

ANDREI'S WEB MINING BLOG

Useless results of curiosity

[HOME](#)[ABOUT ME](#)[RSS FEED](#)

7TH PLACE IN KAGGLE'S DRIVER TELEMATICS CHALLENGE

Posted on [March 17, 2015](#) by [Andrei Olariu](#)

Intro

The purpose of the [AXA Driver Telematics Challenge](#) was to discover outliers in a dataset of trips. We were given a total of 2730 drivers, each with 200 trips. We were told that a few of those 200 trips per driver weren't actually his and the task was to identify which ones. Somewhat similar to authorship attribution on texts.

A drive was composed of a series of GPS measurements taken each second. Each drive started at (0, 0) and all the other points were given relative to the origin, in meters. In order to make it harder to match trips to road networks, the trips were randomly rotated and parts from the start and the end were removed.

If the dataset would have been composed of only one driver, then this would actually be outlier detection. But more drivers means more information available. Using a classification approach makes it possible to incorporate that extra info.

System Overview

Local Testing

For each driver, take 180 trips and label them as 1s. Take 180 trips from other drivers and label them as 0s. Train and test on the remaining 20 trips from this driver and on another 20 trips from other drivers. That's it.

Can we improve on this? Yes. Take more than just 180 trips from other drivers. Best results I've got were with values between 4×180 and 10×180 . In order to avoid an unbalanced training set, I also duplicated the data from the current driver.

Considering there were trips from other drivers that I was labeling as 1s when testing locally, it wasn't possible to get the same score locally as on the leaderboard. Yet the difference between the two scores was very predictable, at around 0.045, with variations of around 0.001. The only times I was unsure of my submissions were when I had a data sampling bug and when I tried a clustering approach using the results from local testing.

Leaderboard Submissions

Pretty much the same logic as above. Train on 190 trips from this driver, along with 190 trips from other drivers, test on the remaining 10 trips from this driver. Repeat 20 times, in order to cover all trips (similar to crossvalidation). If the process was too slow, then repeat 10 times with 180+20 splits. Also apply the data enlargement trick used in local testing.

Ensembling

In production systems, you usually try to balance the system's performance with the computational resources it needs. In Kaggle competitions, you don't. Ensembling is essential in getting top results. I used a linear model to combine together several predictions. The **Lasso model in sklearn** gave the best results, mainly because it is able to compute nonnegative weights. Having a linear blend with negative weights is just wrong.

Caching

When you have an ensemble (but not only then), you will need the same results again and again for different experiments. Having a system for caching results will make your life easier. I cached results for each model, both for local validation, as well as leaderboard submissions. With some models

needing a couple of days to run (with 4 cores at 100%), caching proved useful.

Feature Extraction

Trip Features

These were the best approaches overall, with a maximum local score of 0.877 using Gradient Boosting Trees (that would be an estimated LB score of about 0.922). Some of the features used were histograms and percentiles over speeds, accelerations, angles, speed * angles, accelerations over accelerations, speeds and accelerations over larger windows.

Road segment features

I've seen a lot of talk on the forum of matching trips using Euclidean distance. I tried to go a little further and detect similar road segments. This approach should detect repeated trips, but it should also detect if different trips have certain segments in common. I applied a trajectory simplification algorithm (**Ramer-Douglas-Peucker** available for Python [here](#)), then I binned the resulting segments and applied SVMs or Logistic Regression, like on a text corpus. Local results went up to 0.812 (estimated LB score of around 0.857).

One thing I didn't like about the RDP algorithm was how similar curves could be segmented differently due to how the threshold used by the algorithm was affected by GPS noise. So I built another model. I thought the best way to encode a trip was as a list of instructions, similar to Google Maps directions. I looked at changes in heading, encoding them as left or right turns. This model scored up to 0.803 locally, lower than the RDP model, but in the final ensemble it got a bigger weight.

Movement features

I think these types of features best capture somebody's driving style, although to some degree they also capture particular junction shapes and other road features. The main idea is to compute a few measurements each second, bin them, then treat them as text data and apply SVMs or Logistic Regression. For example, the best scoring model in this category (local score of just under 0.87) binned together the distances and the angles computed over 3-second intervals on smoothed versions of the trips (smoothing done using the **Savitzky-Golay** filter in [scipy](#)). After binning and converting to text data, I

applied Logistic Regression on n-grams, with n between 1 and 5.

I tested over 100 individual models to select a final ensemble of 23 models. Local score was 0.9195 and the final LB score was 0.9645 (public LB), **0.9653** (private LB).

My code is available on [GitHub](#).

This entry was posted in [Kaggle](#). Bookmark the [permalink](#).

[← PhD Achievement Unlocked](#)

5 THOUGHTS ON “7TH PLACE IN KAGGLE’S DRIVER TELEMATICS CHALLENGE”



Alejandro Simkievich says:

March 17, 2015 at 6:09 pm

Thanks a lot for sharing Olariu. I myself got 0.81 (entered a bit late, in the last 30 days) but will definitely check your code and play around a bit.

Reply



Niko Steinhoff says:

March 17, 2015 at 11:18 pm

Hi Andrei,
thanks a lot for sharing your approach and your code. I learned a lot,