DATA SCIENCE CHALLENGE

The Floow

Lefteris Vazaios
leftvaz@gmail.com

Contents

Introduction

The aim of this assignment is to analyse data from 25 car journeys and determine which of them contain events that might be accidents. The data contain GPS and accelerometer measurements.

The code for this analysis was written in Python. The accompanying text file contains instructions on how to use it. This document is a report of the methodology and findings.

I have intentionally used the most common python packages for Data Science to ensure the program will be easily run, with minimal setup. I have also limited the number of graphs for performance issues.

The data

As mentioned, there are 25 journeys documented in .csv format. Each of them contains a UNIX epoch timestamp as well as measurements from GPS and accelerometer as follows:

GPS:

- Latitude
- Longitude
- Height
- Accuracy
- Speed
- Bearing

Accelerometer:

- Acceleration in each of the 3 axes (x, y, z)

Importing the data

The first part of the script reads all .csv files from the specified directory (variable 'path'). Depending on the operating system (OS), the path might work as pasted or not. For Windows, I just replaced the separator with '\\' and then let the 'os.sep' function modify the path accordingly. For other OS, using default separators should be fine.

During the import, each file was given a unique 'Journey_ID' and all separate journeys were combined into a single data frame. Moreover, any leading or trailing blanks in variable names were removed.

Pre-processing

The next step was to transform the data according to the problem's specifications.

Firstly, the UNIX epoch timestamp was converted to a regular timestamp using the 'strftime' function from the 'datetime' module. Given that the timestamps were accurate to the microsecond, a suitable format was chosen to take that into account.

Next up, it was mentioned (and observed) that some of the accelerometer values were of considerable higher scale than others. Identifying and replacing those could be done in a number of ways, some more complicated than others. In this case, a very simple solution was implemented: given that a very fast car (e.g. a Jaguar XK) can achieve accelerations of around 6/ms2, we assume that journeys that contain measurements over the absolute value of 5 (cars rarely achieve maximum acceleration on the road) are in G. Those journeys had their x, y, z values divided by 9.81.

Note that this is a simplistic and not necessarily optimal implementation. However, it is only needed in some of the algorithms trained. Methods like standardisation are not recommended since we are actually looking for anomalies (not trying to suppress them).

Due to the nature of the data (measurements from two different sources), there were a lot of missing values. More specifically, GPS measurements had no values for x, y, z while accelerometer measurements had no values for the remaining features. As is common in time series, the forward filling method was used for filling in the blanks (replace each missing value with the previous known value). Because one row still had missing values (due to its position), back filling was also run to completely remove nulls.

Finally, since the phone can move around inside the car, it is not possible to know which of the x, y, z values represents the actual movement. To counteract that, I created a new variable, 'acceleration', which is the maximum absolute value of the x, y, z at each possible moment while retaining the sign of that value.

Intuition

This is a problem of unsupervised anomaly detection, i.e. locating points in the data that have unexpected values, without given labels of actual accidents. In accidents, there is usually an instance of rapid deceleration, which will hopefully be captured by the accelerometer. Since we do not know the orientation of the phone, it is not possible to associate the accident with a sign (+ or -) but it is possible to find high-absolute-value accelerations of different signs happening in rapid succession (resembling the shock of the impact). In cases of accidents, these instances are likely to be followed by periods of immobility. By plotting the acceleration values for some of our journeys, anomalies like the ones described can be seen:

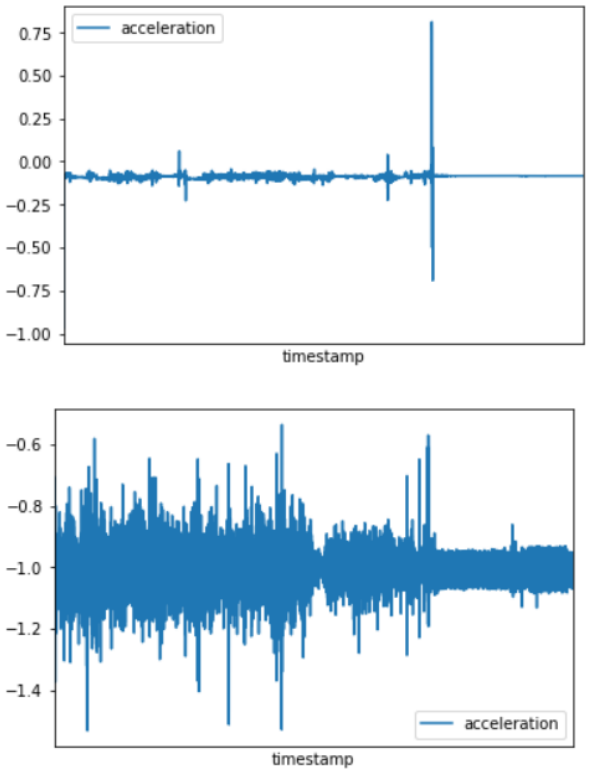

Figure 1: A case where an accident is probably present and another where an accident is unlikely to have happened

In both those graphs, a single journey is plotted. This approach, i.e. treating each journey differently, is expected to be more accurate since it captures the uniqueness of each journey (and therefore the driver, the car, the road and weather conditions at the time, etc.).

Conversely, treating all journeys as part of the same series assumes all these factors are the same. By doing so, however, it is expected to produce a model that generalises better (i.e. less prone to overfitting the current journey/driver). This is the case where different accelerometer scales (G vs. m/s2) matter.

In my proposed solution, I have implemented algorithms for both approaches and then combined their results.

Algorithm implementation

In total, 4 different types of algorithms were used:

- A simple moving average
- Local Outlier Factor (LOF)
- One-Class SVM
- Isolation Forest

All 4 of those were used for each separate journey. The two of them (moving average and LOF) were applied to the entire data frame (i.e. all journeys treated as one).

Moving Average

This is a simple but effective algorithm in which, for each observation, a moving window exists capturing at every timestamp the 45 measurements around it. The size of the window was chosen in a way that could include 4 measurements from GPS (should be 44, rounded to 45).

For this window, the average acceleration (absolute value, to avoid having a mean of 0) is calculated for any particular moment. If an observation has an absolute acceleration that is more than 6 standard deviations away, it is considered an outlier.

Note that 6 standard deviations are a rather strict threshold that only detects extreme outliers. Due to the nature of the problem (i.e. the rarity of accidents) this is acceptable and avoids falsely classifying minor accelerations/decelerations as accidents.

A new column (anomaly_ma) is created on the scored dataset, taking the value of 1 if an anomaly is present according to the algorithm and 0 otherwise.

Local Outlier Factor

LOF is an algorithm that for each observation checks its nearest neighbours (in this case 50) and identifies outliers my measuring the local deviation (in respect to those neighbours).

As a parameter, it is given the expected number of anomalous observations, which I have set to 1 in 10,000 (0.0001). It will then try to identify enough observations that best fit the criteria. Due to its nature, it will always produce at least one anomaly in each time series (the most outlying one, regardless of whether it is actually anomalous given the entire journey). As such, if it is used by itself it will classify all journeys as having an anomaly (sometimes in more than one timestamps although the parameters have been tuned to limit this).

The variables used were acceleration, speed and bearing. The reasoning behind this is that longitude, latitude, accuracy and height do not change dramatically enough to be associated with an anomaly.

The x, y, z features are replaced by acceleration. Naturally, a different set of variables could have been selected. Due to the problem being unsupervised, traditional variable selection techniques (such as stepwise models) cannot be applied.

Once again, a new column (anomaly_lof) is added to the data frame.

One-Class SVM

One-Class SVM is an algorithm used to identify two groups of roughly equal size that are usually clearly separable. However, it can be tuned for anomaly detection by tweaking the parameters. Once again, the same 3 variables were used and the algorithm was taught to expect a 0.0001 outlier percentage.

Due to the nature of the SVM algorithm and its tendency to find groups of roughly equal size, it turned out to be the most 'tolerant' model in terms of classifying anomalies, in that it produced the largest number of outliers per journey.

Those outlier observations were marked as '1' in the anomaly_svm column that was added to the dataset.

Isolation Forest

The last algorithm applied to each journey separately was isolation forest which is based on random forests. Once again, it is passed a parameter of the expected outliers fraction (0.0001).

The difference here is the number of variables used. Because random forest combines different features in each iteration, I used all numeric variables in the dataset. This means that both the acceleration and one or more of the x, y, z values could be present in one of the trees implicitly produced. There is also the option of generating more features (e.g. polynomials). At any rate, the algorithm performed fine and produced the new column anomaly_if.

Moving Average and Local Outlier Factor on the entire dataset

As mentioned, two of those algorithms were also applied to the entire dataset. This involved tweaking the parameters somewhat, to account for the fact that journeys have different characteristics. This is also the case where replacing extreme acceleration values has had some impact on the results.

The two algorithms produced the columns anomaly_ma2 and anomaly_lof2 respectively.

Note that for all the algorithms there are parts in the code that have been commented out and print the following results:

- *For each journey, the number of normal observations and the number of anomalies*
- *For the entire algorithm, which journeys have at least 1 anomaly*

The following 2 lines of code do exactly that for the moving average algorithm. Replacing the column name gives the results for a different algorithm.

```
print(journeys_scored['anomaly_ma'].value_counts())

journeys_scored.loc[journeys_scored['anomaly_ma'] == 1, ['Journey_ID']].Journey_ID.unique()
```

Combining algorithms

Since different algorithms have their own strengths and weaknesses, it is a reasonable idea to combine their results. In essence, if an observation is classified as an anomaly by at least half (3) of the models, then it most likely represents an accident. Essentially, we want an implementation that is accurate enough (hence the number of models required) but tolerant enough to identify dangerous/anomalous incidents, even if they were not actual crashes. In short, recall matters more than precision in such a problem.

The combination is done by creating a condition that simply sums up the anomaly columns created. If the sum is greater than 2, the observation is classified as an anomaly. According to this methodology, the journeys containing an accident are:

2, 4, 5, 10, 12, 15, 21, 22

Note that the first journey is the 0^{th}. So, in terms of file order, these are the third, fifth, sixth, etc. files.

The following table presents the journeys with anomalies, the timestamp and the models that identified it.

	Journey_ID	timestamp	anomaly_ma	anomaly_lof	anomaly_svm	anomaly_if	anomaly_ma2	anomaly_lof2
5580	2	2015-04-20 04:30:45.258670	1	1	1	0	1	0
1042	4	2015-04-03 04:52:18.935260	1	0	1	0	1	0
11716	5	2015-03-16 18:11:14.301690	1	0	0	1	1	0
11255	10	2015-06-20 23:35:20.896120	1	0	1	0	1	0
10229	12	2015-05-17 19:04:23.392600	1	0	1	0	1	0
35359	15	2015-03-19 23:37:29.228870	1	1	1	0	1	0
6495	21	2015-04-24 22:28:41.191820	1	0	1	0	1	0
1921	22	2015-03-16 23:56:35.278370	1	0	0	1	1	0

Table 1: Journeys with accidents

Severity and confidence for each event

The severity of the event can be determined by the force of impact which is directly related to the acceleration at the time of the anomaly. A simplistic approach is to look at the absolute value of the acceleration (potentially focusing on negative values). In this case I tried to define a severity index that gives more value to observations with a high speed and a low acceleration (i.e. high absolute value with a negative sign). The formula is purely intuitive and could definitely be improved:

acceleration^2 / (1 + acceleration + speed)

The following table presents the anomalies with their allocated severity index.

	Journey_ID	timestamp	speed	acceleration	sev_index
5580	2	2015-04-20 04:30:45.258670	0.000000	3.904526	3.108419
1042	4	2015-04-03 04:52:18.935260	0.320000	0.340411	0.069790
10522	5	2015-03-16 18:09:29.263690	5.720000	-5.653305	29.961576
11716	5	2015-03-16 18:11:14.301690	1.770000	-6.411346	11.288506
11255	10	2015-06-20 23:35:20.896120	0.000000	-0.467373	0.410114
10229	12	2015-05-17 19:04:23.392600	2.440000	2.318878	0.933723
35359	15	2015-03-19 23:37:29.228870	0.000000	0.211794	0.037017
6495	21	2015-04-24 22:28:41.191820	11.940000	0.812399	0.047991
1921	22	2015-03-16 23:56:35.278370	16.209999	-4.153137	1.321033

Table 2: Severity index (notional definition)

As for the confidence of the being an accident there are several factors to consider. Potential issues include any sudden acceleration or deceleration (e.g. braking that prevented a crash) as well as irregular phone movements inside the car (e.g. the phone falling). One approach is to once again look at graphs:

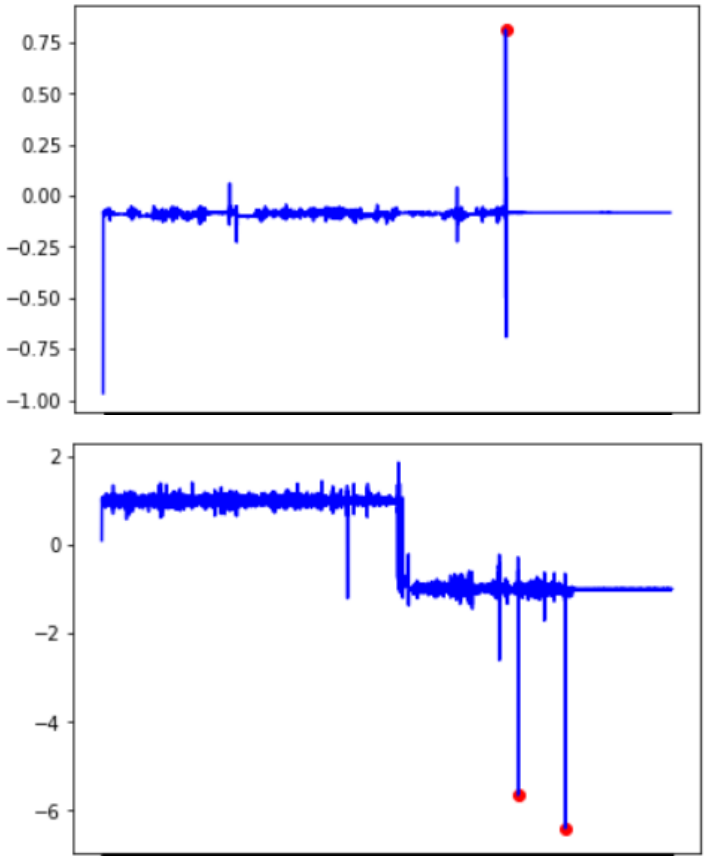

Figure 2: Acceleration values for journeys 21 and 5, which were scored as anomalous by 3 and 4 models respectively. Anomalous points are marked red

It is quite clear that the points represent outliers but it is difficult to determine the reason for such a rapid change in acceleration. Intuitively, it is expected that after an accident, a long period of immobility follows. This seems to be present in the above graphs. Consequently, I calculated for each journey that had an anomaly the average speed and acceleration for the 100 timestamps immediately after the anomaly.

Then I defined two indexes as the difference between the speed at and after the anomaly and the acceleration at and after the anomaly. Once again, the approach is very simplistic and could very likely be improved upon. The table with the results is presented below.

	Journey_ID	timestamp	speed	acceleration	sev_index	acc_after	speed_after	conf_index1	conf_index2
5580	2	2015-04-20 04:30:45.258670	0.000000	3.904526	3.108419	-0.807143	6.4365	4.711669	6.436500
1042	4	2015-04-03 04:52:18.935260	0.320000	0.340411	0.069790	0.095340	5.9974	0.245071	5.677400
10522	5	2015-03-16 18:09:29.263690	5.720000	-5.653305	29.961576	0.990922	21.5390	6.644227	15.819000
11716	5	2015-03-16 18:11:14.301690	1.770000	-6.411346	11.288506	0.990922	21.5390	7.402268	19.769000
11255	10	2015-06-20 23:35:20.896120	0.000000	-0.467373	0.410114	-0.056242	7.9900	0.411131	7.990000
10229	12	2015-05-17 19:04:23.392600	2.440000	2.318878	0.933723	0.794992	12.0419	1.523886	9.601900
35359	15	2015-03-19 23:37:29.228870	0.000000	0.211794	0.037017	0.080598	2.6880	0.131196	2.688000
6495	21	2015-04-24 22:28:41.191820	11.940000	0.812399	0.047991	-0.088600	8.9630	0.901000	2.977000
1921	22	2015-03-16 23:56:35.278370	16.209999	-4.153137	1.321033	-0.807832	7.2399	3.345305	8.970099

Table 3: Confidence Indexes (notional definitions)

Another rather simple option to consider would be how many models scored the observation as an anomaly. In our case, only the journeys 2 and 15 had their anomalies confirmed by 4 models.

Final notes and conclusion

Since this problem is unsupervised, parameters have been tweaked to best fit the training data. In a supervised problem, a proper split into training, validation and tests sets should occur. The parameters would be tweaked to fit the validation set and the accuracy (or precision/recall or F-Score) would be measured on the test set. Ideally, this implementation should be tested on labelled data to determine if it is a good one.

In terms of performance, the program was designed to limit disk input-output. All files were loaded in the main memory and then handled either separately or as a single entity from there. This assumes the available hardware has enough memory to accommodate all the datasets and leave room for computations. If this is not the case (e.g. if the data is too large), a different implementation might be necessary; processing each file as it is being imported is a solution (i.e. training the algorithm during

the file import writing results to disk before moving on to the next file). Graphs have been used extensively for data exploration but they are not used extensively in the code for performance reasons (an obvious alternative would be to plot one in every 5 or 10 observations).

In terms of algorithms, another unsupervised solution might be k-means clustering. However, it is unlikely that it would be able to create clusters of 1 or 2 observations and is generally not suited for outlier detection. When it comes to supervised learning, several algorithms could perform well (most notably neural networks).

In the current solution, possible alterations could include:

- A different approach to distinguishing G or m/s2 for accelerometer values (although as demonstrated, this only mattered in the cases where the entire dataset was handled as one journey).
- Perhaps treating accelerometer and GPS series differently instead of filling blanks.
- Adding new features to the dataset. This might require new measurements, a better knowledge of the industry or more extensive data manipulation (e.g. polynomial terms).
- Trying different algorithms (e.g. moving median instead of moving average) or different parameters for the existing ones.
- Improving severity and confidence indexes.

In conclusion, this is an accurate and easy to modify solution that successfully identifies anomalies in different journeys. While industry knowledge might be needed to define better measurements and visually identify actual accidents, the combination of algorithms provides a model capable of learning the uniqueness of each journey/driver, that can also generalise by learning from different journeys simultaneously.