CS4225 Project Report

Predicting Speed of Road using GPS Traces

Voon Soo Yin **Zhu Yilun** **Han Xiao**

1. Project Introduction

Traffic congestion is a common occurrence in large and growing cities around the world. Not only does an increase in congestion level lead to lower productivity and higher levels of air pollution, it also results in negative social and psychological effects like increased stress and road rage. As city population increases, transportation systems are now struggling to cope with the increased number of vehicles on their roads. There have been many measures to manage traffic such as providing incentives for travelling during non peak periods, but such solutions are not dynamic or effective enough.

With more data collected from sensors and advanced analytics capabilities, there is now a huge potential in using data science to optimise the management of traffic. Decision makers can benefit from data analytics by knowing traffic flow in real time, and even play a part in relieving traffic congestion. For urban planners, understanding and modelling traffic flow can help them to optimise location of infrastructure and facilities. For drivers, near real-time streaming of predicted speed of roads can aid in calculating ETAs and help them to avoid congested paths, thus relieving congestion and improving driving experience. For ride sharing service providers like Uber and Grab, speed prediction of roads can similarly help to calculate ETA and provide data for dynamic pricing of trips.

Road data can be easily accessed from speed cameras, in-built road and gantry sensors to understand traffic congestion trends. However, this can be expensive for areas without the necessary infrastructure that is required for implementation. In order to predict speed, we use a set of GPS traces as it is inexpensive and provides sufficient granularity. GPS traces are widely available because of adoption of mobile phones and the increasing presence of ride sharing or navigation apps. This provides more data for analysis that can increase the precision and accuracy of our prediction.

2. Existing Work

We have identified 3 papers related to our topic, each with different approaches towards traffic flow predictions using GPS data.

In the first paper, “City traffic forecasting using taxi GPS data: A coarse-grained cellular automata model” [1], the proposed model simulates vehicles moving on uniform grids whose size are much larger compared with the microscopic cellular automata model, with data obtained from GPS devices mounted on taxis in Beijing.

The taxi gps data is first cleaned for errors and then mapped to an actual road network with a map matching algorithm OSRM (Open Source Routing Machine). The simulation domain (Central Beijing) in the coarse-

grained model is partitioned into 256 x 256 uniform-grids with 100m x 100m spatial resolution. At each grid, the vehicle is assumed to move in only 4 directions, up, down, left, and right. The average speed V_i and occupancy N_i for each grid is calculated for each of the 4 directions for every 10 minute interval. From the historical data of the speed and occupancy of each grid, a function V(N) was created, which predicts the speed of the grid with a given occupancy. However, due to the way the grids are split and the assumption placed on the movement of vehicles, the speed derived may be faster than the actual speed as diagonal or bent roads are not taken fully into consideration.

In the second paper, "Traffic Congestion Detection Based On GPS Floating-Car Data" [2], traffic congestion information for a big city in China is estimated by estimating travel speeds from GPS floating car data.

The GPS data is first cleaned, and a map matching algorithm is carried out to bind the GPS data to different road segments. After map matching, the average speed V_i between two GPS points binded to the road is calculated by taking the distance on the road L_i divided by the time interval t_i between the two points. The average speed of the entire road segment is calculated by taking the sum of all the V_i on the road segment divided by the total number of V_i calculated on the road segment. The travel speed of each road for a day is calculated, and segmented into varying speed ranges to show the different levels of congestion according to the "Index System of Traffic Management Evaluation of Urban Road".

In the third paper, "Urban Traffic Modelling and Prediction Using Large Scale Taxi GPS Traces" [3], a method to construct a model of traffic density based on large scale taxi traces is proposed, with focus on determining the capacity of each road.

First, a model of the traffic flow in the city is constructed based on taxi GPS logs. A digital map of the city is constructed from the gps traces of taxis, and the route taken by the taxis are then mapped to the digital map. A probabilistic transition matrix is computed to count the frequency of transitions from one road segment to another. The future density of the road segment is predicted by spreading the current density with the transition matrix. The average speed is calculated for each road segment for each minute. Ratio(d) of speed at a particular density level for a road segment is defined as the ratio of average speed above 20km/h to speed below 20 km/h.The capacity of a road segment is determined by taking the density level when ratio(d) for the road drops below 0.4. The condition of a road segment for a time segment is defined as density divided by capacity, which is the proportion of the capacity of the road that is taken up.

From our research, we realised that the cleaning and map matching of the GPS data was a common difficulty faced. We decided to tackle this problem differently with the help of Open Street Map data since road binding has already been carried out for our data. For the traffic flow prediction, we will take reference from the method used in the second paper, "Traffic Congestion Detection Based On GPS Floating-Car Data", to calculate the historical average speed. However, as suggested in the third paper, "Urban Traffic Modelling and Prediction Using Large Scale Taxi GPS Traces", purely historical average may not be a good prediction. As we seek to predict the speed of each road segment, the density prediction method used in the third paper may not be as applicable. Therefore, we propose a different method using time series regression model for speed prediction. All the papers mentioned conducted some form of map matching of the GPS taxi traces, most of these methods require building a special algorithm or building a custom map.

In our experiment, we propose a method that makes use of road binded GPS and simplify the process of map matching with the use of existing tools.

3. Methodology and Experimentation

3.1 Input Data

The input dataset consists of GPS traces from the trajectory data of Didi Express and Didi Premier drivers within the Second Ring Road of Xi'an City in October and November of 2016.

The GPS traces given are bounded by the the geographic coordinates: (108.92309,34.279936, 109.008833,34.278608) and (109.009348,34.207309, 108.921859,34.204946). The measurement interval of the track points is approximately 2-4 seconds and the track points were bound to physical roads so that the trajectory data and the actual road information are matched.

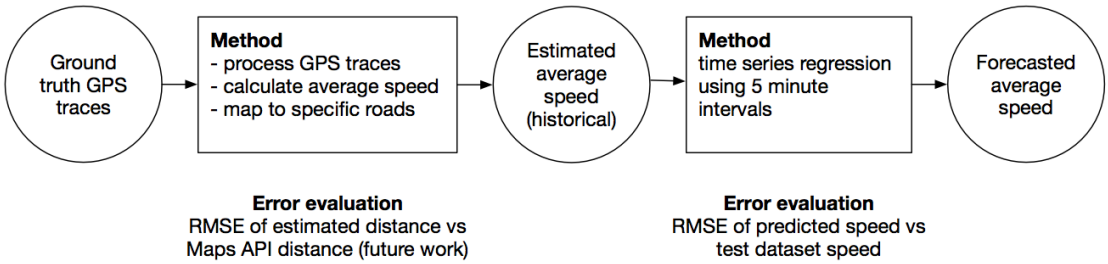
This amounts to around 1536 million rows of data and a large file size of around 159 GB (uncompressed). A sample row of data is shown in the table below (Figure 1), consisting of driver ID (anonymised), order ID (anonymised), timestamp (Unix), longitude and latitude columns.

Driver ID	Order ID	Timestamp	Longitude	Latitude
a01b8439e1e42ffcd286241b04d9b1b5	f11440a64a0f084fe346a398c62aa9ad	1475277506	108.9276	34.27659

Figure 1: Sample row of data

Figure 2: Visualisation of roads and 3 sample trajectories using data from 1st October 2016

3.2 Overview of Method



To obtain the estimated average speed from GPS traces which represents the ground truth, the GPS coordinates and timestamp were first processed and the average speed was calculated and mapped to specific roads. To obtain the forecasted average speed from the estimated average speed, we performed various time series regression methods using 5 minutes intervals and the performance of each predictive model was evaluated using RMSE.

3.3 Step 0: Road Segmentation and Pre-processing of Data

Having only the Didi dataset is not enough to start processing in Spark right away. Given a geolocation data consisting of longitude and latitude, a way is needed to map the geolocation to a road in Xi'an City. This step is called reverse Geocoder Mapping. Preprocessing work must be conducted on the original dataset to prepare for Step 1: Reverse Geocoder Mapping.

A novel approach is proposed in this report. This new approach solves the dependency on the city and allow the algorithm to be applied to any city. Furthermore, this approach is an effective and robust solution, requiring no third party service and not subject to payment of fees. Open Street Map provides detailed mapping roads of all cities around the world free of charge. Even better, it supports downloading all map-related data given a region. In this project, the region of interest is bounded by the Second Ring Road of Xi'an City. This dataset from Open Street Map (OSM) will be referred to as OSM dataset. The OSM dataset is stored in XML format and contains a variety of information, which are not all useful. Node tags and way tags will help with road segmentation. Each way tag contains a number of node tags, which contain longitude and latitude information. Each way tag also contains the type of the road such as highway, residential way, footway, pedestrian or primary way. After careful review of documentation, way tags with the type of "pedestrian", "cycleway", "path", "steps", and "footway" are removed. Since Open Street Map only provides the data and no services to query through the data, pre-processing work is needed. There are four outputs in this step:

1. Output 1: extract all way id with associated node IDs in a text file.
2. Output 2: extract all node IDs with detailed information, namely street name, longitude, and latitude.
3. Output 3: key-value pairs stored in a text file with the key as the node ID and the value as the way ID.
4. Output 4: calculated distance of each road segment stored in a text file.

This novel approach might not seem significantly better, but it builds up for the next step.

3.4 Step 1: Reverse Geocoding

A reverse geocoding library is chosen, which is freely available on Github. The library takes in a bunch of node ID along with longitude and latitude, which are stored in Output 2 of Step 0. Then the library saves and indexes the input data for quicker access and faster computation. The input data came straight from Didi's dataset. The output will remove the driver ID since it does not help in future computation. By going through each line of the input data, feed the longitude and latitude information to the library. The library queries through the input node points and returns the closest node tag ID. With the node tag ID, one can quickly find out the associated way ID using Output 3 from Step 0.

Input Format	Driver ID, Order ID, Timestamp, Longitude, Latitude
Output Format	Order ID, Timestamp, Longitude, Latitude, Node ID, Way ID

Figure 2: Input and Output of Step 1

Since Step 1 involves the processing of the entire Didi dataset, Spark is utilized for better performance. The reverse geocoding library can be imported as a python library, hence PySpark is used for Step 1. Running two workers with 10 cores each, this step took around 3.8 hours to complete, making Step 1 the slowest step in this project. The lengthy operation is caused by two factors:

1. PySpark is known to be slower than Spark using Scala.
2. The reverse geocoding library takes time to find the closest node to the input longitude/latitude.

To improve the performance for this step, the reverse geocoding library is a good place to start. The reverse geocoding library supports multi-threading, which wasn't utilized due to time constraint on this project. Even without multithreading, the library is optimized and designed to take multiple longitude and latitude pairs at a time; however, only one pair is passed in at a time since Didi dataset contains on pair on each line and Spark reads the input data line by line. It is possible to design the Spark pipeline and pass multiple longitude and latitude pairs to the library for better performance. It is also worth the time to find a similar or equivalent reverse geocoding library in Scala.

3.5 Step 2: Speed Calculation

Step 2's input comes directly from Step 1's output. First, make the Order ID be the key for each RDD. Then group all input data by Order ID. The Order ID will be the key and the value will be an array of input data that are part of this Order ID. Within each value array, sort the data by increasing timestamp. Hence, it becomes easy to trace through each trip in increasing timestamp as if the driver is driving through the city. For each trip, take two consecutive data and calculate the speed. For example, assume the Order ID is 426 and the trip only has four input data namely 1, 2, 3 and 4. By taking two consecutive data, the processing order would be (1,2), (2, 3) and (3, 4). Speed is calculated using distance in meter divided by time delta in seconds. For each pair, distance is calculated based on longitude and latitude using Haversine formula. The time delta can be obtained by subtracting the latter timestamp with the former.

Input Format	Order ID, Timestamp, Longitude, Latitude, Node ID, Way ID
Output Format	Timestamp_start, Timestamp_end, Way ID, NodeID_1, NodeID_2, Speed

Figure 3: Input and Output of Step 2

The Haversine formula calculates the direct distance between two geolocation coordinates. The driver might not travel in straight lines all the time. For example, the driver could be going through a roundabout or the road might be generally straight with a slight curve. Hence, the Haversine distance could be shorter than the actual distance. As a result, the speed calculation could be slower than the actual speed. Since the driver reports the location of the vehicle very 2 to 4 seconds, the distance between the two coordinates in each pair will be close to each other. The high frequency of location reports reduces the error.

Coded in Scala, Step 2 takes 43 minutes to finish with 10 cores on two workers. There is a significant improvement in execution time from Step 1 to Step 2. Distance calculation could be improved by using Google's Baidu's API to find out the driving distance between two points for more accurate speed calculations. Using third-party API would incur charges and introduce delay in processing speed due to network delay on REST API requests. Haversine formula is adopted in the end since the error of the Haversine formula is small enough thanks to the high frequency of location reports.

3.6 Step 3: Filter and Aggregate Data according to Time Intervals

Step 3's input comes from Step 2's output and the road segment distance calculated during the preprocessing step. The aim of Step 3 is to produce the features that can be used in the machine learning model in the later step. The prediction will be done for every 5 minute interval, and since the aim of predicting traffic speeds is to optimise travel decisions and avoid congestion, the prediction time frame is restricted to 6am to 12am, when people are more likely to travel. The features are the average speed and density of each road segment for every five minute interval between 6am to 12am.

First, the timestamp is grouped into five minute intervals by converting the unix time to the local time (GMT+8) and then rounding down each timestamp to the nearest five minute interval. The timestamps are filtered such that only timestamps within the 6am to 12am interval remains. Next, with the WayID and timestamp as key, and speed as value, a groupByKey is carried out to group the speed for each way for each time interval. From the grouped speed, the average speed is calculated by taking an average of the speeds within the time interval, and the density is calculated by taking the distance of the road segment divided by the number of vehicles (the number of speed data).

Input Format	Timestamp_start, Timestamp_end, Way ID, NodeID_1, NodeID_2, Speed
Output Format	<WayID Timestamp>, <Average Speed, Density>

Figure 4: Input and Output of Step 3

By counting the number of values for each WayID, the number of time intervals with data for each WayID can be calculated. To have a better accuracy for the predictive model, only WayIDs with complete data for every 5 minute time interval is considered.

The figure on the right shows the top 10 WayIDs ordered according to the count of the number of time intervals. The total number of time intervals within the time range is 61 (days) * 18 (hours) * 12 (intervals) = 13176 time intervals. The top 5 WayIDs are then chosen for the predictive modelling segment as data is present for all the time intervals.

WayID	Count
55696766	13176
245286913	13176
245402052	13176
259480952	13176
310063377	13176
254607360	13175
382975417	13175
235264610	13174
262891135	13174
457336082	13174

Figure 5: Top 10 filtered WayIDs

Coded in Scala, Step 3 takes approximately 23 minutes to finish with only 3 cores on two workers. This is a improvement in execution time from Step 2 to Step 3. This can be attributed to the reduction in size of the input data after every step.

3.7 Step 4: Predictive Modelling

The average speed data from the preprocessing steps were split into training and test dataset in the ratio of around 8:2, resulting in first 47 days of data in training dataset and last 14 days in test dataset. Five different predictive methods were explored in this paper - namely persistence model, historical average, elastic net regression, SARIMA and LSTM.

3.7.1 Persistence Model

The persistence forecast model is used as a baseline to evaluate the effectiveness of other more sophisticated models. In this model, the speed of the prior time step (t-1) is used to predict the observation at the current time step (t). This is implemented by taking the last speed data point from the training data and history accumulated by walk-forward validation and using that to predict the speed at the current time. The yellow line represents predicted values (using t-1) while the green line represents the actual values.

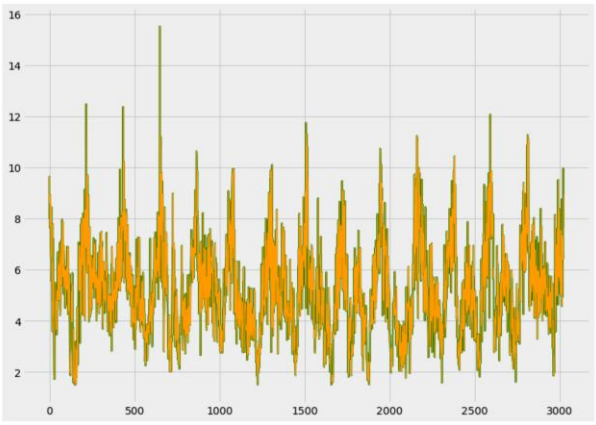


Figure 6: Predicted and actual values of test data using persistence model for WayID 245286913

3.7.2 Historical Daily Average

The historical daily average model is used to calculate the average speed at every time interval in the training dataset in order to predict the speed for comparison with test dataset. For instance, the average speed at 5pm across all days in the training dataset is used to compare with the speed of the route at 5pm in the test dataset. This model works on the assumption that peak periods and non-peak periods coincides at the same time interval every day (i.e. there should be more cars on the road before meal times and lesser cars on the road during non-peak hours like wee hours in the morning).

3.7.3 Elastic Net Regression

A linear regression model may not be suitable as there is a tendency of overfitting in time series modelling and variables may be correlated, resulting in high variance in ordinary least squares (OLS) parameter estimates. As such, regularization is used to decrease this variance at the cost of introducing some bias. The elastic net model, which uses a convex combination of L1 (which penalizes sum of squared coefficients) and L2 penalty (which penalizes the sum of absolute values of the coefficients), was hence chosen for this analysis using Spark MlLib. Elastic net minimises the loss function shown, where the alpha hyper-parameter is between 0 and 1 and controls how much L2 or L1 penalization is used (0 is ridge, 1 is lasso). After experimenting with various parameters, the regParam was chosen to be 0.01 while the elasticNetParam was chosen to be 0.8.

$$L_{enet}(\hat{\beta}) = \frac{\sum_{i-1}^{n}(y_i - x_i'\hat{\beta})^2}{2n} + \lambda\left(\frac{1-\alpha}{2}\sum_{j-1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j-1}^{m}|\hat{\beta}_j|\right)$$

3.7.4 LSTM

Regression models, however, are unable to model the complexity of sequence dependency in time series data. The Long Short Term Memory Network (LSTM), a type of recurrent neural network (RNN), is a commonly used deep learning technique to address time series prediction problems. It is trained using back propagation through time and one distinct benefit of LSTM is that it can learn and remember over long sequences of data points and does not depend on a pre-specified window lagged observation as input. The

network used in this paper has a visible layer with 1 input, a hidden layer with 2 LSTM blocks, and an output layer that makes a single value prediction. The model uses mean squared error as the loss function and ran with 20 epochs and a batch size of 1 due to computational constraints.

3.7.5 SARIMA

Autoregressive Integrated Moving Average (ARIMA), is a forecasting method commonly used for univariate time series data. However, ARIMA models do not take into account the seasonal component that exist in our dataset. Seasonal Autoregressive Integrated Moving Average (SARIMA), is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component. It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series, as well as an additional parameter for the period of the seasonality. Given that there is a high likelihood of seasonality in our dataset (presence of peak and non peak), SARIMA was used and the best parameters were found using grid search.

4. Discussion of Results

4.1 Predictive Modelling

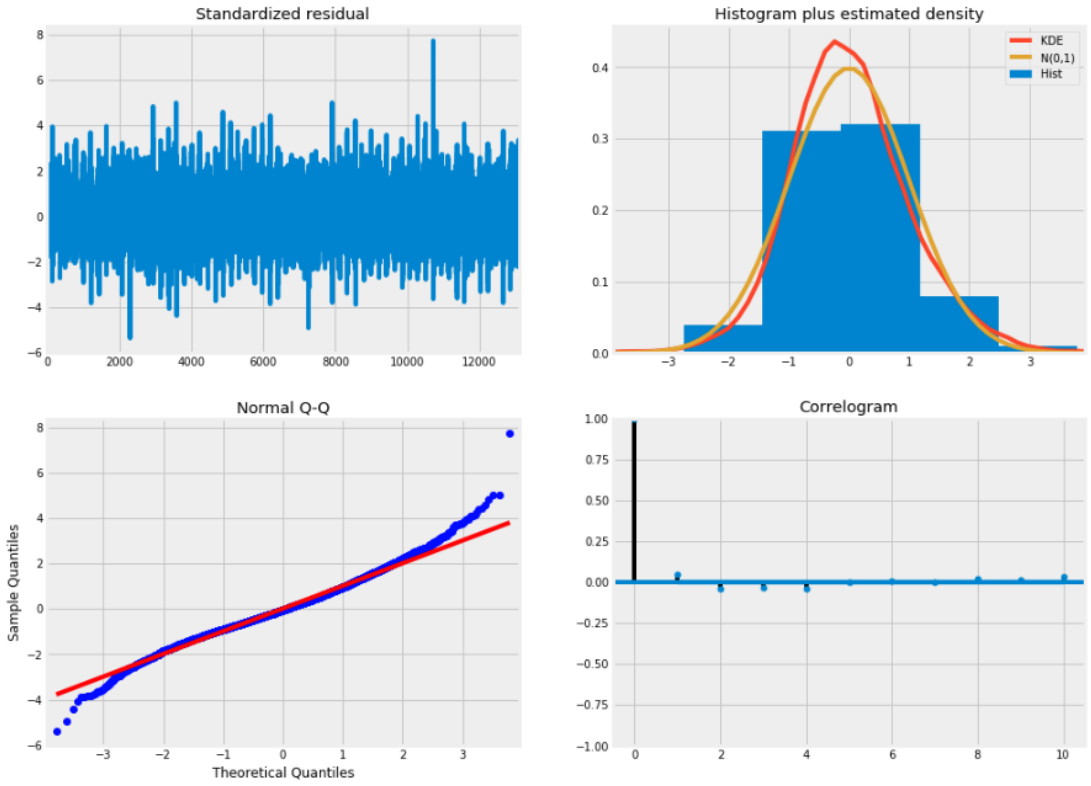
To evaluate the performance of each model, the square root of the variance of the residuals, or root mean squared error (RMSE) score is used. RMSE indicates the absolute fit of the model to the data – how close the observed data points are to the model's predicted values. Lower RMSE scores indicate better fit. It is observed that the SARIMA model performs best consistently for all way IDs compared to the other four models.

WayID	Persistence Model (Baseline)	Historical Daily Avg	Elastic Net Regression	LSTM	SARIMA
55696766	0.634	0.69	0.74	0.57	0.55
259480952	1.002	1.75	1.76	1.01	0.95
245402052	1.242	1.20	1.25	1.14	1.03
310063377	1.389	1.25	1.34	1.28	1.2
245286913	1.312	1.28	1.24	1.20	1.11

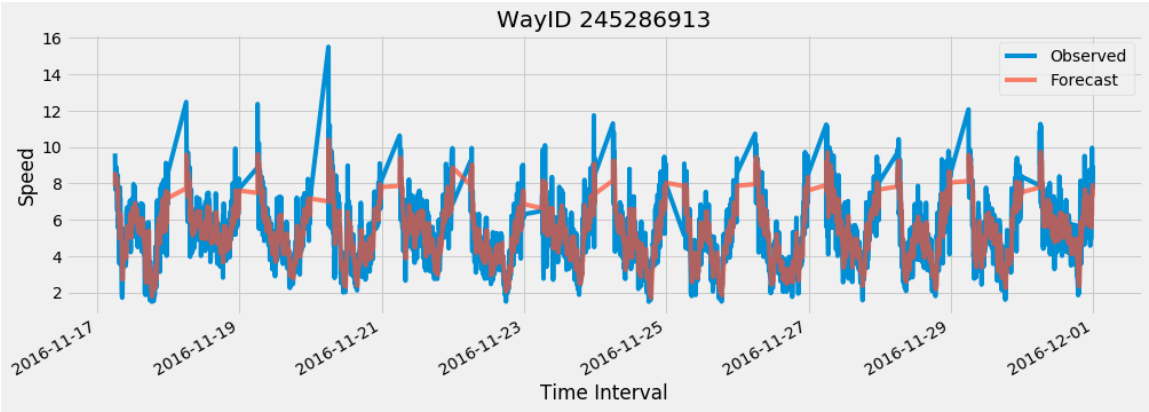
Figure 7: RMSE for each model

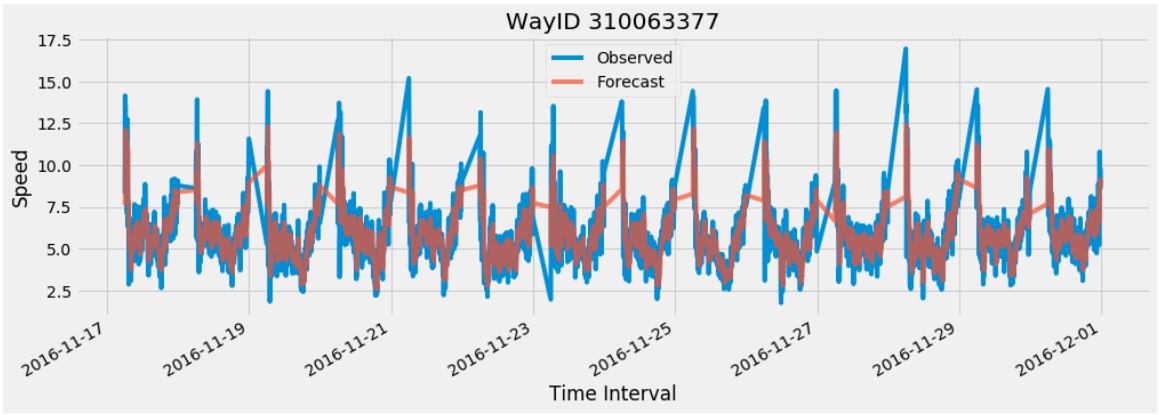
To evaluate if the SARIMA model was a good fit, plot diagnostics was used to learn about the residual plots of the SARIMA model. As can be seen from the figure below, the residual errors (top left) seem to fluctuate around a mean of zero and have a uniform variance. The density plot (top right) also suggests normal distribution with mean zero. The bottom left graph also shows the red dots falling in line with the red line indicating that the distribution is not skewed. The bottom right graph, or the correlogram, shows

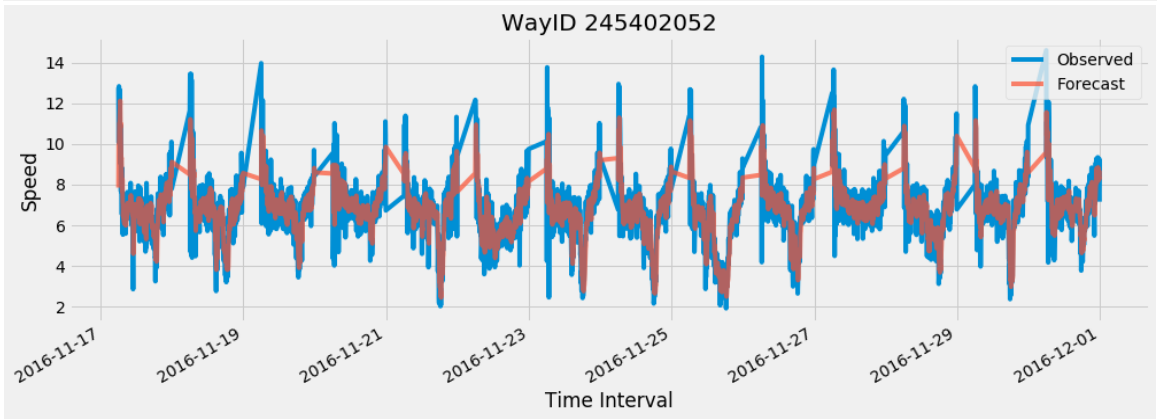
that the residuals are not autocorrelated. This means that there is no need to look for more predictors in the model as there is no pattern in the residual errors which are not explained by the model.

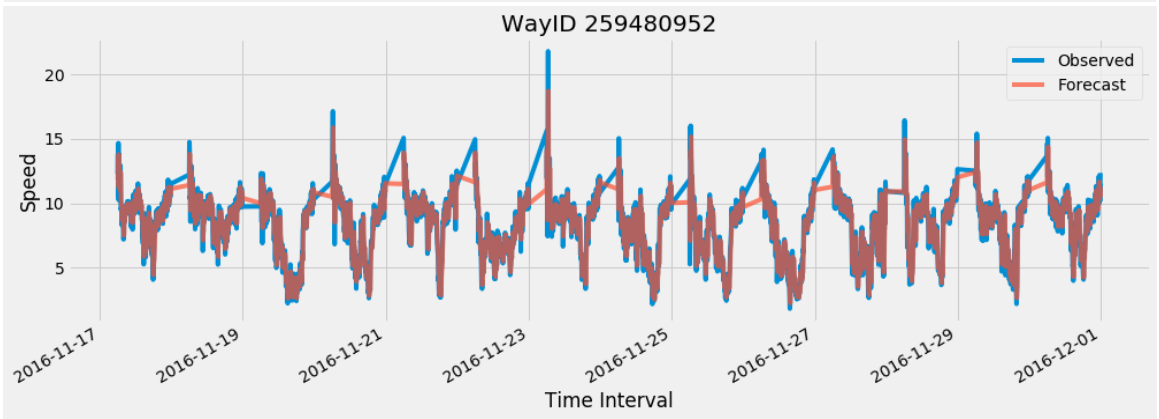


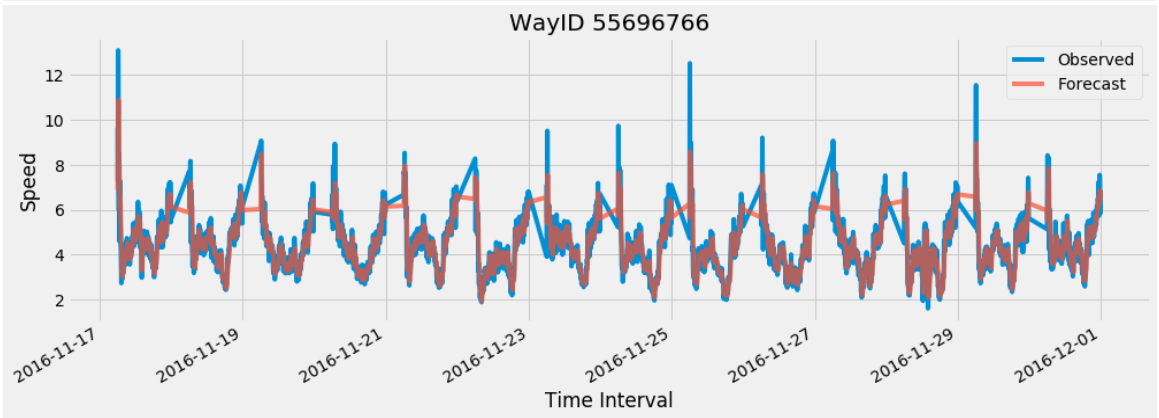
The predicted values was plotted against the actual values for each of the five way IDs using the SARIMA model (figures shown below).




WayID 310063377
Observed
Forecast
Speed
17.5
15.0
12.5
10.0
7.5
5.0
2.5
2016-11-17
2016-11-19
2016-11-21
2016-11-23
2016-11-25
2016-11-27
2016-11-29
2016-12-01
Time Interval


WayID 245402052
Observed
Forecast
Speed
14
12
10
8
6
4
2
2016-11-17
2016-11-19
2016-11-21
2016-11-23
2016-11-25
2016-11-27
2016-11-29
2016-12-01
Time Interval


WayID 259480952
Observed
Forecast
Speed
20
15
10
5
2016-11-17
2016-11-19
2016-11-21
2016-11-23
2016-11-25
2016-11-27
2016-11-29
2016-12-01
Time Interval


WayID 55696766
Observed
Forecast
Speed
12
10
8
6
4
2
2016-11-17
2016-11-19
2016-11-21
2016-11-23
2016-11-25
2016-11-27
2016-11-29
2016-12-01
Time Interval

4.2 Map Visualisation

The final output of the project is the visualisation of predicted speeds for the 5 road segments on the map for a given time interval.

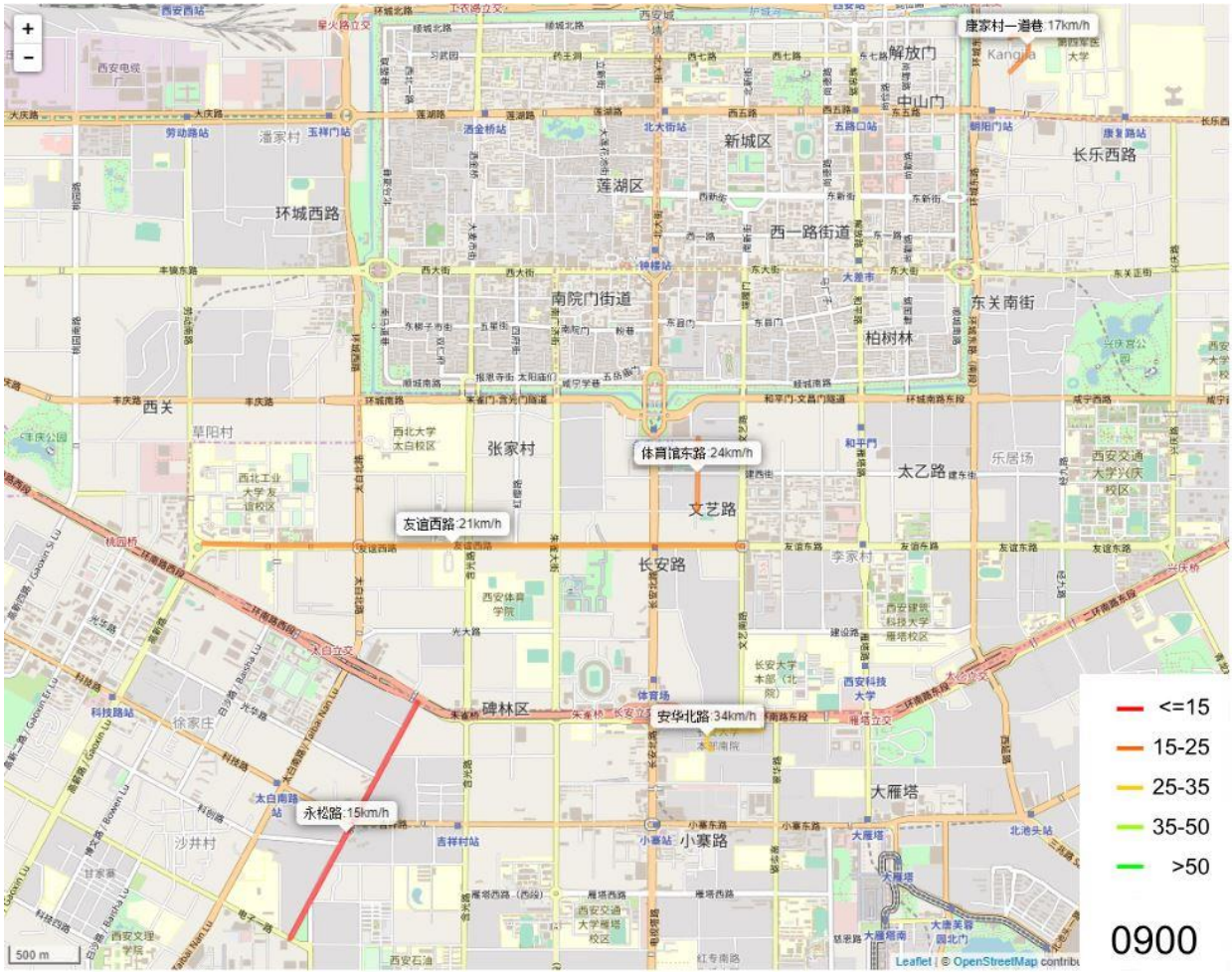


Figure 8: Map Visualisation for 0900hrs

For a given time interval, the road segments will be color coded according to their speed, as shown in the figure above. Following the method in the paper "Traffic Congestion Detection Based On GPS Floating-Car Data" [2], the traffic speed is divided into 5 intervals according to the "Index System of Traffic Management Evaluation of Urban Road". The intervals are: serious congestion (<=15km/h), congestion (<=25km/h), normal (<= 35km/h), smooth (<= 50km/h) and fast (> 50km/h).

The visualisation is done with the help of the Overpass API, which integrates with Open Street Map data. This method provides the best accuracy for road segment visualisation as the roads are segmented according to the way segmentation on Open Street Map.

5. Problems Encountered and Lesson Learnt

5.1 Challenge 1: Road Segmentation & Pre-processing of Data

In similar projects, road segmentation is the hardest part and is crucial to the success of the project. To obtain an accurate mapping of the city, some papers suggest to manually label the city roads. Manual labeling would be tedious, yet labeling work has to be re-done when applying the same algorithm to a different city. Here are the challenge involving road segmentation:

1. How to efficiently segment the roads relatively quickly and without manual labeling?
2. Does the segmentation method accurately split up the roads in the city? A road could be very long spanning across the entire city. It doesn't make sense to predict one speed for the entire road.
3. Does the segmentation method help identify the corresponding road given geolocation data on this road?
4. Can the segmentation method be easily applied to other cities around the world? This step might not be mandatory, but was considered due to the uncertainty of using either the Didi data for Xi'an or Grab data for Singapore. In the early phase of this project, students were expected to experiment with Didi's Xi'an City dataset and later ran the code on Grab's Singapore dataset. This requirement is later relaxed and students were allowed to use the Didi dataset for the entire project.

To avoid manual labeling, some papers proposed to split up the city into small grids, so any geolocation can be mapped to a small grid. The grid has a rectangular shape and four geolocations defining the boundaries of the grid. It is easy to tell if a geolocation data is within a grid or not with simple comparisons of longitude and latitude. With this grid splitting approach, it is possible for a road to straddle between two grids and cause inaccuracy in the final prediction. It is also possible for a grid to contain no road at all. The size of the grid also needs to be tuned to fit different city roads. Engineers have to re-tune the parameter when applying the same algorithm on different cities because each city could have different urban planning approaches. Hence, the grid splitting approach is not an optimal solution.

To obtain a more accurate city mapping data, third party APIs could be used to avoid tedious work. For example, Google's Snap to Road API takes geolocation data and map it to the closest road. However, Google doesn't operate in China and Google's data can be inaccurate in China. On the flip side, Baidu is allowed to operate in China, but Baidu's data can be inaccurate outside of China. As nice as these API sounds, there's a rate limit for each service request and is subject to payment of fees especially when there are almost 200 GB of data to process.

As discussed in Section 3.3, a novel approach is proposed in this report. Using the street data from Open Street Map with some custom pre-processing work, all four points raised in this challenge is resolved.

1. With the help of OSM data, no manual labelling is required to segment the roads. The roads are segmented already by OSM.
2. The definition of a "way" in OSM doesn't necessarily mean the entire length of the way. When a very long road branches out to another street, OSM considers this segment of the very long road a "way". Hence, one speed is predicted for each "way", avoiding the problem of having one speed for a very long road. This segmentation method also aligns with the project's need. When Road A is congested and Road B intersects with Road A, it is highly likely that traffic will be diverted to

Road B. Before the intersection of Road B, Road A could be heavily congested. After the intersection of Road B, Road A would likely be a lot less congested.

3. Each "way" is associated with many nodes. Each node contains the longitude and latitude of the node. Given a huge set of nodes in the city, it becomes easy to map a given location of the driver, in longitude and latitude, to one of the nodes in the set. This mapping process can be done by almost any reverse geocoding library freely available on Github.
4. Since OSM is not limited by country as Baidu and Google are, OSM is community driven and contains street data for any parts of the world with relatively high accuracy.

5.2 Challenge 2: Custom Python Library on Azure

The reverse geocoding library used in this project is designed to be a python library, which can be installed using "pip install reverse_geocoder" and imported using "import reverse_geocoder as rg" in a python script. On Azure, the PySpark session requires a different approach. When the cluster is created, users have to submit a script action, which contains a URL link to a custom installation script. To achieve this, a custom script is created and uploaded to Github. This script can be found here and installs the reverse geocoding library using Pip and Anaconda. The script action documentation on Azure did a poor job explaining this process. It took a long time to find the right path to the Anaconda that PySpark is using.

5.3 Challenge 3: Resource Configuration on Azure

This challenge is unexpected in the planning phase. The project requires multiple steps of Spark job to achieve the final output. Most steps take less than one hour to complete; however, Step 1 takes about 4 hours to execute. Azure's default configuration times out and kills the operation after 60 minutes. Custom configuration is needed to extend this time limit on Ambari. Whiling digging through the configuration settings on Ambari, another setting is tweaked for performance gain. By default, the Azure Spark cluster is created with two workers, each with 16 cores. Ambari's default settings utilize both workers, but only 2 cores on each worker. When modifying the settings on the number of cores used, Ambari will restart some of the services, which takes around 5 minutes. If the job takes a short time to finish with 2 cores on each worker, it might not worth the time to increase the cores and wait for services to restart. For long jobs such as Step 1, it is absolutely worth the time to increase the number of cores to obtain performance gain.

For Step 1, Spark splits the job into 500 tasks assigning one task with some data to each core. With 2 cores each on two workers, it takes an average of 9 minutes to complete each task. In other words, it takes about 18.75 hours to finish Step 1. With 10 cores each on two workers, it takes only 3.8 hours to finish Step 1, which is roughly about 5 times faster. The cost of the server remains the same. Since each worker comes with 16 cores by default, the charge is the same regardless of whether all the cores are utilized or not.

5.4 Challenge 4: Missing Data

Missing data is a common problem in training machine learning models. In the case of this project, there are missing data in the intermediate result as well. Before training the models, the average speed of each road segment for every 5 minute interval between 6am to 12am is calculated (Step 3). However, the GPS data may not be existent for every 5 minute interval as Taxis may not be passing through the road segment

every 5 minute interval for the entire time frame. For less popular roads, the number of time intervals with no GPS data can be huge. There are 3 options to tackle this challenge:

1. Reduce the granularity of time intervals. (E.g. Make predictions every 10 minute interval instead)
2. Interpolate the missing data
3. Focus on roads with complete data

Reducing the granularity might not be a good solution, as it would affect the usability of the application. Traffic for highly popular roads can change quickly over time and reducing the granularity may result in the predictive model missing out on certain trends. For less popular roads, even if the granularity is reduced, there will still be missing data. Interpolating missing data is a complicated problem, and it may introduce error to the data set. For road segments with many consecutive time intervals with missing data, interpolation could not be done as it would be very inaccurate. These errors will affect the evaluation of predictive models.

The chosen method for this project is to focus on the roads with complete data first. Roads with large amounts of missing data for time intervals are often less traveled, and less likely to be congested as well. Thus, building a model for these roads are less of a priority. By focusing on the roads with complete data, we are able to better assess the effectiveness of the techniques used to build predictive model without introducing any inaccuracies through interpolation. After reviewing the results from the first round of predictive models, interpolation of data can be done for road segments with a few missing time intervals in the future.

5.4 Challenge 5: Encoding Spatial Characteristics

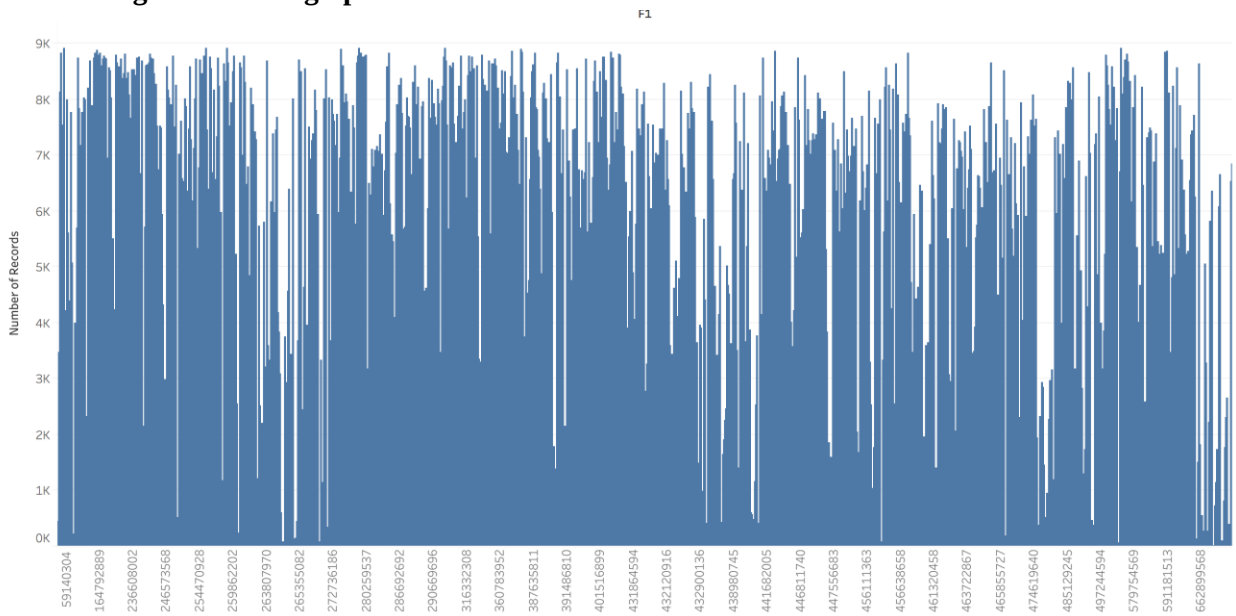


Figure 9: The number of 5 minute time intervals of each WayID

There is most likely spatial correlation between routes that are nearby - i.e. a road is likely to be influenced by the roads that are linked to it. However, this was not taken into account by our predictive model as it was difficult to 1) model the roads as a graph, and 2) attribute each speed datapoint to a particular direction on a bi-directional road. The figure above plots the number of 5 minute time intervals (that contains average speed data) of each way ID, and it is evident that there is a significant number of roads with very little data. This missing data prevents us from using speed of adjacent roads as features in our predictive model for encoding spatial relationship. Hence, we opted to only analyse the temporal (autocorrelation) relationship using SARIMA model for each of the selected way ID.

6. Personal Contribution

Han Xiao

Han Xiao was in charge of doing research on road segmentation in the planning phase. After finalizing the proposal, he implemented 3 steps: pre-processing of Open Street Map data, pre-processing of Didi Data and speed calculation. Since he is the first person in the team to run pyspark jobs and Spark in Scala job, he also figured out ways to speed up the processing speed by utilizing more cores on each worker and ways to include third party python libraries. Han Xiao also participated in the pitch talk, video presentation, proposal writing and report writing phase with the team.

Zhu Yilun

Yilun was in charge of doing research on existing methods of map matching during the proposal phase, and proposed the method of using Open Street Map data and an offline reverse geocoding library to help with the reverse geocoding process to map match the taxi GPS data to calculate historical speeds of road segments.

During the execution phase, Yilun was in charge of “Step 3”, the filter and aggregation of data according to time intervals to generate the features required for the training of the predictive models. Yilun was also in charge of the final map visualisation of the predicted speeds.

Voon Soo Yin

During the planning phase, Soo Yin first performed exploratory data analysis with Tableau 1) on raw GPS traces to understand the spatial distribution of the data, and 2) on processed speed data to find temporal patterns in average speeds of each selected way.

During the execution phase, Soo Yin prepared the training and test dataset, and researched and implemented suitable time series regression methods for this problem. She then compared and analysed the prediction results with RMSE, and visualised the predicted speed against the actual speed of the chosen model.

7. Project Summary

In this project, the speed of road segments given a time interval is predicted using GPS data obtained from taxi traces. The dataset used is provided by Didi for the Second Ring Road of Xi'An City for the months Oct 2016 to Nov 2016, which totals up to be 60 days, and 1536 million rows of data. The first 47 days of average speed data is used to train the various predictive models and the last 14 days of data is used for model evaluation. The evaluation was done by calculating the RMSE of the predictive model. Out of the five chosen models (Persistence model, Historical Daily Average, Elastic Net Regression, LSTM, and SARIMA), SARIMA was found to consistently outperform the rest of the models.

Given time limitations, the project solely focused on evaluating the effectiveness of various predictive models. While there could be errors in the map matching of GPS traces or the calculation of distances of road segments, an assumption was made that these errors will be minimal since the GPS data is collected at a granularity of every 2 to 4 seconds. This assumption and error with the distance calculation might lead to the calculated historical speeds, and hence predicted speeds, being slower than actual speed. This however, will not affect the evaluation and effectiveness of the predictive models. The evaluation of error of the map matching methods can be done by comparing the calculated road segment distance with the actual, and is something that can be done in the future.

8. References

[1] Hu, Yucheng, Minwei Li, Hao Liu, Xiaolu Guo, Xiaowei Wang, and Tiejun Li. "City traffic forecasting using taxi GPS data: A coarse-grained cellular automata model." arXiv preprint arXiv:1612.02540, 2016.
[2] Z. Yong-Chuan, Z. Xiao-Qing, Z. Li-Ting, and C. Zhen-Ting, "Traffic Congestion Detection Based On GPS Floating-Car Data," Procedia Engineering, vol. 15, pp. 5541–5546, 2011.
[3] P. S. Castro, D. Zhang, and S. Li, "Urban Traffic Modelling and Prediction Using Large Scale Taxi GPS Traces," Lecture Notes in Computer Science Pervasive Computing, pp. 57–72, 2012.