Implementation Of A TTE System

First Report

11812804 董　正
11813225 王宇辰
11811305 崔俞崧

Supervisior: 宋轩

Department of Computer Science and Engineering

Apr. 2021

Content

1 Preliminaries

1.1 Review

1.1.1 TTE

Travel Time Estimation (TTE) is one of the most important researching topic in the traffic forecasting field. Estimating the travel time of any path in a city is of great importance to traffic monitoring, route planning, ridesharing, taxi dispatching, etc. On Sep. 2020, DeepMind published a blog named *Traffic prediction with advanced Graph Neural Networks*. This blog briefly described the whole industrial structure of estimated times of arrival (ETAs) techniques applied in Google Map but did not given any detailed implementation or any code. Our work is based on the model structure of TTE proposed in the blog.

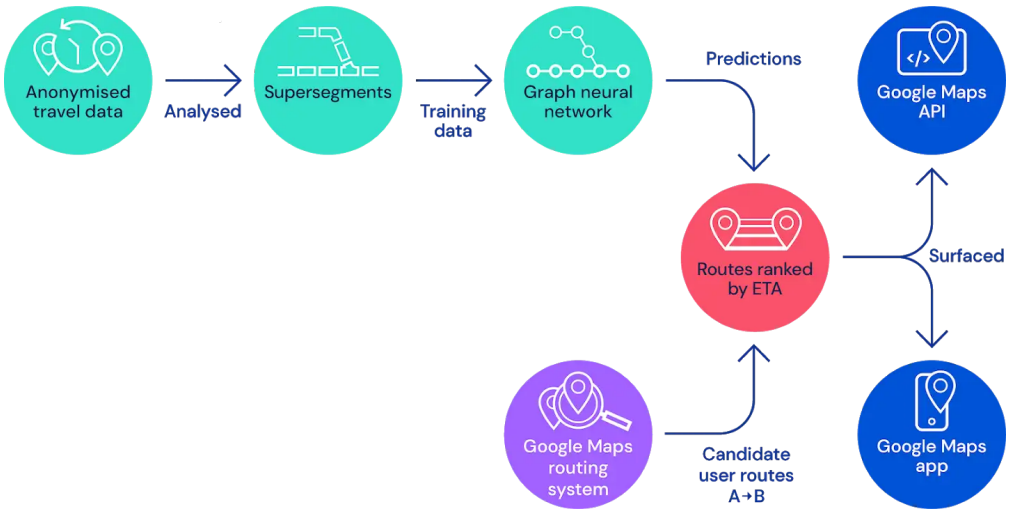


Figure 1: Architecture

1.1.2 Goal

Our ultimate goal (tentative) is to implement the industrial structure and apply it to the open source databases in China, then compare the performance with the state-of-the-art structures and find its application value. This semester, we will implement a TTE system base on the work we done in the last term, combining *supersegment* and *DeepTTE*. We will try to work out an interactive application with graphical user interface.

1.2 Introduction

In the last stage, we have processed some open source data. And we got deep in the code of *DeepTTE* and researched on a new concept called **Travel Time Index (TTI)**. In addition, we proposed a new computing process of *Supersegment* and gave a simple demo.

Breifly, we will state our work in this report as

- xxx by 王宇辰
- xxx by 崔俞崧
- Code implementation of *Supersegment* by 董正

2 xxx

3 xxx

4 Code Implementation of Supersegment

4.1 Dataset

- Data source: Didi Chuxing GAIA Intiative
- Region: Chengdu
- Time: 2018-10-01 to 2018-12-01
- Content:
  - GPS track of taxis with timestamps
  - Coordinate of road and district boundaries
  - TTI and average speed of districts

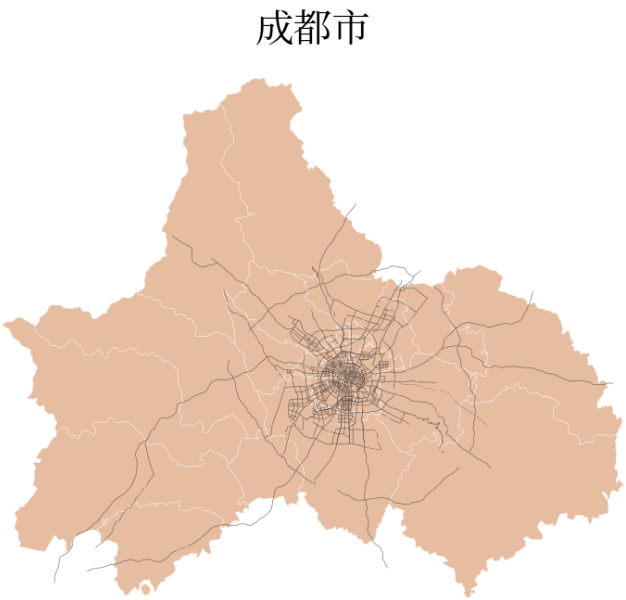


Figure 2: Cheng Du

4.2 Model Design

In this section, I will propose a model to compute Supersegments. At the very first, I need to ensure that I will base on **speed** to do the division. There is a three-step approach of my design:

1. Locate GPS coordinates of taxis into corresponding roads
2. Use the timestamps to calculate average speed and regard it as the instantaneous speed of midpoint
3. Apply clustering algorithm to these midpoints
4. Intersect road and clusters to get segments

First, since we know the boundaries of the roads, we can determine the coordinate is on which road. After that, we draw the GPS points on the road. Because we know the timestamp of each point, we can calculate the average speed of two adjacent points and just consider it as the speed of their midpoint. Note that the closer the points are, the closer the average speed is to the actual instantaneous speed. So the best way is to select adjacent points if our data is abundant.

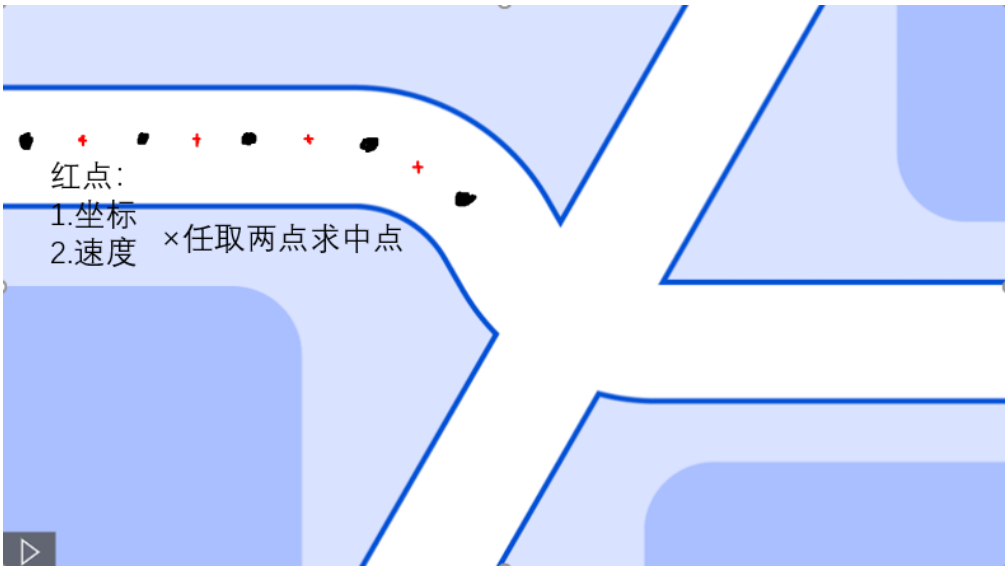
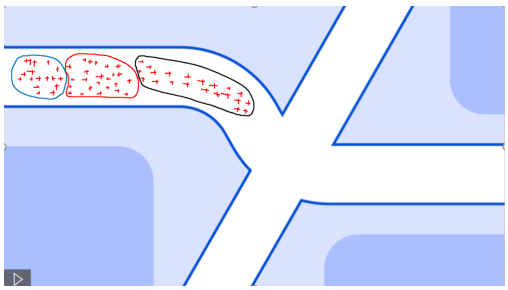


Figure 3: Find the midpoints

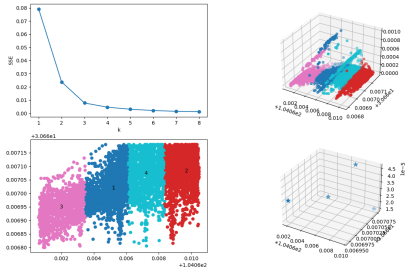
Next step is to apply clustering algorithm like *K-Means* or *MeanShift* to find a certain number of clusters of the midpoints. Every midpoint has three features:

- longitude
- latitude
- speed

So actually this is a three-dimension clustering, but the result we need is two-dimensional, which means we need to balance the weight of these features carefully to get a correct partition.



(a) Clusters



(b) K-Means Example

Finally, compute the convex hull of each cluster and find its intersection with the roads. Therefore, we have partitioned the roads into several segments.

4.3 Code Implementation

4.3.1 Map Matching

The first step is to locate GPS point into roads. This problem is called **Map Matching**.

In our dataset, a road is represented as a line, however, it should be an area in real world. Besides, it is hard to match a point to a 2-D line. Therefore, we need to convert a road to an area in advance.

Use method `buffer()` in *shapely* package.

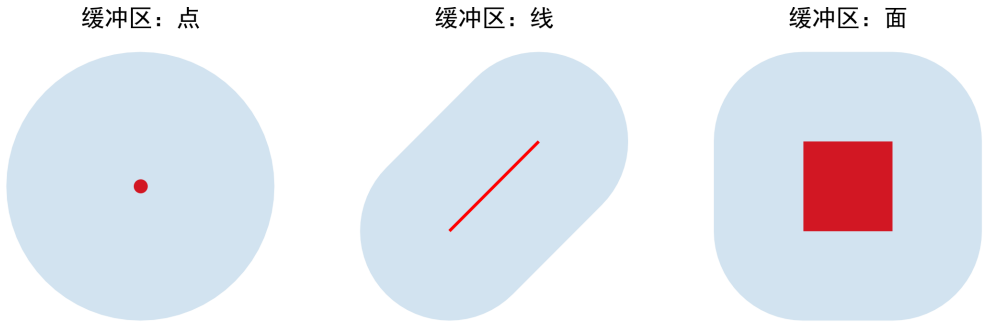


Figure 4: `buffer()`

The `buffer()` method will convert a `Line` to a `Polygon`.

Take 羊市街 + 西玉龙街 and the tracks of 10,000 taxis as an example. Actually there are three roads. Apply this function to the roads and we will get

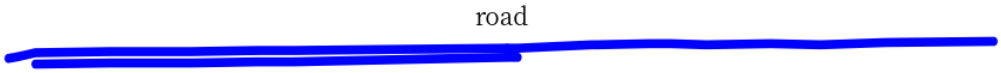


Figure 5: Buffered Roads

Secondly, filter the data. Use method `geopandas.GeoSeries.total_bounds` to get the bottom left corner and the upper right corner of these roads, then screen out the points which are not in the rectangle area.

Finally, for each point, find which road it in by `contains()` method in *shapely* package.

4.3.2 Calculate Midpoints

In this step, we calculate the coordinate and speed of the midpoints.

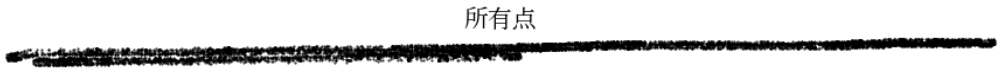


Figure 6: Matched Points

First thing to notice is that we should use the point in the same track, otherwise the timestamps are not continuous, resulting in a wrong speed.

In addtion, after working out the midpoints, we need to match these points again into the roads because there are some arc-shaped roads so that the midpoints may be in the interior of the arc.

Figure 7: Midpoints

4.3.3 Clustering

In this step, we use *KMeans* or *MeanShift* in *sklearn* package.

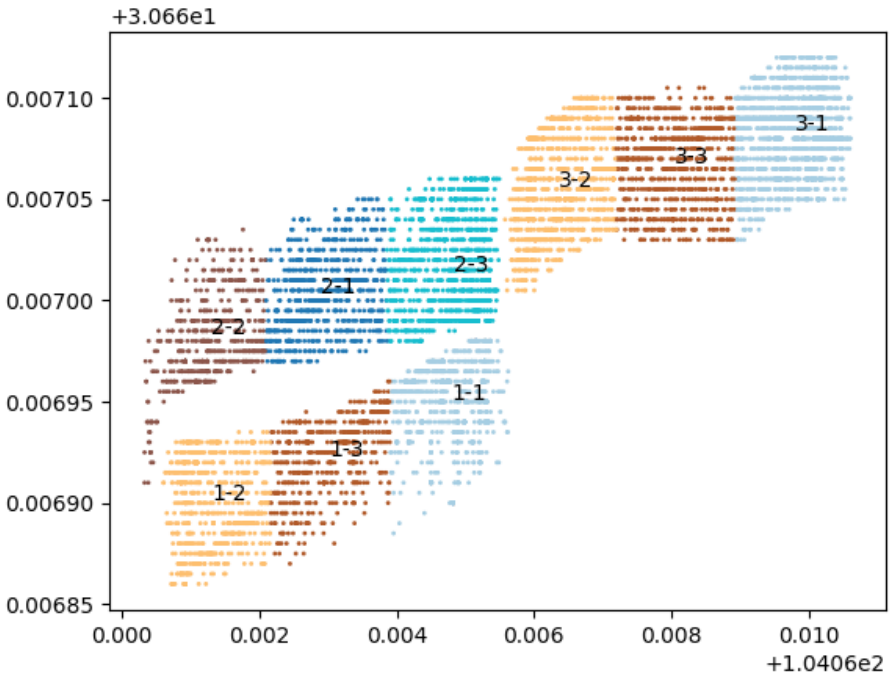


Figure 8: KMeans

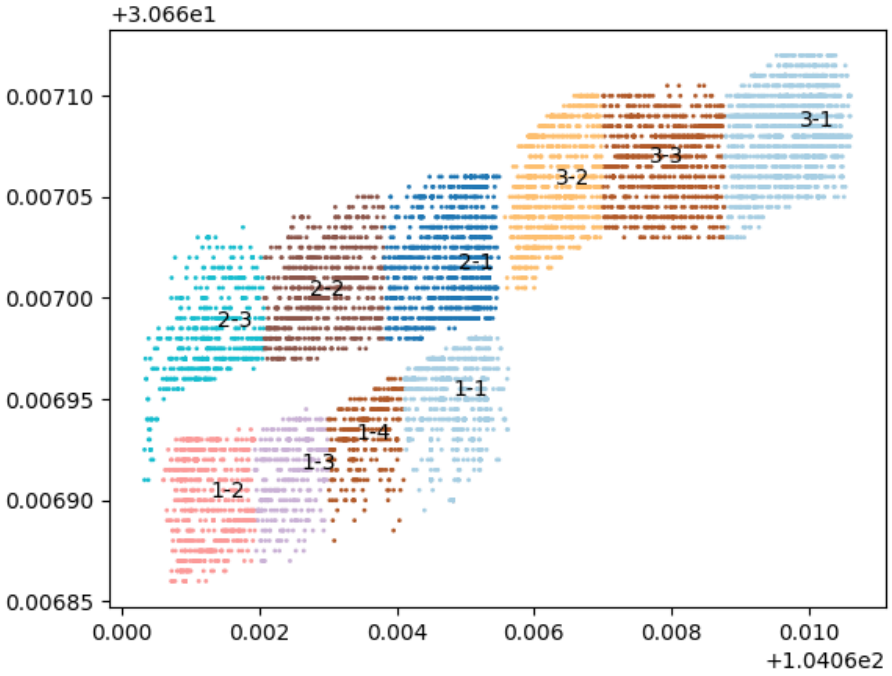


Figure 9: MeanShift

4.3.4 Intersection

The last step is to compute the intersection of the clusters and the roads.

First we need to calculate the convex hull of the clusters, because we cannot use the intersection of line and points. Use `geopandas.GeoSeries.convex_hull` to compute the convex hull of the clusters. Then use `intersection()` method in *shapely* package to get the intersection.

Figure 10: Segments