Partitionares

Ruben Navasardyan

July 2022

IPA (formerly HarvardX-v2)

This algorithm works by taking a minimum interval size m defined by the number of time-step points in a given interval. I will take the interpolated signal function to be $\mathcal{I}$

The algorithm starts by taking the first timestep as the pivot and the next $m-1$ points will be considered as the minimum first interval (combined together with the pivot). The algorithm further calculates both the mean and the trimmed standard deviation of these timesteps, it later constructs a linear function of the form $f(x) = s(x - x_0) + \mathcal{I}(x_0)$ where s is

$$s = \frac{mean\{\mathcal{I}(x_1) \to \mathcal{I}(x_m)\} - \mathcal{I}(x_0)}{x_m - x_0}$$

Moreover, starting from $m+1$ timestep, we check if $f(x_i) - \textbf{stdev} \leq \mathcal{I}(x_i) \leq f(x_i) + \textbf{stdev} \quad x_i \in [x_{m+1}, x_{\text{end}}]$. If the above fails at some i', then we take $x_{i'}$ as the new temporary pivot and if the statement $f(x_i) - \textbf{stdev} \leq mean\{\mathcal{I}(x_{i'+1}) \to \mathcal{I}(x_{i'+m})\} \leq f(x_i) + \textbf{stdev}$ is true then the $x_{i'}$ is removed as a temporary pivot and the new whole interval is $x_0 \to x_{i'+m}$ and the cycle continues.

On the other hand, if $f(x_i) - \textbf{stdev} \leq mean\{\mathcal{I}(x_{i'+1}) \to \mathcal{I}(x_{i'+m})\} \leq f(x_i) + \textbf{stdev}$ fails, then we would end the interval on the temporary pivot (excluding) and the temporary pivot would become the new pivot and the whole function would be called recursively on this new interval that starts from the new pivot and ends at the original end of the interval.

In the end, the class has an attribute *lister* that is a list of tuples of start and ending timestamps for every subinterval.

Step by step of IPA

Given the interpolated signal data as a function $\mathcal{I}(t)$ and the minimum intervals size m as a number of time steps the partition algorithm works as follows:

1. Pivoting timestep t_i is assigned to be the first timestep i.e. $t_i = t_0$ and we assign the last timestep to be t_l
2. Calculate the standard deviation and the mean of the function values on the interval starting from the timestep following the pivot and ending at the minimum interval timestep (calculated from the pivot) getting $\sigma = \textbf{stdev}\{\mathcal{I}(t_{i+1}) \to \mathcal{I}(t_{i+m})\}$ and $\mu = \textbf{mean}\{\mathcal{I}(t_{i+1}) \to \mathcal{I}(t_{i+m})\}$
3. Derive a linear function of the form $f(t) = s(t - t_i) + \mathcal{I}(t_i)$ where s is

$$s = \frac{\mu - \mathcal{I}(t_i)}{t_{i+m} - t_i}$$

4. Starting from t_{i+m+1} iteratively verify that the statement $f(t) - \sigma \leq \mathcal{I}(t) \leq f(t) + \sigma \quad (1)$ holds for $t \geq t_{i+m+1}$ and there are multiple ways this iterative process terminates:
   (a) if (1) holds for every $t \geq t_{i+m+1}$ and hence the algorithm terminates at $t = t_l$ producing one single interval $[t_i, t_l]$
   (b) if (1) fails for some $t_{i'}$ then consider $l - i'$ (l is the index of the last timestep):
      i. if $l - i' \leq m$ then disregard that $t_{i'}$ doesn't satisfy (1) and have the algorithm terminate after producing the $[t_i, t_l]$ interval
      ii. if $l - i' > m$ then we evaluate a new conditional statement $f(t_{i'+m}) - \sigma \leq \mathbf{mean}\{\mathcal{I}(t_{i'+1}) \rightarrow \mathcal{I}(t_{i'+m})\} \leq f(t_{i'+m}) + \sigma \quad (2)$ and consider the last set of cases:
         A. if (2) is true then proceed to the start of step 4 but use $t_{i'+m+1}$ instead of t_{i+m+1}
         B. if (2) is false then generate the interval $[t_i, t_{i'}]$, assign $t_{i'}$ to be the new pivot and start the same process again from step 2.

HarvardX (old and inefficient algo)

This older algorithm does significantly more complicated mathematics and for that reason (mainly because of the root finding algorithm) takes up quite a lot of time on intervals with large number of timesteps.

It works iteratively through a while loop where it starts from 2 up until some specified number that will represent the largest number of partitions of the interval. These iterations represent the number of subintervals that will be generated, we'll denote them i.

It starts off by passing the signal function that is given as an array (possibly interpolated nparray) and passing it through the sin function and linearly interpolating it. It's passed through the sin function in order to convert rapid changes in the data (that result in steep sections of the graph) to a new graph where the steeper (more volatile) the signal data is, the higher he number of root there are in the corresponding section of the newly interpolated sin function.
The algorithm further finds the roots of this interpolated function (all roots will be found in most cases unless multiple occur within one timestep). This root finding algorithm is the reason to the low efficiency of this algorithm.

Once it's done with finding the roots, it proceeds into the aforementioned loop. For a given iteration i, the algorithm partitions he interval into i number of equal subintervals and using the above root finding algorithms output (that is a list of roots values) we calculate the density of roots (ie the total number of roots in an interval) in each one of these subintervals. Once it's done calculating the density of every subinterval, it calculates the median of all of these densities and categorizes them as having a density higher or lower/equal. Moreover, it calculates the mean of both the lower/equal and higher densities and subtracts the difference getting the Δ_{avg} (in case if either one of these categories is empty, Δ_{avg} is 0 by default).

It later proceeds to do the same steps of partitioning and calculating the new Δ_{avg} for $i + 1 <$ *specified limit* and compare with the previous Δ_{avg} for i and if there is more that a 5% increase from the previous one then the Δ_{avg} for $i + 1$ is set as the new "official" Δ_{avg} and these steps of recalculating a new Δ_{avg} and comparing repeat up until the loop ends leaving only the highest Δ_{avg} as the one to be used and its corresponding i values as the maximum number of partitions that the original interval will have. The reason to why I take the largest statistically significant (set at 5%) value of Δ_{avg} is that it represents the largest difference between the highest and the lowest average

densities meaning fewer high density points in the low density category and fewer low density points in the high density category.

Lastly, once the algorithm is finished picking the appropriate i value, it proceeds to merge the consecutive subintervals that have a "close enough" of a density in order to remove unnecessary partition of the interval and save time computing the ODE values.

The concept of "Close enough" is defined recursively with the help of the standard deviation of all densities. It starts by taking the first subinterval and checking whether the next interval's density is within the standard deviation of the previous interval's density. If it t within the standard deviation,then the two subintervals are merged and the mean f their densities in calculated, next the third subinterval is taken and we compare whether its density is within the standard deviation of the now merged subinterval whose density we take as the mean of the first and the second intervals' densities, if it it, we merge all three subintervals together and we calculate the new merged subintervals' density as the mean of the densities of the three subintervals and proceed the same way up until there is a subinterval whose density is not within the standard deviation of the previous (merged) subinterval's density. In that case we consider this latest subinterval as the new starting point and we continue to check and merge the subsequent intervals.

Once the algorithm is finished with the whole check and merge part, it return a list of tuples of two points, starting and ending timestamps of every interval.