Estimator: An Effective and Scalable Framework for Transportation Mode Classification over Trajectories

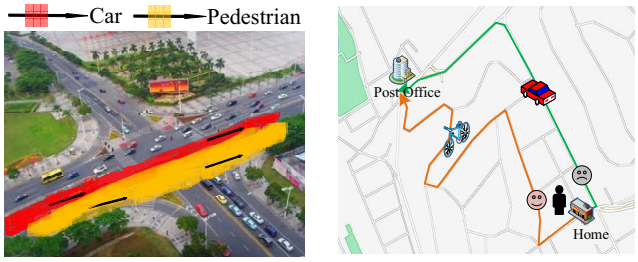
Danlei Hu, Ziquan Fang, Hanxi Fang, Tianyi Li, *Member, IEEE*,
Chunhui Shen, Lu Chen, Yunjun Gao, *Member, IEEE*

***Abstract*—Transportation mode classification, the process of predicting the class labels of moving objects' transportation modes, has been widely applied to a variety of real-world applications, such as traffic management, urban computing, and behavior study. However, existing studies of transportation mode classification typically extract the explicit features of trajectory data but fail to capture the implicit features that affect the classification performance. In addition, most of the existing studies also prefer to apply RNN-based models to embed trajectories, which is only suitable for classifying small-scale data. To tackle the above challenges, we propose an effective and scalable framework for transportation mode classification over GPS trajectories, abbreviated Estimator. Estimator is established on a developed CNN-TCN architecture, which is capable of leveraging the spatial and temporal hidden features of trajectories to achieve high effectiveness and efficiency. Estimator partitions the entire traffic space into disjointed spatial regions according to traffic conditions, which enhances the scalability significantly and thus enables parallel transportation classification. Extensive experiments using eight public real-life datasets offer evidence that Estimator i) achieves superior model effectiveness (i.e., 99% *Accuracy* and 0.98 *F1-score*), which outperforms state-of-the-arts substantially; ii) exhibits prominent model efficiency, and obtains 7–40x speedups up over state-of-the-arts learning-based methods; and iii) shows high model scalability and robustness that enables large-scale classification analytics.**

***Index Terms*—Transportation Mode Classification, Trajectory Data Mining, Deep Learning**

I. Introduction

With the ubiquitous uses of GPS-equipped devices and mobile computing services (e.g., Twitter, Weibo, and other location-based apps), massive GPS trajectories are collected. The collected data enables to describe the mobility characteristic of moving objects such as bikes, buses, taxis, and pedestrian [37]. Moreover, it has motivated various trajectory-based pattern mining tasks that provide location-based services such as traffic management [27], [36], urban computing [28], and behavior study [18]. Being a typical and fundamental pattern mining task, transportation mode classification aims to classify trajectories from different moving objects according to their travel modes, e.g., taxi-mode and walking-mode. Note that, a moving object could change its transportation modes during traveling. For example, a city commuter may first take a bus, then take the subway, and finally walk, when commuting from home to his/her working place. Given a trajectory dataset generated by different moving objects without label information (for the sake of trajectory privacy preserving [12]), enabling to detect the transportation mode of any trajectory can benefit a variety of services, such as traffic planning [27], tourism demand analysis [2], and mobility study [5]. Two motivating examples illustrate this.



(a) Traffic Planning (b) Travel Demand Analysis

Fig. 1. Transportation Mode Classification

As shown in Fig. 1(a), transportation mode classification can find there are more pedestrian trajectories than vehicle trajectories crossing this crossroad in a certain time period. Hence, a flyover or underpass may be constructed to ease traffic congestion.

Fig. 1(b) shows two paths marked by green and orange colors, respectively, between home and post office. With transportation classification over historical trajectories between two locations, we identify that the green path is generally traveled by cars, while the orange path is generally traveled by bikes, and thus, we can recommend the desirable mode for the user according to his/her habits.

Existing studies of transportation mode classification primarily focus on **trajectory classification**, which leverages the spatio-temporal features of moving object trajectories to facilitate classification [2], [28], [34]. Specifically, existing transportation mode classification studies can be generally divided into two categories: traditional machine learning based methods and deep-learning based methods. Methods in the former category manually extract the explicit features (e.g., speed, acceleration, and stay time) of trajectories, feed these features into classifiers (e.g., k-nearest neighbour (kNN) model [39], hidden markov (HMM) model [16], and support vector machine (SVM) model [3]). However, they cannot capture the complex and hidden features of trajectories due to their

Manuscript received July 29, 2022. *(Corresponding Author: Lu Chen)*

Danlei Hu, Ziquan Fang, Lu Chen and Yunjun Gao are with the College of Computer Science, Zhejiang University, Hangzhou 310007, China (e-mail: dlhu@zju.edu.cn; zqfang@zju.edu.cn; luchen@zju.edu.cn; gaoyj@zju.edu.cn).

Hanxi Fang is with the School of Earth Science, Zhejiang University, Hangzhou 310007, China (e-mail: hanxif@zju.edu.cn).

Tianyi Li is with the Department of Computer Science, Aalborg University, Denmark (e-mail: tianyi@cs.aau.dk).

Chunhui Shen is with the Alibaba Group, Hangzhou 310007, China (e-mail: zjusch2@163.com).

simple and manual-dependent operations [5], [11]. Thus, the effectiveness of the classification is limited, especially when the moving objects share very similar mobility characteristics, such as taxi and private car trajectories.

Deep learning has been successfully applied to facilitate tons of services, such as text classification [9], image classification [32], and trajectory representation learning [4], [30], [31], via capturing the non-linear and hidden features. This motivates researchers to study deep-learning based transportation mode classification analyses, which can be mainly categorized into CNN-based methods [8], [11], RNN-based methods [14], [25], [33], and CNN-RNN based methods [15], [26]. Convolution neural network (CNN) based methods typically transform a GPS trajectory into a gray image via feature extraction (i.e., pixel calculation), where the pixel values in the image represent the average speed, the time duration, and other features of the trajectory. Then, existing image classification methods are applied to trajectory classification. However, CNN-based methods focus on the spatial dependencies but ignore the temporal correlations among sampling points in a trajectory. To capture it, recurrent neural network (RNN) based models are developed, which is able to learn time-series sequences. As trajectories can be formulated as a type of time-series sequences, RNN based models can feed trajectories into RNNs to extract their temporal features for transportation classification. More recently, a new branch of studies combines CNN and RNN, i.e., CNN-RNN based methods [8], [11], which takes advantages of CNNs and RNNs to consider both spatial and temporal information of trajectories to improve the performance of classification. However, ST-GRU [26], the state-of-the-art method, suffers from poor performance of RNNs when conducting parallel processing, which degrades its model efficiency and scalability.

Classification effectiveness, training efficiency, and model scalability are primary aspects to measure the performance of learning-based transportation classification [8]. With these in mind, we revisit the problem of transportation mode classification, and propose an effective and scalable framework for transportation mode classification over GPS trajectories, termed **Estimator**, while address challenges below.

Challenge I: How to capture the implicit spatio-temporal characteristics of trajectories to improve the effectiveness of classification analysis? In addition to extracting the explicit features (e.g., travel speed and stay time of moving objects) that are already achieved by existing approaches, we aim to extract hidden mobility features to improve classification. For instance, although taxis and private cars share similar moving features as they both follow vehicle mode, they have different hidden mobility patterns. Specifically, private cars typically travel around several fixed locations (e.g., home and working place), while taxis do not. Capturing such hidden features like fixed locations in certain time periods helps to distinguish the trajectories of taxis and private cars more effectively. However, there are mainly two challenges for capturing the implicit spatio-temporal characteristics of trajectory data. First, the mobility features generally vary a lot across different traffic regions [24], which are difficult to be captured by existing deep-learning based studies [11], [26]. For example, even for the same taxi, its traveling speed is faster in the suburbs while is slower in the city center. However, in this case, existing deep-learning based studies [11], [26] that treat the entire city-wide training trajectories as a whole cannot identify the varied transportation mode of the taxi's trajectory. Second, the features of moving trajectories are unknown and dynamic during the moving process (e.g., whether it has periodic features and how the features evolve), which makes the feature embedding challenging.

Challenge II: How to overcome the limited parallel processing of RNNs to improve the efficiency and scalability of trajectory classification? Although the state-of-the-art CNN-RNN based method [26] outperforms both CNN-based and RNN-based methods, it still exhibits limited capacity of parallel processing due to the inherent computation mechanism of RNNs. Specifically, RNNs scan the sampling points of a trajectory one by one, meaning that RNNs cannot process the next point until the current point has been processed. Consequently, the RNNs-depended methods fail to deal with large-scale classification. Furthermore, although CNNs are able to process sampling points of a trajectory in parallel, the size of convolution kernel limits the ability of capturing spatio-temporal correlations due to the different lengths of trajectories. Put differently, existing CNN based methods cannot efficiently process sequences with varying lengths, degrading the scalability of classification.

To address the first challenge, we extract periodic features of taxis and private cars to distinguish their mobility features. In addition, we partition the entire traffic space into disjointed regions to handle varying traffic conditions. During this process, different features in partitions with different traffic environment are extracted, and thus the effectiveness of classification can be enhanced (cf. ablation study in Section IV). To tackle the second challenge, we develop Estimator, a unified CNN-TCN architecture for effective and efficient transportation mode classification analyses. We extract spatial features by CNN model, and propose to employ temporal convolutional network (TCN) [1] to capture the temporal informatoin of trajectories efficiently. Further, based on the partitioned traffic space, we extend CNN-TCN architecture for parallel transportation mode classification to improve model scalability. Overall, this paper makes the following contributions:

- *CNN-TCN Architecture.* We construct a CNN-TCN architecture to capture the spatio-temporal mobility characteristics of moving objects for transportation mode classification. To the best of our knowledge, this is the first proposal to apply TCN to trajectory data.
- *Hidden Mobility Feature Extraction.* Estimator considers both explicit and implicit mobility features. In terms of explicit features, we capture speed, azimuth, stay time, etc. In terms of implicit features, we capture periodic patterns of taxis and private cars to distinguish different mobility characteristics. Based on this, Estimator enables fine-grained feature embedding to improve effectiveness.
- *Partition and Parallel Training.* We partition the whole city into disjointed areas, i.e., urban center, urban area, and suburb. Moreover, we extend CNN-TCN architecture for parallel transportation mode classification to further improve the model scalability.

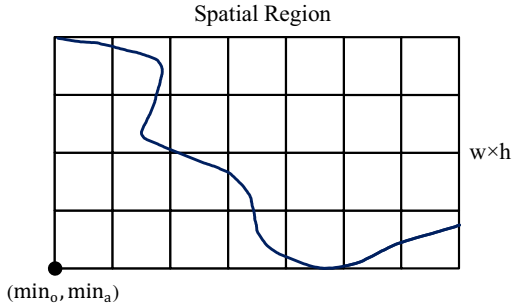


Fig. 2. Trajectory Spatial Region

- *Extensive Experiments.* We conduct extensive experiments on eight real-world trajectory datasets that offer insight into Estimator and demonstrate that it is able to outperform the state-of-the-art competitors in terms of machine learning based and deep learning based methods.

The rest of the paper is organized as follows. Section II presents the problem statement. Section III details our proposed framework Estimator. Section IV reports the experimental results, and the related work is reviewed in Section V. Finally, Section VI concludes the paper.

II. Preliminaries and Problem Definition

We proceed to introduce preliminary definitions, based on which, we define the problem of trajectory mode classification.

Trajectories can be collected by smartphone sensors [15] or GPS-based equipment [8]. Different from GPS-based trajectories which is mainly consisted of the locations and timestamps of sampling points, the sensor-based trajectories contain not only timestamps but also the information of sensor gravity, gyroscope, ambient light and so on. However, the complete records are often not available. In this case, it is necessary to discover transportation mode classification only using locations and timestamps. Thus, this paper focuses on GPS trajectories. A raw trajectory is defined as follows:

Definition 1: **(Raw Trajectory)** *A raw trajectory T is a time-ordered sequence that consists of GPS points $p_i (1 \leq i \leq N)$, i.e., $T = \{p_1, p_2, ..., p_N\}$, where N denotes the length of a trajectory. Each GPS point p_i is in the form of $\langle id, l_o, l_a, t_s \rangle$, where id is the identifier, l_o is the longitude, l_a is the latitude, and t_s is the time when p occurs.*

A trajectory may contain thousands of GPS points (i.e., N is extremely large), and thus, its transportation mode may change during the traveling of the moving object. Following existing methods, trajectory segmentation can be used to detect the evolution of transportation mode. Specifically, we use the stay point detection method [23] to split a raw trajectory into several successive segments to ensure each trajectory segment only features one mode. Unless stated otherwise, we assume that trajectories are segmented in the rest of paper.

Definition 2: **(Mapped Trajectory)** *A mapped trajectory is a trajectory image, which consists of $w \times h$ uniform grid cells. The pixels of each grid cell are represented by an RGB tuple of the mean azimuth, average speed, and stay time of all trajectory points located in this grid cell.*

The spatial region traversed by a mapped trajectory is a minimum bounding rectangle to include all of its GPS points, as shown in Fig. 2. However, images must have same number

Algorithm 1: Mapped Trajectory Generation

Input: a trajectory $T = \{p_1, \ldots, p_N\}$, the number of grid cells $w \times h$, left-bottom corner (min_o, min_a)
Output: the mapped trajectory I_s

Initialize $I_s \in \mathbf{R}^{w \times h \times 3} \leftarrow 0$, $I_M \in \mathbf{R}^{w \times h \times 4} \leftarrow 0$
for *each* p_i *in* T **do**
  $x_i = \lfloor \frac{p_i.l_o - min_o}{h} \rfloor$, $y_i = \lfloor \frac{p_i.l_a - min_a}{w} \rfloor$
  update $[n, p_s, p_e, d]$ of $I_M(x_i, y_i)$ // the number of points, start point, end point and covered distance in grid cell (x_i, y_i)
for *each grid cell* (x, y) **do**
  $p_s = I_M(x, y, 1)$
  $p_e = I_M(i, j, 2)$
  $I_s(i, j, 0) \leftarrow$ Azimuth angle between p_s and p_e
  $ST = p_e.t_s - p_s.t_s$ // stay time
  $I_s(i, j, 1) \leftarrow \frac{I_M(i,j,3)}{ST}$ // speed
  $I_s(i, j, 2) \leftarrow ST$
return I_s

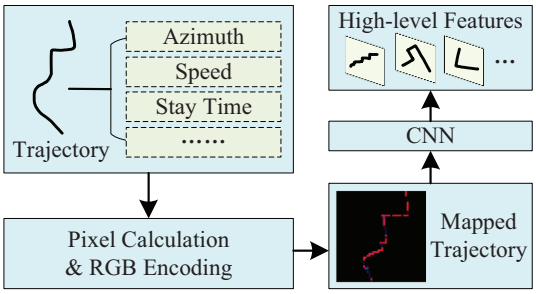


Fig. 3. Trajectory Mapping

of pixels that are directly fed to the training model. Thus, we fix the number of grid cells to $w \times h$ in the mapped image [11]. For each sample point $p_i = \langle l_o, l_a, t_s \rangle$ in a trajectory, its mapped grid cell (x_i, y_i) can be calculated as:

$$x_i = \lfloor \frac{p_i.l_o - min_o}{h} \rfloor, y_i = \lfloor \frac{p_i.l_a - min_a}{w} \rfloor. \quad (1)$$

where (min_a, min_o) is the left-bottom corner of the spatial region of this trajectory. For each grid cell (x, y), we collect the number of points n in (x, y), the start point p_s (i.e., the earliest point occurred in (x, y)), the end point p_e (i.e., the latest point occurred in (x, y)), and the covered distance d (i.e., the trajectory length in (x, y)).

As shown in Fig. 3, we first map an original trajectory into a mapped image according to Definition 2, and then calculate its moving features (i.e., pixels) in each grid cell of the mapped image. The different colors in image grid cells represent different mobility characteristics of the moving object. We detail how to transform an original trajectory into a trajectory image in Algorithm 1. Finally, the mapped image is fed to CNN in order to obtain high level features.

Definition 3: **(Transportation Mode Classification)** *Given a set of raw trajectories generated from moving objects, i.e., $\mathcal{T} = \{T_1, T_2, ..., T_M\}$, we aim to train a classifier to classify them into different groups based on their transportation manners below:*

$$f(\theta) : \mathcal{T} \rightarrow \mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_K. \quad (2)$$

where $\mathcal{T}_j$ $(1 \leq j \leq K)$ denotes a specific transportation mode such as bike, taxi, private-car, bus, or subway, K denotes the

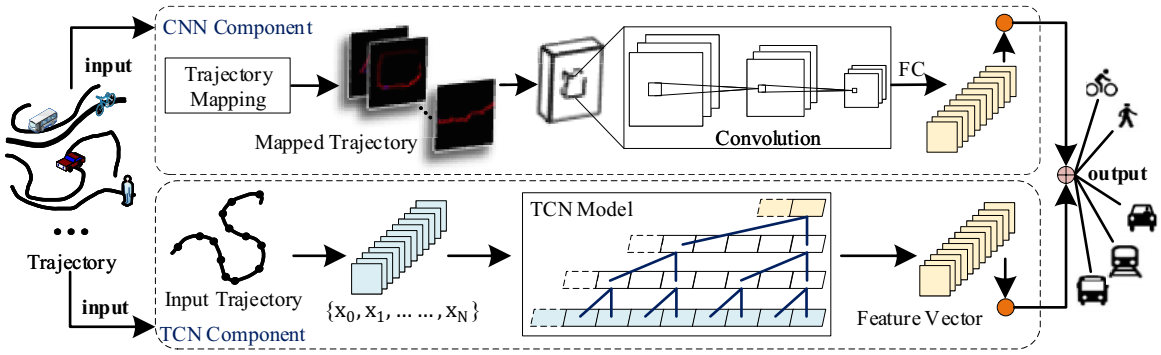


Fig. 4. The CNN-TCN Architecture for Estimator

total number of transportation manners, and θ is the target parameter set of Estimator (to be detailed in Section III).

III. The Proposed Method

We first present the CNN-TCN architecture, and then introduce the optimization techniques including hidden mobility extraction and data partition, which are the basis of Estimator.

A. CNN-TCN Model

We combine CNN and TCN to obtain a CNN-TCN model, as shown in Fig. 4. Specifically, we feed the given trajectories into CNNs and TCNs simultaneously to capture the spatial and temporal dependencies of trajectory data for transportation mode classification. We detail the two components below.

CNN Component. As depicted in the upper part of Fig. 4, we transform raw trajectories into mapped trajectories (trajectory images) through trajectory mapping, i.e., $T \rightarrow MT$ (cf. Definition 2). During the mapping, the spatial features of trajectories can be obtained and represented as pixels of images. Next, we feed the mapped trajectories into CNNs for representation learning, i.e., $MT \rightarrow CT$. Specifically, we utilize ResNet50 [17] for trajectory embedding while capturing the mobility features for transportation mode classification. The training process is detailed as follows.

$$MT_{l+1} = h(MT_l) + \mathcal{F}_{\mathcal{C}}\left(MT_l, W_l\right), \tag{3}$$

$$CT = MT_0 + \sum_{i=0}^{L-1} \mathcal{F}_{\mathcal{C}}\left(MT_i, W_i\right), \tag{4}$$

where the mapped trajectory MT is the input of CNN model, CT is the output of CNN model, $h()$ and $\mathcal{F}_{\mathcal{C}}()$ denote identity mapping and residual in ResNet, respectively, W_l denotes the convolution operation in the l-th layer, and L denotes the total number of layers. Based on Eqs. 3-4, we provide the CNN-based loss function below.

$$L_C = -\frac{1}{n}[CT \ln \hat{CT} + (1 - CT) \ln(1 - \hat{CT})], \tag{5}$$

where CT and $\hat{CT}$ represent the labeled transportation mode and predicted transportation mode, respectively.

TCN Component. Generally, each token in TCNs only contains a timestamp. However, we feed a large number of trajectories into TCNs, and thus, massive timestamps are included in each token. Different from general time series processing, we use multi-channel to record trajectory sequence. Specifically, when feeding trajectories into CNNs (in the upper part) to capture their spatial dependencies, we also feed raw trajectories into TCNs (in the lower part) to capture their temporal correlations at the same time, i.e., $T \rightarrow TT$, as shown in the lower part of Fig. 4. As observed, the TCN model enables reading/embedding an input trajectory sequence in a parallel fashion. The detailed process is as follows.

$$D(p) = (T *_d f)(p) = \sum_{i=0}^{k_s - 1} f(i) \cdot T_{p.id - d \cdot i}, \tag{6}$$

$$TT = Activation(T^{'} + \mathcal{F}_T(T^{'})), \tag{7}$$

where $D(\cdot)$ denotes the dilated convolution operation (i.e., $T \rightarrow T^{'}$) on the point p of the trajectory sequence T, d is the dilated factor, k_s is the size of filter f (i.e., kernel size), *Activation* denotes the active function (i.e., *ReLU*), and TT is the output of TCN model. The dilated convolution operation facilitates to capture all the time span of T. In Eq. 6, $T_{p.id-d\cdot i}$ denotes the history token of T, in order to capture the information in the past time. $\mathcal{F}_T()$ denotes the residual in TCN. In experiments, we set $k_s = 3$ and $d = [1, 2, 4, 8]$ following previous work [1]. The TCN-based loss function L_T is computed as follows:

$$L_T = -\frac{1}{n}[TT \ln \hat{TT} + (1 - TT) \ln(1 - \hat{TT})], \tag{8}$$

where TT and $\hat{TT}$ represent the labeled transportation mode and predicted transportation mode, respectively.

CNN-TCN Combination. We propose the overall training loss function of CNN-TCN model by summing up the CNN-based Loss and TCN-based Loss.

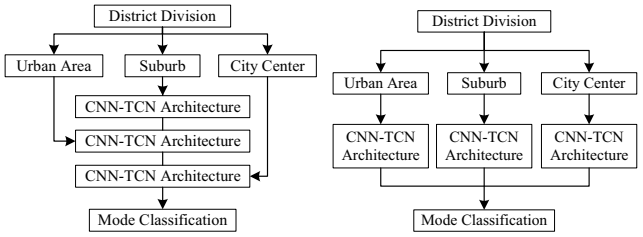
$$\mathcal{L} = \alpha L_C + \beta L_T, \tag{9}$$

$$\alpha = \frac{e^{r_1}}{e^{r_1} + e^{r_2}}, \beta = \frac{e^{r_2}}{e^{r_1} + e^{r_2}}. \tag{10}$$

where α and β are coefficients, and r_1 and r_2 denote the accuracy of CNN-based model and TCN-based model. We calculate the weight coefficients by softmax function.

B. Hidden Mobility Extraction

As discussed in Section I, moving objects such as taxis and private cars share similar mobility characteristics (e.g., moving speed) since they both belong to vehicle transportation, which



(a) Serial Computing (b) Parallel Computing

Fig. 5. Model Training Architecture

makes it challenging for existing methods and our CNN-TCN architecture to distinguish them apart. Fortunately, we observe that the private cars show period moving features (i.e., hidden temporal features) in geographic space. Motivated by this, given a private car's trajectory $T = \{p_1, p_2, ..., p_N\}$, we first extract its time sequence based on the temporal dimension, $T^{(t_s)} = \{p_1.t_s, p_2.t_s, ..., p_N.t_s\}$. Next, we employ the seasonal-trend decomposition procedure (STL) [7] to capture the period features hidden in $T^{(t_s)}$.

STL methods decompose the time series sequence to get trend data, seasonal data and residual data, which is denoted by TR_v, S_v, R_v, respectively. Based on these, the sequence data Y_v is formulated as: $Y_v = TR_v + S_v + R_v$.

Note that periodic data and seasonal data are quite similar, except that the time frequency of seasonal data fluctuations is fixed, while that of periodic data is not. Considering the time correlation among different trajectories of one moving object, we aim to extract the seasonal features of taxis and private cars S_v. The first step is detrending, which subtracts the trend component ${TR_v}^{(k)}$ in the previous iteration (i.e. k-th iteration) from the sequence data Y_v:

$$C_v^{k+1} = Y_v - {TR_v}^{(k)}, \tag{11}$$

Then, cycle-subseries smoothing is used to get the smoothing ${C_v}^{(k+1)}$ (i.e., temporary seasonal series) after extending the cycle-subseries forward and backward for one cycle respectively, while the cycle-subseries low-pass filtering is used to obtain ${L_v}^{(k+1)}$. Here, cycle-subseries is a sequence composed of points in the same position in each cycle. Based on the above illustration, the seasonal data ${S_v}^{(k+1)}$ is obtained:

$${S_v}^{(k+1)} = {C_v}^{(k+1)} - {L_v}^{(k+1)}, \tag{12}$$

In each iteration of inner loop, the seasonal data updates after a series of detrending and smoothing steps as below:

$$S_v^{(k+1)} = S_v^{(k)}, \tag{13}$$

Based on the above processing, we capture the period features (i.e. seasonal data in the trajectory) hidden in $T^{(t_s)}$, i.e., $T^{(t_s)} \rightarrow H$. Next, we add it to the raw trajectory T to get $\hat{T}$, which denotes the trajectory with additional period features H. For each point p_i in T, $\hat{p_i}.t_s$ in $\hat{T}$ is calculated as:

$$\begin{aligned} \hat{p_i}.t_s &= p_i.t_s + w \times H_i \\ (1 \leq i \leq &N, p_i \in T, H_i \in S_v). \end{aligned} \tag{14}$$

Finally, we replace the raw trajectory T with $\hat{T}$ when feeding trajectory to TCNs. This way, Estimator enables to capture the implicit mobility features between taxis and private cars, which significantly improves the classification performance.

TABLE I
STATICS OF EVALUATED DATASETS

Datasets	Number of Trajectories	Max. Length	Min. Length	Ave. Length
Walk	6,600	19837	20	250
Bike	3,787	22080	22	500
Bus	6,002	19605	24	450
Subway	1,017	21995	21	450
Private car	2,662	8395	21	400
Taxi	1,661	10841	19	450
Train	3,251	38294	21	800
Geolife	24,980	38294	19	450

C. Data Partition

We detail another optimization for CNN-TCN architecture, i.e., data partition and parallel computing.

Considering the varying traffic conditions in a city, (e.g., a car may drive fast in suburbs while drive slow in city blocks), we split the entire city region into the urban area, suburb, and urban center by administrative divisions and the traffic road layout. Based on this, we train CNN-TCN architecture in the three partitions to capture mobility features of different traffic conditions, as shown in Fig. 5(a). However, such serial model training in three partitions is inefficient. Therefore, we design a parallel training mechanism, as depicted in Fig.5(b). Specifically, we divide the training data into three sub-datasets based on their spatial locations. Note that, a trajectory may include thousands of GPS points and may cross more than one partitions. In this case, we assign this trajectory to the partition containing maximum number of GPS points. After the parallel model training, we collect and fuse the results (i.e., compute the union of the labeled or predicted transportation modes in each partition) from each partition, as we target the city-level transportation mode classification.

IV. EXPERIMENTS

We report on extensive experiments aimed at evaluating the performance of Estimator. We first present the experimental settings. Then, we compare the classification performance of Estimator with the state-of-the-arts. Next, we study the model efficiency and scalability. Finally, we report the effects of hidden mobility and district partition on classification performance. All implementation codes have been released online[1].

A. Experimental Settings

Datasets. We verify Estimator using Geolife dataset [41][2], which contains 17621 GPS points collected from 182 users. To the best of our knowledge, this is the only public GPS trajectory dataset with transportation mode labels [8], which has been widely used by existing studies of transportation mode classification of GPS trajectories [8], [25], [26]. Geolife dataset contains eleven labeled kinds of transportation modes. Nevertheless, due to the small percentage of the boat, run, airplane, and motorcycle, we mainly detect the remaining seven modes, i.e., walk, bike, bus, subway, private car, taxi, and train. As shown in Table I, we generate seven additional

[1]https://github.com/ZJU-DAILY/Estimator

[2]https://www.microsoft.com/en-us/download/details.aspx?id=52367

TABLE II
CLASSIFICATION PERFORMANCE (I.E., *ACC* AND *F1*) OF ALL THE METHODS

Methods			Geolife	Walk	Bike	Bus	Subway	Pri.Car	Taxi	Train
Machine Learning Based Methods	SVM	ACC	0.532	0.613	0.468	0.464	0.572	0.488	0.561	0.472
		F1	0.48	0.55	0.51	0.49	0.54	0.47	0.51	0.43
	kNN	ACC	0.579	0.802	0.445	0.978	0.934	0.524	0.703	0.533
		F1	0.53	0.78	0.78	0.13	0.84	0.41	0.52	0.42
	DT	ACC	0.694	0.732	0.556	0.433	0.721	0.345	0.597	0.556
		F1	0.58	0.76	0.77	0.46	0.68	0.50	0.54	0.57
	RF	ACC	0.783	0.842	0.396	0.982	0.911	0.845	0.833	0.645
		F1	0.62	0.81	0.78	0.44	0.76	0.59	0.61	0.58
Deep Learning Based Methods	SECA	ACC	0.768	0.746	0.824	0.756	0.732	0.818	0.822	0.856
		F1	0.76	0.81	0.84	0.75	0.70	0.78	0.77	0.80
	ST-GRU	ACC	0.912	0.882	0.843	0.924	0.931	0.897	0.789	0.842
		F1	0.88	0.88	0.92	0.90	0.91	0.84	0.82	0.86
	Estimator	ACC	**0.992**	**0.993**	**0.995**	**0.997**	**0.982**	**1.000**	**0.979**	**0.950**
		F1	**0.98**	**0.97**	**0.98**	**0.98**	**0.97**	**0.97**	**0.98**	**0.92**

TABLE III
MODEL EFFICIENCY EVALUATION

Dataset	SECA		ST-GRU		Estimator	
	TTIME	PTIME	TTIME	PTIME	TTIME	PTIME
Walk	9.36	0.30	51.47	2.34	**0.95**	**0.29**
Bike	10.73	0.29	53.41	2.73	**1.03**	**0.28**
Bus	10.11	0.29	57.16	2.31	**0.60**	**0.28**
Subway	5.79	0.29	34.53	1.36	**0.50**	**0.28**
Pri. Car	5.33	0.30	37.93	1.68	**0.60**	**0.28**
Taxi	3.57	0.29	31.63	1.45	**0.43**	**0.27**
Train	3.25	0.29	28.31	1.11	**0.37**	**0.27**
GeoLife	20.69	0.32	117.49	5.89	**3.23**	**0.30**

datasets from GeoLife (each dataset contains one transportation mode). Based on this, we train and test Estimator on eight datasets (i.e., Walk, Bike, Bus, Subway, Private Car, Taxi, Train and Geolife) respectively, in order to better explore the performance of Estimator for each transportation mode and prove it is optimal for discovering both single transportation mode and multiple transportation modes.

Baselines. We compare Estimator with representative traditional machine-learning based methods (i.e., kNN, RF, SVM, and DT) and the state-of-the-art deep-learning based methods (i.e., ST-GRU and SECA). Specifically, the baselines are:

- **kNN [39]**, a classic method to distinguish different transportation modes using trajectory similarity computation;
- **RF [38]**, a classifier that uses multiple trees to identify different trajectories;
- **SVM [3]**, a linear classifier for classification defined according to the largest interval in the feature space;
- **DT [40]**, a classifier that constructs a decision tree to discover the classification rules hidden in trajectory data;
- **SECA [8]**, a semi-supervised convolutional autoencoder to learn the spatio-temporal features of trajectories for transportation mode classification;
- **ST-GRU [26]**, a GRU-based model with a 1D-CNN to capture the local correlations, which offers the state-of-the-art results for transportation mode classification.

All baselines are trained using the hyperparameters that achieves the highest performance. Here, we list the hyperparameters corresponding to each deep-learning methods.

SECA [8]. We set the number of GPS points to 20, the total distance of a segment to 150 meters and total duration time of the segment less than 1 minutes. In addition, the hyperparameters α and β are initialize to 1 and are both varied from 1 to 0.1 during training.

GRU [26]. We set the soft-embedding dimension to 16 and the segment length to 10. In ST-GRU model, the kernel size in all 1D-convolution operators is set to 3. The model is trained by optimising the cross-entropy loss function on the labels, where the learning rate is set to 0.01. In addition, the model is optimised using Adam optimiser for the first 30 epochs and is finely tuned using SGD optimiser for 50 epochs, where the learning rate is set to 0.0001.

Preprocessing. The preprocessing mainly involves trajectory mapping. Existing methods [11], [34] map the raw trajectories into gray images to extract features representing spatial coordinates and stay time. We enhance this strategy by mapping raw trajectories into color images and extract features representing spatial coordinates, stay time, azimuth, and average speed. The transformed color images are the input of the CNN model.

Hyperparameters. We set the hyperparameters based on the model performance. Specifically, we set the granularity of the grid cell as 40×40. The number of hidden layers is 8 and the number of hidden units of TCN is 25. We adopt Adam optimizer [20] for model training. In addition, the batch size is set to 64, the epoch is set to 20, the gradient clip is set to (i.e., no clip), the dropout is set to 0.05, and the initial learning rate is set to 0.002. We use 80% and 20% of each mode dataset for training and testing for OCT-LSTM, respectively. We implement Estimator in Python and Pytorch. All experiments are conducted on a server with GeForce RTX 3090, 2.40GHz GPU, and 24-GB RAM.

Evaluation metrics. Following most of existing trajectory classification studies [2], we use *ACC* and $F1$ to evaluate the quality, and use *TTIME* and *PTIME* to evaluate the efficiency. $ACC = \frac{100\%}{N}\sum_{i=1}^{N}\left(1-\left|\frac{y_i-\hat{y}_i}{y_i}\right|\right)$, where y_i is the actual transportation mode (i.e., ground truth) of trajectory T_i, $\hat{y}_i$ denotes the predicted transportation mode of trajectory T_i, and N is the number of trajectories. *ACC* can also BE derived by $\frac{TP+TN}{TP+TN+FN+FT}$, where *FP*, *FN*, *TP*, *TN* denote the number of false positive samples, false negative samples, true positive samples, and true negative samples, respectively. $F1 = \frac{2*Precision*Recall}{Precision+Recall} = \left(\frac{2}{2+\frac{FP}{TP}+\frac{FN}{TP}}\right)$, where $Precision = \frac{TP}{TP+TN}$ and $Recall = \frac{TP}{TP+FN}$. *TTIME* and *PTIME* are the training and testing time, respectively, which measure the efficiency of transportation mode classification methods. Obviously, the larger *ACC* and *F1* are, the better classification quality is; and the smaller *TTIME* and *PTIME* are, the better classification efficiency is. Note that,

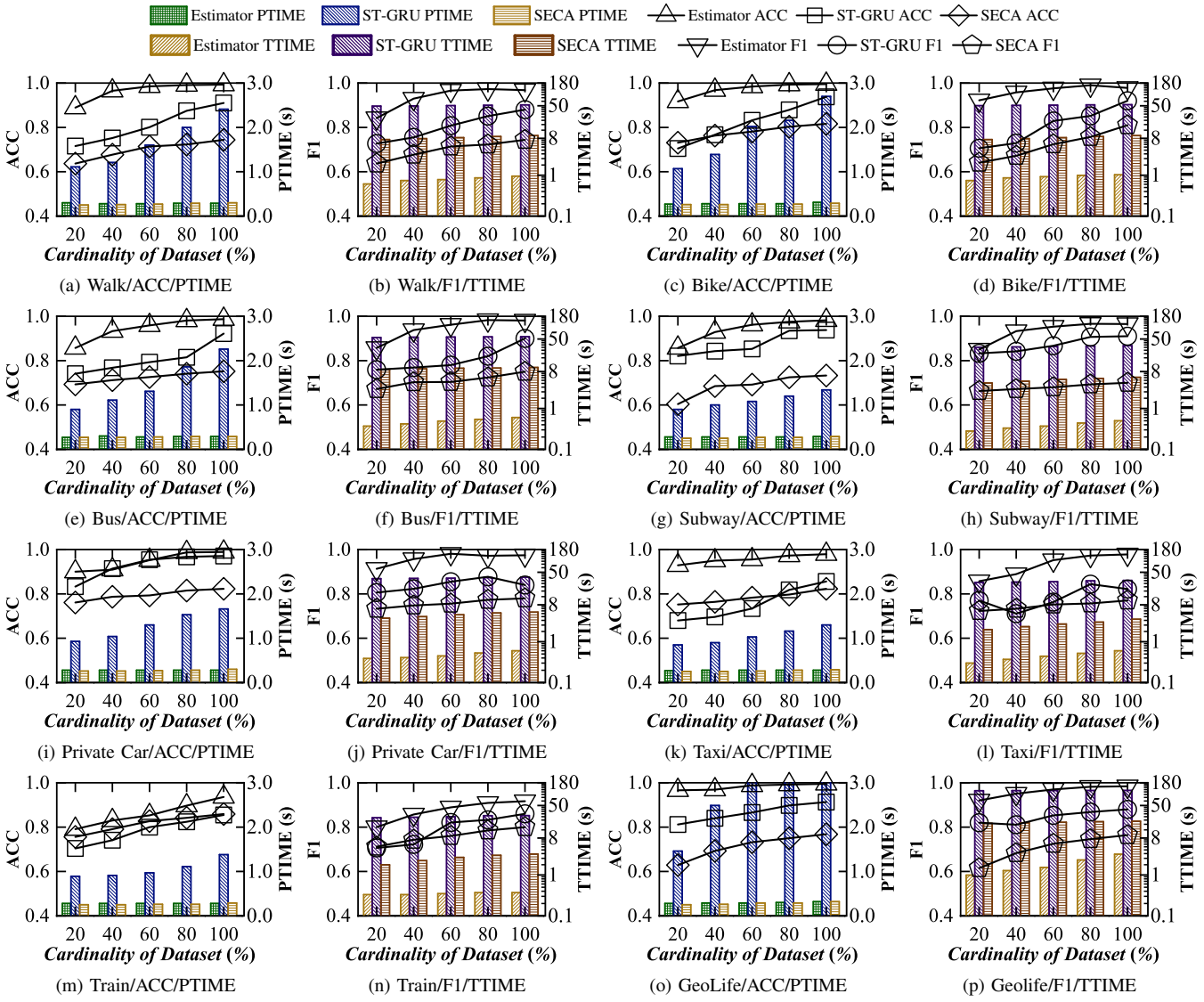


Fig. 6. Model Scalability Evaluation vs. Cardinality of Datasets

we perform binary classification on seven datasets with single mode, while use multi-class classification on Geolife dataset.

B. Classification Performance Evaluation

We compare Estimator and all baselines in terms of the classification performance (i.e., *ACC* and *F1*). Table II reports the results. As observed, deep-learning based methods perform better than machine-learning based methods. This is because, machine-learning based methods rely on manual feature extraction and cannot capture the non-linear dependencies in trajectory data. Specifically, kNN and DT perform worst in all of the baselines, as they simply treat trajectories as time-series sequences, ignoring the temporal correlations between trajectory points. On the other hand, Estimator achieves the best performance in all the cases. This is because, compared with other deep-learning based methods, Estimator considers hidden mobility features and varying traffic conditions that enable embedding and learning more effectively. Based on this, it is easier for Estimator to identify transportation mode via corresponding captured features of different trajectories.

C. Model Efficiency Evaluation

We proceed to evaluate the model efficiency in terms of both the training phase (i.e., *TTIME*) and predicting/testing phase (i.e., *PTIME*), using the eight datasets. Here, we only compare Estimator with start-of-the-art deep-learning based methods because the performance of trajectory mode classification of the machine-learning based methods is low. The results are presented in Table III. The first observation is that Estimator outperforms all the competitors in both training and testing phases. In the training phase, Estimator runs up to 40 times faster than the state-of-the-art competitor ST-GRU. In the classification phase, Estimator achieves up to 94.6% efficiency improvement on Geolife dataset. The second observation is that the efficiency of RNN-based ST-GRU is much lower than that of convolution based methods (i.e., Estimator and SECA). This is because RNN processes trajectory points one by one, while convolution methods process the trajectory in parallel.

D. Model Scalability Evaluation

We study the scalability of Estimator by comparing with SECA and ST-GRU, where the data size of each dataset is

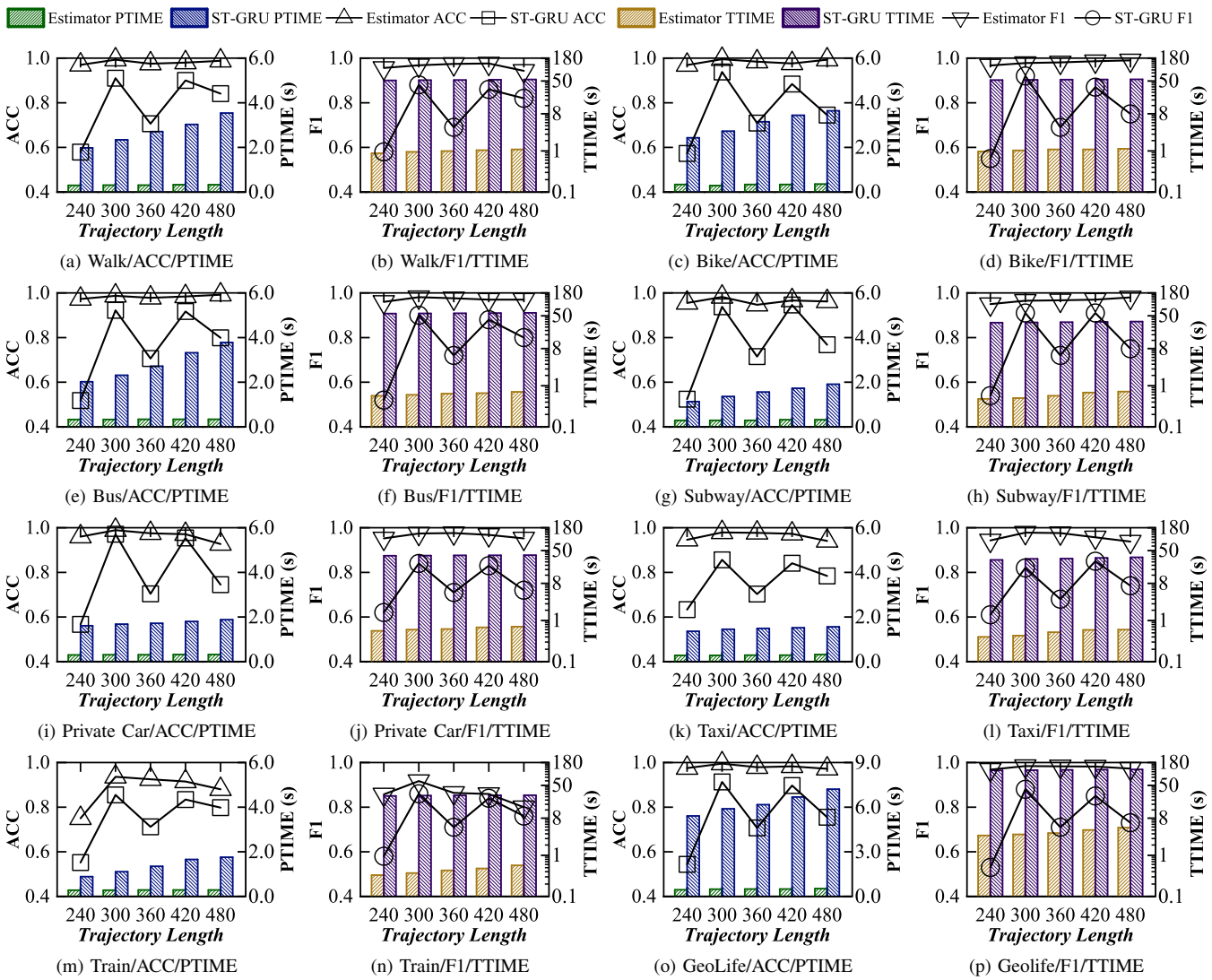


Fig. 7. Model Scalability Evaluation vs. Trajectory Length

varied from 20% to 100% of the total dataset size.

Fig. 6 depicts the experimental results by varying cardinality of datasets from 20% to 100%, where the percentages denote the 20%, 40%, 60%, 80%, 100% of the entire dataset. Note that, we keep the origin spatial region when sampling 20%, 40%, 60%, 80%, 100% trajectories from the origin dataset. We use *ACC*, *F1*, *TTIME* and *PTIME* to evaluate the quality and efficiency. The first observation is that *ACC* and $F1$ increase with the growth of data cardinality. The reason is that the model becomes better with the increasing amount of training data. Second, Estimator achieves the highest *ACC* and $F1$. This is because we not only consider the trajectories' explicit features but also explore their implicit features, where the model learns comprehensive information and thus classification is improved. The third observation is that *TTIME* and *PTIME* of ST-GRU increase rapidly with the growth of cardinality, because more training and testing data are used. However, *TIME* and *PTIME* of SECA and Estimator increase slightly with the growth of cardinality. Moreover, SECA and Estimator have much smaller *TTIME* and *PTIME* than ST-GRU model. This is because GRU (RNN) model sequentially processes trajectories, which degrades the parallel processing. On the other hand, SECA and Estimator utilize convolution layers to process trajectories in parallel. Overall, Estimator shows high scalability on both quality and efficiency, which proves its capability of performing large-scale classification.

Fig. 7 reports the results when varying the trajectory length from 240 to 480. Here, trajectory length denotes the average number of points of a trajectory. Note that SECA develops a segmentation method, which is unable to pre-define the trajectory length. Thus, we only compare Estimator with ST-GRU in this set of experiments. The first observation is that Estimator achieves the best performance. This is because, when the trajectory length is fixed, Estimator can extract more features of trajectories than ST-GRU. In the following experiments, we fix the trajectory length to 300 points for Estimator to achieve the best performance. Second, ACC and $F1$ of ST-GRU fluctuate drastically as the trajectory length increases. Specifically, ST-GRU achieves highest ACC and $F1$ when the trajectory length equals to 300 on Walk, Bike, Bus, Private Car, Taxi, Train and Geolife datasets, and achieves highest ACC and $F1$ when the trajectory length equals to 420 on Subway dataset. This is because, on the one hand, ST-GRU can extract more deep features with the increasing length of data; on the other hand, increasing data lengths require larger receptive field and more dilated convolutional layers,

TABLE IV
ABLATION STUDY OF ESTIMATOR VS. ACC, F1

Variants	Geolife		Walk		Bike		Bus		Subway		Pri.Car		Taxi		Train	
	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1	ACC	F1
CNN Model	0.731	0.722	0.352	0.331	0.867	0.846	0.771	0.745	0.345	0.352	0.495	0.472	0.513	0.484	0.243	0.241
CNN-TCN-B	0.964	0.913	0.959	0.934	0.961	0.941	0.969	0.944	0.873	0.887	0.946	0.918	0.946	0.923	0.708	0.710
CNN-TCN-H	0.984	0.939	0.978	0.947	0.962	0.941	0.969	0.944	0.875	0.887	0.973	0.918	0.961	0.930	0.708	0.710
CNN-TCN-P	0.992	0.974	0.993	0.968	0.995	0.978	0.987	0.981	0.982	0.950	0.989	0.964	0.979	0.938	0.936	0.857

TABLE V
ABLATION STUDY OF ESTIMATOR VS. TTIME, PTIME

Variants	Geolife		Walk		Bike		Bus		Subway		Pri.Car		Taxi		Train	
	TTIME	PTIME	TTIME	PTIME	TTIME	PTIME	TTIME	PTIME	TTIME	PTIME	TTIME	PTIME	TTIME	PTIME	TTIME	PTIME
CNN Model	14.34	4.53	7.41	2.23	7.37	2.22	7.47	2.21	7.23	2.21	7.66	2.24	7.62	2.2	7.18	2.16
CNN-TCN-B	3.26	0.31	0.96	0.27	1.04	0.31	0.64	0.32	0.52	0.30	0.62	0.29	0.61	0.29	0.37	0.29
CNN-TCN-H	3.25	0.30	0.95	0.27	1.04	0.31	0.63	0.32	0.52	0.30	0.6	0.29	0.6	0.28	0.37	0.29
CNN-TCN-P	3.23	0.30	0.95	0.27	1.03	0.29	0.63	0.29	0.5	0.29	0.6	0.28	0.6	0.28	0.37	0.28

leading to more parameters and higher model complexity when capturing the features of a trajectory. This leads to overfitting, which degrades the model performance. By contrast, ACC and $F1$ of Estimator are stable, which proves its robustness. In addition, the training time (*TTIME*) and testing time (*PTIME*) increase with the growth of trajectory length, as more sample points need to be processed. Overall, this set of experiments proves that Estimator is able to perform large-scale transportation mode classification.

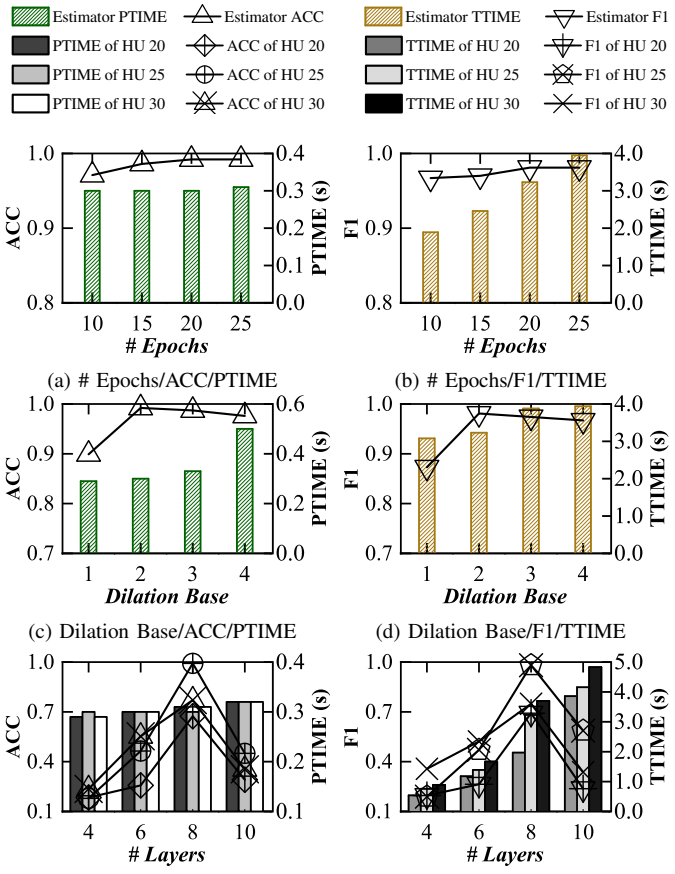
E. Ablation Study

We study the effectiveness of four key components (i.e., CNN model, CNN-TCN based model, hidden features extraction, data partition and parallel computing) embedded in Estimator. Table IV and Table V report the results on eight datasets, where CNN-TCN-B denotes the CNN-TCN based model, CNN-TCN-H denotes CNN-TCN-B with hidden features extraction, and CNN-TCN-P denotes CNN-TCN-H with data partition and parallel computing. Obviously, CNN-TCN-P is Estimator essentially.

The first observation is that CNN-TCN-B outperforms CNN on all the datasets. This is because, TCN component can greatly improve the CNN model performance in terms of both quality and efficiency for transportation mode classification, which verifies the superior performance of TCN model when processing trajectories. The second observation is that CNN-TCN-H outperforms CNN-TCN-B in terms of *ACC* and $F1$ especially on *Geolife*, *taxi* and *private car* datasets. This is because, employing the hidden periodic features between taxis and private cars can improve the model performance for transportation mode classification. Finally, CNN-TCN-P outperforms all the variants of Estimator in terms of *ACC* and $F1$, because it considers the different traffic conditions in the city. Note that, *ACC* and $F1$ are enhanced significantly on *train* dataset in terms of *ACC* and $F1$. The reason is that urban railways are mainly located in the suburbs, and data partition (i.e., urban area, suburb, city center) can reduce the negative effects of other traffic modes on the classification. Overall, this set of experiments validate the effectiveness of four key components of Estimator.

F. Model Hyperparameters Analytics

We evaluate the performance of Estimator under different settings of hyperparameters, including the number of epochs, dilation base, the number of network layers, and the number of hidden units in each layer. Here we only report the experimental results on GeoLife dataset, because the performance on other datasets is similar to that on GeoLife dataset. Note that, the number of epochs used for training reflects the convergence's speed of the model; dilation base used in dilation convolutions reflects the receptive field in TCN; while the number of network layers and the number of hidden units in each layer are also important hyperparameters in deep learning, which reflects the network depth and the capability of capturing features.



(a) # Epochs/ACC/PTIME (b) # Epochs/F1/TTIME
(c) Dilation Base/ACC/PTIME (d) Dilation Base/F1/TTIME
(e) # Layers and Units/ACC/PTIME (f) # Layers and Units/F1/TTIME

Fig. 8. Model Evaluation of Various TCN Hyperparameters

Figs. 8(a) and 8(b) show the results when varying the number of epochs. As observed, *ACC* and $F1$ first increase with the growth of number of epochs and then keep stable

(i.e., the model converges). Note that, Estimator performs well in terms of *ACC* and $F1$ (i.e., more than 0.95) even when the number of epochs is 10. In addition, the training time *TTIME* increases slightly with the growth of number of epochs, while the testing time *PTIME* is relatively stable (i.e, always around 0.3s). This verifies the robustness of Estimator.

Figs. 8(c) and 8(d) show the results under various dilation coefficients. As the upward growth dilated coefficient can be computed as $d = b^i$, we set $b = 1, 2, 3, 4$ to adjust the dilation coefficient. The first observation is that, Estimator achieves the smallest *ACC* and $F1$ when $b = 1$. This is because that the receptive field does not been enlarged if the dilated coefficient is fixed to 1, which fails to capture the relevance between nodes with a long interval. The second observation is that Estimator performs the best when $b = 2$. This is because when $b = 3$ or 4, the receptive field becomes too large. In this case, some local detailed features are lost, which causes negative effects on classification. In terms of efficiency, the training time *TTIME* first increases and then remains stable with the growth of b. This is because the receptive field of TCNs has covered the entire input trajectory when b reaches 3, and there is no additional data to be trained. The testing time *PTIME* first slightly increases as b grows from 1 to 3 but significantly increases when b grows from 3 to 4. This is because, when b grows from 3 to 4, the model processes lots of redundant data due to the large receptive field.

Figs. 8(e) and 8(f) show the results under various numbers of hidden units (denoted as HU = 20, 25, 30) when varying the number of convolutional layers from 4 to 10. The first observation is that, *ACC* and $F1$ first increase and then drop with the growth of the number of layers. The reason is that, on the one hand, more layers reduce the network errors and improve the accuracy; on the other hand, a large number of layers lead to over-fitting. The second observation is that *ACC* and $F1$ are not linearly correlated with the number of hidden units. This is because, given different numbers of network layers, the number of hidden units that achieves the best performance varies. In addition, training time *TTIME* increases slowly with the growth of numbers of layers and units, while the testing time *PTIME* is stable (i.e., always around 0.3s), which suggests the robustness of Estimator. As observed, Estimator achieves the highest *ACC*, i.e., 99.2%, and the highest $F1$, i.e., 0.981, when the depth of neural networks (i.e., the number of network layers) is 8 and the number of hidden units is 25. Hence, we set the number of convolutional layers to 8 and the number of hidden units in each layer to 25 for Estimator in order to achieve the best performance.

V. Related Work

We proceed to review the related work on transportation mode classification, including machine learning based methods and deep learning based methods.

A. Machine Learning based Classification

Earlier proposals of machine learning based trajectory mode classification typically contain three processing phases: trajectory representation, feature extraction, and classifier selection. First, a trajectory can be represented as different formats [37], such as a series of timestamped sample points, a sequence of trajectory segments, and a set of image pixels. Next, machine learning based methods manually extract spatio-temporal features from raw trajectories or trajectory images, which is called feature extraction. As an example, Xu et al. [33] extract the characteristics of every sample point in a trajectory considering both temporal information (e.g., stay time) and spatial information (e.g., speed and acceleration). Similarly, Dabiri et al. [8] transform each trajectory segment into a 4-channel characteristics tensor composed of the jerk, speed, acceleration, and relative distance. To extract features from mapped trajectory images, Endo et al. [11] capture time interval characteristics and geographic location information of the trajectories with grid pixels and then use existing image classification models to classify transportation modes. Finally, these mobility features captured from either raw trajectories or trajectory images are fed into existing popular machine learning classifiers such as k-nearest neighbour (kNN) [39], Hidden Markov Model (HMM) [16], Random Forest (RF) [38], and Support Vector Machine (SVM) [3]. However, they require to extract features manually, which is time-consuming, noise sensitive, and fails to capture the non-linear and hidden dependencies of trajectories. Thus, their performance is limited.

B. Deep Leaning based Classification

Deep neural networks have been widely used in text classification [9], [29] and image classification [19], [32], as it can be performed without manual feature extraction. This leads to the proposals of deep learning based transportation mode classification, including CNN-based methods, RNN-based methods, and CNN-RNN based methods. The CNN-based methods typically use convolution neural networks (CNNs) to capture moving object features. Specifically, Endo et al. [8] convert each trajectory to an image. An image is composed of a set of grids, each of which represent its geographic dependencies. Based on this, the trajectory classification problem is transformed to the image classification problem.

There are several recent works that capture trajectory features by employing spatio-temporal graph attention networks (i.e, GNN-based models) [13], [21]. Note that, spatio-temporal graph can only be applied to road network. However, the movements of some moving objects (e.g., people and bikes) do not follow network constraints. Thus, spatio-temporal graph is mainly applied for traffic prediction, and fails to support transportation mode classification. CNN-based methods only focus on the spatial aspect of trajectories while ignore the temporal correlations between trajectory points. To improve the performance of classification, recurrent neural networks (RNNs) are employed to capture time-series features of trajectories [14]. This is because, the RNN-type models are initially designed for time-series representation learning, which is easy to be adapted to trajectory classification [10]. Compared with CNN-based methods, RNN-based methods attach more importance to the temporal aspect than the spatial aspect.

Recently, the state-of-the-art CNN-RNN based architecture has been developed, which feeds trajectories into both CNN

models and RNN models simultaneously to capture both spatial and temporal mobility features of moving trajectories. Liu et al. [25] present an end-to-end framework based on Bi-LSTM model for transportation mode classification. Specifically, they design a spatio-temporal GRU model combined with CNN for classification analyses [26]. Friedrich et al. [14] combine LSTM and CNN for transportation mode classification of trajectories from smartphone sensors. As we mainly focus on GPS-based trajectories (cf. Section II), we only compared the methods using GPS-based trajectories. Although CNN-RNN based models capture the spatio-temporal information of trajectories, RNNs can only scan a trajectory one by one, which is sub-optimal for parallel processing [35]. Consequently, existing RNNs involved methods (including both RNN-based and CNN-RNN based) are not able to support large-scale trajectory/transportation classification. Motivated by the parallel computing of CNNs, temporal convolutional networks (TCNs) [1] are proposed, which enable reading and embedding time-series sequences in a parallel manner. TCN models have achieved great success in many time-series learning tasks such as action detection [22], probability predictiuon [6], and machine translation [1]. To the best of our knowledge, this is the first time to apply TCNs to transportation mode classification tasks.

VI. Conclusions

In this paper, we propose an effective and scalable framework, Estimator, for transportation mode classification over GPS trajectories. Estimator is able to capture hidden spatial-temporal features of trajectories. To improve classification effectiveness, Estimator exacts hidden features, and considers the varying traffic conditions. To improve classification efficiency and scalablity, Estimator combines CNN and TCN to parallel learn the hidden features. Extensive experiments conducted on eight real-life datasets confirm that Estimator is able to outperform the state-of-the-art methods in terms of both effectiveness and efficiency. In the future, it is of interest to extend Estimator to other mining tasks such as road planning and transportation emissions detection.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *CoRR*, abs/1803.01271, 2018.

[2] J. Bian, D. Tian, Y. Tang, and D. Tao. Trajectory data classification: A review. *TIST*, 10(4):33:1–33:34, 2019.

[3] A. Bolbol, T. Cheng, I. Tsapakis, and J. Haworth. Inferring hybrid transportation modes from sparse GPS data using a moving window SVM classification. *Comput. Environ. Urban Syst.*, 36(6):526–537, 2012.

[4] H. Chen, H. Yin, T. Chen, Q. V. H. Nguyen, W. Peng, and X. Li. Exploiting centrality information with graph convolutions for network representation learning. In *ICDE*, pages 590–601, 2019.

[5] X. Chen, J. Pang, and R. Xue. Constructing and comparing user mobility profiles. *TWEB*, 8(4):21:1–21:25, 2014.

[6] Y. Chen, Y. Kang, Y. Chen, and Z. Wang. Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, 399:491–501, 2020.

[7] R. B. Cleveland and W. S. Cleveland. Stl: A seasonal-trend decomposition procedure based on loess. *JOFFSTAT*, 6(1), 1990.

[8] S. Dabiri, C. Lu, K. Heaslip, and C. K. Reddy. Semi-supervised deep learning approach for transportation mode identification using GPS trajectory data. *TKDE*, 32(5):1010–1023, 2020.

[9] B. Dai, J. Li, and R. Xu. Multiple positional self-attention network for text classification. In *AAAI*, pages 7610–7617, 2020.

[10] J. L. Elman. Finding structure in time. *CogSci*, 14(2):179–211, 1990.

[11] Y. Endo, H. Toda, K. Nishida, and J. Ikedo. Deep feature extraction from trajectories for transportation mode estimation. *PAKDD*, pages 54—66, 2016.

[12] J. Feng, Y. Li, C. Zhang, F. Sun, F. Meng, A. Guo, and D. Jin. Deepmove: Predicting human mobility with attentional recurrent networks. In *WWW*, pages 1459–1468, 2018.

[13] J. Feng, Y. Li, C. Zhang, F. Sun, F. Meng, A. Guo, and D. Jin. Deepmove: Predicting human mobility with attentional recurrent networks. In *WWW*, pages 1459–1468, 2018.

[14] B. Friedrich, B. Cauchi, A. Hein, and S. J. F. Fudickar. Transportation mode classification from smartphone sensors via a long-short-term-memory network. *CoRR*, abs/1910.04739, 2019.

[15] B. Friedrich, C. Lübbe, and A. Hein. Combining LSTM and CNN for mode of transportation classification from smartphone sensors. In *ISWC*, pages 305–310, 2020.

[16] A. Y. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng. Deep speech: Scaling up end-to-end speech recognition. *CoRR*, abs/1412.5567, 2014.

[17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[18] S. Heldens, N. Litvak, and M. van Steen. Scalable detection of crowd motion patterns. *TKDE*, 32(1):152–164, 2020.

[19] A. Imran, C. Huang, H. Tang, W. Fan, Y. Xiao, D. Hao, Z. Qian, and D. Terzopoulos. Self-supervised, semi-supervised, multi-context learning for the combined classification and segmentation of medical images (student abstract). In *AAAI*, pages 13815–13816, 2020.

[20] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[21] Y. H. Lau and R. C. Wong. Spatio-temporal graph convolutional networks for traffic forecasting: Spatial layers first or temporal layers first? In *SIGSPATIAL*, pages 427–430, 2021.

[22] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager. Temporal convolutional networks for action segmentation and detection. In *CVPR*, pages 1003–1012, 2017.

[23] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W. Ma. Mining user similarity based on location history. In *ACM-GIS*, page 34, 2008.

[24] T. Li, L. Chen, C. S. Jensen, and T. B. Pedersen. TRACE: real-time compression of streaming trajectories in road networks. *PVLDB*, 14(7):1175–1187, 2021.

[25] H. Liu and I. Lee. End-to-end trajectory transportation mode classification using bi-lstm recurrent neural network. In *ISKE*, pages 1–5, 2017.

[26] H. Liu, H. Wu, W. Sun, and I. Lee. Spatio-temporal GRU for trajectory classification. In *ICDM*, pages 1228–1233, 2019.

[27] S. Ma, Y. Zheng, and O. Wolfson. T-share: A large-scale dynamic taxi ridesharing service. In *ICDE*, pages 410–421, 2013.

[28] J. D. Mazimpaka and S. Timpf. Trajectory data mining: A review of methods and applications. *SIS*, 13(1):61–99, 2016.

[29] A. Moreo, A. Esuli, and F. Sebastiani. Learning to weight for text classification. *TKDE*, 32(2):302–316, 2020.

[30] J. Paparrizos and M. J. Franklin. GRAIL: efficient time-series representation learning. *VLDB*, 12(11):1762–1777, 2019.

[31] J. Sun, Y. Li, H. Fang, and C. Lu. Three steps to multimodal trajectory prediction: Modality clustering, classification and synthesis. In *ICCV*, pages 13230–13239, 2021.

[32] Y. Wang, D. He, F. Li, X. Long, Z. Zhou, J. Ma, and S. Wen. Multi-label classification with label graph superimposing. In *AAAI*, pages 12265–12272, 2020.

[33] D. Xu, W. Cheng, D. Luo, X. Liu, and X. Zhang. Spatio-temporal attentive RNN for node classification in temporal attributed graphs. In *IJCAI*, pages 3947–3953, 2019.

[34] X. Yang, K. Stewart, L. Tang, Z. Xie, and Q. Li. A review of GPS trajectories classification based on transportation mode. *Sensors*, 18(11):3741, 2018.

[35] Z. Yu and G. Liu. Sliced recurrent neural networks. In *COLING*, pages 2953–2964, 2018.

[36] H. Yuan and G. Li. A survey of traffic prediction: from spatio-temporal data to intelligent transportation. *Data Sci. Eng.*, 6(1):63–85, 2021.

[37] Y. Zheng. Trajectory data mining: An overview. *TIST*, 6(3):29:1–29:41, 2015.

[38] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W. Ma. Understanding transportation modes based on GPS data for web applications. *TWEB*, 4(1):1:1–1:36, 2010.

[39] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W. Ma. Understanding mobility based on GPS data. In *UbiComp*, volume 344, pages 312–321, 2008.
[40] Y. Zheng, L. Liu, L. Wang, and X. Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *WWW*, pages 247–256, 2008.
[41] Y. Zheng, L. Wang, R. Zhang, X. Xie, and W. Ma. Geolife: Managing and understanding your past life over maps. In *MDM*, pages 211–212, 2008.