are sampled at very different rate, the sparse trajectories may contain less than one point per day. Even the time span of the trajectories can be disjoint. Most prior work is

Table 5.8 Performance comparison for AUI

Experiment	AUI	SIG	WJS	k-BCT	MHD
CN-Sparse vs. CN-Dense (coverage = 59.72%)	99.94%	99.84%	99.08%	98.80%	78.13%
CN-Part1 vs. CN-Part2 (coverage = 88.35%)	99.80%	99.57%	99.41%	98.20%	70.39%
CB-Sparse vs. CB-Dense (coverage = 73.31%)	97.39%	97.53%	70.87%	91.59%	87.37%
CB-Part1 vs. CB-Part2 (coverage = 70.66%)	91.36%	81.67%	80.43%	65.93%	62.40%
UT-Sparse vs. UT-Dense (coverage = 72.84%)	95.63%	95.20%	76.02%	71.48%	87.32%
UT-Part1 vs. UT-Part2 (coverage = 60.81%)	90.09%	80.96%	74.97%	61.38%	50.89%

not available for our problem setting. Here we compare with two feasible existing similarity measures. ①

Adelfio et al.[101] extended Hausdorff distance and proposed the *modified Hausdorff distance* (MHD). For two trajectories T_A and T_B, MHD is defined as:

$$\mathrm{MHD}(T_A, T_B) = \frac{1}{|T_A|} \sum_{p_a \in T_A} \min_{p_b \in T_B} \mathrm{Dis}(p_a, p_b)$$

Chen et al.[97] present a similarity measure called *k Best Connected Trajectories* (k-BCT). Specifically, for two trajectories T_A and T_B, k-BCT is defined as:

$$k\text{-BCT}(T_A, T_B) = \sum_{p_a \in T_A} e^{-\min_{p_b \in T_B} \mathrm{Dis}(p_a, p_b)}.$$

To make a fair comparison, we adjust the threshold value for each measure and compare the error rates under the same coverage. The results are shown in Figure 5.6.

We then show the performance evaluation of AUI. We set the thresholds $\theta_{\mathsf{SIG}} = 12$ and an extremely small threshold $\theta_{\mathsf{WJS}} = 0.005$. In Table 5.8, we label the coverage of AUI and compare the performance for the other similarity measures under the same coverage.

① These methods are available but not designed for our setting.

5.6.4 Discussions

Based on the experiment results, we can easily see the advantages of the signal based similarity and AUI in our problem.

For all the 6 experiments, the signal based similarity outperforms other measures significantly. As we can see, the weighted Jaccard similarity performs well when the trajectories are sampled at similar rates (Figure 5.6(b), Figure 5.6(d), Figure 5.6(f)). However, it causes high error rate when the trajectories are sampled at very different rates (Figure 5.6(a), Figure 5.6(c), Figure 5.6(e)). The reason is that the weighted Jaccard similarity essentially measures the similarity of the spatio-temporal point sets. When the trajectories are sample at very different rates, the point sets of the same user can become totally distinct. On the other hand, MHD and k-BCT capture the geometrical distance features between the trajectories. These two measures are not too sensitive to the sample rates. Nevertheless, when the trajectories have significant overlaps, it is hard to identify the users by only using the distance features (Figure 5.6(d), Figure 5.6(e), Figure 5.6(f)). Furthermore, we stress that for the trajectories distributed on a vast spatial domain, there usually exist several points which are far from each other (e.g., the spatio-temporal points in city B and city C in Figure A.4(c)). For such case, the modified Hausdorff distance can be very large which is easy to lead to mistakes (Figure 5.6(a) and Figure 5.6(b)).

Despite the signal based similarity out performs the other measures for all the experiments. For real applications, it is hard to assign a proper threshold value or determine the coverage in advance. For example, by setting $\theta_{\mathrm{SIG}} = 0.95$, it is enough to achieve an accuracy of 97.90% under the coverage of 90% for the experiment CN-Sparse vs. CN-Dense. However, for the experiment UT-Part1 vs. UT-Part2, we need to set the threshold θ_{SIG} equal to 10.95 to achieve the same coverage and the accuracy is only 71.40%. AUI could determine the coverage according to the cohesion of the data set. As show in Table 4, for the easiest case (CN-Part 1 vs. CN-Part 2), AUI achieves the coverage of 88.35% and the corresponding accuracy of 99.80%. For the hardest cases (UT-Sparse vs. UT-Dense and UT-Part1 vs. UT-Part2), AUI covers only 72.84% and 60.81% of the trajectories but much higher accuracies (95.63% and 90.09%) than

the other measures under the same coverage.

5.7 Related Work

The studies of user identification focus on the concept of uniqueness in human mobility. It has been shown that people tend to visit places where they visited regularly in the past, which we refer to as "significant places"[9,99,102,103]. Zang and Bolot[103] showed that given each individual's top n significant places with highest visiting frequencies, we can uniquely identify a small subset of users from a very large-scale anonymized dataset. Montjoye et al.[99] showed that the human mobility retain highly unique even if we coarsen the location points. We stress that these work investigated the uniqueness or the user identification problem in a single mobility data set, i.e., given several historical location points which are already contained in the data set and retrieve the trajectories that match the given points. Rossi et al.[98] presented a technique for identifying users with previously unseen data, i.e., the location points that are not included in the original data set used for model training. We point out that in their method[98], the unseen data in their experiment is *sampled without replacement* from the original data set which differs from our problem. As we showed in Section 5.6, their method does not handle the user identification across heterogeneous data sources effectively. Furthermore, in a related work, Crandall et al.[9] used sparse geo-tags in social platforms to infer the social ties between different users. They showed that the strength of social ties is highly correlated with geographical co-occurrences of the users under various spatial granularities.

The user similarity search problem aims to retrieve similar user items from the entire database, based on the personal traits extracted from their past behaviors, in our case, their spatio-temporal mobility patterns. As we claimed in Section 5.1, user identification can be regarded as a special case of user similarity search. An essential ingredient in most methods is to define the similarity measure between a pair of user trajectories. Considerable amount of definitions have been proposed in the past. Most of them are extensions or variants of traditional methods, including Dynamic Time Warping (DTW)[94], Longest Common Subsequence (LCSS)[88], Edit Distance on Real

Penalty (ERP)[93], Edit Distance on Real Sequences (EDR)[92], etc. In addition, Li et at.[89] proposed a hierarchical graph based similarity measurement (HGSM) which takes into account both sequential and hierarchical property of geographic locations in user trajectories. These measures utilized the temporal closeness of trajectories, i.e., two trajectories are similar if they co-occurred in approximately the same place at approximately the time. Chen et al.[97] defined a trajectory similarity measure called *k Best Connected Trajectories* (*k*-BCT) based on the spatial distance and the order constraint in trajectory. Since *k*-BCT aims to search trajectories from a database using a small set of locations as queries, directly deploying it could lead to high error rate when the trajectories have a significant overlap as we showed in Section 5.6. We refer the interested readers to a recent survey[104] for more details about user similarity search. In the recent research[95], Ranu et al. studied the similarity measure of trajectories under inconsistent sampling rates. They formulated a robust distance function called *Edit Distance with Projections (EDwP)* to match trajectories under inconsistent and variable sampling rates. However, their method needs to infer the movement of users such as the speed which is not available in our problem.

There are several improved algorithms for user identification problem. Chen et al.[105] presented a novel formulation for user behavior representation, called STUL. In their recent work[106], they further proposed another kernel density estimation function, and an entropy-based weight scheme to calculate the identification confidence in each cell. Their methods achieved remarkable performances, especially for the location based social network data. We stress that these methods are following studies of our proposed model in this chapter.

5.8 Conclusion

In this chapter, we study the method of identifying users from heterogeneous data sources. In our problem, the trajectories in mobility data that we deal with are sampled at very different rates and extremely noisy. To address this challenge, we formulate the user identification problem over large scale heterogeneous mobility data sets and

present a MapReduce-based framework called *Automatic User Identification* (AUI). AUI is based on a novel similarity measure called the *signal based similarity*. The signal based similarity utilizes the co-occurrence representations of users' trajectories and calculates the similarity scores in multi-resolutions. We conduct extensive experiments and show that the signal based similarity significantly outperforms the existing similarity measures for user identification problem. Furthermore, we adopt a rejection strategy to reduce misidentification when the application scenarios are sensitive to error cases. The experimental results show that our rejection strategy can further improve the accuracy of our framework.

第 6 章 Conclusions and Future Work

6.1 Conclusions

The ubiquity of GPS-embedded devices such as the smart phones, loop sensors, has resulted in an explosion of spatio-temporal data. As we claimed in Chapter 1, mining the underneath wealth of spatio-temporal data is of great importance in various of industrial and commercial applications. However, in many real scenarios, such as traffic recommendation and location based social network, spatio-temporal data is usually massive, noisy and in very large scales. The underlying patterns in spatio-temporal are affected by complex spatial correlations, temporal dependencies, and many external factors. In order to achieve meaningful and accurate results in mining spatio-temporal data, it is important to extract effective feature representations from data. In this thesis, we study learning representations of spatio-temporal data. We present the challenges in learning spatio-temporal data representations and we use three concrete examples to illustrate how to learn the representations while addressing the corresponding challenges.

In Chapter 3, we study capturing complex correlations in spatio-temporal data representations. We use the travel time estimation problem as a concrete example. We propose an end-to-end framework based on deep neural networks called DeepTTE. Our model contains a geo-convolutional component which learns the spatial correlations in a fine granularity, a recurrent component which captures the temporal dependencies between different roads, and an attribute component which models the external factors. Such architecture automatically learns the feature representations of road segments data effectively. A multi-task learning component is further given on the top of DeepTTE which predicts the travel time by combining the collective estimation and individual estimation.

In Chapter 4, we study handling missing values while learning representations of

spatio-temporal data. We propose BRITS, a novel method to use recurrent dynamics to learn the representations of complex data correlations, and impute the missing values according to the learned representation. Our model does not impose assumptions over the data-generating process. Alternatively, we use a bi-directional recurrent neural network to learn the representations of captured correlations, and predict the missing values directly. Our model treats the missing values as variables of RNN graph. The imputation errors for missing values in both forward and backward directions can be fully backpropagated, which makes the imputation results more accurate. The missing value imputations are jointly trained with the classification/regression tasks. Experiment results show that our model demonstrates more accurate results for both imputation and classification/regression than state-of-the-art methods.

In Chapter 5, we further study the representation learning of extremely massive spatio-temporal data. Specifically, we consider the user identification problem as an example. In this problem, we consider the method of identifying users across heterogeneous data sources. The mobility datasets in this problem are very noisy and sampled at very different rates, which leads to extremely complex scenarios in identifying users. We propose the signal based similarity for measuring trajectory similarities in massive mobility datasets. In signal based similarity, we use the co-occurrence representations of users' trajectories, and we calculate the similarity scores based on such representations in multiple resolutions. We present a MapReduce-based framework called Automatic User Identification (AUI) which enables us to calculate signal based similarities at very large scales. We conduct extensive experiments. The experimental results show that our model significantly outperforms the existing methods.

The frameworks and models we used in this thesis provide a guidance in principle on learning representations of spatio-temporal data. Especially, the proposed methods in this thesis are able to effectively capture the complex correlations, impute the missing values and handle the massiveness in spatio-temporal data.

6.2 Future Directions

Learning representations of massive spatio-temporal data is very important yet challenging. It is still a hot topic in both academic and industrial areas. In this section, we present two future directions of spatio-temporal data representation learning.

First, for some extremely complex scenarios, learning effective feature representations still rely on manually designed rules. For example, for the user identification problem in Chapter 5, we design the signal based similarity in an ad-hoc way. In recent studies, some researchers used *graph embedding* to identify the users from mobility datasets with no hand-crafted features[107,108]. However, their methods apply to relatively regular mobility data with nearly the same sampling rates. To the best of our knowledge, there is still no effective method which solves user identification problem in very massive cases without hand-crafted features. In practice, learning feature representations from raw data with carefully devised deep architectures enables us to discover useful hierarchical features automatically. However, devising such architectures is highly non-trivial and the running time is less efficient for very large scale datasets. Alternatively, the hand-crafted features are usually robust and highly scalable but require much human-effort. The trade-off between two types of feature representations is an intriguing direction for future work.

Another interesting avenue is *knowledge transfer* for representation learning of spatio-temporal data. Many tasks in mining spatio-temporal data share similar patterns. For example, suppose that we have collected the congestion level of each road in a city in a past period. We aim to predict the congestion level of each road in the next few minutes. Such problem is highly related to the travel time estimation problem as we presented in Chapter 3. The travel time and congestion level are both affected by the spatial correlations and temporal dependencies of adjacent road segments. Thus, it helps better predict the congestion levels if we can transfer such knowledge in travel time estimation into the congestion level prediction problem. A related research topic is *transfer learning* where we focus on storing the knowledge learned from a source domain and applying it to a related target domain[109]. However, most of existing transfer learning methods focus on the vision tasks, such as the image classification,

object detection. Few work considers knowledge transfer for spatio-temporal data.

参考文献

[1] Krogh B, Andersen O, Lewis-Kelham E, et al. Trajectory based traffic analysis. Proceedings of Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2013. 536–539.

[2] Li P H, Yiu M L, Mouratidis K. Historical Traffic-Tolerant Paths in Road Networks. ACM Conference on Advances in Geographic Information Systems, 2014.

[3] Hefez I, Kanza Y, Levin R. Tarsius: A system for traffic-aware route search under conditions of uncertainty. Proceedings of Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2011. 517–520.

[4] Zhang J D, Chow C Y, Li Y. LORE: Exploiting sequential influence for location recommendations. Proceedings of Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2014. 103–112.

[5] Su H, Zheng K, Huang J, et al. Crowdplanner: A crowd-based route recommendation system. Proceedings of Data Engineering (ICDE), 2014 IEEE 30th International Conference on. IEEE, 2014. 1144–1155.

[6] Su H, Zheng K, Zeng K, et al. Making sense of trajectory data: A partition-and-summarization approach. Proceedings of Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE, 2015. 963–974.

[7] Dai J, Yang B, Guo C, et al. Personalized route recommendation using big trajectory data. Proceedings of Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE, 2015. 543–554.

[8] Spertus E, Sahami M, Buyukkokten O. Evaluating similarity measures: a large-scale study in the orkut social network. Proceedings of Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining. ACM, 2005. 678–684.

[9] Crandall D J, Backstrom L, Cosley D, et al. Inferring social ties from geographic coincidences. Proceedings of the National Academy of Sciences, 2010, 107(52):22436–22441.

[10] Bouros P, Sacharidis D, Bikakis N. Regionally influential users in location-aware social networks. Proceedings of Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2014. 501–504.

[11] Kanza Y, Kravi E, Motchan U. City nexus: discovering pairs of jointly-visited locations based on geo-tagged posts in social networks. Proceedings of Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2014. 597–600.

[12] Kang W, Tung A K, Zhao F, et al. Interactive hierarchical tag clouds for summarizing spatiotemporal social contents. Proceedings of Data Engineering (ICDE), 2014 IEEE 30th International Conference on. IEEE, 2014. 868–879.

[13] Jiang J, Lu H, Yang B, et al. Finding Top-k Local Users in Geo-Tagged Social Media Data. ICDE, 2015.

[14] Kreindler D M, Lumsden C J. The effects of the irregular sample and missing data in time series analysis. Nonlinear Dynamical Systems Analysis for the Behavioral Sciences Using Real Data, 2012. 135.

[15] Wang J, De Vries A P, Reinders M J. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. Proceedings of Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval. ACM, 2006. 501–508.

[16] Fung D S. Methods for the estimation of missing values in time series. 2006..

[17] Azur M J, Stuart E A, Frangakis C, et al. Multiple imputation by chained equations: what is it and how does it work? International journal of methods in psychiatric research, 2011, 20(1):40–49.

[18] Che Z, Purushotham S, Cho K, et al. Recurrent neural networks for multivariate time series with missing values. arXiv preprint arXiv:1606.01865, 2016..

[19] Yang B, Guo C, Jensen C S. Travel cost inference from sparse, spatio temporally correlated time series using markov models. Proceedings of the VLDB Endowment, 2013, 6(9):769–780.

[20] Wang J, Gu Q, Wu J, et al. Traffic Speed Prediction and Congestion Source Exploration: A Deep Learning Method. Proceedings of Data Mining (ICDM), 2016 IEEE 16th International Conference on. IEEE, 2016. 499–508.

[21] Pan B, Demiryurek U, Shahabi C. Utilizing real-world transportation data for accurate traffic prediction. Proceedings of Data Mining (ICDM), 2012 IEEE 12th International Conference on. IEEE, 2012. 595–604.

[22] Wang Y, Zheng Y, Xue Y. Travel time estimation of a path using sparse trajectories. Proceedings of Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2014. 25–34.

[23] Ansley C F, Kohn R. On the estimation of ARIMA models with missing values. Proceedings of Time series analysis of irregularly observed data. Springer, 1984: 9–37.

[24] Dong W, Junbo Z, Wei C, et al. When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks. AAAI, 2018.

[25] Cao W, Wang D, Li J, et al. BRITS: Bidirectional Recurrent Imputation for Time Series. arXiv preprint arXiv:1805.10572, 2018..

[26] Cao W, Wu Z, Wang D, et al. Automatic user identification method across heterogeneous mobility data sources. Proceedings of Data Engineering (ICDE), 2016 IEEE 32nd International Conference on. IEEE, 2016. 978–989.

[27] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT Press, 2016.

[28] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks, 2015, 61:85–117.

[29] Theano-Deep Learing Tutorial. http://deeplearning.net/tutorial/lenet.html.

[30] Kim Y. Convolutional neural networks for sentence classification. Proceedings of In EMNLP. Citeseer, 2014.

[31] Hochreiter S, Schmidhuber J. Long Short-Term Memory. Neural Computation, 1997, 9(8):1735.

[32] Cho K, Van Merriënboer B, Gulcehre C, et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078, 2014..

[33] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 51(1):107–113.

[34] Jenelius E, Koutsopoulos H N. Travel time estimation for urban road networks using low frequency probe vehicle data. Transportation Research Part B: Methodological, 2013, 53:64–81.

[35] Gal Y, Ghahramani Z. A theoretically grounded application of dropout in recurrent neural networks. Proceedings of Advances in neural information processing systems, 2016. 1019–1027.

[36] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014..

[37] Krizhevsky A, Sutskever I, Hinton G E. Imagenet classification with deep convolutional neural networks. Proceedings of Advances in neural information processing systems, 2012. 1097–1105.

[38] Zhang J, Zheng Y, Qi D, et al. DNN-Based Prediction Model for Spatio-Temporal Data. ACM SIGSPATIAL 2016, 2016.

[39] Graves A, Mohamed A r, Hinton G. Speech recognition with deep recurrent neural networks. Proceedings of Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. IEEE, 2013. 6645–6649.

[40] Yao S, Hu S, Zhao Y, et al. Deepsense: A unified deep learning framework for time-series mobile sensing data processing. Proceedings of Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 2017. 351–360.

[41] He K, Zhang X, Ren S, et al. Deep Residual Learning for Image Recognition. Computer Science, 2015..

[42] Wang H, Kuo Y H, Kifer D, et al. A simple baseline for travel time estimation using large-scale trip data. Proceedings of Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2016. 61.

[43] Clevert D A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus). arXiv preprint arXiv:1511.07289, 2015..

[44] Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks. Proceedings of Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011. 315–323.

[45] Kingma D, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014..

[46] Friedman J, Hastie T, Tibshirani R. The elements of statistical learning, volume 1. Springer series in statistics Springer, Berlin, 2001.

[47] Zhong S, Ghosh J. A new formulation of coupled hidden Markov models. Dept. Elect. Comput. Eng., Univ. Austin, Austin, TX, USA, 2001..

[48] Sevlian R, Rajagopal R. Travel Time Estimation Using Floating Car Data. arXiv preprint arXiv:1012.4249, 2010..

[49] Wang D, Cao W, Xu M, et al. ETCPS: An Effective and Scalable Traffic Condition Prediction System. Proceedings of International Conference on Database Systems for Advanced Applications. Springer, 2016. 419–436.

[50] Rahmani M, Jenelius E, Koutsopoulos H N. Route travel time estimation using low-frequency floating car data. Proceedings of Intelligent Transportation Systems-(ITSC), 2013 16th International IEEE Conference on. IEEE, 2013. 2292–2297.

[51] Yuan J, Zheng Y, Xie X, et al. T-drive: enhancing driving directions with taxi drivers' intelligence. Knowledge and Data Engineering, IEEE Transactions on, 2013, 25(1):220–232.

[52] Yang B, Dai J, Guo C, et al. PACE: a PAth-CEntric paradigm for stochastic path finding. The VLDB Journal, 2018, 27(2):153–178.

[53] Dai J, Yang B, Guo C, et al. Path cost distribution estimation using trajectory data. Proceedings of the VLDB Endowment, 2016, 10(3):85–96.

[54] Song X, Kanasugi H, Shibasaki R. Deeptransport: Prediction and simulation of human mobility and transportation mode at a citywide level. IJCAI, 2016.

[55] Zhang J, Zheng Y, Qi D. Deep Spatio-Temporal Residual Networks for Citywide Crowd Flows Prediction. AAAI, 2017.

[56] Dong W, Li J, Yao R, et al. Characterizing Driving Styles with Deep Learning. CoRR, 2016, abs/1607.03611.

[57] Bauer S, Schölkopf B, Peters J. The arrow of time in multivariate time series. Proceedings of International Conference on Machine Learning, 2016. 2043–2051.

[58] Batres-Estrada B. Deep learning for multivariate financial time series, 2015.

[59] Che Z, Purushotham S, Cho K, et al. Recurrent neural networks for multivariate time series with missing values. Scientific reports, 2018, 8(1):6085.

[60] Liu Z, Hauskrecht M. Learning linear dynamical systems from multivariate time series: A matrix factorization based framework. Proceedings of Proceedings of the 2016 SIAM International Conference on Data Mining. SIAM, 2016. 810–818.

[61] Xingjian S, Chen Z, Wang H, et al. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. Proceedings of Advances in neural information processing systems, 2015. 802–810.

[62] Rani S, Sikka G. Recent techniques of clustering of time series data: a survey. International Journal of Computer Applications, 2012, 52(15).

[63] Wang D, Cao W, Li J, et al. DeepSD: supply-demand prediction for online car-hailing services using deep neural networks. Proceedings of Data Engineering (ICDE), 2017 IEEE 33rd International Conference on. IEEE, 2017. 243–254.

[64] Yu H F, Rao N, Dhillon I S. Temporal regularized matrix factorization for high-dimensional time series prediction. Proceedings of Advances in neural information processing systems, 2016. 847–855.

[65] Brakel P, Stroobandt D, Schrauwen B. Training energy-based models for time-series imputation. The Journal of Machine Learning Research, 2013, 14(1):2771–2797.

[66] Ozaki T. 2 Non-linear time series models and dynamical systems. Handbook of statistics, 1985, 5:25–83.

[67] Basharat A, Shah M. Time series prediction by chaotic modeling of nonlinear dynamical systems. Proceedings of Computer Vision, 2009 IEEE 12th International Conference on. IEEE, 2009. 1941–1948.

[68] Pascanu R, Mikolov T, Bengio Y. On the difficulty of training recurrent neural networks. Proceedings of International Conference on Machine Learning, 2013. 1310–1318.

[69] Sussillo D, Barak O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. Neural computation, 2013, 25(3):626–649.

[70] Ian G, Yoshua B, Aaron C. Deep Learning. Book in preparation for MIT Press, 2016.

[71] Bengio S, Vinyals O, Jaitly N, et al. Scheduled sampling for sequence prediction with recurrent neural networks. Proceedings of Advances in Neural Information Processing Systems, 2015. 1171–1179.

[72] Yi X, Zheng Y, Zhang J, et al. ST-MVL: filling missing values in geo-sensory time series data. 2016..

[73] Silva I, Moody G, Scott D J, et al. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. Proceedings of Computing in Cardiology (CinC), 2012. IEEE, 2012. 245–248.

[74] Kaluža B, Mirchevska V, Dovgan E, et al. An agent-based approach to care in independent living. Proceedings of International joint conference on ambient intelligence. Springer, 2010. 177–186.

[75] Bradley A P. The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern recognition, 1997, 30(7):1145–1159.

[76] Moritz S, Bartz-Beielstein T. imputeTS: Time Series Missing Value Imputation in R. The R Journal, 2017, 9(1):207–218.

[77] Lipton Z C, Kale D, Wetzel R. Directly modeling missing data in sequences with RNNs: Improved classification of clinical time series. Proceedings of Machine Learning for Healthcare Conference, 2016. 253–270.

[78] Choi E, Bahadori M T, Schuetz A, et al. Doctor ai: Predicting clinical events via recurrent neural networks. Proceedings of Machine Learning for Healthcare Conference, 2016. 301–318.

[79] Yoon J, Zame W R, Schaar M. Multi-directional Recurrent Neural Networks: A Novel Method for Estimating Missing Data. 2017..

[80] Harvey A C. Forecasting, structural time series models and the Kalman filter. Cambridge university press, 1990.

[81] Li L, McCann J, Pollard N S, et al. Dynammo: Mining and summarization of coevolving sequences with missing values. Proceedings of Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2009. 507–516.

[82] Adomavicius G, Tuzhilin A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. Knowledge and Data Engineering, IEEE Transactions on, 2005, 17(6):734–749.

[83] Shang S, Ding R, Yuan B, et al. User oriented trajectory search for trip recommendation. Proceedings of Proceedings of the 15th International Conference on Extending Database Technology. ACM, 2012. 156–167.

[84] Lu E H C, Chen C Y, Tseng V S. Personalized trip recommendation with multiple constraints by mining user check-in behaviors. Proceedings of Proceedings of the 20th International Conference on Advances in Geographic Information Systems. ACM, 2012. 209–218.

[85] Chen Z, Shen H T, Zhou X. Discovering popular routes from trajectories. Proceedings of Data Engineering (ICDE), 2011 IEEE 27th International Conference on. IEEE, 2011. 900–911.

[86] Chen L, Cong G, Cao X, et al. Temporal spatial-keyword top-k publish/subscribe. Proceedings of Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE, 2015. 255–266.

[87] Zheng B, Yuan N J, Zheng K, et al. Approximate keyword search in semantic trajectory database. Proceedings of Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE, 2015. 975–986.

[88] Vlachos M, Kollios G, Gunopulos D. Discovering similar multidimensional trajectories. Proceedings of Data Engineering, 2002. Proceedings. 18th International Conference on. IEEE, 2002. 673–684.

[89] Li Q, Zheng Y, Xie X, et al. Mining user similarity based on location history. Proceedings of Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems. ACM, 2008. 34.

[90] Wang H, Liu K. User oriented trajectory similarity search. Proceedings of Proceedings of the ACM SIGKDD International Workshop on Urban Computing. ACM, 2012. 103–110.

[91] Ying J J C, Lu E H C, Lee W C, et al. Mining user similarity from semantic trajectories. Proceedings of Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks. ACM, 2010. 19–26.

[92] Chen L, Özsu M T, Oria V. Robust and fast similarity search for moving object trajectories. Proceedings of Proceedings of the 2005 ACM SIGMOD international conference on Management of data. ACM, 2005. 491–502.

[93] Chen L, Ng R. On the marriage of lp-norms and edit distance. Proceedings of Proceedings of the Thirtieth international conference on Very large data bases-Volume 30. VLDB Endowment, 2004. 792–803.

[94] Yi B K, Jagadish H, Faloutsos C. Efficient retrieval of similar time sequences under time warping. Proceedings of Data Engineering, 1998. Proceedings., 14th International Conference on. IEEE, 1998. 201–208.

[95] Ranu S, Deepak P, Telang A D, et al. Indexing and matching trajectories under inconsistent sampling rates. Proceedings of Data Engineering (ICDE), 2015 IEEE 31st International Conference on. IEEE, 2015. 999–1010.

[96] Frentzos E, Gratsias K, Theodoridis Y. Index-based most similar trajectory search. Proceedings of Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. IEEE, 2007. 816–825.

[97] Chen Z, Shen H T, Zhou X, et al. Searching trajectories by locations: an efficiency study. Proceedings of Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010. 255–266.

[98] Rossi L, Musolesi M. Spatio-temporal Techniques for User Identification by means of GPS Mobility Data. EPJ Data Science, 2015, 4:1–16.

[99] Montjoye Y A, Hidalgo C A, Verleysen M, et al. Unique in the Crowd: The privacy bounds of human mobility. Scientific reports, 2013, 3.

[100] Ioffe S. Improved consistent sampling, weighted minhash and l1 sketching. Proceedings of 10th International Conference on Data Mining. IEEE, 2010. 246–255.

[101] Adelfio M D, Nutanong S, Samet H. Similarity search on a large collection of point sets. Proceedings of Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. ACM, 2011. 132–141.

[102] Gonzalez M C, Hidalgo C A, Barabasi A L. Understanding individual human mobility patterns. Nature, 2008, 453(7196):779–782.

[103] Zang H, Bolot J. Anonymization of location data does not work: A large-scale measurement study. Proceedings of Proceedings of the 17th annual international conference on Mobile computing and networking. ACM, 2011. 145–156.

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

[105] Chen W, Yin H, Wang W, et al. Exploiting Spatio-Temporal User Behaviors for User Linkage. Proceedings of Proceedings of the 2017 ACM on Conference on Information and Knowledge Management. ACM, 2017. 517–526.

[106] Chen W, Yin H, Wang W, et al. Effective and Efficient User Account Linkage Across Location Based Social Networks. Proceedings of Data Engineering (ICDE), 2018 IEEE 33rd International Conference on.

[107] Pang J, Zhang Y. DeepCity: A feature learning framework for mining location check-ins. arXiv preprint arXiv:1610.03676, 2016..

[108] Backes M, Humbert M, Pang J, et al. walk2friends: Inferring Social Links from Mobility Profiles. Proceedings of Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2017. 1943–1957.

[109] Pan S J, Yang Q. A survey on transfer learning. IEEE Transactions on knowledge and data engineering, 2010, 22(10):1345–1359.

致　谢

五年前，对科研一无所知的自己，仅带着对清华的向往，开始了自己的博士生涯。许多个夜晚，我曾一边对着满屏的代码愁眉不展，一边幻想着自己毕业的那天。如今，却真的要离开朝夕相伴五年的校园了。

感谢我的导师李建教授五来年对我的帮助。能够成为他早期指导的学生之一，是我的荣幸。李老师在学术上的知识之广，功底之深，让我敬佩不已。许多次讨论中，他深入浅出的讲解让我受益匪浅。更重要的是，李老师让我明白了，理论对于一个计算机从业人员的重要性，以及无论多么复杂的理论，背后都有着简单而美丽的数学解释。也正因为坚信这两点，我才能够在许多对我而言看似困难的问题中，沉下心来一点一点解决。

感谢我的女友，亦是科研中的队友，王栋对我的陪伴和鼓励。读博期间，我们一起工作，熬夜，对着论文发呆，因为科研的进展而欢呼雀跃。每当我遇到生活学习中的困难时，她总是给予我最大的支持。这些时光，亦是我在校园中最珍贵的回忆。

感谢我的同学，金逸飞，房智轩，傅昊，我的师弟李志泽，施宇，还有我的师兄黄凌霄，刘宇，金恺等等。我在学术问题上常常和他们请教讨论，并收获了很多知识很灵感。

感谢我的父母和家人。他们总是非常开明，希望我能健康快乐，从来不让我背负太多的负担。而在我最需要帮助的时候，他们又总是无条件的支持我，充当我的后盾。他们伟大的爱与奉献，让我更加坚强。

最后，也感谢这五年来认识的许多前辈和朋友。当年百度的海山，浩文，滴滴的王征，边威，微软的张拯，边江，应策，今日头条的周浩，亦锬，李磊，以及许许多多帮助过我的人。

五年博士生涯，并没有太多让自己满意的成绩，总不免觉得遗憾。然而我相信这才仅仅是一个开始。真心感谢每一个支持我，鼓励我，帮助过我的人，之后的旅程中，希望我不会辜负你们对我的帮助。谢谢。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

附录 A 中文摘要

A.1 引言

A.1.1 研究背景

近年来，通信技术的不断发展和基于定位服务(LBS)的普及，为人们的日常生活带来极大的便利。举例来说，人们用具有定位功能的智能终端来查找目的地位置，规划路线，或者在社交平台上分享轨迹；研究机构利用传感器监控交通流量，或者进行车辆调度等。在这些场景中，大量的，杂乱的数据记录源源不断的产生出来。这些数据同时包含了时间信息和空间信息，因此我们把这种数据称为**时空数据** (spatio-temporal data)。对时空数据的挖掘为许多工业，商业应用带来非常大的潜在价值，例如交通分析[1–3]，旅行路线推荐[4–7]，以及基于位置的社交网络[8–13]。

典型的时空数据挖掘算法通常包含两个阶段。在第一个阶段，我们从原始数据中抽取出有用的信息，并把这类信息用一个固定长度的向量表示。在第二个阶段，基于前面抽取的特征，我们训练一系列机器学习模型（例如，逻辑回归，概率图模型），从而解决我们的问题。为了获得准确并有意义的结果，在第一阶段获得原始数据的有效特征表达(feature representations)尤为重要。然而，由于时空数据自身的特点，学习时空数据的有效表达往往面临诸多挑战：

- （**复杂的数据关联**） 时空数据通常包括非常复杂的数据关联。由于时空数据自身的特性，数据中通常同时存在着时间和空间上的关联性。例如，在一个交通系统中，一条道路的交通状况与该道路的历史交通状况相关，同时也与它相邻的道路的交通状况相关。另外，数据中的时空特征也受许多外部因素的影响，例如天气，节假日等。一个有效的特征表示应当能够捕捉数据中的复杂关联。
- （**粒度敏感性**） 时空数据对数据粒度较为敏感。由粗粒度造成的信息损失(information loss)可能会对模型的性能造成不可估量的负面效果。与视觉，自然语言处理等任务相比，在时空数据的表示学习中，充分保留细粒

度信息显得尤为重要。例如，在图片分类中，如果我们将原始图片换成相对粗糙的图片，我们依然有较高概率识别出图片所代表的类型（如，猫，狗）。然而，在城市交通中，如果我们搞混两条道路，即使这两条道路距离十分接近，他们的属性可能表现的非常不同（如，拥堵等级，交通流等）。

- **（缺失数据）** 时空数据通常非常不规则，并含有较高噪音。由于通信错误和不可预知的设备故障，数据中经常包含大量的缺失数据。现有的方法往往通过简单的插值或平滑对缺失数据进行简单补全[14–17]。由于时空数据包含复杂的数据关联，简单的平滑方法往往不能获得准确的补全结果[18]。
- **（数据环境嘈杂）** 在许多工业应用中，时空数据通常非常杂乱并且规模巨大。例如，Google的地图服务在每个月有高达百万的活跃用户。不同用户之间的使用特征互不相同。我们在设计表示学习算法时，应当同时考虑算法对于海量规模下的杂乱数据的鲁棒性。

在这篇论文中，我们针对以上挑战，研究有效的时空数据的表示学习算法。我们介绍三个具体的问题来阐述如何应对我们所提出的挑战。在接下的三个小节中，我们具体描述三个问题，并总结我们的贡献。

A.1.2 捕获复杂数据依赖

在第A.2节，我们研究如何在数据表征中捕获复杂的依赖关系。同时我们也考虑了时空数据对数据粒度敏感的问题。我们以交通到达时间预测问题为例。给定一个特定的路径，以一系列互相连接的路段表示，一个对应的出发时间，以及诸多外部因素（例如天气状况，驾驶习惯等），我们的目标是估计司机穿过给定路径的时间。在这个问题中，出行时间受到多种因素的影响，包括地理依赖，时间依赖，以及外部环境因素。之前的研究主要关注预测单个路段，或者子路段的出行时间，并将预测的单个路段加和。这种方法没有考虑不同路段时间的关联，从而导致了不精确的结果。贡献：我们提出了一个基于深度学习的端到端的模型，叫做DeepTTE。DeepTTE直接估计整条路径的到达时间。我们在经典的卷积操作上，加入了地理信息，从而提出了一种基于地理位置的卷积操作，使得数据中的细粒度信息得以保留。我们在地理卷积操作上叠加了一

层递归神经网络层，来进一步捕获时间上的依赖。通过该结构，我们能够直接从原始数据中有效地学习出路径数据的特征表示。在模型顶端，我们加入了一个多任务学习（multi-task learning）层。该层同时估计每个子路段的通行时间和整个路段的通行时间。我们在两个现实数据集上进行了大量实验，结果表明我们的模型明显超过现在已有的模型。

A.1.3 补全缺失数据

在第A.3节中，我们研究如何应对在表征学习中遇到的数据缺失问题。给定多个相互关联的时间序列，我们的目标是预测时间序列的标签，同时对缺失数据进行补全。现有的缺失补全的方法依赖于对数据分布的强假设，例如时间序列所在空间为线性动态系统[14,16,17,23]。然而，由于时空数据经常包含非常复杂的数据关联，这种假设通常难以成立。

贡献: 在这篇论文中，我们提出BRITS （Bi-directional Recurrent Imputation Time Series），一种关于缺失数据补全的新颖的方法。我们提出的方法不对数据的产生过程进行任何假设。取而代之，我们用一个双向递归神经网络来同时捕捉时间和空间的数据关联，并将捕获到的关联用一系列隐状态(hidden state)表示。我们基于这些隐状态直接对缺失数据进行预测。我们在三个真实数据集上对我们的方法进行评估。实验表明我们的方法比现有方法，在分类任务和补全任务上，均有显著提高。

A.1.4 处理嘈杂数据环境

在第A.4节中，我们进一步研究极端嘈杂数据环境下的表示学习。我们以用户身份识别问题为例。在这个问题中，我们研究在复杂多源(heterogeneous)轨迹数据中识别出同一用户。给定两个从不同数据源头（例如带地理标签的签到数据，导航轨迹等）生成的大规模的轨迹数据集，我们的目标是在不同的轨迹数据集中找到一对由相同用户生成的轨迹。这个问题中我们面临一系列及其复杂的情况。例如，(a) 在不同数据源中，不同轨迹的采样频率差距非常大；(b) 对一些稀疏轨迹而言，推测轨迹中的任何移动信息都是非常困难的；(c) 对一起居住和工作的人们，他们的轨迹往往有非常多的重合部分，从而使得我们非常容易误识别为同一用户。还有诸多其他的复杂情况，这里无法一一列举。

贡献：我们提出一种基于Map-Reduce的框架叫做*Automatic User Identification* (AUI)。该框架非常容易部署并且可以轻易扩展到非常大规模的轨迹数据集中。AUI基于一种全新的轨迹相似性度量方法，叫做*signal based similarity* (SIG)。这种度量方法能够对采样频率非常不同，并且包含大量噪音的轨迹对计算相似性。

在SIG中，我们把用户的轨迹表示为一系列的相遇事件，并根据这些相遇事件在多维粒度上分别计算轨迹的相似性。这种相似性度量在大规模的杂乱轨迹数据上表示出非常强的鲁棒性。我们进行了许多实验评测。实验结果表明我们的方法比现有方法显示出更好的准确性和鲁棒性。

A.1.5 论文组织

论文剩下的章节组织如下：在第A.2节中，我们描述了如何在数据表征中捕获复杂依赖。我们具体描述了我们在预测交通到达时间中提出的DeepTTE算法。这一节内容基于我们之前工作[24]。在第A.3节中，我们描述如何用BRITS算法解决时空数据中，数据缺失的问题。这一节的内容基于我们之前的工作[25]。在第A.4节中，我们研究极端嘈杂数据环境下的表示学习问题。我们描述我们的AUI框架来高效的解决嘈杂轨迹数据中的用户识别问题。这一节基于我们之前的工作[26]。

A.2 时间预估问题中的复杂数据依赖

预测城市中任意路径（由一系列连接的子路段表示）的行程时间，对于交通监控，路线规划，出租车/优步调度等都非常重要。然而，这是一个非常具有挑战性的问题，受到了各种复杂因素的影响，包括空间相关性，时间依赖性，外部条件（如天气，交通灯）。为了有效地从输入数据中提取特征表示，我们必须仔细考虑这种相关性。之前的工作通常侧重于估计单个路段或子路径的行驶时间，然后将这些时间加和，这会导致估计不准确。因为这些方法没有考虑道路道路交叉点/交通信号灯，并且局部错误可能会累积。为了解决这些问题，我们提出了一个针对交通到达时间预测的，端到端的模型，叫做DeepTTE。我们的模型可以高效的对整体路段进行准确估计。

A.2.1 引言

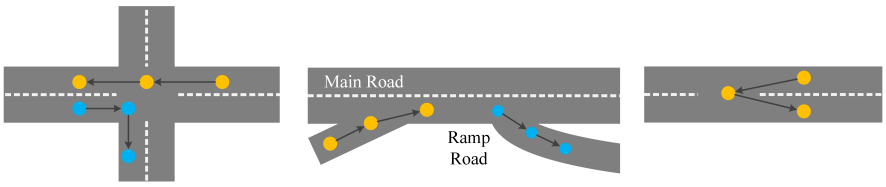
给定一个由一系列相互连接的子路段构成的路径，预测该路径的通行时间，是道路规划，导航，交通调度等应用中的基础问题。当用户搜索备选路径时，一个准确的到达时间预估能够帮助人们更好的规划路径并且规避掉拥堵的路径，从而进一步减少交通拥堵。尽管交通到达时间问题在过去被广泛研究，一个准确的到达时间估测算法依然是一个具有挑战性的问题，并受到多种因素的影响：

1) 分段预测 *vs.* 整体预测: 我们可以通过两种方法对到达时间进行预测：a) 分段预测首先将道路划分为一些子路段，并且对这些子路段分别进行预测，再将预测结果进行加和从而得到总的出行时间。b) 整体预测直接对整条路径的出行时间进行预测。尽管分段预测[19–22]对每一段子路径的出行时间可以准确地估计，这种方法无法刻画整体道路中的复杂路况，包括道路交叉，交通信号灯，道路拐弯处等。除此之外，对于子路段的预测误差可能累加从而造成不准确的结果。整体预测方法 (例如 Jenelius等人的工作[34])能够较好的捕获上文提到的交通路况。但是，随着待预测的路径长度的增加，历史数据中与该路径重合的数据会急剧减少，从而降低了我们预测的置信度。在许多情况下，历史数据中甚至没有数据与待预测道路吻合。

2) 多样的复杂因素: 城市交通受到空间关联，时间依赖，以及多种外部因素的影响。空间关联往往非常复杂多样化，如图 A.1。给定三个轨迹点，图 A.1中呈现出了几种不同的驾驶情况，分别代表直走，右转，调头，并入主路。显式抽取这类特征非常耗时，甚至在情况更为复杂的情况下仅依赖人工抽取几乎是不可能的。因此，我们需要采用隐式的方法。另外，空间关联随着时间变化会不停改变。例如图A.1 (b)中，在晚高峰时间，主路车流量较大，当司机从支路并入主路时会变得非常慢，从主路拐向支路的速度非常快。但是在非高峰时期，显然并入主路可以花费更少时间。而且，交通状况受到多种外部因素的影响，例如天气，驾驶习惯，星期等等。

为了解决上述挑战，在这一章里，我们提出一个端到端的框架叫做DeepTTE。我们这里列举我们工作的主要贡献：

- 我们提出一个时空模块来同时学习原始数据中的时间依赖与空间依赖。详细来说，我们的时空模块包含两个部分：a) 基于地理信息的卷积层：该层



(a) Go straight (yellow) or turn right (blue)　(b) Drive into main road (yellow) or ramp road (blue)　(c) Turn around

图 A.1 多种不同路径情况。

将原始的GPS序列转换为一系列特征图 (feature map)，从而能够从原始序列中隐式的捕捉局部地理关联（像图A.1中的情况一样）；b) 递归神经网络学习之前获得的特征图之间的时间关联。时空模块直接从GPS数据中高效的学习出其有效特征表示。

- 我们提出一个多任务学习模块，通过引入一个多任务损失函数，以同时预测整条道路的通行时间以及每一部分子路段的通行时间，从而能够在分段预测与整体预测两者之间找到一个较好的折中点。为了准确的估计整段道路的通行时间，我们还设计了一种多因素注意力机制来学习不同子路段对应隐向量的权重。
- 我们提出了一个基础属性模块，来整合多种外部因素，包括天气状况，星期，轨迹距离，驾驶习惯等等。我们将学习到的外部因素的特征表示传递给模型的其余部分，以用来提高模型整体的预测效果。
- 我们在两个大规模真实数据集上进行了大量实验，包括成都出租车的GPS定位数据和北京出租车的GPS定位数据。两个数据集上的相对误差分别为11.89%和10.92%。相比已有方法，我们的方法显示出更准确的效果。

A.2.2 公式化定义与模型架构

在本节中，我们首先对我们的问题进行形式化定义。

定义 1 (历史轨迹)： 我们定义一个历史轨迹 T为一个连续的历史GPS点的序列，即$T = \{p_1, \ldots, p_{|T|}\}$,[①]。每个GPS点$p_i$包含纬度($p_i.lat$)，经度($p_i.lng$)和

① 通常GPS设备每隔固定时间会产生一个轨迹。这可能会导致我们的模型学习到不合理的特征（例如，简单地统计GPS记录的数量）。为了避免这种情况，我们重新采样每个历史轨迹，使得两个连续点之

时间戳($p_i.ts$)。此外，对于每个轨迹，我们记录其外部因素，例如开始时间 (timeID)，星期几 (weekID)，天气状况（weather）和相应的司机标识 (driverID)。

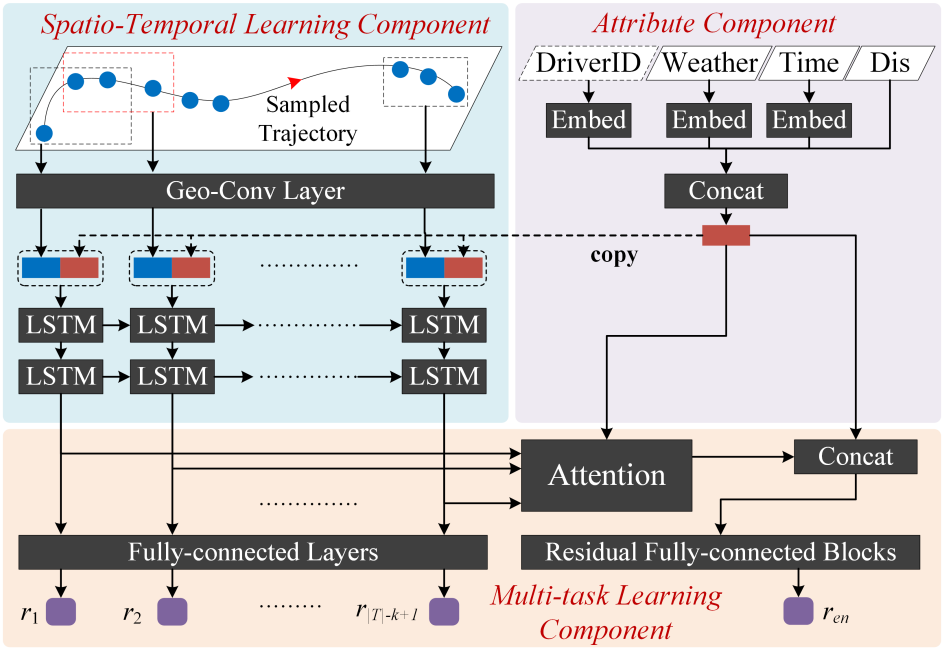


图 A.2 DeepTTE的架构。 Dis: distance; concat: concatenate.

定义 2 (问题目标)： 在训练阶段，我们将学习如何从定义1所定义的历史轨迹中，提取时空模式，从而在给定的路径和相应的外部因素下估计行驶时间。在测试阶段，给定路径P，我们的目标是通过P和相应的外部因素来估计从起点到终点的旅行时间。我们假设旅行路径P由用户指定，或由路线规划应用程序生成。

在测试阶段，为了使测试数据与训练数据一致，我们将路径P转换为具有相等距离差的一系列位置点。每个位置都表示为一对经度和纬度坐标。

备注： 在我们的实验中，为了生成测试数据，我们移除了历史轨迹中的时间戳，并将每个轨迹重新采样为具有相等距离间隙的GPS位置序列。在本章中，我们不考虑如何优化路径P。

间的距离差距大约为200至400米。

接下来，我们将描述我们提出的DeepTTE的模型结构，如图A.2所示。DeepTTE由三个组件组成：基础属性模块，时空模块和多任务学习模块。基础属性模块用于处理外部因素（例如天气）和给定路径的基本信息（例如开始时间）。其输出作为其他模块输入的一部分。时空模块是从原始GPS位置序列中，学习空间相关性和时间相关性的主要模块。最后，多任务学习组件基于前两个模块，估计给定路径的行程时间，从而能够平衡分段估计和整体估计两种估计方法。

A.2.3 相关工作

目前已有大量关于旅行时间估计的文献，我们只提几个与我们工作密切相关的工作。

A.2.3.1 分段预测方法

交通出行时间预测问题在之前已经被广泛研究[21,47,48]。然而，这些方法只估计了单个路段的行程时间，而没有考虑道路之间的相关性。Yang等人[19]使用时空隐马尔可夫模型来形式化相邻道路之间的关系。Wang等人[49]基于交通状况时间序列，观察到两个有用的相关性特征，从而构建模型改进了预测方法。Wang等人[20]提出了一种误差反馈循环卷积神经网络，称为eRCNN，用于估算每条单独道路上的交通速度。这些研究考虑了不同道路之间的相关性。不过，他们专注于精确估计单个路段的行程时间或速度。正如我们提到的，路径的行程时间会受到各种因素的影响，例如道路交叉口的数量和路径中的交通灯。简单地汇总路径中路段的行程时间将不会导致准确的结果[34]。

A.2.3.2 整体预测方法

Rahmani等人[50] 基于历史数据估计整条路径的旅行时间。但是，基于历史平均值的模型可能导致结果准确性较差。而且，由于新的查询路径可能不包括在历史数据中，所以这种方法还面临数据稀疏问题。Yuan等人[51]根据出租车的历史轨迹建立了一个地标图，每个地标都代表一条路。他们根据地标图估计路径的旅行时间分布。然而，由于地标是从k频繁穿越的道路中选择的，所以旅行记录较少的道路无法被准确估计。此外，Wang等人[22]根据历史数据中的子轨

迹，估计路径的行程时间。他们使用张量分解来补全稀疏的子轨迹，并且这种方法有效地提高了准确性。尽管如此，它仍然存在数据稀疏问题，因为很少有司机访问过很多子轨迹。

A.2.3.3 基于深度学习的预测方法

最近，深度学习技术展现了时空数据挖掘问题的优势。 Song等人[54]建立了一个名为DeepTransport的智能系统，用于模拟全市范围内的人员流动和交通模式。 Zhang等人[55]提出了一个预测人群流动的深度时空残差网络。Dong等人[56]研究了通过叠加递归神经网络，描述不同驾驶员的驾驶风格。

A.3 基于循环动力系统的缺失补全

在本节中，我们研究如何在学习时空数据特征表示的同时，处理缺失值。给定多个（空间）相关的时间序列数据，如何学习他们的特征表示以预测其类别标签，并补全缺失值？现有的补全方法通常对基础数据生成过程强加假设，如状态空间中的线性性。在本章中，我们提出了一种用于时间序列数据缺失值补全的新方法BRITS 。我们提出的方法有两个优点：(a) 它可以处理时间序列中的多个相关缺失值; (b)它可以将时间序列的补全模型拓展到非线性动力系统。我们在三个真实世界的数据集上评估我们的模型。实验表明，我们的模型在补全和分类/回归精度方面优于已有的方法。

A.3.1 引言

多元时间序列数据在许多领域都有很丰富的应用，如金融市场建模[57,58]，医疗保健[18,60]，气象学[61,62]，和交通工程[55,63]。时间序列被广泛用作相应领域应用中的分类/回归信号。但是，由于意外事故（如设备损坏或通信错误），时间序列中经常包含缺失值，进而降低了下游应用的性能。

以前的许多工作都提出用统计学和机器学习方法来解决上述问题，然而这总是需要对缺失值进行相当强的假设。我们也可以使用古典统计时间序列模型（如ARMA或ARIMA）（例如Ansley等人的工作[23]）来填充缺失值。但是这些模型在经过差分后本质上是线性模型。 Kreindler等人[14]假设数据是平滑的，

即在一定周期内数据没有突然的波动，因此通过平滑附近值可以实现补全缺失值。矩阵补全也用于解决缺失值问题[15]。但它通常只适用于静态数据，并需要较强的假设，如低秩性。我们还可以通过拟合一个参数化数据生成模型来预测缺失值[16,17]，该模型假定时间序列数据遵循假设模型的分布。这些假设使得相应的补全算法不能够很好泛化，当假设不成立时，算法性能较不理想。在本章中，我们提出BRITS ，一种填补高维时间序列缺失值的新方法。在内部，BRITS 采用了递归神经网络（RNN）[31,32]来同时补全缺失值并学习输入数据的特征表示，并且BRITS 对数据没有任何特定的假设。

以前的许多工作都使用非线性动力系统来进行时间序列预测[65–67]。在我们的例子中，我们将动力系统实例化为双向RNN，即用双向循环动力系统补全缺失值。特别的，我们做出以下技术贡献：

- 我们设计了一个新的双向RNN模型来补全缺失值。我们利用RNN来学习数据的特征表示和预测缺失值，而非类似在Che等人的工作[18]中调整平滑权重。我们的方法没有强加特定的假设，因此比以前的算法更容易泛化。
- 使用双向循环动力系统，我们可以从两个方向获得的错误反馈，并进一步推断缺失值。具体而言，我们将缺失值视为双向RNN计算图中的变量，并将其加入反向传播过程。在这种情况下，缺失值在前后方向上都会得到延迟梯度，并通过加入一致性约束条件，使得缺失值的估计更加准确。
- 我们在一个神经网络中同时进行缺失值补全和应用的分类/回归预测。这在一定程度上缓解了从缺失补全到表示学习之间的错误传播问题。另外，分类/回归的监督使缺失值的估计更准确。
- 我们在一个人工数据集和两个真实世界的数据集（空气质量数据和医疗数据）上评估我们的模型。实验结果表明，我们的模型在补全和分类/回归精度方面都优于现有的模型。

A.3.2 公式化定义

定义 3 (多元时间序列)： 我们将具有T个观测值的多元时间序列记做$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$。其中第$t$个观测值$\mathbf{x}_t \in \mathbb{R}^D$包含$D$维特征$\{x_t^1, x_t^2, \ldots, x_t^D\}$，并且在时刻$s_t$被观测记录（为了方便，我们假设$s_1 = 0$）。事实上，由于意外的事故，例如设备损坏或通信错误，$\mathbf{x}_t$可能会有缺失的值（例如，在图 A.3中，$x_1^3 \in \mathbf{x}_1$是

缺失值）。为了表示$\mathbf{x}_t$中的缺失值，我们引入一个掩码向量$\mathbf{m}_t$，其中，

$$\mathbf{m}_t^d = \begin{cases} 0 & if\ x_t^d\ is\ not\ observed \\ 1 & otherwise \end{cases}.$$

在很多情况下，缺失数据可能出现在多个连续的位置（例如，图A.3中的蓝色部分）。我们将δ_t^d定义为最后一次观测时间到当前时间s_t的时间间隔，

$$\delta_t^d = \begin{cases} s_t - s_{t-1} + \delta_{t-1}^d & if\ t > 1, \mathbf{m}_{t-1}^d = 0 \\ s_t - s_{t-1} & if\ t > 1, \mathbf{m}_{t-1}^d = 1 \\ 0 & if\ t = 1 \end{cases}.$$

请参见图A.3

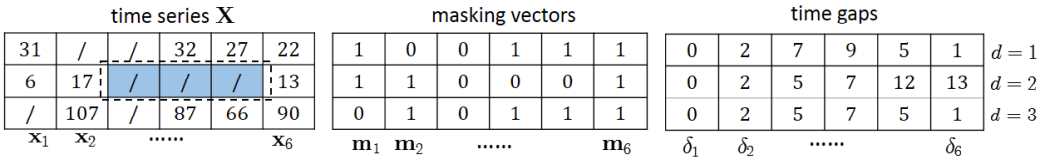


图 A.3 一个带有缺失数据的多元时间序列的例子。$\mathbf{x}_1$到$\mathbf{x}_6$分别在$s_{1...6} = 0, 2, 7, 9, 14, 15$时刻观测到。考虑$\mathbf{x}_6$中第2维特征，其最后一次观测发生在$s_2 = 2$，因此我们有$\delta_6^2 = s_6 - s_2 = 13$。

A.3.3 相关工作

关于时间序列中缺失补全有大量文献，我们只提几个密切相关的。插值方法试图为观测值拟合一条平滑曲线，从而通过局部插值[14,16]来重建缺失值。这种方法没有考虑变量之间随时间变化的关系。包括ARIMA，SARIMA等在内的自回归方法，消除了时间序列数据中的非平稳部分，并对序列中的平稳部分进行参数拟合。状态空间模型进一步结合了ARIMA和卡尔曼滤波器[16]，从而提供了更准确的结果。这些方法通常适用于单变量时间序列插值。对于多变量时间序列估算，多元链式方程插值法（MICE）[17]首先任意初始化缺失值，然后通过链式方程估计每个缺失变量。概率图模型[81]为每个缺失值引入一个潜在变量，并通过学习变量之间的转换矩阵来挖掘潜在变量。还有一些数据驱动的补全方法。 Yi等人[72]为具有地理特征的空气质量数据，提出了基于地理位置的的缺失补全方法。 Wang等人[15]通过协同过滤来补全推荐系统中的缺失值。

最近，一些研究人员试图使用递归神经网络来计算缺失值[18,77,78]。递归神经网络经常与分类/回归部分一起训练，从而显着提高了准确性。然而，他们认为缺失值可以表示为全局平均值和最新观测值的组合。他们使用RNN以平滑的方式补全缺失数据，但忽略了动态系统中变量的关联性。与现有方法不同，我们的模型直接对动力系统进行近似，并且表现出优异的结果。

A.4 嘈杂数据环境下的用户身份识别

随着基于位置的服务和应用的普及，大量的移动数据源源不断的从异构数据源（例如不同的GPS嵌入式设备，移动应用或基于位置的服务提供商）生成。在本节中，我们研究在这些异构数据源中，根据轨迹自动识别用户的有效方法。我们提出了一个名为*Automatic User Identification*（AUI）的MapReduce的框架，该框架易于部署并可以扩展到非常大的数据集。我们的框架基于一个叫做signal based similarity（SIG）的全新相似性度量。它从不同数据源收集的轨迹中，测量用户轨迹的相似性，并对采样频率差异较大，噪声较高的情况有较好的鲁棒性。在SIG相似度中，我们将两个用户的共现事件 (co-occurrence event) 在多分辨率下表示，并基于所获得的表征计算相似性分数。这种相似性度量对于大规模高噪音数据集来说非常有效。我们进行了广泛的实验评估，结果显示我们的框架显着优于现有方法。

A.4.1 引言

无处不在的基于位置的服务和应用，使人们能够在日常生活中使用GPS嵌入式设备进行导航，规划旅行路线和共享地理位置信息。每天，大量的时空数据流源源不断的产生出来。这些数据为挖掘人类轨迹行为模式和特征提供了新的机会。最近出现了越来越多的由移动数据支持的研究。同时，此类移动数据的挖掘在各种工业和商业应用中也显示出巨大的潜力，包括交通分析[1–3]，旅行推荐[4–7,82–85],基于位置的社交网络[8–13]，地理搜索[86,87]等。

在实际应用中，移动数据通常是从异构数据源产生的，例如不同的GPS嵌入式设备，移动应用程序或LBS提供商等。在本文中，我们旨在研究从异构数据源收集的移动数据集中，识别用户的有效方法。一个密切相关的话题是用户

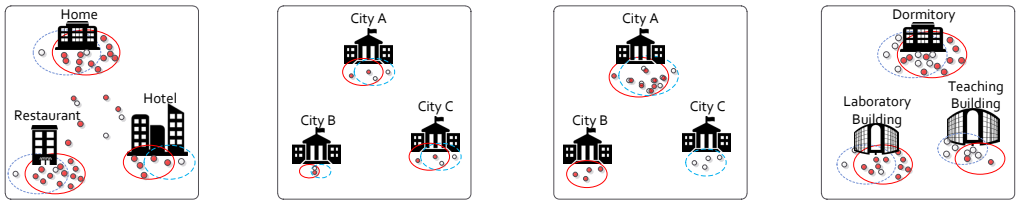
相似性搜索，其目标是检索具有类似时空模式的用户子集。本质上，我们可以将用户识别问题看作用户相似性搜索问题的一个特例。对于每个轨迹，我们从另一个数据源中检索最相似的轨迹，并确定它们是否属于同一个人。

尽管用户相似性搜索已经被相当广泛地研究了，但是从异构数据源收集的移动性数据的研究并没有太多工作，而这些数据来源要复杂得多。在我们的实验中，数据是从各种移动应用程序收集的，包括导航数据，地图查询，社交平台中的地理标记记录等。为了使我们的介绍更加具体，我们首先说明一些我们观察到的数据特性。

- 我们所研究的问题中，最显著的特性是轨迹采样率在不同数据集中差异非常大。例如，GPS导航数据的轨迹通常以非常高的速率采样，而地理标签或地图查询以极低的速率采样。即使在相同的数据源中，使用频率在不同的时间间隔内也会发生剧烈变化。另一方面，大多数先前的工作使用大致一致（并且通常是密集的）采样数据[83,88,89,92]。
- 对于一些稀疏的轨迹，几乎不可能推断任何用户移动的信息。例如，对于我们数据集中的地理标签签到数据，每个用户平均每2.63天就只生成一条GPS记录。以往的大部分工作都是基于用户的移动行为，如速度，移动方向，时空接近度等来衡量轨迹相似性（即两个轨迹只有在几乎同时出现在大致相同的地方才相似）[88,92–96]。但是，由于极度的稀疏性，我们的数据中没有这些功能。
- 具有密切关系的用户，其轨迹通常有明显的重叠。例如，我们调查了一些T大学学生的轨迹数据。他们大多数的轨迹在他们的办公楼和宿舍。这种重叠使得很难区分它们是否为同一用户。然而，在此情况下，以前的工作很少关注这种情况。
- 对于不同的数据源，轨迹可以在时间上不相交。例如，同一组用户在两个数据集中有不相同的标识。一个在一月收集，另一个在二月收集。特别是对于商人来说，他们可能会在几个月内去几个不同的城市，这就很难衡量他们的相似之处。

为了给读者提供更直观的解释，我们在例1中展示了一个例子。

例 1： 在图*A.4*中，我们展示了在真实数据集中观察到的四种典型情况（还有很多其他情况或这些情况的组合，这里不可能详尽无遗地列出）。



(a) Trajectories of the same person which are sampled at very different rates.

(b) Trajectories of the same person which co-occurred in several places far apart from each other.

(c) Trajectories of the same person which are disjoint in several cities.

(d) Trajectories of two school mates which have significant overlap.

图 A.4　我们的轨迹数据中四种典型的情况。

*(a)(b)(c)*三个图中，分别表示两个轨迹（来自不同的数据源）属于同一个人，*(d)*图显示属于两个不同人（同学关系）的轨迹。

- 在*(a)*中，白色轨迹的采样速度远低于红色轨迹，但它们都经常出现在几个固定的地方。如*(b)*图所示，这种共现对识别同一个人非常重要，特别是当他们发生的距离很远时，比如在几个不同的城市。
- 然而，在*(c)*中，我们从数据中观察到，同一个人的轨迹在多个城市也可能是不相交的。当轨迹在时间上不相交时尤其会经常发生这种情况。然而，由于它们在一个城市（城市*A*）中共现频率极高，我们仍然可以确定它们属于同一个人。
- 在*(d)*中，由于同一校园中的轨迹具有明显的重叠，很难区分是否为同一个用户。尽管如此，实验楼的红色轨迹更多，而白色的轨迹更倾向于聚集在教学楼。这种模式使我们有可能从中区分它们。 □

上述特性使我们的用户识别问题与之前截然不同。为了应对这一挑战，我们提出了一个基于MapReduce的框架，称为*Automatic User Identification*（AUI），它基于一种新颖的轨迹相似性度量方法。AUI易于部署，并可以扩展到非常大的数据集。我们在下面总结我们的技术贡献：

- 我们在大规模异构移动数据集上形式化描述了用户识别问题，并提出一个名为AUI的基于MapReduce的框架。

- 我们基于MapReduce的框架设计了一个有效的过滤策略。使用过滤策略，对于每个轨迹，我们只需要将其与少量候选项进行比较。过滤策略是AUI可以扩展到非常大的数据集的基础。
- 我们通过考虑共现的频率和发生的位置，设计了一种称为基于*signal based similarity* (SIG)的新颖的轨迹相似性度量。由于我们的数据是从许多不同的数据源收集的，因此我们不会假设移动数据的任何属性（例如采样率，时间跨度）。我们表明，与现有的方法相比，我们的方法可以有效处理极其嘈杂的情况。实验结果表明，我们的相似性度量对于异构数据源的用户识别问题更加鲁棒和准确。
- 我们采取拒绝策略来减少错误识别案例。许多应用场景对错误识别高度敏感，即一些错误的情况可能会导致严重的后果。我们的策略显著提高了框架的准确性。
- 我们通过6个不同情况下的实验来评估我们的框架。对于最简单的情况（31,511个来自全国的用户），我们获得99.94%的准确性。对于最困难的情况（在同一校区学习的14,115个大学生），我们获得了90.09%的准确性。而在这种情况下，现有最好的方法只能达到61.38%的准确性。

A.4.2 公式化定义与模型架构

我们首先针对我们的问题提出几个有用的定义。

定义 4 (轨迹)： 轨迹$T = \{p_1, p_2, \ldots, p_{|T|}\}$是一个具有时间空间属性的点序列。每个点$p_i$与三个属性$x, y, t$相关联，其中$(x, y)$ ① 表示p_i的坐标，t表示记录p_i的时间戳。

这里，我们强调我们数据中的轨迹可能非常稀疏，部分用户平均每天产生的定位可能少于1个点。

定义 5 (轨迹数据集)： 轨迹数据集D被定义为一些用户轨迹的集合。D 中的每个轨迹T 与一个标识 $T.id$相关联。

① 我们在实验中使用mercator坐标

定义 6 (匹配轨迹)： 假设两个轨迹T_A, T_B是来自不同的轨迹数据集中的两段轨迹。如果T_A和T_B是由同一用户生成的，则我们称T_B为T_A的匹配轨迹。

我们的问题定义如下。给定两个轨迹数据集D_A和D_B（通常从两个不同的数据源收集），对于D_A中的每个轨迹T_A，我们的目标是确定D_B中是否存在T_B，使得T_B是 T_A的匹配轨迹。我们不对轨迹做任何特殊性质假设。因此，轨迹的采样率可能非常高或非常低。此外，D_A和D_B可能在时间上不相交，即，它们分别在不同的时间段中收集得到。

我们首先解释我们用户识别算法的底层原理。 Montjoye等人[99]表明人类的出行轨迹具有非常高的唯一性。每个人都有自己固有的出行模式。人们往往会反复访问他们经常去过的地方。即使对于关系非常密切的用户来说，他们仍然具有明显不同的出行模式。人类出行轨迹的这种唯一性使我们能够识别属于来自不同数据源中的同一用户轨迹。

为了处理极其稀疏的轨迹（稀疏性使得我们难以从数据中推断用户的移动模式），我们取一段很长的时间段（例如，3个月期间的轨迹）并观察用户的累积轨迹。通过累积的历史轨迹数据，我们更容易推断出用户的出行模式，例如他/她倾向于访问的地点。图A.5显示了我们的数据集中的一个例子，其中该用户在2015年2月至5月平均每日生成5个轨迹点。通过积累所有这些轨迹点，我们可以清楚地看到这个用户通常在家里和工作场所。此外，他/她曾在过去的3月内访问过王府井和北京植物园。

受此启发，我们考虑多粒度下轨迹对的共现位置以及频率，并基于这些因素来定义我们的轨迹相似性。

A.4.3 相关工作

基于轨迹的用户识别主要关注于人类轨迹行为中唯一性 (uniqueness)的研究。现有结果已经表明，人们往往会去往过去经常访问的地方，我们称之为关键位置[9,99,102,103]。Zang和Bolot[103]表明，给予每个人最高的访问频率n关键的位置，我们可以从一个非常大规模的匿名数据集中唯一地识别出一小部分用户。Montjoye等人[99]提出，即便我们将原始数据进行模糊化，人类的移动轨迹依然保留了非常强的唯一性。我们强调这些工作调查了单个移动数据集中的唯

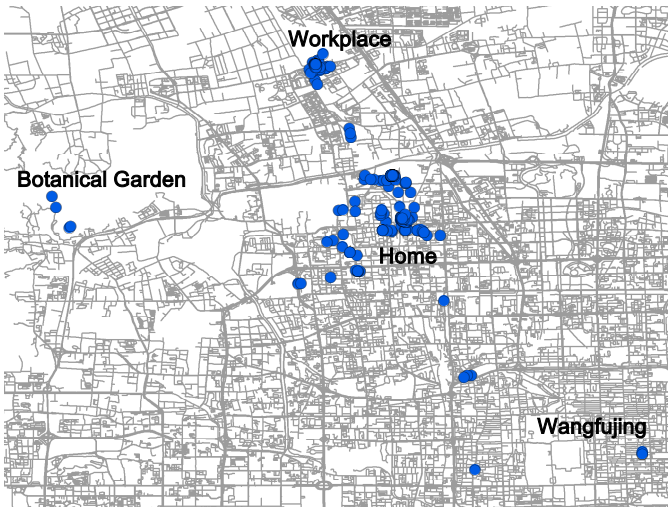


图 A.5　某轨迹示意图。

一性以及用户识别问题，即：给定了已包含在数据集中的几个历史位置点，检索与给定数据匹配的轨迹。Rossi等人[98]提出了一种全新的方法，即使关键位置点不包含在原始数据集中，依然能够根据这些关键点识别出其代表的用户。我们指出在该工作[98]中，他们实验中未出现的关键位置点是从原始数据集中抽样出来的，与我们的问题有所不同。针对我们的问题，它们的方法不能有效处理跨异构数据源下的用户识别。此外，在另一项相关工作中，Crandall等人[9]在社交平台中，使用稀疏的地理标签来推断不同用户之间的社交关系。他们表明，社会关系的强度与用户在不同空间粒度下的地理共现性高度相关。

用户相似性搜索问题旨在基于从用户过去的轨迹行为中提取个人特征，进而从整个数据库中检索具有相似轨迹的用户（在我们的问题中，即要抽取用户轨迹中的时空特征）。正如我们所声明的，用户识别可以看作是用户相似性搜索的特例。大多数方法的最根本因素是解决如何定义一对用户轨迹之间的相似性。过去已经提出了相当多的相似性度量的定义。其中大多数是传统方法的扩展或变体，包括动态时间规整（DTW）[94]，最长公共子序列（LCSS）[88]，编辑距离（包括ERP， EDR等）[92,93]等。另外，Li等人[89]提出了一种基于层次图的相似性度量（HGSM），它考虑了用户轨迹中地理位置的顺序和分层属性。这些方法利用了轨迹的时间接近性，即如果它们大致在同一时间共同出现，则两个轨迹判定为相似。Chen等人[97]基于轨迹中的空间距离和顺序约束来定义称为k-BCT的轨迹相似性度量。k-BCT旨在使用少量位置作为查询词，从数据库中

搜索轨迹。因此在用户识别问题中使用它，可能会导致轨迹重叠时的错误率明显提高。在最近的研究中，Ranu等人[95]研究了不一致采样率下轨迹的相似性度量。他们制定了一个称为编辑与投影距离（EDwP）的稳健距离函数来匹配可变采样率下的轨迹。然而，他们的方法需要推断用户的移动信息，在我们的问题中，由于潜在的极度稀疏性，有时移动信息并不可推测。

在近期的研究中，有一系列针对用户身份识别的改进方法。Chen等人提出一种对用户轨迹行为的全新表示方法叫做STUL[105]。基于该工作，他们进一步提出一个基于密度函数和熵的方法，来估计不同网格上的用户身份识别的置信度。该方法得到了非常良好的效果，尤其是对于基于位置的社交数据。我们需要强调的是，他们的方法是我们在本章中所提出方法的后续工作。

A.5 总结

GPS嵌入式设备（如智能手机，环路传感器）的普及导致了时空数据的爆炸式增长。正如我们所说的，挖掘时空数据潜在的价值，在各种工业和商业应用中具有重要意义。然而，在许多实际情况下，例如交通推荐和基于位置的社交网络，时空数据通常是呈现海量规模，并具有较大噪音。时空数据中的潜在模式 (pattern) 受复杂的空间相关性，时间依赖性和许多外部因素的影响。为了在挖掘时空数据方面取得有意义和准确的结果，从数据中提取出有效的特征表示尤为重要。在本论文中，我们研究时空数据的表征学习。我们提出时空数据表征学习的诸多挑战，并且我们用三个具体的例子来说明如何在学习时空数据的特征表示的同时，解决相应的挑战。

在第A.2节中，我们研究了如何在数据表征中捕获复杂的时空依赖。我们以交通出行时间预测问题为例。在这个问题中，我们提出了一个基于深度神经网络的端到端框架，叫做DeepTTE。我们的模型包含一个基于地理信息的卷积模块，用来捕捉数据中的细粒度空间关联，一个递归神经网络模块，用来学习不同道路之间的时间依赖，以及一个外部属性模块用来处理各类外部因素的影响。这种架构可以有效地，自动地，学习给定数据的时空特征表示。在DeepTTE的最顶层，我们还设计了一个多任务学习模块，以结合分段道路预测和整体道路预测两种方法，从而达到更好的效果。

在第A.3节中，我们研究处理时空数据中的缺失值。我们提出了一种全新的缺失补全方法BRITS。BRITS利用循环动力系统来学习具有复杂依赖关系的时空数据特征表示，并根据学习到的特征表示来进行缺失补全。我们的模型不对数据生成过程进行特殊性假设。取而代之，我们使用双向递归神经网络来学习捕获到的相关性的特征表示，并直接预测缺失值。我们的模型将缺失值视为RNN计算图的变量。补全结果误差在反向传播过程中可以被充分优化，从而使得补全结果更加准确。同时缺失补全算法与给定时间序列的分类/回归模块进行联合训练。实验结果表明，我们的模型比现有的方法在补全和分类/回归方面都表现出更准确的结果。

在第A.4节中，我们进一步考虑极端嘈杂时空数据下的表示学习。作为一个具体的例子，我们研究了异源轨迹数据中识别同一用户的方法。这个问题中涉及到的轨迹数据集非常嘈杂，其采样速度差异较大，导致识别用户的场景同样变得非常复杂。我们提出signal based similarity (SIG)作为轨迹相似性度量，来测量大规模轨迹数据集中的轨迹相似性。在SIG中，我们使用用户轨迹的共现事件作为数据表示方法。基于该表示法，我们在多分辨率下计算用户相似性得分。我们提出了一个名为 Automatic User Identification (AUI) 的基于MapReduce的框架，它使我们能够在非常大的数据规模上快速计算SIG相似性。我们进行了广泛的实验。实验结果表明，我们的模型显著优于现有的方法。

我们在本论文中使用的框架和模型，为时空数据的表示学习方法提供了原则性的指导。特别的，本文提出的方法能够捕捉时空数据中复杂的依赖关系，处理缺失值，并有效处理数据嘈杂问题。

A.6 未来工作

海量时空数据的表示学习，是一个非常重要，且具有挑战性的问题。它在学术和工业领域仍然是一个热门话题。在本节中，我们将介绍时空数据表示学习的两个未来方向。

首先，对于一些极其复杂的场景，学习有效的特征表示仍然依赖手动设计的规则。例如，对于第A.4节中的用户识别问题，我们依然针对数据采用了一种

人为特殊设计的算法，来计算用户轨迹之间的相似性。在最近的研究中，一些研究人员使用Graph Embedding的方法，在极少依赖人工特征的情况下，从轨迹数据识别出不同用户[107,108]。然而，他们的方法只适用于采样率几乎相同的，相对规整的移动数据。就目前所知，仍然没有有效的方法，能在不大量依赖人工规则情况下，对嘈杂的大规模轨迹数据进行用户识别。在实际应用中，通过精心设计的深度模型，从原始数据中直接学习特征表示，可以使得我们能够自动发现数据中有用的层次特征。然而，这些结构的设计同样并不容易，并且运行时间对于大规模数据集而相对较低效。相比之下，基于人工规则的特征表示通常稳定性强并且很容易在大规模数据集上扩展，但需要耗费更大量的人力。如何在两种特征表示方法之间的取到一个好的平衡点，是我们考虑的未来工方向之一。

另一个研究方向是如何利用迁移学习来帮助学习时空数据的特征表示。许多时空数据挖掘任务中有诸多共性模式。例如，假设我们已经收集了过去一个城市中每条道路的拥堵程度。我们旨在预测未来几分钟内每条道路的拥堵程度。这个问题与我们的交通出行时间预测问题高度相关，如我们在第A.2节中介绍的那样，旅行时间和拥堵程度都受相邻路段的空间相关性和时间依赖性的影响。因此，如果我们能够将出行时间估计问题中学习到的特征，迁移到拥堵等级预测问题中，会有利于我们更精确的估计每条道路的拥堵程度。在迁移学习中，我们专注于将原始问题中学到的知识，应用到相关的目标问题中[109]。然而，现有的转移学习方法大多集中在视觉任务上，如图像分类，对象检测等。很少有工作考虑关于时空数据的迁移学习问题。

个人简历、在学期间发表的学术论文与研究成果

个人简历

1990年10月5日出生于山西省太原市。

2009年9月考入武汉大学计算机学院，并于2013年7月获得工学学士学位。

2013年7月进入清华大学交叉信息研究院攻读博士学位至今。

发表的学术论文

[1] **Wei Cao**, Jian Li, Yufei Tao, Zhize Li. On Top-k Selection in Multi-Armed Bandits and Hidden Bipartite Graphs. 29th Neural Information Processing Systems (NIPS), 2015

[2] **Wei Cao**, Jian Li, Shimin Li, and Haitao Wang. Balanced Splitting on Weighted Intervals. Operation Research Letters (ORL), 2015.

[3] **Wei Cao**, Zhengwei Wu, Dong Wang, Jian Li, Haishan Wu. Automatic User Identification Method across Heterogeneous Mobility Data Souces. 32nd IEEE International Conference on Data Engineering (ICDE), 2016.

[4] Dong Wang, **Wei Cao**, Mengwen Xu and Jian Li. ETCPS: An Effective and Scalable Traffic Condition Prediction System. 21st International Conference on Database Systems for Advanced Applications (DASFAA), 2016.

[5] **Wei Cao**, Jian Li, Haitao Wang, Kangning Wang, Ruosong Wang, Raymond Chi-Wing Wong and Wei Zhan. k-Regret Minimizing Set: Efficient Algorithms and Hardness. 20th International Conference on Database Theory (ICDT), 2017.

[6] Dong Wang, **Wei Cao**, Jian Li and Jieping Ye. DeepSD: Supply-Demand Prediction for Online Car-hailing Services using Deep Neural Networks. 33rd IEEE International Conference on Data Engineering (ICDE), 2017.

[7] Dong Wang, Junbo Zhang, **Wei Cao**, Yu Zheng and Jian Li. When Will You Arrive? Estimating Travel Time Based on Deep Neural Networks. 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.