THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2

Preliminary Data Processing

2.1 Data Collection and Cleaning

The taxi GPS data used in this project is collected from the Computational Sensing Lab [16] at Tsinghua University, Beijing, China. The data set contains approximately 83 million time-stamped taxi GPS records collected from 8,602 taxis in Beijing, from 1 May 2009 to 30 May 2009. The data set consists of seven original fields as shown in Table 2.1. Longitude and latitude in the data set are defined in the WGS-84[1] standard coordinate system, which is the reference coordinate system used by the GPS.

The original data set came in a binary file format. After it was decoded and imported into a MySQL database, the first step in data cleaning was **to delete all records with zero value in the SPEED field**, since when a taxi is stationary it yields no valuable information about the *trajectory* it is moving along. While being stationary could be due to a traffic jam, this kind of information is well captured by the time span between the last and the next *non-stationary* data point.

[1] World Geodetic System

Field	**Explanation**
CUID	ID for each taxi
UNIX_EPOCH	Unix timestamp in milliseconds since 1 January 1970
GPS_LONG	Longitude encoded in WGS-84 multiplied by 10^5
GPS_LAT	Latitude encoded in WGS-84 multiplied by 10^5
HEAD	Heading direction in degrees with 0 denoting North
SPEED	Instantaneous speed in metres/second (m/s)
OCCUPIED	Binary indicator of whether the taxi is hired (1) or not (0)

Table 2.1: Seven original fields in the data set

In addition, all records must have a *unique* pair of **CUID** and **UNIX_EPOCH** fields, since it is not possible for a taxi to appear in two different locations at the same moment in time. This kind of error is likely due to some errors in aggregating the original data set.

2.2 Reverse Geocoding

After the data set was cleaned, the next step was to map each GPS data point to a real street based on its longitude and latitude, which is also known as *reverse geocoding*. A number of algorithms [7] have been proposed for this purpose, but most of them require an additional GIS[2] database of the road network in Beijing. This project adopted an alternative strategy which leveraged on the existing public API[3]s for reverse geocoding.

Currently, a number of online mapping platforms provide reverse geocoding services as part of their developer APIs. Amongst others, Google Maps and Baidu Maps

[2]Geographic Information System
[3]Application Programming Interface

offer relatively stable and fast reverse geocoding services. However, due to the "China GPS shift problem" [13] where longitudes and latitudes encoded in WGS-84 format are required by regulations to be shifted by some variable amounts when displayed on a street map, Google Maps is not able to display a GPS data point correctly because it only supports WGS-84 formats. Figure 2-1 illustrates the effect of such shift, where the correct location is displayed on the **right**.

Figure 2-1: An example of China GPS shift problem

Baidu Maps, on the other hand, has been using their own coordinate system, BD-09, which is an improved version of the Chinese official coordinate system, GCJ-02. Baidu Maps does provide a set of APIs to convert WGS-84 coordinates into BD-09 ones. To reverse-geocode GPS data points, longitudes and latitudes must first be converted into BD-09 format. Then another set of APIs from Baidu Maps can be used for reverse geocoding. To store the converted longitudes and latitudes as well as the street names obtained from reverse geocoding, four new fields were added to the original data set as shown in Table 2.2.

In order to use Baidu Maps APIs for coordinate conversion, the following system architecture was set up as shown in Figure 2-2. The Apache HTTP server hides the MySQL database and sends HTTP POST requests containing the original coordinates

Field	Explanation
DataUnitID	Nominal primary key for each record
UTC	UNIX_EPOCH in human readable format
BD09_LONG	Longitude encoded in BD-09 format
BD09_LAT	Latitude encoded in BD-09 format
STREET	Street name

Table 2.2: Additional fields added to data set

to Baidu Maps Web API to get the converted coordinates. Then it updates the database through PHP *mysqli* utility.

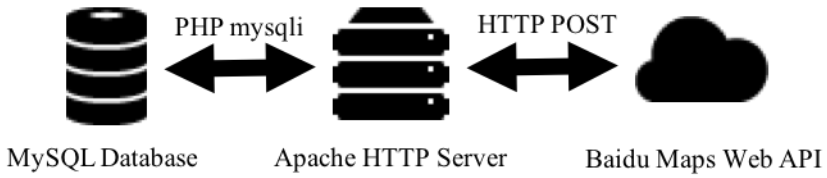

Figure 2-2: System architecture for coordinate conversion

After longitudes and latitudes were converted from WGS-84 format to BD-09 format, Baidu Maps Web API was used to reverse geocode all GPS data points. However, the system architecture was slightly changed, to accommodate the change in technology used. For reverse geocoding, AJAX[4] was used to communicate with the Baidu Maps Web API for speed and unlimited number of requests per day. Therefore, one additional layer was added to the existing system architecture as shown in Figure 2-3.

Executed in a web browser environment, AJAX sent HTTP POST requests to the Apache HTTP server to fetch the converted coordinates in BD-09 format which were

[4]Asynchronous JavaScript and XML

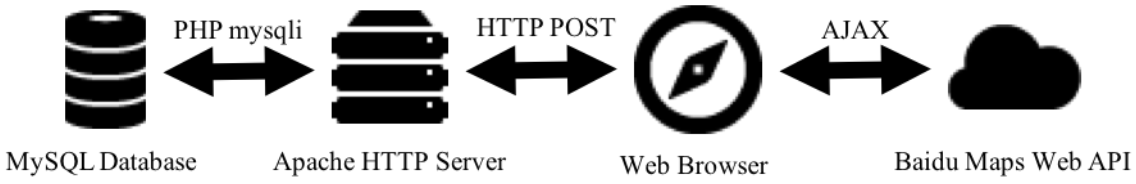

Figure 2-3: System architecture for reverse geocoding

subsequently sent to the Baidu Maps Web API server *asynchronously* via HTTP GET requests. Once the server responded with the name of the street, AJAX updated the database by sending another HTTP POST request to the Apache HTTP server. The asynchronous nature of this architecture, however, caused a few problems which are addressed in Section 2.3.

2.3 Outlier Detection

2.3.1 Motivation for Outlier Detection

The Baidu Maps Web API for reverse geocoding is stable and fast, but does not produce no errors. Sometimes, a GPS data point may be mapped to a main road but actually it is on the side road, which is one of the limitations mentioned in Section 1.2, or it is actually mapped to a street that Baidu Maps does not recognise. In neither case will Baidu Maps produce a correct reverse geocoding. Moreover, the reverse geocoding process is asynchronous, which means that it is being performed in the background in parallel with the main application thread. Therefore, it is inevitable that some street names may get lost when the records are being updated or a record is updated with a wrong street name. Figure 2-4 shows a drastic example.

In this example, Baidu Maps believes that all data points plotted belong to a particular street. But when plotted on a 2-D plane, these data points almost represent

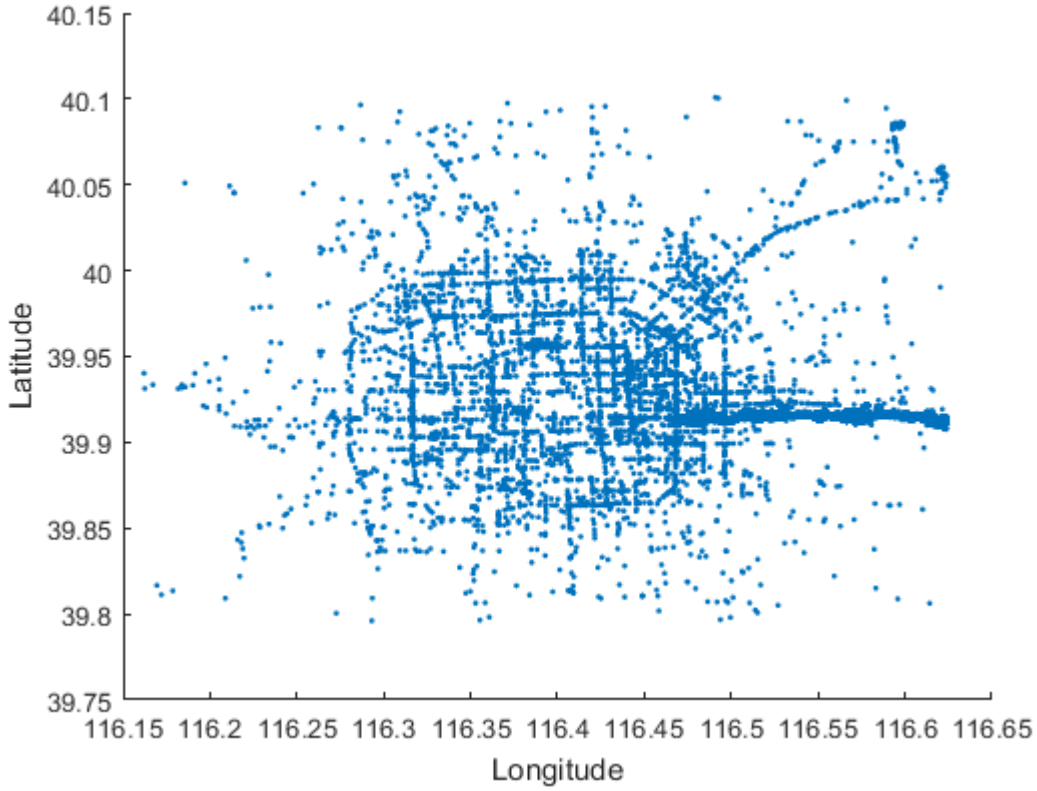

Figure 2-4: An example of outliers

the *entire* road network in Beijing. The actual, correct street is represented in the figure as the *thickest* blue line on the right half of the figure with a longitude ranging from 116.45°E to 116.65°E. Erroneous records like those not on the thickest line are known as *outliers* and must be properly identified and removed. This project proposes a novel outlier detection approach based on unsupervised learning whose principle behind is based on Theorem 1.

Definition 4 (*Reasonable reverse geocoder*)**.** A reasonable reverse geocoder always gives its best matching from a GPS data point to a street whenever possible and has an accuracy more than 50%.

Theorem 1 (*Majority Clustering Theorem*)**.** If a *reasonable reverse geocoder* is used to reverse-geocode a set of GPS data points which are mapped to a particular street *in reality*, then, when plotted on a 2-D plane, majority (more than 50%) of the points must be clustered together to form a rough shape that is similar to the shape of the street that they are supposed to be mapped to.

Proof. Proof by contradiction. Assume, for the purpose of contradiction, majority (more than 50%) of the data points that are *indeed* located on the same street are scattered arbitrarily on a 2-D plane after being reverse-geocoded by a reasonable reverse geocoder. In particular, when plotted on a 2-D plane, majority of them *do not* form any similar shape to that of the street they are supposed to be mapped to. Then, the majority must have been erroneously mapped to some other streets because no single street covers the whole city's road network. Thus, the reasonable reverse geocoder has only achieved an accuracy that is less than 50%, which contradicts Definition 4 of a reasonable reverse geocoder. □

2.3.2 Outlier Identification

Apparently, Baidu Maps provides a reasonable reverse geocoder because it is of industrial-grade quality and has an accuracy larger than 50%. Therefore, if a set of points belong to a particular street, after reverse-geocoded by Baidu Maps, majority of them should be clustered to assume a rough shape of that street according to Theorem 1. Based on that, an unsupervised learning technique — clustering can be used to separate the correctly mapped data points from the outliers.

Many clustering techniques are available [10]. Since each record can be represented graphically as a point on a 2-D plane with longitude as the x axis and latitude as the y axis, a **self-organising feature map** (SOFM) [6] seems to be an appropriate technique to use.

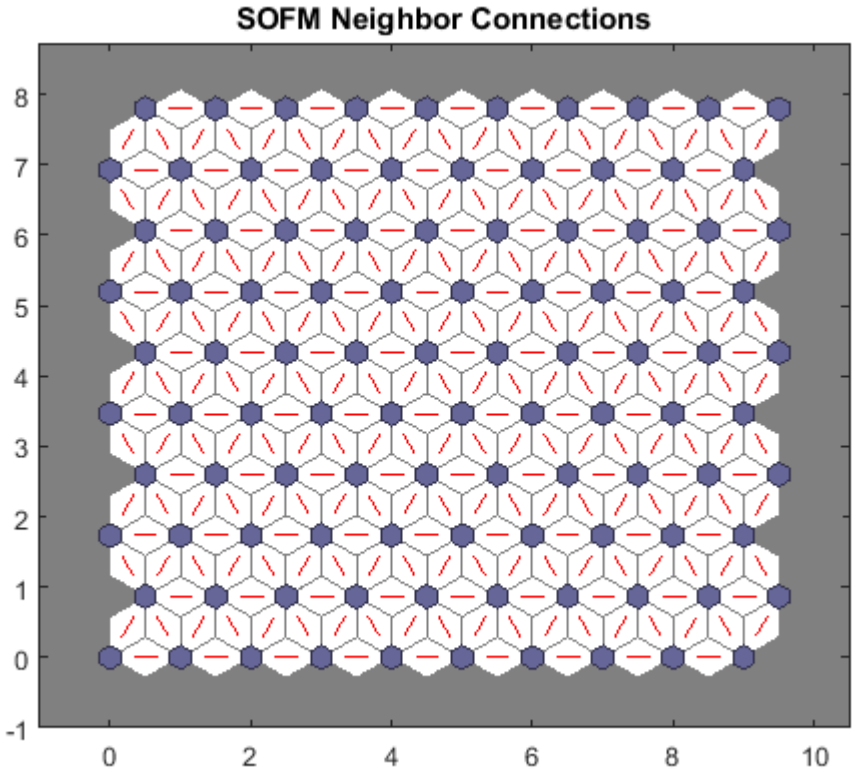

Figure 2-5: An example of SOFM

A self-organising feature map is a form of artificial neural network. It consists of a pre-defined number of interconnected neurons distributed over a 2-D plane as shown in Figure 2-5. Prior to training, the neurons are randomly scattered among the data points and gradually move to the *centroids* of the data clusters they represent as they learn the *features* of the training data. Upon termination of the training, all data points near to a particular neuron, in terms of Euclidean distance[5], are assigned to the cluster that neuron represents. Figure 2-6 shows the results after the clustering is completed.

It is clear from the figure that while some neurons represent the clusters of outliers,

[5]Other distance measures are also possible.

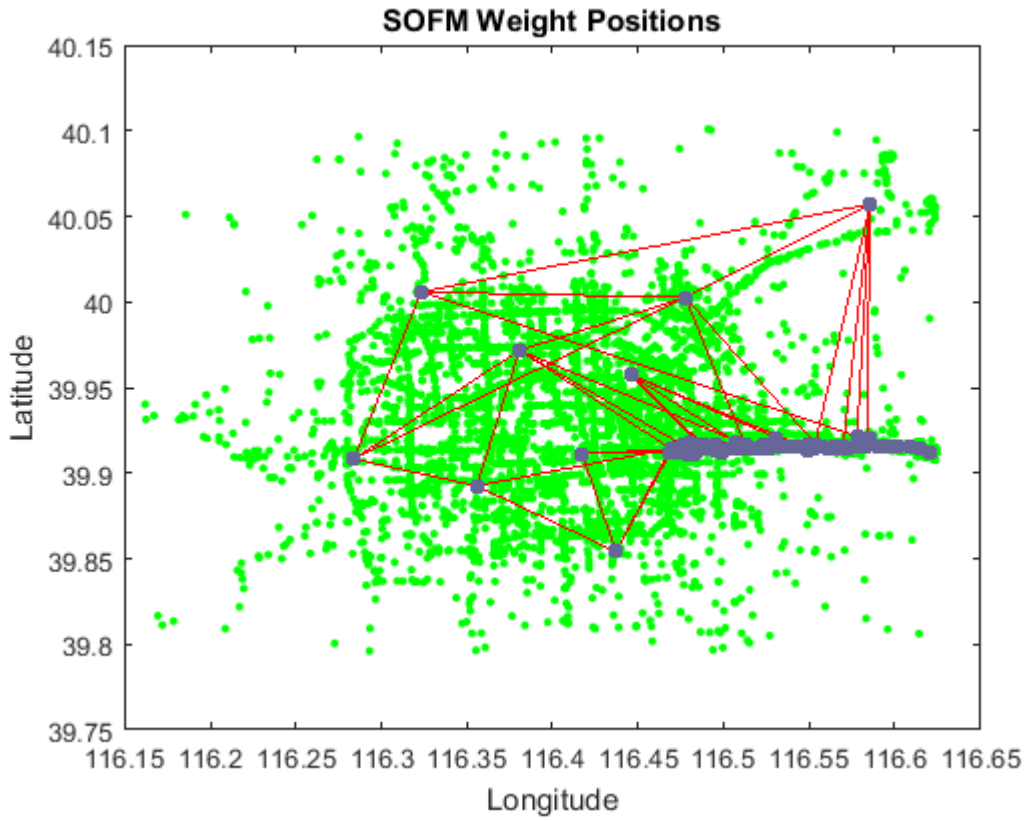

Figure 2-6: Neuron positions after training

majority of the neurons are clustered to *cover* the correct street they should represent. A 10×10 SOFM was used in this project, so there were at most 100 neurons or equivalently, 100 clusters. Each cluster had a different size. To ensure a thorough removal of the outliers, **only the top 50% largest clusters were considered as clusters of correct data points which are called "legal clusters". All other clusters were deemed as clusters of outliers**.

2.3.3 Outlier Removal

Once the legal clusters were identified, a distance threshold was set to remove outliers so that **whenever the minimum distance between a data point and all cen-**

troids of the legal clusters was above the threshold, that data point would be considered as an outlier and removed. The python-like pseudocode in Listing 2.1 describes this idea with more clarity. Source code is listed in Appendix A.2.

Listing 2.1: Pseudocode for outlier removal

```
1 Input: a set of GPS records, a set of centroids and a distance threshold
      d_max
2 Output: a set of GPS records with all outliers removed
3
4 for recd in records:
5     min_dist = ∞
6     for ctrd in centroids:
7         min_dist = min(min_dist, get_distance(recd, ctrd))
8     if min_dist > d_max:
9         records.remove(recd)
```

However, the *distance* between a data point and a centroid is not as straightforward as Euclidean distance. A centroid, to some extent, can be imagined as a *real* point on the Earth's surface. To calculate the distance between a data point and a centroid is to calculate the **spherical distance** between them, which is given by the Haversine Formula in Theorem 2.

Theorem 2 (*Haversine Formula*)**.** Given two points $P(\lambda_1, \varphi_1)$ and $Q(\lambda_2, \varphi_2)$ on the surface of a sphere, where λ and φ represent longitude and latitude in radians, their spherical distance (the distance along a great circle of the sphere) is given by [9]

$$d = 2R \arcsin \sqrt{\sin^2 \frac{\varphi_2 - \varphi_1}{2} + \cos \varphi_1 \cos \varphi_2 \sin^2 \frac{\lambda_2 - \lambda_1}{2}} \tag{2.1}$$

where R is the radius of the sphere.

Since the Earth is not a perfect sphere but a spheroid, R varies with latitude.

Theorem 3 suggests how to calculate the Earth radius at any latitude.

Theorem 3 (*Radius at any Latitude*)**.** Given a latitude φ in radians, a polar radius R_p and an equatorial radius R_e, the spheroid's radius at that latitude is given by [12]

$$R_\varphi = \sqrt{\frac{(R_e^2 \cos\varphi)^2 + (R_p^2 \sin\varphi)^2}{(R_e \cos\varphi)^2 + (R_p \sin\varphi)^2}} \tag{2.2}$$

It is known that $R_e = 6,378,137$ metres and $R_p = 6,356,752$ metres on the Earth [14] and that Beijing's latitude is about 39°N. Therefore, the distance between a data point and a centroid can be calculated. For this project, two distance thresholds were selected: 30 metres and 50 metres. The thresholds were set in a way that it ensured there was sufficient data for subsequent machine learning tasks while the estimates of travel time were as least affected as possible by outliers. Should the thresholds be set to a too small value, the remaining data could not have been sufficient; on the other hand, however, if the thresholds were set to a too big value, the accuracy of the final results would have been subject to outliers.

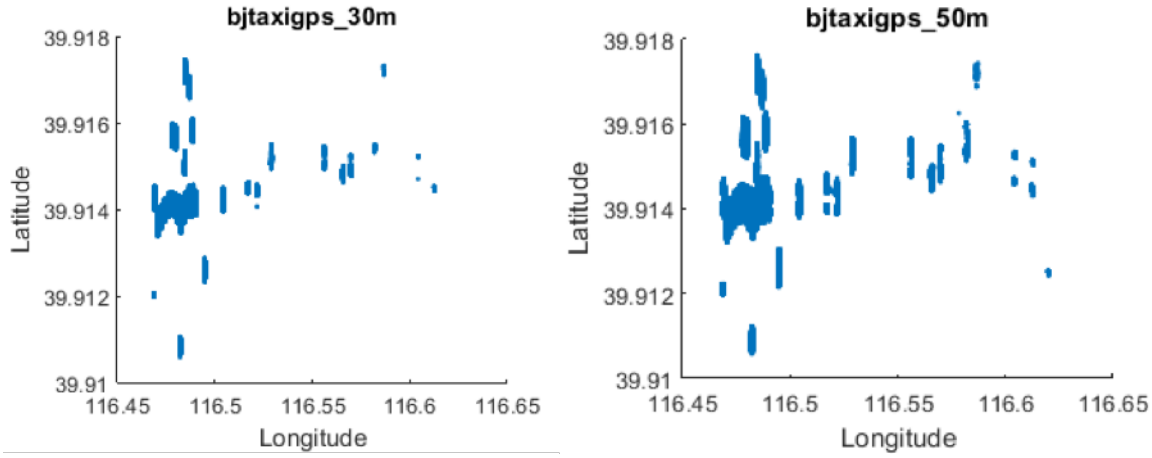

Figure 2-7: Plot of data points after outlier removal

After the outliers were removed, two data sets remained. They are hereinafter re-

ferred to as **bjtaxigps_30m**, where outliers were filtered by a distance threshold of 30 metres and **bjtaxigps_50m**, where outliers were filtered by a distance threshold of 50 metres, respectively. bjtaxigps_30m contains approximately 51 million records while bjtaxigps_50m has 59 million. All algorithms and procedures described hereinafter are applicable to both data sets. Figure 2-7 gives a plot of the GPS data points after outliers were removed from both data sets. Clearly, the longitude and latitude now have a much smaller *range* and the data points roughly form a shape similar to that of the street they are supposed to be mapped to.