**Description for codes and data**

**Inferring transportation modes from GPS trajectories using a Convolutional Neural Network**

**Sina Dabiri a,\*, Kevin Heaslip a**

a Charles E. Via, Jr. Department of Civil and Environmental Engineering, 301, Patton Hall, Virginia Tech, Blacksburg, VA 24061, United States

\* Corresponding Author

Email Addresses: [sina@vt.edu](mailto:sina@vt.edu), kheaslip@vt.edu

* **All codes in this folder have been implemented in the Python. The main required libraries include: geopy, sklearn, tensorflow, keras, numpy, and math.**
* **Please read all the following comments step-by-step before using the codes and data inside this folder. Furthermore, in each Python file, follow the comments for a better understanding of the flow and commands of the Python file.**

1. The Geolife GPS trajectory data set version 1.3 was collected through 182 users. In the user guide of the Geolife data set, it has been mentioned that 73 users have labeled their trajectories with transportation mode. However, in the latest version of 1.3, only 69 users with transportation labels were found. Two files are associated with these users. One is the ‘Trajectory’ folder, which contains a lot of GPS trajectory files with the ‘plt’ file extension. For each user, we combine all GPS files to one file with name: ‘CombinedID.plt’, where ID indicates the number associated with the user. For example, ‘Combined10’ includes all GPS trajectory files of the user number 10. The text file ‘labels.txt’ is another user’s file that stores the transportation labels corresponding to GPS files in the ‘Trajectory’ folder. We change the name of ‘labels.txt’ to ‘labelsID.txt’ so every user has a unique label file. All ‘CombinedID.plt’ and ‘labelsID.txt’ files for 69 users are stored in the following link:

https://drive.google.com/drive/folders/0B\_IlYd4BWRmBTXdPSjVGemFpV1E?usp=sharing

Before proceeding to the step 2, please download the main dataset from the above link.

1. Use the ‘LabelMatrix-TimeDays-TrajectrotyMatrix.py’ Python file. The input for this file is all GPS trajectories and labels files for 69 users from the downloaded folder in the step 1. This Python file first converts the starting time and ending time in the labels files to the number of days to be consistent with the date format in the GPS files. Since not all GPS logs for each user have been annotated with a label, this Python file detects the GPS logs which have been annotated by the user and assign the appropriate label to each GPS log. At the end, the output of this Python file is a list of arrays. Each array, which is associated with a user, is a collection of GPS logs, where each GPS log consists of four columns: latitude, longitude, date, and transport mode label.
2. Use the ‘Instance\_Creation’ Python file. Using the output of step 2 as the input of this python file, this code first creates the fixed-size segments. Then, it calculates the motion characteristics (e.g., speed, acceleration, jerk, and bearing rate for each GPS log). Then, all data preprocessing steps described in the paper are applied to create GPS segments, which are then fed into the CNN architecture. In other words, the input layer for the CNN architecture is built in this Python file.
3. Use the ‘Keras\_Data\_Creation.py’ Python file. This Python code creates the input layer for the CNN architecture in such a way to be compatible with the sequential models in the Keras library.
4. Use the ‘CNN-Keras.py’ Python file. This Python code creates a CNN architecture using the Keras sequential model. This files creates the best CNN configuration in the paper, i.e., model E. However, a user can change the layer patterns to create other configurations as well. The confusion matrix is also obtained in this file.
5. Use the ‘CNN-Ensemble-Keras.py’ Python file. This Python code is the same as the model in step 5 but creates the ensemble of 7 best CNNs.
6. Use the ‘HandCrafted\_features.py’ Python file. This Python code uses the output of step 3 to create 11 hand-crafted features for feeding into classical machine learning algorithms.
7. Use the ‘Classical\_ML Methods.py’ Python file. This Python code uses the output of step 7 to first train and tune the hyperparameters of four classical machine learning algorithms including KNN, DT, SVM, RF, and MLP. Then, the test accuracy, average precision, average recall, and average F-score of these models are calculated and reported in the paper.
8. Use ‘CNN-A-For Comparison.py’ Python file. This Python code trains and tunes the shallowest and simplest CNN configuration for comparison with other classical learning algorithms. The performance metrics of the model are also computed in this file.
9. Use the ‘Figure-TestTrain Accuracy-epoch.py’ Python file. This Python code uses one of the outputs in the step 5 to plot Fig. 3 in the paper.

Please contact with us through [sina@vt.edu](mailto:sina@vt.edu) if you have any question regarding the data and codes.

Thanks,

Sina