Topological Data Analysis for Machine Learning
Lecture 2: Computational Topology

Bastian Rieck

Pseudomanifold

Preliminaries

Do you have feedback or any questions? Write to bastian.rieck@bsse.ethz.ch or reach out to @Pseudomanifold on Twitter. You can find the slides and additional information with links to more literature here:

https://topology.rocks/ecml_pkdd_2020

Recap

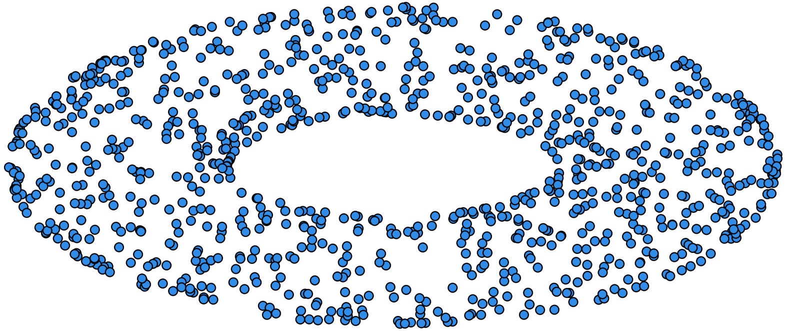
- To distinguish between topological objects, we can use *Betti numbers*.
- Betti numbers count high-dimensional holes.
- Their calculation requires a simplicial complex and some linear algebra.

In this lecture

Going from theory to practice

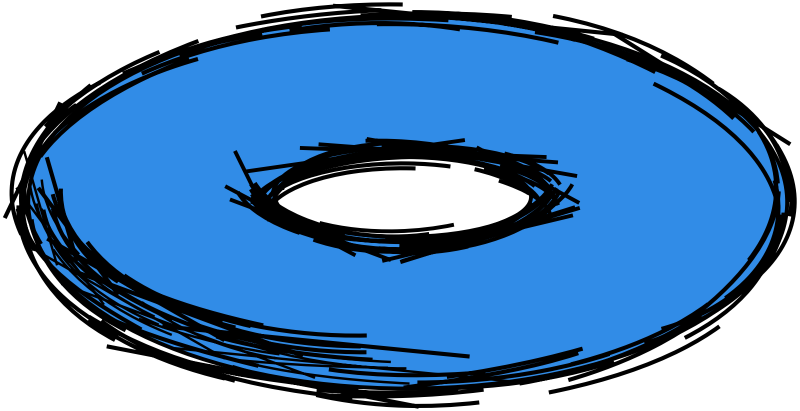
Real-world objects are typically not described in terms of simplicial complexes. How to bridge this gap?

Example



What we get

Example



What we see

From point clouds to simplicial complexes

Vietoris–Rips complex

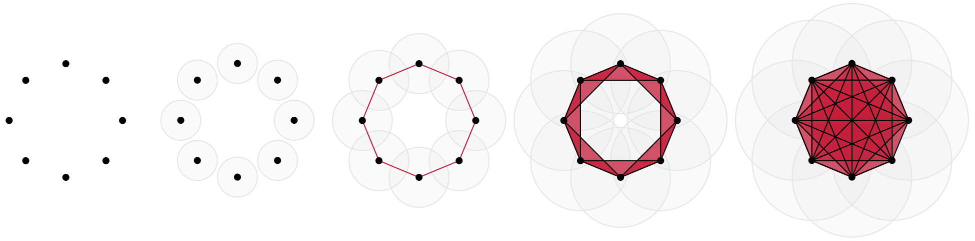
Given a set of points $\mathcal{X} = \{x_1, \dots, x_n\}$ and a metric dist such as the Euclidean distance, pick a threshold ϵ and build the Vietoris–Rips complex $\mathcal{V}_\epsilon$ defined as:

$$\mathcal{V}_\epsilon(\mathcal{X}) := \{\sigma \subseteq \mathcal{X} \mid \forall u, v \in \sigma : \operatorname{dist}(u, v) \leq \epsilon\}$$

Equivalently, $\mathcal{V}_\epsilon$ contains all simplices whose *diameter* is less than or equal to ϵ.

Example

Vietoris–Rips construction



$\epsilon = 0.0$ $\epsilon = 0.1$ $\epsilon = 0.2$ $\epsilon = 0.5$ $\epsilon = 1.0$

Draw Euclidean balls (circles) of diameter ϵ and create a k-simplex σ for each subset of $k + 1$ points that intersect pairwise.

Some details about this construction

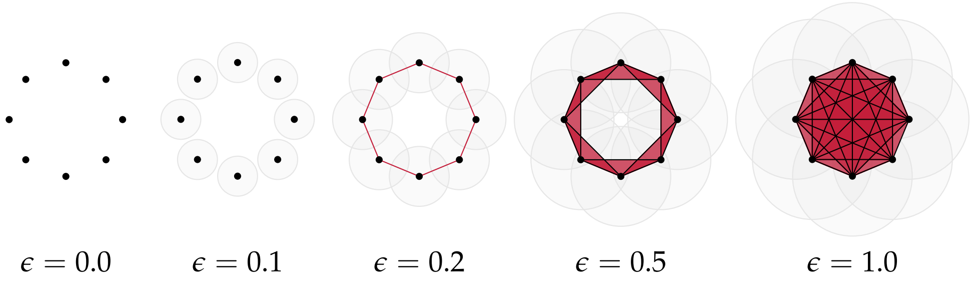
- This construction dates back to a 1927 article by Leopold Vietoris[1], who is shown on the left.
- A 2010 paper by Afra Zomorodian[2] describes several construction algorithms.
- The basic idea is to build higher-dimensional simplices *inductively* from lower-dimensional ones.
- In the worst case, the Vietoris–Rips complex will contain all 2^n subsets of its underlying point cloud $\mathcal{X}$!

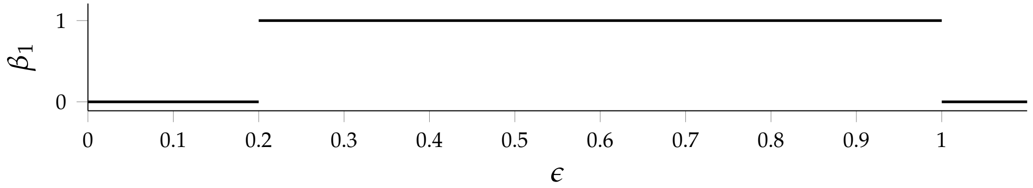
[1] L. Vietoris, 'Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen', *Mathematische Annalen* 97.1, 1927, pp. 454–472

[2] A. J. Zomorodian, 'Fast construction of the Vietoris–Rips complex', *Computers & Graphics* 34.3, 2010, pp. 263–271

Example

The Betti numbers of a Vietoris–Rips complex





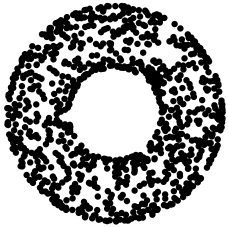
Issues with this approach

- How to pick ϵ?
- There might not be one ‘correct’ value for ϵ.
- Computationally inefficient; matrix reduction has to be performed for *every* simplicial complex.

Just calculate topological features for *all* possible scales!

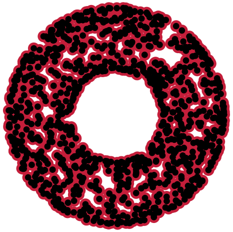
Intuition

Go through all scales and *track* topological features



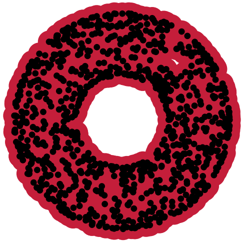
Intuition

Go through all scales and *track* topological features



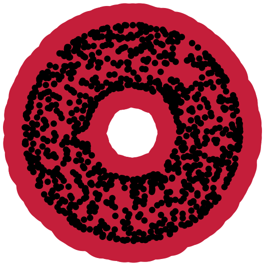
Intuition

Go through all scales and *track* topological features



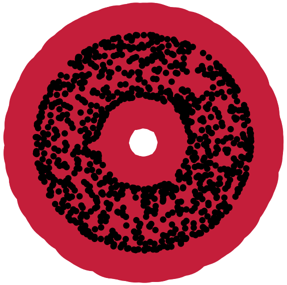
Intuition

Go through all scales and *track* topological features



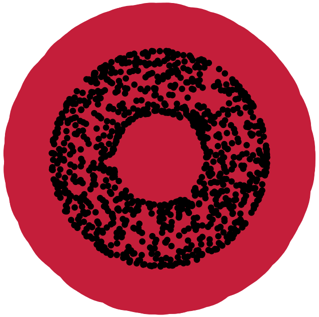
Intuition

Go through all scales and *track* topological features



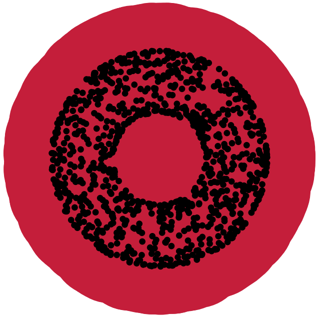
Intuition

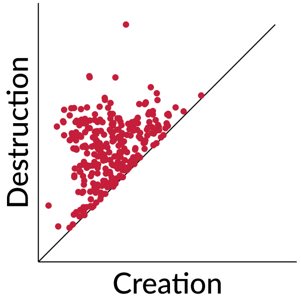
Go through all scales and *track* topological features



Intuition

Go through all scales and *track* topological features

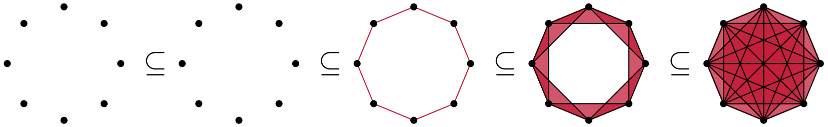


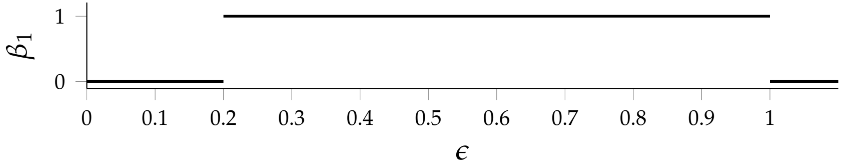


Nesting property

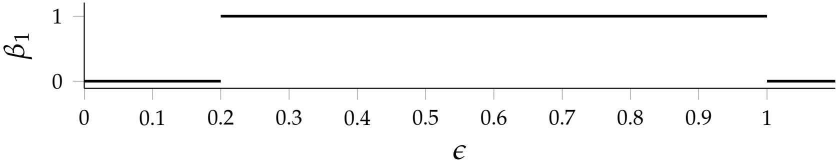
Given $\epsilon_1 \leq \epsilon_2$, we have $\mathcal{V}_{\epsilon_1} \subseteq \mathcal{V}_{\epsilon_2}$. This *nesting property* is the key towards improved calculations!

Example





Filtrations



The Betti number of the data *persists* over a range of the threshold parameter ϵ. To formalise this, assume that simplices in the Vietoris–Rips complex are added one after the other. This gives rise to a *filtration* , i.e.

$$\emptyset = \mathrm{K}_0 \subseteq \mathrm{K}_1 \subseteq \cdots \subseteq \mathrm{K}_{n-1} \subseteq \mathrm{K}_n = \mathcal{V}_\epsilon,$$

where each K_i is a valid simplicial subcomplex of $\mathcal{V}_\epsilon$.

Chain complexes and filtrations

Since $\mathrm{K}_i \subseteq \mathrm{K}_j$ for $i \leq j$, we obtain a sequence of homomorphisms connecting the homology groups of each simplicial complex, i.e.

$$f_p^{i,j} \colon H_p(\mathrm{K}_i) \to H_p(\mathrm{K}_j),$$

which in turn gives rise to a sequence of homology groups, i.e.

$$0 = H_p(\mathrm{K}_0) \xrightarrow{f_p^{0,1}} H_p(\mathrm{K}_1) \xrightarrow{f_p^{1,2}} \dots \xrightarrow{f_p^{n-2,n-1}} H_p(\mathrm{K}_{n-1}) \xrightarrow{f_p^{n-1,n}} H_p(\mathrm{K}_n) = H_p(\mathcal{V}_\epsilon),$$

with p denoting the dimension of the corresponding homology group.

Interlude

This is not ‘abstract nonsense’

The fact that we can *reformulate* the previously-seen concepts in the context of a filtration illustrates how *generic* this formulation is!

See the blog post ‘What is a Functor?’ by Tai-Danae Bradley for an excellent explanation of this.

Persistent homology group

Given two indices $i \leq j$, the pth persistent homology group $H_p^{i,j}$ is defined as

$$H_p^{i,j} := Z_p\left(\mathrm{K}_i\right) / \left(B_p\left(\mathrm{K}_j\right) \cap Z_p\left(\mathrm{K}_i\right)\right),$$

which contains all the homology classes of K_i that are still present in K_j.

Implication

We can calculate a new set of homology groups alongside the filtration and assign a 'duration' to each topological feature.

Persistent homology

Tracking of topological features

- *Creation* in K_i : $c \in H_p(\mathrm{K}_i)$, but $c \notin H_p^{i-1,i}$
- *Destruction* in K_j : c is created in K_i, with $f_p^{i,j-1}(c) \notin H_p^{i-1,j-1}$ and $f_p^{i,j}(c) \in H_p^{i-1,j}$

The *persistence* of a class c that is created in K_i and destroyed in K_j is defined as

$$\mathrm{pers}(c) := |\,\mathrm{w}(j) - \mathrm{w}(i)|,$$

where $\mathrm{w}\colon \mathbb{Z} \to \mathbb{R}$ assigns each simplicial complex of the filtration a weight, such as an associated distance, or an index. Persistence thus measures the 'scale' at which a certain topological feature occurs.

Standard filtrations

The distance filtration

Given a distance metric dist, such as the Euclidean metric, the *distance filtration* assigns weights based on pairwise distances between points:

$$\mathrm{w}(\sigma) := \begin{cases} 0 & \text{if } \sigma \text{ is a vertex} \\ \mathrm{dist}(u, v) & \text{if } \sigma = \{u, v\} \\ \max_{\tau \subseteq \sigma} \mathrm{w}(\tau) & \text{else} \end{cases}$$

Simplices need to be sorted in *ascending* order of their weights; in case of a tie, faces precede co-faces.

Persistent homology is capable of *preserving* distances under random projections[3].

[3]D. R. Sheehy, 'The Persistent Homology of Distance Functions under Random Projection', *Proceedings of the 30th Annual Symposium on Computational Geometry*, 2014, pp. 328–334

Standard filtrations

The sublevel set filtration

Given a scalar function $f\colon \operatorname{vertices}(\mathrm{K}) \to$ defined on the vertices of a simplicial complex, such as a temperature measurement, the *sublevel set filtration* propagates those weights through a simplicial complex:

$$\mathrm{w}(\sigma) := \begin{cases} f(v) & \text{if } \sigma = \{v\} \\ \max_{\tau \subseteq \sigma} \mathrm{w}(\tau) & \text{else} \end{cases}$$

Simplices need to be sorted in *ascending* order of their weights; in case of a tie, faces precede co-faces.

Conversely, one can calculate the *superlevel set filtration* by using $\min$ instead of $\max$ and sorting in simplices in *descending* order of their weights.

Example

Boundary matrix calculation alongside a filtration

$$M = \begin{array}{c|ccccccc|c} & a & b & c & ab & bc & ac & abc & \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ab \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & bc \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ac \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc \end{array}$$

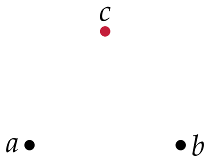
Example

Boundary matrix calculation alongside a filtration

$$M = \begin{array}{c|ccccccc|c} & a & b & c & ab & bc & ac & abc & \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & 0 & a \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & b \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ab \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & bc \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ac \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc \end{array}$$

Example

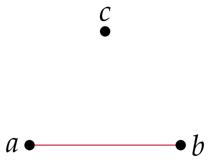
Boundary matrix calculation alongside a filtration



$$M = \begin{array}{c} \\ \\ \end{array}\begin{array}{ccccccc|c} a & b & c & ab & bc & ac & abc & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & b \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & c \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & ab \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & bc \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & ac \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc \end{array}$$

Example

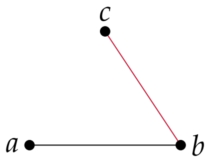
Boundary matrix calculation alongside a filtration



$$M = \begin{array}{c|ccccccc|c} & a & b & c & ab & bc & ac & abc & \\ \hline & 0 & 0 & 0 & 1 & 0 & 0 & 0 & a \\ & 0 & 0 & 0 & 1 & 0 & 0 & 0 & b \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ab \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & bc \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ac \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc \end{array}$$

Example

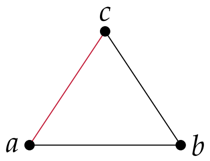
Boundary matrix calculation alongside a filtration



$$M = \begin{array}{c|ccccccc|c}
 & a & b & c & ab & bc & ac & abc & \\
 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & a \\
 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & b \\
 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & c \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ab \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & bc \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ac \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc
\end{array}$$

Example

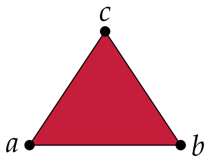
Boundary matrix calculation alongside a filtration



$$M = \begin{array}{c|ccccccc|c} & a & b & c & ab & bc & ac & abc & \\ \hline & 0 & 0 & 0 & 1 & 0 & 1 & 0 & a \\ & 0 & 0 & 0 & 1 & 1 & 0 & 0 & b \\ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & c \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ab \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & bc \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & ac \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc \end{array}$$

Example

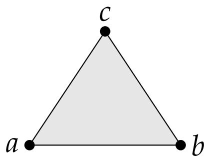
Boundary matrix calculation alongside a filtration



$$M = \begin{array}{c|ccccccc|c} & a & b & c & ab & bc & ac & abc & \\ \hline & 0 & 0 & 0 & 1 & 0 & 1 & 0 & a \\ & 0 & 0 & 0 & 1 & 1 & 0 & 0 & b \\ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & c \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & ab \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & bc \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & ac \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc \end{array}$$

Example

Boundary matrix calculation alongside a filtration



$$M = \begin{array}{c} \\ \\ \end{array}\begin{array}{ccccccccc} & a & b & c & ab & bc & ac & abc & \\ \left(\right. & 0 & 0 & 0 & 1 & 0 & 1 & 0 & \left.\right) a \\ & 0 & 0 & 0 & 1 & 1 & 0 & 0 & b \\ & 0 & 0 & 0 & 0 & 1 & 1 & 0 & c \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & ab \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & bc \\ & 0 & 0 & 0 & 0 & 0 & 0 & 1 & ac \\ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & abc \end{array}$$

Boundary matrix reduction by column operations

```
Let $M$ be a *boundary matrix*
for $i = 1$ do
    while $\exists i' < i : \mathrm{low}(i') = \mathrm{low}(i) \neq 0$ do
        $M(i) = M(i) \oplus M(i')$
    end while
end for
```

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Boundary matrix reduction by column operations

```
Let $M$ be a *boundary matrix*
for $i = 1$ do
    while $\exists i' < i : \operatorname{low}(i') = \operatorname{low}(i) \neq 0$
    do
        $M(i) = M(i) \oplus M(i')$
    end while
end for
```

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Boundary matrix reduction by column operations

```
Let $M$ be a *boundary matrix*
for $i = 1$ do
    while $\exists i' < i : \mathrm{low}(i') = \mathrm{low}(i) \neq 0$ do
        $M(i) = M(i) \oplus M(i')$
    end while
end for
```

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Boundary matrix reduction by column operations

```
Let $M$ be a *boundary matrix*
for $i = 1$ do
    while $\exists i' < i : \mathrm{low}(i') = \mathrm{low}(i) \neq 0$ do
        $M(i) = M(i) \oplus M(i')$
    end while
end for
```

$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Boundary matrix reduction by column operations

```
Let $M$ be a *boundary matrix*
for $i = 1$ do
    while $\exists i' < i : \mathrm{low}(i') = \mathrm{low}(i) \neq 0$
    do
        $M(i) = M(i) \oplus M(i')$
    end while
end for
```

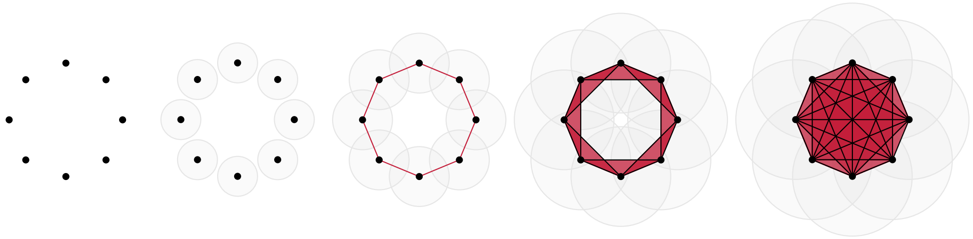
$$M = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Using the reduced boundary matrix

$$\begin{array}{cccccccc} a & b & c & ab & bc & ac & abc & \\ \left(\begin{matrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{matrix}\right) & \begin{matrix} a \\ b \\ c \\ ab \\ bc \\ ac \\ abc \end{matrix} \end{array}$$

- If column i is empty, then σ_i is a *positive* simplex that *creates* a topological feature.
- If column j is non-empty with $\mathrm{low}(j) = k$, then σ_j is a *negative* simplex that *destroys* the topological feature created by σ_k.
- For example, simplex abc destroys the cycle created by ac.

Illustrative example I



$\epsilon = 0.0$ $\epsilon = 0.1$ $\epsilon = 0.2$ $\epsilon = 0.5$ $\epsilon = 1.0$

Here, the topological feature is the circle that underlies that data. Since it persists from $\epsilon = 0.20$ to $\epsilon = 1.0$, its persistence is $\mathrm{pers} = 1.0 - 0.20 = 0.80$.

Topological features and how to track them

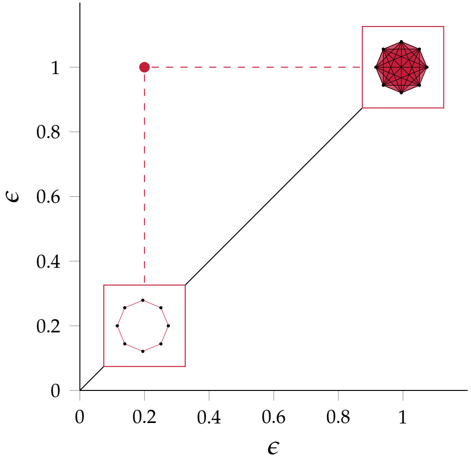
Types of topological features

- Dimension 0: *connected components*
- Dimension 1: *cycles*
- Dimension 2: *voids*

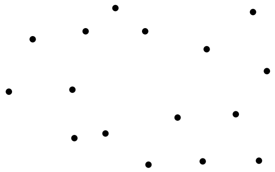
Given a topological feature with associated simplicial complexes K_i and K_j, store the point $(\mathrm{w}(i), \mathrm{w}(j))$ in a *persistence diagram*.

If a feature is *never* destroyed, we assign it a weight of ∞.

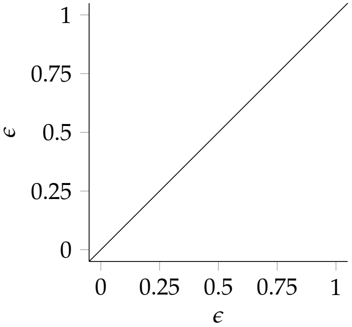
Example persistence diagram



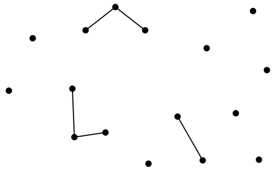
Illustrative example II



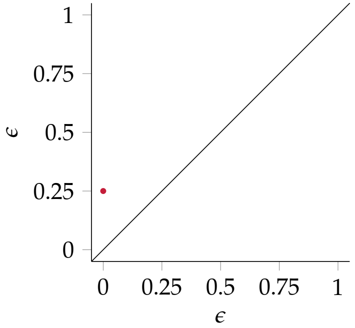
$\epsilon = 0.00$: 16 connected components



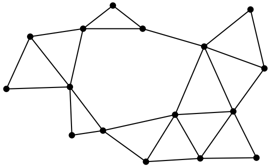
Illustrative example II



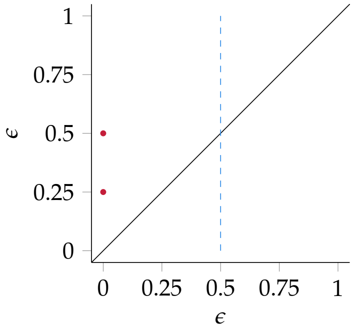
$\epsilon = 0.25$: 11 connected components



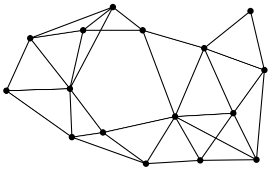
Illustrative example II



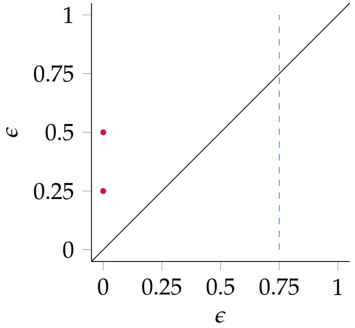
$\epsilon = 0.50$: 1 connected component, 12 cycles



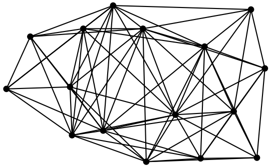
Illustrative example II



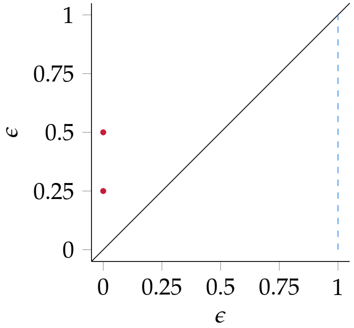
$\epsilon = 0.75$: 1 connected component, 19 cycles



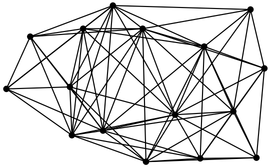
Illustrative example II



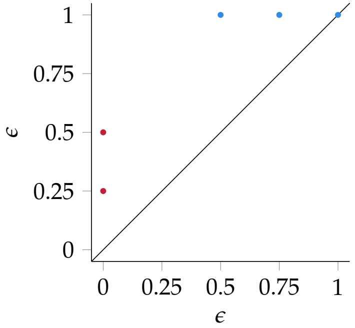
$\epsilon = 1.00$: 1 connected component, 57 cycles



Illustrative example II



$\epsilon = 1.00$: 1 connected component, 57 cycles



Calculation in practice

Ripser: efficient computation of
Vietoris–Rips persistence barcodes

Ulrich Bauer*

August 8, 2019

arXiv:1908.02518v1 [math.AT] 7 Aug 2019

We present an algorithm for the computation of Vietoris–Rips persistence barcodes and describe its implementation in the software Ripser. The method relies on implicit representations of the coboundary operator and of the filtration order of the simplices, avoiding the explicit construction and storage of the filtration coboundary matrix. Our implementation shows substantial improvements over previous software both in time and memory usage.

1 Introduction

Persistent homology is a central tool of computational topology and topological data analysis. It captures topological features of a *filtration*, a one-parameter family of growing topological spaces, and tracks their life span throughout the parameter range in the form of a collection of intervals called the *persistence barcode*. One of the most common geometric constructions for such a filtration is the Vietoris–Rips complex, which is constructed from a finite metric space by connecting a subset of the points by a simplex whenever the diameter of the subset does not exceed a specified threshold.

The computation of persistent homology has attracted strong interest in recent years [7, 23], with at least 14 different implementations publicly available to date [2–4, 15, 17, 21, 24, 25, 28, 30, 31, 33–36]. Over the years, dramatic improvements in performance have been achieved, as demonstrated in recent benchmarks [29].

The predominant approach to persistence computation consist of two steps: the generation of a filtration boundary matrix, and the computation of persistence barcodes using matrix reduction. Among the fastest codes for the matrix reduction step is PHAT [3], which has been created with the goal of assessing and understanding the relation and interplay of the various optimizations proposed in the literature for the matrix reduction algorithm. In the course of that project, it became evident that often the construction of the filtration boundary matrix becomes the bottleneck for the computation of Vietoris–Rips barcodes.

The approach followed in Ripser [1] is to avoid the construction and storage of the filtration boundary matrix as a whole, discarding and recomputing parts of it when necessary. In particular, instead of representing the coboundary map explicitly by a matrix data structure, it is represented as an algorithmic operator, recomputing the coboundary of a simplex whenever needed. The filtration is specified using another algorithmic operator for comparing simplices with respect to their appearance in the filtration order. The initial motivation for pursuing this strategy was purely to reduce the memory usage. Perhaps surprisingly, the method turns out to be substantially faster than accessing the coboundary from memory. This effect can be explained by the fact that on current computer architectures, memory access is much more expensive than elementary arithmetic operations.

The computation of persistent homology implemented in Ripser is based on matrix reduction and uses four key optimizations in order to achieve an efficient implementation, two of which have been proposed in the literature before. While our implementation is specific to Vietoris–Rips filtrations, these ideas are also applicable to persistence computations for other filtrations as well.

Clearing birth columns The standard matrix reduction algorithm does not make use of the special structure of a boundary matrix D, which satisfies $D^2 = 0$, i.e., boundaries are always cycles. Ignoring this structure leads to a large number of unnecessary matrix operations in the matrix reduction, computing a large number of cycles that are not used subsequently. The *clearing* optimization (also called the *twist* optimization), suggested by Chen and Kerber [6], avoids the computation of those cycles.

*Technical University of Munich (TUM), Germany. mail@ulrich-bauer.org

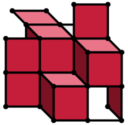
1

- Calculations are bounded by matrix reduction complexity.
- Smart implementations make a difference.
- U. Bauer's `Ripser`[4] is extremely fast.

Speed-ups involve different orderings for the column reduction steps, using implicit representations (which are more efficient than explicit ones), and much more.

[4]U. Bauer, 'Ripser: efficient computation of Vietoris–Rips persistence barcodes', 2019

Topological features of non-simplicial domains



Persistent homology is *not* restricted to the 'triangular' setting. It is also possible to define a filtration over cubical domains[5].

Potential data sources

- Images
- Finite element simulations
- Voxel spaces[6]

The nice thing is that all considerations and concepts apply virtually unchanged!

[5]The image of the 'cubical complex' is modelled after R. Ghrist's excellent book 'Elementary Applied Topology'

[6]B. Rieck et al., 'Uncovering the Topology of Time-Varying fMRI Data using Cubical Persistence', 2020

Current research directions

Properties of filtrations

Can we find filtrations that are more *robust* to noise, easier to calculate, and more expressive?

Sparse filtrations

Simplices are not equally important. Can we find *sparse* filtrations consisting of fewer simplices but with essentially the same topological features?

Take-away messages

- Point clouds can be converted into simplicial complexes.
- Persistent homology is the multi-scale equivalent of simplicial homology.
- The calculation of persistent homology *also* boils down to linear algebra.
- Filtrations are the key for tracking topological features.

https://topology.rocks/ecml_pkdd_2020