Dictionaries of Cuebiq data and notes of data processing

By Feilong Wang, 04/27/2019

Dictionaries

Dictionary of *raw* data:

Below is an email from Cuebiq describing a set of raw data:

> Data Folders:
>
> We've stored the data into a series of gzipped files, partitioned into folders by date. The folder names will represent the day on which the data was processed, so you would see folders like:
>
> 2018010100 - meaning the data was processed on January 1, 2018
>
> 2018010200 - meaning the data was processed on January 2, 2018
>
> 2018010300 - meaning the data was processed on January 3, 2018
>
> We process our data at 12AM UTC and there can also be a certain lag in the data that we receive as it's batched on device before being sent to our servers, therefore the timestamps in each dated folder may not represent only that day and could potentially include some data from additional dates.
>
> The rule of thumb is that if you will have 90+% of data for a given day looking at the data for that day and the following day. So for example, if you wanted to have the data for January 1 you'd look at:
>
> 2018010100 - meaning the data was processed on January 1, 2018
>
> 2018010200 - meaning the data was processed on January 2, 2018
>
> Data Format:
>
> The actual fields in the data are structured as below (fields are separated by TAB):
>
> Timestamp (In Unix), CuebiqID (device ID hashed with our proprietary algorithm), device_type, lat, long, accuracy, timezone offset from UTC
>
> So for example: 1514752983 e6ff28c99fb2e7d40806a2ac6db82241ad8e6606d5e2ce285b8876151e1b730e 1 47.1901741 -122.5091901 12 -28800

More understandings

Refers to the data spec from Cuebiq for more detail on each field *"US InsightQ Mobility Academic spec.pdf* or *Analytics M.X spec.pdf* (included in the folder).

Refer to a report "promises of transportation big data, 2019" that is a deliverable of the FHWA project in 2018. Refer to the paper led by Feilong Wang "extracting trip ends from multi-sourced data, 2019"

Dictionary of *sorted* data:

sorted data are results of preprocessing of raw data, including combining observations of each ID, sorting them in the time order, removing duplicates.

A field is attached giving the human time of each record. The field *time_zone_offset* from the raw data is removed to save space.

Sorted data are cut into files, each of which contains 5000 or 10000 IDs. IDs are shuffled before split into small files. That is to say, each file contains a random sample of IDs.

Dictionary of *processed* data:

The folder contains data after identifying trip ends. Pls refer to the paper led by Feilong Wang 2019 for how to identify trip ends from *sorted* data. Pls refer to "*data dictionary for trip_identified.docx*" (included in the folder) for a dictionary of each field. The same as *sorted* data, data are cut into small files, each of which contains 5000 or 10000 IDs.

Notes

Notes for sorting raw data:

- some unexpected records: accuracy have '\N' value;
- longtitude/latitude have no digits (e.g., 45 -125);
- duplicated records: two rows of records are completely the same; same timestamp and gps coordinates but different location accuracy;
- accuracy could be as large as 562119 meters (be careful when loading data into database as in DB 562119 is loaded as 32768 using a SmallInt datatype);
- timezone can be -36000 which is Alaska time (2 hours later than PST (loaded as -32768 in DB using SmallInt));

Addressing "out of memory":

- read 10000 IDs (2000 if you get a small memory), scan through all files to get observations of these IDs, do calculations, save results on disk, read next 10000 IDs and fetch results from disk and do your final calculation. You may find that sorted and processed data have already been cut into files, each of which contains 10000 or 5000 IDs.
- Use a database. Using database may be not good depending on your applications. It is good in the phase of sorting data, where we need to combine observations belonging to one ID but contained in a lot of small files. Time saving using a database is limited as it takes a lot of time just loading all files into a table of DB and indexing the table. After loading the data, it is relatively fast to fetch observations of one ID. DB has its advantages when need some frequent data manipulations, for example, when we try to fetch one specified ID. For the use of sorted and processed data, DB has no advantage as we are reading files in a sequential order.

Notes for processing data (e.g., identification of trip ends):

- Most of IDs can be processed quickly, but some "outliers" can really block your algorithm: your calculation can never come to the end even you wait for 2 days. You may find that one ID has

about 77178 records: *f8afa76b6aaaa5cd0c07171f96a2caf9d29911018b44a149d02df1ac910ef4f8* in the dataset part201805 (PSRC 201801 to 201803).

- The use of multi-processing to address "outliers": when doing multi-processing, sort your IDs in the memory in the reverse order according to the number of observations so that we avoid address "outliers" at the end of run time and have to wait. Use the function *pool.apply_async* instead of *pool.map* in Python.
- Some functions are optimized to save calculation time, and the tradeoff is that the functions are not easy to read. For example, originally, it takes a few lines in the function finding out whether the diameter of a cluster (defined as the longest distance between any two locations in the cluster) exceeds 200 meters. To save time, the function is rewritten and uses a lot of heuristic rules.
- Python environment is critical in this step. Please download and use anaconda2_4.0.0_win64. anaconda2_2019.03_win64 will give different results: seems that gps points close in space are not clustered and the reason is not clear. Try "conda env create –f environment04302019.yaml" to build your new environment.

Code sharing

Codes for sorting and processing data can be found at github https://github.com/feilongwang92/mining-app-data.git.