Runtimes Writeup

Annu Indraganti

January 2025

1 Introduction

This experiment was a means of understanding the runtime of the classical approach to finding persistence diagrams for a set of data and a new approach that was proposed in paper [1].

The classical approach begins a set of data points of size d and a grid matrix of size $n \times n$. A function is then applied on the grid against the set of data, such as nearest-neighbors (NN), kernel density estimation (KDE), or distance-to-measure (DTM). We then pass this grid of values into a Cubical function, which calculates the Cubical complex of this grid. From there, we are able to calculate a persistence diagram using this complex. This general strategy of finding persistence diagrams is effective and has its theoretical bases that stand well. Now, to shorten the scope, we will only consider finding the grid of calculated values. In this part of the process, the theoretical claims state that the runtime of these operations on a grid is approximately $O(n^3)$, a fairly expensive operation.

The new approach to finding persistence diagrams attempts to remove the aspect of a grid entirely. By using weighted filtrations as described in [1], we are able to bypass the grid and only have our algorithms scale by the number of data points. According to theoretical backings, this scales to $O(d^{2.7})$ approximately.

2 Methodology

I made functions in Julia that calculates the grid of values the classical way. To calculate values the new way, I used the RobustTDA.jl package. I tested these functions across four data sizes of 100, 300, 700, and 1,000. For my functions, which are the only ones that depend on a grid size, I tested these across four grid sizes which were 50, 100, 150, and 200. This means that the first grid was a 50×50 grid and so on. To evaluate the functions, I used BenchmarkTools.jl which ran each of the functions ten times for a single result. In each of the runs, the minimum, maximum, median, and mean runtimes were extracted and recorded.

3 Results

To begin, the plot below illustrates the runtime scaling as it pertains to each dataset size as the scale of the grid increases. Keep in mind that though these plots are next to each other, the scales of the y-axis vary. The y-axis scales from $1e^6$ to $1e^9$ for each grid size.

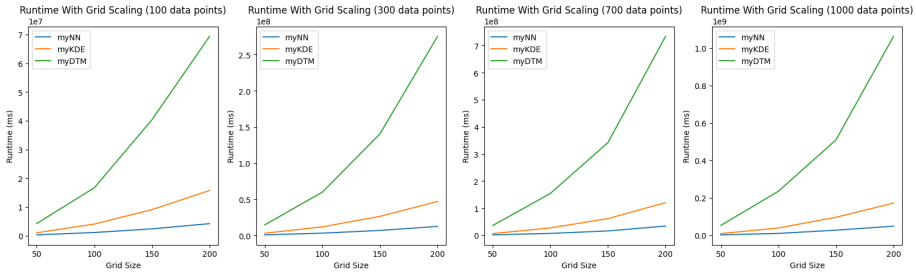

Figure 1: Grid Scaling Per Data Size

These figures all illustrate the same concept. The grid scaling follows some type of exponential curve as the grid sizes increase across all dataset sizes. As the dataset size grows, the scale of the time also increases. We also see which calculation is fastest, that being NN, then KDE, then DTM. However, to tell really how the dataset size scales in runtime, we need to see more graphs. Below are the results:

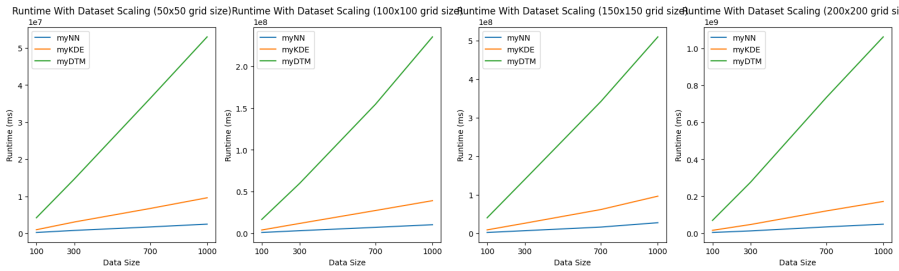

Figure 2: Data Scaling Per Grid Size

As we can see, the dataset size only scales the runtime linearly in comparison to the grid size. The calculation method is also ordered the same as before in terms of speed. This makes it a minor term in comparison to the grid size. Once again, for the images above, keep in mind that the scale of the y-axes are different for each subsequent image.

Now to compare the RobustTDA functions, we will first start with the same process above but only considering the dataset size scaling since they do not rely

on grid size. Below are the results: This shows an interesting result. The scaling

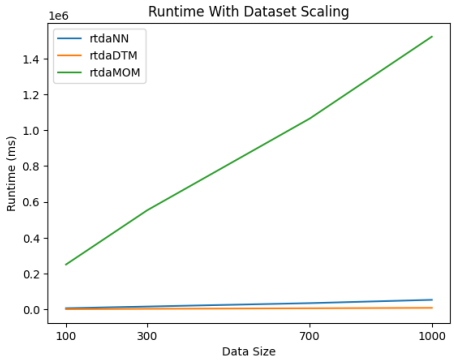

Figure 3: RTDA Function Runtimes Per Dataset Size

for each calculation type seems to be linear. We also see that the y-axis is scaled the lowest of any of the graphs we have seen, indicating that the RobustTDA function perform faster than the classical functions that I created. We also see that the DTM function performed the best out of all functions, even better than the nearest-neighbors function, which was a change from the classical functions.

4 Conclusion

The overarching conclusion was that the RobustTDA functions perform faster than the classical functions. The results also confirm that the scaling of the classical functions as it pertains to grid size is much greater than the scaling as it pertains to the number of data points. This confirms the general theoretical statements mentioned in the introduction.

As for the shortcomings, we may need to further test the RobustTDA functions on greater dataset sizes. Since the sizes of the datasets was relatively small, truly gaining an understanding of the runtime curve was not possible. This will be something I further explore.

5 Sources

[1]: Hal. (n.d.). `https://hal.science/hal-02093445/file/DTM-filtrations_SoCG.pdf`