MASARYKOVA UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
ÚSTAV MATEMATIKY A STATISTIKY

Diplomová práce

BRNO 2025 TOMÁŠ PETIT

MASARYKOVA
UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
ÚSTAV MATEMATIKY A STATISTIKY

Topological data analysis

Diplomová práce

Tomáš Petit

Vedoucí práce: prof. RNDr. Jan Slovák, DrSc. **Brno 2025**

Bibliografický záznam

Autor:	Bc. Tomáš Petit Přírodovědecká fakulta, Masarykova univerzita Ústav matematiky a statistiky
Název práce:	Topological data analysis
Studijní program:	Matematika
Studijní obor:	Matematika
Vedoucí práce:	prof. RNDr. Jan Slovák, DrSc.
Akademický rok:	2024/2025
Počet stran:	?? + ??
Klíčová slova:	Topologie; Algebraická Topologie; Homologie; Persistentní Homologie; Topologická analýza dat; TDA; Topologické Metody

Bibliographic Entry

Author:	Bc. Tomáš Petit Faculty of Science, Masaryk University Department of mathematics and statistics
Title of Thesis:	Topological data analysis
Degree Programme:	Mathematics
Field of Study:	Mathematics
Supervisor:	prof. RNDr. Jan Slovák, DrSc.
Academic Year:	2024/2025
Number of Pages:	?? + ??
Keywords:	Topology; Algebraic Topology; Homology; Persistent Homology; Topological data analysis; TDA; Topological Methods

Abstrakt

V této bakalářské/diplomové/rigorózní práci se věnujeme ...

Abstract

In this thesis we study ...

MUNI
SCI

MASARYKOVA UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
KOTLÁŘSKÁ 2, 611 37 BRNO
IČ: 00216224
DIČ: CZ00216224

ZADÁNÍ DIPLOMOVÉ PRÁCE

Akademický rok: 2024/2025

Ústav:	Přírodovědecká fakulta
Student:	Bc. Tomáš Petit
Program:	Matematika
Specializace:	Matematika

Ředitel *ústavu* PřF MU Vám ve smyslu Studijního a zkušebního řádu MU určuje diplomovou práci s názvem:

Název práce:	Topological data analysis
Název práce anglicky:	Topological data analysis
Jazyk závěrečné práce:	angličtina

Oficiální zadání:
Goal: The goal is to understand the concepts and tools of Topological Data Analysis, and to be ready to use them in practical tasks. Aim: Depending on the results of the initial period, the student will either focus on theoretical understanding and original research in Mathematics and Statistics, or the focus will be on smart use of advanced tools in solving practical problems, including the implementation issues. One of the resources for real data requiring sophisticated analysis will come form the project Machine Learning in Nanomaterial Biocompatibility Assessment (MUNI/G/1125/2022).

Literatura: RAÚL RABADÁN, ANDREW J. BLUMBERG, Topological Data Analysis for Genomics and Evolution, Cambridge University Press, 2020, DOI: 10.1017/9781316671665

Vedoucí práce:	prof. RNDr. Jan Slovák, DrSc.
Datum zadání práce:	20. 9. 2023
V Brně dne:	25. 7. 2024

Zadání bylo schváleno prostřednictvím IS MU.

Bc. Tomáš Petit, 16. 10. 2023
prof. RNDr. Jan Slovák, DrSc., 17. 10. 2023
RNDr. Jan Vondra, Ph.D., 18. 10. 2023

Poděkování

Na tomto místě bych chtěl(-a) poděkovat ...

Prohlášení

Prohlašuji, že jsem svoji bakalářskou/diplomovou práci vypracoval(-a) samostatně pod vedením vedoucího práce s využitím informačních zdrojů, které jsou v práci citovány.

Prohlašuji, že jsem svoji rigorózní práci vypracoval(-a) samostatně s využitím informačních zdrojů, které jsou v práci citovány.

Brno xx. měsíce 20xx

.....

Tomáš Petit

Contents

List of used notation

Pro snažší orientaci v textu zde čtenáři předkládáme přehled základního značení, které se v celé práci vyskytuje.

$\mathbb{C}$ množina všech komplexních čísel
$\mathbb{R}$ množina všech reálných čísel
$\mathbb{Z}$ množina všech celých čísel
$\mathbb{N}$ množina všech přirozených čísel
$\mathbb{C}$ množina všech komplexních čísel
$\mathbb{R}$ množina všech reálných čísel
$\mathbb{Z}$ množina všech celých čísel
$\mathbb{N}$ množina všech přirozených čísel
$\mathbb{C}$ množina všech komplexních čísel
$\mathbb{R}$ množina všech reálných čísel
$\mathbb{Z}$ množina všech celých čísel
$\mathbb{N}$ množina všech přirozených čísel
$\mathbb{C}$ množina všech komplexních čísel
$\mathbb{R}$ množina všech reálných čísel
$\mathbb{Z}$ množina všech celých čísel
$\mathbb{N}$ množina všech přirozených čísel
$\mathbb{C}$ množina všech komplexních čísel
$\mathbb{R}$ množina všech reálných čísel
$\mathbb{Z}$ množina všech celých čísel

Introduction

To add later.

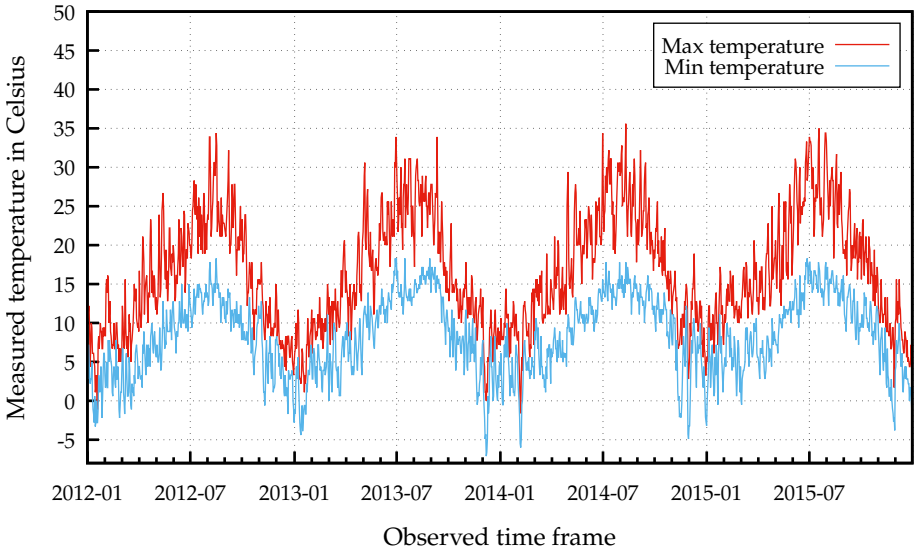
Chapter 1

Why Topology?

Data has shape. This is hardly a new or revolutionary idea in the realm of data analysis and statistics. It is an assumption that we make all the time, even if we do not say it out loud. Whenever one tries to construct a linear regression model, we all have the mental image of a straight line in our minds, which should roughly approximate the data. This is then generalized via hyperplanes in higher dimensions.

Another example would be periodic time series or signals – we all expect to see a "loop" of some sort, given a long enough time interval between the measurements, see for example 1.1. It isn't much of a stretch to imagine that we could use this loop to try to approximate the period of the time series (something that we will actually do in the following chapters, after introducing the necessary mechanisms).

Figure 1.1: Example plot of seasonal temperature changes in Seattle throughout the years.



Clustering algorithms – be it k-means, hierarchical clustering and so on – all

work with the shape and geometry of our data explicitly by partitioning the available search space into the distinct clusters, once a measure of distance (which doesn't need to be a metric, per se) is chosen. The list could go on, but I believe the point was already made. Historically and traditionally, an appropriate analytic model was derived and constructed for each of the above-mentioned methods with a rich theoretical background to justify the results.

While this approach works for most simple cases the average data analyst will encounter on an Excel spreadsheet, thanks to the advancements made in computing power and software engineering, we're collecting massive amounts of complicated, high-dimensional data living faster than we did before, especially in the fields of biology and medical sciences.

Figure 1.2: Time-scaled phylogenetic tree of H3 influenza viruses inferred by BEAST using molecular clock model.

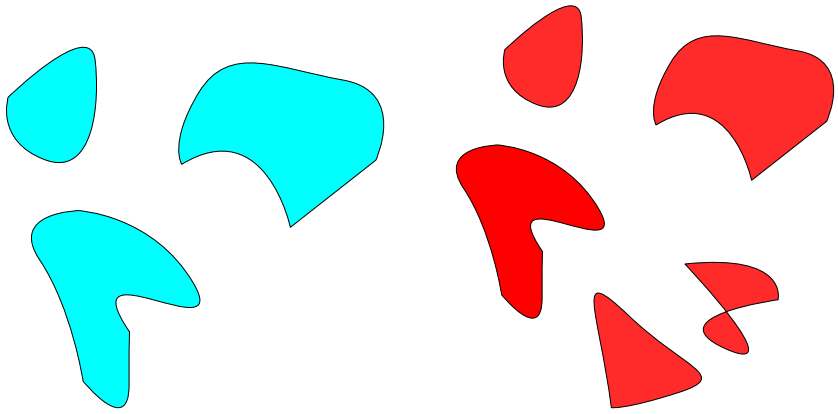
A good example of that would be phylogenetic and evolutionary trees, like the one in 1.2. We might be interested to know whether any mutation occurred, where did they happen and how far were they transmitted down the tree. We might look for re-combinations of the genomic material or both horizontal and vertical transfer of it. All those questions pertain to the shape of the phylogenetic tree and its branching. [1]

The situation only gets more complex with each passing day and batch of collected data. One *could* try to develop analytical models for each case, provide the rigorous theory and obtain the conclusions that they seek. But a far more reasonable and general method would be to instead study the shape itself and approximate the datasets by selecting the shape that describes it the "best". This is

[1] Data downloaded from the following link, visited on the 01.08.2024

where topology comes into play, as this gives us the tools and theory to do exactly that, with the help of algebraic topology.

Algebraic topology will help us to qualitatively distinguish the two situations in 1.3, where even on an intuitive level we can see that the difference lies in the number of "blobs" in each figure; or more rigorously in the number of connected components.

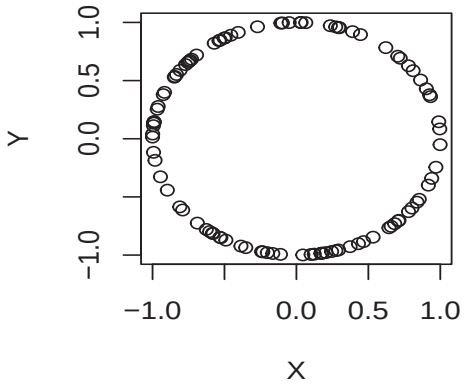


(a) 3 blobs (b) 5 blobs

Figure 1.3: Counting the number of blobs in each figure.

Likewise, using the already established machinery, we will be able to describe and quantify the fact that in 1.4 there is a hole in the middle of it.

Figure 1.4: Data sampled from a unit circle characterized by the distinct hole in the middle.



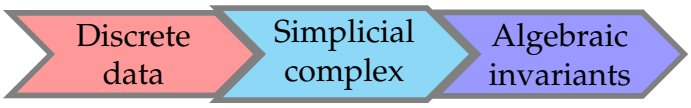
Both of these can be thought of as counting $n-$dimensional holes: connected components being of dimension 0, while the hole in 1.4 is a $1-$dimensional one. We will see how this approach and its extension will be the foundation of what we call TDA – Topological Data Analysis.

Chapter 2

Simplicial Complexes and Homology

The goal of this and the following chapters is to establish and set up the pipeline for extracting the algebraic invariants of our data. Usually, we can only work with sampled and discrete data coming from some set of measurements. As such, we can't directly use methods of algebraic topology, since we won't typically be working with discrete topological spaces and to properly use these methods, we would need an uncountable amount of data; something that isn't feasible from a computational point of view.

This forces us to use different methods to somehow approximate and recover the topology of the ambient space given only a finite set of points. Secondly, we also need to consider the *scale* of the data – some interesting properties may be more apparent only after we "zoom" in closely on them, some may not become apparent at all. All in all, we will construct the following pipeline:



and repeat this step for all scales at once, effectively measuring the evolution of the algebraic invariants through the changes in the feature scale.

2.1 Simplicial complexes

Definition 2.1.1 (Simplex)**.** For $k \geq 0$, a k-simplex σ of dimension k in a Euclidean space $\mathbb{R}^n$ is the convex hull of a set P of $(k+1)$ affinely independent points in $\mathbb{R}^n$. For $0 \leq m \leq k$, an m-face of σ is an m-simplex that is the convex hull of a nonempty subset of P. A *proper face* of σ is a simplex that is the convex hull of a proper subset of P (any face except σ). $(k-1)$ faces of σ are called *facets* of σ.

Typically, we refer to a 0-simplex as a *vertex*, a 1-simplex as an *edge*, a 2-simplex as a *triangle* and so on. An illustration of those can be seen in 2.1.

Definition 2.1.2 (Geometric simplicial complex)**.** A *geometric simplicial complex* K is a set with finitely many simplices that satisfy the following:

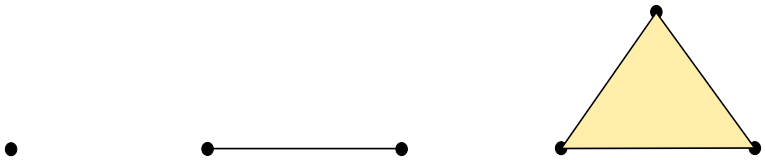


Figure 2.1: From the left: a 0-simplex, a 1-simplex and a 2-simplex

- K contains every face of each simplex in K.
- For any two simplices $\sigma, \tau \in K$, their intersection $\sigma \cap \tau$ is either empty or a face or both σ and τ.

This is also known as a *triangulation*, where the *dimension* k of K is the maximum dimension of any simplex in K. The two definitions above are highly geometric and easy to visualize and imagine. The next definition is more technical and abstract but nonetheless important.

Definition 2.1.3 (Abstract simplex)**.** A collection K of non-empty subsets of a given set $V(K)$ is an *abstract simplicial complex*, if every element $\sigma \in K$ has all of its non-empty subsets $\sigma' \subseteq \sigma$ also in K. Each element σ with a cardinality $|\sigma| = k + 1$ is called a k-simplex and each of its subsets $\sigma' \subseteq \sigma$ with $|\sigma'| = k' + 1$ is called a k'-face. Finally, a $(k-1)$-face of a k-simplex is called its *facet*.

Remark. One could also dually define a k-coface, cofacet and its codimension but it's not terribly important.

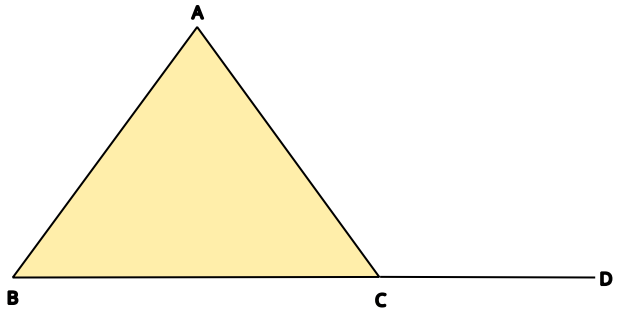


Figure 2.2: A simplicial complex with 4 vertices, 4 edges and 1 triangle.

A geometric simplicial complex K in $\mathbb{R}^n$ is called a *geometric realization* of an abstract simplicial complex K', if and only if there is an embedding $e : V(K') \to \mathbb{R}^n$, that takes every k-simplex $\{v_0, \ldots, v_k\}$ in K' to a k-simplex in K that is the convex hull of $e(v_0), \ldots, e(v_k)$. An example is shown in 2.2 as this is the geometric realization of the abstract complex with vertices A, B, C, D, edges $\{A, B\}, \{A, C\}$, $\{B, C\}, \{C, D\}$ and 1 triangle $\{A, B, C\}$.

Definition 2.1.4 (Underlying space)**.** The *underlying space* of an abstract simplicial complex K, denoted by $|K|$, is the pointwise union of its simplices in its geometrical realization, i.e., $|K| = \bigcup_{\sigma \in K} |\sigma|$, where $|\sigma|$ is the restriction of this realization on σ. If K is geometric, then its geometric realization can be taken as itself.

Unless it is considered necessary, we won't be making the distinction between the two due to this equivalence between geometric and abstract simplicial complexes.

Definition 2.1.5 (k-skeleton)**.** For any $k \geq 0$, the k-skeleton of a simplicial K complex, denoted by K^k, is the subcomplex formed by all simplices of dimension at most k.

Given this, in 2.2, the 1-skeleton consists of the vertices A, B, C, D and the edges joining those.

2.2 Nerves, Čech and Rips complexes

Given any open cover of a topological space, we are able to construct a simplicial complex on top of it. As we'll see, there isn't only one kind of complex we can build, depending on the properties we're looking for and its size, which has to be considered whenever we talk about any software implementation of the algorithms.

Definition 2.2.1 (Nerve)**.** Given a finite collection of sets $\mathfrak{U} = \{U_\alpha\}_{\alpha \in A}$, we define the *nerve* of the set $\mathfrak{U}$ to be the simplicial complex $N(\mathfrak{U})$, whose vertex set is the index set A, and where a subset $\{\alpha_0, \ldots, \alpha_k\} \subseteq A$ spans a k-simplex in $N(\mathfrak{U})$ if and only if $U_{\alpha_0} \cap \ldots \cap U_{\alpha_k} \neq \emptyset$.

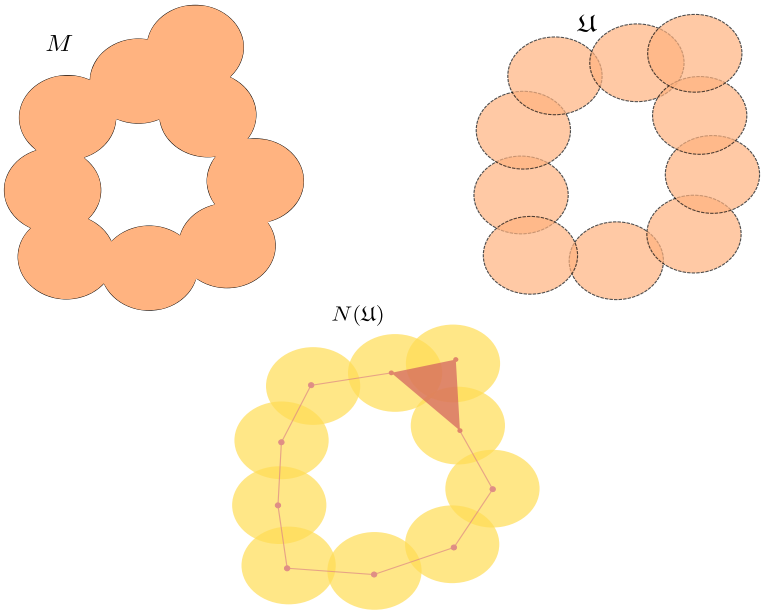


Figure 2.3: An example of a space M, its open cover $\mathfrak{U}$ and its nerve $N(\mathfrak{U})$.

The following important theorem about nerves tells us when are the nerves "equivalent" to the original space. There are various formulation of this statement but since we are working primarily with finite metric spaces, we'll adopt the appropriate version for it.

Theorem 2.2.1 (Nerve theorem)**.** Given a finite cover $\mathfrak{U}$ (open or closed) of a metric space M, the underlying space $|N(\mathfrak{U})|$ is homotopy equivalent to M, if every non-empty intersection $\cap_{i=0}^{k} U_{\alpha_i}$ of cover elements is homotopy equivalent to a point, i.e., contractible.

For those interested in a proof of this statement, see [Bor48] for example. From this we can see, that the nerve is homotopy equivalent to M in 2.3. Now we can finally present the first construction of an abstract simplicial complex using the concept of a nerve, given a finite subset P of a metric space (M, d).

Definition 2.2.2 (Čech complex)**.** Let (M, d) be a metric space and P a finite subset of it. Given a real $\varepsilon > 0$, the Čech complex $\mathbb{C}^{\varepsilon}(P)$ is defined to be the nerve of the set $\{B(p_i, \varepsilon)\}$, where

$$B(p_i, \varepsilon) = \{x \in M \mid d(p_i, x) \leq \varepsilon\}.$$

One can easily deduce that if M happens to be a Euclidean metric space, according to Theorem 2.2.1, the Čech complex will be homotopy equivalent to the space of union of the balls. The Čech complex has nice theoretical properties, but the one predominantly implemented in most software packages is the Vietoris-Rips complex bellow.

Definition 2.2.3 (Vietoris-Rips complex)**.** Let (P, d) be a finite metric space. Given a real $\varepsilon > 0$, the Vietoris-Rips (VR for short) complex is the abstract simplicial complex $\mathbb{VR}^{\varepsilon}(P)$, where a simplex $\sigma \in \mathbb{VR}^{\varepsilon}(P)$, if and only if $d(p, q) \leq 2\varepsilon$ for every pair of vertices of σ.

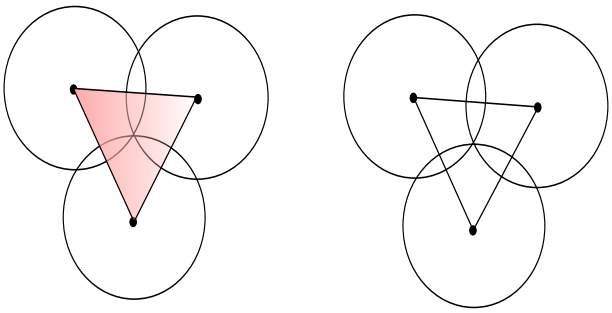


Figure 2.4: On the left, the VR complex of three pairwise intersecting circles. On the right, the corresponding Čech complex.

A simple example, where we can see the difference between the Čech and VR complex can be seen on 2.4. For an interactive visualization of the two, the reader may want to access the following sites - [Sus24] and [Nat24]. With the movement of the sliders there, the reader is introduced to another important concept that we will make use of - changing the radius r and constructing a growing sequence of either Čech or VR complexes.

Nevertheless, we don't have to worry too much about the differences between the two, given the following result.

Theorem 2.2.2. Let P be a finite subset of a metric space (M, d). Then

$$\mathbb{C}^{\varepsilon}(P) \subseteq \mathbb{V}^{\varepsilon}(P) \subseteq \mathbb{C}^{2\varepsilon}(P). \tag{2.1}$$

2.3 Sparse complexes

While the VR complex is the one you will usually encounter in most talks about TDA, it has a size problem. More often than not, both the Čech and VR complexes grow too large, even in low dimensions. A VR complex constructed out of a few thousand points can easily have millions of triangles. A short example of this behaviour can be seen in 2.5. As such, sometimes it is computationally less expensive to use more sparse alternatives instead. [1]

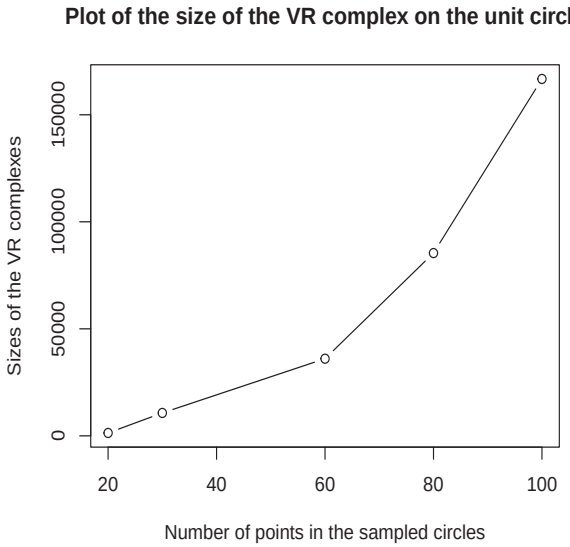


Figure 2.5: An illustration of the growth of the VR complex when we increase the number of sampled points.

2.3.1 Delaunay complex

This complex is usually used in various applications, such as mesh generation or 3D triangulation (for example, see the following). However, it remains computationally expensive in dimensions greater than 3, and other complexes are preferred instead.

Definition 2.3.1 (Delaunay complex)**.** Let P be a finite point set in R^n. A k-simplex σ is called *Delaunay*, if its vertices are in P and there is an open n-ball whose boundary contains the boundary of this ball. A *Delaunay complex* of P, denoted by Del P, is the simplicial complex with vertices in P, in which every simplex is Delaunay and $|$ Del $P|$ coincides with the convex hull of P.

An example of a Delaunay complex can be seen on 2.6. The Delaunay complex is dual to another construction that you may or may not encounter in the wild - the Voronoi diagram.

[1] Technically, the graph is based on the Rips filtration, a term we'll see in the future although it doesn't change the point presented here.

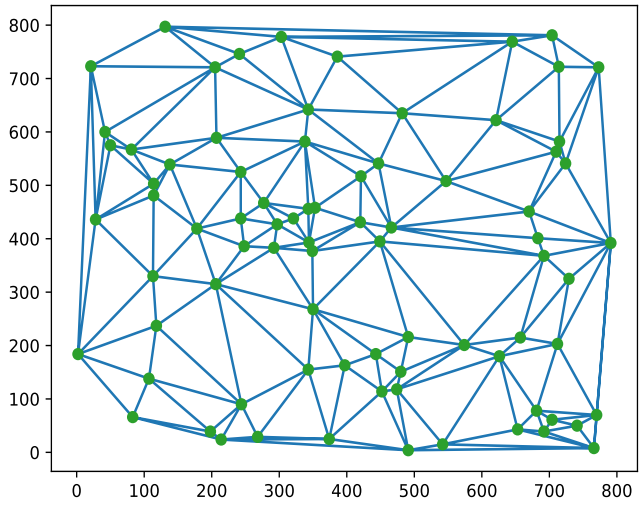


Figure 2.6: Delaunay complex on a sample of randomly generated points.

Definition 2.3.2 (Voronoi diagram)**.** Given a finite point set $P \subset \mathbb{R}^n$ in generic position, the Voronoi diagram Vor(P) of P is the tessellation of the embedding space $\mathbb{R}^n$ into convex cells V_p for every $p \in P$ where

$$V_p = \{x \in \mathbb{R}^n \mid d(x,p) \leq d(x,q), \forall q \in P\}.$$

Additionally, a k-face of Vor(P) is the intersection of $(d-k+1)$ Voronoi cells.

The duality between a Delaunay complex and Voronoi diagram is better expressed through this little theorem.

Theorem 2.3.1. For $P \in\subset \mathbb{R}^n$, Del($P$) is the nerve of the set of Voronoi cells $\{V_p\}_{p\in P}$, which is a closed cover of $\mathbb{R}^n$.

More specifically, a Delaunay k-simplex in Del(P) is dual to a Voronoi $(d-k)$-face in Vor(P). That duality can be seen on 2.7. [2] The reasons why Delaunay complexes are popular in dimensions < 3 are the following

Theorem 2.3.2. A triangulation of a point set $P \subset \mathbb{R}^n$ is a geometric simplicial complex whose vertex set is P and whose simplices tessellate the convex hull of P. Among all triangulations of a point set $P \subset \mathbb{R}^n$, Del(P) achieves the following:

1. In $\mathbb{R}^2$, Del(P) maximizes the minimum angle of triangles in the complex.
2. In $\mathbb{R}^2$, Del(P) minimizes the largest circumcircle for triangles in the complex.
3. For a simplex in Del(P), let its min-ball be the smallest ball that contains the simplex in it. In all dimensions, Del(P) minimizes the largest min-ball.

Unfortunately, the size of a Delaunay complex is $O(n^{\lceil d/2 \rceil})$, with the same cost in computation (see [Cha93]).

[2] Both 2.6 and 2.7 were generated using [VGO+20].

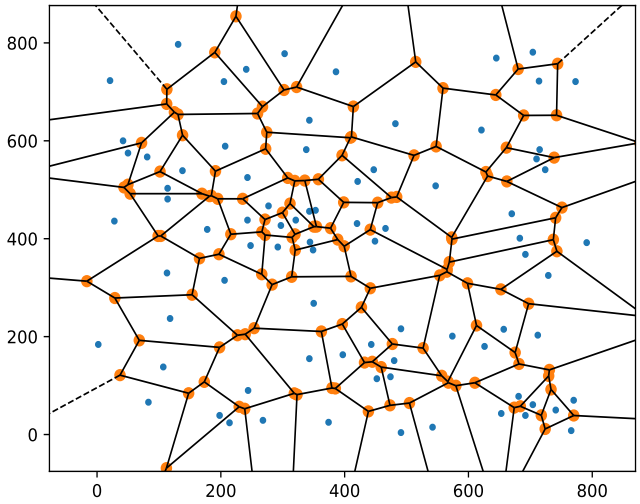


Figure 2.7: Voronoi diagram on a sample of randomly generated points dual to 2.6.

2.3.2 Alpha complex

Alpha complexes are parametrized subcomplexes of the Delaunay complex by some real $\alpha \geq 0$. More specifically, for a given point set P and some $\alpha \geq 0$, an alpha complex consists of all simplices in Del(P) that have a circumscribing ball of radius at most α. An alternative but equivalent definition follows like this - for each point $p \in P$, let $B(p, \alpha)$ be the closed ball of radius α centered at p. Consider the closed set D_p^α be defined as:

$$D_p^\alpha = \{x \in B(p, \alpha) \mid d(x, p) \leq d(x, q), \forall q \in P\}.$$

Then the alpha complex $\text{Del}^\alpha(P)$ is the nerve of the closed sets $\{D_p^\alpha\}_{p \in P}$. We can use the duality of the Delaunay complex to introduce a third definition - an alpha complex contains a k-simplex $\sigma = \{p_0, \ldots, p_k\}$, if and only if $\bigcup_{p \in P} B(p, \alpha)$ meets the intersection of Voronoi cells $V_{p_0} \cup \ldots \cup V_{p_k}$.

2.3.3 Graph induced complex

Lastly, we present here another sparse complex that instead uses subsampling to tackle the size problem that the VR or Čech complexes have, while capturing the topology and even the geometry of the point cloud more efficiently. This construction was introduced in [DFW13].

Definition 2.3.3 (Graph induced complex)**.** Let (P, d) be a metric space, where P is a finite set and $G(P)$ be a graph with vertices in P. Let $Q \subseteq P$ and let $\nu : P \to Q$ the mapping that sets $\nu(p)$ to be any point in argmin $d(p, Q)$. The *graph induced complex* (GIC) $\mathbb{G}(G(P), Q, d)$ is the simplicial complex containing a k-simplex $\sigma =$

$\{q_1, \ldots, q_{k+1}\}$, $q_i \in Q$, if and only if there exists a $(k+1)$-clique in $G(P)$ spanned by vertices $\{p_1, \ldots, p_{k+1}\} \subseteq P$ so that $q_i \in \nu(p_i)$ for each $i \in \{1, 2, \ldots, k+1\}$.

For the input graph $G(P)$ we may consider the neighborhood graph $G^\alpha(P) := (P, E)$, where there is an edge $\{p, q\} \in E$, if and only if $d(p, q) \leq \alpha$. If P is sufficiently dense, then $G^\alpha(P)$ should capture the local neighborhoods of the sample points.

In the following, the quality of the sampled space after subsampling with Q will be quantified with a parameter $\delta > 0$.

Definition 2.3.4. A subset $Q \subseteq P$ is called a δ-sample of a metric space (P, d), if the following condition holds:

- $\forall p \in P$, there exists a $q \in Q$, so that $d(p, q) \leq \delta$.

It is called δ-sparse, if the previous and next condition hold together:

- $\forall (q, r) \in Q \times Q$ with $q \neq r$, $d(q, r) \geq \delta$.

The metric itself is usually taken to be either the Euclidean metric or the graph metric d_G derived from the input graph $G(P)$. In those two cases we have several existence and inference results involving the GIC; see [DFW13] again for that. The paper also includes an empirical comparison of the GIC with the VR complex together with another sparse complex, the Witness complex, which we won't discuss in this thesis. The simulations show that the GIC can have a size similar to the Witness complex (which is usually smaller but also fails to capture the topology more because of it) and maintains the accuracy of the Rips complex.

2.4 Chains, cycles and homology

We make the daring assumption that the reader has some knowledge of algebra, group theory and topology in order to understand the next part. If not, maybe an appendix will be written in the future, who knows?

2.4.1 Chains

If K is a simplicial k-complex with a m_p number of p-simplices, then a p-chain c in K is simply the formal sum of p-simplices multiplied by some coefficients from a ring R. Addition of p-chains is defined in the following natural way: if $c = \sum \alpha_i \sigma_i$ and $c' = \sum \alpha'_i \sigma_i$, then

$$c + c' = \sum_{i=1}^{m_p} (\alpha_i + \alpha'_i) \sigma_i.$$

Generally speaking, the multiplication by coefficients from R turn chains into an R-module. In the context of this work, we will only consider coefficients coming from some field $\boldsymbol{k}$, more specifically from $\boldsymbol{k} = \mathbb{Z}_2$. This is computationally convenient because it turns chain addition into, for example in the case of 1-chains,

$$(e_1 + e_2 + e_3) + (e_1 + e_2 + e_4) = e_3 + e_4,$$

since in $\mathbb{Z}_2$ we have that $0 + 0 = 0, 0 + 1 = 1$ and $1 + 1 = 0$. This also implies that

$$c + c = \sum_{i=1}^{m_p} = (\alpha_i + \alpha_i)\sigma_i = 0\sigma_i = 0,$$

i.e., p-chains with $\mathbb{Z}_2$ addition form a group, the identity here is the chain $0 = \sum_{i=1}^{m_p} 0\sigma_i$, and the inverse of c is c itself due to the above. We call this group the *p-th chain group* $C_p(K)$.

2.4.2 Boundary operator and cycles

Since our goal is to study the changes in topology across different scales, we need a way to connect higher dimensional chains with lower dimensional ones. For that, we introduce the boundary operator: Given a p-simplex $\sigma = \{v_0, \ldots, v_p\}$, let

$$\partial_p \sigma = \sum_{i=0}^{p} \{v_0, \ldots, \hat{v}_i, \ldots, v_p\},$$

where $\hat{v}_i$ means that this vertex is omitted. Extending this definition to a p-chain, we get the boundary operator homomorphism $\partial_p : C_p \to C_{p-1}$ that returns a $(p-1)$-chain when given a p-chain in the following manner

$$\partial_p c = \sum_{i=1}^{m_p} \alpha_i(\partial_p \sigma_i), \text{ for } c = \sum_{i=1}^{m_p} \alpha_i \sigma_i \in C_p.$$

A couple edge cases have to be considered. For $p = 0$, we get that $\partial_0 c = \emptyset$. The chain group C_{-1} has only one element 0. And if K is a k-complex, then $C_p = 0$ for $p > k$. In general, most textbooks and articles will present the form of the boundary operator as follows:

$$\partial_p \sigma = \sum_{i=0}^{p} (-1)^i \{v_0, \ldots, \hat{v}_i, \ldots, v_p\},$$

which differs from our choice by the factor of $(-1)^i$. But remember that we're working in $\mathbb{Z}_2$, where the multiplicative factor becomes unnecessary due to the addition laws. As such, the boundary operator applied to the 2-chain $\{a, b, c\}$ gives us

$$\partial_2(abc) = ab + bc + ca,$$

instead of $ab - bc + ca$ as you may encounter in other texts.

Applying the boundary operator twice in a row turns out to produce the empty chain.

Lemma 2.4.1. For $p > 0$ and any p-chain c, $\partial_{p-1} \circ \partial_p(c) = 0$.

Proof. Trivial; can be found in any textbook about algebraic topology, for example [Hat02]. □

By extending the boundary operator now to chain groups, we can construct a sequence called a *chain complex*:

$$0 = C_{k+1} \xrightarrow{\partial_{k+1}} C_k \xrightarrow{\partial_k} \cdots \longrightarrow \cdots \longrightarrow C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} C_{-1} = 0.$$

It can be worthwhile to mention that for $p \geq -1$, each of the C_p is a vector space.

2.4.3 Cycle and boundary groups

Just like with the chain groups, we can identify two other natural groups of interests - the *cycle* and *boundary* groups. We start by defining what a cycle is - a p-chain c is a p-cycle if $\partial c = 0$, i.e., a chain with an empty boundary.

The p-cycles form the *p-th cycle group* Z_p with the same chain addition operation inherited from chain groups. From the above, we can see that $\ker \partial_p = Z_p$.

We saw that after applying the boundary operator to the 2-chain $\{a, b, c\}$, the result was

$$\partial_2(abc) = ab + bc + ca.$$

This 1-chain is a 1-cycle since

$$\partial_1(ab + bc + ca) = (a + b) + (b + c) + (c + a) = 0.$$

It's not difficult to conclude from this that the boundary of a p-chain is a $(p-1)$-cycle. This begs the question - which $(p-1)$-chains can be obtained this way? We call the set of all $(p-1)$-chains that are the result of an application of the boundary operator ∂_p on p-chains the *$(p-1)$-th boundary group* $B_{p-1} = \partial_p(C_p)$. This is equivalent to saying that $B_{p-1} = \text{im } \partial_p$. It is left as an exercise for the reader to check that $\partial_{p-1} B_{p-1} = 0$ for all $p > 0$, which implies that $B_{p-1} \subseteq Z_{p-1}$.

Theorem 2.4.1. For a simplicial k-complex,

- $C_0 = Z_0$ and $B_k = 0$.
- For $p \geq 0$, $B_p \subseteq Z_p \subseteq C_p$.
- Both B_p and Z_p are vector spaces.

2.4.4 Homology

We can finally define the most important notion in this work - homology. Homology is primarily used to quantify and measure the failure of a cycle to be a boundary by putting cycles that differ by a boundary in the same equivalence class.

Definition 2.4.1 (Homology group)**.** For $p \geq 0$, the *p-th homology group* is the quotient group $H_p = Z_p/B_p$. Since we're using $\mathbb{Z}_2$ as our coefficient field, H_p is a vector space, where we call its dimension the p-th Betti number

$$\beta_p := \text{dim } H_p.$$

The equivalence classes are called *homology classes* in this context. Two cycles $c, c' \in Z_p$ are in the same homology class, iff $c \in c' + B_p$, which under $\mathbb{Z}_2$ coefficients translates to $c + c' \in B_p$.

We have the following useful statements with our choice of field coefficients:

Theorem 2.4.2. For $p \geq 0$,

- H_P is a vector space when defined over $\mathbb{Z}_2$.
- The Betti number, $\beta_p = \text{dim } H_p$, is given by $\beta_p = \text{dim } Z_p - \text{dim } B_p$.
- There are exactly 2^{β_p} homology classes in H_p when defined over $\mathbb{Z}_2$.

Generally speaking, H_p isn't always a vector space over $\mathbb{Z}$ as there might be torsion subgroups.

Now what is homology useful for, in the context of statistics and data analysis? The homology group $H_n(X)$ of some topological space, more specifically its Betti number, tells us the number of "holes" with a n-dimensional boundary in our data. For example, a 0-dimensional hole is merely the gap between any two components. It implies then that $H_0(X)$ desribes the path-connected components in X. This turns out to be an strong, invariant characteristic of the dataset that we work with and allows us to discover high-dimensional structures that normally we wouldn't be able to notice.

A few informal examples may be useful to the reader; the proofs of which use more complicated techniques such as the Mayer-Vietoris sequence and so they will be omitted since this isn't an algebraic topology textbook. It isn't hard to see that S^1, the one-dimensional sphere, has only 1 path-connected component and 1 one-dimensional hole in the middle. We can then describe its homology groups as

$$H_k(S^1) = \begin{cases} \mathbb{Z} & k = 0, 1 \\ \{0\} & \text{otherwise} \end{cases}$$

The corresponding Betti numbers here would be $\beta_0 = \beta_1 = 1$ and $\beta_k = 0, \forall k > 1$. This aligns with our intuition that a circle has a hole in the middle, one boundary line trapping said hole and no other high-dimensional structure. The argument can be extended for the general n-sphere to produce

$$H_k(S^n) = \begin{cases} \mathbb{Z} & k = 0, n \\ \{0\} & \text{otherwise} \end{cases}$$

With a little more effort and imagination, it isn't hard to see that the homology groups for the 2-dimensional torus T^2 are

$$H_k(T^2) = \begin{cases} \mathbb{Z} & k = 0, 2 \\ \mathbb{Z} \times \mathbb{Z} & k = 1 \\ \{0\} & \text{otherwise} \end{cases}$$

where the two 1-dimensional holes describe the circles around the width and length of the torus. Once again, in general the situation is

$$H_k(T^n) = \begin{cases} \mathbb{Z}^{\binom{n}{k}} & 0 \leq k \leq n \\ \{0\} & \text{otherwise} \end{cases}$$

2.4.5 Induced homology

To have a better understanding of homology, we have to know how it behaves if we move from one topological space to another. In our case, we will look into simplicial complexes and simplicial maps between them since we can only ever approximate a continuous space. Simplicial maps take cycles to cycles and boundaries to boundaries, which suggests the following definition.

Definition 2.4.2 (Chain map)**.** Let $f : K_1 \to K_2$ be a simplicial map. The chain map $f_\# : C_p(K_1) \to C_p(K_2)$ corresponding to f is defined as follows. If $c = \sum \alpha_i \sigma_i$ is a p-chain, then $f_\#(c) = \sum \alpha_i \tau_i$, where

$$\tau_i = \begin{cases} f(\sigma_i), & \text{if } f(\sigma_i) \text{ is a } p\text{-simplex in } K_2 \\ 0 & \text{otherwise.} \end{cases}$$

Lemma 2.4.2. Let $f : K_1 \to K_2$ a simplicial map. Let $\partial_p^{K_1}$ and $\partial_p^{K_2}$ denote the boundary homomorphisms in dimension $p \geq 0$. Then, the induced chain maps commute with the boundary homomorphisms, i.e., $f_\# \circ \partial_p^{K_1} = \partial_p^{K_2} \circ f_\#$.

This result can be better expressed via the commutativity of the diagram below:

$$\begin{array}{ccc} C_p(K_1) & \xrightarrow{f_\#} & C_p(K_2) \\ \downarrow{\scriptstyle \partial_p^{K_1}} & & \downarrow{\scriptstyle \partial_p^{K_2}} \\ C_{p-1}(K_1) & \xrightarrow[f_\#]{} & C_{p-1}(K_2) \end{array}$$

Essentially, if we start from the top left, the path that we choose doesn't matter because we'll reach the same chain in the end. From the above, since $B_p(K_1) \subseteq Z_p(K_1)$, we see that $f_\#(B_p(K_1)) \subseteq f_\#(Z_p(K_1))$, which allows us to have a well defined induced map in the quotient space

$$f_*(Z_p(K_1)/B_p(K_1)) := f_\#(Z_p(K_1))/f_\#(B_p(K_1)).$$

From the commutativity of the diagram, $f_\#(Z_p(K_1)) \subseteq Z_p(K_2)$, likewise for $B_p(K_1)$, which then induces a homomorphism between the homology groups

$$f_* : Z_p(K_1)/B_p(K_1) \to Z_p(K_2)/B_p(K_2),$$

or $f_* : H_p(K_1) \to H_p(K_2)$. We can now state the desired result.

Lemma 2.4.3. For two contiguous maps $f_1 : K_1 \to K_2$ and $f_2 : K_1 \to K_2$, the induced maps $f_{1_*} : H_p(K_1) \to H_p(K_2)$ and $f_{2_*} : H_p(K_1) \to H_p(K_2)$ are equal.

2.4.6 Cohomology

Cohomology is the dual concept of homology. While in general cohomology has a richer structure and is a finer invariant than homology, in our case we won't explore the differences between the two as most of the popular software uses homology for their calculations. As usual, we will work with coefficients in $\mathbb{Z}_2$.

A p-cochain is a homomorphism $\phi : C_p \to \mathbb{Z}_2$ from the chain group to the coefficient ring $\mathbb{Z}_2$. The cochain is determined by its values on every p-simplex $\sigma \in K$, i.e., a p-chain $c = \sum_{i=1}^{m_p} \alpha_i \sigma_i$ assigns a value

$$\phi(c) = \alpha_1 \phi(\sigma_1) + \ldots + \alpha_{m_p} \phi(\sigma_{m_p})$$

that is either 0 or 1. It isn't hard to verify that $\phi(c + c') = \phi(c) + \phi(c')$. The cochain that gives the value 1 to a simplex is called its dual cochain c^*. p-cochains form the cochain group C^p dual to C_p with addition defined by $(\phi + \phi')(c) = \phi(c) + \phi'(c)$ and scalar multiplication $(\alpha\phi)(c) = \alpha\phi(c)$, where we used $\mathbb{Z}_2$ addition and multiplication. Hence, C^p is a vector space.

Similarly to chain boundaries, there is the notion of a cochain coboundary $\delta_p : C^p \to C^{p+1}$. For a p-cochain ϕ, its $(p+1)$-coboundary is given by the homomorphism $\delta\phi : C^{p+1} \to \mathbb{Z}_2$ defined as $\delta\phi(c) = \phi(\partial c)$ for any $(p+1)$-chain c. Repeated iteration of the δ operator gives us the sequence

$$0 = C^{-1} \xrightarrow{\delta_{-1}} C^0 \xrightarrow{\delta_0} \ldots \xrightarrow{\delta_{k-1}} C^k \xrightarrow{\delta_k} C^{k+1} = 0.$$

p-coboundaries form the coboundary group and vector space B^p with addition and scalar multiplication being the same as in C^p. Finally, we only need cocycles to define what the cohomology group is. A p-cochain ϕ is called a p-cocycle if its coboundary $\delta\phi$ is a zero homomorphism. The set of p-cocycles forms the group and vector space Z^p with the operations induced by C^p once again.

The coboundary operator δ also satisfies the same property as the boundary ∂.

Lemma 2.4.4. For $p > 0$, $\delta_p \circ \delta_{p-1} = 0$, which implies $B^p \subseteq Z^p$.

Definition 2.4.3 (Cohomology group)**.** Since B^p is a subgroup of Z^p, the p-th cohomology group is the quotient group Z^p/B^p.

Just like for the homology groups, a simplicial map $f : K_1 \to K_2$ induces a homomorphism f^* between cohomology groups; this time in the *opposite* direction.

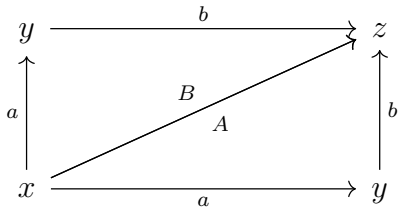
Lemma 2.4.5. A simplicial map $f : K_1 \to K_2$ induces a homomorphism $f^* : H^p(K_2) \to H^p(K_1)$ for every $p \geq 0$.

2.5 Practical questions

Now that we have a basic rundown of what simplicial complexes and homology are, we can move towards more practical questions. How do we actually compute the homology of an object? What is the available software? And the most important question, since we always have to work with discrete samples from some given space - can we accurately approximate the homology of a space given enough sample points?

2.5.1 Calculating homology

We can start by constructing a simplicial complex homeomorphic to the $2-$sphere S^2 as follows



In this relatively simple example, its chain complex is characterized by

$$\cdots \longrightarrow 0 \xrightarrow{\partial_3} \langle A, B\rangle \xrightarrow{\partial_2} \langle a, b, c\rangle \xrightarrow{\partial_1} \langle x, y, z\rangle \xrightarrow{\partial_0} 0$$

Our goal will be to compure the homology group H_0 of this simplicial complex, and by extension, of S^2. We already know that $\ker \partial_0\langle x, y, z\rangle$ is $\langle x, y, z\rangle$ itself. Now we need to figure out what Im ∂_1 is. For that, the behaviour of ∂_1 on any $1-$chain $la + mb + nc$ is needed.

Following the properties of the maps ∂_i, we have that

$$\begin{aligned}\partial_1(la + mb + nc) &= l\partial_1(a) + m\partial_1(b) + n\partial_1(c)\\ &= l(y - x) + m(z - y) + n(z - x)\\ &= (-l - n)x + (l - m)y + (m + n)z.\end{aligned}$$

We can represent this chain with a $3-$tuple $[l, m, n]^T \in \mathbb{Z}^3$ since this is a finitely generated abelian group of rank 3. Writing the equation above as a tranformation

$$[l, m, n]^T \to [-l - n, l - m, m + n]^T,$$

allows us to represent the effect of ∂_1 as a matrix

$$[\partial_1] = \begin{pmatrix} -1 & 0 & -1 \\ 1 & -1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Elementary row-reduction methods yield

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

From this matrix, we can easily find a basis of Im ∂_1

$$[-1, 1, 0]^T, [0, -1, 1]^T,$$

which corresponds to $(y - x)$ and $(z - y)$. Computing H_0 now amounts to computing the quotient

$$H_0 = \langle x, y, z\rangle / \langle y - x, z - y\rangle.$$

We can identify $(y - x)$ and $(z - y)$, which forces the quotient homomorphism q : Ker $\partial_1 \to H_0$ to behave as follows:

$$q(y - x) = q(y) - q(x) = 0 \Longrightarrow q(y) = q(x)$$
$$q(z - y) = q(z) - q(y) = 0 \Longrightarrow q(z) = q(y)$$

To obtain the quotient, we first swap y by x:

$$H_0 = \langle x, x, z\rangle / \langle 0, z - x\rangle = \langle x, z\rangle / \langle z - x\rangle$$

Then we swap z with y:

$$H_0 = \langle x, x\rangle / 0 = \langle x\rangle$$

Therefore $H_0 \cong Z$ and the $2-$sphere S^2 has only one connected component - itself. The essence of this example lies in reducing the calculation of homology groups to matrix manipulations. Typically, a homology solver will make use of the Smith normal form of a matrix.

Definition 2.5.1 (Smith normal form)**.** For any $m \times n$ matrix A, there exists an invertible $n \times n$ matrix U and an invertible $m \times m$ matrix V such that UAV is

equal to a matrix of the form

$$D = \begin{bmatrix} \alpha_1 & 0 & 0 & & \dots & 0 & \\ 0 & \alpha_2 & 0 & & & 0 & \\ 0 & 0 & \ddots & & & & \\ \vdots & & & \alpha_r & & & \\ & & & & 0 & & \\ 0 & & & & & \ddots & \\ & & & & & & 0 \end{bmatrix}$$

Furthemore, if A is an integer matrix, then U, V, D are all integer matrices and the α_i satisfy $\alpha_i | \alpha_{i+1}$ for all $1 \leq i < r$.

For the computation of the homology groups, we use the following theorem that we won't prove here.

Theorem 2.5.1. Let A be a $m \times n$ integer matrix, B an $l \times m$ integer matrix such that $BA = 0$. Then

$$Ker(B)/Im(A) = \bigoplus_{i=1}^{r} \mathbb{Z}/\alpha_i \oplus \mathbb{Z}^{m-r-s},$$

where $r = rank(A), s = rank(B)$ and $\alpha_1, \dots, \alpha_r$ are the non-zero elements on the diagonal of the Smith normal form of A.

Most homology solvers make use of additional optimization steps before resorting to matrix algebra. Libraries such as Linbox [lin24] (written in C++) can be used to compute the Smith normal form. The library Perseus [Vid] (written in C++) computes both "standard" and persistent homology, which we will discuss in later chapters. Kenzo [J. 99] (written in Common Lisp) can additionaly give presentations of homology groups, implementing methods of so-called *Constructive Algebraic Topology*. Gmsh [GR09] provides homology solvers for finite element meshes. There are many more open-source libraries in the wild, some of which we'll use in our applied examples.

But what about the stability and precision of this process? In the example we presented, there wasn't anything to worry about. On the other hand, what we will usually end up working with are simplicial complexes generated from sample data; the Čech and Vietoris-Rips complex, for example. Not only do we have to deal with errors of measurements and noise in our data but also the failure of our generated complex to match the "true" topological space where our data lives. The more precise formulation of this problem can be stated as: Let (P, d) be a finite metric space of samples from a topological space A. When is $|\mathbb{VR}^{\varepsilon}(P, d)|$ or $|\mathbb{C}^{\varepsilon}(P, d)|$ homotopy equivalent to A?

We will attack the idea of *geometric sampling* in later chapters as well, specifically, when we will introduce the idea of statistical testing and estimation while

working with homology groups. For now, we can settle on a basic, minimal guarantee result known as the *Niyogi-Smale-Weinberger theorem*. The idea of the theorem is that if we have sufficiently many points sample uniformly from a compact Riemannian manifold $M \subset \mathbb{R}^n$, there exists an isomorphism

$$H_*\left(\bigcup_{x\in M} B_\varepsilon(x)\right) \cong H_*(M)$$

with high probability for a suitable choice of ε. For a manifold $M \subset \mathbb{R}^n$, the choice of ε is influenced by two factors – its intrinsic curvature and how twisted the embedding into $\mathbb{R}^n$ is. The quantity used to encode the appropriate interval of ε values is called the *condition number*. The condition number is more precisely defined as the minimum radius at which the tubular neighbourhood of a manifold self-intersects.

Theorem 2.5.2 (Niyogi-Smale-Weinberger theorem)**.** Let M be a compact submanifold of $\mathbb{R}^n$ with condition number τ and let $\{x_1, \ldots, x_k\}$ be a set of points drawn from M according to the volume measure. Fix $0 < \varepsilon < \tau/2$. Then if

$$k > \beta_1 \left(\log(\beta_2) + \log(1/\delta)\right),$$

there is a homotopy equivalence

$$\left(\bigcup_{z\in\{x_1,\ldots,x_k\}} B_\varepsilon(z)\right) \cong M$$

between the union of balls and M with probability $> 1 - \delta$ and the homology groups coincide. Here

$$\beta_1 = \frac{\text{vol}\,(M)}{\cos^n(\theta_1)\text{vol}\,(B^n_{\varepsilon/4})}$$

and

$$\beta_2 = \frac{\text{vol}\,(M)}{\cos^n(\theta_2)\text{vol}\,(B^n_{\varepsilon/8})}$$

where $\theta_1 = \arcsin(\frac{\varepsilon}{8\tau}), \theta_2 = \arcsin(\frac{\varepsilon}{16\tau})$ and vol (B^n_r) denotes the volume of the $n-$dimensional ball of radius r.

For example, the condition number of a sphere is its radius. For the unit $1-$sphere $S^1 \subset \mathbb{R}^2$, $\delta = 0.01$ and $\varepsilon = \frac{1}{4}$, we obtain

$$\beta_1 = 512, \quad \beta_2 = 2048,$$

meaning that the minimum number of sample required is

$$512(7.6 + 4.6) \sim 6260.$$

While the result is theoretically important, its practical use is limited. The estimation of the condition number is usually difficult or simply impossible, and the obtained bound is too large, as the above example demonstrates. As such, we will rely on other methods and results that are computationally feasible while lowering the number of samples needed.

Chapter 3

Topological Persistence and Persistent Homology

The *Niyogi-Smale-Weinberger* theorem tells us that it is possible to hope to recover the underlying topology and structure of the space from a limited number of samples from some experiment or study when the distance between the samples is smaller than what is referred to as its *feature scale*.

The problem now lies in finding this feature scale of the object, since this is usually unknowable in advance outside of simulations. Simultaneously, limiting ourselves to looking for a single value of ε is not the right path either. Some features of the object may become visible for different values of ε, i.e., regions may connect at smaller distances and holes may be created at larger ones. Neither of the two is intrinsically more valuable than the other without additional context. As such, this should motivate us to instead look at multiple feature scales at once and observe the changes in the homology groups for a range of values of ε. Features that exist for a wider range of values are more likely to be more informative about the underlying structure than features that only live for short intervals.

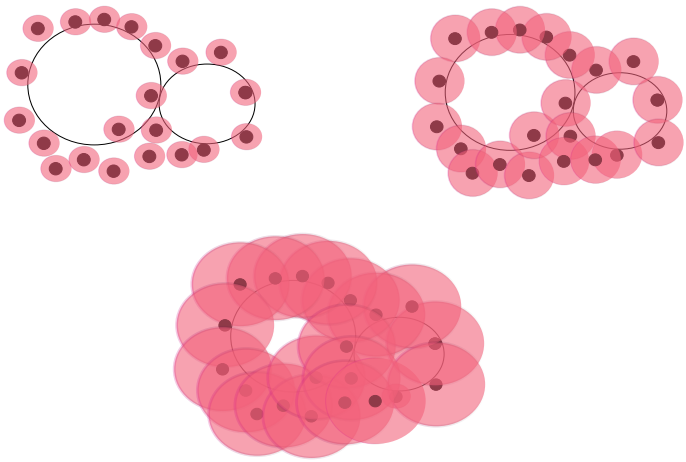


Figure 3.1: A random sample of points from two touching circles and the growing sublevel sets of the distance function to the points.

We can consider the point cloud P in 3.1 sampled from a curve. Ideally, the goal would be to confirm the existence of one connected component, the two circles touching together, and the presence of two holes, i.e., two loops in total. Even better, we would like to see that one loop is larger and more prominent than the other. This is what the main tool of our work captures – *persistent homology*.

If we consider the distance function $r : \mathbb{R}^2 \to \mathbb{R}$, where $r(x) = d(x, P)$, the minimum distance of x to the points in P. Taking a look at the sublevel sets of r, $r^{-1}[-\infty, a]$ for some $a \in \mathbb{R}^+ \cup \{0\}$. In this instance, the sublevel sets are unions of closed balls of radius a with the center being the points. By increasing the a from 0 to infinity, numerous holes will appear, but only the two large loops will live longer than the rest until the entirety of P is covered.

The idea is to translate this rate of survival on the level of points to the survival of individual homology classes whenever we increase the sublevel sets. This is robust since topological "noise" dies out quickly in comparison to the interesting topological features. Persistent homology effectively takes a function defined on the topological space and observes the evolution of homology classes over increasing sublevel sets of the function.

It should be noted that there are two different scenarios where persistence takes place. One is where our function is defined on a topological space, which would require the use of singular homology; previously established to be computationally intractable. The other, the one we are interested in, is the case where the function is defined on a simplicial complex. The sequence of sublevel sets there is simply the nested sequence of subcomplexes, usually referred to as a *filtration*. Naturally, here we will be working with simplicial homology, which we know how to efficiently compute.

3.0.1 Filtration and persistence

Space filtration of topological spaces

Let $f : \mathbb{T} \to \mathbb{R}$ be a function defined on a topological space $\mathbb{T}$. If we denote $\mathbb{T}_a = f^{-1}(-\infty, a]$, the sublevel set of the function f for the value a, then certainly this inclusion holds:

$$\mathbb{T}_a \subseteq \mathbb{T}_b \quad \text{for } a \leq b.$$

The point is to consider a sequence of real numbers $a_1 \leq a_2 \leq \ldots \leq a_n$ to create a sequence of nested subspaces of $\mathbb{T}$

$$\mathcal{F}_f : \varnothing = \mathbb{T}_{a_0} \hookrightarrow \mathbb{T}_{a_1} \hookrightarrow \mathbb{T}_{a_2} \hookrightarrow \ldots \hookrightarrow \mathbb{T}_{a_n}$$

We call this sequence a *filtration* $\mathcal{F}_f$. Ideally, the points $a_1 \leq \ldots \leq a_n$ would be where the homology groups of the sublevel sets go through some change, i.e., new features are born or old ones die. Realistically speaking, we usually cannot hope to guess those values *apriori* and are forced to iterate over a much larger range of admissible values.

The inclusion maps between each sublevel set induce a linear map in the singular homology groups (since we consider general topological spaces). Denoting $\iota : \mathbb{T}_{a_i} \to \mathbb{T}_{a_j}$, $i \leq j$, the inclusion map $x \mapsto x$, we have an induced homomorphism

$$h_p^{i,j} = \iota_* : H_p(\mathbb{T}_{a_i}) \to H_p(\mathbb{T}_{a_j})$$

for all $p \geq 0$, $0 \leq i \leq j \leq n$. Doing this, we transform the nested sublevel sets into a structure we call a *homology module*:

$$0 = H_p(\mathbb{T}_{a_0}) \hookrightarrow H_p(\mathbb{T}_{a_1}) \hookrightarrow H_p(\mathbb{T}_{a_2}) \hookrightarrow \ldots \hookrightarrow H_p(\mathbb{T}_{a_n}).$$

To put it back into perspective, the homomorphism $h_p^{i,j}$ sends the homology classes of $\mathbb{T}_{a_i}$ to those of $\mathbb{T}_{a_j}$. Those that die are precisely the ones that become trivial, while others may survive. This is the information contained in Im $h_p^{i,j}$.

Simplicial filtration of simplicial complexes

To avoid the computation of singular homology groups for each of the sublevel sets, we accept the loss of information that comes with using a discrete model to replace a continuous one. We first replace the topological space with a generated simplicial complex on top of the sampled points. From there, the singular homology groups are naturally replaced with simplicial homology groups. For point cloud data, we replace the continuous union of balls with either their nerve, Čech or Vietoris-Rips complex. With other types of data, the first step is to usually use some transformation to convert the samples into a point cloud of data. We'll see a couple of methods while working with time series or signals, for example.

The discrete analogue of what we described above goes as follows: a filtration $\mathcal{F} = \mathcal{F}(K)$ of a simplicial complex K is a sequence of nested subcomplexes

$$\mathcal{F} : \varnothing = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K,$$

or equivalently

$$\mathcal{F} : \varnothing = K_0 \hookrightarrow K_1 \hookrightarrow \ldots \hookrightarrow K_n = K.$$

The filtration $\mathcal{F}$ is called *simplex-wise* if either $K_i \backslash K_{i-1}$ is empty or a single simplex for every $i \in [1, n]$. Having defined what the simplicial filtration is, we need to place some restrictions on the function f for it to be properly defined.

Given a simplicial complex K and a function $f : K \to \mathbb{R}$. We call f a *simplex-wise monotone* function if for every $\sigma' \subseteq \sigma$, we have $f(\sigma') \leq f(\sigma)$. This guarantees that the sets $f^{-1}(-\infty, a]$ are subcomplexes for every $a \in \mathbb{R}$. Therefore, denoting $K_i = f^{-1}(-\infty, a_i]$ and setting $a_0 = -\infty$, we immediately get the filtration:

$$\varnothing = K_0 \hookrightarrow K_1 \hookrightarrow \ldots \hookrightarrow K_n = K.$$

A *vertex function* $f : V(K) \to \mathbb{R}$ is defined on the vertex set $V(K)$ of the complex K. Once again, a filtration can be constructed from a function like this. Vertex functions are related to what we call piecewise linear functions, or PL-functions.

Definition 3.0.1 (PL-functions)**.** Given a simplicial complex K, a *piecewise-linear* (PL) function $f : |K| \to \mathbb{R}$ is defined to be the linear extension of a vertex function $f_V : V(K) \to \mathbb{R}$ defined on vertices of K, so that for every point $x \in |K|$, $\bar{f}(x) = \sum_{i=1}^{k+1} \alpha_i f_V(v_i)$, where $\sigma = \{v_1, \ldots, v_{k+1}\}$ is the unique lowest dimensional simplex of dimension $k \geq 0$ containing x and $\alpha_1, \ldots, \alpha_{k+1}$ are the barycentric coordinates of x in σ.

PL-functions are closely related to vertex functions since a vertex functions defined a PL function on the underlying space $|K|$ of K by interpolating f over the simplices, while the restriction of a PL function to vertices gives us the corresponding vertex function. Finally, any simplicial filtration $\mathcal{F}$ can be induced by a function as follows:

Definition 3.0.2 (Filtration function)**.** If a simplicial filtration $\mathcal{F}$ is obtained from a simplex-wise monotone function or a vertex function f, then $\mathcal{F}$ is induced by f. If $\mathcal{F}$ is given without any explicit function, we say that $\mathcal{F}$ is induced by the simplex-wise monotone function f, where every simplex $\sigma \in (K_i \backslash K_{i-1})$ for $K_{i-1} \neq K_i$ is given the value $f(\sigma) = i$.

Furthermore, a PL-function $f : |K| \to \mathbb{R}$ naturally provides a vertex function $f_V : V(K) \to \mathbb{R}$.

3.0.2 Persistence

In either of the cases, we obtain at the end a homology module:

$$H_p\mathcal{F} : 0 = H_p(X_0) \to H_p(X_1) \to \ldots \to H_p(X_n) = H_p(X)$$

with the maps $h_p^{i,j} : H_p(X_i) \to H_p(X_j)$ where $X_i = \mathbb{T}_{a_i}$ if $\mathcal{F}$ is a space filtration of a topological space or $X_i = K_i$ if $\mathcal{F}$ is a simplicial filtration of a simplicial complex K.

As already mentioned, we are interested in observing when and where homology classes are born and when they die. We will call homology modules as persistence modules to point out the fact that we can replace homology groups with vector spaces.

Definition 3.0.3 (Persistent Betti numbers)**.** The *p-th persistent homology groups* are the images of the homomorphisms, that is, $H_p^{i,j} = \text{Im } h_p^{i,j}$ for $0 \leq i \leq j \leq n$. The *p-th persistent Betti numbers* are the dimensions $\beta_p^{i,j} = \text{dim } H_p^{i,j}$.

Catching when a class is born or dies requires precision since when one class is born, others that are the sum of this class are also born, and we want to avoid counting duplicates. A similar case holds also for the case when a class dies. One can conclude the following directly from the definitions:

Lemma 3.0.1. $H_p^{i,j} = Z_p(X_i)/(B_p(X_j) \cap Z_p(X_i))$ and $\beta_p^{i,j} = \text{dim } H_p^{i,j}$.

Since $X_i \subseteq X_j$, then $Z_p(X_i)$ is the subgroup of $Z_p(X_j)$ and the above is well-defined. We can now formally define when a class dies and is born.

Definition 3.0.4 (Birth and death)**.** A non-trivial p-th homology class $\xi \in H_p(X_a)$ is born at X_i, $i \leq a$, if $\xi \in H_p^{i,a}$ but $\xi \notin H_p^{i-1,a}$. Conversely, a non-trivial p-th homology class $\xi \in H_p(X_a)$ dies entering X_j, $a < j$, if $h_p^{a,j-1}(\xi)$ is not zero but $h_p^{a,j}(\xi) = 0$.

It is good to point out that not all classes born in some X_i necessarily die entering some X_j later; some classes persist throughout the entire lifetime of the module.

Theorem 3.0.1. Let $[c] \in H_p(X_{j-1})$ be a p-th homology class that dies entering X_j. Then it is born at X_i if and only if there exists a sequence $i_1 \leq i_2 \leq \ldots \leq i_k = i$ for some $k \geq 1$ so that $0 \neq [c_{i_l}] \in H_p(X_{j-1})$ is born at X_{i_l} for every $l \in \{1, \ldots, k\}$ and $[c] = [c_{i_1}] + \ldots + [c_{i_k}]$.

This may be interpreted in the sense that the death of a homological class can be though of as the merging of several classes, the youngest one signaling the birth point. For each $X_i, i = 0, \ldots, n$, there is an association with the value of the function f determining the filtration $\mathcal{F}$. In the case of space filtrations, we have $f(X_i) = a_i$, where $X_i = \mathbb{T}_{a_i}$. For simplicial filtrations, $f(X_i) = a_i$, where $a_i = f(\sigma)$ for any $\sigma \in X_i$ when f is simplex-wise monotone. If f is a vertex function, we know that it can be extended to a simple-wise monotone function.

3.0.3 Persistence diagrams

While recording the birth and death times of individual classes is the building block for all subsequent analysis, on its own, it isn't very descriptive or amenable to more precise statistical analysis. Humans are also visual creatures and the ability to represent the homological module graphically would be beneficial. This visual representation is what we call a *persistence diagram*.

We start by considering the extended plane $\bar{\mathbb{R}}^2 := (\mathbb{R} \cup \{\pm\infty\})^2$. A birth at a_i is paired with its death at a_j as a point (a_i, a_j). For this, we have to define a *persistence pairing function* $\mu_p^{i,j}$, where strictly positive values correspond to multiplicities of points in the persistence diagram. We also have to deal with classses that never die. For this case, we extend the module on the right by declaring $H_p(X_{n+1}) = 0$.

Definition 3.0.5. For $0 < i < j \leq n+1$, define

$$\mu_p^{i,j} = (\beta_p^{i,j-1} - \beta_p^{i,j}) - (\beta_p^{i-1,j-1} - \beta_p^{i-1,j}).$$

The first term on the right counts the independent classes that are born at or X_i and die upon entering X_j. The second term, on the other hand, counts the number of classes that are born at or before X_{i-1} and die upon entering X_j. All in all, $\mu_p^{i,j}$ counts the number of independent classes that are born at X_i and die at X_j.

When $j = n+1$, $\mu_p^{i,n+1}$ counts the classes that remain alive till the end of our filtration, i.e., those that never die. Symbolically, to emphasize it, we equate $n+1$ with ∞, take $a_{n+1} = a_\infty = \infty$. In some libraries, this can be observed directly, as the y-axis will have the infinity symbol attached at the top and the classes that do not die will sit there.

Now we can finally approach one of the most used and well-known methods to represent the results of the filtration - the *persistence diagram*.

Definition 3.0.6 (Class persistence)**.** For $\mu_p^{i,j} \neq 0$, the persistence Pers($[c]$) of a class $[c]$ that is born at X_i and dies at X_j is defined as Pers($[c]$) = $a_j - a_i$. When $j = n + 1 = \infty$, Pers($[c]$) equals $a_{n+1} - a_i = \infty$.

Definition 3.0.7 (Persistence diagram)**.** The persistence diagram $\mathrm{Dgm}_p(\mathcal{F}_f)$ (or $\mathrm{Dgm}_p f$) of a filtration $\mathcal{F}_f$ induced by a function f is obtained by drawing a point (a_i, a_j) with non-zero multiplicity $\mu_p^{i,j}$, $i < j$, on the extended plane $\bar{\mathbb{R}}^2 := (\mathbb{R} \cup \{\pm\infty\})^2$ where the points on the diagonal $\Delta : \{(x, x)\}$ are added with infinite multiplicity.

As an example, we can examine a similar situation to the one in the previous section: We will consider a randomly sampled set of points from a circle and add to that sample some Gaussian noise, see 3.2.

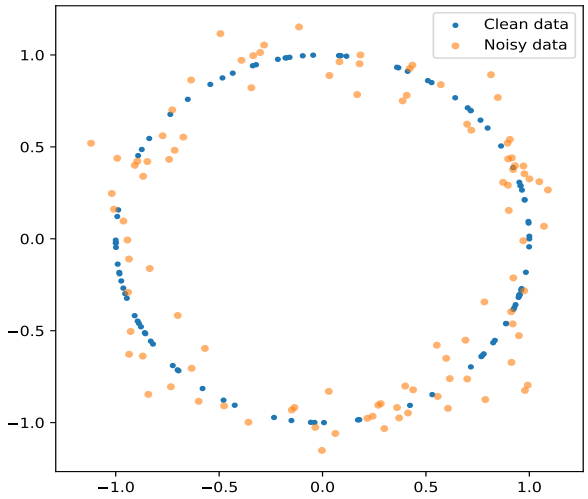
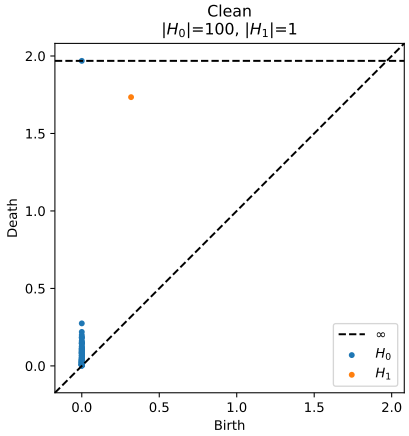


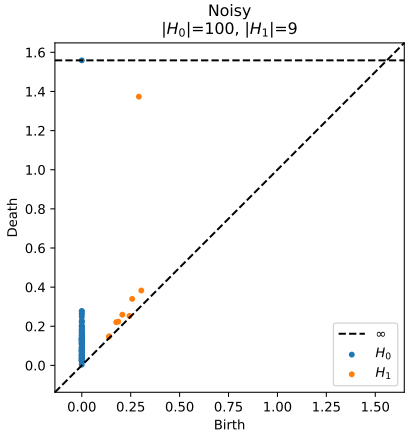
Figure 3.2: A random sample drawn from the unit circle and one drawn from the same circle with additional Gaussian noise.

To compute the persistence diagrams, we will use the *scikit-tda* [ST19] Python library. It follows the *scikit* API design, making it compatible with libraries such as NumPy [HMvdW+20], Matplotlib [Hun07] or other *scikit*-like libraries, for example *scikit-learn* [PVG+11] (general machine learning algorithms), *scikit-image* [VdWSNI+14] (image processing algorithms) or *scikit-fda* [RCTCB+24] (functional data analysis library). In contrast to other libraries that will be used, *scikit-tda* is more high-level, trying to hide the granular details away from the user. The code for the following example can be found in the file *PersistentHomology01.ipynb*.

The resulting persistence diagrams can be seen on 3.3 side by side. In both diagrams, we see a lot of blue dots at the beginning of the filtration, those being



(a) Clean sample



(b) Noisy sample

Figure 3.3: Persistence diagrams of the random "clean" and noisy circle samples. We can see that the essential features are preserved even in the presence of noise.

the points that quickly merge together into the single connected component – the boundary of the circle – and stay this way, its homology class never dying and displayed at the top of the diagram on the infinity line. In general, a class born at a_i and never dying will be represented as a point (a_i, ∞). Some authors call these points *essential persistent points* with *essential homology classes*. The first persistent homology groups inform us about the presence of a one-dimensional hole in both diagrams, characterized by the one orange dot near the top.

In the noisy version there appear multiple smaller orange dots near the diagonal at the start; their presence signalling some sort of topological and stochastic noise since those classes die quickly after being born. It is also worthwhile to point out that the significant orange dot appears on different scales for the clean and noisy circle, marking another difference between the two that would be harder to confirm purely through some statistical testing.

The diagonal is purely a technical addition although it highlights the fact that a class cannot die before it is even born. Some packages prefer to plot in the upper

triangle, some choose the lower one, so be cautious and look at the axes.

One of the many alternative representations of persistence that we will look at and work with are *persistence barcodes*. Each birth-death pair (a_i, a_j) is represented as a line segment $[a_i, a_j)$ called a bar. One such example can be seen on 3.4.

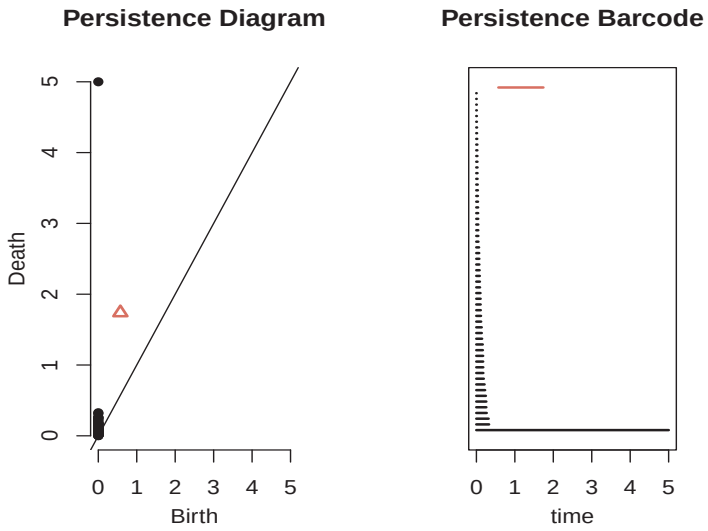


Figure 3.4: The corresponding representation of a persistence barcode next to its persistence diagram.

Keep in mind that the barcodes do not contain more information than their diagram – they are equivalent representations of the same fact. They *can* be used in further analysis in different manners, but they do not fundamentally reveal anything new that wasn't already obtained. We don't care about any particular ordering of the barcodes, but the convention is to sort the bars in order of their birth time.

If we take a step back and look at the construction of the persistence module in the case of $p = 0$, what we are doing, after some thought, corresponds precisely to single-linkage hierarchical clustering that we know from regular statistical and data mining methods. Indeed, we start with our set of points as the individual cluster centers and gradually increase the ball radius, checking for links between the points, which creates the corresponding clusters. Persistence barcodes encode this information in a 1 to 1 correspondence to a dendrogram, as can be seen in 3.5.

Dendrograms will add the "global" cluster to which all the points will belong after a certain height. This is ignored in the barcodes, since no additional homology classes are either born or die after this points. Thus, in 3.5, the barcodes end with time 10, corresponding to the height 10 in the dendrogram.

On a last note, there exists a nice connection between the persistent Betti numbers and persistence diagrams.

Theorem 3.0.2. For every pair of indices $0 \leq k \leq l \leq n$ and every p, the p-th persistent Betti number satisfies $\beta_p^{k,l} = \sum_{i \leq k} \sum_{j > l} \mu_p^{i,j}$.

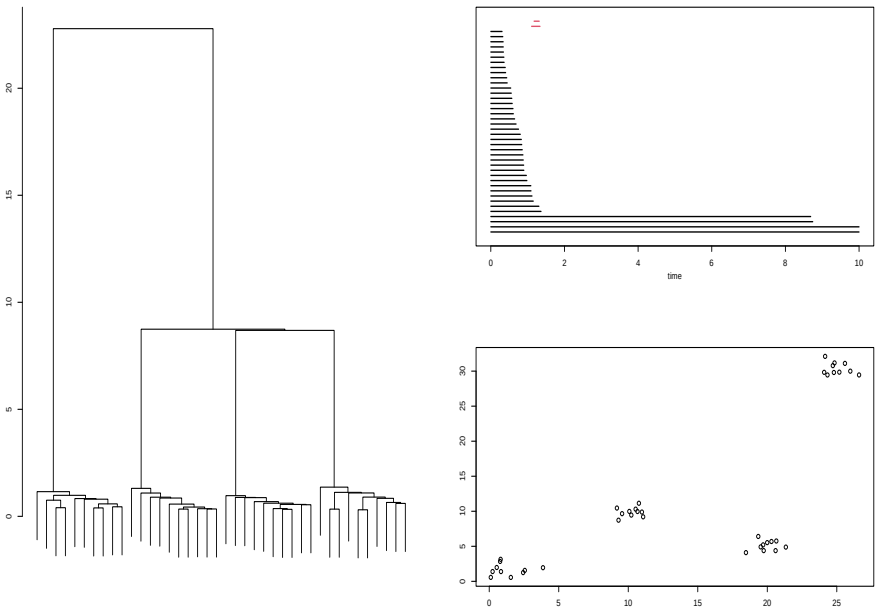


Figure 3.5: The computation of persistence barcodes for $p = 0$ is equivalent to single-linkage clustering of the data. Looking at the count of bars at each time and the dendrogram heights, we see that they keep track of the same information.

Stability of persistence diagrams under perturbation

Given a function f that induces the filtration $\mathcal{F}$, a persistence diagram $\text{Dgm}_p(\mathcal{F}_f)$ summarizes topological features of a simplicial complex generated on a point cloud. While this provides a rough idea of what higher-dimensional structures we might be working with, this is hardly enough to be useful. We would like to compare and test diagrams, pass the results into well-known machine learning algorithms and so forth. For that to be feasible, we require persistence diagrams to be stable under perturbations – in the form of stochastic noise present in measurements but also under slight changes of f itself. This can be achieved because the set of barcodes forms a metric space, allowing us to impose and test various metrics. The important results are that diagrams are fortunately stable in certain metrics that we will now introduce.

Let $\text{Dgm}_p(\mathcal{F}_f)$ and $\text{Dgm}_p(\mathcal{F}_g)$ be two persistence diagrams for two functions f and g. We will consider bijections between the points from the diagram $\text{Dgm}_p(\mathcal{F}_f)$ to $\text{Dgm}_p(\mathcal{F}_g)$. Since the points on the diagonal are all with infinite multiplicity, we can "borrow" those points to define the bijections, if needed.

Definition 3.0.8 (Bottleneck distance)**.** Let $\Pi = \{\pi : \text{Dgm}_p(\mathcal{F}_f) \to \text{Dgm}_p(\mathcal{F}_g)\}$ be the set of all bijections between the persistence diagrams. The bottleneck distance between two diagrams is defined as

$$d_b(\text{Dgm}_p(\mathcal{F}_f), \text{Dgm}_p(\mathcal{F}_g)) = \inf_{x \in \Pi} \sup_{x \in \text{Dgm}_p(\mathcal{F}_f)} ||x - \pi(x)||_\infty.$$

The norm considered here is the usual L^∞ norm. We additionally define $\infty - \infty = 0$ for computational reasons.

It should be noted that d_b is only a metric on the space of diagrams, not necessarily on the space of homology modules $H_p(\mathcal{F})$ as even the first axiom $d_b(X, Y) = 0$ iff $X = Y$ may not hold.

Theorem 3.0.3. Let $f, g : K \to \mathbb{R}$ be two simplex-wise monotone functions giving rise to two simplicial filtrations $\mathcal{F}_f$ and $\mathcal{F}_g$. Then, for every $p \geq 0$,

$$d_b(\text{Dgm}_p(\mathcal{F}_f), \text{Dgm}_p(\mathcal{F}_g)) \leq ||f - g||_\infty.$$

In the case of space filtrations, since we are dealing with a continuous version of the above, we need to impose another condition on the function f. We call a function $f : X \to \mathbb{R}$ *tame*, if the homology groups of the sublevel sets are of finite rank and they change only at finitely many values. We call those values *critical*.

Theorem 3.0.4. Let X be a triangulable space. Let $f, g : X \to \mathbb{R}$ be two tame functions inducing space filtrations $\mathcal{F}_f$ and $\mathcal{F}_g$, where the values for sublevel sets include critical values. Then, for every $p \geq 0$,

$$d_b(\text{Dgm}_p(\mathcal{F}_f), \text{Dgm}_p(\mathcal{F}_g)) \leq ||f - g||_\infty.$$

Another distance that is worth mentioning is the so called q-Wasserstein distance, which compares persistence diagrams.

Definition 3.0.9 (Wasserstein distance)**.** Let Π be the set of bijections described in the definition of the Bottleneck metric. For any $p \geq 0, q \geq 1$, the q-Wasserstein distance is defined as

$$d_{W,q}(\text{Dgm}_p(\mathcal{F}_f), \text{Dgm}_p(\mathcal{F}_g)) = \inf_{\pi \in \Pi} \left[\sum_{x \in \text{Dmg}_{p(\mathcal{F}_f)}} (||x - \pi(x)||_q)^q \right]^{1/q}.$$

In comparison to the Bottleneck metric, the Wasserstein metric has a slightly weaker stability property.

Theorem 3.0.5. Let $f, g : X \to \mathbb{R}$ be two Lipschitz functions on a triangulable compact metric space X. Then there exist constants C and k depending on X and the Lipschitz constants of f and g so that for every $p \geq 0, q \geq k$,

$$d_{W,q}(\text{Dgm}_p(\mathcal{F}_f), \text{Dgm}_p(\mathcal{F}_g)) \leq C||f - g||_\infty^{1-k/q}.$$

Although we don't have to panic as the situation is improved when we consider the L^q metric instead, recently proven in [ST20]:

Theorem 3.0.6. Let $f, g : K \to \mathbb{R}$ be two simplex-wise monotone functions on a simplicial complex K. Then we have

$$d_{W,q}(\text{Dgm}_p(\mathcal{F}_f), \text{Dgm}_p(\mathcal{F}_g)) \leq ||f - g||_q,$$

where we define

$$||f - g||_q = (\sum_{\sigma \in K} |f(\sigma) - g(\sigma)|^q)^{1/q},$$

assuming that both f and g are defined on the same complex K.

We won't go into much detail about the actual computation of either the bottleneck or Wasserstein distances. The main takeaway should be that Wasserstein distances are more expensive to compute, since we essentially have to find a minimum weight perfect matching in weighted bipartite graphs. Bottleneck distances, on the other hand, are cheaper since we're only considering perfetch matchings in bipartite graphs.

To demonstrate some of the notions we have introduced, we will look at a couple simulations and compare their persistence diagrams, barcodes and mutual distances.

While persistence diagrams, and by extension persistence barcodes, are stable under perturbations, there is a natural limit to this robustness. For example, outliers are known to generally skew the picture of the dataset and this holds true here as well. A simple example is in 3.6.

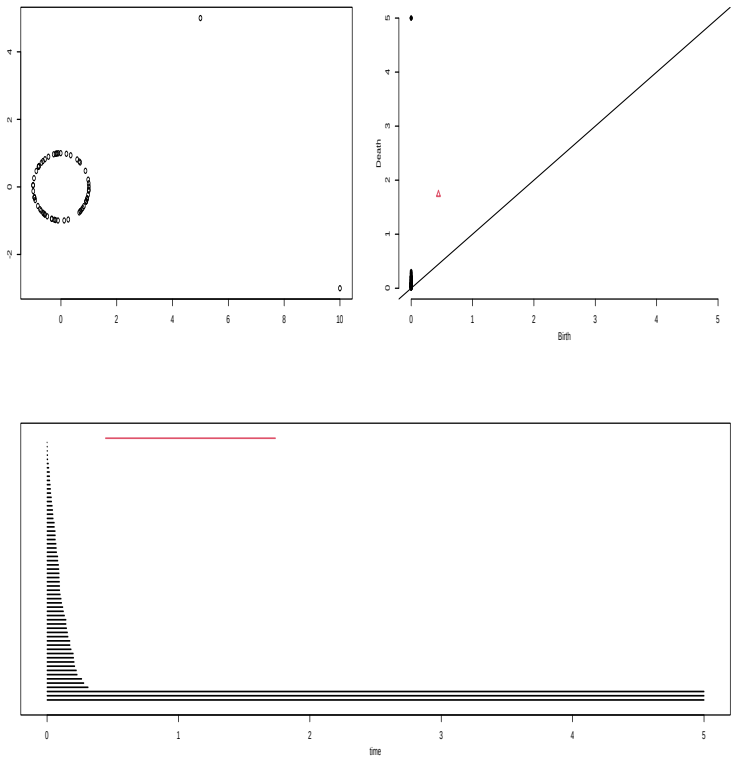


Figure 3.6: Adding two additional points far away from the rest is correctly represented in the barcodes with three long bars representing three connected components. This fact is not as easily seen in the diagram.

The typical solution to this problem is to do a lot of preprocessing of the data *before* applying either topological or statistical methods. Exploratory data analysis, different location or qualitative summaries can be used to filter out outliers. Similarly, nested sub-structures can be harder to identify, since multiple short bars will be generated in their presence. We can see this in 3.7.

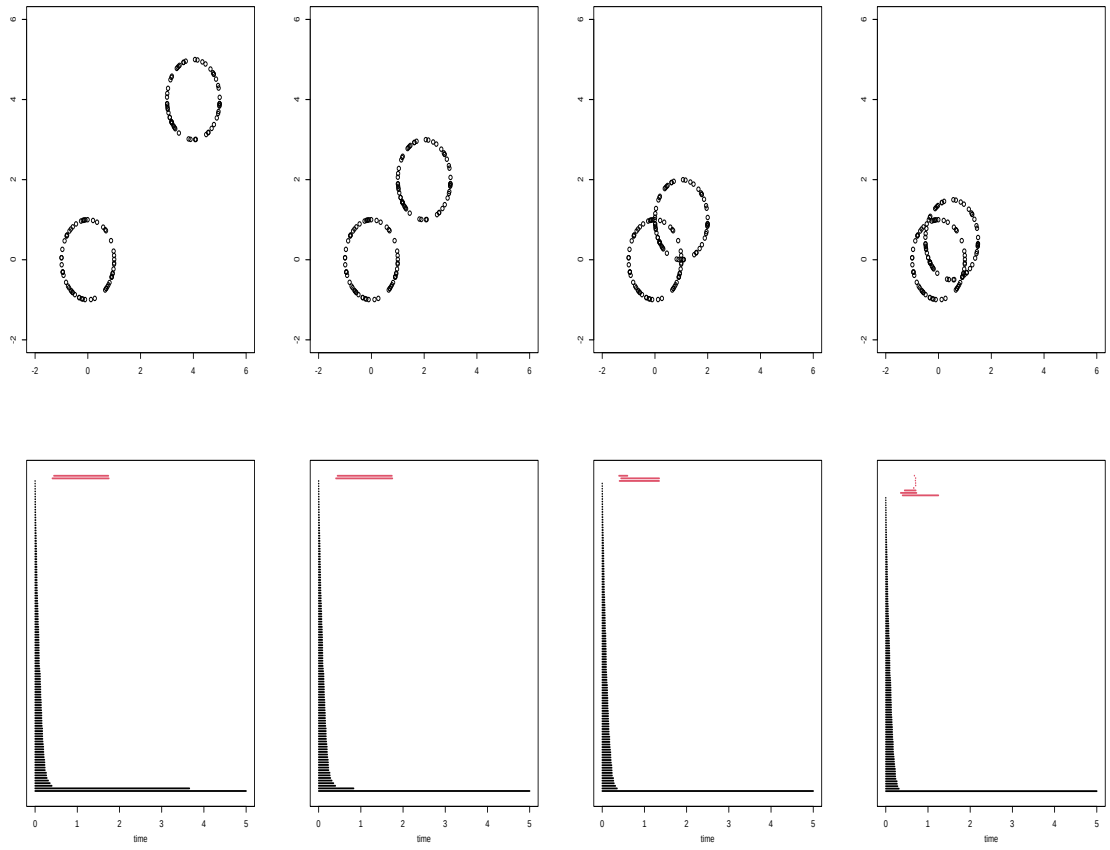


Figure 3.7: With two disjoint circles slowly approaching each other, we can see how the barcodes evolve. A non-trivial intersection causes a third bar to appear, representing the given loop. Moreover, multiple smaller bars generated by small loop show up the closer the circles are.

Compare this with 3.8. Moving from top-left to bottom-right, we see that increase of red triangles indicating the presence of loops. In the last image, the presence of many short bars is reflected in the presence of multiple triangles around the diagonal, both confirming that the lifetimes of these objects are short when compared to the triangles above.

Another metric that you may come across is the so-called Gromov-Hausdorff metric.

Definition 3.0.10. Let A and B be non-empty subsets of a metric space (P, d). Then

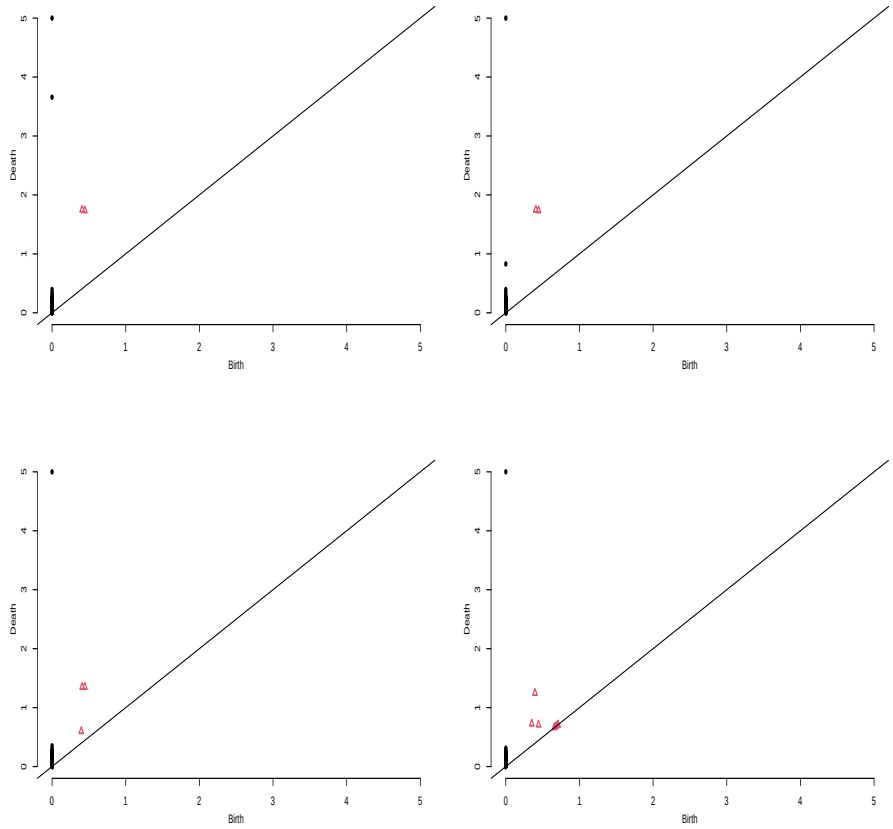


Figure 3.8: With two disjoint circles slowly approaching each other, we can see how the barcodes evolve. A non-trivial intersection causes a third bar to appear, representing the given loop. Moreover, multiple smaller bars generated by small loop show up the closer the circles are.

the *Hausdorff distance* between A and B is defined as

$$d_H(A,B) = \max\left(\sup_{a\in A}\inf_{b\in B} d(a,b), \sup_{b\in B}\inf_{a\in A} d(a,b)\right),$$

or equivalently as

$$d_H(A,B) = \inf_{\varepsilon>0}\{B \subseteq A_\varepsilon, A \subseteq B_\varepsilon\},$$

where A_ε and B_ε denotes the set of all points within $\varepsilon-$distance of A and B.

A simple example of what this means geometrically can be found in 3.9.

In general, we cannot assume that the metric spaces A and B are subsets of some common ambient space. To remedify this problem, we can simply consider the infimum of the Hausdorff metric over all isometric embeddings of A, B to some larger metric space. This correction is what we call the Gromov-Hausdorff metric.

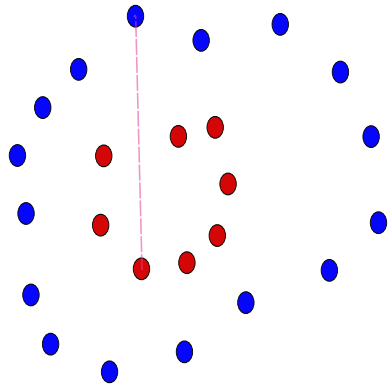


Figure 3.9: The Hausdorff metric is given by the point in A with the largest distance to the closest point in B.

Definition 3.0.11. Let (P, d_P) and (Q, d_Q) be compact metric spaces. The *Gromov-Hausdorff* distance between P and Q is defined as

$$d_{GH}((P, d_P), (Q, d_Q)) = \inf_{\theta_1: P \to Z,\, \theta_2: Q \to Z} d_H(P, Q),$$

where θ_1 and θ_2 are isometric embeddings of (P, d_P) and (Q, d_Q) in (Z, d_Z).

The usefulness of this comes from the following fact.

Theorem 3.0.7. The Gromov-Hausdorff distance is a metric on the set of isometry classes of compact metric spaces.

The definition above doesn't lend itself to computation in a straightforward fashion. For that reason, there exists an alternative formulation: let $\mathcal{R}$ be a mapping between P and Q such that there exists a tuple with the first coordinate in P and second in Q. Then, the Gromov-Hausdorff distance can be formulated as

$$d_{GH}((P, d_P), (Q, d_Q)) = \inf_{\mathcal{R} \in P \times Q} \frac{1}{2} \left(\sup_{(x,x') \in \mathcal{R}, (y,y') \in \mathcal{R}} |d_P(x, y) - d_Q(x', y')| \right).$$

We can see that the Gromov-Hausdorff distance, in the same spirit as the bottleneck distance, roughly measures the maximum distortion in the best matching between P and Q, as opposed to two sets of persistence diagrams.

The stability theorems that we introduced fist can be expressed in terms of the Gromov-Hausdorff distance. The first approach is sometimes called the "Morse filtration" approach to persistent homology, while the one using the Gromov-Hausdorff distance is what we call the "point-cloud" approach.

All the versions have a common generalization in the algebraic stability theorem, which, ignoring technical details, says that for persistence technical modules that are $\kappa-$interleaved (a precise way to express what being approximately isomorphic is), then the resulting barcodes are within κ in the bottleneck metric. [CCSG+09].

Chapter 4

Applications of TDA

In this section, we will go over some of the more popular methods, techniques and algorithms that are used within Topological Data Analysis. The most theoretically demanding one is Persistent Homology that we worked on introducing in the previous chapters in the most basic and stripped down manner to not overwhelm the reader. We will look at the popular clustering and dimensionality reducing algorithms such as UMAP and Mapper, both closely related to each other. In the direction of homology, we will look at other equivalent and potentially more useful representations such as persistence images, landscapes and entropy. We will demonstrate them on the classical setting of point clouds but also time series and graphs.

All methods will be accompanied by examples, both toy examples to better understand and real-life data to show one can use the methods to obtain new information. It is hopelessly naive to believe that we can summarize all the ways TDA and its surrounding methods have been and can be used to. As such, the reader will be pointed to different articles that cover more intricate applications that go beyond the scope of this work.

4.1 UMAP

UMAP, short for *Uniform Manifold Approximation and Projection for Dimension Reduction*, is a dimension reduction technique, similar to something like t-SNE, that is commonly used for dataset visualization, clustering, filtering or embedding within neural networks. UMAP and Mapper complement each other and are usually used together, hence why we talk about UMAP first.

UMAP is built on three, relatively reasonable, assumptions about your dataset:

- The data is uniformly distributed on a Riemannian manifold.
- The Riemannian metric is locally constant or can be approximated as such.
- The Riemannian manifold is locally connected.

If the assumptions are satisfied, it is then possible to model the ambient manifold with what we call a *fuzzy* topological structure. We will briefly explain and

summarize the workings of the algorithm, although it is recommended to read the original article [MHM20] for a more detailed explanation.

The first step is to approximate the manifold the data supposedly lies on. Assuming that the manifold isn't known in advance (perhaps from some theoretical description of the model), we need to approximate the geodesic distance. It simplifies the problem to assume that the data is uniformly distributed on the manifold. Given this, we can center balls of an appropriate radius around each point to obtain a crude approximation of the manifold. We can then approximate geodesic distances from each ball to its neighbour.

Unfortunately, this means that we have independent notions of distances that may or may not be compatible. To merge all the local metric spaces into one compatible global structure, we utilize fuzzy simplicial sets, where a fuzzy set is characterized by a carrier set A and a map $\mu : A \to [0, 1]$ we call the membership function. This turns membership from a property attaining two values – true or false – into a property taking values in the unit interval. Each metric space is translated into a fuzzy simplicial set via a categorical construction (the fuzzy singular set functor).

The goal, once we have constructed our fuzzy simplicial set, is to find a low-dimensional representation of our data that closely matches the topological structure. This optimal embedding is found after minimizing the error between the two representations with respect to cross entropy of the fuzzy sets. From a purely computational perspective, UMAP can be described in terms of weighted graphs; placing UMAP in the class of graph learning algorithms such as Isomap, the above-mentioned t-SNE and Laplacian Eigenmaps.

We will go over some examples of UMAP on well-known datasets and later compare the obtained results with the one we will see with Mapper in the next section.

4.1.1 Penguin dataset

We will start with a simple example using the known Penguin dataset [GWF14]. The simplicity of the dataset will make it easier to highlight and interpret the results from UMAP. The dataset has few features, as seen in 4.1, with some data missing and replaced with NaNs. While it would be recommended to use some form of imputation to replace them in a proper study, we will simply drop the missing values in this example without worrying too much.

The dataset itself contains measurements about bill dimensions and flipper lengths, alongside metadata regarding the penguin's body mass and sex. The values we will focus on are the first four, ignoring the species and island variables. There are a total of 333 penguins in this dataset, and we can start with a simple pairwise scatter plot matrix to get a rough idea of what we're dealing with, seen in 4.1

For this task, we will use the Python library UMAP [MHSG18] to reduce the dimension in a way that preserves the topology of the 4-dimensional space we are working with. We can first remove the missing values and the columns we

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	male	2007
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	female	2007
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	female	2007
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN	2007
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	female	2007

Table 4.1: First five rows of the Penguin dataset.

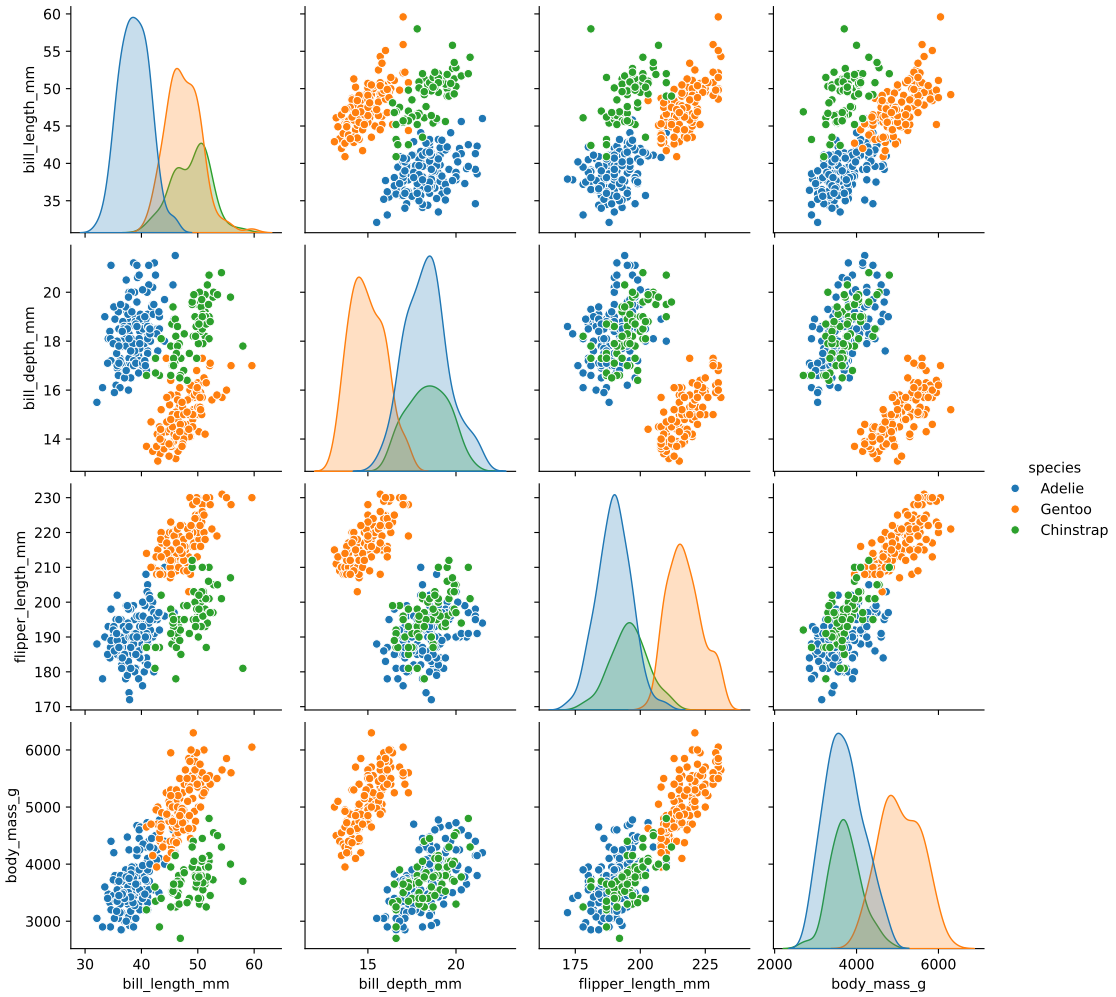


Figure 4.1: Pairwise plots of the Penguin dataset for the bill length, bill depth, flipper length and body mass variables.

won't need for this short analysis. Since we are working with different scales, it is generally useful to convert the features into z-scores or use a different scaler process.

The result of reducing the dimension can be seen in 4.2. Comparing it to 4.1, it does a good job of capturing the relationships between the three groups. Although, in this case specifically, we already found everything we needed in the scatter plot matrix, which was only possible because we worked with four dimensions. Once the number of features is higher, a scatter plot matrix becomes

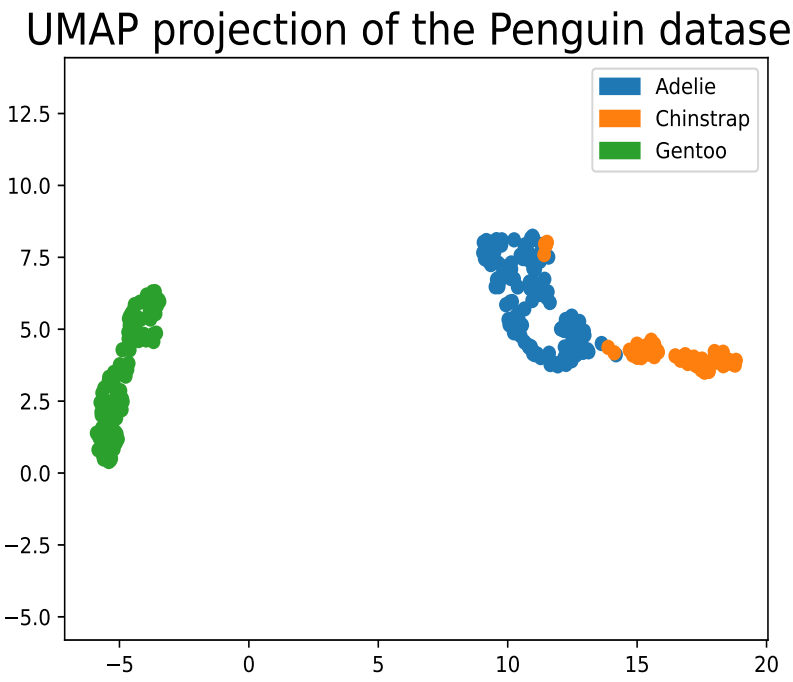


Figure 4.2: UMAP dimension reduction result on the Penguin dataset.

unwieldy and difficult to read properly.

4.1.2 Digits dataset

For an example with a high number of features, we can consider the known digits dataset [AK98], one of the many toy datasets used to validate both old and new algorithms. We can iterate over the individual images to display them on a grid in 4.3 to see what we are working with.

The digits dataset contains 1797 bitmaps of hand-written digits in 10 classes – each class corresponding to its digit. Most digits are somewhat readable but there are few that are simply too blurred to be considered useful for any further analysis. Such bitmaps should be separated into their own cluster of outliers, creating one additional class in the end. The usual goal is to classify and cluster each bitmap into its appropriate class. We will see that with UMAP, we obtain satisfying results with a simple application of the algorithm. We see in 4.4 that reducing the dimension with UMAP already gave us a rather solid clustering of the digits with significant clusters. It shouldn't come off as a suprise that fairly distinct digits like 0, 9, 2, 6 or 7 are far away from the rest; whereas digits 1, 8, 3 lie close to each other.

We will look into the digits dataset two more times – once with Mapper in the next section and the second time using homology groups instead.

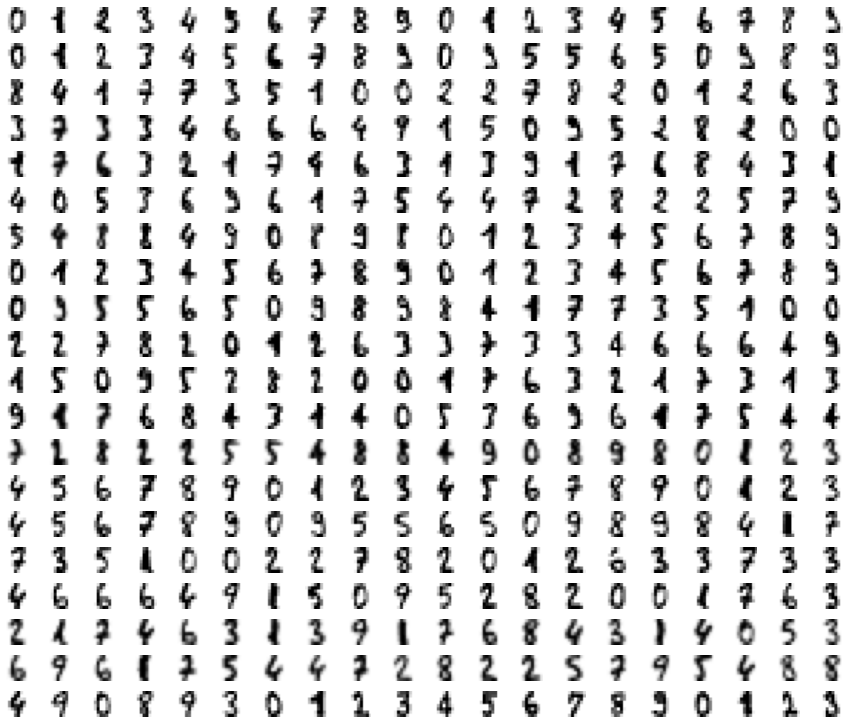


Figure 4.3: First few images of the digits dataset

4.1.3 Fashion and clothes

We can look at a much larger dataset to see how UMAP compares. Fashion-MNIST [XRV17] is a dataset consisting of 60,000 examples of Zelando's clothing articles, each article being a 28x28 grayscale image associated to one of 10 classes. The classes go as follow, labelled from 0 to 9 – T-shirt, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle Boot. As was the case for the digits dataset, one is typically interested in classifying and clustering the clothing articles while removing the images that are too blurred. Applying UMAP, we can see the resulting projection on 4.5. On a more technical note, the MNIST dataset should be ideally replaced with a different one since it is too easy for modern machine learning algorithms, is usually overused and cannot represent modern Computer Vision tasks. But since this text isn't about Computer Vision, we can safely proceed.

It intuitively makes sense to see that shoes should be generally projected close to each other, as we see with sneakers, sandals and ankle boots. Trousers are found in the top-left corner, isolated from the rest. In the *UMAP examples* notebook, we also provide an interactive version of the resulting projection on a subset of Fashion-MNIST, allowing the user to see the labels of individual points upon zooming.

An interesting property that UMAP allows us to study is the connectivity of the topological representation of the original high-dimensional data. This can be useful for diagnostic purposes, but in the case of this text, it corresponds to study-

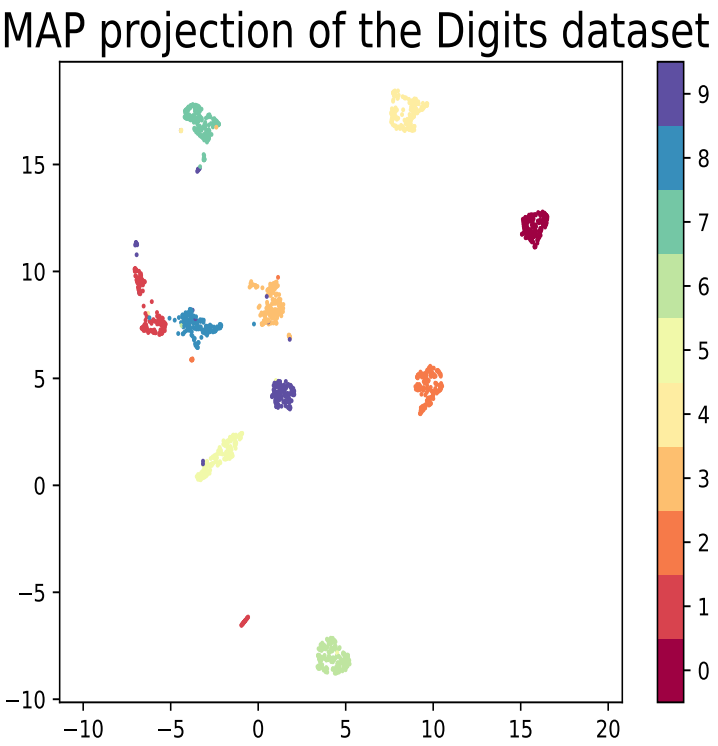


Figure 4.4: Result of UMAP on the digits dataset.

ing the connected components. It should be noted that computing and plotting the connectivity of the dataset can be rather expensive, so it is recommended to check subsamples instead. We can see examples of it on 4.6 and 4.7.

4.2 Mapper

As was mentioned in the beginning of this text, the methods of persistence topology can become computationally infeasible for larger datasets, making them practically unusable for datasets with measurements in the counts of millions and above. This was seen in the rapidly-growing number of simplices in the Vietoris-Rips complex. Furthermore, interpreting the output of persistence homology for high-dimensional data can be difficult to interpret (what does the presence of a 1-dimensional loop in a 50-dimensional space indicate?). One approach to tackle this issue is to combine persistence with clustering, resulting in the Mapper algorithm that we're going to discuss.

We're going to introduce the multiscale clustering method of Singh, Mémoli and Carlsson [SMC+07]. Let's assume that the data is presented as a finite metric space (P, d). Let us then choose:

- a *filter function* $f : P \to \mathbb{R}^n$

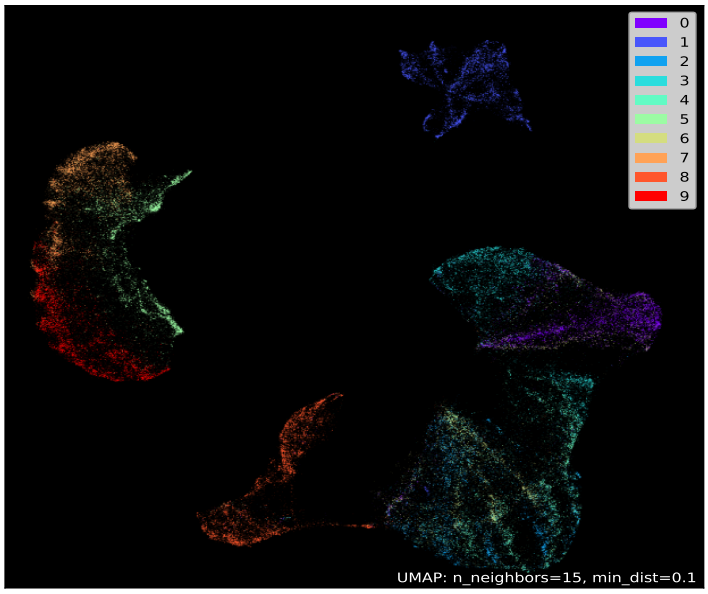


Figure 4.5: Result of UMAP on the much larger Fashion-MNIST dataset.

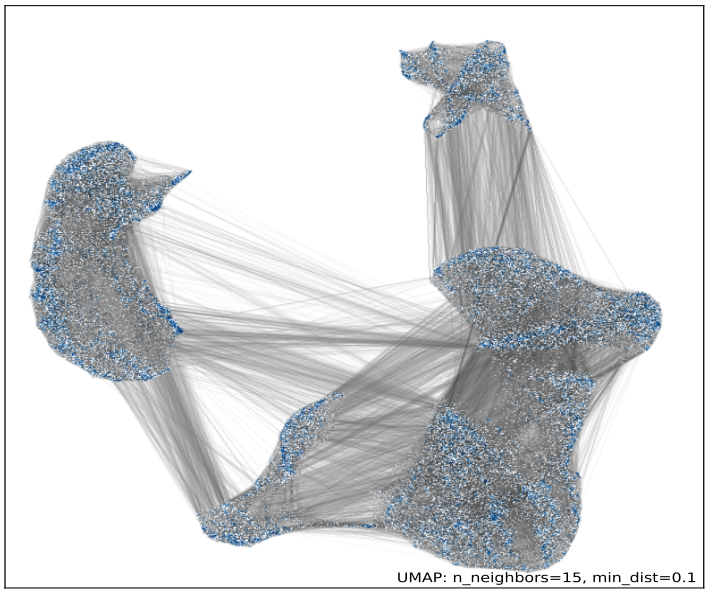


Figure 4.6: Points of connectivity of Fashion-MNIST.

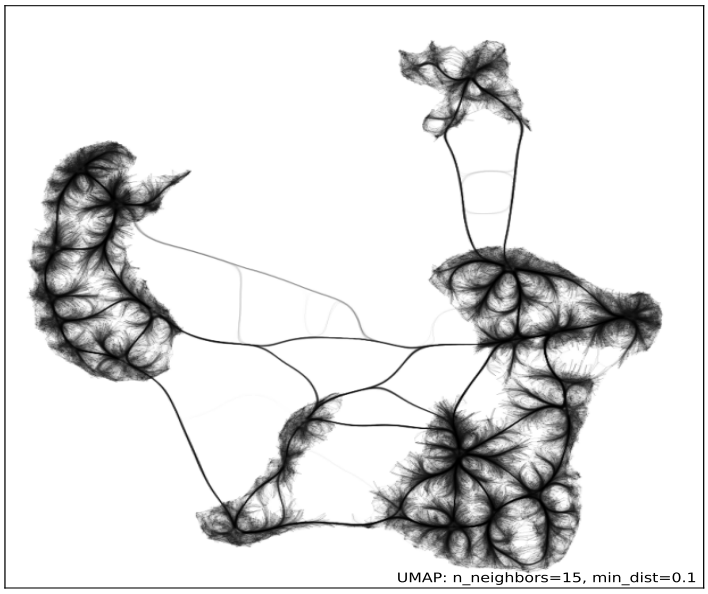


Figure 4.7: A less busy view of connectivity of Fashion-MNIST.

- a cover $\mathcal{C} = \{U_\alpha\}$ of the range of f in $\mathbb{R}^n$

The cover is typically taken to be a collection of overlapping cubes. In $\mathbb{R}$, we would be looking at a collection of closed intervals. Software packages will want you to specify the number of covering intervals and the percentage of overlap you allow. We then proceed with the algorithms as follows:

1. Cluster each inverse image $f^{-1}(U_\alpha) \subseteq P$, regarded as a metric subspace of P, for all $U_\alpha \in \mathcal{C}$. Denote by $C_{\alpha,i}$ the i-th cluster. Here, a point can appear in multiple nodes due to the overlap.

2. Form a graph where the vertices are given by the clusters $C_{\alpha,i}$ as α and i vary and there is an edge between $C_{\alpha,i}$ and $C_{\alpha',j}$ when

$$C_{\alpha,i} \cap C_{\alpha',j} \neq \varnothing$$

3. Assign a color to each vertex in the graph corresponding to a particular cluster $C_{\alpha,i}$ according to the average value of f on $x \in C_{\alpha,i}$.

Mapper's strength lies in the flexibility of choices we can make during the creation of the graph. For the projection, we can take any projection method from mathematics, statistics, machine learning or econometrics – PCA, t-SNE, UMAP, even something as simple as projecting on the $x-$axis.

The same applies for the choice of clustering algorithms. We can pick any method – hierarchical, density-based and any distance metric. The distance metric does not need to satisfy the triangle inequality, giving us the option to choose from the wider range of similarity metrics like the Kullback-Leibler divergence and so on.

Because Mapper is primarily a visual tool, most software packages include additional tooling to improve upon the basic algorithm and help with further exploration. That may involve a combination of the following:

- Size the nodes by a function of interest (number of members inside the cluster, etc.)
- Color the nodes by a function of interest (average spendings, number of purchases, etc.)
- Shape the nodes given some condition (circles for group 1, squares for group 2, etc.)
- Size the graph edges by a function of interest (number of overlapping cluster members, etc.)
- Color the graph edges by a function of interest (average color of connected nodes, etc.)
- Shape the graph edges given a condition (dotted lines for group 1, solid line elsewhere, etc.)
- Descriptive statistics on the nodes and the graph

All in all, this makes Mapper one of the most flexible approaches to dataset visualizations, giving the user the option the construct the final graph exactly the way they intended. For the Mapper examples, we will make use of the Python libraries *scikit-tda* [ST19] and *giotto-tda* [TLT+20].

We can start with simple, synthetic examples before moving onto real data to see what the results of Mapper are and how they differ when we change the parameters or used methods.

4.2.1 Toy examples

We will start by generating 5000 points sampled from two nested circles in the plane with some added Gaussian noise. We can see the plotted dataset in 4.8. In our first attempt, we will project our data onto the $x-$ and $y-$axes, choose 10 covering intervals with a 30% overlap and use the DBSCAN clustering algorithm.

In figure 4.9 we can see that Mapper succesfuly captured the salient features - those being the two holes in the nested circles. The coloring of the node used the default settings of *giotto-tda* – coloring the nodes by the average row index of the data in the cluster; with 5000 generated points, we thus have 5000 rows. The node size here is given by the number of data points in the node.

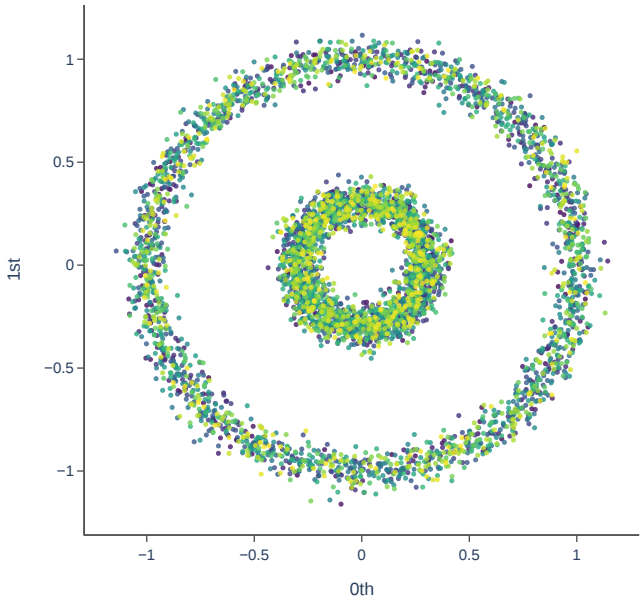


Figure 4.8: Our toy example for now - two nested circles.

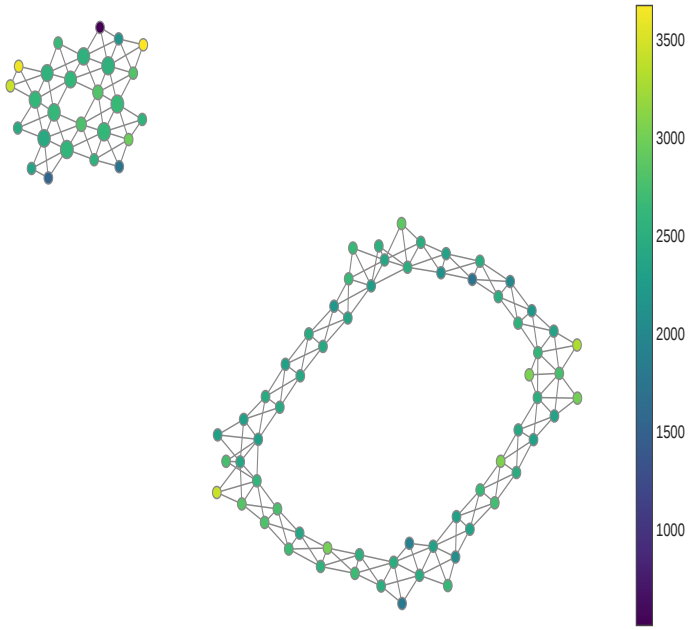


Figure 4.9: First result of applying Mapper.

The coloring or the size do not convey any particularly interesting information in this case; here we were more interested to recover the presence of the two underlying holes. On the other hand, coloring by the average value of the $x-$ or $y-$coordinates would be more instructive, as revealed on 4.10. Here we see that the $x-$coordinates of the inner circle, closer to the origin, are colored in such a way that lies near the 0 point of the spectrum. We would see a similar, but flipped, version of the coloring had we used the y-coordinates instead.

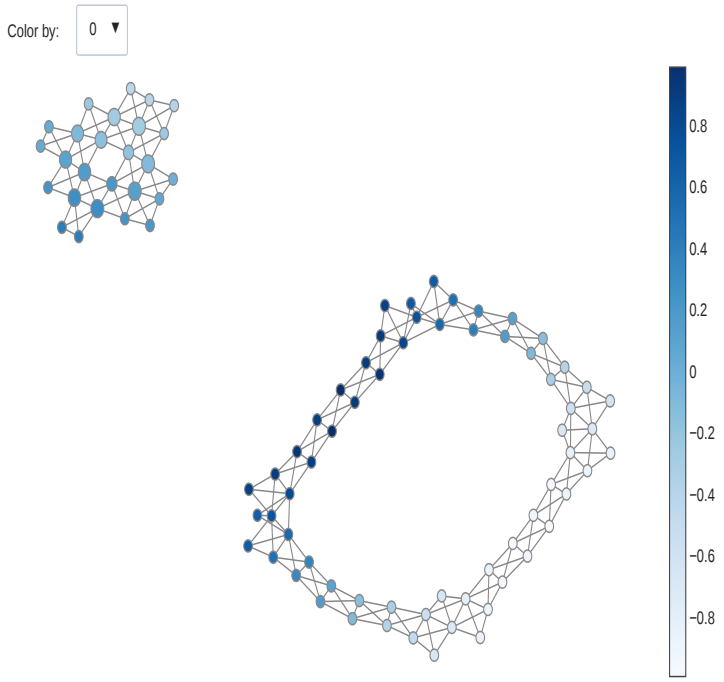


Figure 4.10: Nodes colored by the average x-coordinate.

We can, if we want to, color by something like the first principal component of PCA, as seen in 4.11. In this context, it doesn't reveal anything particularly interesting but it may be more useful in applications where PCA is actively used.

As mentioned above, we can colord by proportions of some categorical variable – one that is either already present or one that we create. Let us try to separate the inner and outer circle more cleanly. We can add a categorical column to the dataframe, with a value of "A" for the outer circle and "B" for the inner circle, where the testing criterium is whether

$$x^2 + y^2 < \frac{1}{4}.$$

The result of this can be seen on 4.12, where the inner and outer circle are now exactly separated. As with the coordinates, we would see a flipped version of this graph for group "B". It should also be noted that the color spectrum only attains two values – $\{0, 1\}$.

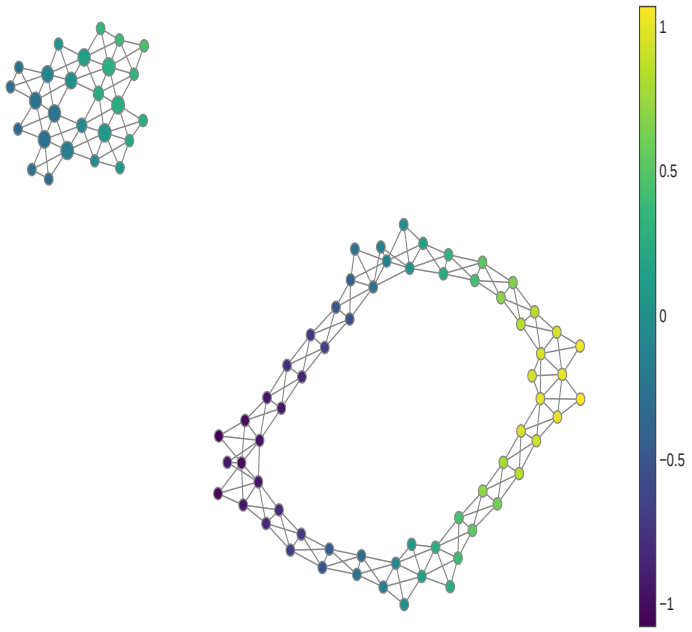


Figure 4.11: Nodes colored by the average first PCA component.

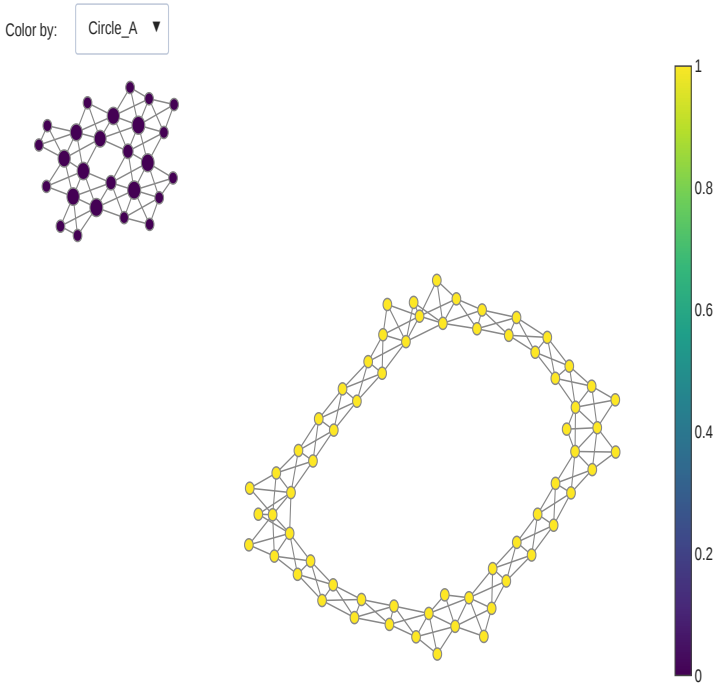


Figure 4.12: Nodes colored by categorical variables.

Finally, we can take a look and compare what happens when we use a different filtration function on the dataset. For example, we could project by taking the sum of the (x, y) coordinates. This results in 4.13.

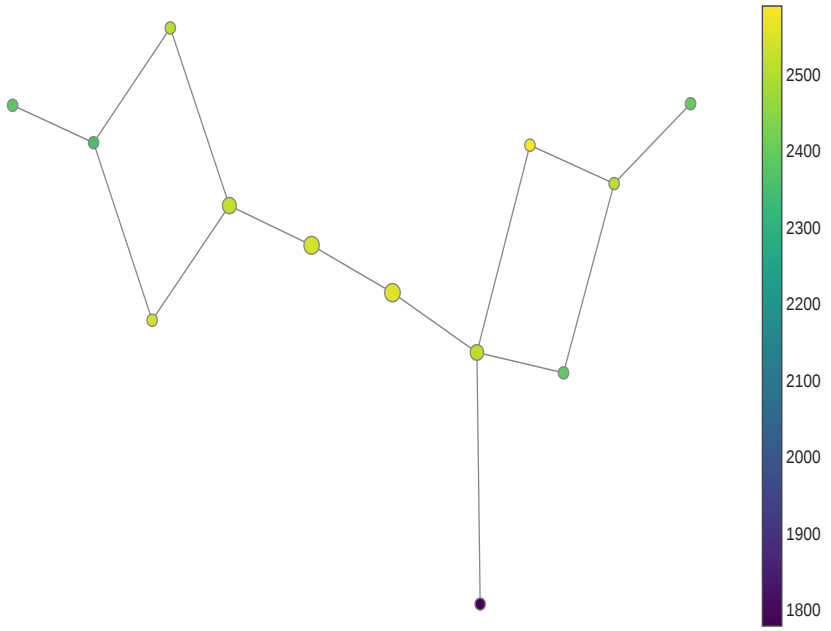


Figure 4.13: The effect of a different filtration function; in this case, the sum of the individual coordinates.

The presence of the two loops was still properly encoded, although this time we don't have the same level of separation as we did by projecting on the individual coordinates.

4.2.2 Stock market data

We can now take a look at how one could potentially use Mapper to improve your analysis of stock market data. Specifically, we will look at the closing prices of the SP500 index companies, starting from 01.01.2024 up to 01.01.2025. The choice of date is arbitrary, and one could easily choose a much larger time interval to consider, but for the sake of this example, one year will suffice. The data was fetched from Yahoo Finance through their API and stored in a data frame. A simple indicator of interest when dealing with stock market data is the log-return of the stock price. As such, our first filtration function will be the log-percent return after the data is normalized.

Our goal here will be to cluster the company tickers in a way that similar behaviour of the stock value is represented in similar tickers sharing the same cluster. We could take the transpose of our dataset instead and try to cluster the 252

trading days of the year. This could perhaps be useful if we wanted to analyze which days or periods of the year showed significant drops or increases in the SP500 index. We could then correlate the clusters with geopolitical events such as the US presidential election, which brought an increase in stock prices for large tech companies, for example.

After normalizing the dataset, we first reduced the dimension to 150 via the Isomap algorithm before using UMAP to reduce it to 2. Our clustering algorithm of choice here was DBSCAN with the cosine metric. Using *scikit-tda*, the result is stored in a *html* file that you can open in your web browser. All the files for this section can be located in the *figures/* folder. Our first view can be seen on 4.14.

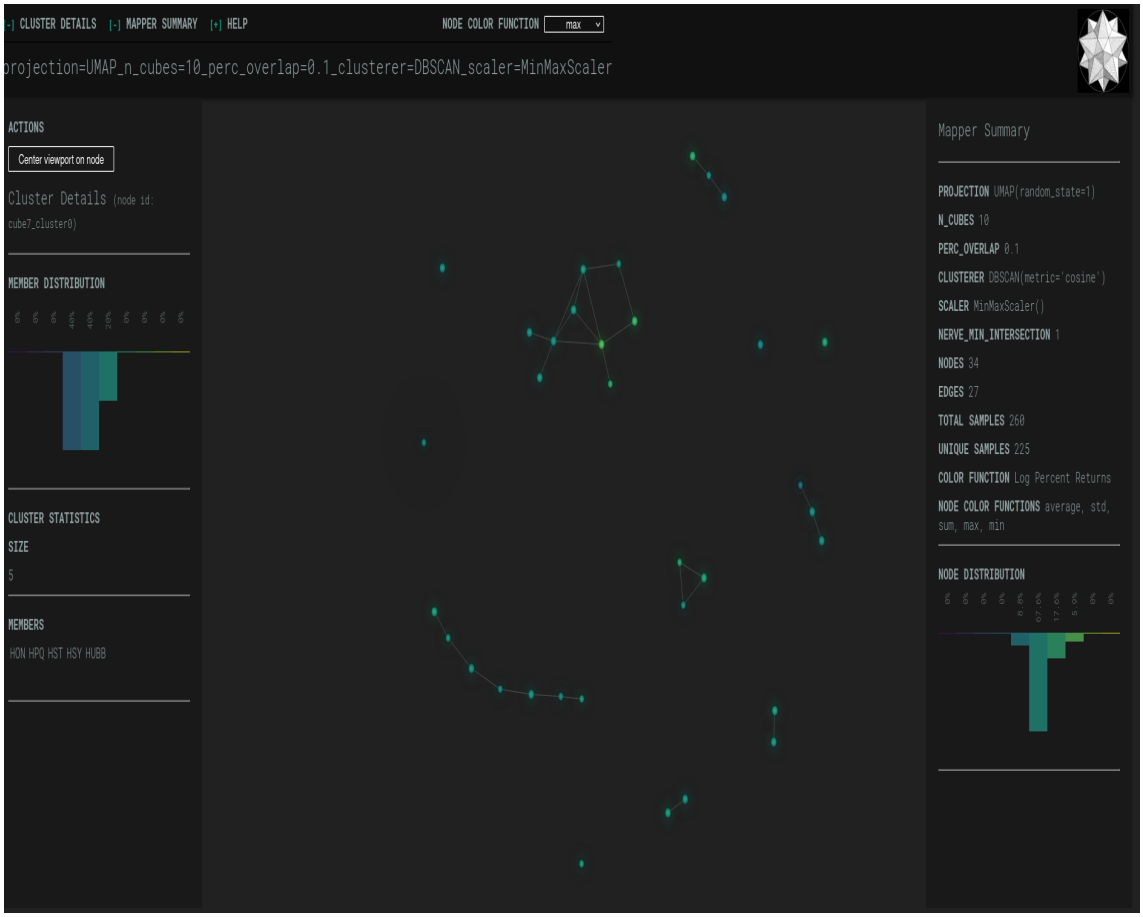


Figure 4.14: View of the Mapper clustering summary with *scikit-tda*.

We interpret the results as follows – given that our color function is the log-percent return of the individual tickers, the clusters correspond to tickers who share a similar trading profile between each other. That is, underperformers would have darker colors, while companies that performed exceptionally well would be on the brighter end of the spectrum. In the graph 4.14, we added multiple cluster coloring rules to compare. As mentioned previously, the default strategy is to color the final cluster by the average of the nodes inside. Here, we can also color by the average, maximum, minimum, the sum or the standard deviation. The only interesting results are when we look at the minimum and

maximum log-percent returns of the clusters. While we have some stocks that performed better than the rest, most companies had a similar run during that one year. One could anticipate greater differences had we considered the closing prices over multiple years instead.

We could also choose a different color function, such as the volatility of the underlying stock. The notion of volatility we will look at is quite crude – we simply take the standard deviation of the stock prices over the 252 days. Stock traders and market specialists would throttle us for this gross simplification, which is why we ignore their opinions and continue. The different color function drastically changes the resulting graph, as seen in 4.15.

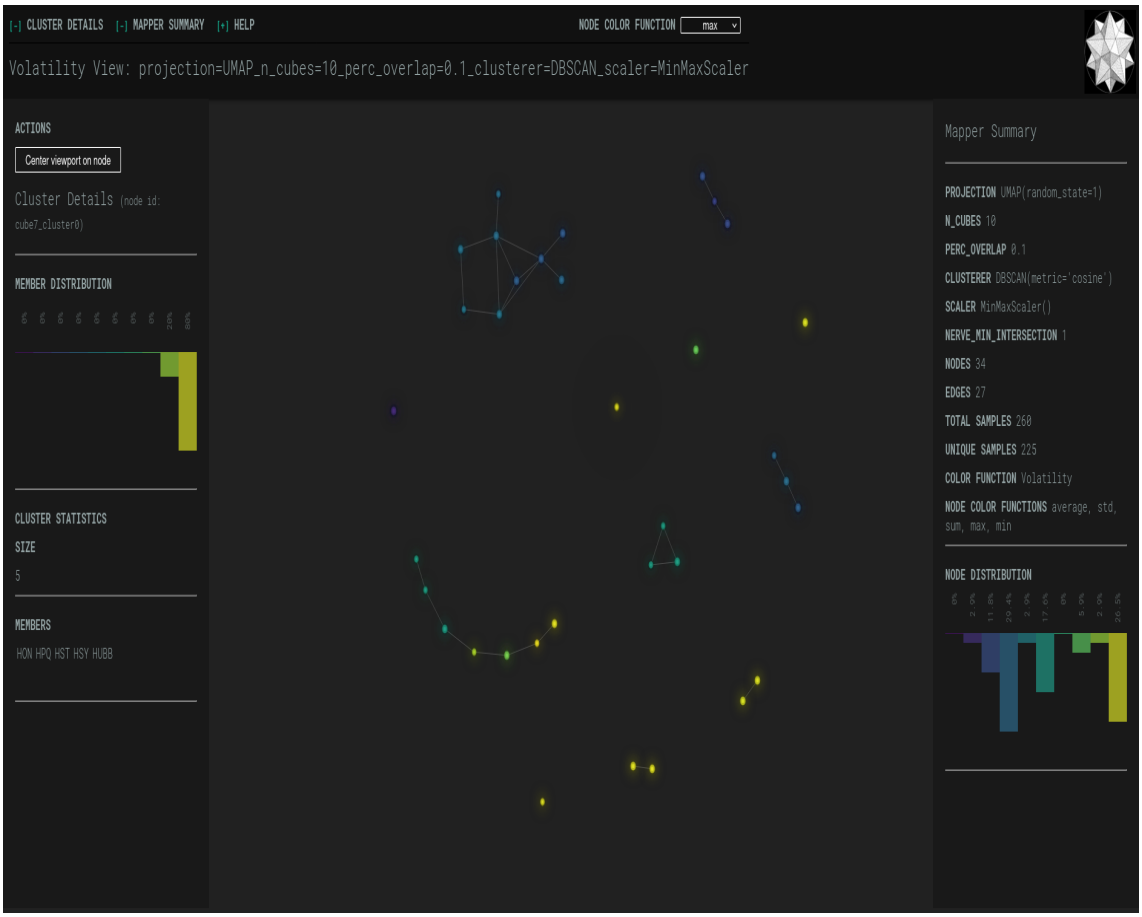


Figure 4.15: Mapper graph of the SP500 data with the color function being the volatility of the stock.

Here we use the maximum of the volatility to color the resulting clusters, looking for stocks with an unusually high variance in the closing prices. Already, we can see that we have more outliers on both ends of the color spectrum, compared to the log-percent return. The benefit of the Mapper implementation in *scikit-tda* is that it allows us to see directly which SP500 tickers are in the clusters by simply hovering over them. It also gives us the cluster ID, allowing us to plot the cluster data separatly from the rest. This way, we can plot the dark purple cluster, suggesting the lowest maximal volatility of closing prices in 4.16.

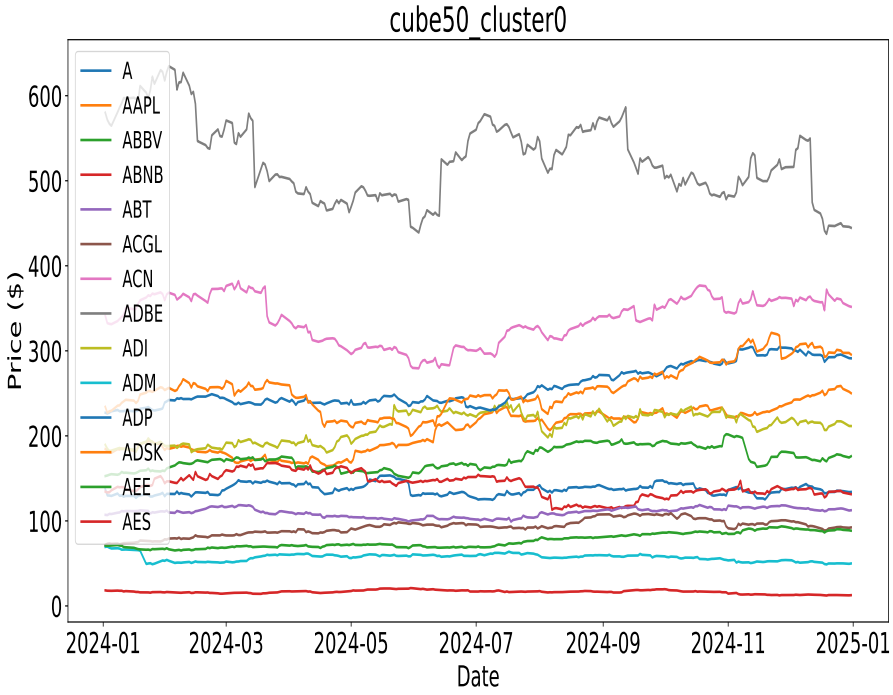


Figure 4.16: Cluster of tickers with the lowest maximal volatility.

We can contrast it with the bright yellow clusters, the ones with the highest maximal volatility, in 4.17.

At first glance, the closing prices of the bottom four tickers do not seem highly volatile due to the wide price range we are plotting them against. Zooming on one of them, for example, HPQ – the ticker of the HP company known for printers – reveals a more interesting structure in 4.18.

One could then continue with a more refined and detailed analysis of the clusters we obtained, looking into the differences and common points between them. Different coloring functions and rules could also be used, alongside other projection and clustering algorithms. All this is to show that Mapper is a powerful tool in the hands of someone with industry and theoretical knowledge of the data they have to work with, allowing them to carefully choose which algorithms, metrics and coloring functions they believe to be of importance.

4.2.3 Digits dataset, revisited

We can take a look at the digits dataset again, this time using Mapper instead of UMAP on its own. This example should highlight the fact that we can customize the resulting graph as much as we want to. In this case, we will customize the tooltips that show whenever we hover over one of the clusters. In 4.19, we have chosen our custom tooltip function to be the images themselves, giving us the ability to directly compare the different clusters. For the projection function, we have used the t-SNE algorithm, followed by DBSCAN to do the clustering. The

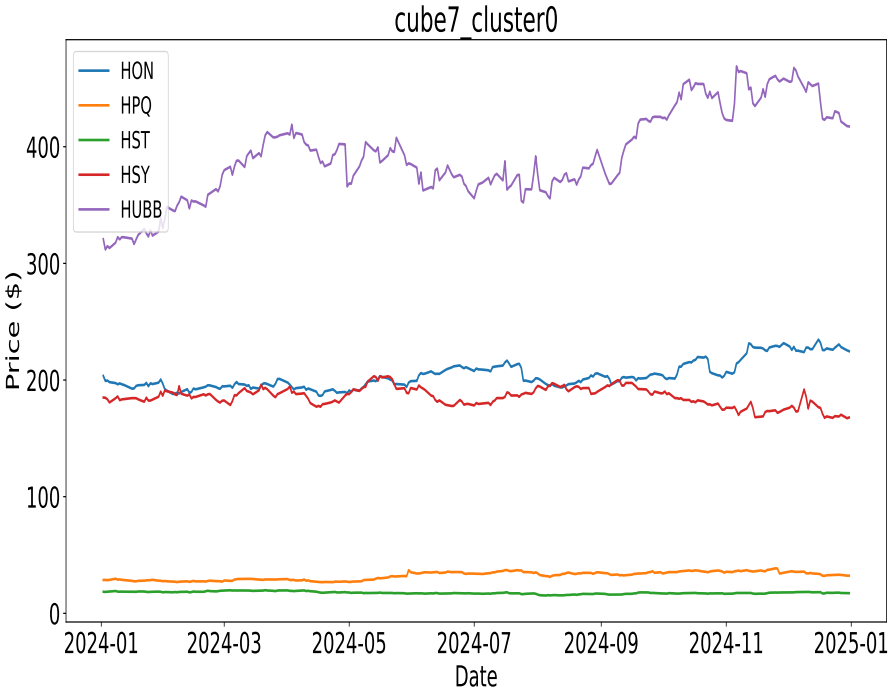


Figure 4.17: Cluster of tickers with the highest maximal volatility.

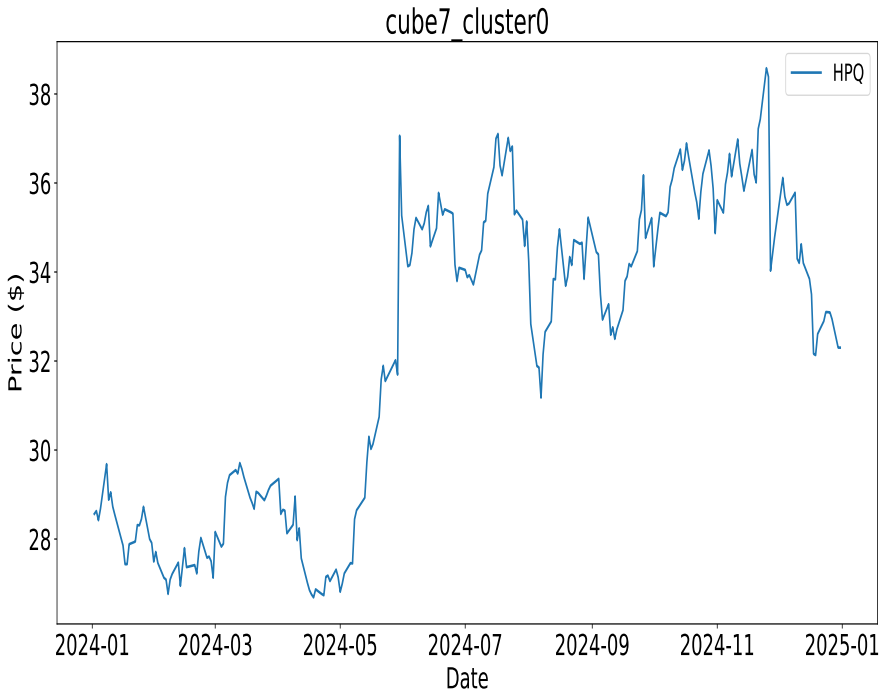


Figure 4.18: Stock prices of the company HP during 2024-2025.

color function here is the label of the digit, and we kept the default rule of assigning the mean color to the cluster. This gives us a nice separation of the digits, all differentiated by a different color value. In the top right corner, we can see a cluster with multiple colors where different digits were blurred enough for DBSCAN to consider them as equal. Our custom tooltip can immediately show us which digits we are looking at.

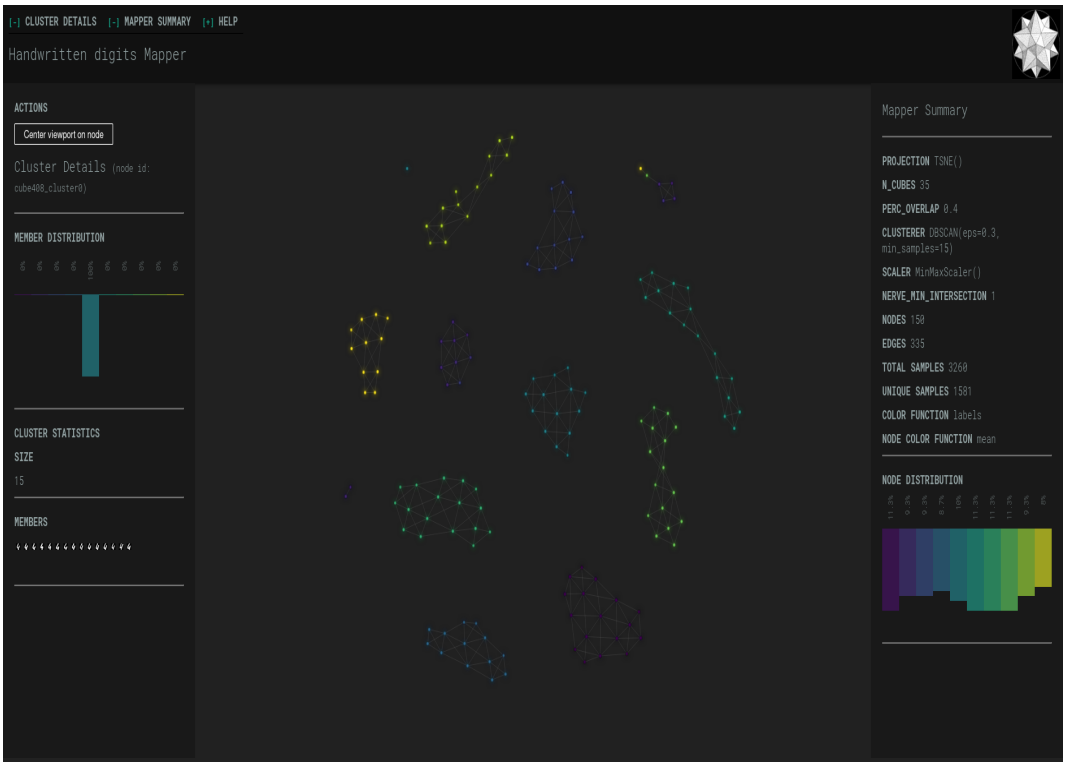


Figure 4.19: Mapper graph of the digits dataset with a custom tooltip function.

We can compare this with the Mapper graph where our tooltip function is the label of the digit, as seen in 4.20. This won't reveal any new information but it could be useful to compare the label of the digit with the image when the blur makes the digit illegible. The *html* files for the two graphs can be located in the *figures/* folder.

4.3 Persistent homology in practice

At last, we can start with example where we will compute homology groups of the datasets at hand. The key takeaway here should be that persistent homology alone usually doesn't reveal all the information that we would need. Instead, we use it to extract topological features from a dataset, vectorize them and send them into a machine learning pipeline as additional training data. If used correctly, topological features can improve the performance and precision of your model, leading to superior results. The downside is the high computational cost of the

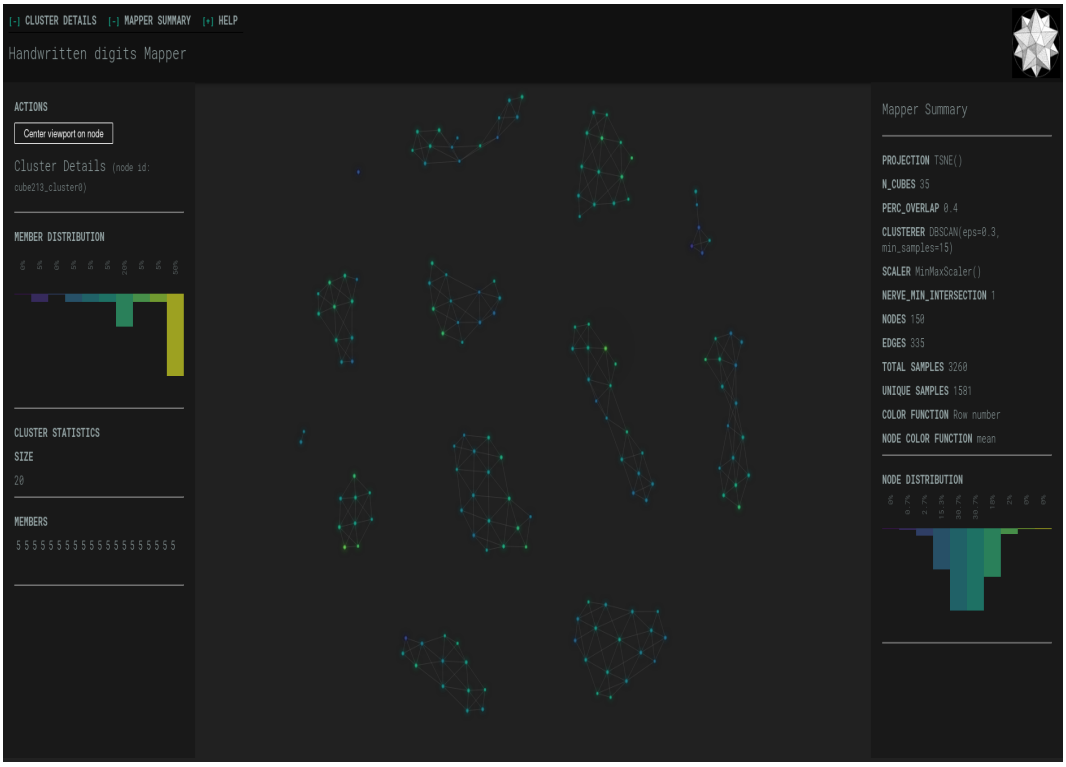


Figure 4.20: Mapper graph of the digits dataset with labels.

procedure, making this approach not viable for datasets in the order of millions of rows and above.

4.3.1 Digits, one the last time

We will revisit the digits dataset again, this time using persistent homology to build a *scikit* classifier. As we will see in the examples to follow, persistent homology is usually used to extract topological features that can be later used in standard machine learning algorithms or statistical analysis. We have 70,000 images, each having 784 features representing a pixel intensity. The first task is to reshape the vectors into arrays of dimension 28x28. To save computational time, we will work with a smaller subset again. We split the full dataset into a training a testing dataset of sizes 60 and 10, respectively.

Because images are made of pixels and are not point clouds, we use filtrations of cubical complexes instead of simplicial ones. Ignoring technical details, all that this does is replace simplices with little cubes. The homology groups carry the same meaning as before. It simply turns out that when working with images, using cubes is more natural than using a simplicial object. To use cubical complexes, the filtrations are built from binary images, forcing us to convert the greyscale image to binary by applying a treshold on the pixel values. In this example here, we used a treshold of 0.4.

Given a binary image $\mathcal{B}$ of some digit, we will use the *radial filtration* $\mathcal{R}$, which assigns to each pixel p a value corresponding to the distance from some predefined

center c of the image

$$\mathcal{R}(p) = \begin{cases} ||c - p||_2 & \text{if } \mathcal{B}(p) = 1 \\ \mathcal{R}_\infty & \text{if } \mathcal{B}(p) = 0 \end{cases}$$

where $\mathcal{R}_\infty$ is the distance of the pixel that is the furthest from c. To reproduce the filtered images from the original article, we have chosen c to be the point $c = (20, 6)$. Applying the radial filtration $\mathcal{R}$, we effectively transform the binary image into a greyscale one with pixel values increasing as you move further away from the point c. These pixel values can be used to define a filtration of cubical complexes $\{K_i\}$, where K_i contains all pixels with value less than the $i-$th smallest pixel value in the greyscale image. In later examples, we will see that this is what we call the sublevel set filtration of the image's cubical complex.

Once we have created our filtration, computing the persistent homology of it is straightforward. For example, if we take an image corresponding to the digit 8, its homology groups are found in 4.21.

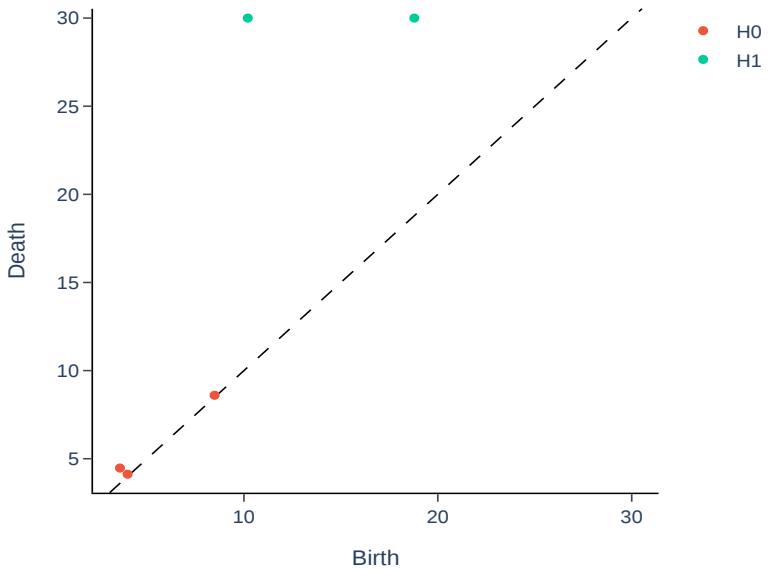


Figure 4.21: Cubical persistence homology of the digit 8.

Our sanity check works – we see two H_1 generators corresponding to the two loops in the digit 8 and a single H_0 for the single connected component. Most methods require some sort of scaling for better results, which we can do with the persistence diagram as well, see 4.22.

The final step is to vectorize our persistence diagrams, since most machine learning pipelines only accept an array of vectors. This can be achieved in multiple ways. For our single image of the digit 8, we used a Gaussian kernel and

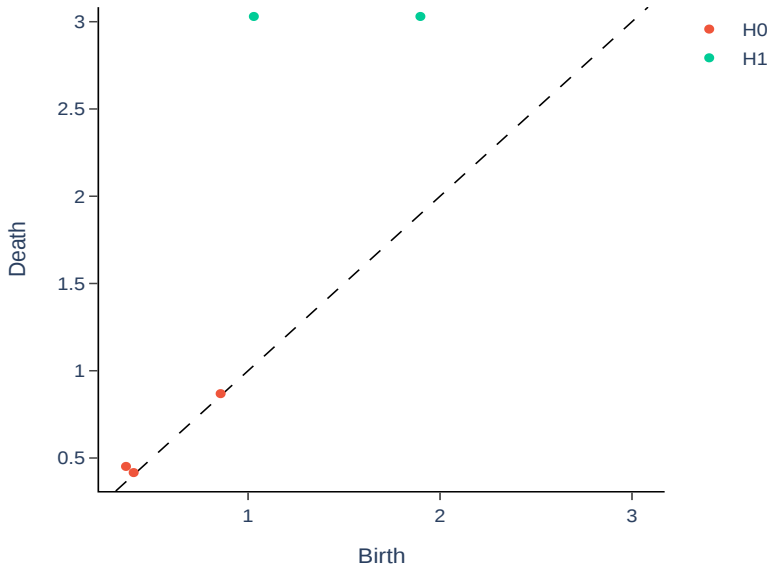


Figure 4.22: Scaled cubical persistence homology of the digit 8.

convolved the diagram with it. From this, we managed to extract a vector of amplitudes. While this was done for a single image, we may want to extract a variety of features over the whole training dataset.

To do this, we can augment our radial filtration with a *height filtration* $\mathcal{H}$, which is defined by choosing a vector $v \in \mathbb{R}^2$ representing a direction and by assigning values $\mathcal{H}(p) = \langle p, v \rangle$ based on the distance of p to the hyperplane defined by v. We will pick a uniform set of directions and centers for our filtrations, as we lack any prior information that could guide our investigation. Our method of generating features from the persistence diagrams will be what we call *persistence entropy*. This is simply the entropy of the points of the diagram. More specifically, the persistence entropy of a diagram D is defined by

$$E(D) = -\sum_{i \in I} p_i \ \log(p_i),$$

where

$$p_i = \frac{d_i - b_i}{L_D} \quad \text{and} \quad L_D = \sum_{i \in I} (d_i - b_i),$$

where d_i and b_i are the death and birth times of the equivalence classes.

Doing all of this, we managed to extract 476 topological features per image. Some of the features will be highly correlated and normally you would want to filter those out. Despite all of that, we can now happily send them into a Random Forest classifier and test our accuracy. In this example, we managed to achieve an

accuracy of 80% on our small dataset, using nothing but topological features. A better result would be achieved by applying a feature selection strategy and using the original data in the classifier as well.

As a little sidenote, we may think of the choice of direction and center as hyperparameters. We can then proceed with a greedy hyperparameter search over a grid to see what is the direction, homology direction and number of trees that give us the best accuracy. Running the search, we found that the optiomal number of trees is 500, the optimal homology dimension is H_0 and the optimal filtration direction is the vector $[1, 0]$. This intuitively makes sense when we compare the digits 6 and 9 with each other.

The notebook with the entire pipeline can be found in the *scripts/* directory under the name *Digits_classification.ipynb*.

4.3.2 Sublevelset filtration in general

Summary

Summary of the work.

Appendix A

Bibliography and sources

[AK98] E. Alpaydin and C. Kaynak. Optical Recognition of Handwritten Digits. UCI Machine Learning Repository, 1998. DOI: https://doi.org/10.24432/C50P49.

[Bor48] Karol Borsuk. On the imbedding of systems of compacta in simplicial complexes. *Fundamenta Mathematicae*, 35:217–234, 1948.

[CCSG+09] Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J Guibas, and Steve Y Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 237–246, 2009.

[Cha93] Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[DFW13] Tamal K. Dey, Fengtao Fan, and Yusu Wang. Graph induced complex on point data. 2013.

[GR09] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79:1309–1331, 2009.

[GWF14] Kristen B. Gorman, Tony D. Williams, and William R. Fraser. Ecological sexual dimorphism and environmental variability within a community of antarctic penguins (genus pygoscelis). *PLOS ONE*, 9(3):1–14, 03 2014.

[Hat02] A. Hatcher. *Algebraic Topology*. Algebraic Topology. Cambridge University Press, 2002.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and

Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[J. 99] J. R. Garcia, F. Siret, Y. Sergeaert. Kenzo - a symbolic software for effective homology computation. https://www-fourier.ujf-grenoble.fr/~sergerar/Kenzo/, 1999. [Online; accessed 25-January-2025].

[lin24] Linbox: Exact computational linear algebra. https://linalg.org/, 2024. [Online; accessed 25-January-2025].

[MHM20] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[MHSG18] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018.

[Nat24] Nathaniel Saul. Čech complex playground — sauln.github.io. https://sauln.github.io/blog/nerve-playground/, 2024. [Online; accessed 16-August-2024].

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RCTCB+24] Carlos Ramos-Carreño, José L. Torrecilla, Miguel Carbajo Berrocal, Pablo Marcos Manchón, and Alberto Suárez. scikit-fda: A Python Package for Functional Data Analysis. *Journal of Statistical Software*, 109(2):1–37, May 2024.

[SMC+07] Gurjeet Singh, Facundo Mémoli, Gunnar E Carlsson, et al. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *PBG@ Eurographics*, 2:091–100, 2007.

[ST19] Nathaniel Saul and Chris Tralie. Scikit-tda: Topological data analysis for python, 2019.

[ST20] Primoz Skraba and Katharine Turner. Wasserstein stability for persistence diagrams. *arXiv preprint arXiv:2006.16824*, 2020.

[Sus24] Sushovan Majhi. Vietoris-rips complex — smajhi.com. http://www.smajhi.com/tutorials/topology/rips.html, 2024. [Online; accessed 16-August-2024].

[TLT+20] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal Medina-Mardones, Alberto Dassatti, and Kathryn Hess. giotto-tda: A topological data analysis toolkit for machine learning and data exploration, 2020.

[VdWSNI+14] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[Vid] Vidit Nanda. Perseus, the persistent homology software. http://www.sas.upenn.edu/~vnanda/perseus. [Online; accessed 25-January-2025].

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.