

**MASARYKOVA UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
ÚSTAV MATEMATIKY A STATISTIKY**

Diplomová práce

BRNO 2025

TOMÁŠ PETIT

Topological data analysis

Diplomová práce

Tomáš Petit

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:

20 + 75

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:

20 + 75

Keywords:

Topology; Algebraic Topology; Homology; Persistent Homology; Topological data analysis; TDA; Topological Methods

Abstrakt

V této diplomové práci se věnujeme stručnému úvodu do oblasti topologické analýzy dat (TDA) a použití výpočetní topologie a geometrie. Práce je psána pro čtenáře, který nemá silné zázemí v topologii či algebře. V prvních dvou kapitolách představíme základy teorie k pochopení používaných metod. V kapitole třetí a čtvrté prezentujeme použití vysvětlených metod na příkladech v jazyce R a Python. Zdrojový kód je uveden v repozitáři práce.

Abstract

In this thesis, we introduce the reader to the fields of topological data analysis (TDA), computational topology and geometry. This work is intended for people without a strong background in topology or algebra. The first two chapters are dedicated to presenting the theory needed to interpret the results of the most used methods. Chapters 3 and 4 cover examples and practical case studies in the R and Python programming languages. The source code can be found in the repository of this work.

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 from 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 své rodině a rodičům za jejich neustálou podporu při mému studiu. Taktéž bych chtěl poděkovat vedoucímu své práce za mnohé kávy a diskuze, které proběhly během těch dvou let.

Prohlášení

Prohlašuji, že jsem svoji diplomovou práci vypracoval samostatně pod vedením vedoucího práce s využitím informačních zdrojů, které jsou v práci citovány. ■

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

Brno 4. května 2025

.....

Tomáš Petit

Contents

Introduction	1
Kapitola 1. Why Topology?	3
Kapitola 2. Simplicial Complexes and Homology	7
2.1 Simplicial complexes	7
2.2 Nerves, Čech and Rips complexes	9
2.3 Sparse complexes	11
2.3.1 Delaunay complex	11
2.3.2 Alpha complex	13
2.3.3 Graph induced complex	14
2.4 Chains, cycles and homology	14
2.4.1 Chains	15
2.4.2 Boundary operator and cycles	15
2.4.3 Cycle and boundary groups	16
2.4.4 Homology	17
2.4.5 Induced homology	18
2.4.6 Cohomology	19
2.5 Practical questions	20
2.5.1 Calculating homology	20
Kapitola 3. Topological Persistence and Persistent Homology	25
3.1 Filtration and persistence	26
3.1.1 Space filtration of topological spaces	26
3.1.2 Simplicial filtration of simplicial complexes	27
3.2 Persistence	28
3.3 Persistence diagrams	29
3.3.1 Stability of persistence diagrams under perturbation	34
Kapitola 4. Applications of TDA	41
4.1 UMAP	41
4.1.1 Penguin dataset	42
4.1.2 Digits dataset	43
4.1.3 Fashion and clothes	44

4.2 Mapper	46
4.2.1 Toy examples	50
4.2.2 Stock market data	52
4.2.3 Digits dataset, revisited	55
4.3 Persistent homology in practice	59
4.3.1 Digits, one the last time	59
4.3.2 Sublevelset filtration in general	62
4.3.3 Sliding Windows	67
Summary	75
Bibliography and sources	77

List of Figures

1.1	Example plot of seasonal temperature changes in Seattle throughout the years.	3
1.2	Time-scaled phylogenetic tree of H3 influenza viruses inferred by BEAST using molecular clock model.	4
1.3	Counting the number of blobs in each figure.	5
1.4	Data sampled from a unit circle characterized by the distinct hole in the middle.	5
2.1	From the left: a 0-simplex, a 1-simplex and a 2-simplex	8
2.2	A simplicial complex with 4 vertices, 4 edges and 1 triangle.	8
2.3	An example of a space M , its open cover \mathfrak{U} and its nerve $N(\mathfrak{U})$	9
2.4	On the left, the VR complex of three pairwise intersecting circles. On the right, the corresponding Čech complex.	10
2.5	An illustration of the growth of the VR complex when we increase the number of sampled points.	11
2.6	Delaunay complex on a sample of randomly generated points in \mathbb{R}^2	12
2.7	Voronoi diagram on a sample of randomly generated points dual to 2.6.	13
2.8	Cochain complex.	20
2.9	Chain complex example	21
3.1	A random sample of points from two touching circles and the growing sublevel sets of the distance function to the points.	25
3.2	A random sample drawn from the unit circle and one drawn from the same circle with additional Gaussian noise.	30
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.	32
3.4	The corresponding representation of a persistence barcode next to its persistence diagram.	33
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.	33
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.	36

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.	37
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.	38
3.9 The Hausdorff distance is determined by the point in A with the largest distance to the closest point in B (and vice versa).	39
4.1 Pairwise plots of the Penguin dataset for the bill length, bill depth, flipper length and body mass variables.	44
4.2 UMAP dimension reduction result on the Penguin dataset.	45
4.3 First few images of the digits dataset	45
4.4 Result of UMAP on the digits dataset.	46
4.5 Result of UMAP on the much larger Fashion-MNIST dataset.	47
4.6 Points of connectivity of Fashion-MNIST.	47
4.7 A less busy view of connectivity of Fashion-MNIST.	48
4.8 Our toy example for now - two nested circles.	50
4.9 First result of applying Mapper.	51
4.10 Nodes colored by the average x -coordinate.	51
4.11 Nodes colored by the average first PCA component.	52
4.12 Nodes colored by categorical variables.	53
4.13 The effect of a different filtration function; in this case, the sum of the individual coordinates.	54
4.14 View of the Mapper clustering summary with <i>scikit-tda</i>	55
4.15 Mapper graph of the SP500 data with the color function being the volatility of the stock.	56
4.16 Cluster of tickers with the lowest maximal volatility.	57
4.17 Cluster of tickers with the highest maximal volatility.	57
4.18 Stock prices of the company HP during 2024-2025.	58
4.19 Mapper graph of the digits dataset with a custom tooltip function. .	58
4.20 Mapper graph of the digits dataset with labels.	59
4.21 Cubical persistence homology of the digit 8.	61
4.22 Scaled cubical persistence homology of the digit 8.	61
4.23 The periodic signal $\cos(2\pi t) + t$	63
4.24 The sublevelset filtration of the signal.	64
4.25 Image containing some Guassian blobs.	65
4.26 The result of taking the sublevelset filtration of the Gaussian blobs. .	65
4.27 Example image of wood cells.	66
4.28 Persistence diagram of the grayscale image of the cells.	66
4.29 Smoothed out version of the grayscale image of the cells.	67
4.30 Persistence diagram of the smoothed out image.	68

4.31	Filled out image of cells.	68
4.32	Persistence diagram of a black hole.	69
4.33	Ordinary cosine wave	70
4.34	Quasi-periodic wave	70
4.35	The corresponding persistence diagrams	71
4.36	Optimal embeddings, after a reduction with PCA	72
4.37	Persistence diagrams of the optimal embeddings	72
4.38	The noisier our signal is, the less informative the geometry of its embedding and the persistence diagram becomes.	73

List of Tables

4.1	First five rows of the Penguin dataset.	43
-----	---	----

Introduction

Any thesis or research should answer one question before it is published, before it is even written – Who is the intended audience? Knowing the target audience determines the structure and interpretation of the work itself. We can answer that question right now – the ideal reader is someone working in statistics, data analysis, biology, or any of the natural sciences who is interested in using a geometrical approach to a problem they might be looking at. People working in those areas usually do not have a rich background in algebraic topology or even topology itself. As such, we have made the decision to cut costs and introduce the basic theory needed to interpret the results of computing the persistent homology of some object and how to work with those results further. This means that proofs are largely omitted and only referenced in textbooks or articles that the reader may wish to explore. The point of this work is to introduce someone from a non-topological field to Topological Data Analysis and teach them how to work with the methods, not drown them in technical details. We will primarily focus on three fundamental methods of TDA (Topological Data Analysis) – UMAP, Mapper and Persistent Homology. Due to the amount of technical details and theory needed to understand Persistent Homology, two chapters are dedicated to it, while UMAP and Mapper can be understood easily. Multiple Jupyter Notebooks and scripts are included for the reader to play with, experiment with, familiarise themselves with the current state-of-the-art libraries, and potentially use those methods in their own research in the future.

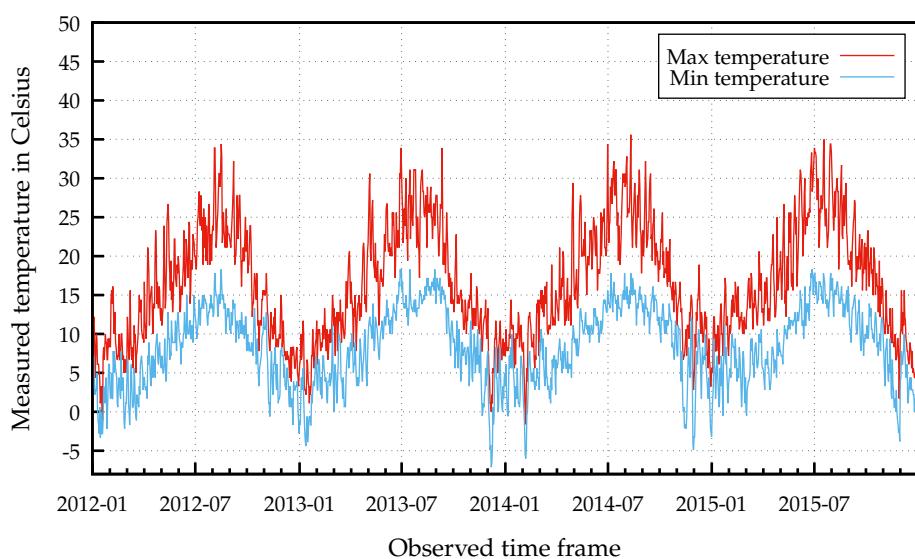
Chapter 1

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.

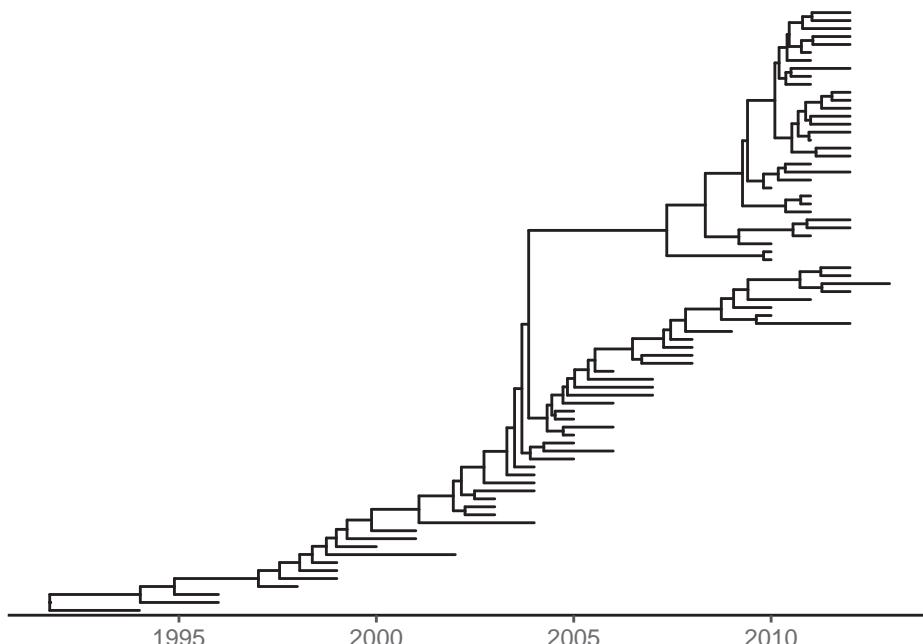


Clustering algorithms – be it k-means, hierarchical clustering, and so on –

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

¹Data 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.

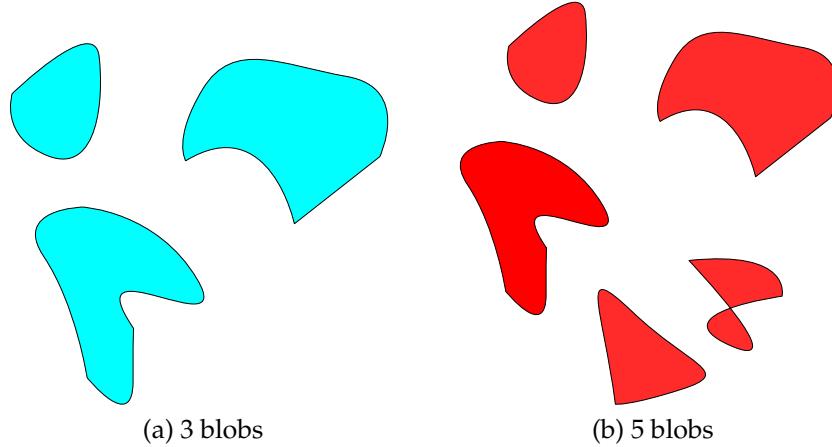
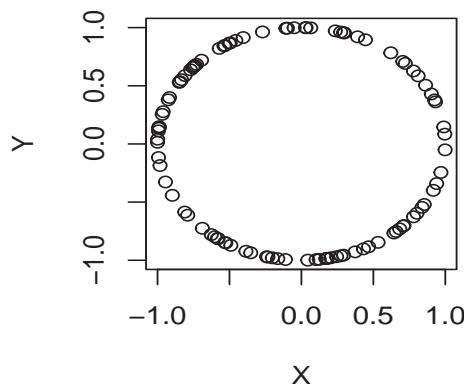


Figure 1.3: Counting the number of blobs in each figure.

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

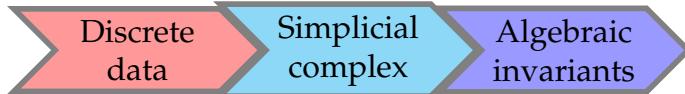
call TDA – Topological Data Analysis. The principal sources for the theoretical part of this work are the books [DW22] and [RB19].

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:

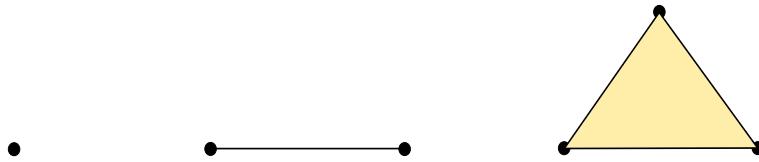


Figure 2.1: From the left: a 0-simplex, a 1-simplex and a 2-simplex

- 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.

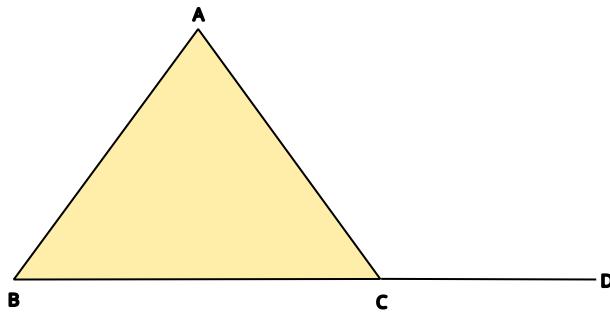


Figure 2.2: A simplicial complex with 4 vertices, 4 edges and 1 triangle.

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 $e : V(K') \rightarrow \mathbb{R}^n$, that takes every k -simplex $\{v_0, \dots, v_k\}$ in K' to a k -simplex in K that is the convex hull of $e(v_0), \dots, e(v_k)$. 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 $\{\alpha_0, \dots, \alpha_k\} \subseteq A$ spans a k -simplex in $N(\mathfrak{U})$ if and only if $U_{\alpha_0} \cap \dots \cap U_{\alpha_k} \neq \emptyset$.

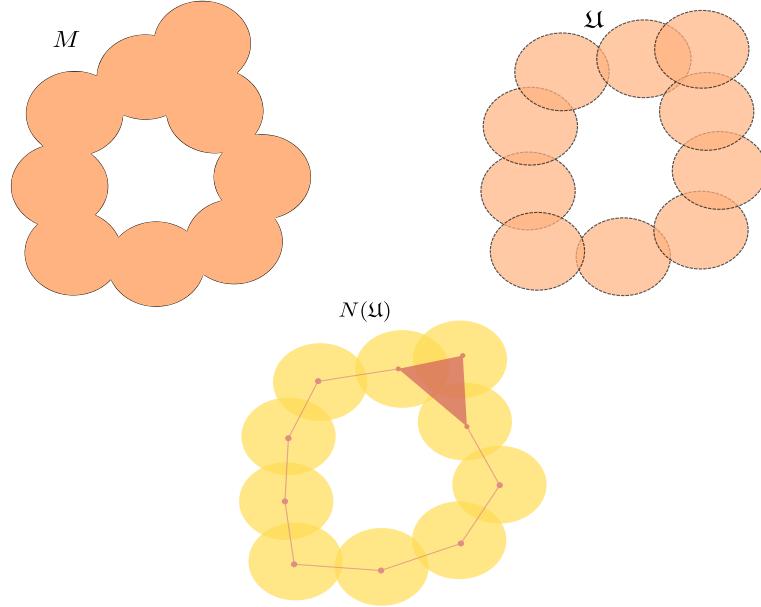


Figure 2.3: An example of a space M , its open cover \mathfrak{U} and its nerve $N(\mathfrak{U})$.

The following important theorem tells us when the nerves are “equivalent” to the original space. There are various formulations 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 for all $p_i \in P$

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

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 .

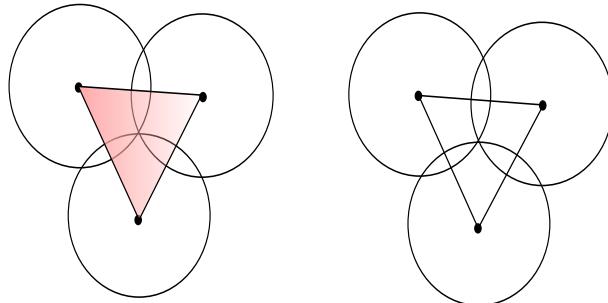


Figure 2.4: On the left, the VR complex of three pairwise intersecting circles. On the right, the corresponding Čech complex.

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

$$\mathbb{C}^\varepsilon(P) \subseteq \mathbb{VR}^\varepsilon(P) \subseteq \mathbb{C}^{2\varepsilon}(P). \quad (2.1)$$

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

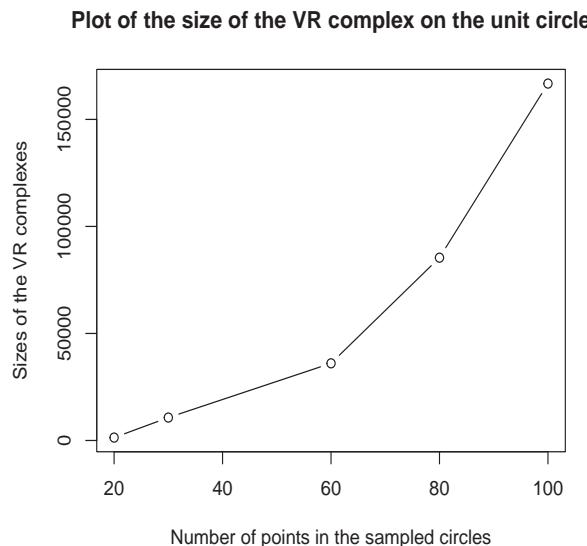


Figure 2.5: An illustration of the growth of the VR complex when we increase the number of sampled points.

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 its vertices and is empty, containing no point in P . A *Delaunay complex* of P , denoted by $\text{Del } P$, is the geometric simplicial complex with vertices in P , in which every simplex is Delaunay and the underlying space $|\text{Del } P|$ coincides with the convex hull of P .

¹Technically, 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.

In \mathbb{R}^2 , a Delaunay complex is made out of Delaunay triangles and all of their lower-dimensional faces. In \mathbb{R}^3 , a Delaunay complex is made out of Delaunay tetrahedra and all of their lower-dimensional faces and so forth.

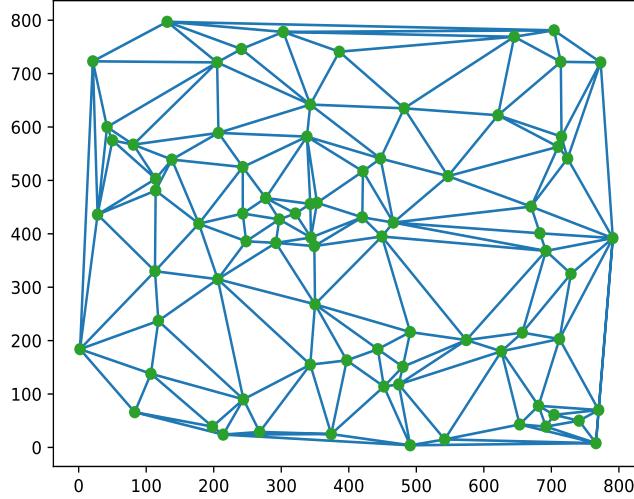


Figure 2.6: Delaunay complex on a sample of randomly generated points in \mathbb{R}^2 .

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.

Definition 2.3.2 (Voronoi diagram). Given a finite point set $P \subset \mathbb{R}^n$ in generic position, the Voronoi diagram $\text{Vor}(P)$ 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 $\text{Vor}(P)$ 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 $P \subset \mathbb{R}^n$, $\text{Del}(P)$ 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 $\text{Del}(P)$ is dual to a Voronoi $(d - k)$ -face in $\text{Vor}(P)$. 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 $P \subset \mathbb{R}^n$, $\text{Del}(P)$ achieves the following:

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

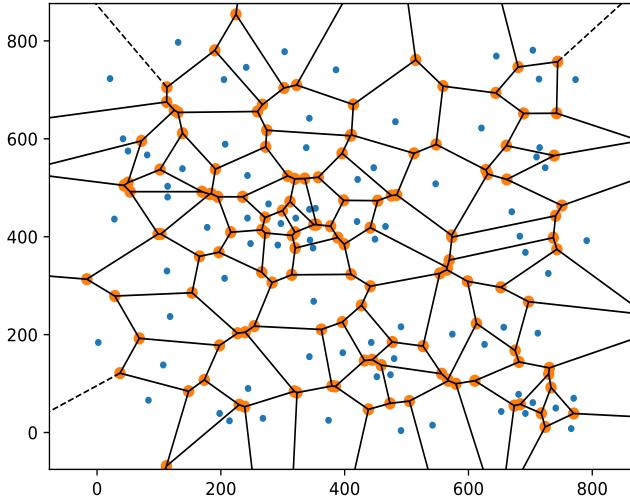


Figure 2.7: Voronoi diagram on a sample of randomly generated points dual to 2.6.

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

2.3.2 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 $\text{Del}(P)$ 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 $\sigma = \{p_0, \dots, p_k\}$, if and only if $\bigcup_{p \in P} B(p, \alpha)$ meets the intersection of Voronoi cells $V_{p_0} \cup \dots \cup V_{p_k}$.

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 $\nu : P \rightarrow Q$ the mapping that sets $\nu(p)$ to be any point in $\operatorname{argmin} d(p, Q)$. The *graph induced complex* (GIC) $\mathbb{G}(G(P), Q, d)$ is the simplicial complex containing a k -simplex $\sigma = \{q_1, \dots, q_{k+1}\}$, $q_i \in Q$, if and only if there exists a $(k+1)$ -clique in $G(P)$ spanned by vertices $\{p_1, \dots, p_{k+1}\} \subseteq P$ so that $q_i \in \nu(p_i)$ for each $i \in \{1, 2, \dots, k+1\}$.

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 maintain 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 future³; who knows?

³It turns out that this was a goal too ambitious for the author to undertake.

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 and most textbooks, we will mainly consider coefficients coming from some field \mathbf{k} , more specifically from $\mathbf{k} = \mathbb{Z}$. This means that, for example, adding two 1-chains works in the following natural manner:

$$(2e_1 + 3e_2 + 9e_4) + (e_1 + 10e_2 + 6e_5) = 3e_1 + 13e_2 + 9e_4 + 6e_5,$$

where the simplices with zero coefficients are ignored.

It should be noted that most software packages will work with coefficients in the field \mathbb{Z}_2 , even allowing the user in some cases to provide their own field of choice, if needed. 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 $1 + 1 = 0$.

p -chains additionally form a group, a free abelian group to be precise, that we call 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 $\sigma = (v_0, \dots, v_p)$, let

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

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

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

A couple special 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 $p > k$. 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]. \square

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

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

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 see that after applying the boundary operator to the 2-chain (a, b, c) , the result is

$$\partial_2(abc) = bc - ac + ab.$$

This 1-chain is a 1-cycle since

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

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 $(p-1)$ -st boundary group $B_{p-1} = \text{im } \partial_p(C_p)$. This is equivalent to saying that $B_{p-1} = \text{im } \partial_p$. It is straightforward to see 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 $H_p = Z_p / B_p$. We call its rank the p -th Betti number

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

The equivalence classes are called *homology classes* in this context. Two cycles $c, c' \in Z_p$ are in the same homology class, iff $c \in c' + B_p$.

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, $\beta_p = \text{rank } H_p$, is given by $\beta_p = \text{rank } Z_p - \text{rank } B_p$.
- There are exactly 2^{β_p} homology classes in H_p when defined over \mathbb{Z}_2 .

The reader should be made aware of the fact that when we talk about homology, we are actually talking about a *homology theory*. The most familiar usage of it is for the homology of a topological space. In its generality, homology is a sequence of functors satisfying a collection of axioms.⁴ The important property is that any homology theory satisfying these axioms will yield the same homology groups. This makes it that we usually don't specify which homology theory we have used, unless needed. In our case, the two theories we will work with – simplicial homology and its general version, singular homology – are equivalent, for all our purposes.

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 $H_0(X)$ describes the path-connected components in X . This turns out to be a 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 singular homology groups as

⁴The interested reader might want to search for *Eilenberg-Steenrod axioms*.

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

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. If we wanted to compute its simplicial homology, we would replace the sphere S^1 with a triangle, approximating the continuous structure with a discrete approximation. 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

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

Since computers can only ever approximate a continuous medium with a discrete version, all software packages and computations are done using simplicial homology. As mentioned above, this isn't a loss due to the axiomatic nature of homology.

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. Simplicial maps take cycles to cycles and boundaries to boundaries, which suggests the following definition.

Definition 2.4.2 (Chain map). Let $f : K_1 \rightarrow K_2$ be a simplicial map. The chain map $f_{\#} : C_p(K_1) \rightarrow C_p(K_2)$ 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 $f : K_1 \rightarrow K_2$ 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:

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

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

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

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

Lemma 2.4.3. For two contiguous maps $f_1 : K_1 \rightarrow K_2$ and $f_2 : K_1 \rightarrow K_2$, the induced maps $f_{1*} : H_p(K_1) \rightarrow H_p(K_2)$ and $f_{2*} : H_p(K_1) \rightarrow H_p(K_2)$ are equal, where two simplicial maps $f_1 : K_1 \rightarrow K_2$, $f_2 : K_1 \rightarrow K_2$ are contiguous if for every simplex $\sigma \in K_1$, $f_1(\sigma) \cup f_2(\sigma)$ is a simplex in K_2 .

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 libraries use homology groups for their calculations.

A p -cochain is a homomorphism $\phi : C_p \rightarrow \mathbb{Z}$ from the chain group to the coefficient ring \mathbb{Z} . 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

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

Similarly to chain boundaries, there is the notion of a cochain coboundary $\delta_p : C^p \rightarrow C^{p+1}$. For a p -cochain ϕ , its $(p+1)$ -coboundary is given by the homomorphism $\delta\phi : C^{p+1} \rightarrow \mathbb{Z}$ defined as $\delta\phi(c) = \phi(\partial c)$ for any $(p+1)$ -chain c . Repeated iteration of the δ operator gives us the sequence in 2.8.

p -coboundaries form the coboundary group B^p with addition and scalar multiplication being the same as in C^p . Finally, we only need cocycles to define what

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

Figure 2.8: Cochain complex.

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 Z^p with the operations induced by C^p once again.

The coboundary operator δ also satisfies the same property as the boundary ∂ , that is, $\delta\phi \circ \delta\phi = 0$.

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 $f : K_1 \rightarrow K_2$ induces a homomorphism f^* between cohomology groups; this time in the *opposite* direction.

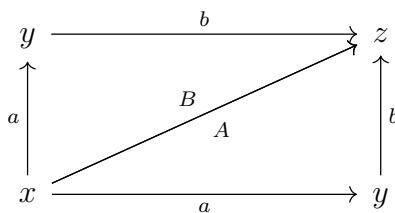
Lemma 2.4.5. A simplicial map $f : K_1 \rightarrow K_2$ induces a homomorphism $f^* : H^p(K_2) \rightarrow H^p(K_1)$ for every $p \geq 0$.

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 can be described, for example, by 2.9.

Our goal will be to compute the simplicial homology group H_0 of this simplicial complex, and due to the equivalence of the homology theories we are working

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

Figure 2.9: Chain complex example

with, 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 $\text{Im } \partial_1$ is. For that, the behaviour of ∂_1 on any 1-chain $la + mb + nc$ is needed.

Following the properties of the maps ∂_1 , 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 transformation

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

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 $\text{Im } \partial_1$

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

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 $q : \text{Ker } \partial_1 \rightarrow H_0$ to behave as follows:

$$\begin{aligned}q(y - x) &= q(y) - q(x) = 0 \implies q(y) = q(x) \\ q(z - y) &= q(z) - q(y) = 0 \implies q(z) = q(y)\end{aligned}$$

To obtain the quotient, we first replace 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 replace z with y :

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

Therefore $H_0 \cong \mathbb{Z}$ 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

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

Furthermore, 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

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

where $r = \text{rank}(A)$, $s = \text{rank}(B)$ 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 additionally 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 live. 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 sampled 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 $\{x_1, \dots, x_k\}$ be a set of points drawn from M according to the volume measure. Fix $0 < \varepsilon < \tau/2$. Then if

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

there is a homotopy equivalence

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

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

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

and

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

where $\theta_1 = \arcsin(\frac{\varepsilon}{8\tau})$, $\theta_2 = \arcsin(\frac{\varepsilon}{16\tau})$ and $\text{vol}(B_r^n)$ 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.

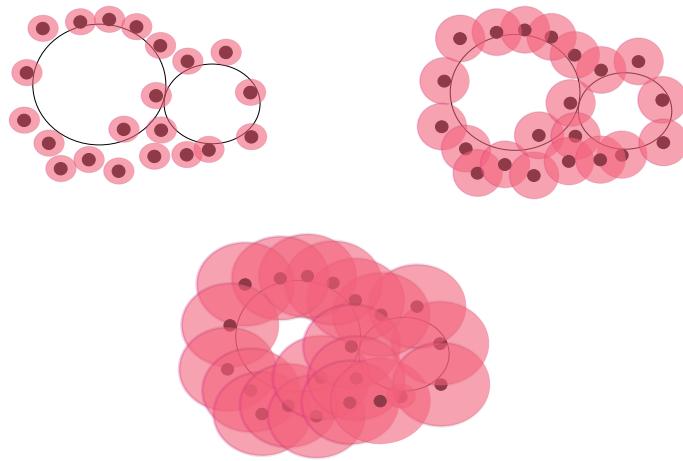


Figure 3.1: A random sample of points from two touching circles and the growing sublevel sets of the distance function to the points.

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 $r : \mathbb{R}^2 \rightarrow \mathbb{R}$, 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.

3.1 Filtration and persistence

3.1.1 Space filtration of topological spaces

Let $f : \mathbb{T} \rightarrow \mathbb{R}$ 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 $a_1 \leq a_2 \leq \dots \leq a_n$ to create a sequence of nested subspaces of \mathbb{T}

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

We call this sequence a *filtration* \mathcal{F}_f . Ideally, the points $a_1 \leq \dots \leq a_n$ 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 *a priori* 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 $\iota : \mathbb{T}_{a_i} \rightarrow \mathbb{T}_{a_j}$, $i \leq j$, the inclusion map $x \mapsto x$, we have an induced homomorphism

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

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*:

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

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 $\text{Im } h_p^{i,j}$.

3.1.2 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

$$\mathcal{F} : \emptyset = K_0 \subseteq K_1 \subseteq \dots \subseteq K_n = K,$$

or equivalently

$$\mathcal{F} : \emptyset = K_0 \hookrightarrow K_1 \hookrightarrow \dots \hookrightarrow K_n = K.$$

The filtration \mathcal{F} is called *simplex-wise* if either $K_i \setminus K_{i-1}$ 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 $f : K \rightarrow \mathbb{R}$. 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:

$$\emptyset = K_0 \hookrightarrow K_1 \hookrightarrow \dots \hookrightarrow K_n = K.$$

A *vertex function* $f : V(K) \rightarrow \mathbb{R}$ 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 3.1.1 (PL-functions). Given a simplicial complex K , a *piecewise-linear* (PL) function $f : |K| \rightarrow \mathbb{R}$ is defined to be the linear extension of a vertex function $f_V : V(K) \rightarrow \mathbb{R}$ 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 $\sigma = (v_1, \dots, v_{k+1})$ is the unique lowest dimensional simplex of dimension $k \geq 0$ containing x and $\alpha_1, \dots, \alpha_{k+1}$ 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 3.1.2 (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 $\sigma \in (K_i \setminus K_{i-1})$ for $K_{i-1} \neq K_i$ is given the value $f(\sigma) = i$.

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

3.2 Persistence

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

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

with the maps $h_p^{i,j} : H_p(X_i) \rightarrow H_p(X_j)$ 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 3.2.1 (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 $\beta_p^{i,j} = \dim H_p^{i,j}$.

Catching when a class is born or dies requires precision since when a new class is born, many other classes that are sum of this new 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 3.2.1. $H_p^{i,j} = Z_p(X_i)/(B_p(X_j) \cap Z_p(X_i))$ and $\beta_p^{i,j} = \dim H_p^{i,j}$.

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 3.2.2 (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 3.2.1. 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 $i_1 \leq i_2 \leq \dots \leq i_k = i$ 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 $l \in \{1, \dots, k\}$ and $[c] = [c_{i_1}] + \dots + [c_{i_k}]$.

This may be interpreted in the sense that the death of a homological class can be thought of as the merging of several classes, the youngest one signaling the birth point. For each X_i , $i = 0, \dots, n$, 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.

3.3 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 classes that never die. For this case, we extend the module on the right by declaring $H_p(X_{n+1}) = 0$.

Definition 3.3.1. 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 3.3.2 (Class persistence). For $\mu_p^{i,j} \neq 0$, the persistence $\text{Pers}([c])$ of a class $[c]$ that is born at X_i and dies at X_j is defined as $\text{Pers}([c]) = a_j - a_i$. When $j = n + 1 = \infty$, $\text{Pers}([c])$ equals $a_{n+1} - a_i = \infty$.

Definition 3.3.3 (Persistence diagram). The persistence diagram $\text{Dgm}_p(\mathcal{F}_f)$ (or $\text{Dgm}_p f$) 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.

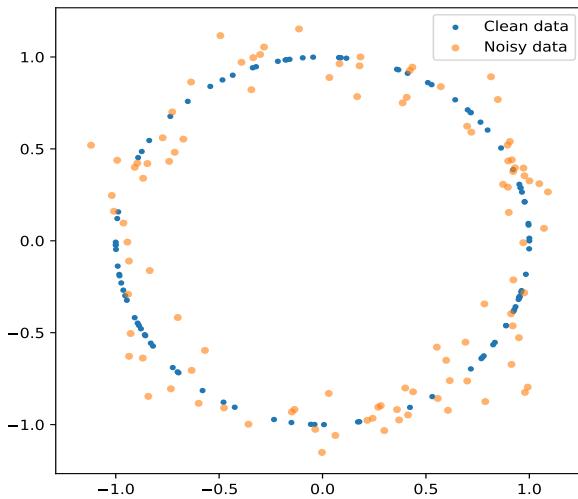


Figure 3.2: A random sample drawn from the unit circle and one drawn from the same circle with additional Gaussian noise.

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* [PVG¹¹] (general machine learning algorithms), *scikit-image* [VdWSNI¹⁴] (image processing algorithms) or *scikit-fda* [RCTCB²⁴] (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 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 of length $a_j - a_i$, plotted against its length, called a bar. One such example can be seen on 3.4.

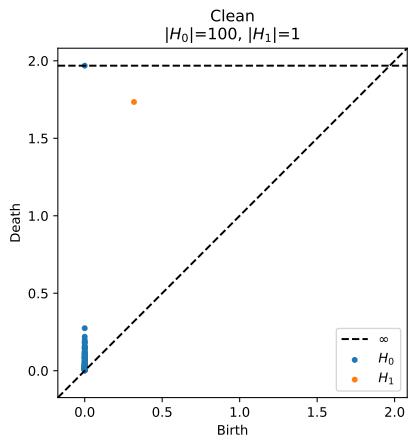
Keep in mind that the barcodes do not contain more information than their diagram – they are equivalent representations of the same fact. They *can* be used in further analysis in different manners, but they do not fundamentally reveal anything new that wasn't already obtained. We don't care about any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

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.

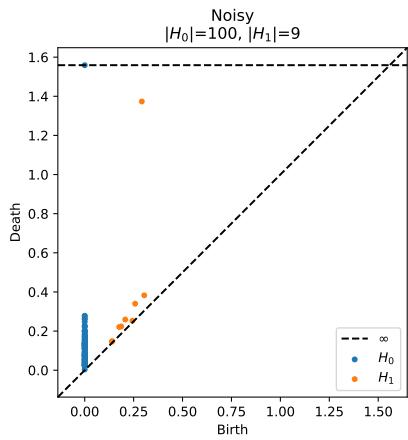
Dendrograms will add the “global” cluster to which all the points will belong after a certain height. This is ignored in the barcodes, since no additional homology classes are either born or die after this points. Thus, in 3.5, the barcodes end with time 10, corresponding to the height 10 in the dendrogram.

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

Theorem 3.3.1. For every pair of indices $0 \leq k \leq l \leq n$ and every p , the p -th



(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.

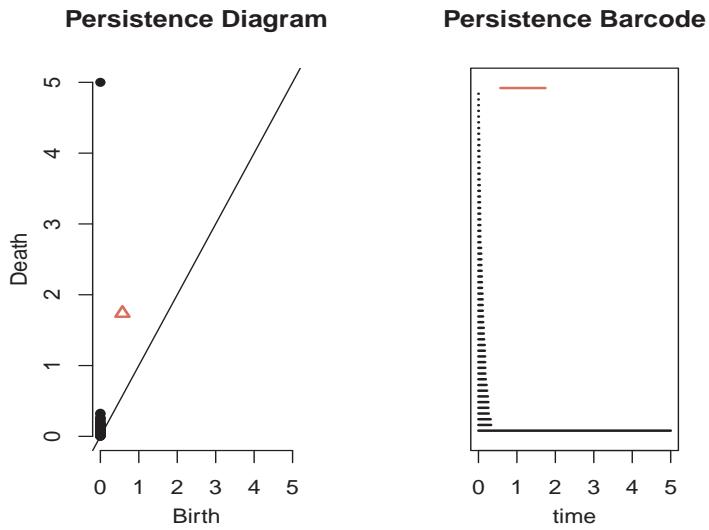


Figure 3.4: The corresponding representation of a persistence barcode next to its persistence diagram.

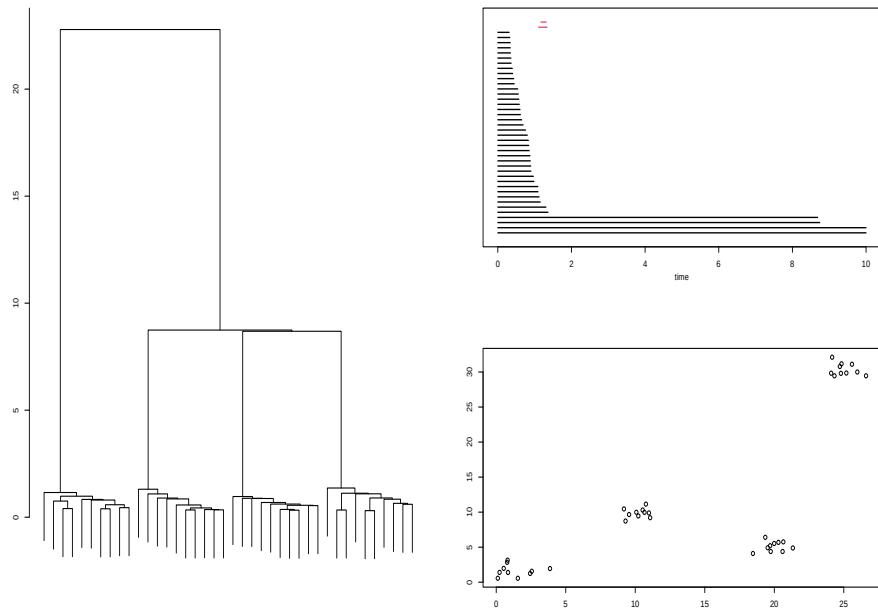


Figure 3.5: The computation of persistence barcodes for $p = 0$ is equivalent to single-linkage clustering of the data. Looking at the count of bars at each time and the dendrogram heights, we see that they keep track of the same information.

persistent Betti number satisfies $\beta_p^{k,l} = \sum_{i \leq k} \sum_{j > l} \mu_p^{i,j}$.

3.3.1 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 3.3.4 (Bottleneck distance). Let $\Pi = \{\pi : \text{Dgm}_p(\mathcal{F}_f) \rightarrow \text{Dgm}_p(\mathcal{F}_g)\}$ be the set of all bijections between the persistence diagrams. The bottleneck distance between two diagrams is defined as

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

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 3.3.2. Let $f, g : K \rightarrow \mathbb{R}$ 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 filtrations of general topological spaces (and hence singular homology), 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 $f : X \rightarrow \mathbb{R}$ *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 3.3.3. Let X be a triangulable space. Let $f, g : X \rightarrow \mathbb{R}$ 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 3.3.5 (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

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

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

Theorem 3.3.4. Let $f, g : X \rightarrow \mathbb{R}$ 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 3.3.5. Let $f, g : K \rightarrow \mathbb{R}$ 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

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

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 perfect 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.

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.

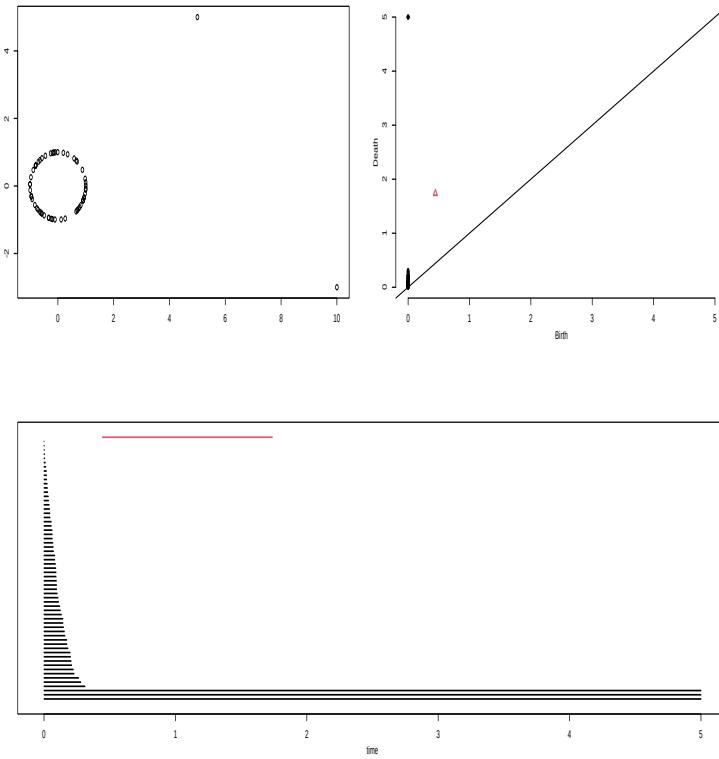


Figure 3.6: Adding two additional points far away from the rest is correctly represented in the barcodes with three long bars representing three connected components. This fact is not as easily seen in the diagram.

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.

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

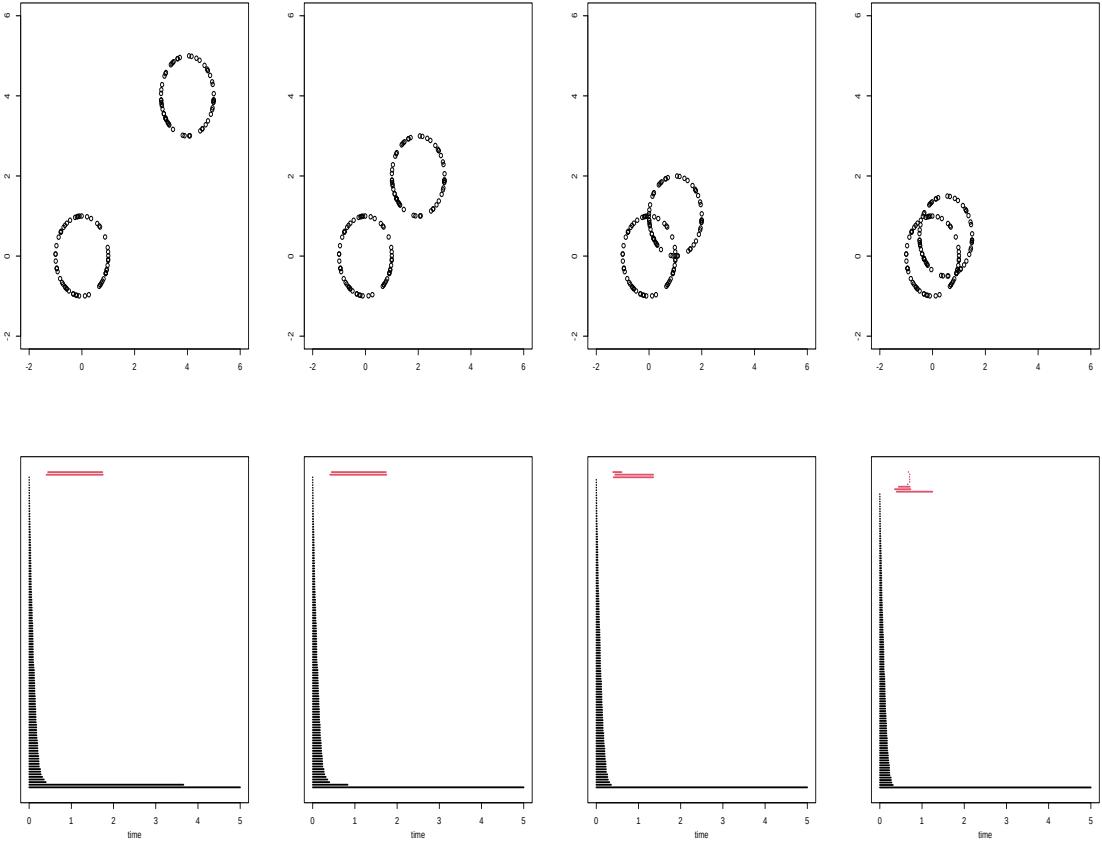


Figure 3.7: With two disjoint circles slowly approaching each other, we can see how the barcodes evolve. A non-trivial intersection causes a third bar to appear, representing the given loop. Moreover, multiple smaller bars generated by small loop show up the closer the circles are.

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 remedy 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.

Definition 3.3.7. Let (P, d_P) and (Q, d_Q) be compact metric spaces. The *Gromov-Hausdorff* distance between P and Q is defined as

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

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

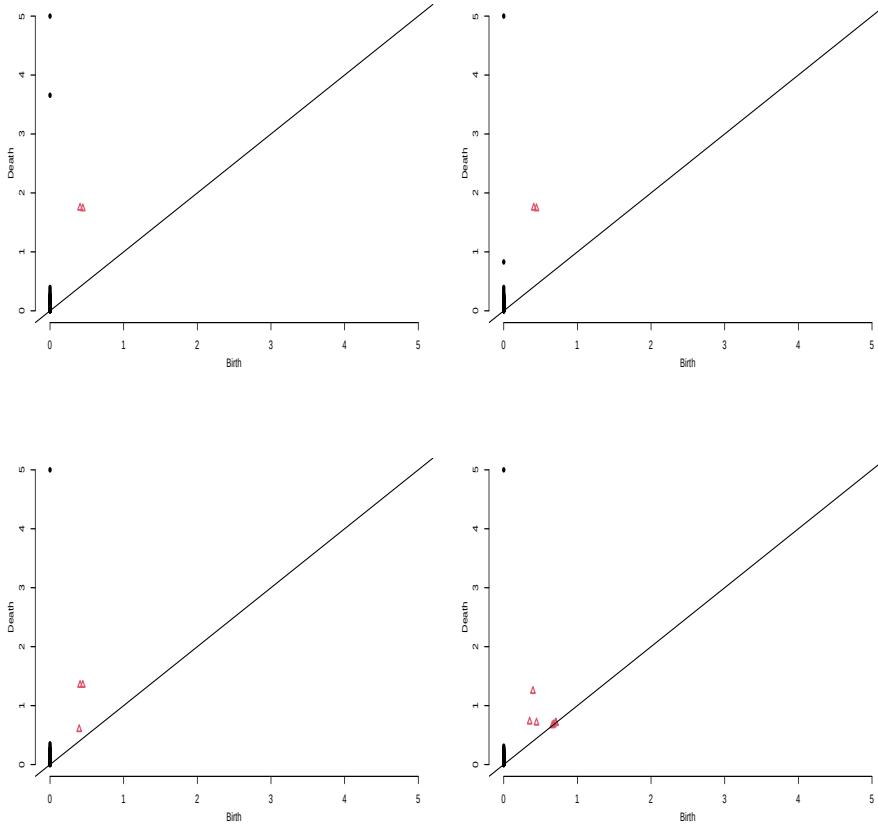


Figure 3.8: With two disjoint circles slowly approaching each other, we can see how the barcodes evolve. A non-trivial intersection causes a third bar to appear, representing the given loop. Moreover, multiple smaller bars generated by small loop show up the closer the circles are.

The 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 component 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.

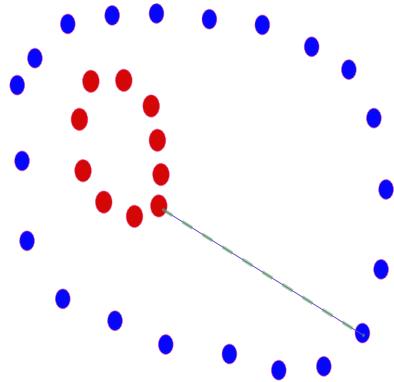


Figure 3.9: The Hausdorff distance is determined by the point in A with the largest distance to the closest point in B (and vice versa).

All the versions have a common generalization in the algebraic stability theorem, which, ignoring technical details, says that for persistence technical modules that are κ -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 popular methods, techniques, and algorithms that are used within Topological Data Analysis. The most theoretically demanding one is Persistent Homology which 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 and entropy. We will demonstrate them in the classical setting of point clouds but also in 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 too. 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.

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.

Let the input be $X = \{X_1, \dots, X_N\}$. 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 is beneficial to assume that the data is uniformly distributed on the manifold, with results being only valid in the limit of infinite data if the assumption isn't made. If we assume that the manifold has

a Riemannian metric not inherited from its ambient space, we can always find a metric for which the data is approximately uniformly distributed with regard to it. The next steps are only possible due to the following theoretical result:

Lemma 4.1.1. Let (\mathcal{M}, g) be a Riemannian manifold in an ambient \mathbb{R}^n , and let $p \in \mathcal{M}$ be a point. If g is locally constant about p in an open neighbourhood U such that g is a constant diagonal matrix in ambient coordinates, then in a ball $B \subseteq U$ centered at p with volume $\frac{\pi^{n/2}}{\Gamma(n/2+1)}$ with respect to g , the geodesic distance from p to any point $q \in B$ is $\frac{1}{r}d_{\mathbb{R}^n}(p, q)$, where r is the radius of the ball in the ambient space and $d_{\mathbb{R}^n}$ is the existing metric on the ambient space.

The proof of this can be found in [MHM20], Appendix A. Assuming uniform distribution, any ball of fixed volume will contain roughly the same number of points of X , regardless of the position of the center of the ball. Conversely, any ball centered at X_i containing exactly the k -nearest neighbours of X_i should roughly have the same fixed volume, regardless of $X_i \in X$. Thanks to the lemma above, we can approximate the geodesic distance from X_i to its neighbours by normalising distances with respect to the distance to the k^{th} nearest neighbor of X_i .

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 $\mu : A \rightarrow [0, 1]$ 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 categorical¹ 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 the 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.

¹Categorical in this case means as in terms of category theory.

	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.

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 matrix² 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 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 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 handwritten 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 surprise that fairly distinct digits like 0, 9, 2, 6, or 7 are far away from the rest, whereas digits 1, 8 and

²A pairwise scatter plot consists of plotting the Cartesian product of all variables against each other. When plotting one variable against itself, it instead produces a density approximation.

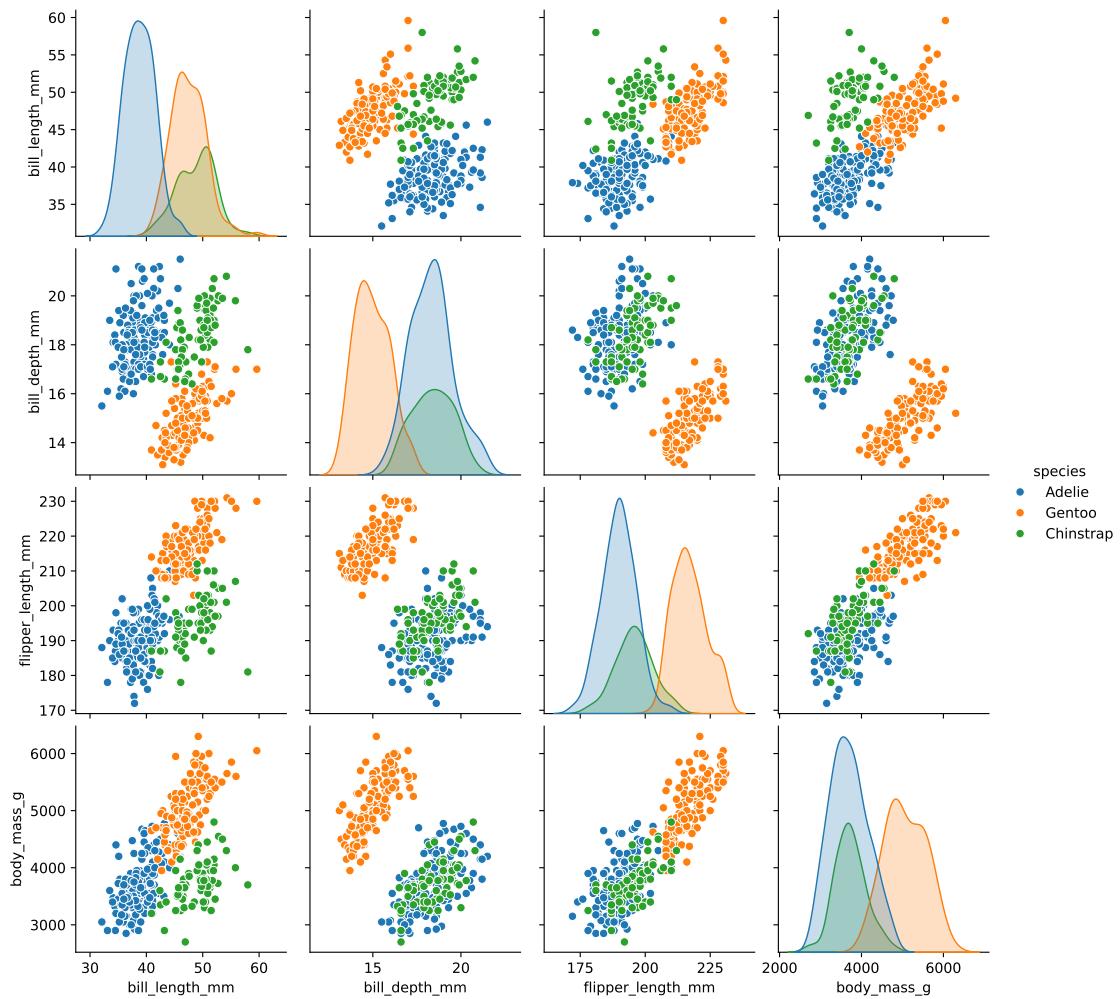


Figure 4.1: Pairwise plots of the Penguin dataset for the bill length, bill depth, flipper length and body mass variables.

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.

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 follows, labeled 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

UMAP projection of the Penguin dataset

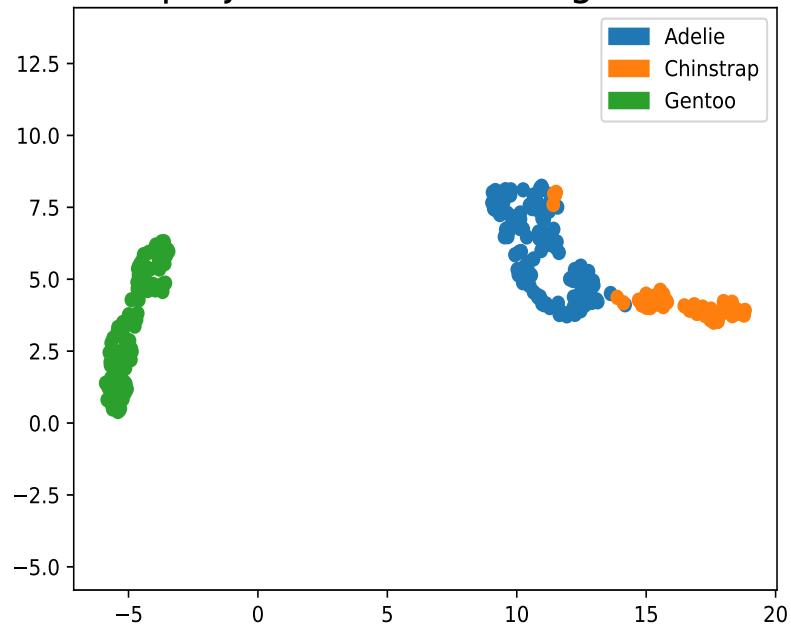


Figure 4.2: UMAP dimension reduction result on the Penguin dataset.

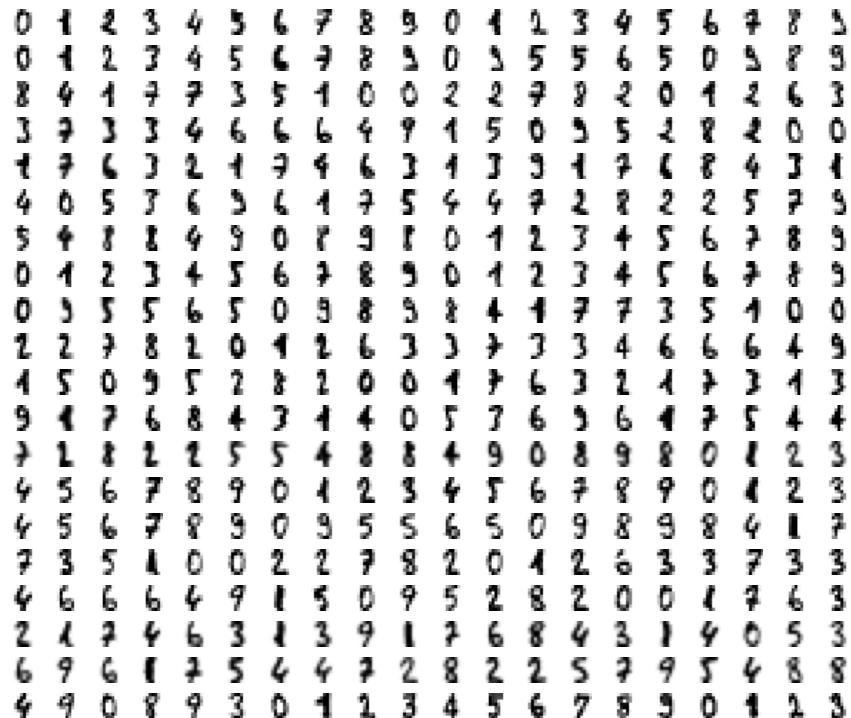


Figure 4.3: First few images of the digits dataset

UMAP projection of the Digits dataset

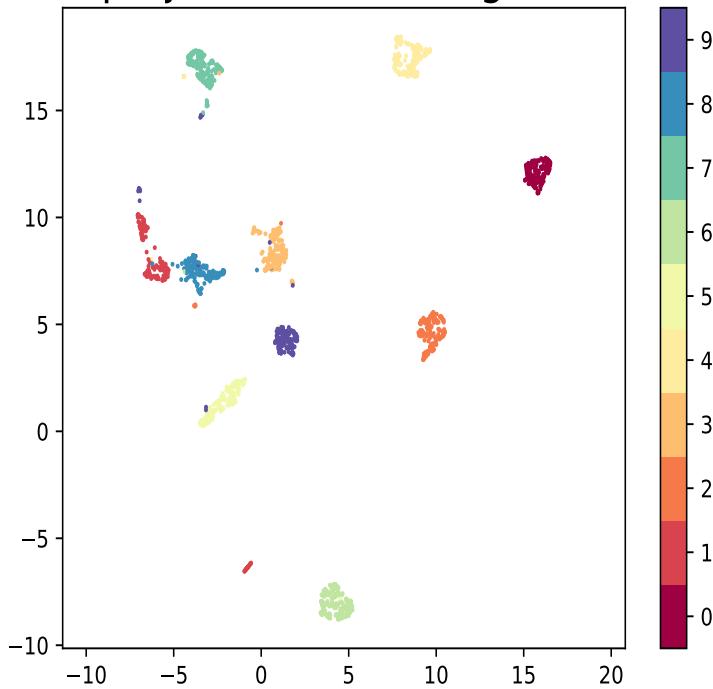


Figure 4.4: Result of UMAP on the digits dataset.

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 studying 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 prac-

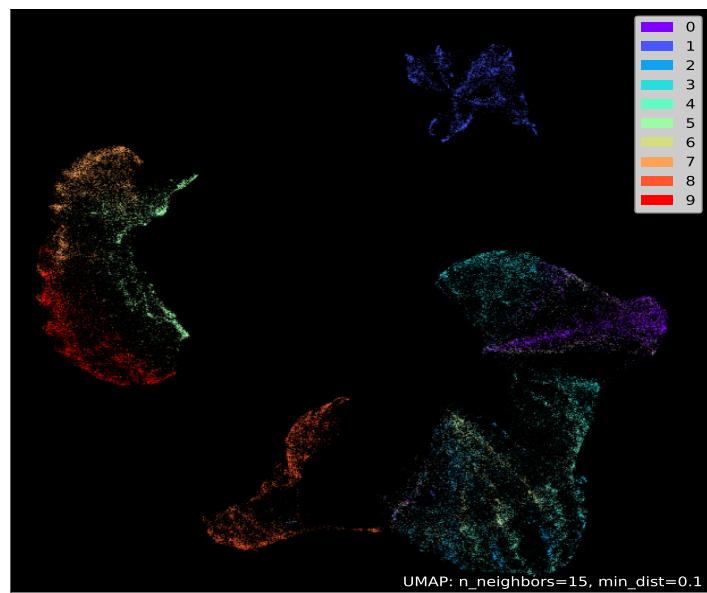


Figure 4.5: Result of UMAP on the much larger Fashion-MNIST dataset.

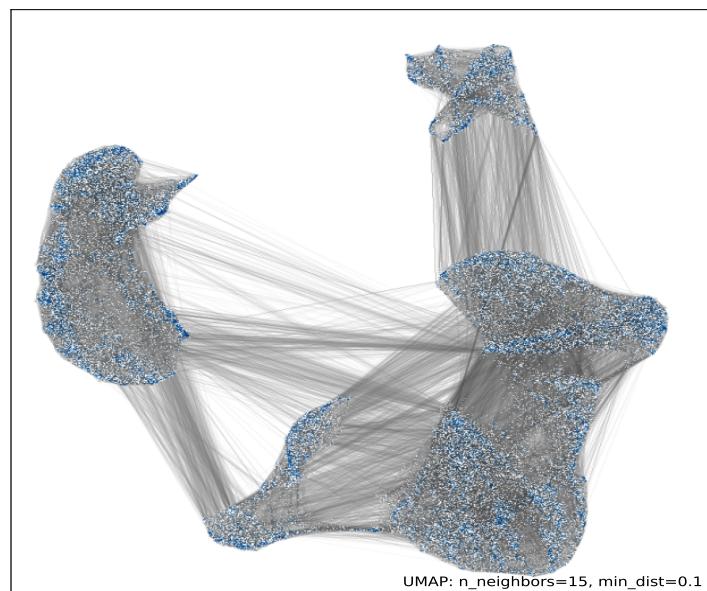


Figure 4.6: Points of connectivity of Fashion-MNIST.

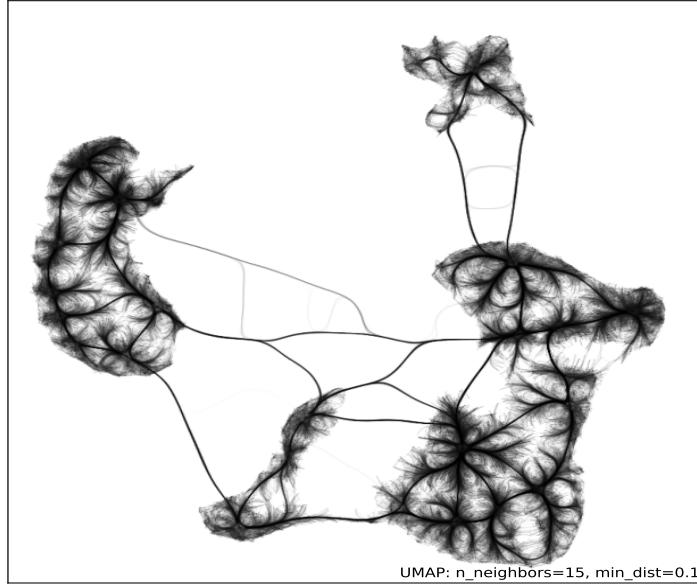


Figure 4.7: A less busy view of connectivity of Fashion-MNIST.

tically 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* $f : P \rightarrow \mathbb{R}^n$
- 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

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

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 to 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 successfully captured the salient features - those being the two holes in the nested circles. The coloring of the node used the default settings of *giootto-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.

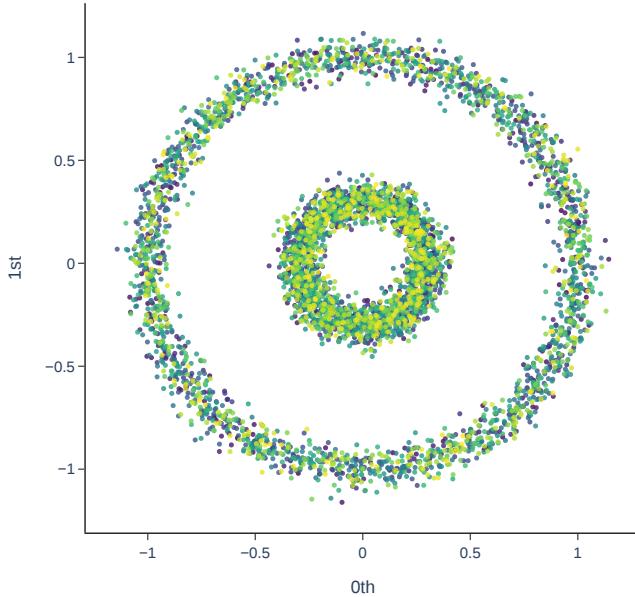


Figure 4.8: Our toy example for now - two nested circles.

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.

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 color by proportions of some categorical variable – one that is either already present or one that we create. Let us try to separate

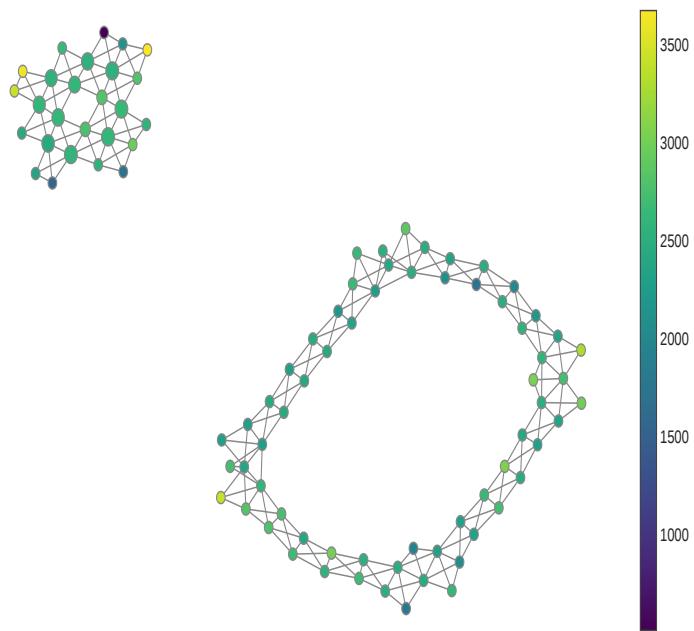
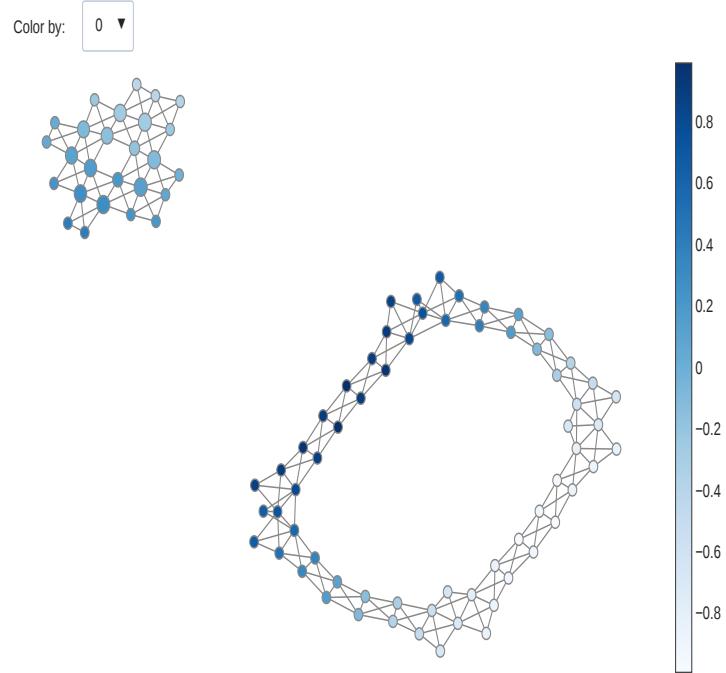


Figure 4.9: First result of applying Mapper.

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

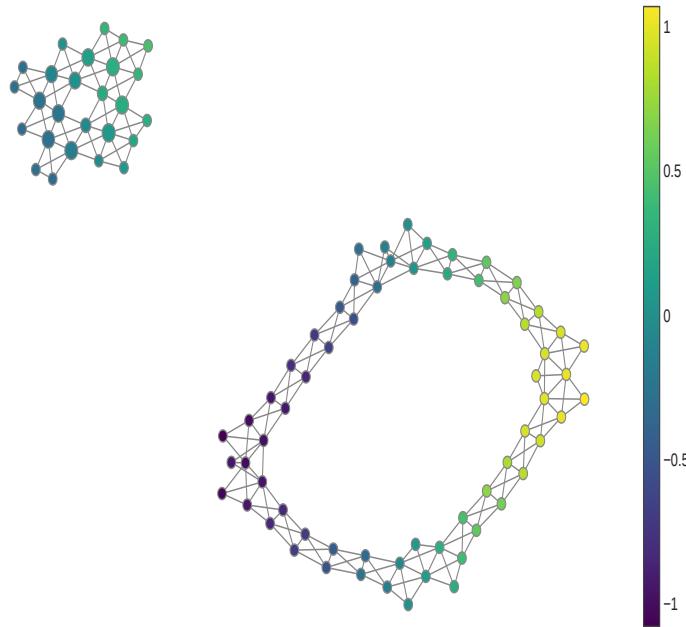


Figure 4.11: Nodes colored by the average first PCA component.

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 – $\{0, 1\}$.

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.

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

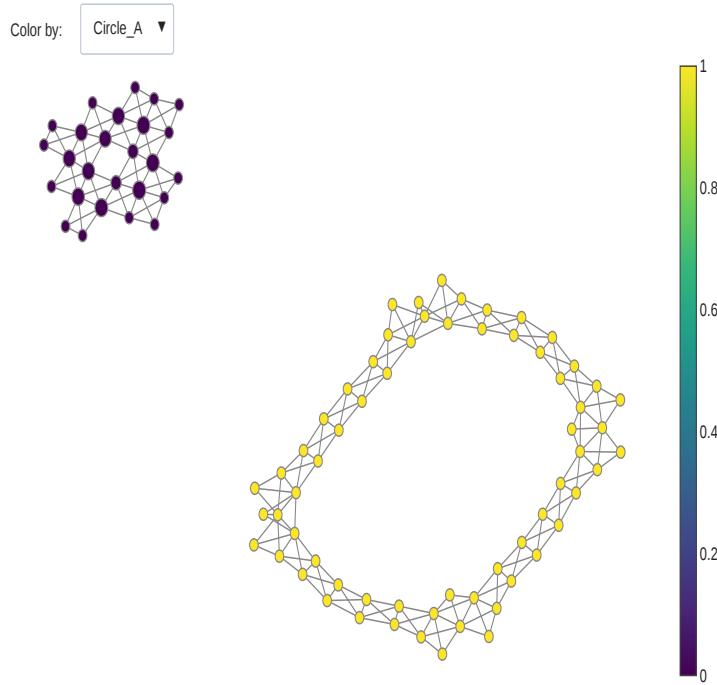


Figure 4.12: Nodes colored by categorical variables.

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](#).

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

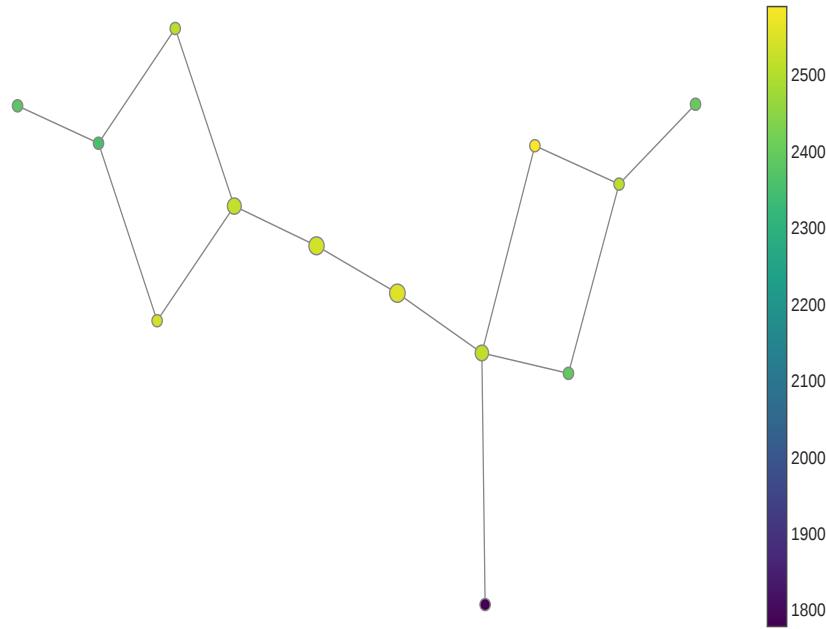


Figure 4.13: The effect of a different filtration function; in this case, the sum of the individual coordinates.

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.

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 separately from the rest. This way, we can plot the dark purple cluster, suggesting the lowest maximal volatility of closing prices in 4.16.

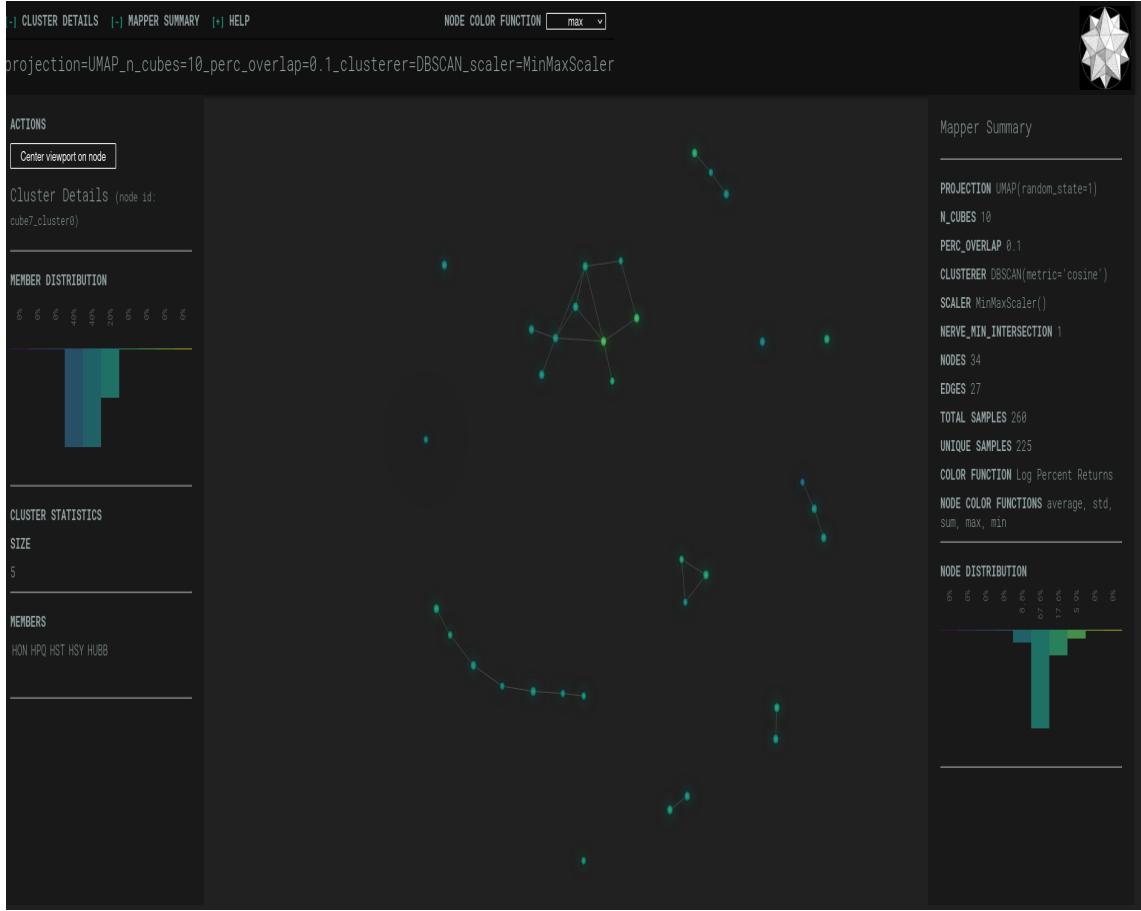


Figure 4.14: View of the Mapper clustering summary with *scikit-tda*.

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

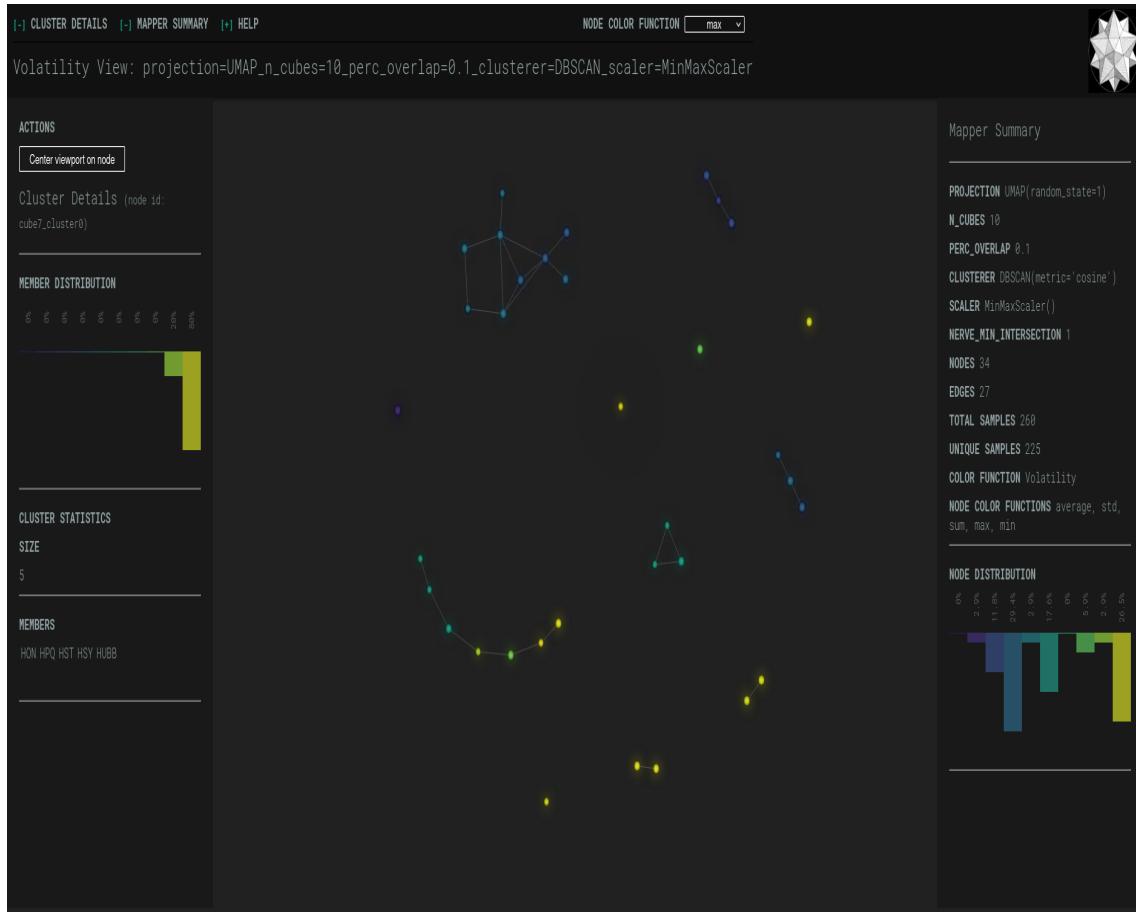


Figure 4.15: Mapper graph of the SP500 data with the color function being the volatility of the stock.

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 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.

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.

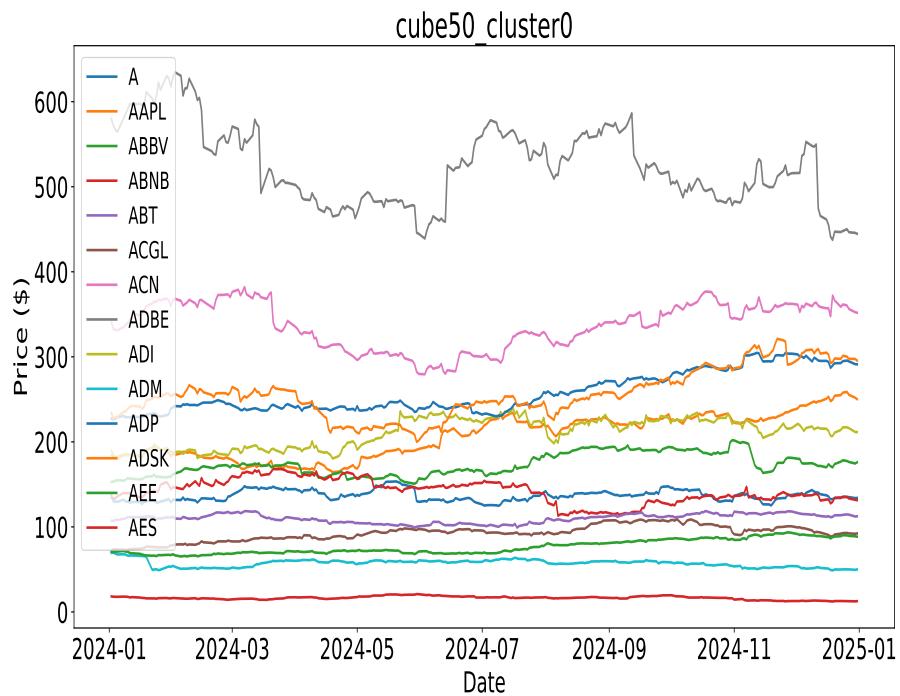


Figure 4.16: Cluster of tickers with the lowest maximal volatility.

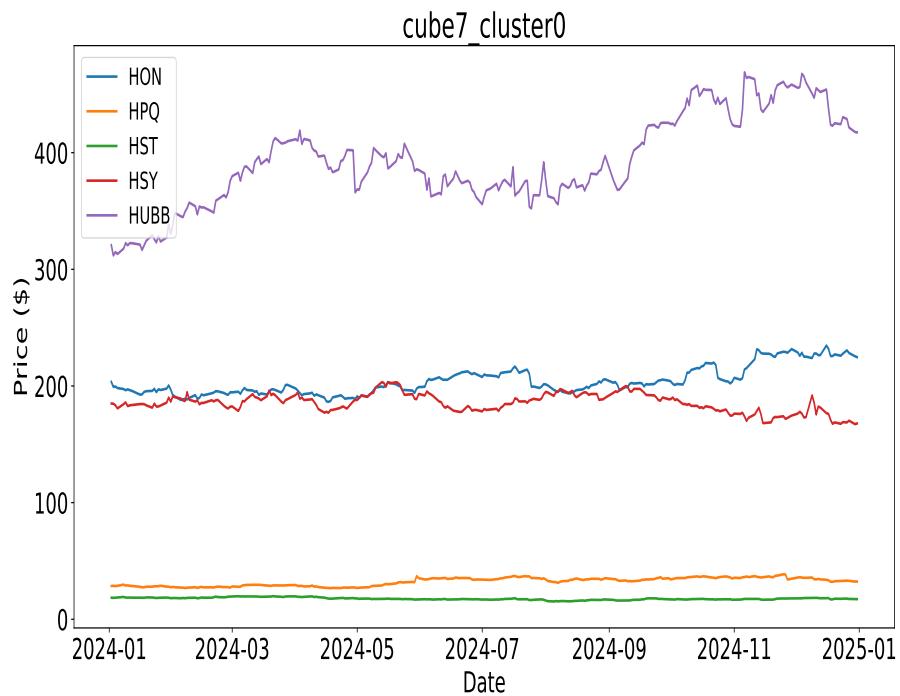


Figure 4.17: Cluster of tickers with the highest maximal volatility.

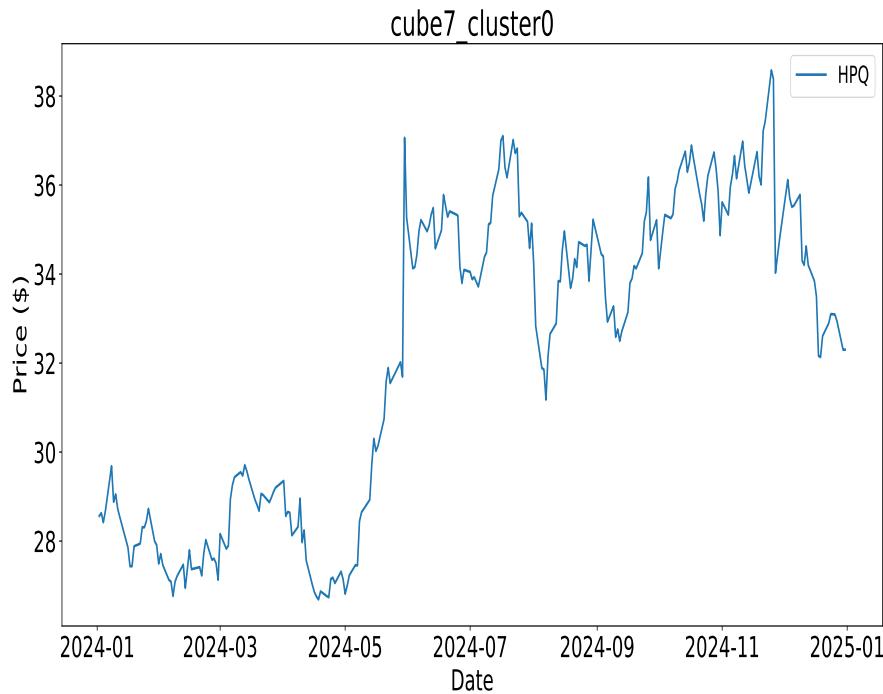


Figure 4.18: Stock prices of the company HP during 2024-2025.

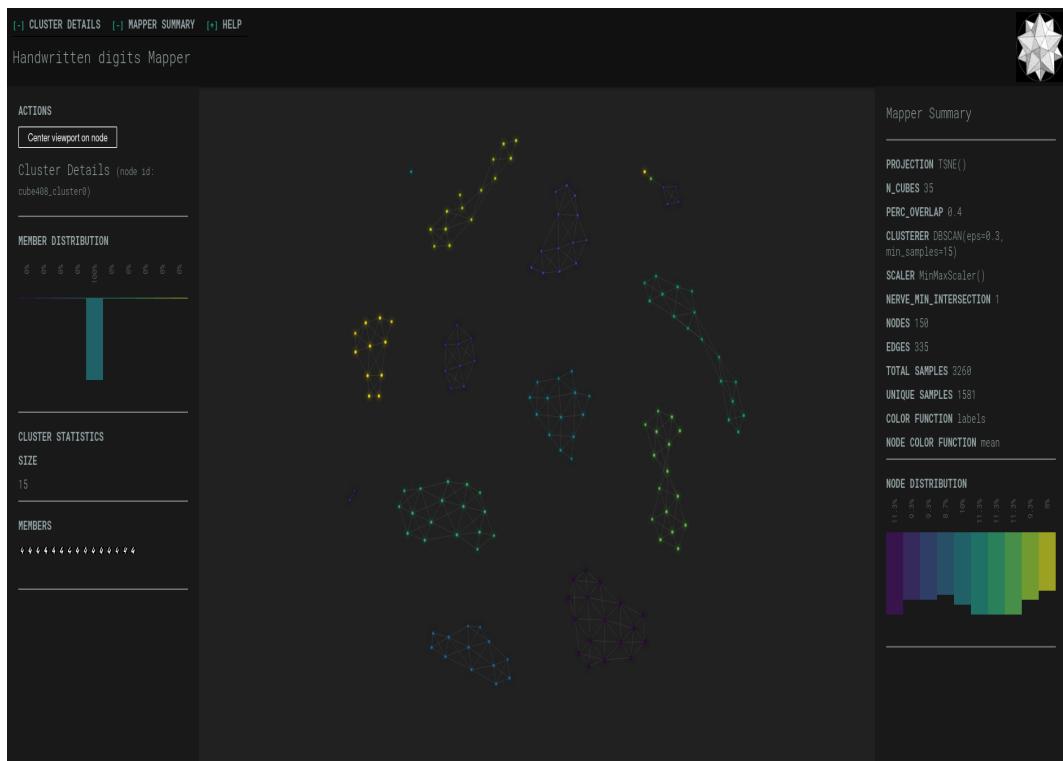


Figure 4.19: Mapper graph of the digits dataset with a custom tooltip function.

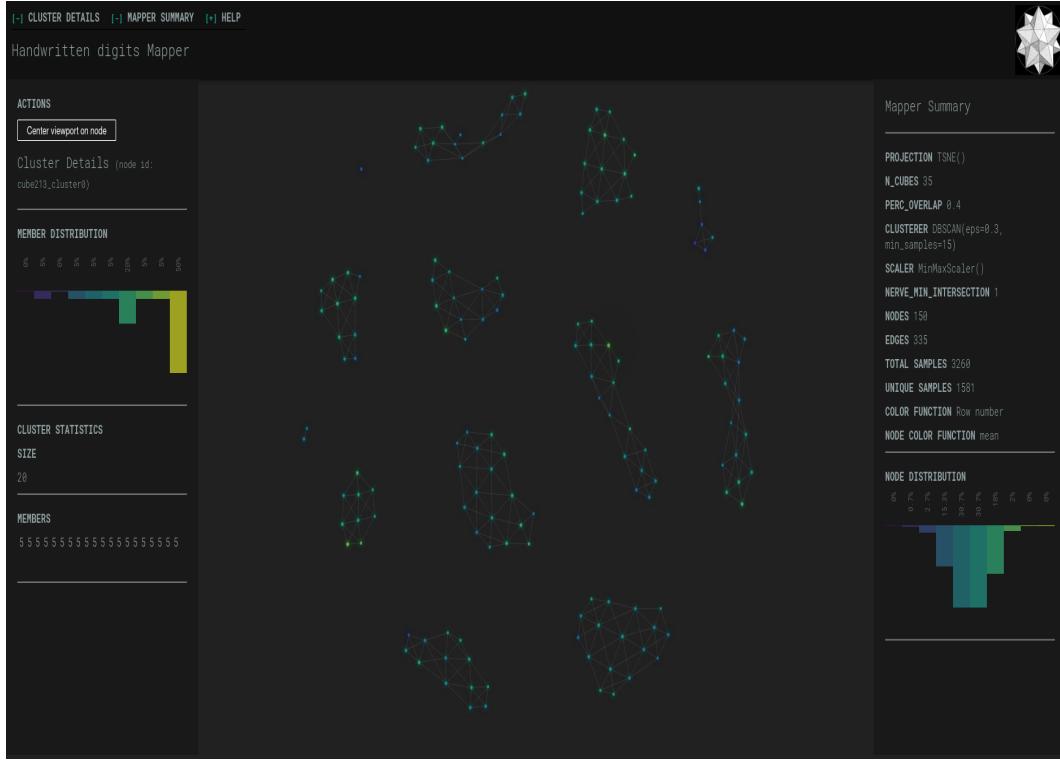


Figure 4.20: Mapper graph of the digits dataset with labels.

4.3 Persistent homology in practice

At last, we can start with an 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 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 and 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 will be the same, as per the axiomatic nature of homology theories. 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 grayscale image to binary by applying a threshold to the pixel values. In this example here, we used a threshold 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}_∞ 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 grayscale 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 values less than the i -th smallest pixel value in the grayscale 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.

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 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

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

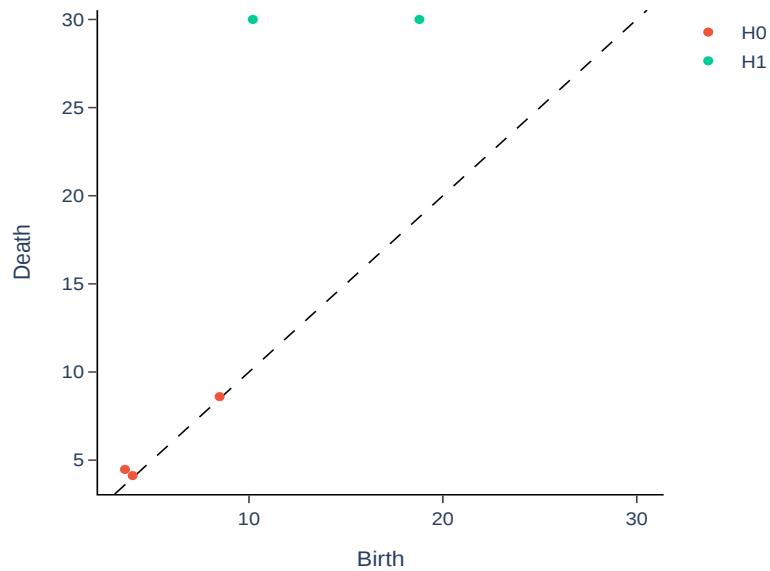


Figure 4.21: Cubical persistence homology of the digit 8.

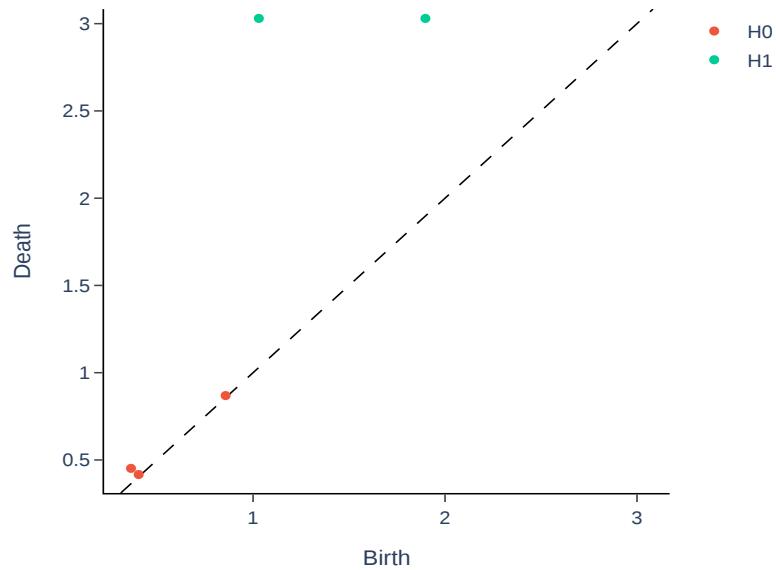


Figure 4.22: Scaled cubical persistence homology of the digit 8.

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 the direction, homology dimension, and number of trees give us the best accuracy. Running the search, we found that the optimal 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

In this section, we will construct our filtrations ourselves, specifically the sublevelset filtration (sometimes also called the lower star filtration) of some topological space. It is a stable way of describing and looking for critical points – local minima and maxima. This will be immediately clear when we start working with time series or signals but the approach will be demonstrated on data like images as well.

Intuitively, we compute the null homological group H_0 of the connected components that are constructed when a “rising pools of water” slowly fill out the space. When a new pool is formed at a local minimum, a class is born; when two pools merge together at a local maximum, the class dies.

Time Series

We will start with a simple example of a periodic time series where the local minima and maxima are visible. The function we will work with can be seen on [4.23](#).

The next step is to construct a sparse distance matrix to perform the sublevelset filtration. This has the added benefit of decreased computational cost due to the sparsity and a more fine-grained control of how we build the complex. For each point in the time series, there is a vertex, which is born at the height of the point in the time series. This represents the initial state of the filtration. Since we are interested in the sublevelset filtration, we will add edges at a height equal to the larger of the two heights of the vertices they connect. The edges naturally connect

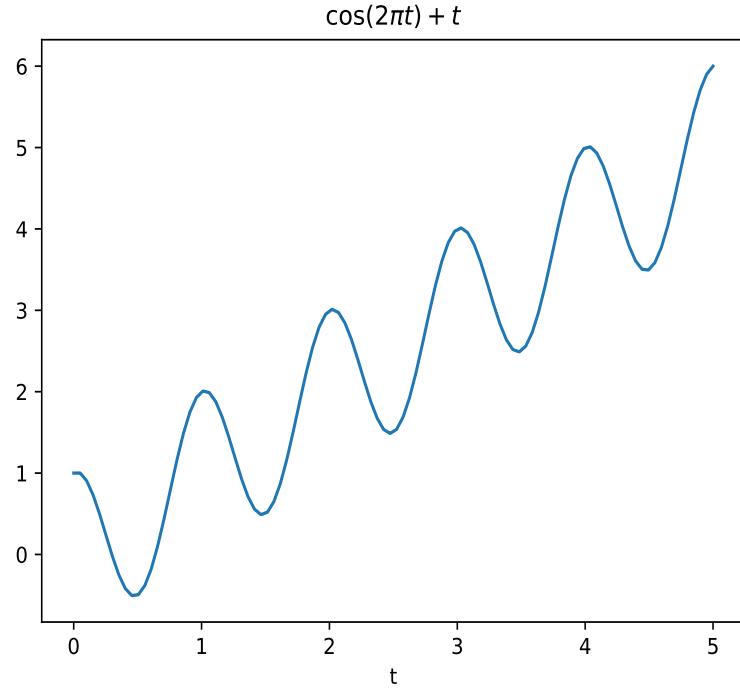


Figure 4.23: The periodic signal $\cos(2\pi t) + t$.

adjacent points in the time series since, usually, we measure the signal in time and time only goes from left to right.

In more practical terms, we add the vertex birth times along the diagonal of the distance matrix. Once we have our distance matrix, we can directly pass it as input for the computation of the persistence diagram. With some minor graphical touches, we can plot the results next to one another in 4.24.

We see that the birth times of the individual homology classes here nicely map to the local minima of the signal, while the death times map to the local maxima in a 1-to-1 correspondence. We could do the opposite, filling the “pools of water” from the top instead, and this would give us the superlevelset filtration of the time series. Of course, this amounts to calculating the sublevelset filtration of the same function but with a flipped sign in front.

Image Filtrations

Sublevelset filtration of images works in the exact same manner, allowing us to explore local minima as birth times and saddle points as death times while looking at the zero-dimensional homology groups. The definition of the function that provides the filtration is in the same spirit as with the time series – we construct a sparse distance matrix in which every pixel in the image is a vertex. Every vertex is connected to its 8 neighbors (or fewer, if it is a boundary pixel). The function value on the edge is taken to be the maximum of the two pixel values it connects. We can start with a simple image containing some Gaussian blobs, see 4.25. There

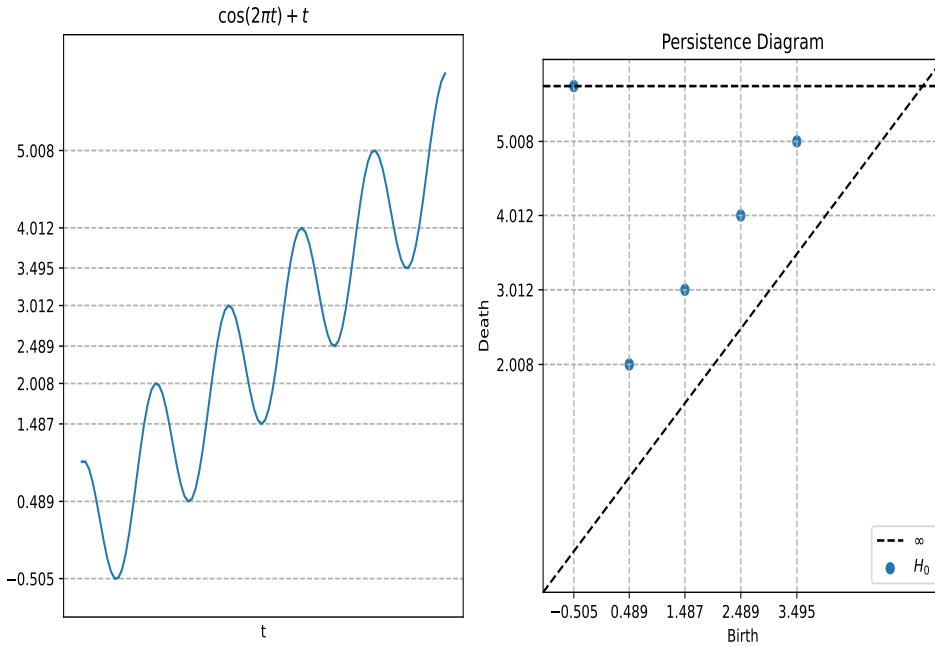


Figure 4.24: The sublevelset filtration of the signal.

are three negative Gaussians present, one reaching its minimum at -3, the other at -2, and the last at -1.

For images, there already exists a pre-built function that we can use instead of writing it ourselves. Computing the persistence diagram of this image gives us 4.26. Just like with the time series example, the three dots correspond to each of the 3 Gaussians, each of which is born at the respective local minimum. Two of them die around 0, and the third is born at -3, absorbing all classes into itself and never dying.

Let us look at a slightly more exciting example in 4.27. Here we have a zoomed-up image of some wood cells. The first step is to convert this image to a grayscale one. Doing so, we see that the interiors of the cells have high values. They also meet a saddle point somewhere on their boundary with a value close to 0. We want to ideally identify the interiors of the cells in an automated fashion. Since this is our goal, we are interested in a local maximum of large persistence within each cell; that is, we want to perform superlevelset filtration of the image.

But as mentioned above, we don't need to hack a suboptimal solution; we can simply pass the negative of the image and compute its sublevelset filtration instead. The result of this we can see in 4.28. While the diagram is a useful first step, we need to extract the cell interior in some way out of this. We can do this by selecting a persistence threshold, above which we will consider a dot to be associated with a cell.

We can add some white noise to the pixels and apply a uniform filter to add a small amount of blur to the image. This would ideally increase the chance of

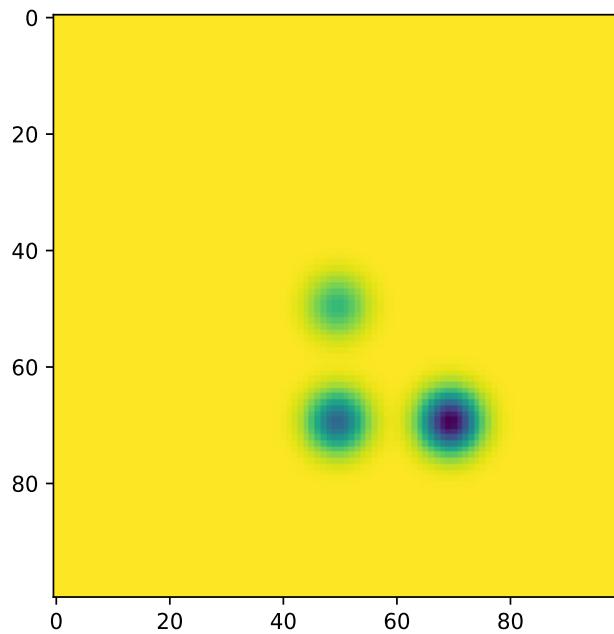


Figure 4.25: Image containing some Gaussian blobs.

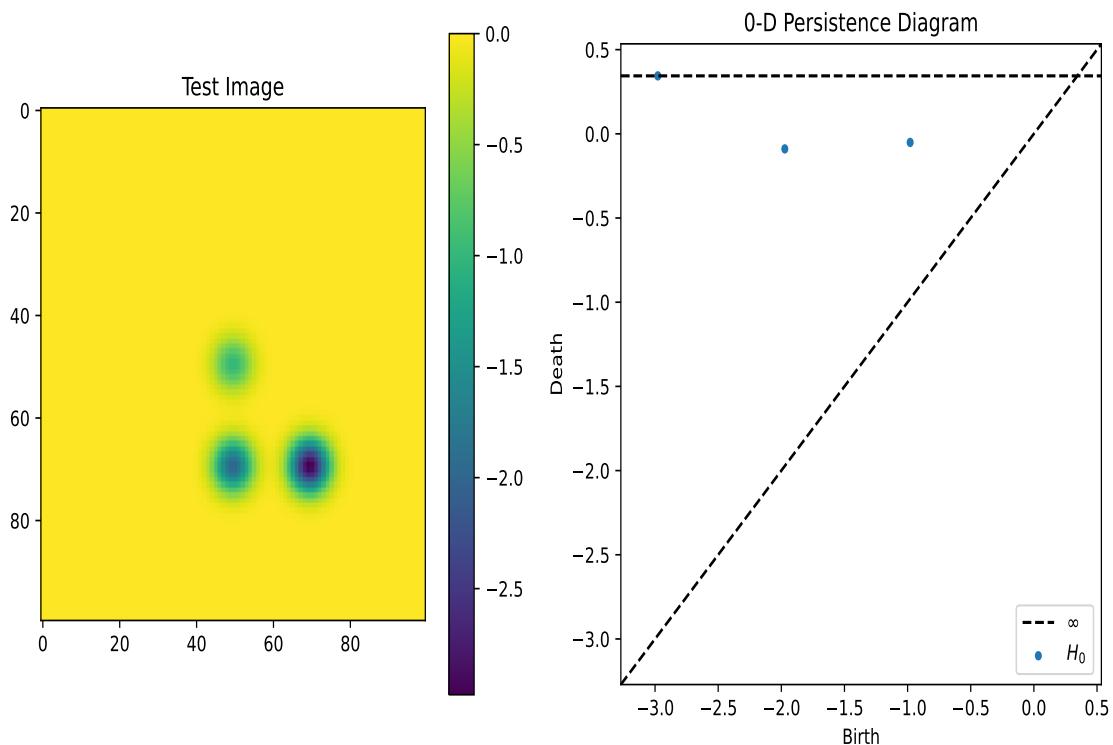


Figure 4.26: The result of taking the sublevelset filtration of the Gaussian blobs.

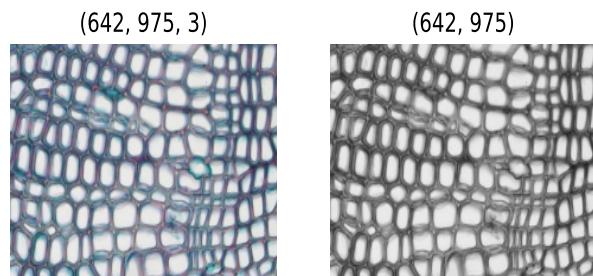


Figure 4.27: Example image of wood cells.

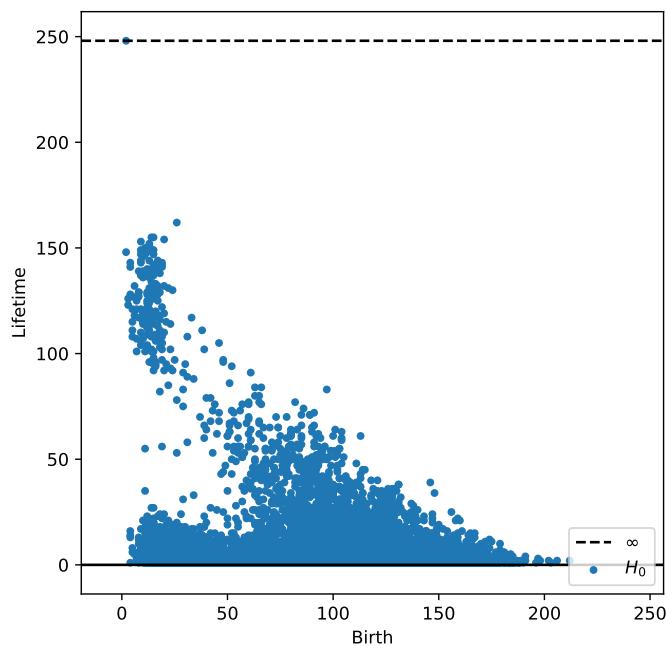


Figure 4.28: Persistence diagram of the grayscale image of the cells.

the maximum to being closer to the center of the cell, the result of which is seen in 4.29. If we compute the persistence diagram of the smoothed image, the difference is visible in 4.30.

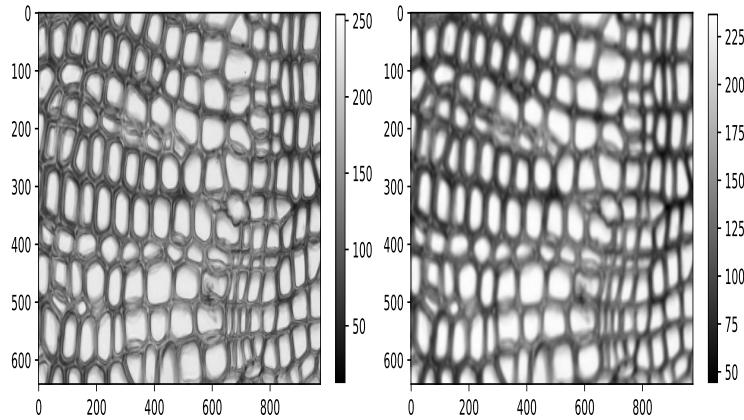


Figure 4.29: Smoothed out version of the grayscale image of the cells.

We choose a cutoff of all points with a lifetime greater than 70 in this example. Extracting the pixel representatives satisfying this condition and plotting them in the original image results in 4.31. Using a naive approach to the problem, we obtained reasonable results. There are cells with duplicates, and some have no points at all, but most have only one. Tuning the threshold and the amount of noise added could further improve this procedure without relying on advanced algorithms from image processing.

On an ending note, we can give a formal proof of the fact that a black hole really has a hole inside, using the famous image and computing its sublevelset filtration as we did above.

4.3.3 Sliding Windows

The second approach when working with time series or signals is to make use of Taken's embedding theorem and sliding windows. [Tak06]. A delay embedding theorem gives us conditions under which we can reconstruct a chaotic dynamical system from a sequence of observations. Taken's theorem specifically gives us conditions under which we can reconstruct the attractors of the dynamical system.

Using Taken's theorem, we can transform the signal into a point cloud, allowing us to directly compute its persistent homology groups without further

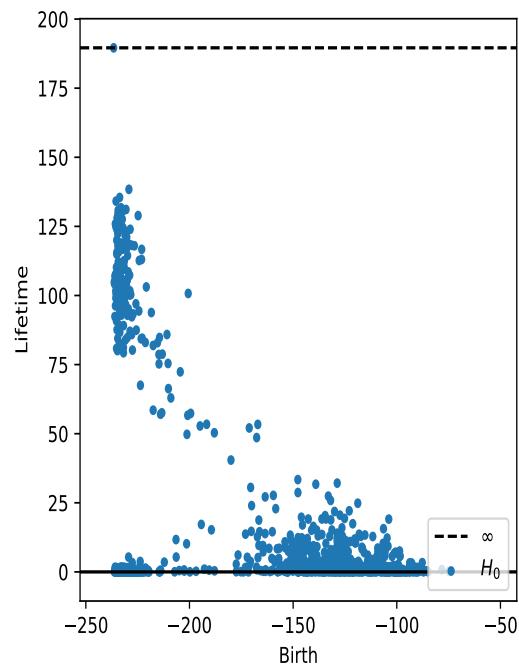


Figure 4.30: Persistence diagram of the smoothed out image.

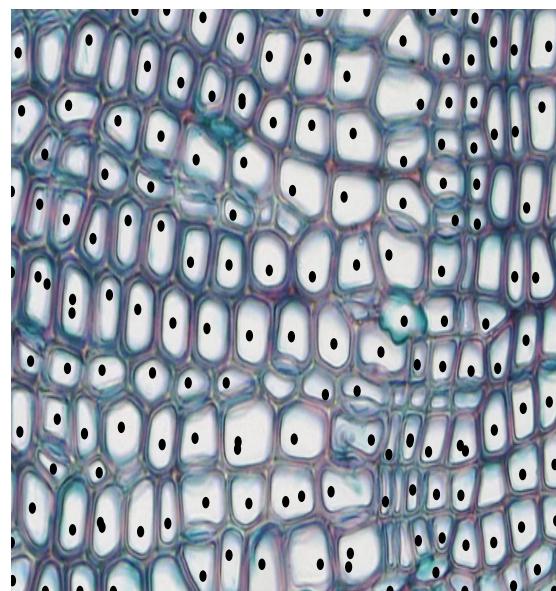


Figure 4.31: Filled out image of cells.

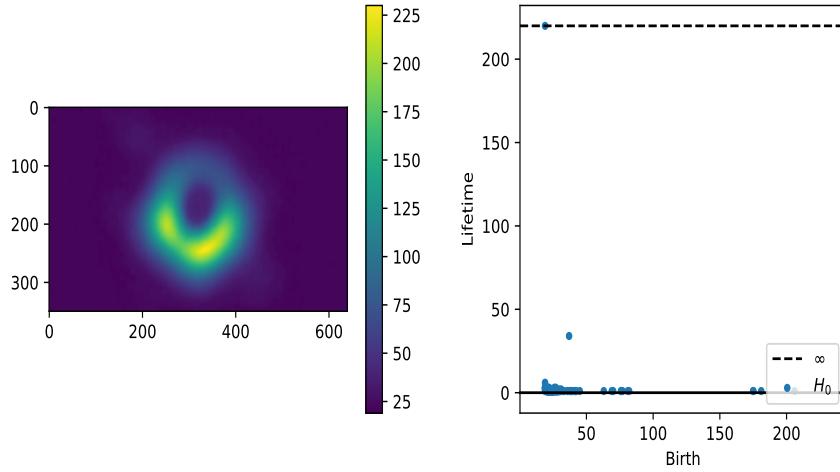


Figure 4.32: Persistence diagram of a black hole.

transformations. Formally, given a time series $f(t)$, we extract vectors of the form

$$f_i = [f(t_i), f(t_i + \tau), f(t_i + 2\tau), \dots, f(t_i + (d-1)\tau)] \in \mathbb{R}^d,$$

where d is the dimension of the embedding and τ is the time delay parameter. Typically, we call the quantity $(d-1)\tau$ the *window size* and the difference $t_{i+1} - t_i$ the *stride*. The underlying idea here is, as long as f has a non-trivial recurrent structure, to use Taken's theorem to construct an embedding of the signal that preserves the topological qualities and then compute its persistent homology.

We may start with a simple cosine wave. The choice of stride and window size is the hardest problem when using a sliding window technique, as it is not exactly obvious which values will give us an “optimal” embedding. Later we’ll see that we can search for them automatically, but for now, let us choose $d = 3$, $\tau = 8$ and a stride of 10. With this choice of parameters, the resulting embedding in \mathbb{R}^3 is shown in 4.33.

We can compare this with a quasi-periodic signal of the form

$$f(t) = \cos(t) + \cos(\pi t),$$

as seen in 4.34, where we used $d = 3$, $\tau = 16$ and a stride of 3. The difference in the geometry of the embedding should be immediately clear, as in the case of the cosine wave, we have recovered a perfect ellipse capturing the clean, periodic behaviour of the signal.

We can confirm this by looking at the persistence diagrams that we compute from the embeddings in 4.35. In the first diagram, we see a single point associated with the 1-dimensional persistent homology. This represents the single loop found in the embedding and is a direct consequence of the periodic nature of the signal.

Instead of manually picking the values for the embedding, we can use the following two techniques to search for them automatically:

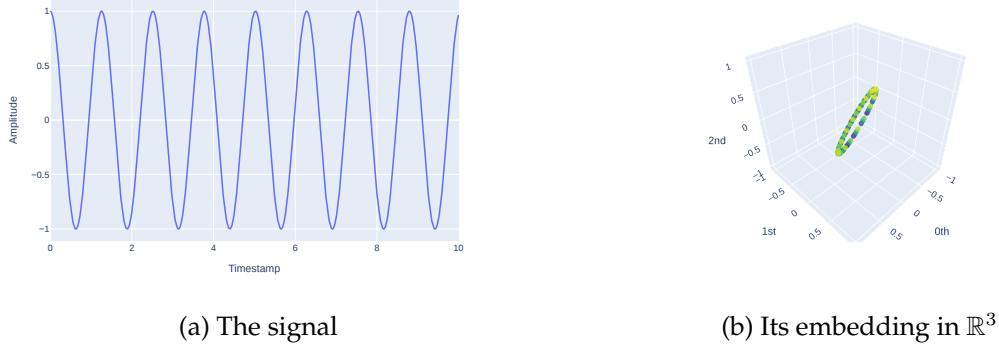


Figure 4.33: Ordinary cosine wave

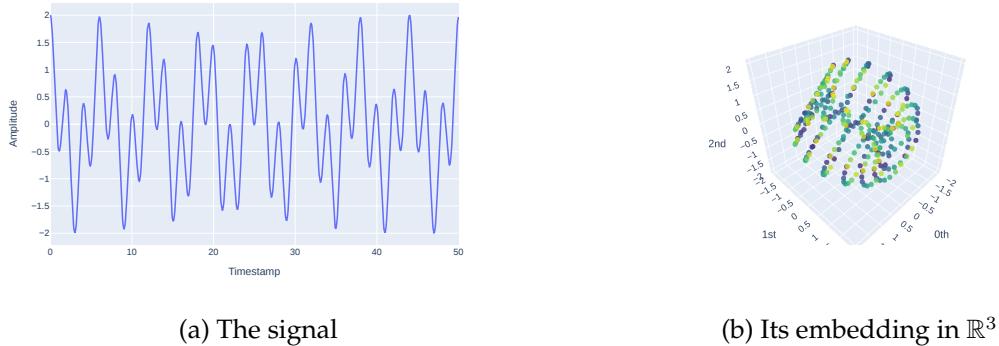


Figure 4.34: Quasi-periodic wave

- Mutual information for τ
- False nearest neighbours for d

To determine an optimal τ , we first calculate the maximum x_{\max} and minimum x_{\min} values of the time series, dividing the interval between them into a large number of bins. Let p_k be the probability that an element of the time series is in the k th bin and let $p_{j,k}$ be the probability that x_i is in the j th bin while $x_{i+\tau}$ is in the k th bin. Then we define the mutual information as:

$$I(\tau) = - \sum_{j=1}^n \sum_{k=1}^n p_{j,k}(\tau) \log \left(\frac{p_{j,k}(\tau)}{p_j p_k} \right),$$

where n is the number of bins on the interval $[x_{\min}, x_{\max}]$. The first minimum of this expression gives us the optimal time delay.

The false nearest neighbours algorithm, first published in [KBA92] with an overview in [RM97], serves to identify the optimal embedding dimension of a dynamical system in order to unfold the attractor of the underlying system. An

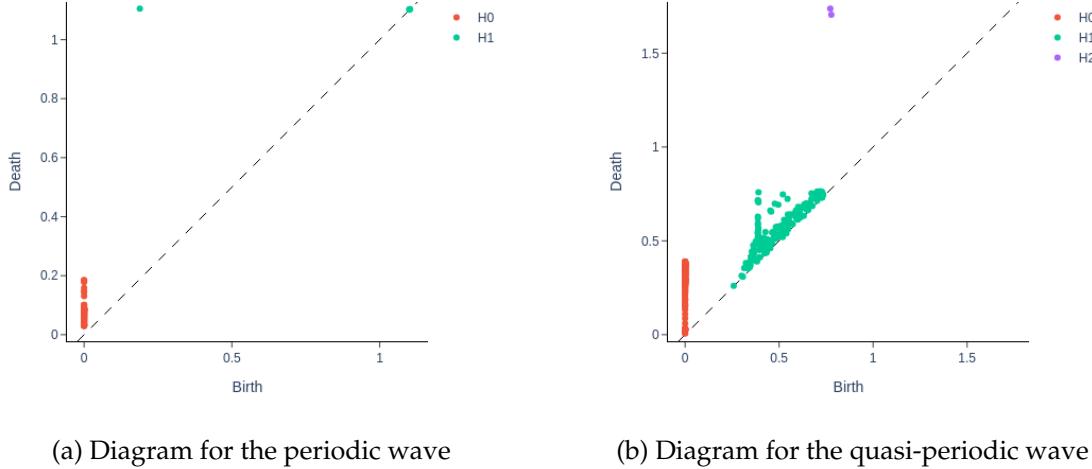


Figure 4.35: The corresponding persistence diagrams

attractor is often a compact object in the phase space, thus acquiring neighbours. If our embedding dimension is too small to unfold the attractor, not all the points that lie close to each other will be neighbours, simply appearing as such due to a projection to a lower-dimensional space. We are thus interested in the change of distance between the embedded vector and its nearest neighbour when we go from a dimension d to $d + 1$. More precisely, if we denote $y(n) = (x(n), x(n + \tau), \dots, x(n + (d - 1)\tau))$, its r th nearest neighbour by $y(n)^{(r)}$, then the square of Euclidean distance between the two is

$$R_d^2(n, r) = \sum_{k=0}^{d-1} [x(n + k\tau) - x^{(r)}(n + k\tau)]^2.$$

The natural criterion for catching the correct dimension is that the increase of distance between $y(n)$ and $y^{(r)}(n)$ will be large when going from d to $d + 1$, that is

$$\frac{|x(n + \tau d) - x^{(r)}(n + \tau d)|}{R_d(n, r)} > R_{\text{tol}},$$

where R_{tol} is some threshold. Any neighbour for which the inequality above is true is designated to be a false neighbour. It turns out that $R_{\text{tol}} \geq 10$ clearly identifies false neighbours and it is sufficient to consider only the nearest neighbours, $r = 1$, hence the name.

We have to choose an upper bound for both d and τ for the search to have a limit. We have chosen $d = 30$ and a $\tau = 30$ with a stride of 5. Once the search was finished, we have obtained optimal parameters $(d, \tau) = (6, 29)$. Due to the higher dimension, we used PCA to visualize the results in \mathbb{R}^3 , as seen in 4.36.

Just as before, we can then compute their persistence diagrams, seen in 4.37. For the periodic signal, it is essentially the same. In the case of the quasi-periodic

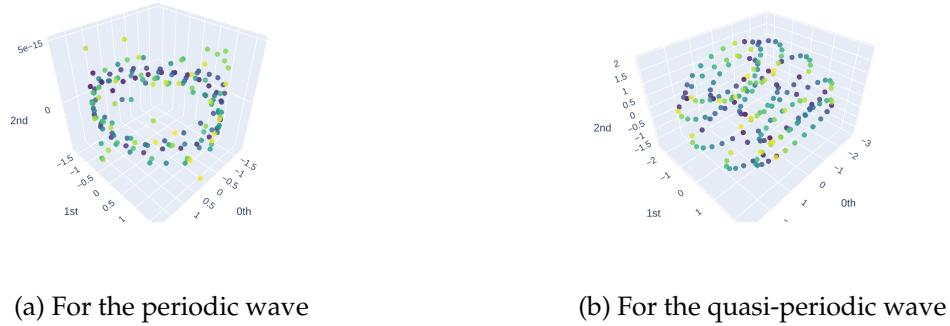


Figure 4.36: Optimal embeddings, after a reduction with PCA

wave, there is less noise on the diagonal. Most importantly, we see two prominent H_1 points and one H_2 point, which corresponds to the signature of a hypertorus.

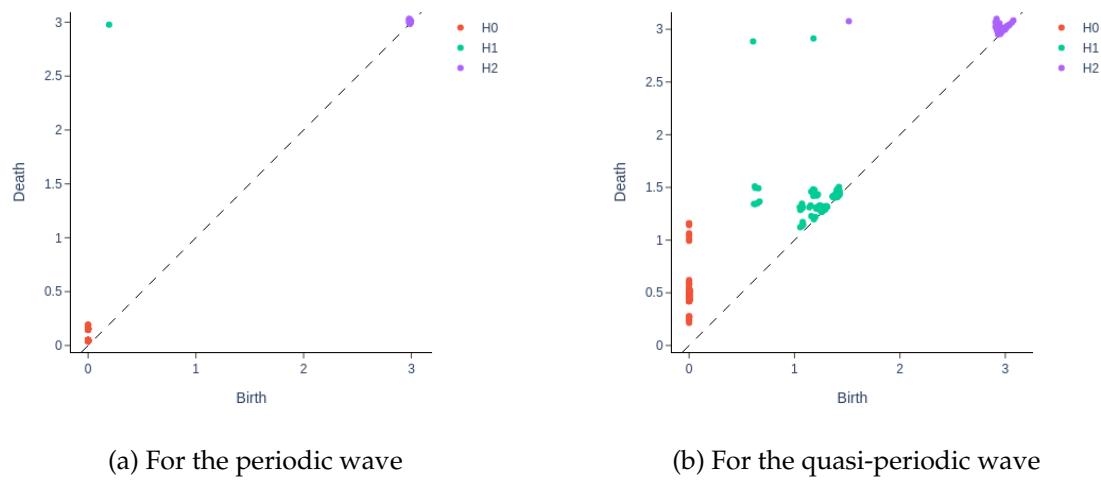


Figure 4.37: Persistence diagrams of the optimal embeddings

Using sliding windows and Taken's embedding gives us tools to help us classify or cluster time series based on the topology of their embeddings. The notebook for the examples above can be found in *src/Sliding Windows 1.ipynb*. In the notebook *src/Sliding Windows 2.ipynb*, we provide further examples and simulations of sliding window embeddings, allowing the reader to play around with the parameters and see how the change in parameters affects the change in geometry and in the resulting persistence diagrams in a more interactive fashion. For example, we can see how the embedding is progressively more chaotic the more stochastic noise we add into the original signal and how noisy the persistence diagram becomes in 4.38.

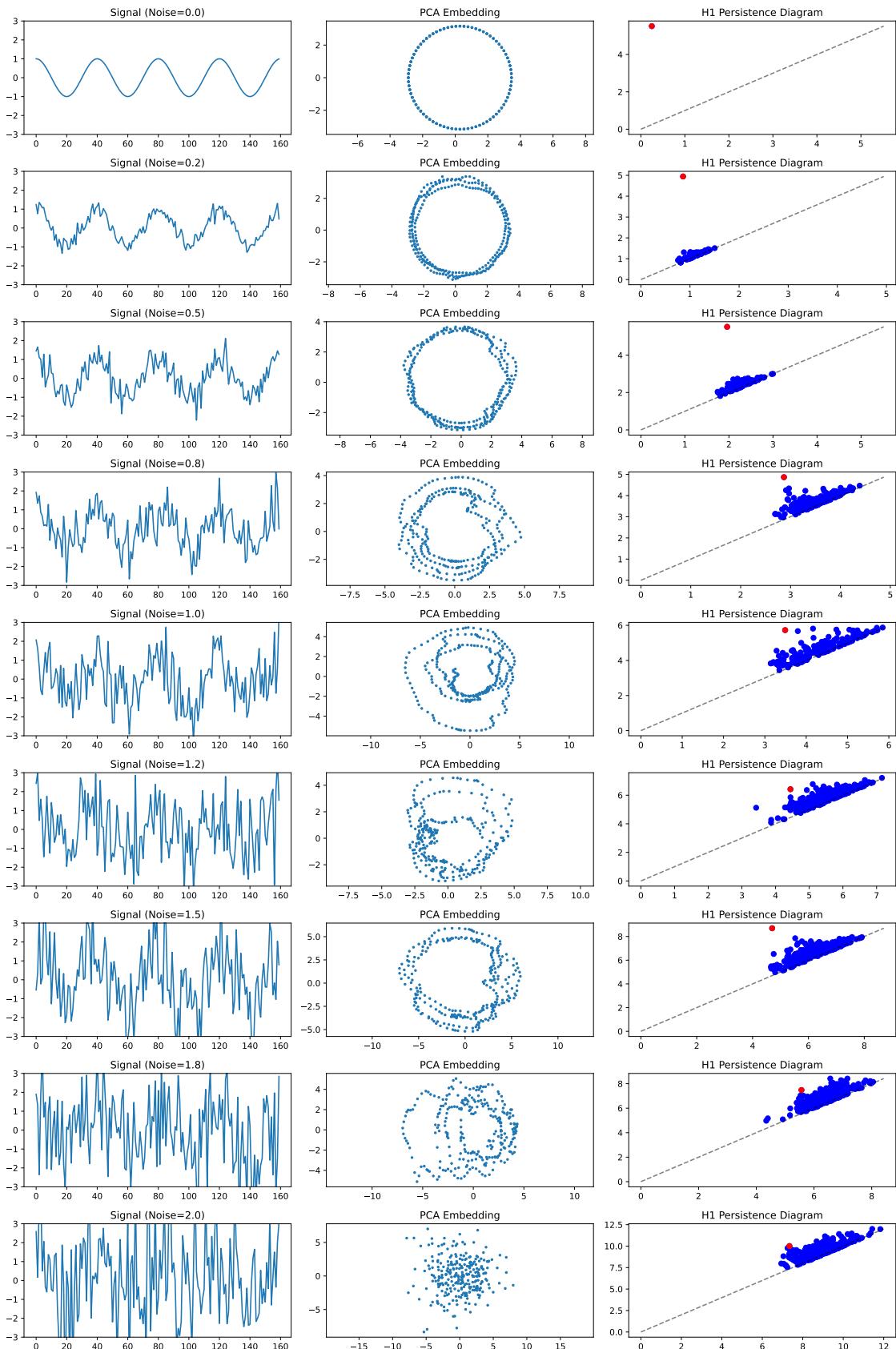


Figure 4.38: The noisier our signal is, the less informative the geometry of its embedding and the persistence diagram becomes.

Summary

We spent the first two chapters building the theory behind the most difficult concept in Topological Data Analysis – simplicial complexes and persistent homology. Once we established the concepts that make the theory usable in practice, that is, stability and robustness against noise, we began investigating its properties. In the practical part of the work, we showed examples of the three methods that are covered under the term Topological Data Analysis – UMAP, Mapper, Persistent Homology and its equivalent representations. If the reader managed to get this far and understand the essence of it, they will be more than capable of picking up any article where those same methods are used.

To direct the reader in further and recent publications about TDA, we provide a short and incomplete list of articles that came out within the past two years that might be of interest: [WMFC25], [APM⁺25], [JDY⁺25], [JDL25], [Wis25], [ACS⁺25], [MIMI], [PMD⁺24], [LI24], [HLMMG24], [CGVC25], [Arf24], [dSRB25], [SPEI25], [BLM24].

Additionally, a comprehensive review of some applications of the Mapper algorithm is in the article [MUZ25]. If the reader wanted to join a community of researchers and scientists using methods from TDA and applied algebraic topology in general, they might want to visit [AAT].

The entire repository of the thesis, along with all the figures, source code, and data sources, can be found on the following link: <https://github.com/xpetitt123/Master-Thesis>.

Bibliography and sources

- [AAT] Aatrn - applied algebraic topology research network. <https://www.aatrn.net/>. [Online; accessed 21-April-2025].
- [ACS⁺25] Arvindh Arun, Karuna K Chandra, Akshit Sinha, Balakumar Velayutham, Jashn Arora, Manish Jain, and Ponnurangam Kumaraguru. Topo goes political: Tda-based controversy detection in imbalanced reddit political data. *arXiv preprint arXiv:2503.03500*, 2025.
- [AK98] E. Alpaydin and C. Kaynak. Optical Recognition of Handwritten Digits. UCI Machine Learning Repository, 1998. DOI: <https://doi.org/10.24432/C50P49>.
- [APM⁺25] Dashti A Ali, Jacob J Peoples, Ramtin Mojtabaei, Kaitlyn S Kobayashi, William R Jarnagin, Richard KG Do, and Amber L Simpson. Leveraging persistent homology for liver tumour classification. In *Medical Imaging 2025: Computer-Aided Diagnosis*, volume 13407, pages 407–417. SPIE, 2025.
- [Arf24] Badredine Arfi. The promises of persistent homology, machine learning, and deep neural networks in topological data analysis of democracy survival. *Quality & Quantity*, 58(2):1685–1727, 2024.
- [BLM24] Evelyn A Boyd, Kelly Best Lazar, and Stephen Moysey. Big data to support geoscience recruitment: Novel adoption of topological data analysis in geoscience education. *Bulletin*, 136(3-4):1458–1468, 2024.
- [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.
- [CGVC25] Manuel Cuerno, Luis Guijarro, Rosa María Arnaldo Valdés, and Fernando Gómez Comendador. Topological data analysis in air

- traffic management: The shape of big flight data sets. *Plos one*, 20(2):e0318108, 2025.
- [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.
- [dSRB25] Ketson RM dos Santos, Amir Rezaie, and Katrin Beyer. Topological data analysis-based damage indices for plastered stone masonry walls under cyclic loading. *Engineering Structures*, 322:119088, 2025.
- [DW22] Tamal Krishna Dey and Yusu Wang. *Computational topology for data analysis*. Cambridge University Press, 2022.
- [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.
- [HLMMG24] Enrique Hernández-Lemus, Pedro Miramontes, and Mireya Martínez-García. Topological data analysis in cardiovascular signals: An overview. *Entropy*, 26(1):67, 2024.
- [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].
- [JDL25] Cristian B Jetomo and Mark Lester D De Lara. Filipino sign language alphabet recognition using persistent homology classification algorithm. *PeerJ Computer Science*, 11:e2720, 2025.
- [JDY⁺25] Jiayv Jing, Ling Ding, Xu Yang, Xu Feng, Jinchao Guan, Hong Han, and Hainian Wang. Topology-informed deep learning for pavement crack detection: Preserving consistent crack structure and connectivity. *Automation in Construction*, 174:106120, 2025.
- [KBA92] Matthew B Kennel, Reggie Brown, and Henry DI Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Physical review A*, 45(6):3403, 1992.
- [LI24] Aleksei Luchinsky and Umar Islambekov. Tdavec: Computing vector summaries of persistence diagrams for topological data analysis in r and python. *arXiv preprint arXiv:2411.17340*, 2024.
- [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.
- [MIMI] Yoh-ichi Mototake, Shinichi Ishida, Norihiro Maruyama, and Takashi Ikegami. Topological data analysis of large swarming dynamics.
- [MUZ25] Vine Nwabuisi Madukpe, Bright Chukwuma Ugoala, and Nur Fariha Syaqina Zulkepli. A comprehensive review of the mapper algorithm, a topological data analysis technique, and its applications across various fields (2007-2025). *arXiv preprint arXiv:2504.09042*, 2025.
- [Nat24] Nathaniel Saul. Čech complex playground — sauln.github.io. <https://sauln.github.io/blog/nerve-playground/>, 2024. [Online; accessed 16-August-2024].
- [PMD⁺24] Martin Cramer Pedersen, Sourav Mukherjee, Amin Doostmohammadi, Chandana Mondal, and Kristian Thijssen. Active particles knead three-dimensional gels into porous structures. *Physical Review Letters*, 133(22):228301, 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.
- [RB19] Raúl Rabadán and Andrew J Blumberg. *Topological data analysis for genomics and evolution: topology in biology*. Cambridge University Press, 2019.
- [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.
- [RM97] Carl Rhodes and Manfred Morari. The false nearest neighbors algorithm: An overview. *Computers & Chemical Engineering*, 21:S1149–S1154, 1997.
- [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.
- [SPEI25] Laura Selicato, Alessandro Pagano, Flavia Esposito, and Matteo Icardi. Topological data analysis for resilience assessment of water distribution networks. *Mathematics and Computers in Simulation*, 231:62–70, 2025.
- [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].
- [Tak06] Floris Takens. Detecting strange attractors in turbulence. In *Dynamical Systems and Turbulence, Warwick 1980: proceedings of a symposium held at the University of Warwick 1979/80*, pages 366–381. Springer, 2006.
- [TLT⁺20] Guillaume Tauzin, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Aníbal 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] Vudit Nanda. Perseus, the persistent homology software. <http://www.sas.upenn.edu/~vnanda/perseus>. [Online; accessed 25-January-2025].
- [Wis25] Jim Wiseman. Persistent homology of nba lineups usage. *Journal of Sports Analytics*, 11:22150218251324877, 2025.
- [WMFC25] Dongjun Wei, Minjia Mao, Xiao Fang, and Michael Chau. Short-phd: Detecting short llm-generated text with topological data analysis after off-topic content insertion. *arXiv preprint arXiv:2504.02873*, 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.

