ECML/PKDD 15: Taxi Trajectory Prediction (I)

Philippe Remy
13th October 2015

Introduction

The taxi industry is evolving rapidly. New competitors and technologies are changing the way traditional taxi services do business. While this evolution has created new efficiencies, it has also created new problems.

One major shift is the widespread adoption of electronic dispatch systems that have replaced the VHF-radio dispatch systems of times past. These mobile data terminals are installed in each vehicle and typically provide information on GPS localization and taximeter state. Electronic dispatch systems make it easy to see where a taxi has been, but not necessarily where it is going. In most cases, taxi drivers operating with an electronic dispatch system do not indicate the final destination of their current ride.

Another recent change is the switch from broadcast-based (one to many) radio messages for service dispatching to unicast-based (one to one) messages. With unicast-messages, the dispatcher needs to correctly identify which taxi they should dispatch to a pick up location. Since taxis using electronic dispatch systems do not usually enter their drop off location, it is extremely difficult for dispatchers to know which taxi to contact.

To improve the efficiency of electronic taxi dispatching systems it is important to be able to predict the final destination of a taxi while it is in service. Particularly during periods of high demand, there is often a taxi whose current ride will end near or exactly at a requested pick up location from a new rider. If a dispatcher knew approximately where their taxi drivers would be ending their current rides, they would be able to identify which taxi to assign to each pickup request.

The spatial trajectory of an occupied taxi could provide some hints as to where it is going. Similarly, given the taxi id, it might be possible to predict its final destination based on the regularity of pre-hired services. In a significant number of taxi rides (approximately 25%), the taxi has been called through the taxi call-center, and the passenger's telephone id can be used to narrow the destination prediction based on historical ride data connected to their telephone id.

In this challenge, it is asked to build a predictive framework that is able to infer the final destination of taxi rides in Porto, Portugal based on their (initial) partial trajectories. The output of such a framework must be the final trip's destination (WGS84 coordinates).

Three models are considered: Random Forest, Extra Trees and RANSAC. Also, we compare them with several benchmarks. Finally, results are proposed and conclusions drawn.

Empirical Analysis

Introduction

The list of the data files along with their descriptions is available online[1]. In this short analysis, we consider two of them: `train.csv (1.9GB)` and `test.csv (0.3MB)`. The training and testing data are transformed to reveal some hidden properties and to fit the input requirements of the models. The transformations are presented in this section.

Presentation of the Data

The training set describes a complete year (from 01/07/2013 to 30/06/2014) of the trajectories for all the 442 taxis running in the city of Porto, in Portugal. It is composed of 9 columns: `TRIP_ID`, `CALL_TYPE`, `ORIGIN_CALL`, `ORIGIN_STAND`, `TAXI_ID`, `TIMESTAMP`, `DAYTYPE`, `MISSING_DATA` and `POLYLINE`. The operations on the raw data used are listed below.

- `TRIP_ID`: It contains an unique identifier for each trip. **It is discarded**.
- `CALL_TYPE`: It identifies the way used to demand this service. It may contain one of three possible values (dispatched from the central, trip was demanded directly to a taxi driver on a specific stand, trip demanded on a random street). **It is factorized with** `pandas.factorize()`.
- `ORIGIN_CALL`: It contains an unique identifier for each phone number which was used to demand, at least, one service (if trip dispatched from the central). **It is factorized since no ordering makes sense with identifiers**.
- `ORIGIN_STAND`: It contains an unique identifier for the taxi stand. It identifies the starting point of the trip. **It is factorized**.
- `TAXI_ID`: It contains an unique identifier for the taxi driver that performed each trip. **It is factorized**.
- `TIMESTAMP`: Unix Timestamp (in seconds). It identifies the trip's start. **A UNIX Timestamp can be hard to deal with. For example, a pattern occurring at the same hour every day would be missed if the Timestamp is given "as it is" to the learning algorithm. Indeed, the algorithm would probably not know that it should use the modulo function to reveal interesting patterns. Therefore, the Unix Timestamp is converted to the hour of the day where the trip started.**
- `DAYTYPE`: It identifies the daytype of the trip's start. It assumes one of three possible values (trip started on a holiday, trip started on a day before a holiday, normal day). **It is factorized.**
- `MISSING_DATA`: It is FALSE when the GPS data stream is complete and TRUE whenever one (or more) locations are missing. **This field is useful to drop**

[1] `https://www.kaggle.com/c/pkdd-15-predict-taxi-service-trajectory-i/data`

the training data rows with missing values. It is possible because the proportion of missing data rows is very small.

- `POLYLINE`: It contains a list of GPS coordinates (i.e. WGS84 format) mapped as a string. The beginning and the end of the string are identified with brackets (i.e. [and], respectively). Each pair of coordinates is also identified by the same brackets as [`LONGITUDE`, `LATITUDE`]. This list contains one pair of coordinates for each 15 seconds of trip. The last list item corresponds to the trip's destination while the first one represents its start. **This is a valuable field where a lot of inference can be made. From this field, we consider the first and last known locations for sake of simplicity. Also, we compute the instant speeds, the average speed and the duration. The distance between two GPS measures is measured with the Haversine distance.**

Recall that the Haversine distance is given by:

$$
\begin{aligned}
a &= \sin^2(\Delta\phi) + \cos(\phi_1)\cos(\phi_2)\sin^2(\Delta\lambda/2) \\
c &= 2\tan^{-1}(\sqrt{a}, \sqrt{1-a}) \\
d &= Rc
\end{aligned}
$$

where ϕ is latitude, λ is longitude, R is earth's radius whose mean radius is 6371km. The data columns presented to the learning algorithms are: `CALL_TYPE`, `ORIGIN_CALL`, `ORIGIN_STAND`, `TAXI_ID`, `DAYTYPE` along with the new freshly created columns:

- `FIRST_LAT (LAT1)`: First known latitude taken from `POLYLINE`.
- `LAST_LAT (LAT2)` Last known latitude taken from `POLYLINE`.
- `FIRST_LONG (LONG1)`: First known longitude taken from `POLYLINE`.
- `LAST_LONG (LONG2)`: Last known longitude taken from `POLYLINE`.
- `HOUR`: Hour of the ride (range: 0-23). From `TIMESTAMP`.
- `DURATION`: Duration of the ride in seconds (range: 0-7200). From `POLYLINE`.
- `AVG_SPEED`: Average speed of the ride in Km/h (range: 0-130). From `POLYLINE`.
- `LAST_SPEED`: Last known speed in Km/h (range: 0-130). From `POLYLINE`.
- `BEFORE_LAST_SPEED`: Before last known speed in Km/h (range: 0-130). From `POLYLINE`.

The training data rows are filtered to discard any incoherences in the set. One row is dropped if:

- `MISSING_DATA` was set to True.
- Any of the speeds is above 130 Km/h. It seems reasonable since the speed limit is 120 on Portugal highways and the rides are located in Lisbon.
- A duration of 0 seconds is considered invalid. It must be strictly positive.

Once done, `TRIP_ID`, `TIMESTAMP`, `MISSING_DATA`, `POLYLINE` are dropped since they are no longer meaningful. Then, the `N/A` values are filled with the arbitrary value (-1). This value was selected instead of 0 because it does not interfere with the factors generated by `pandas.factorize()`. Finally, the data frames are converted to arrays of floating values.

Learning model

Given the data presented in the previous section, we have to predict the dropoff point, i.e. the final latitude and longitude. Three models are considered here: RANSAC[2], Extras Trees[3] and Random Forest[4]. RANSAC belongs to the family of Generalized Linear Models while the Extras Trees and Random Forest belong to the ensemble family, where the goal is to combine predictions of several base estimators to improve robustness over a single estimator. Also for assessment, we consider three benchmarks: Dummy regression[5], all the final latitudes and longitudes pointing to the city center, and where the final known latitude and longitude are used for the dropoff. Finally, some model parameters are tuned to improve the performance.

Results

The optimal number of trees (`n_estimators`) is around 50. It corresponds to the threshold where the out-of-bag score becomes steady (0.974). All features are taken for every tree. Hence, `max_features` is set to `auto`. It seems that the parameter `min_samples_leaf` has no big impact on the inference for this dataset. Several values `[1,5,10,50,100,200,500]` were tested and it showed no improvements on the MSE. Table 0.1 recaps the results. The Kaggle score corresponds to the Mean Squared Error (MSE with the Haversine distance) on the unknown validation set available on submission. The leader board is a theoretical criterion to have an idea of how good the solution is. It turns out that the Random Forest performs slightly better than the Extra Trees. Also, both RFs and ETs clearly outperform the Generalized Linear Model approach.

In practice, some users reported that RFs are often more compact than ETs. ETs are generally cheaper to train from a computational point of view but can grow much bigger. ETs can sometime generalize better than RFs but it is hard to guess when it is the case without trying both first.

[2] `http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RANSACRegressor.html#sklearn.linear_model.RANSACRegressor`

[3] `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html#sklearn.ensemble.ExtraTreesRegressor`

[4] `http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor`

[5] `http://scikit-learn.org/stable/modules/generated/sklearn.dummy.DummyRegressor.html#sklearn.dummy.DummyRegressor`

Algorithm	Kaggle Score (MSE)	Leader board Rank
Random Forest Regressor	2.56814	84/381 (Top 25%)
Extra Trees Regressor	2.57451	90/381 (Top 25%)
RANSAC GLM Regressor	2.60440	127/381
Dummy Regressor	3.90373	306/381
Benchmark city center	4.18350	315/381
Benchmark last known location	2.60663	202/381

Table 0.1: Results of the learning process on the online Kaggle validation set

As a conclusion, the three learning algorithms clearly outperform the three benchmarks. It means that using machine learning algorithms seems beneficial in the prediction of the final destination of the taxis.

Some further work could be done on tuning the parameters in a more exhaustive way. Also, we did not use the file `metaData_taxistandsID_name_GPSlocation.csv` which lists the ids and locations of the taxi stands. We think that it could make sense as a taxi is more likely to end up there. Also, some more measures could be derived from the data, i.e acceleration, profile of the drivers (if a driver is more likely to take only long rides). Finally, online learning (e.g. with neural networks) could be done with the `POLYLINE` variable. The algorithm could learn on how to predict and correct with the next points in a sequential way, as `POLYLINE` is a list of spatial points.

Questions and Answers

Why are you choosing the methods that you choose?
Random Forests are easy to fit and usually give good results. That was the case here. Also, it was interesting to compare how it performs with a Generalized Linear Model.

What was the hardest part?
The hardest part was to deal with a very large dataset of almost 2GB. Hopefully, I could use my computer desktop remotely (Core i7, 8GB of memory, SSD).

What was the easiest part?
Although it was my first time using Python with machine learning, I must confess that it was the easiest part.

What did you enjoy?
Pretty much everything. The subject was very interesting. and Kaggle is very well designed. It was my first Kaggle challenge.

Can you think of some interesting business ideas that could be built around a solution for this problem?

By predicting the final destination of a taxi, we improve the efficiency of the dispatching system. It is particularly true in periods of high demands, where it is important to know approximately the final location of each taxi in order to optimize the service. It is worth noting that the calibration phase takes a long time (>20min). Once the task is performed, the values are predicted very fast. In other words, the solution could be integrated in a business environment.

The source code is available at `https://github.com/philipperemy/Kaggle-PKDD-Taxi-I`.