Summary

September 29, 2020

1 Summary of work done

Note that this notebook summarises the work done *after* all the work done for the Movement Clustering project. If you are unfamiliar with it, it might be worth reading Summary.ipnyb under 'Clustering Movement Data'.

The notation used here will be as follows:

Acceleration: a_i, where i indicates the degrees of freedom. The Einstein convention is used here, so, the dot product (total acceleration) for example: $a_i a_i = a_1 a_1 + a_2 a_2 + a_3 a_3$

Gyration: ω_i

Time (continuous): t in units of seconds

Time (discontinuous): $\hat{t}$ in units of seconds

The difference between the two different times is that one is representative of *real time*, so when plotted, where there are gaps in the data, there will be gaps in the plot. The second time makes more sense in terms of the *index* of the dataframe that it represents. So that if you have two times, $\hat{t}_1 = 1$ and $\hat{t}_2 = 2$, then $\neg\Box(\hat{t}_2 - \hat{t}_1 = 1)$, since the actual time difference between them could be any arbitrary value.

Below is an example:

The first plot is gyration vs. real time (from data generated by Rohan). Note that there are many gaps in the data, this is because either some of that data has been filtered, or because the measurements were made at different times.

The second plot is gyratino vs. discontinuous time (from the same data set). Notice how all the gaps have been removed.

Notice that the data sparsity and the small volume make it difficult for forecasting methods that *understand* the time DoF to be used, since predictions from it would be, in effect, meaningless. Another question that is raised is that due to the fact that there isn't that much data, machine learning models used might not be robust, since they would only understand a limited dataset (this wasn't a problem with clustering, due to the fair assumption that 'insignificant' movements are likely to be uniform for all users). That said, a couple of possible machine learning models will be discussed.

It's important to be mindful of the **ultimate** goal of this project. At the end, the deployed model should ideally be able to do the following: 1. Detect when the movements are lower "on average" (this is analogous to 'overflow' from the fluid detection module, i.e. a tap that is left open and thus the flow is higher, on average) 2. Adapt to the fact that the average movements will be decreasing as a function of time (because the person is getting older, and their movements less vigorous) 3. Detect that there is an overall decrease in daily activity (as measured by how 'strong' the activity is, how frequent naps are – this links to 2.) 4. Detect that there is a high lack of motion (i.e. if the person removed their device or is critically ill), and detect the opposite, when there is a high amount of motion that is abnormal (if the device is attached to a dog) 5. Detect irregular movements (i.e. movements that don't have the same pattern that one would expect)

To be able to achieve all of these, it is likely that a combination of models may have to be used. For now, this project focuses on detecting when the movements are lower "on average" within a pre-defined time window, for example: 5 hours.

1.1 Finding time difference between different datapoints

For data exploration purposes, a class was created to visually inspect the seasonality of the data. The class works well for monthly and daily seasonality, but needs some refinement for subdaily (or custom time) data. As of 28.09.2020, this does not exist in it's own .py file

1.2 Moving average

Seeing as a suitable machine learning model was difficult to find, it was decided that a moving average would be used for the time being. This moving average would calculate the the average for a pre-specified time window.

A class was created that can calculate the moving average for time series data. A sample 'How to use this' notebook can be found in the parent directory of the repository.

For Rohan's data, here are the results:

For Ignacio's data, here are the results:

There are a couple of things to keep in mind with this class: Here are a couple of suggested improvements:

1. adding meaningful xticks (especially that of the graphs in the second row).
2. add some functionality for determining when averages exceed the bounds of historical data. There are multiple ways to do this, but good data must be collected first
    - define thresholds for erroneous data, i.e. if a new value, exceeds bounds of the maximum and minimum for the past 3 months, then alert the user. The thresholds can be defined using max/min, but could alternatively be defined using the quartiles, standard deviation, etc…
    - in addition to that described above, count the number of times the thresholds are exceeded (this would prevent issuing an alert due to a single erroneous value). Note that in this case, a time window of 5 hours may be too large.
    - forecast the moving average as a time series problem, and use ML methods to predict anomalies
    - Other non-timeseries anomaly detection ML algorithms could be tried, where now, unlike with the clustering problem, TIME can also be used as a degree of freedom for the analysis (note that if you are considering using machine learning methods, it might be

worth filling the ‘gaps’ in the data with zeros, so that they are ‘points’ to be considered in the machine learning process)

3. fix the way the class stores items in memory. Currently the class has lots of class variables, most of which are lists. Now, this was the first time I used classes in Python, but essentially one thing I learnt was that class variables don’t reset if they are lists (I tried with integers, and they reset fine). I’m not sure why, it might be that i’ve missed something. But either way, what happens now is that if you run a cell that instantiates the class, and then you add datapoints, more than once, then the datapoints from each instantiation get added on top of each other. I was not able to figure out how to make sure that the parent class instantiates all the variables once only.
4. make the class more efficient: this was my first time using classes in python, and as such there’s bound to be inefficiencies in the code. Some of it might be due to the logic too. For deployment purposes (long term, when the product is to be released), it’s worth taking a look at making the process more efficient and modularising the code / making it cleaner
5. there are some improvements to be made in temrs of readability of the code, for example, the DoF is used multiple times in the class but re-defined using different names, i.e. averages.shape[1] or len(self.data[0][0])

1.3 Other ideas

1. One idea was to calculate the power of the signal for a certain time-frame and compare it to previous powers (analogous to using the moving average method, but with signal power). The advantage that this method *may* have over the moving average is that it can capture a lack of data too. So if for the pre-specified time-frame, 90% of the data is 0 (because say the elder is napping), then it would capture that in how ‘powerful’ the signal is. It may also be able to capture some of the nuances in the signal, for example how continuous it is. It was noted that the sample for ‘falling’ data had a noticeable high power; worth investigating.|