Data Analysis Report: Root Insurance Co

Issac Lee[a]

[a]Department of Statistics & Actuarial Science, 241 Schaeffer Hall, Iowa City, Iowa 52242-1409

This version was compiled on November 4, 2019

This report illustrates how I approach the solution and make a decision for Root insurance's data analysis project. The goal of this project is to find the best-matched OBDII trip with a given trip data, which is recorded by a customer's smartphone. In the first chapter, we learn how to load the data set into the system. Next, we pick one sample trip from each sensor and explore the data structure. Using these sample data, we build a model and extend it to the whole data set in the later chapter. Lastly, we illustrate the instructional manual for the written `R` functions in the `Rootinsuarnce` package.

Data Preparation. There are two telematics data set from independent sensors: GPS in a smartphone, OBDII. The two data sets are stored in two separate files. Here we assume that the two `JSON` files are extracted in the working directory, which means you have the following two files in the working directory:

mobile_trips.json, obd2_trips.json

Data load and structure. Since the two files use the `JSON` format, we use the `jsonlite` package to load the data.

```
library(jsonlite)

# read mobile trip & obd2 trip
mobile_data <- fromJSON("./mobile_trips.json")
obd2_data <- fromJSON("./obd2_trips.json")
```

The smartphone data set consists of 44 trips, and the OBDII data set consists of 41 trips. The column names of each data set and the data types are as follows:

- OBDII data
  - trip_id (`char`), timestamp (`dbl`), speed (`int`)
- Mobile data
  - trip_id (`char`), created_at (`dbl`), timestamp (`dbl`), speed (`dbl`), accuracy (`int`)

Among these columns, we use only `timestamp` and `speed` columns for the detecting algorithm.

Visualization of sample trips. Fig 1 shows the speed graph of a sample trip from OBDII data. As we can see, the OBDII recorded the trip for about 1,200 seconds, and we do not know the unit of the vehicle speed. Fig 2 shows the speed graph of a sample trip from mobile data. Note that the scale of the x-axis has been adjusted to zero. By examing the data set, we can realize that the first trip from each source corresponds to each other, as in Fig. 1 and Fig 2. The whole trip data from the smartphone corresponds to the trip data from OBDII around 300 seconds.

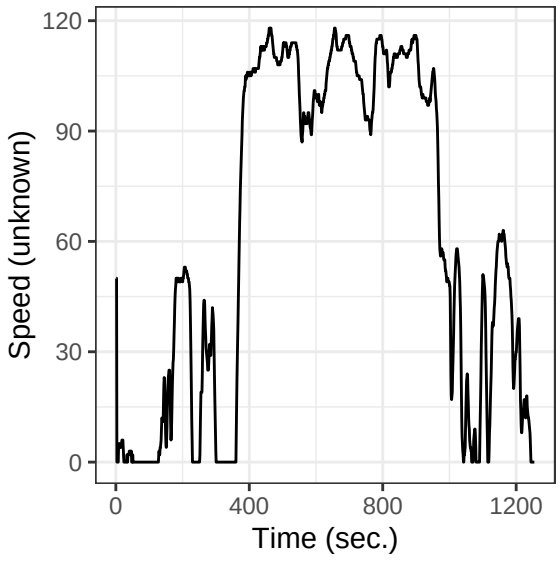


Fig. 1. A sample of speed graph of OBDII (the 1st trip).

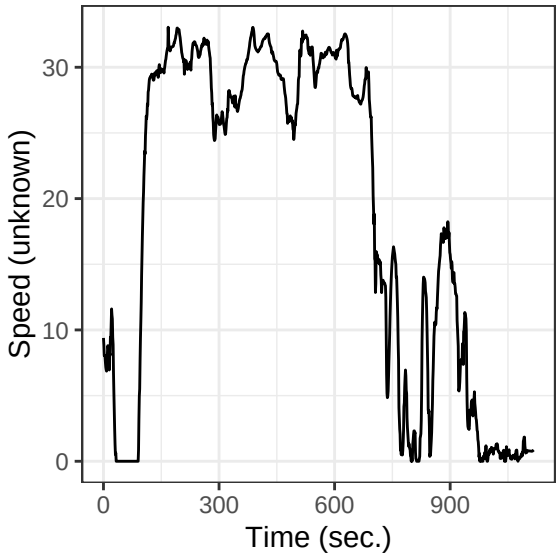


Fig. 2. A sample of speed graph of Smartphone (the 1st trip).

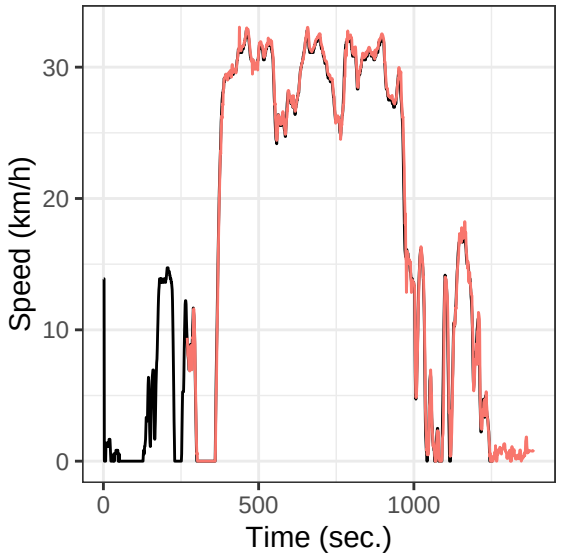


Fig. 3. A sample output of matched speed graph.

Determine the conversion factor between speed scales. Using this knowledge we found, we can figure out there is a conversion factor between the two different speed scale from each source. If these sensors recorded the same trip, their maximum speed of the trip should be the same. Thus, we can calculate the conversion factor as follows:

```
# Conversion factor
max(obd2_trip_sample1$speed) /
  max(mobile_trip_sample1$speed)
```

```
#  [1] 3.570348
```

We can see that it is around 3.6. There are many units for speed, such as miles per hour (mph), kilometers per hour (km/h), etc. Using the cue that the smartphone sensor should use either `mph` or `km/h`, we can easily guess that the OBDII uses `m/s` since the relationship between `km/h` and `m/s` is as follows:

$$1\ m/s = 3.6\ km/h$$

Thus, using the conversion factor and the lagging time (about 300 sec.), we can guess our final output should be similar to Fig 3. From now on, we use `km/h` as a speed scale since the resulting plot re-confirms that the conversion factor is right.

The detection algorithm of the lagging time. The next step is to automatically determine the lag time given that we have two matched trips. We use the same sample trips used in the previous section. In Fig. 3, the exact lag time for smartphone trip data was 270 seconds. To detect the lagging time, we need to consider every possible combination of these two trips by using the sliding window algorithm. This Wikipedia web page has an excellent visualization of the concept of the algorithm. We consider the speed graph from OBDII as a fixed function and the speed graph from a smartphone as a floating function in the algorithm.

Let $x \in \mathbb{R}^+$ be a real positive vector of size n_x whose elements represent the speeds or a trip from a smartphone, while $y \in \mathbb{R}^+$ and n_y represents the speed and the size of the speed vector from OBDII respectively.

$$x = (x_1, x_2, ..., x_{n_x})^T$$
$$y = (y_1, y_2, ..., y_{n_y})^T$$

To implement the sliding window algorithm, we uses a dummy variable `k` from 1 to $n_x + n_y$ to search all the possible overlapped combination of the two graph. For example, when $k = 1$, we consider the situation where x_{n_x} and y_1 are overlapped each other. When $k = 2$, (x_{n_x-1}, x_{n_x}) and (y_1, y_2) are considered. Thus, for any $k \leq n_x + n_y$ where $k \in \mathbb{N}$, the overlapped vectors, x^* and y^* can be written as follows;

$$x^* = (max(n_x - k, 1), ..., n_x - max(0, k - n_y))^T$$
$$y^* = (max(k - n_x, 1), ..., min(n_y, k))^T \quad [1]$$

Equation 1 shows the compact expression for the three cases:

- Case 1: $k < n_x$ and $k < n_y$
  - $x^* = (n_x - k, ..., n_x)$
  - $y^* = (1, ..., k)$
- Case 2: $k > n_x$ and $k < n_y$
  - $x^* = (1, ..., n_x)$
  - $y^* = (k - n_x, ..., k)$
- Case 3: $k > n_x$ and $k > n_y$
  - $x^* = (1, ..., n_y - (k - n_x))$
  - $y^* = ((k - n_x), ..., n_y)$

Measure for the similarity. There are many measures for the similarity of the two functions whose domains are the same: area under the difference between the two functions, the maximum difference of the two functions, etc. Among these, we use the following measure for detecting the similarity between the two-speed graphs:

$$f(x, y) = \frac{\sqrt{\left(\sum_{i \in A} (x_i - y_i)^2\right)}}{|A|} + \frac{\lambda}{|A|} \quad [2]$$

where the vector x and y are the speed vector from smartphone and OBDII, and the set A is the collection of the pair of coordinates of OBDII and smartphone speed vectors overlapped each other for fixed k. Note that the function $|.|$ indicates the cardinality of a set. The reason for the division in Equation 2 is to calculate the average of the errors.

Also, the penalty function, the second term in Equation 2, prevents the case where the dissimilarity of a vector is so small since the length of the overlapped is too short. We can put the weights on the penalty using λ, and we use 10 for λ value for this project. Since the value of $f(x, y)$ decreases when the two vectors, x and y, are similar to each other, the interpretation of the measure should be dissimilarity of the two vectors. Fig. 4 shows the dissimilarity concerning the k from 1 to 2354, which is the summation of the length of the speed vectors. The index, which makes the dissimilarity to be the smallest value, is $k = 1374$. According to Equation 1, this corresponds to the 255th timestamp of the OBDII trip.

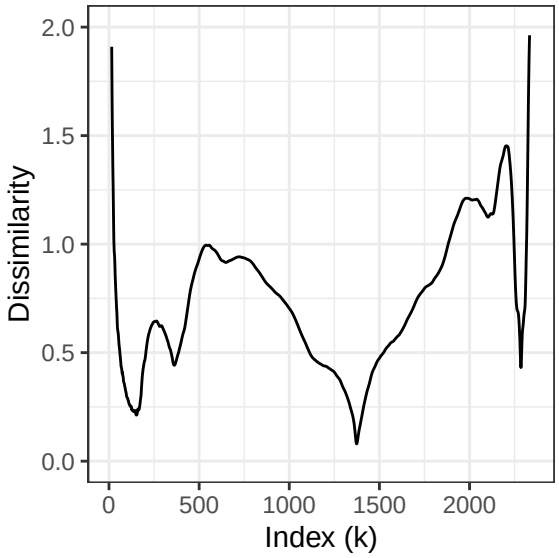


Fig. 4. Dissimilarity measure for with respect to k. First trips from OBDII and smartphone.

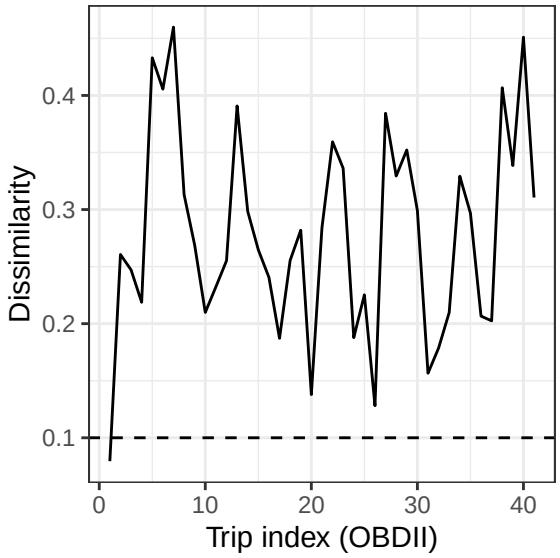


Fig. 5. Minimum values of dissimilarity for each trips in OBDII data set

Find the best-matched trips. In the previous section, we have discussed how to find the lagging time using dissimilarity measure. To find a best-matched OBDII trip for a given smartphone trip, we find the OBDII trip whose minimum of the dissimilarity with the given smartphone trip is the lowest among the whole collection of OBDII trips in the data set. However, since we can not guarantee that the lowest minimum of the dissimilarity implies that the two trips matched each other, we set 0.1 as a threshold. Thus, if the lowest minimum dissimilarity is less than 0.1, we conclude that the pair of OBDII and smartphone trips matched with each other. Fig. 5 represents the minimum of dissimilarity for each trip in OBDII data with the first trip in the mobile data set. The dotted line, in Fig. 5, indicates the threshold for the matched trip. Since the dissimilarity of the first trip is lowest among OBDII trips and it is less than 0.1, we select that the first trip in OBDII data as the best-matched trip with the first trip in mobile data.

User manual.

Installation. To install the `RootInsuarnce` package, save the `RootInsurance_0.1.0.tar.gz` file into your `R` working directory, then run the following code.

```
install.packages("./RootInsurance_0.1.0.tar.gz",
                 repos = NULL, type = "source")
```

After the installation, you can load the package as follows:

```
library(RootInsurance)
```

Load telematics data. Put your telematics file (JSON file format) in the working directory. `LoadTelematics` function uses `jsonlite` package to load the telematics trip using a list format in `R` as at the beginning of this document.

```
# read mobile trip & obd2 trip
mobile_data <- LoadTelematics("yourfile.json")
obd2_data <- LoadTelematics("yourfile.json")
```

Find the best OBDII trip for a given smartphone trip. The following `R` code selects the first smartphone trip in the mobile data set as a target trip. Next, it finds the best-matched OBDII trip using the `FindBestTrip` function and saves the result into the `match_info` variable.

```
# Select the second trip in the mobile data set
seleted_trip <- mobile_data[[1]]

# Find the best trip from OBDII data set
match_info <- FindBestTrip(seleted_trip, obd2_data)
```

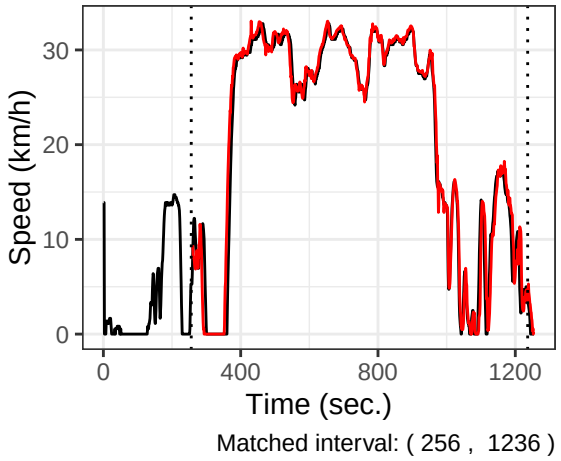
The `match_info` variable has the following seven information:

```
summary(match_info)
```

```
#                 Length Class  Mode
#  start_index_ref 1      -none- numeric
#  start_index_tar 1      -none- numeric
#  overlap_length  1      -none- numeric
#  dissimilarity   1      -none- numeric
#  macthed_trip    1      -none- numeric
```

Visualization of the matching info.. Once we find the best matching trip, you can visualize it using `VisTrip` function as follows:

```
# Visualization
VisTrip(seleted_trip, obd2_data, match_info)
```



Note that if there is no best matching OBDII trip, this function shows a possible candidate OBDII trip with a warning title.

```
# Select the second trip in the mobile data set
seleted_trip <- mobile_data[[40]]

# Find the best trip from OBDII data set
match_info <- FindBestTrip(seleted_trip, obd2_data)

# Visualization
VisTrip(seleted_trip, obd2_data, match_info)
```