TDA+Neuroscience course Project: Shape Analysis and classification via Persistent Homology, Part Two

March 27, 2023

This part of the project will be due by 5:00pm Eastern time on Friday, April 7th. You will need to submit a document to Nate and/or Alex via email by the end of this date, with solutions to questions and problems in the following indicated by "**For submission:**". We will have meetings during the week of April 3rd-7th where we discuss your results and the remainder of the project. The new code pertinent to this phase of the project is under the "bottleneck" folder of the github: `https://github.com/TDA-and-Neuro/tda-and-neuro.github.io/tree/master/TDA%2BNeuro%20Project%20SP%202023`. Note to work properly, the folder "bottleneck" must be on the same level/in the same directory as the folder "processed" containing the result of what you computed in the previous phase of the project. If you have an issue remaining with the previous stage of the project and don't have this "processed" folder, that was added to the github as well.

1 Computing bottleneck distances

Previously, you have taken in each of the shapes from the TOSCA database, sub-sampled the shapes to a reasonable number of points, computed geodesic distances for each of these sub-samples, and computed the persistent homology in dimensions 0, 1, and 2, for each resulting metric space using the Vietoris-Rips filtration.

Along the way, you also looked at the discriminative power of the diameter of the shapes. The diameter is a quick and easy invariant to compute, but as we saw, its simplicity made it not particularly strong at telling different shapes apart in general.

In this part of the project, we will use the bottleneck distance between the persistence diagrams that were computed previously to compare shapes. All the necessary new code is in the aforementioned bottleneck folder. Study and run the script `testBottleneck.m`.

For submission: For each of the three bottleneck distance computations with fixed (non-random) diagrams, determine a matching which realizes the bottleneck distance, and show the computation of the cost of this matching.

After this, you'll need to run `computeBottleneckAll.m`. As-is, this code will compute the bottleneck distance between the 0th-dimensional persistence diagrams from all the shapes. Keep in mind that the code will take some time to run (up to a few hours and it cannot be interrupted so take this into account when you decide to run the code).

For submission: Run the code `computeBottleneckAll.m` as-is. Include an image of the resulting plot that you have once the code has finished running. Explain the results, and how this plot compares to the analogous one you computed via diameters.

Note that the last line of the code will save the distance matrix you computed. You will adjust this code and run it multiple times, so be careful with the distance matrix files and rename and/or move them so that they don't get overwritten! If you do, you will need to run the code again from scratch to get them back. We will need these distance matrix files for the next stage of the project.

Observe that the code you ran in the last step took a long time to run, as the number of points in the persistence diagrams is quite large. We did not use the current state-of-the-art algorithm for bottleneck

distance computation, which is implemented in a software package called Hera, but even using said software there is a computational strain for large persistence diagrams (though the tolerance is higher).

Instead of computing the bottleneck distance between PH0 of all the shapes, there are ways in which we can perform a similar task in less time. First, we can modify the code to only work with a subset of shapes at a given time. Lines 6-12 in the code allow one to select only particular classes of shapes to consider, which would cut down on runtime, or if you only want to compare certain shape types simultaneously. Also, there is a variable NP in line 2 of the code, which you can set to a value lower than 200. Doing so truncates the persistence diagrams to only contain the NP most persistent features. This is done via the code `trimBarcode.m`, which you should study. For example, try setting NP = 50, and seeing how much faster the code runs. For the remaining portions, this is a reasonable value to have for NP, though you may try other values. Intuitively, the performance of the approach should improve as we increase NP, but this will make the code take longer to run as a trade-off.

As noted, the code as-is only computes the Bottleneck distance between the PH0 barcodes. Comment out the code for PH0, and follow its example to write code (in the same file `computeBottleneckAll.m`) to perform the same task for PH1, both the bottleneck distance computations and plots (recall, PH1 will be stored in PDs{2}).

For submission: Run the code `computeBottleneckAll.m` after the alterations, and include an image of the resulting plot once the code has finished running. How does the bottleneck distance matrix between dimension 1 persistence diagrams compare to that from dimension 0 persistence diagrams?

Lastly, comment out the code you added for PH1, and add in code to perform the same tasks with PH2.

For submission: Run the code `computeBottleneckAll.m` after the alterations, and include an image of the resulting plot once the code has finished running. Now compare how bottleneck distance performed at discriminating shapes via each of the three different dimensions of persistent homology. Include in your submission an attachment with your version of `computeBottleneckAll.m` after all the alterations you've made.

By looking at the distance matrices, you've gotten an intuitive idea of the performance of bottleneck distance for classifying the shapes in this particular dataset. In the next stage of the project, we will use the distance matrices we stored to analyze the classification via various visualization methods, including dendrograms which we covered in the first project of the semester.