BMTC Bus Timing Prediction

Aayush Grover
IMT2016005
Aayush.Grover@iiitb.org

Aditya Gulati
IMT2016052
Aditya.Gulati@iiitb.org

I. Introduction

A. *The Problem Statement*

Given the start position (latitude and longitude), end position (it's just latitude and longitude) and the time when the trip started (date and time), give a good estimate of how long a trip will take.

B. *The Story Behind The Problem*

The current BMTC timetable is unrealistic at best. The assumption made is that the buses are moving at a constant speed everyday, at all times. This is the easiest way to do it, but traffic patterns vary a lot. Hence, this approximation ends up giving a timetable which no one even looks at. The idea behind this project is to fix this problem.

C. *Why Is This Hard?*

Traffic patterns vary a lot. We would need a lot of data to ensure consistency. Working with a lot of data is hard. We had only a small sample of 6000 buses and that alone could not be loaded into our RAM. Dealing with the sheer volume of data was one of the biggest challenges.

II. The Data

A. *What Did The Data Have?*

Our dataset came directly from GPS devices installed on a bunch of BMTC buses. We had data from about 6000 buses over one week. The data we received had the following columns:

- **Bus ID** : An ID associated with every bus
- **Latitude** : The latitude of the bus location
- **Longitude** : The longitude of the bus location
- **Speed** : The speed at which the bus was travelling
- **Angle** : The direction in which the bus was moving
- **Timestamp** : The time at which the entry was recorded (date and time)

B. *Challenges With The Data*

The data was not directly usable. Listed below are some of the challenges we faced with the data.

- The data was big - really big. It was about 14 GB making it impossible to load in our RAM at one shot.
- The data was not clean. There were outliers and zero values.
- The data was not structured - it was just locations with time-stamps. There was no structuring by trips.

Through the document, we will describe our approach to tackle these problems.

C. *What Did We Do With This Data?*

Our work with this data can be roughly divided into two parts:

- Pre-process the data: Bring it into a form where we can work with the data
- Engineer features: Come up with the best set of features that describe the data

III. Pre-Processing The Data

The following section contains details about the pre-processing steps we took and our reasons for taking them.

A. *Drop Unnecessary Columns*

We were told by a domain expert (Prof. V N Muralidhara) that the *speed* and *angle* data could not be completely relied on and were hence dropped.

B. *Remove Outliers*

On analysing the latitude and longitude, we saw that most of the data lies in the following range:

- *Latitude:* Between 12 and 13
- *Longitude:* Between 77 and 78

This makes sense since this is roughly where Bangalore lies. However, we observed that there were some entries at $(0, 0)$ and some entries at $(99, 999)$. These were clearly erroneous values since we knew that the buses were in the vicinity of

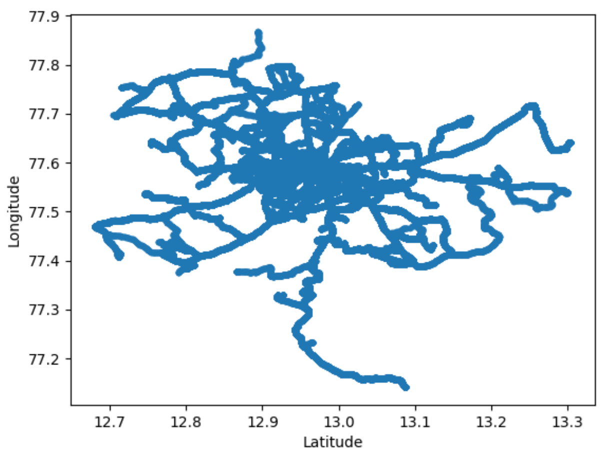

Fig. 1. Training Data (latitude vs longitude)

Bangalore. Also, given that the buses ply on fixed routes within the city, the locations could not possibly be this far. Thus, such entries were (safely) assumed to be device errors and were dropped. To do this, we took the mean of the data and dropped all values that were more that 10 away from the mean (either latitude or longitude). Given the variation in the data, 10 was a very safe margin.

Summary: Since the buses can't move very far from the mean, values which were obvious device errors were removed.

C. *Pick a Subset of Buses*

This was done for two reasons:

- There were a lot of buses - around 6000. This was a lot of data to handle. So, as a starting point, we decided to take pick (random) subset of the buses to train on. We picked 60 buses (about 13 lakh entries) as a starting point.
- Some of the buses were a part of the test set. We were given a dictionary which told us which buses were a part of the test set. These buses had to be removed from the training set.

With a subset of the buses, it was easier to build a model. The data we used was not representative of the original problem but we believe this is an OK starting point. Once we have a model that works well in a small part of the city, we could just feed it with enough data to learn traffic patterns scross the entire city.

Summary: Since the data was very large, a subset of buses was taken as a starting point.

D. *Split By Bus ID and Sort By Time*

Our idea of a general mode for this problem is simple: make a model which takes any two lat-long values and the time when the first lat-long value was recorded. To generate this data, we split our data according to the bus id. Once this is done, each file is sorted by time. This is useful for the next task.

E. *Combine Consecutive Rows*

The data we want to train our model on needed two lat-long points and the time taken to travel between these points. Since the data we had only contained locations of a bus at a particular time, some processing had to be done. This is where the previous step helped. As each bus was separated by ID and sorted by time, we could pick consecutive rows (since we can be sure they are the two closest points that are a part of one buses trip). Had we not segregated the data by bus ID, it would become hard two entries for one trip. Our new row had the two lat-long values, the time stamp when the first lat-long value was recorded and the time difference between the two positions (the time taken is what we are trying to predict). The bus ID was removed since it should not have any bearing on the time taken. The model should only learn about travel time from spatial and temporal features.

Summary: From the data split by bus ID, we combined two consecutive rows to generate data with two positions in space, the start time and the time taken to travel between them. However, there is one more small problem to be fixed.

F. *Interpolate Time*

There was one glaring issue with our new data, the time difference between any two entries for a bus was roughly constant (about 10 seconds). This made sense since the devices were designed to record the buses location at almost constant intervals of time.
Since we were trying to predict the time difference this was a problem. If more than 80% of our entries said that the time difference was 10 seconds, it really didn't matter what model we used. Any model would just take in two lat-long values and say the time taken to go between them was 10 seconds. To fix this, we decided to interpolate our data. This was done in two ways:

- Instead of picking only two consecutive entries, we picked a row and any one of the next 5 rows. This way, we extended the time difference from 10 seconds to a value between 10 and 60 seconds. However, most of the times were still multiples of 10.
- The next approximation we made to increase the spread of times was to assume the bus travelled in a straight line between the two positions. Then, we picked a random number between 0 and 1 (from a uniform distribution) and used this as the fraction of distance to consider. In short, we picked a random fraction of the time and used that as the time interval by adjusting the second lat-long point by means of the *section formula*.

Thus, we ended up with a better spread of data. However, if we are given two lat-long positions where the time taken in reality to cover that distance is more than 60 seconds, our model will struggle.
This will not be a problem in our case since we know that the testing data has points that are not very far away.

Summary: To ensure that all our data doesn't show a time difference of 10 seconds, we took an arbitrary point between the two lat-long points rather than the lat-long points obtained from the data.

G. *Shuffling The Data*

Before we could go ahead with training with our model (after engineering a few features) we shuffled the data so that there was no hidden dependence captured in the order in which the model sees this data.

IV. Engineering Features

To help our model perform better, we engineered the following features:

A. *Split Time Stamp Into Date and Time*

The time-stamp we had was a string which was a concatenation of the date and the time. For the model to be able to use the information here, we need to encode it into some sort of numbering. To start this off, we first kept the date and time as two separate values.

B. *Encode Time*

To encode the time of the day, we started by splitting the time stamp into three separate features: *hours, minutes and seconds*. This sort of encoding might have worked. Regardless, we felt that this was not the best that could be done. This sort of encoding increased the dimensionality of the problem and we wanted to avoid an unnecessary increase in the number of dimensions.
We used the following encoding scheme instead:

$$minutes = minutes + (1/60) \times seconds$$
$$hours = hours + (1/60) \times minutes$$

Thus, we encoded the time into one number between 0 and 23.99.
To ensure that our data was split almost evenly across the whole day, we made a box plot of the day of the week vs. the time taken as seen in Figure 2. The box plot showed us that our data is almost uniformly spread across all 24 hours everyday.

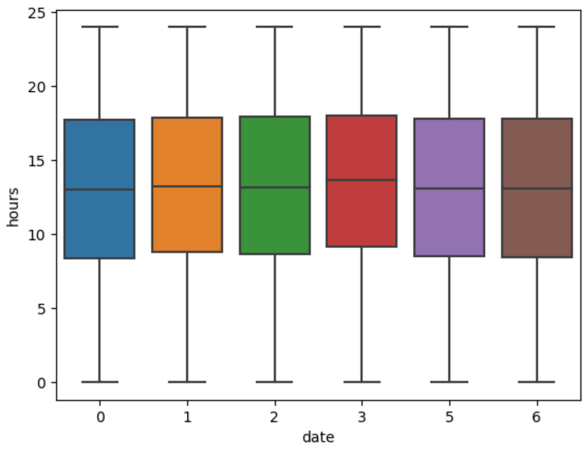

Fig. 2. Day of the week vs. Time taken

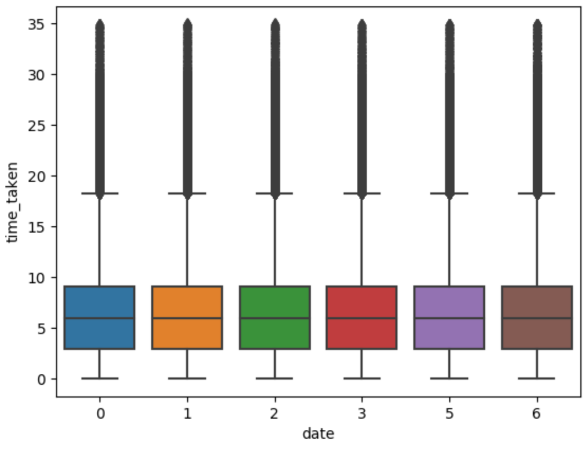

Fig. 3. Time taken vs. Day of The Week

C. *Encode Date*

There are multiple patterns in traffic cycles based on temporal features. There are daily, weekly, monthly and yearly traffic cycles. However, we had data only for one week. Thus the best we could try and do was capture the traffic changes in one day. Since these cycles would vary depending on the day of the week (weekdays have more traffic than weekends, for example) we decided that the only useful information that the *date* provided was the day of the week. Thus, date was encoded as a number representing the day of the week where 0 represents $Sunday$ and 6 represents $Saturday$.
Figure 3 shows the variation of time taken to travel against the day of the week. This plot is on the data where the time was interpolated (as described in section 3.6). What was surprising for us was that the distribution was exactly the same regardless of the day of the week. Regardless, we

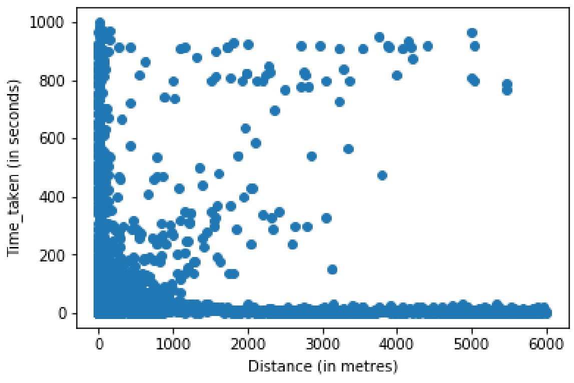

Fig. 4. Distance vs. Time Taken

included this feature because it seemed logical. To us it seems unreasonable to say the time taken to travel does not depend on the day of the week.

Summary: Given that we had data only from one week, the only useful information that the date provided was the day of the week.

D. *Calculate Distance*

Distance between two points logically does impact the time taken to travel between them. However, since we were passing the lat-long values of the points, our initial thought was that the model should be able to pick this up on its own. The results were surprising. On calculating the haversine distance and passing that as a feature, all our models performed better (it makes sense that something like a decision tree would need this since it can't ask $logical$ questions about the data, but we did think that linear regression would be able to pick it up). Thus, this was added as a feature.
Figure 4 is a plot of the distance vs. time taken after interpolating.

Summary: The distance between two points was added as a feature to help the model pick up on importance of how distance impacts time taken.

E. *Model Traffic Cycles*

Since we had data for a week we were trying to model the traffic cycle over a week. Each day can be seen as having two different traffic peaks - one in the mornings (when people leave for work) and one in the evenings (when people leave work). This is something that is well known about Bangalore traffic (the only domain knowledge needed here is hours wasted sitting in a car stuck in this traffic wondering about why you got so late).
Thus, we can see that a week can be divided into 14 parts. Since this is expected to repeat on a weekly basis, each week can be seen as having a time period of 14. To model this aspect of the time of the day, the following 28 features were added:

$$sin\_time\_i = sin(\frac{2 \times \pi \times i}{14} \times encoded\_time)$$
$$cos\_time\_i = cos(\frac{2 \times \pi \times i}{14} \times encoded\_time)$$

where i ranges from 1 to 14.
These 28 features were expected to help the model see the time of the day in terms of the cycles we expect. Turns out, they did - adding these features made a significant improvement to our score. Also, it goes without saying that the old encoding of the time was no longer used as a feature directly, it was passed to calculate these features and then removed from the training data.

F. *Binning Lat-Long Values*

We divide our lat and $long$ values into 1000 bins each where each bin value is arithmetic mean of all the values in the bin.

G. *Our Final Dataset*

The final data that we trained on had the following features:

- **Lat1**
- **Long1**
- **Lat2**
- **Long2**
- **Day of the week**
- **Distance**
- **Encoded time features (28 of these)**

V. Modelling The Data

Listed below are the various models we tried, our reasons for using them, and the scores we obtained. The error metric used on Kaggle is RMSE. We split our data into a train and validation set ($75 - 15$ split). The test data was on Kaggle.

A. *Linear regression*

The time taken by the model was a non-linear parameter, so initially we did not expect linear regression to work. However, the time taken is sort of a sinusoidal function. Adding the sinusoidal features described in section 4.5 should have taken care of this. Our assumption was that these features should make the data linearly separable. Hence we tried linear regression. The scores we obtained on the validation set were:

- R^2 *Score:* 0.00014255317949296575
- *RMSE:* 9.602649197273449

Clearly this did not perform very well. Our guess was that our assumption of the data being linearly separable was wrong. So we thought a tree based of approach might help.

B. *Decision trees*

As mentioned above, we thought the data might not be linearly separable. Hence, we tried out decision trees. Also, features like day of the week, which are not directly correlated with the target variable can be used better here. Below are the scores we obtained on the validation set:

- R^2 *Score:* 0.12395405511599933
- *RMSE:* 9.349433833966861

These results are much better than linear regression. Our guess for why this happened is the day of the week parameter that we passed. Since trees worked so well, we thought an ensemble of trees would work much better.

C. *Random forest*

Given the results we obtained with decision trees, we expected an ensemble of trees to give us a much better result. Below are the scores we obtained on the validation set:

- R^2 *Score:* 0.13077148652226867
- *RMSE:* 9.312983911267393

The R^2 score and the $RMSE$ were marginally better. We did expected an ensemble to perform much better. Out of curiosity, we thought of using a boosting based approach.

D. *Ada Boost*

We did expect to get a better score on Ada Boost. However, Ada Boost did not converge (even on our reduced training set). We attributed this to having a very large data set and hence did not wait for this model to be completely trained.

E. *Our Final Model*

Given the errors obtained from the above models, we decided to use **random forests**. We did not try any other mixtures of models for reasons described in detail in section 5.7. We knew that the main reason for a bad score was lack of data and any other model would be built over these basic models and would most probably suffer from the same problems.

F. *Running Our Model On The Test Data*

Our model was trained to take two lat-long points as input and predict the time taken to travel between them. The test data was structured a little differently. It had a trip consisting of 100 points from a trip that a bus took and the time when the trip started.

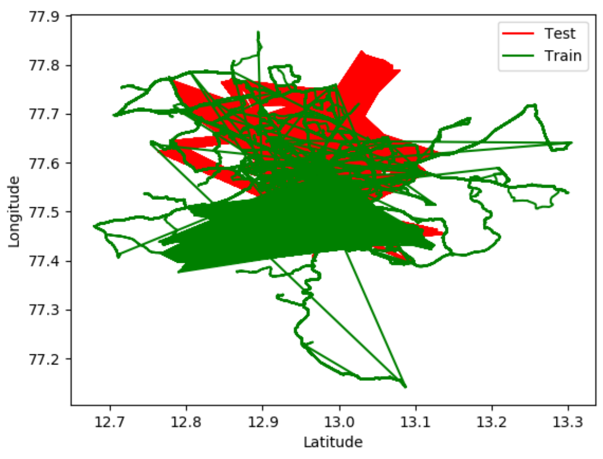

Fig. 5. Training and Testing Data

To run our model on this data we picked two lat-long points at a time and the start time of this trip (between two points) was the start time of the previous trip plus the time taken for that trip (time taken to travel between those two lat-long points). For the first pair, the start time was given.
In this manner, we iterated over all 99 pairs and predicted the time for the total trip.

Summary: Since the structure of the training and testing data was different (slightly), we couldn't directly run our model on the test data. The differences are highlighted above.

G. *Where our model failed*

According to us, our model is performing very well. When we test on our own data, we get a reasonable score. Also, as a sanity check we took two arbitrary points within the area we trained on and compared our models prediction to the time predicted by Google Maps. The time difference was within a minute.
However, when we submitted our predictions on Kaggle we got an RMSE of about 15000. We believe this is because the test data includes areas our model hasn't been trained on (we took a random subset). To check this, we plotted the train data against the test data as seen in figure 5. There are clearly areas which we haven't covered. We believe these areas are what caused our error. We don't believe that this means that are model is not generalizable, it just means that the training data is not representative of the problem (which is true, we've trained it only on 60 out of 6000 buses). So our score doesn't mean that our model is not good, it just needs to see more data.
We do not have the computation power needed to train a model on such large amounts of data. Instead, we thought we could put our data on the cloud and let the computations happen there. However, given the bandwidth we have, it

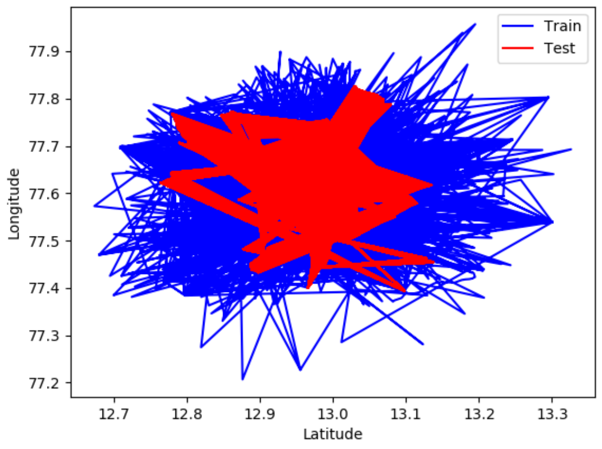

Fig. 6. Training and testing data with all buses

would take about 100 hours to finish the upload of the raw data. Hence, we chose not to do that.
The best we could do was supply the model with all the bus trips but running it overnight just gave us data for one day. This data has been plotted in figure 6. As we can see, in such a situation the model would perform well. However, our model performed worse on this data since it had data only from one day.

Summary: If we had more data and more computation power, our model would perform much better than this. We used a small training data set so that the computation is feasible but that caused errors because there is no way that the subset we took can be representative of the problem (since the subset we took was too small).