1 Activity scoring based on the time

As any other resource, an activity is scored via the standard formula:

$$S = \sum_{i=1}^{n} e^{-d_i} e^{k_i \log(l_i)}$$

Apart from that score, there is a need for smarter recognition of user's usage patterns based on the time and location.

The standard way of processing arrays of data and recognizing event frequencies is to use FFT, but in this case it doesn't work as advertised simply because there's not enough data for FFT to work out its magic.

The alternative is to make a more specific, per use-case tailored system. Essentially, the most common usage patterns can be divided into three different types:

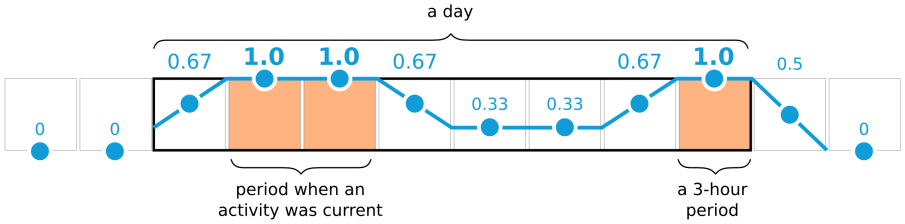
1. Specific weekdays
2. "Time-of-the-month"
3. Every n days

1.1 Specific weekdays

The time interval we will be watching in this case is a week. It is divided into 7 days, each divided into 8 three-hour segments so that we can detect a pattern that is not only specific to a specific weekday but also to a time of the day.

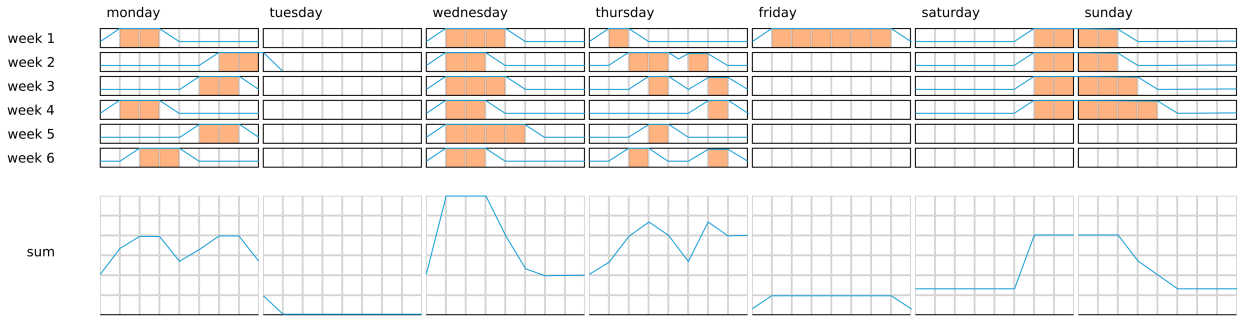
The collected data is as follows – for each week, an array of 56 elements with range of $[0, 1]$ is remembered. If an activity was used during a segment, the segment is called "active". The actual value for each segment is calculated like this:

- An active segment has a 1.0 score
- If the segment is next to an active segment, and it is in the same day as an active segment, the score is 0.67
- If the activity was used in the same day as an active segment, but is not next to an active segment, the score is 0.33
- If the segment is next to an active one, but there is no active segment in the same day, the score is 0.5



The collected data is summed to get the score for each activity for the segment the user is currently in. It should be sufficient to sum only the data from the last 4-8 weeks to get optimal results. The alternative would be to use a similar function to the one that does the regular resource scoring, but in this case there is no real need for exponential time-based deterioration.

The different possible situations are illustrated in the following graph:



1.2 "Time-of-the-month"

Here, we are trying to detect when in the month the user likes to use a specific activity. Since the different months have different number of days, we can't make a normal one element per day array to store the scores.

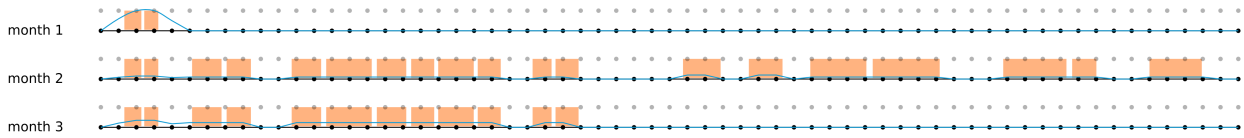
One possibility would be to create an array of 31 element and for months that have less days than that, just to leave the last index(es) unused. This wouldn't work well for the cases when the user does something every last day of the month – the system would get an approximately 0.5 score for all 30th and 31st Something in the year, while it would get a near-zero for the last day of February.

The approach used here is to treat a month as a $[0, 1]$ interval and instead of registering a day as n-th of Something, we will register it as:

$$P = \frac{n}{number\ of\ days\ of\ Something}$$

This way, the end of the month will always be 1, no matter which month is in question.

$[0, 1]$ is a continuous interval, and as such, we can not store it in the computer as such. Since the maximum frequency of data in this case is 31, according to the Nyquist-Shannon theorem for the audio signal digitalization, we can use a discrete array with the size of 62 elements to store this array without any loss of information. 62 is not a pretty number, so we will use the closest power of 2 which is 64.



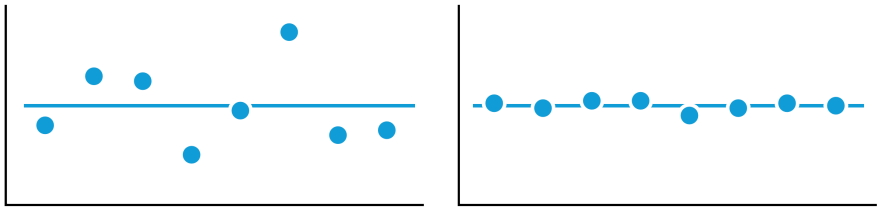
Every node will get the score by calculating the ratio of the time that an activity was running in the segment the node represents. The numbers will be normalised so that the whole month always has a score of 1.0. The reason for the normalization is that we want to differentiate the cases where some activity is really tied to some part of the month from those that are used all over the month.

1.3 Every n days

The last part is detecting whether an activity is used every n days, with the possible small inconsistencies.

This is the simplest of all the methods, since it can be implemented by calculating the distances between every start of an activity, calculating the arithmetic mean (or finding the best constant approximation by using some alternative method), and checking whether the dispersion is small enough.

The following graph demonstrates the difference between these two cases – they have the same mean, but in the left case it is obvious that there is no usage pattern because the dispersion is too high.



2 Activity scoring based on the context

The previous methods can easily be extended to support having different scores for different context conditions such as the location.

Simply, instead of having one week-data for one week, we could have one record for one location. This shouldn't be done on per-latitude/longitude, but only for the registered locations the user saved as important.

If the location is not important for the specific recommendation, the score can be simply calculated as a sum of scores for all locations.