Persistent Homology

Greg Hamilton

30.11.20

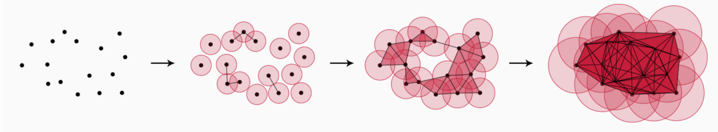
Outline

Motivation: Brain Artery Tree Structure

- Goal: assess statistical difference in brains of different ages via structure of artery trees
- Apart from clustering, artery trees might exhibit "looped-ness", or even voids
- Feature extraction is a common goal for machine learning techniques, but what about near-topological structures?
- Persistent homology provides a quantitative tool to ascertain large scale structure in data, in a manner that's semi-robust to noise.

High Level Description of Persistent Homology

- Given a point cloud of data, define distances between data points
- Consider all of the points that satisfy a pairwise distance relation/threshold. These points define a simplicial complex.
- For each distance threshold, we have a different simplicial complex, inducing a nested sequence of ever-expanding simplicial complexes.
- We can calculate the "topology" of each simplicial complex
- We can track the topology through the sequence of nested simplicial complexes and thereby identify large-scale structure.

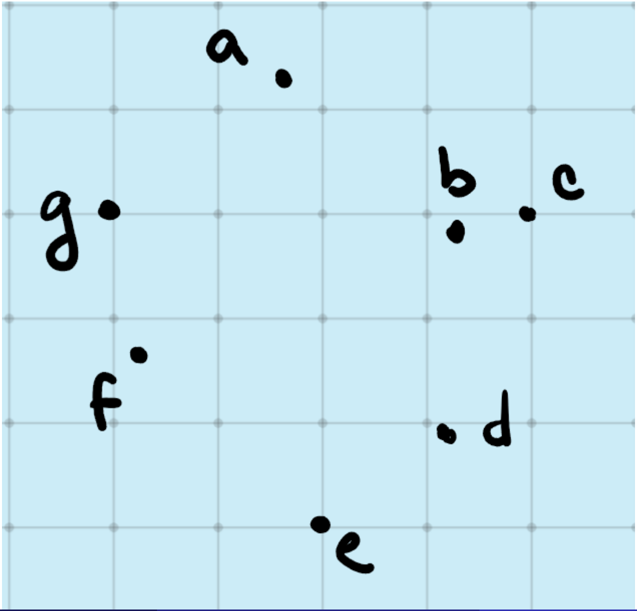


https://christian.bock.ml/posts/persistent_homology/

Worked Example

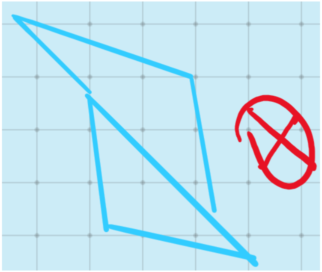
We'll work through an example of calculating homology, and then look at some Python code for persistent homology.

A point cloud



What's a simplicial complex?

- 0-simplices are points
- 1-simplices are lines
- 2-simplices are triangles
- 3-simplices are tetrahedra
- non-empty intersection of two simplices is a face of both (e.g., if two triangles are touching, then they share a full edge or just a point, no half-assing).

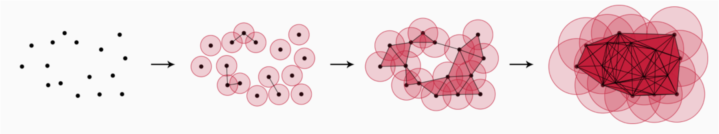


Choose your metric!

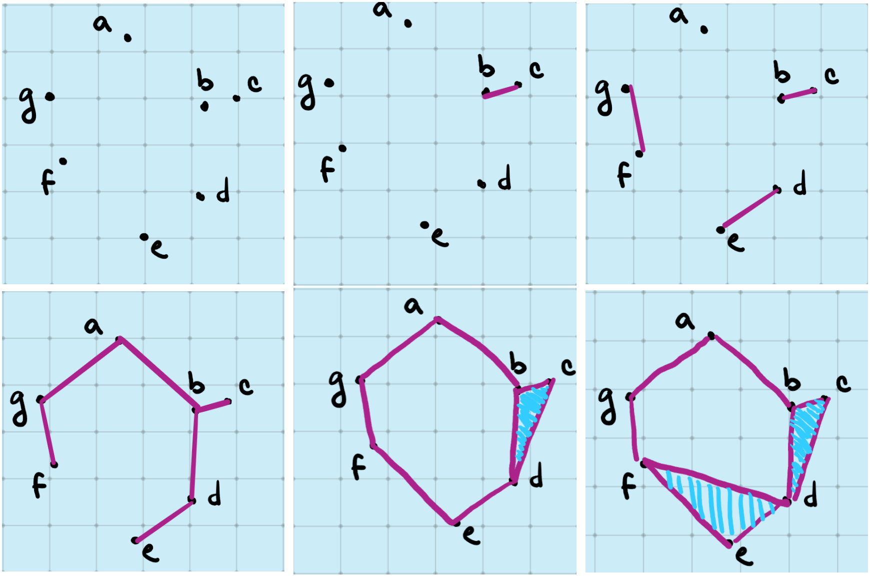
Main three:

- Vietoris-Rips
- Cech
- α-complex

Apparently you can also use similarity measures (edit distance, Levenshtein, etc.)



An incomplete sequence of simplicial complexes



Calculating homology

- Denote the formal sum of n-simplices in K_i by $C_n^{(i)}$ (for those keeping score at home, our field is $\mathbb{Z}_2$).
- The boundary operator $\partial_n : C_n^{(i)} \rightarrow C_{n-1}^{(i)}$ maps faces to boundaries, so we have a *chain complex*

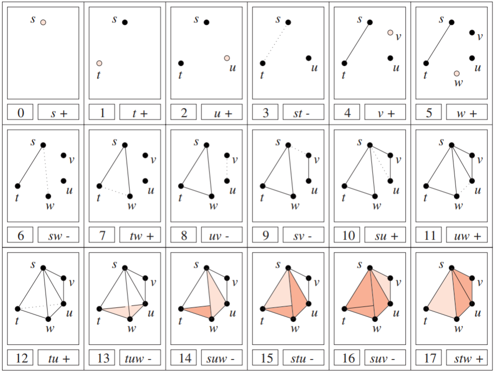
$$\dots \xrightarrow{\partial_{n+1}} C_n^{(i)} \xrightarrow{\partial_n} C_{n-1}^{(i)} \xrightarrow{\partial_{n-1}} \dots C_1^{(i)} \xrightarrow{\partial_1} C_0^{(i)} \xrightarrow{\partial_0} \varnothing \tag{1}$$

- Note that $\partial^2 = 0$, which implies the boundary of a boundary is $\varnothing$
- This further implies that the image of ∂_n is in the kernel of ∂_{n-1}
- Denote $Z_n := \ker\partial_n$ the *cycles*, and $B_n := \mathrm{im}\partial_{n+1}$ the *boundaries*
- The n-th homology group $H_n^{(i)} := Z_n^{(i)} / B_n^{(i)}$
- rank of $H_n^{(i)}$ is the n-th Betti number

Computationally...

- $\partial_n : C_n \to C_{n-1}$ is just a matrix
- $\text{im}\partial_n$ is identical to an orthonormal basis for the image of ∂_n
- $\ker\partial_n$ is the kernel of an operator, which can be found by Gaussian elimination (rref)
- You can find the Betti numbers without ever actually computing the quotient groups (rank becomes subtraction when taking quotients of vector spaces)
- If you can determine when an added simplex is a cycle, then the determination of Betti numbers is even easier:

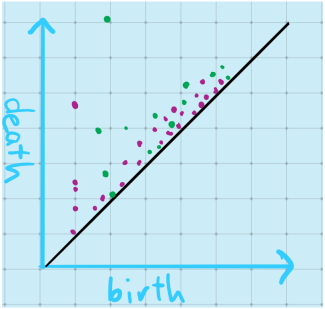
Computationally...



Edelsbrunner, Letscher Zomorodian Topological Persistence and Simplification. Discrete Comput Geom 28, 511–533 (2002). https://doi.org/10.1007/s00454-002-2885-2

Persistence

- Homomorphism $f_p^{ij} : H_p(K_i) \to H_p(K_j)$ helps us identify when a homology group is born and dies
- More particularly, keeping track of filtration levels when a simplex births a cycle and another kills the same cycle (makes a boundary)
- We can keep this information in a *persistence diagram*
- The farther a data point is from the diagonal, the longer-lived (and thus, more large scale) structure it is



Computational Packages

- Several packages available with Python wrappers
- Wide variety of metrics available, distance function can be entered as a NumPy array, etc.
- Let's look at some Python code using GUDHI