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

February 21, 2023

This phase of the project will be due March 3rd. We will discuss the progress during meetings the week of February 27th-March 3rd and you should have a written submission by March 3rd answering the questions asked throughout the document.

1 Introduction

This project aims to give hands-on experience at a real-world application of Topological Data Analysis.

The first part of the project is an initial exploration of a dataset, including looking at plots, formulating hypotheses about the data, and applying persistent homology to the data. The dataset we will work with is the Non-Rigid World Tosca database of 3D shapes. This database has nearly 150 different shapes from different classes of creatures (dogs, cats, horses, gorillas, etc.) The long-term goal of the entire project is to use our TDA techniques to be able to discriminate between the different creatures present. In other words, we will show our TDA techniques allow us to tell apart different creatures, regardless what pose they may be in.

At least the outline for all code used in this project will be provided, with you sometimes needing to make alterations or additions. All the code (aside from Ripser, see Preparation section item 2 below) can be found in our course github, under the folder "TDA+Neuro Project SP 2023". If you are having any issues with the code as-is, please contact the instructors.

1.1 The Non-Rigid World database

Here's the description of the database we'll work with:

Three-dimensional nonrigid shapes in a variety of poses for non-rigid shape similarity and correspondence experiments. The database contains a total of 148 objects, including 9 cats, 11 dogs, 3 wolves, 17 horses, 15 lions, 21 gorillas, 1 shark, 24 female figures, and two different male figures, containing 15 and 20 poses. The database also contains 6 centaurs, and 6 seahorses for partial similarity experiments. Each object contains approximately 3000 vertices. Two representations are available: MATLAB file (.mat) and ASCII text files containing the 1-based list of triangular faces (.tri), and a list of vertex XYZ coordinates (.vert). A .png thumbnail is available for each object.

For each class (i.e. dogs, cats, horses, gorillas, etc) there are several different poses:

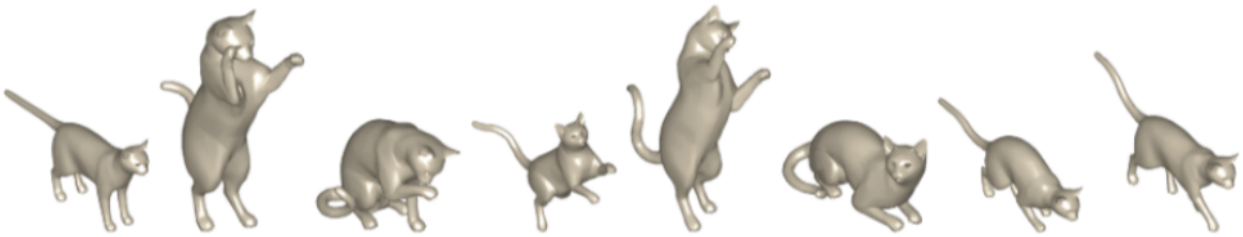

Figure 1: The different poses of a cat in the database.

2 Preparation

Items to be completed before the start of the project:

1. Review concepts: Rips persistent homology, bottleneck distance, subsampling methods.
2. Install Ripser on your computer via the following site: `https://github.com/ctralie/Math412S2017`. Follow the installation instructions on that link. You will be using the function `RipsFiltrationDM.m` from this package. Familiarize yourself with that function, and test out the script `TorusExample.m`. If you have Windows instead of Linux or Mac OS, you will need to change some of the code (to account for the fact Windows uses a .exe extension on executable files.) Let Nate know if you are unable to run/an error results when you try to run `TorusExample.m`. The folder "Math412S2017" should exist on the same level as the folder "nonrigid3d" from Dropbox.
3. Topics to investigate on your own:
   - Geodesic distance on a graph. Dijkstra's algorithm, see: `https://www.mathworks.com/help/matlab/ref/graph.distances.html`.
   - Hierarchical clustering. You will have completed the exercise by now developing your own single-linkage hierarchical clustering code, but see the built-in MATLAB version at `https://www.mathworks.com/help/stats/hierarchical-clustering.html`.
   - Multidimensional Scaling (MDS). See: `https://www.mathworks.com/help/stats/cmdscale.html`. This will be used in a later stage of the project.

3 Geodesic distances

The goal is to be able to discriminate shapes from different classes without being overly sensitive to the fact that many different poses exist within shapes from the same class.

For this reason, we will use geodesic instead of Euclidean distances. In order to compute geodesic distances, we'll use the data contained in the shape data field carefully. The data file for each pose, for example `cat1.mat`, contains a struct with fields `X, Y, Z` and `TRIV`. This last field contains a triangulation of the shape. The function `util/buildGraph.m` builds a (weighted) graph from the data of `X, Y, Z` and `TRIV`.

Study this function. Then study and test the script `preprocessing/testCat1.m`. The native MATLAB function we are using to compute distances on a graph is "distances" which is based on Dijkstra's algorithm.

Write a paragraph explaining what occurred when the script `preprocessing/textCat1.m` ran, what the output of the function represents, and how it can be used.

Next, write a paragraph explaining why we are using geodesic distance instead of Euclidean distance. Browse through the pictures of the shapes from the database and find some shape which for which you expect the Vietoris-Rips persistent homology of that shape under geodesic and Euclidean distance to be different. Explain why you think this difference will exist, and describe what you expect the two different persistence diagrams under the two different distances to be like.

4 Preprocessing the shapes

The first step in the pipeline is to preprocess the shapes in order to reduce the number of points from several thousands to a few hundred so that Ripser can handle them quickly. The following tasks are all done via the function `preprocessing/preprocessAll.m`, which you should run. For each shape in the folder `nonrigid3d:`

1. read the shape into MATLAB.
2. create a (weighted) graph representation with `util.buildGraph.m`
3. subsample it via FPS. The function `preprocessing/processAll.m` uses `NFPS = 200` by default.

4. use the MATLAB function `distances.m` to compute geodesic distances on the graphs.

5. compute the Vietoris-Rips barcodes for persistent homology in dimensions 0, 1, and 2 for each shape. Based on our preprocessing methods, how many points do you expect to have in the 0-dimensional persistence diagram for each shape and why?

6. create a struct called `shape` and attach fields `name, dm` (for the distance matrix).

7. save the resulting struct to the folder `processed`. NOTE: the code will write `pre_cat0.mat` as a result of preprocessing `cat0.mat` and so on.

All of the above will be executed by running `preprocessing/processAll.m`. Be sure to thoroughly read and understand this code.

4.1 Initial tests to do

Go to the folder `processed`. Load a few of the preprocessed shapes and check how many points you have in each barcode. To do this, (for example) while in the folder "processed" run `x = load('pre_cat0.mat')` and then `x.shape.PDs{1}` and you will see the zeroth-dimensional persistent homology of the first cat pose. If you browse, you will see that some of the shapes first-dimensional persistence barcodes have hundreds of points! This may potentially be an issue later, but we will navigate that in a later part of the project.

4.2 Initial baseline test: comparing diameters

Different shapes in the database have different sizes. As such the *diameter* of a shape (defined as the maximal distance across all pairs of points) might already be a reasonable way to discriminate between different classes. Try constructing the matrix `dDiam` such that `dDiam(i,j)` equals $|diam(S_i) - diam(S_j)|$ for all shapes S_i and S_j. See `computeDiamAll.m`, and study and run this script to perform the above task. Explain the results. Was diameter a good way to discriminate between the different classes? Do you expect a persistent homology based approach would be better?