DATA SCIENCE WORK SAMPLE: TELEMATICS

February 17, 2019

Mohammed Samiul Saeef
The University of Texas at Arlington
Department of Computer Science

Contents

1 Overview

Figure 1 shows a flow chart of the project.

2 Project Setup

1. Written in Python 3.7.1 (Anaconda distribution 4.5.12).
2. The program was run on an Intel Core i7-7700HQ 2.80GHz machine with 16 GB installed memory.

3 Assumptions

1. All timestamps are in sorted order.
2. There is no clear definition given for **accuracy** in Mobile trip data. I assumed that this value indicates how accurately the mobile phone sensor measured speed during a specific timestamp. If accuracy value is very low, the speed reading was almost accurate. If the value is high, the data point is probably an outlier, i.e: noise in data.
3. OBDII data does not have any accuracy measure. I assumed that OBDII data is accurate, thereby I did not perform any outlier detection/removal/replacement on OBDII data.
4. Each device is using a specific speed unit, i.e: if speed of a trip in OBDII is measured in miles per hour (mph), that means all OBDII speeds are in mph. At the same time, mobile device may have different unit like kilometers per hour (kmph) while OBDII is using mph, and vice versa.
   The bottom line is, I assumed that for a specific device the speed unit is fixed.
5. There will be no missing data. As the sample data set does not have any missing data, I did not handle missing data in my solution.

4 Data Preprocessing

4.1 Timestamp shifting

The plot for each trip was shifted to left on the graph so that the timestamps starts at zero. This was attained by subtracting a trip's minimum timestamp from all timestamps of the trip.

4.2 Outlier handling

For each trip, I normalized the accuracy values to keep them between 0 and 1.0. This normalized value now represents the probability for a data point to be a noise in the data. If accuracy is zero, the data point will not be considered as a noise. I used this formula for normalization:

$$accuracy_i = \frac{accuracy_i - min(accuracy)}{max(accuracy) - min(accuracy)}$$

This normalized accuracy readings were used to scale the speed data. Instead of detecting outliers, I readjusted the amplitude of the data. To do this, besides the speed value of a point in the data set, I considered an **estimated speed value** of that point. The estimated speed value is (i) for the first timestamp of the trip: the trip's mean speed, (ii) for other timestamps: the speed value of the previous timestamp. The absolute value of the difference between the original speed and estimated speed is the range of possible adjustment. Now, using the accuracy of the data point, I calculated the new adjusted speed value of a data point i:

$$newSpeed_i = speed_i + (speed_i - esimated\_speed_i) \times accuracy_i$$

Figure 2 shows a simple sketch to understand the process. Here S1, S2 and S3 are three consecutive speed values in a trip. If S3 is an outlier, the estimated value for S3 will be same as its previous value S2. Now the range of adjustment will be the distance OS3 (red line) in the figure. The new position of S3 after readjusting will be somewhere in between O and S3 along the line OS3, depending on the magnitude of accuracy of S3.

Thus if accuracy is 0 (lower value means more accurate), speed will remain unchanged. Higher the accuracy, higher will be the probability of its being an outlier, thus greater

will be the adjustment. An example of outlier handing can be seen in figure 3.

4.3 Reducing information using interpolation

In the given data sets, for some trips there are more than thousand data points for a single trip. Running polynomial time similarity measure algorithm for this large set of points would be inefficient. Thus instead of thousand data points, I decided to use less then hundred points while not ruining the trend of the original time series. For this, I set the parameter *interpolation_size = 50*, which means there will be 50 equidistant timestamps over the trip interval. I used **Piecewise Cubic Hermite Interpolating Polynomial (Pchip)** [1] to get the speeds for the corresponding 50 timestamps. Initially I was doing Cubic Spline Interpolation, but it was causing overshoot problem (negative speed value generation during interpolation), so I swtiched to Pchip. Figure 4 shows an example of an interpolated mobile data.

4.4 Scaling amplitude

As per point 4 in section 3, a device always measures using a single speed unit, but different device may have different speed units. Speed is a **ratio** [2] type numeric data, i.e: 0 (zero) speed implies non-existence of speed. So between any two units of speed there must be a fixed ratio. The naivest method to calculate it would be taking the ratio of the mean speed of two data sets. But what I observed from my driving experience and observing the trip data sets is, there is usually lots of very low values (near or equal to zero) which possibly occur due to the braking events (brake at signal, sudden hard-brakes, waiting for long in the red traffic light, etc.). These frequent low speed values do not contribute much to vary the ratio between two speed units. So, I

- computed mean speed of a single device data set.
- then I filtered out the speed values of the data set which are lower than the mean, and created a new set with only speed values **above the mean**.
- Now I calculated the mean of this new set of speeds.
- This way, I got two mean values: one from Mobile data, another from OBDII data. I calculated the ratio between these two means and got a speed scaling factor.

I scaled the amplitude of the plots using this scaling factor so that plots of the two devices scale to a similar range.

5 Finding Matched Trip Pairs

5.1 Slicing and Matching

Step 1 - Slicing OBDII: For each Mobile trip, I tried to find its best match among the set of OBDII trips. To find the matching pair, the algorithm picked up a Mobile trip and then checked its similarity with all OBDII trips. When the algorithm picks up an OBDII trip to compare with a Mobile trip, it starts comparing at the initial timestamp of the OBDII data. Then iteratively it keeps slicing the OBDII plot along the time axis, shifting it to the left and makes a new comparison with the Mobile data plot. Figure 5 shows how the slicing works. In the left graph, OBDII seems to have a match with the Mobile, but it's right shifted. So the algorithm takes the OBDII plot and starts slicing and shifting it to the left along the time axis and computes similarity between the plots after each slicing. I named it 'slicing', because in the right graph almost 33% data of OBDII is lost due to the 'slicing'. In each iteration the Mobile plot and the sliced OBDII plot are compared using **Dynamic Time Warping**.

Step 2 - Slicing Mobile: The same process above is repeated, except this time the order is swapped. The algorithm now compares each OBDII trip agains all Mobile trips to find the best match, and this time the Mobile data is sliced instead of OBDII. **The reason** to do this step is to handle cases where shifting the sliced OBDII trip to left does not find any match. Consider figure 6. If we keep shifting the OBDII plot to the left, we will never find the match with Mobile plot. For this, we need to shift the mobile plot to left. So here Slicing OBDII will not work, but Slicing Mobile will do the job.

Now after step 1 & 2, we have two sets of matching pairs.
set A: For each of the Mobile data, its best match with an OBDII trip
set B: For each of the OBDII data, its best match with a Mobile trip.

Using B, I updated the pairs in A. To understand it better, lets consider an example where we denote Mobile trip as M and OBDII trip as O. Say, we have a pair **p1** (M_i, O_j) in A, and a pair **p2** (O_k, M_i) in B. If the score of p2 is **better** than p1, we will replace p1

with a new pair **p3** (M_i, O_k). Because we have found a better match for M_i in B, which is O_k. The updated set A is the **final output** of my program.

Parameters: Two parameters *slicing_factor* and *match_threshold* are used for slicing, for which I used values 0.5 and 0.3 respectively (values should be between 0 and 1.0). *slicing_factor* determines how much a trip data will be sliced, e.g: *slicing_factor=0.5* means we will slice upto 50% of the data, then we will stop. If this value is large like 0.9, we may get small slice which will not find a good match (consider the case when a tiny slice of OBDII data matches with a tiny interval of a Mobile trip, is it even a proper match?). *match_threshold* defines the notion of 'similarity', i.e: what percentage of overlap between plots will be considered as similar. For example, *match_threshold=0.3* means if a pair of intervals (not the entire plot) has less than 30% overlap, we will ignore this pair of trips and skip measuring their similarity.

5.2 Dynamic Time Warping

To compute similarity between two time series, I used Dynamic Time Warping (DTW) [3]. The reason for using DTW is:

1. The time interval in the Mobile and OBDII plots are not same, that means there is no one-to-one correspondence between the two set of timestamps. To measure similarity using euclidean distance or correlation the gap between timestamps should be same for Mobile and OBDII data in order to find distance/correlation between data points.

   Consider figure 7. Here I have shown 4 **consecutive** data points (Black = Mobile, Blue = OBDII) from a Mobile and OBDII plot each. Apparently, shifting the OBDII plot a bit right will yield a good match, yet we can not do correlation/euclidean measurements as blue points and black points are not equally spread (these methods require one-to-one timestamp correspondence). This is the main reason behind choosing DTW to measure similarity between time series. However, correlation/euclidean approach would also be possible, but in that case every time we compare two trips, we need to create two interpolated plots for each with same time interval. This is an expensive approach, in contrast to mine which only needs to interpolate a trip just once.

2. Complexity of DTW is $O(nm)$ where n is the length of the first time series and m is the length of the second time series. While performing DTW multiple times on long time series data, this can get very expensive. To speed up the process I enforced a locality constraint [4]. This works under the assumption that it is unlikely for a data point in Mobile to match with a point in OBDII if their timestamps are too far apart. The threshold is determined by the parameter *window_size*. This way, only mappings within this window are considered which speeds up the inner loop in DTW.

6 Results

I ran the program using different values of the parameters, but **not** varying over a large range. The parameters and their range of values were like below:
$window\_size = 5 \sim 10$
$interpolation\_size = 50 \sim 100$
$scaling\_factor = 0.5 \sim 0.6$
$match\_threshold = 0.3 \sim 0.5$

The program generates 44 figures corresponding to mobile trips $0, 1, 2, 3...43$ **(zero based indexing)** in the *mobile_trips.json*. I have shown the match for the 19th mobile trip in figure 8.
In figure 8, green and red line represent mobile data and OBDII data respectively. Two vertical black lines mark the matched interval, which is the interval of time during which sensor data from both sources is present. This is a proper subset of both trips. The time axes are scaled to a common time so that the matched intervals lined up on the graph. We can see, the best match found for mobile trip 19 is OBDII trip 22.

Execution time to complete running the program and generating figures was $160 \sim 200$ seconds depending on the magnitude of parameters, which is roughly 3 minutes. For the provided large data sets and accuracy of the solution, the execution time seems good.

7 Attempts and Ideas for Improvement

1. I used a scaling factor to match the Mobile plot amplitude with OBDII amplitude. I implemented the scaling factor measuring technique based on the my driving

experience in and empirical observation of the entire data set. However, some existing standard method could be used to compute such a scaling factor.

2. The system could be designed in way so that no scaling factor is needed at all. I now feel intrigued to develop a robust algorithm which will require no scaling factor and will detect trend similarity between time series pairs even if their amplitudes differ.

3. Instead of finding time series similarity using dynamic time warping, a carefully chosen correlation based method may have yielded better result. However, I also tried using **Pearson correlation** (creating interpolated time series pairs of Mobile and OBDII trip, where timestamp intervals for both data are same) as similarity measure, but that performed significantly worse in finding trip matches.

4. Outliers could be handled differently. Instead of normalizing the accuracy values of each Mobile trip, using a logarithmic scale for the accuracy data might be a more insightful approach. I also tried a **weighted dynamic time warping** technique where I used scaled Mobile accuracy values as weights, but that did not perform well either.

5. There is a trade-off between execution speed and result accuracy. I tried to balance between these two. Next time I will use more powerful resources so that I can entirely focus on getting more accurate results.

References

[1] `https://docs.scipy.org/doc/scipy-0.18.1/reference/generated/scipy.interpolate.PchipInterpolator.html#scipy.interpolate.PchipInterpolator`.

[2] `https://towardsdatascience.com/data-types-in-statistics-347e152e8bee`.

[3] `https://www.aaai.org/Papers/Workshops/1994/WS-94-03/WS94-03-031.pdf`.

[4] `http://alexminnaar.com/time-series-classification-and-clustering-with-python.html`.

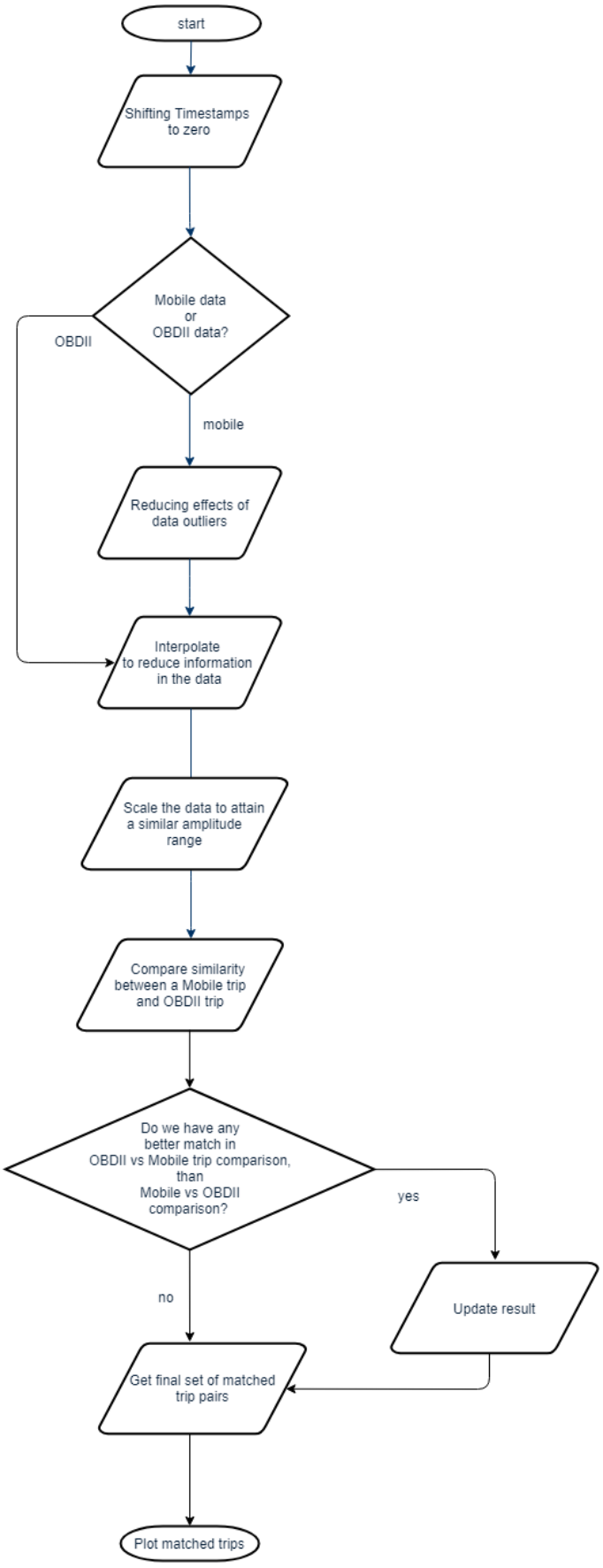

Figure 1: Flow chart of the project

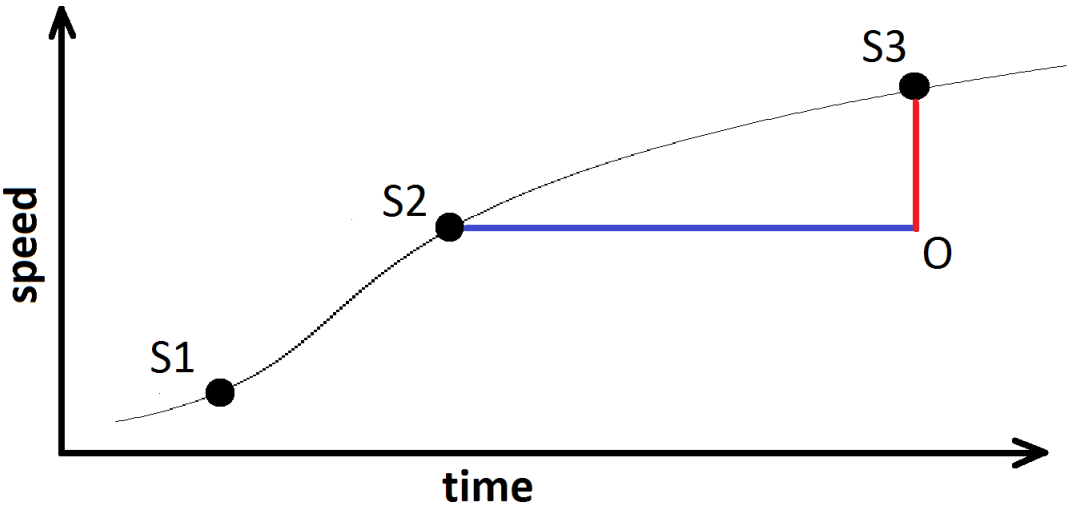

Figure 2: speed estimation

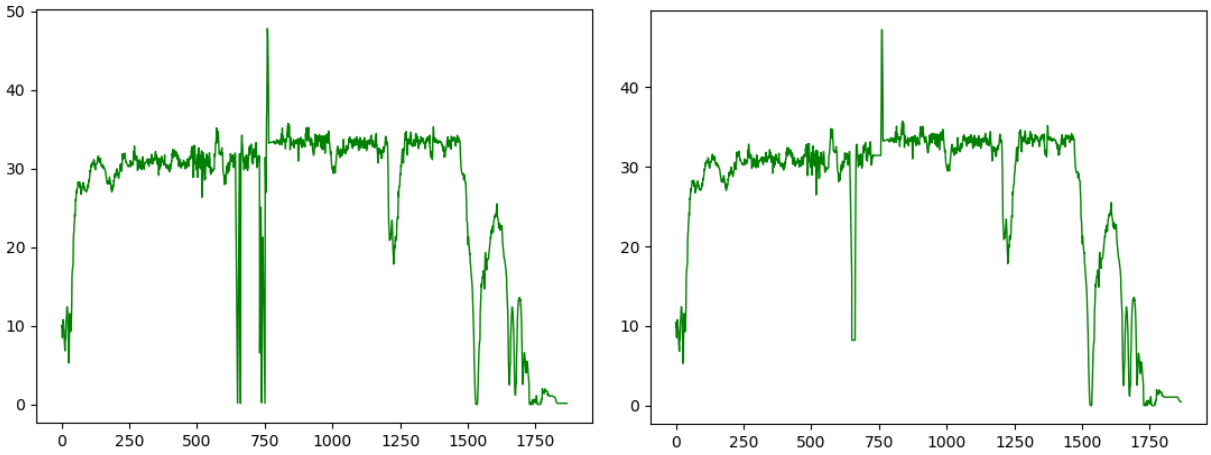

Figure 3: Outlier effect minimized for Mobile trip #22: with outliers (left), after reducing outlier effect (right)

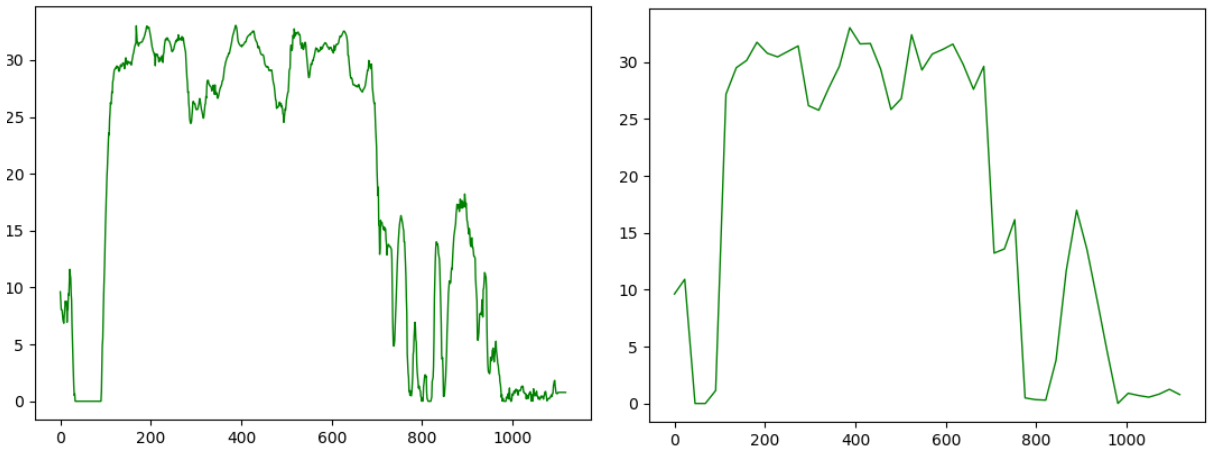

Figure 4: Mobile trip #0 before interpolation (left), after interpolation (right)

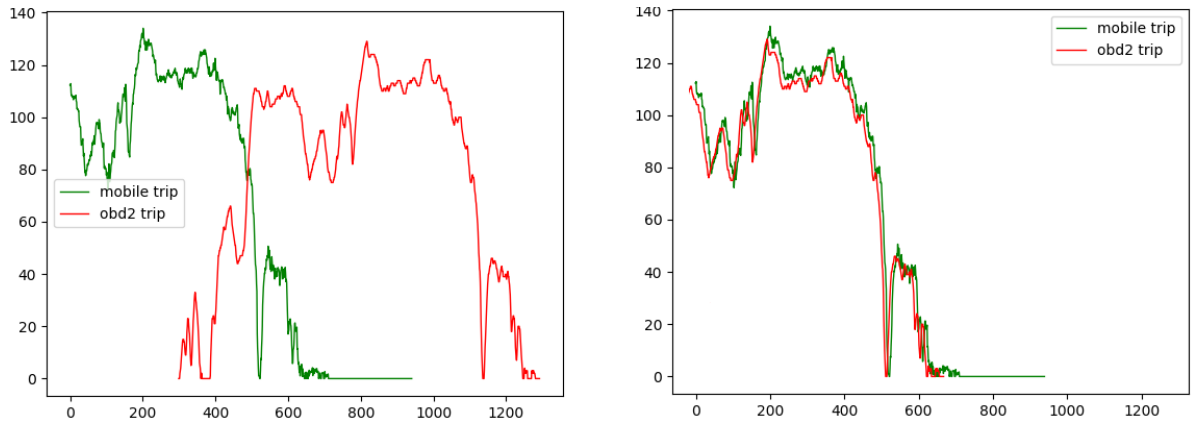

Figure 5: the plots are in their original position (left), OBDII plot is shifted to left side (right)

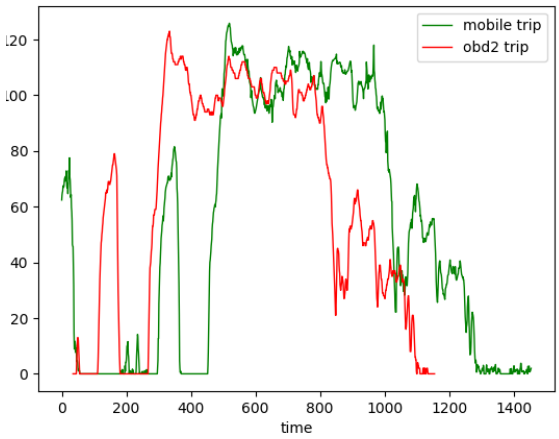

Figure 6: OBDII data is left shifted, so it needs to move a little bit right

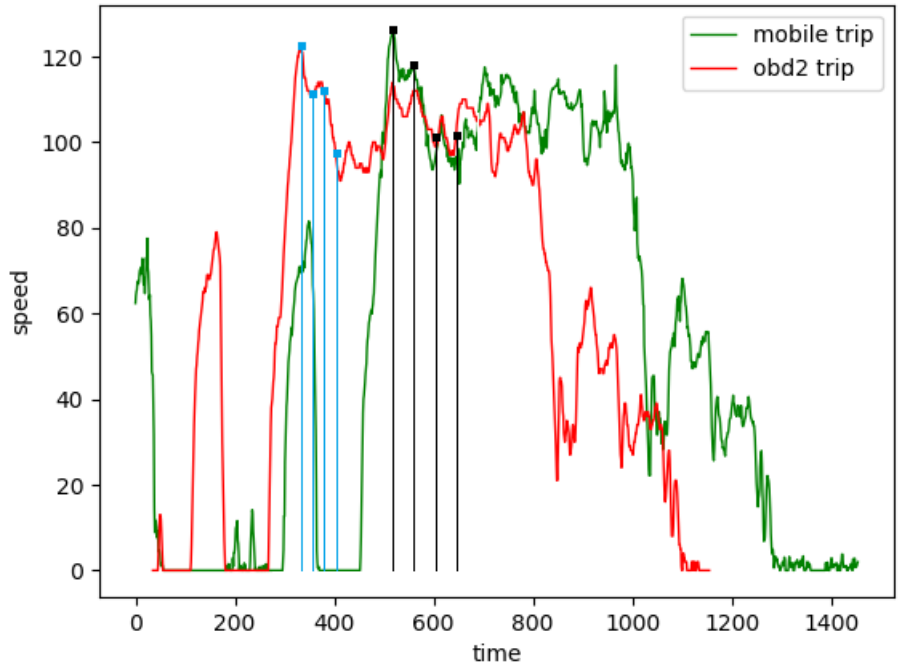

Figure 7: The points of Mobile data have larger timestamp intervals than OBDII data

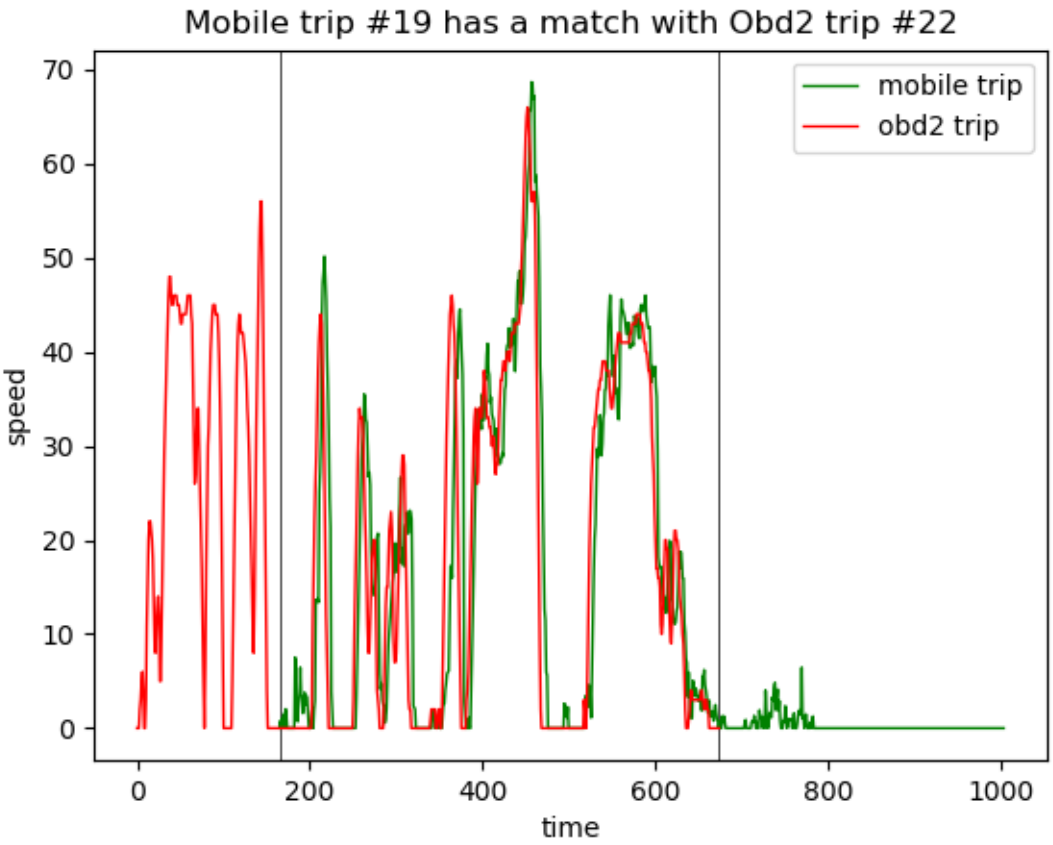

Figure 8: a sample of two matched trips