Topological Mode Analysis Tool (ToMATo): A Clustering Method Based on the Topological Data Analysis (TDA)

Submitted by

Ananya Sharma (2023MAS7117)
Harshit Joshi (2023MAS7141)

Submitted to

Prof. Niladri Chatterjee

Department of Mathematics

INDIAN INSTITUTE OF TECHNOLOGY DELHI
HAUZ KHAS, NEW DELHI-110016 INDIA

April 20, 2025

Abstract

This report presents an in-depth study of **Topological Mode Analysis (ToMATo)**, a clustering technique leveraging **topological persistence** to identify meaningful structures in data. The study begins with foundational concepts in **topology and homology**, detailing their computation and relevance in data analysis. A significant focus is placed on **persistent homology and persistence diagrams**, which are used to extract topological features from point clouds, images, and graphs. The methodology involves constructing filtrations to analyze data shape and structure, aiding in robust clustering. The proposed approach is evaluated in various scenarios, demonstrating its effectiveness in uncovering complex patterns. The results indicate that **ToMATo provides an efficient, noise-resistant clustering framework**.

Contents

Chapter 1

Basics of Topology

1.1 Introduction

Clustering, is a crucial tool for analyzing and interpreting data across various fields. Some commonly used clustering techniques are:

- **K-means:** Among clustering techniques, the K-means algorithm is widely used due to its simplicity and efficiency. It aims to partition data into k clusters by positioning cluster centers and defining cluster boundaries in a way that minimizes the sum of squared distances between data points and their respective cluster centers. However, K-means and its variants struggle with non-convex clusters, leading to suboptimal results.

- **DBSCAN:** It is a density-based clustering method which operates under the assumption that data points originate from an unknown density function f. The clustering task then involves understanding the structure of f using the given samples. A common approach is to threshold the density at a fixed level α and consider the connected components of the superlevel set $F^\alpha = f^{-1}([\alpha, +\infty))$ as clusters while treating the remaining data points as noise. However, due to the fixed threshold α, these methods struggle with hierarchical datasets where multiscale clustering patterns exist.

- **Mean Shift Ckustering:** It is a mode-seeking clustering that identifies the local maxima (modes) of the density function f to serve as cluster centers and assigns data points based on their basins of attraction. It first estimates the density function using a multivariate kernel density estimate and then applies gradient ascent to locate the modes. The clusters are defined by these basins of attraction. However, density gradients and extrema are often unstable, making their approximation challenging. To address this, Mean Shift employs smoothing before the hill-climbing step, but this raises the issue of determining an appropriate smoothing level to balance noise removal and signal preservation.

In this report, we adopt a more reactive approach known as **Topological Mode Analysis Tool (ToMATo)** that using topological persistence can improve cluster stability by detecting and merging unstable clusters post-computation. While similar in nature to Mean Shift, this method leverages persistence to explicitly link input parameters with the resulting number of clusters. By estimating the persistence of density peaks, a hierarchical structure of clusters is formed. Notably, prominence—a measure of a peak's relative significance—is more stable than absolute height. For instance, a small bump in a high-density region may exhibit a large absolute height but low prominence, making persistence-based methods more reliable for clustering.

Before exploring the algorithm behind ToMATo, it is essential to first understand some fundamental concepts of topology.

- **Topological spaces:** Topology studies the properties of shapes and spaces that remain unchanged under continuous transformations. It defines the structure of a set by determining neighborhood relationships.

Example: Let $X = [0, 1]$. If we define a distance(metric) on X as $d_1(x, y) = 1$ for $x \neq y$ and $d_1(x, y) = 0$ for $x = y$ then shape of X would resemble an infinite number of points dispersed in a very high-dimensional space. While using $d(x, y) = |x - y|$ results in a stick of length 1.

In applied settings, datasets often have natural metrics: point clouds use spatial distances, graphs rely on node connectivity, and images consider pixel adjacency. However, some data types, like RNA sequencing, require explicitly defined metrics to establish meaningful relationships.

- **Simplicial Complexes:** It is a discrete mathematical structure used in topology to study shapes and spaces. It consists of vertices, edges, and higher-dimensional simplices like triangles (2-simplices) and tetrahedra (3-simplices), which are combined in a structured way.

- **k-dimensional manifold:** A **k-dimensional manifold** is a space where every small neighborhood resembles a ball in R^k, independent of the surrounding space.

  Example: a circle is 1-dimensional since its local structure is a line, while a sphere is 2-dimensional as its neighborhoods resemble flat surfaces.

- **Boundary:** The boundary of a k-dimensional manifold consists of points whose neighborhoods resemble a half-space in R^k. A sphere has no boundary, while a disk has a boundary (a circle), and a solid ball has a boundary (a sphere). The boundary of a k-dimensional manifold is usually a $(k-1)$-dimensional manifold without a boundary. Also, the boundary of a boundary is always empty.

- **Topological Equivalence:** Topological equivalence means two spaces are the same in topology if one can be continuously deformed into the other without tearing, gluing, or collapsing. This is mathematically captured by a **homeomorphism**, a continuous bijection with a continuous inverse.

  Example: a circle and an ellipse are topologically equivalent, but a circle and a figure-eight are not, as the latter has an extra hole.

  Homotopy is a more flexible deformation concept that allows collapsing but not tearing or gluing.

  Example: a disk is homotopic to a point since it can be shrunk continuously.

  Homotopy helps classify spaces by preserving key topological features like connectivity and the number of holes.

- **Topological Invariant:** It is a property of a space that remains unchanged under homeomorphisms and helps distinguish non-equivalent spaces. Examples include the number of components and homology groups.

  Example: a sphere and a cube are topologically equivalent because they can be deformed into each other without tearing or gluing. However, a torus (donut shape) is not equivalent to a sphere because it has a hole on its surface that cannot shrink to a point.

1.2 Homology

Homology is a fundamental invariant in topology that captures information about a space's structure by examining its holes/cavities of various dimensions. Homology identifies holes of different dimensions in a space:

0-holes (H_0): Connected components of the space.

1-holes (H_1): Non-contractible loops (e.g., circles on a torus).

2-holes (H_2): Cavities (e.g., hollow interiors of a sphere or torus).

Betti numbers: It is defined as the rank of a homology group $H_k(X)$ that counts the number of independent k-dimensional holes in a space.

Example:

1. A torus has $\beta_0 = 1$ (one connected component), $\beta_1 = 2$ (two loops), and $\beta_2 = 1$ (one cavity).

2. A solid ball has $\beta_2 = 0$ (no 2-holes), but removing two inner balls creates $\beta_2 = 2$ (two separate cavities).

1.2.1 Computation of Homology

Homology is a mathematical operation whose inputs are topological spaces and outputs are groups. Homology is a fundamental concept in topology that helps classify and analyze different topological spaces by studying their holes or cavities in various dimensions. It assigns homology groups $H_k(X)$ to a given space X, where each group captures the structure of non-collapsible k-dimensional submanifolds, often referred to as k-holes.

Key Idea Behind Computing Homology:

- A k-hole in a space X is detected by finding its k-dimensional boundary that cannot be continuously deformed to a point.
- Instead of directly identifying missing regions in X, homology uses the boundaries of these regions to infer their existence.
- A fundamental principle in topology states: The boundary of a boundary is always empty. Mathematically, if Ω is a $(k+1)$-dimensional domain in X with boundary $S = \partial\Omega$, then:

  $$\partial(\partial\Omega) = \partial S = \emptyset$$

  This means that to detect cavities, we first find k-dimensional cycles (closed submanifolds with no boundary in X), and then remove those that bound a domain, leaving only the true cavities.

In TDA, we work with **simplicial homology**, which represents a space X as a simplicial complex (a collection of simple building blocks like vertices, edges, and triangles). This approach allows for efficient computation of homology using discrete structures. We will describe simplicial homology with $\mathbb{Z}_2$ coefficients, the most common approach in TDA. The main steps are:

1. **Representation of k-simplices:** A k-simplex is represented by listing its $k+1$ vertices.

   Example: A 1-simplex (edge) with endpoints v_2, v_4 is written as: $e = [v_2, v_4]$. A 2-simplex (triangle) with vertices v_1, v_5, v_7 is written as: $\tau = [v_1, v_5, v_7]$

   Since we are using $\mathbb{Z}_2$-coefficients, the order of the vertices does not matter.

2. **k-chains $C_k(X)$:** The group $C_k(X)$ consists of all k-dimensional subcomplexes, which represent k-submanifolds (with or without boundary). Let $\sigma_1, \sigma_2, \ldots, \sigma_n$ be the k-simplices in X, and let the coefficient group be $\mathbb{Z}_2$. Then a k-chain is:

   $$c = a_1\sigma_1 + a_2\sigma_2 + \cdots + a_n\sigma_n \quad \text{where } a_i \in \mathbb{Z}_2$$

   and $C_k(X) \cong \mathbb{Z}_2^n$.

   Example: A simplicial complex X with three edges e_1, e_2, e_3 has a group: $C_1(X) = \mathbb{Z}_2^3$ where each edge acts as a generator.

3. **Boundary Operator ∂_k:** It maps k-chains to their $(k-1)$-dimensional boundaries.

   Example: For an edge $e = [v_0, v_1]$, the boundary is $\partial_1[e] = v_0 + v_1$. For a triangle $\tau = [v_0, v_1, v_2]$, the boundary is $\partial_2\tau = [v_0, v_1] + [v_1, v_2] + [v_2, v_0]$.

   The boundary operator is a linear transformation and can be represented as a matrix.

4. **k-cycles $Z_k(X)$:** The kernel of the boundary operator ∂_k, meaning all k-chains with no boundary.

   Example: Consider Figure 1.1 with four vertices $\{v_1, v_2, v_3, v_4\}$ and four edges $\{e_1, e_2, e_3, e_4\}$ where $e_1 = [v_0, v_1]$, $e_2 = [v_1, v_2]$, $e_3 = [v_2, v_3]$ and $e_4 = [v_3, v_4]$.

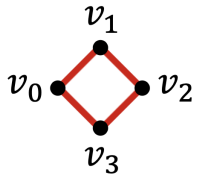


Figure 1.1: A square-shaped simplicial complex

Here, $C_1(\mathcal{X}) = \mathbb{Z}_2^4$ and $C_0(\mathcal{X}) = \mathbb{Z}_2^4$. Suppose $\sigma_1 = e_1 + e_2$; then

$$\partial_1 \sigma_1 = (v_0 + v_1) + (v_1 + v_2) = v_0 + v_2,$$

meaning that σ_1 represents a 1-subcomplex with a boundary. However, if $\sigma_2 = e_1 + e_2 + e_3 + e_4$, then

$$\partial_1 \sigma_2 = (v_0 + v_1) + (v_1 + v_2) + (v_2 + v_3) + (v_3 + v_0) = 2(v_0 + v_1 + v_2 + v_3) = 0,$$

indicating that σ_2 represents a 1-subcomplex with no boundary.

5. **k-boundaries $B_k(X)$:** While we identified all k-subcomplexes with no boundary, we must eliminate the ones that bounds a domain in X. The set of k-boundaries, denoted by $B_k(X)$, is the image of the boundary operator ∂_{k+1}. Mathematically, $B_k(X) = \partial_{k+1} C_{k+a}(X) \subset C_k(X)$.

   Example: Consider a triangle $\tau = [v_1, v_2, v_3]$ with edges e_1, e_2 and e_3.

   The boundary operator ∂_2 maps 2-chains (triangles) to 1-chains (edges):

   $$\partial_2 \tau = e_1 + e_2 + e_3$$

   This means the boundary of the triangle is the sum of its three edges.

   Also,

   $$\partial_1(e_1 + e_2 + e_3) = (v_0 + v_1) + (v_1 + v_2) + (v_2 + v_0) = 0$$

   So, $e_1 + e_2 + e_3$ has no boundary, meaning it is a 1-cycle.

   We found that $e_1 + e_2 + e_3$ is a cycle because it has no boundary. However, this cycle is also the boundary of the 2-simplex τ, meaning it is a 1-boundary. Since every 1-cycle is also a 1-boundary in this case, there are no true 1-dimensional holes.

6. **Homology Group $H_k(X)$:** It is defined as the quotient group:

   $$H_k(X) = Z_k(X)/B_k(X)$$

   This captures the number of true k-dimensional holes in X by identifying k-cycles that are not boundaries of higher-dimensional simplices.

   Example: Consider hollow tetrahedron as in Figure 1.2.

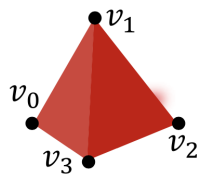


Figure 1.2: Hollow Tetrahedron

The hollow tetrahedron X consists of four 2-simplices (triangles): $\tau_1 = [v_0, v_1, v_2], \tau_2 = [v_1, v_2, v_3], \tau_3 = [v_0, v_1, v_3]$ and $\tau_4 = [v_0, v_2, v_3]$, six 1-simplices (edges): $e_1 = [v_0, v_1], e_2 = [v_1, v_2], e_3 = [v_2, v_3], e_4 =$

$[v_3, v_0], e_5 = [v_0, v_2]$, and $e_6 = [v_1, v_3]$, four 0-simplices (vertices): v_0, v_1, v_2 and v_3. Thus, the chain groups are:

$$C_2(X) = \mathbb{Z}_2^4, \quad C_1(X) = \mathbb{Z}_2^6, \quad C_0(X) = \mathbb{Z}_2^4.$$

Computation of $H_2(X)$ (Detecting 2D Cavities): The kernel of ∂_2, denoted $Z_2(X)$, consists of 2-chains whose boundary sums to zero. The only generator of $Z_2(X)$ is $\tau_1+\tau_2+\tau_3+\tau_4$. Since there is no 3-simplex in X, the image of ∂_3 is trivial. Thus, the second homology group is:

$$H_2(X) = Z_2(X)/B_2(X) = \mathbb{Z}_2.$$

This confirms that X has one 2-dimensional cavity, consistent with it being a hollow tetrahedron.

Computation of $H_1(X)$ (Detecting 1D Holes): The 1-cycles $Z_1(X)$ consist of edge combinations whose boundaries vanish *i.e.* $Z_1(X) = \mathbb{Z}_2^3$. The 1-boundaries $B_1(X)$, which are the boundaries of the triangles, are $B_1(X) = \mathbb{Z}_2^3$. Since every 1-cycle is also a 1-boundary, we get:

$$H_1(X) = Z_1(X)/B_1(X) = \{0\}.$$

Thus, there are no 1-dimensional holes in X.

Computation of $H_0(X)$ (Detecting Connected Components): The 0-cycles $Z_0(X)$ consist of all vertex sums with no boundary. The 0-boundaries $B_0(X)$ are given by edges connecting vertices. Since X is a single connected component, we obtain:

$$H_0(X) = \mathbb{Z}_2.$$

This confirms that X has **one connected component**, as expected.

Thus, X is a hollow tetrahedron with a single enclosed cavity, no 1-dimensional holes, and a single connected component.

Chapter 2

Persistent Homology and Persistence Diagram

The main idea behind persistent homology (PH) is to capture the hidden shape patterns in the data by using algebraic topology tools. PH achieves this by keeping track of the evolution of the topological features (k-holes, components, loops, and cavities) created in the data while looking at it in different resolutions. In simple terms, PH can be summarized as follows:

1. **Filtration:** Generate a nested sequence of simplicial complexes derived from the data.
2. **Persistence Diagrams:** Record the evolution of topological features across this sequence.

2.1 Constructing Filtrations

To study the given data in different resolutions, PH generates a nested sequence of simplicial complexes $K_1 \subset K_2 \subset \cdots \subset K_n$ induced from the data. Such a sequence is called a *filtration*. The filtration process involves examining the data at different resolutions by adjusting a "scale parameter". The choice of this "scale parameter" can greatly influence the performance of the method. For each data type, there are well-established methods to construct filtrations that have proven highly effective in their respective contexts.

2.1.1 Filtrations for Point Clouds

Filtrations for point clouds involve constructing a nested sequence of simplicial complexes to analyze the topological structure of the data at different scales. Given a point cloud $X = \{x_1, x_2, \ldots, x_m\} \subset \mathbb{R}^N$ we define a filtration by progressively increasing a scale parameter r, leading to a sequence of simplicial complexes $K_1 \subset K_2 \subset \cdots \subset K_n$.. This process allows for multi-scale analysis of the shape and connectivity of the point cloud.

Neighborhood-Based Filtration: A simpler way to define a filtration is by considering r-neighborhoods. Let $B_r(x) = \{y \in \mathbb{R}^N \mid d(x, y) \leq r\}$ be the closed r-ball around a point x, then r-neighborhood of X is:

$$N_r(X) = \bigcup_{i=1}^{m} B_r(x_i),$$

Choosing a sequence of increasing radii $0 = r_1 < r_2 < \cdots < r_n$, where r_n is the maximum distance between points in X, results in a nested sequence $N_{r_1}(X) \subset N_{r_2}(X) \subset \cdots \subset N_{r_n}(X)$, forming a topological filtration.

Rips Complex (Vietoris-Rips Complex): To effectively use computational tools, we translate $N_r(X)$ into a simplicial complex filtration. A k-simplex $\sigma = [x_{i_0}, x_{i_1}, \ldots, x_{i_k}]$ belongs to the Rips complex $R_r(X)$ if all pairs in σ are within distance r, i.e.,

$$d(x_{i_m}, x_{i_n}) < r, \quad \forall 0 \leq m, n \leq k.$$

Example: Consider 8-shaped point cloud with 24 points.

```
theta1 = np.linspace(0, 2*np.pi, 12, endpoint=False)
theta2 = np.linspace(0, 2*np.pi, 12, endpoint=False)

scale_upper = 0.5
x1 = scale_upper * np.sin(theta1)
y1 = scale_upper * np.cos(theta1) + 1

x2 = np.sin(theta2)
y2 = np.cos(theta2) - 0.5

points = np.vstack((np.column_stack((x1, y1)), np.column_stack((x2, y2))))
points += 0.02 * np.random.randn(*points.shape)
```

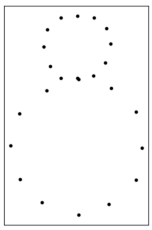


Figure 2.1: 8-shaped point cloud

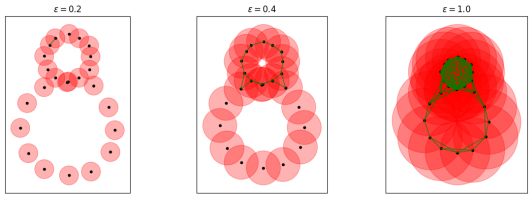


Figure 2.2: Filtration steps for different ϵ

```
from gudhi import RipsComplex

def draw_vr_complex(points, epsilon, ax):
    patches = [Circle(p, epsilon) for p in points]
    collection = PatchCollection(patches, color='red', alpha=0.3)
    ax.add_collection(collection)

    ax.scatter(points[:, 0], points[:, 1], c='black', s=10, zorder=2)

    rips_complex = RipsComplex(points=points, max_edge_length=epsilon)
    simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)

    for simplex in simplex_tree.get_skeleton(2):
        if len(simplex[0]) == 2:
            i, j = simplex[0]
            p1, p2 = points[i], points[j]
            ax.plot([p1[0], p2[0]], [p1[1], p2[1]], c='green', linewidth=0.8)

    ax.set_aspect('equal')
    ax.set_xticks([])
    ax.set_yticks([])
    ax.set_title(f"$\\varepsilon = {epsilon}$", fontsize=12)

fig, axs = plt.subplots(1, 3, figsize=(12, 4))

for ax, eps in zip(axs, [0.2, 0.4, 1.0]):
    draw_vr(points, epsilon=eps, ax=ax)

plt.tight_layout()
plt.show()
```

At $\epsilon = 0.2$: Small clusters of connected points appear, but the two loops of the "8" are still separate. Only 1-simplices (edges) form between very close points.

At $\epsilon = 0.4$: The two loops of the "8" become more connected, but there is still a visible hole in each loop. 2-simplices (triangles) start forming.

At $\epsilon = 1.0$: The two loops merge into a single connected structure. Many higher-dimensional simplices appear, filling in the shape. Eventually, the structure loses its detailed shape and resembles a single cluster.

2.1.2 Filtrations for Images.

Filtrations for images involve progressively revealing structures within an image using a sequence of nested binary images. Two commonly used methods are:

1. **Sublevel Filtration:** A sublevel filtration consists of a sequence of binary images where pixels are included if their intensity is below or equal to a threshold. This filtration helps analyze how features emerge as the intensity threshold increases.
2. **Superlevel Filtration** A superlevel filtration works in reverse, activating pixels with intensity greater than or equal to a threshold.

   Let us have a 5×5 image represented by pixel intensity values.

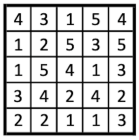


Figure 2.3: Each number represents a grayscale intensity (e.g., 1 for dark pixels, 5 for bright pixels).

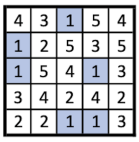


Figure 2.4: For threshold $t = 1$, only pixels with value ≤ 1 are activated.

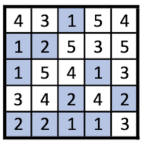


Figure 2.5: For threshold $t = 2$, pixels with value ≤ 2 are now included.

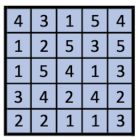


Figure 2.6: For threshold $t = 5$, all pixels are included.

2.1.3 Filtration for Graphs

Unlike standard filtrations used for point clouds and images, graph filtrations have multiple methods that can significantly impact machine learning (ML) models.

Types of Graph Filtrations

For a given graph $G = (V, \epsilon)$ there are two primary types of graph filtrations:

1. **Filtrations through node/edge functions:** Assigns a function to nodes or edges (e.g., degree, centrality, betweenness) and activates nodes in increasing order of function value. This process forms a nested sequence of subgraphs and associated simplicial complexes, often using clique complexes.
2. **Graph Distance-Based Filtrations:** It uses distances between nodes as the basis for filtration. It treats the graph as a point cloud and applies Rips filtration. It builds a sequence of graphs by adding edges between nodes within increasing distance thresholds. The final filtration leads to a complete graph.

2.2 Persistence Diagrams

After constructing the filtration $K_1 \subset K_2 \subset \cdots \subset K_n$ for a data type X, PH systematically tracks the evolution of topological features (k-holes) in the filtration sequence and records this information in a **persistence diagram**. Evolution of persistence diagram can be understood as following:

- For each $k-$hole σ, PH records its first appearance in the filtration sequence, denoted K_{io} , and its first disappearance in a later complex, K_{jo} .

- **Birth time:** We define $b_\sigma = \epsilon_{io}$ as the birth time where $\{\epsilon_i\}_1^n$ is the threshold set used for the filtration.
- **Death time:** We define $d_\sigma = \epsilon_{jo}$ as the death time ofσ where $\{\epsilon_i\}_1^n$ is the threshold set used for the filtration.
- **Lifespan of σ:** The difference $d_\sigma - b_\sigma$ is called the lifespan of σ.

  Example: If a k-hole first appears in K_3 and disappears in K_7, we mark the birth time as$b = \epsilon_3$ and the death time as$d = \epsilon_7$. The lifespan is then $\epsilon_7 - \epsilon_3$.
- We represent each $k-$hole, σ with a 2-tuple (b_σ, d_σ) to denote its birth and death times in the filtration. The collection of all such 2-tuples is called the persistence diagram (PD).
- 0 lifespan is considered a trivial topological feature, and they are represented with diagonal elements $(x = y)$ in the PD.
- **Persistence Barcode:** It uses bars instead of 2-tuples.

2.2.1 Interpretation of Persistence Diagrams

A persistence diagram $PD_k(X)$ records the k-dimensional topological features of the data X as a collection of points in $\mathbb{R}^2$. For example, $PD_0(X)$records 0-holes (components), and $PD_1(X)$ records 1-holes (loops) appearing in the filtration sequence $\{K_i\}$. Each point $q_j = (x_j, y_j)$ represents a k-dimensional hole σ_j.

How to differentiate between an important topological feature and noise?

This can be understood through an example. Suppose we have two points, $q_1 = (0.3, 9.7)$ and $q_2 = (4.2, 4.6)$, in $PD_2(X)$. For $q_1 = (0.3, 9.7)$, the cavity q_1 first appears in k_3 and persists until k_{97}, indicating a long lifespan. Conversely, for $q_2 = (4.2, 4.6)$, the cavity q_2 first appears in k_{42} and persists only until k_{46}, indicating a short lifespan. This suggests that q_1 represents an important topological feature of the data X, while q_2 is likely just topological noise.

In general, features with long lifespans (where $y - x$ is large) are located far from the diagonal and are considered significant (or "big") features. Features with short lifespans (where $y - x$ is small) are close to the diagonal and are considered insignificant (or "small") features.

Example: Let us consider 8-shaped figure. Red bars represent $PD_0(X)$ and blue bar represents $PD_1(X)$. Initially, X has 24 components that corresponding to 24 red bars. As ϵ increases, these components merge.

```
from gudhi import RipsComplex
from gudhi.persistence_graphical_tools import plot_persistence_barcode,
                                              plot_persistence_diagram

def plot_persistence(points, epsilon):
    rips_complex = RipsComplex(points=points, max_edge_length=epsilon)
    simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
    persistence = simplex_tree.persistence()

    return persistence


epsilons = [0.2, 0.4, 1.0]
for eps in epsilons:
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))

    draw_vr_complex(points, epsilon=eps, ax=axs[0])

    persistence = plot_persistence(points, epsilon=eps)
    axs[1].set_title("Persistence Barcode", fontsize=12)
    plot_persistence_barcode(persistence, axes=axs[1])

    axs[2].set_title("Persistence Diagram", fontsize=12)
    plot_persistence_diagram(persistence, axes=axs[2])

    plt.tight_layout()
    plt.suptitle(f"$\\varepsilon = {eps}$", y=1.02, fontsize=16)
    plt.show()
```

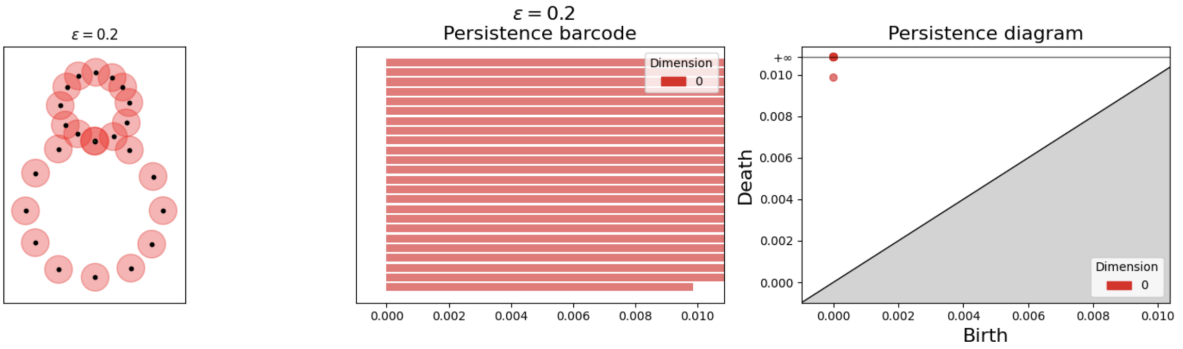


Figure 2.7: At $\epsilon = 0.2$, top 23 red bars remain.

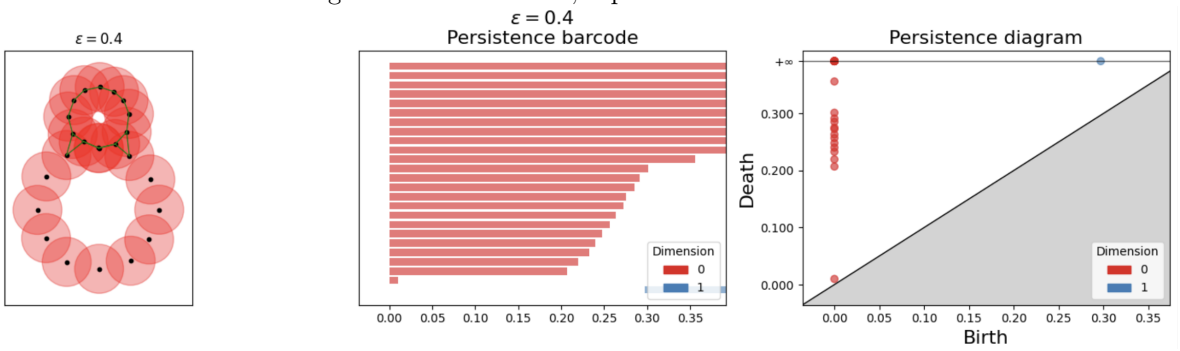


Figure 2.8: At $\epsilon = 0.4$, only 10 red bars remain. One blue bar also emerges corresponding to smaller loop around $\epsilon = 0.3$

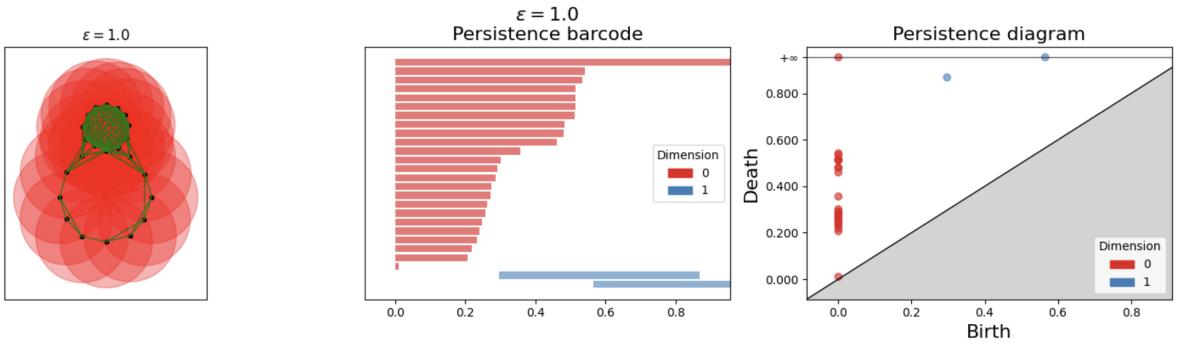


Figure 2.9: At $\epsilon = 1.0$, he space becomes fully connected and all red bars terminate except for the top ∞-bar. Two blue bars also emerges corresponding to smaller and larger loops.

Chapter 3

Topological Mode Analysis Tool

ToMATo (Topological Mode Analysis Tool) is a **density-based clustering algorithm** that leverages **topological methods** to detect clusters based on the gradient of an underlying density function. The key idea is to connect points based on function values and form a collection of **spanning trees** that correspond to **ascending regions**.

ToMATo takes in three inputs: the neighborhood graph G, the density estimator $\tilde{f}$, and the merging parameter τ.

Graph G

The method make use of graphs that do not require the geographic coordinates of the data points at hand (only pairwise distances are used) nor estimates of the density at extra points. Few methods to convert point cloud data into graphs are:

- **ϵ-neighborhood graph:** Two vertices v_i and v_j are connected if the distance $d(v_i, v_j) \leq \epsilon$. Since all distances between connected vertices are bounded by ϵ, this graph is usually considered unweighted.
- **k-nearest neighbor graph:** Vertex v_i is connected to v_j if v_j is one of the k-nearest neighbors of v_i. To obtain an undirected graph:
  - Connect v_i and v_j if one is in the k-nearest neighbors of the other (standard k-nearest neighbor graph).
  - Connect v_i and v_j only if both are in each other's k-nearest neighbors (mutual k-nearest neighbor graph).

  The edge weights correspond to the similarity s_{ij} of the respective vertices.
- **Fully connected graph:** All vertices are connected if they have positive similarity, and edges are weighted accordingly. This is meaningful when similarity models local relationships. For instance, the Gaussian similarity function

$$s(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

  can be used, where σ controls the width of the neighborhoods.

Density estimator $\tilde{f}$

A widely used density estimator is the **truncated Gaussian estimator**. The truncated Gaussian estimator is given by:

$$f(x) = \frac{1}{|L|} \sum_{i \in L} K(d(x, p_i)),$$

where the summand is defined as:

$$K(d(x,p_i)) = \begin{cases} e^{-\frac{d^2(x,p_i)}{2h}} & d(x,p_i) \leq h, \\ 0 & \text{otherwise.} \end{cases}$$

Here, $d(\cdot,\cdot)$ is the Euclidean distance, which can also be used to compute the matrix D.

Another estimator is the **distance-to-measure** to estimate the density. It computes the root-mean-squared distance to the k-nearest neighbours:

$$f(x) = \sqrt{\frac{1}{k}\sum_{i=1}^{k} d^2(x,p_i)}.$$

Merging Parameter τ

During the merging phase, ToMATo eventually merges all clusters of prominence less than into clusters of prominence at least τ. ToMATo operates in two stages. First, the algorithm is run with $\tau = +\infty$, merging all clusters to compute the PD. The PD is then analyzed to select an appropriate τ value, which determines the number of clusters. In the second run, the chosen τ is used to generate the final clustering results.

3.1 Algorithm

The ToMATo algorithm is applied in a discrete setting given some point cloud in Euclidean space. The goal is to find a suitable clustering using results from persistent homology. We only look at the zeroth homology group H_0. Some important definitions are:

Superlevel-Sets: The set $\mathcal{F}^\alpha := f^{-1}([\alpha, +\infty))$ is called the superlevel-set of index $\alpha \in \mathbb{R}$. For any $x \in U$ and $\alpha \in \mathbb{R}$, let $C(x,\alpha) \subseteq \mathcal{F}^\alpha$ denote the path-connected component of $\mathcal{F}^\alpha$ containing x.

Superlevel-Set Filtration: We call the collection of superlevel-sets $\{f^{-1}([\alpha,+\infty))\}_{\alpha\in\mathbb{R}_+}$ a superlevel-set filtration. In order for $\{\mathcal{F}^\alpha\}_\alpha$ to resemble a filtration, we have to let α go from $+\infty$ to $-\infty$.

Graph Construction $R_\delta(L)$

The dataset is represented as a graph $R_\delta(L)$, where points are connected if they are within a certain distance δ. Each vertex x_i has a function value $f(x_i)$, which can represent a density estimation or another meaningful function.

Gradient Approximation

The gradient at each vertex x_i is approximated by connecting it to its highest-function-value neighbor in the graph $R_\delta(L)$. If a vertex has no higher-value neighbors, it is declared a local maximum (i.e., a peak in the density).

Union-Find Data Structure

Used to efficiently track clusters and merge them when necessary.

Spanning Trees and Ascending Regions

The spanning trees of the graph represent clusters, where each cluster is associated with a local maximum. Trees grow by attaching points to their highest-value neighbors, forming density-based clusters.

3.1.1 Clustering

Input Parameters

- f: An n-dimensional vector storing function values for each vertex.
- D: An $n \times n$ symmetric distance matrix defining pairwise distances.
- δ: Neighborhood radius threshold for graph $R_\delta(L)$.
- τ: Persistence threshold to filter out noise.

Step 1: Sorting the Points by Function Value

Sort the index set L by decreasing function values $f(x)$. This ensures we process high-density points first.

Step 2: Initialize the Union-Find Structure

The union-find data structure U helps manage merging of clusters efficiently.

Step 3: Iterate Over the Points in Decreasing Order

For each vertex x_i, proceed as follow:

1. **Find Neighbors with Higher Function Values**
   - Compute the upper star S_i, i.e., neighbors of x_i in $R_\delta(L)$ with higher function values.
   - If $S_i = \emptyset$, then x_i is a local maximum.
     - Set $g(i) \leftarrow$ null (gradient is zero).
     - Create a new cluster containing only i.
2. **Otherwise:**
   - Connect x_i to the neighbor with the highest function value.
   - Attach x_i to the tree containing this neighbor.
   - Merge trees using union-find, ensuring clusters grow.
3. **Merge Clusters Based on Function Value and Persistence Threshold τ**
   - If $f(i)$ is too small compared to its parent cluster, it may be considered noise and ignored.

Step 4: Output Meaningful Clusters

Clusters corresponding to peaks with persistence $\geq \tau$ are returned.

3.1.2 Merging

To efficiently manage merges between clusters, we use a union-find data structure. Each entry in this structure corresponds to a collection of spanning trees within the Rips graph $R_\delta(L)$.

The root of an entry e, denoted $r(e)$, is the vertex in e with the highest function value. This represents the local peak of f within that cluster. The merging process simulates the merging of basins of attraction in a continuous density function.

Merging Strategy: We process vertices in decreasing order of function values, merging clusters based on persistence.

1. For each vertex x_i, we consider edges in its upper star (neighboring vertices with higher function values).
2. Let e_i be the entry in the union-find structure containing x_i.

3. If an adjacent entry e_j has a root $r(e_j)$ with lower function value than $r(e_i)$, we merge e_j into e_i if and only if the prominence of $r(e_j)$ is below threshold τ:

$$f_r(e_j) - f_i < \tau$$

4. Once all non-prominent neighbors are merged, we check if e_i itself should be merged.

5. Let e be the neighbor with the highest root. If the prominence of $r(e_i)$ is less than τ, merge e_i into e:

$$f_r(e_i) - f_i < \tau$$

6. The algorithm outputs the final set of clusters, filtering out those with root values below τ.

3.1.3 Final Output

The algorithm outputs a collection of clusters in the union-find structure, retaining only those with:

$$f_r(e) \geq \tau$$

This ensures that small, insignificant clusters (outliers) are removed.

3.2 Testing on 2-D Datasets

For testing five different datasets were generated using **sklearn.datasets** and clustered using K-Means, DBSCAN and ToMATo. K-Means and DBSCAN are imported using sklearn library while **ToMATo is imported using gudhi library**.

```
from sklearn.datasets import make_circles, make_moons, make_blobs
from sklearn.metrics import adjusted_rand_score
from sklearn.cluster import KMeans, DBSCAN
from gudhi.clustering.tomato import Tomato
```

- **Circle:** The dataset contains two circular clusters, with one inside the other.

```
make_circles(n_samples=n_samples, factor=0.5, noise=0.05, random_state=seed)
```

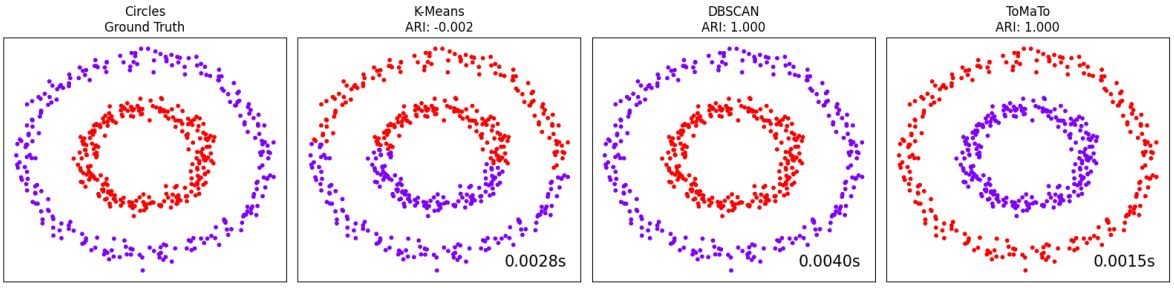


Figure 3.1: Clearly, $K-$Means suffers with non-linearly separable data. DBSCAN and ToMaTo perform perfectly, as they can handle non-spherical clusters.

- **Moons:** The dataset consists of two interleaving half-moon shapes.

```
make_moons(n_samples=n_samples, noise=0.05, random_state=seed)
```

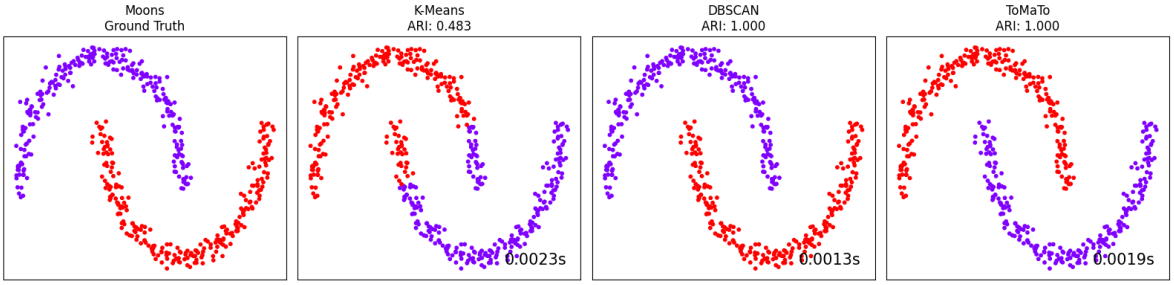


Figure 3.2: K-Means tries to partition the data into clusters by minimizing the variance within each cluster. However, due to its assumption of spherical clusters, it fails to correctly separate the two moon-shaped structures. DBSCAN and ToMaTo perform perfectly, as they can identify non-convex structures.

- **Anisotropic:** The dataset consists of three elongated clusters.

```
    X_aniso_base, y_aniso = make_blobs(n_samples=n_samples, random_state=
                                       random_state_aniso)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X_aniso_base, transformation)
```

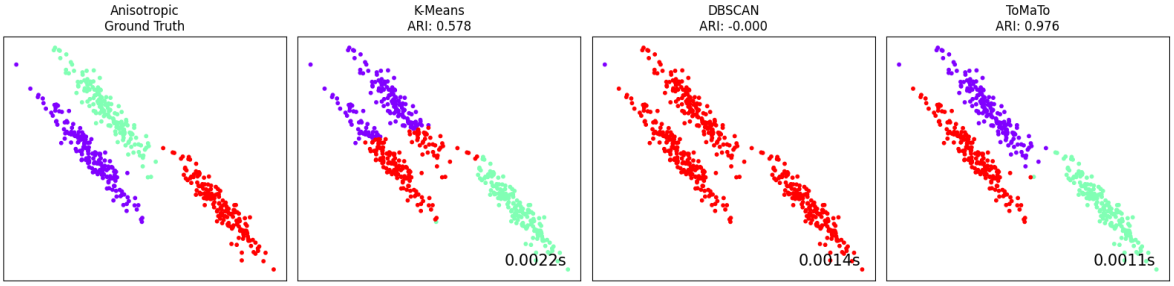


Figure 3.3: K-Means struggles again with non-convex data. DBSCAN is density-based, and works well for clusters of similar density and arbitrary shapes but due to varying densities it fails. ToMATo identifies all 3 anisotropic clusters almost perfectly.

- **Blobs:** The dataset consists of three blobs.

```
    make_blobs(n_samples=n_samples, random_state=seed)
```

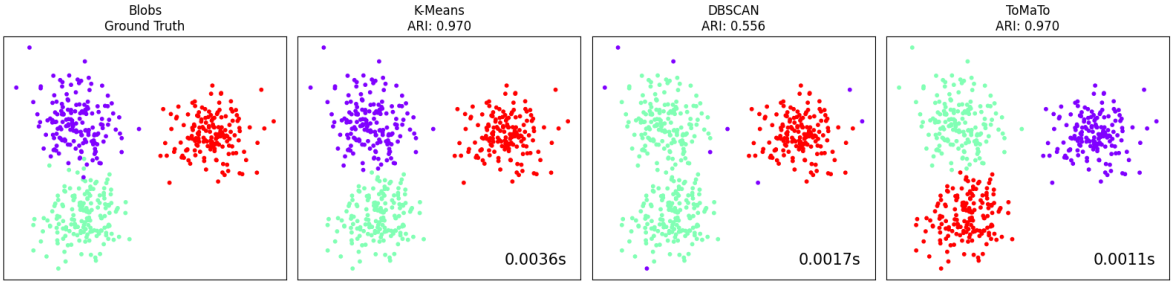


Figure 3.4: K-Means assumes spherical clusters, which aligns well with the blob structure. DBSCAN fails due to its density-based approach, which is not well-suited for this dataset. ToMaTo correctly identifies the three clusters and is the fastest method here, making it a strong alternative to $K-$Means.

- **Blobs with different variences:** The dataset consists of three blobs with varying density.

```
    make_blobs(n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], random_state=seed
                                                                        )
```

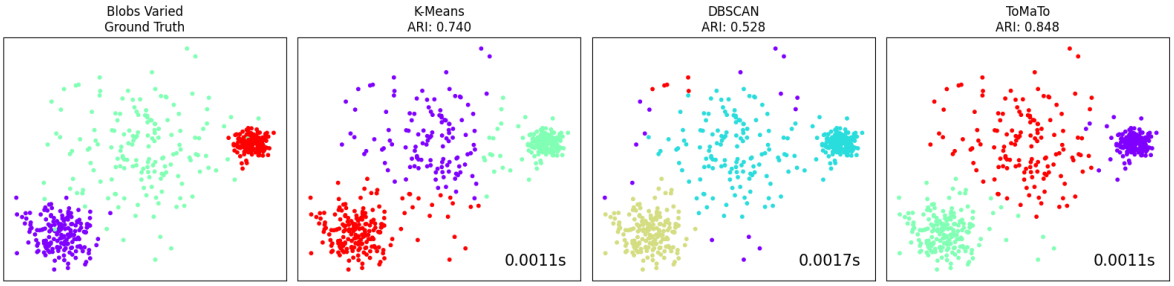


Figure 3.5: K-Means assumes spherical clusters, it struggles with clusters of varying density. DBSCAN can detect arbitrary-shaped clusters but struggles with density variations. ToMaTo seems to provide the best clustering performance on this dataset.

3.3 Testing on Real World Dataset

For this application, we use data from the Palmer Penguins dataset, which includes the target variable (species) and six additional features: island (nominal; 3 levels), culmen (bill) length (continuous), culmen (bill) depth (continuous), flipper length (continuous), body mass (discrete), and sex (nominal; 2 levels). The dataset originally contains 344 instances, but 2 with missing values were removed for this study.

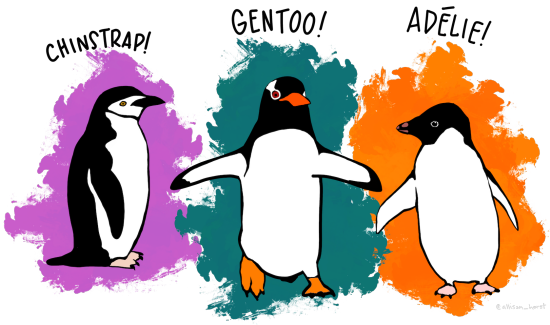


Figure 3.6: Meet the Palmer Penguins

```
df = pd.read_csv('penguins_lter.csv')
df = df.drop(columns=['studyName', 'Sample Number', 'Region', 'Island', 'Stage', '
                                        Individual ID', 'Clutch Completion', 'Date
                                        Egg', 'Sex', 'Delta 15 N (o/oo)', 'Delta 13
                                         C (o/oo)', 'Comments'])
df.dropna()
```

ToMATo is limited to 2D numerical scatter data, we created three plots to compare culmen length (x-axis) with culmen depth, flipper length (middle column), and body mass (right column) to evaluate how TDA performs.

```
df['Species'] = pd.Categorical(df['Species'])
df['Species'] = df['Species'].cat.codes
features = df.drop(columns=['Species'])
scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

scaled_features1 = scaled_features[:, [0, 1]]
```

```
scaled_features2 = scaled_features[:, [0, 2]]
scaled_features3 = scaled_features[:, [0, 3]]
```

Culmen length (x-axis) vs culmen depth (y-axis)

```
tomato1 = Tomato(n_clusters=3)
tomato1.fit_predict(scaled_features1)
```

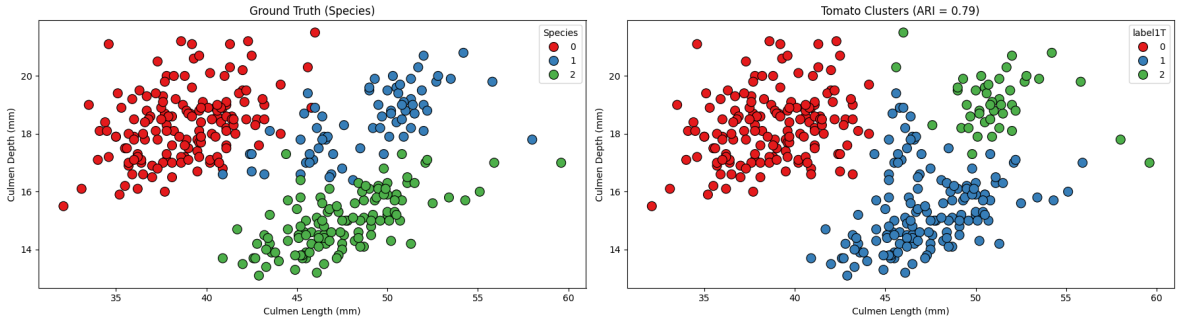


Figure 3.7: TDA performs good with ARI of 0.79.

Culmen length (x-axis) vs flipper length (y-axis)

```
tomato2 = Tomato(n_clusters=3)
tomato2.fit_predict(scaled_features1)
```

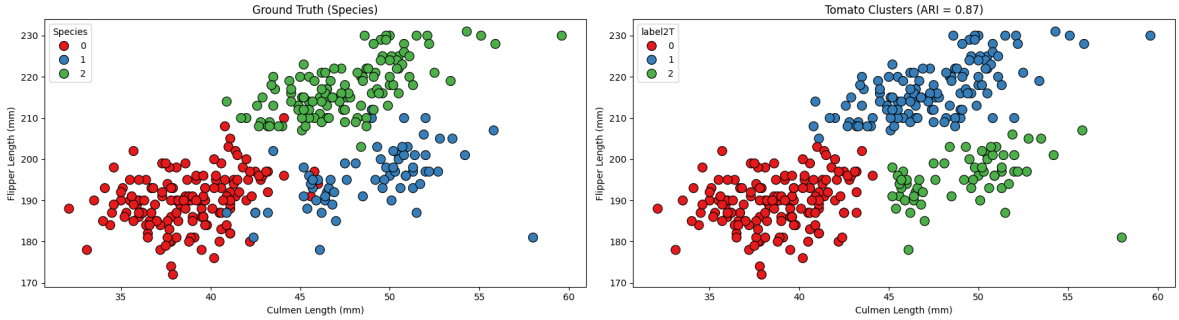


Figure 3.8: TDA performs good with ARI of 0.87.

Culmen length (x-axis) vs body mass (y-axis)

```
tomato3 = Tomato(n_clusters=3)
tomato3.fit_predict(scaled_features1)
```

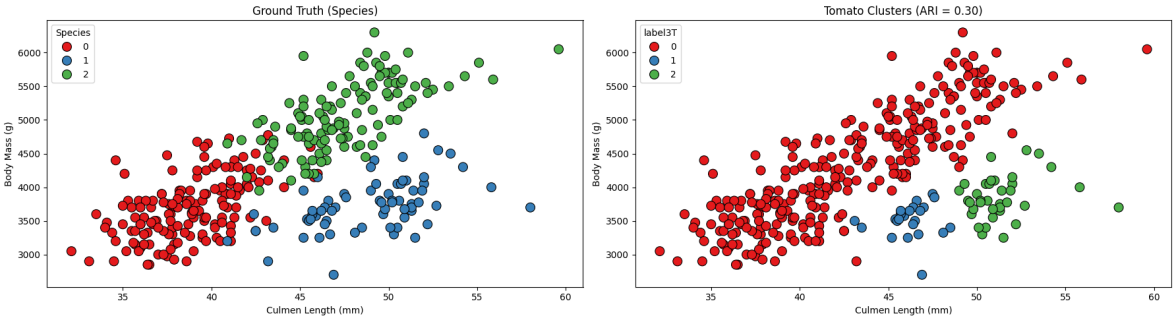
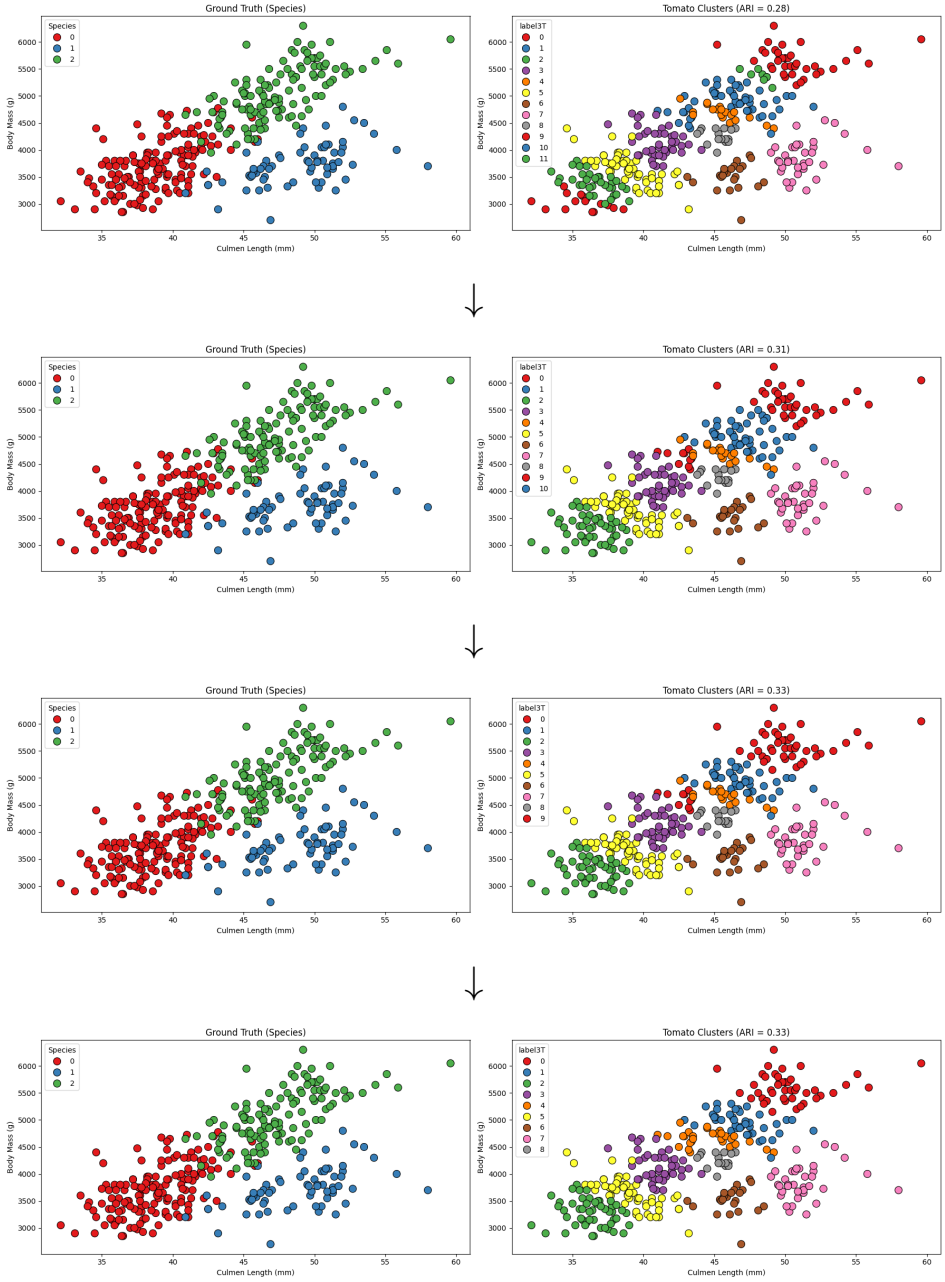
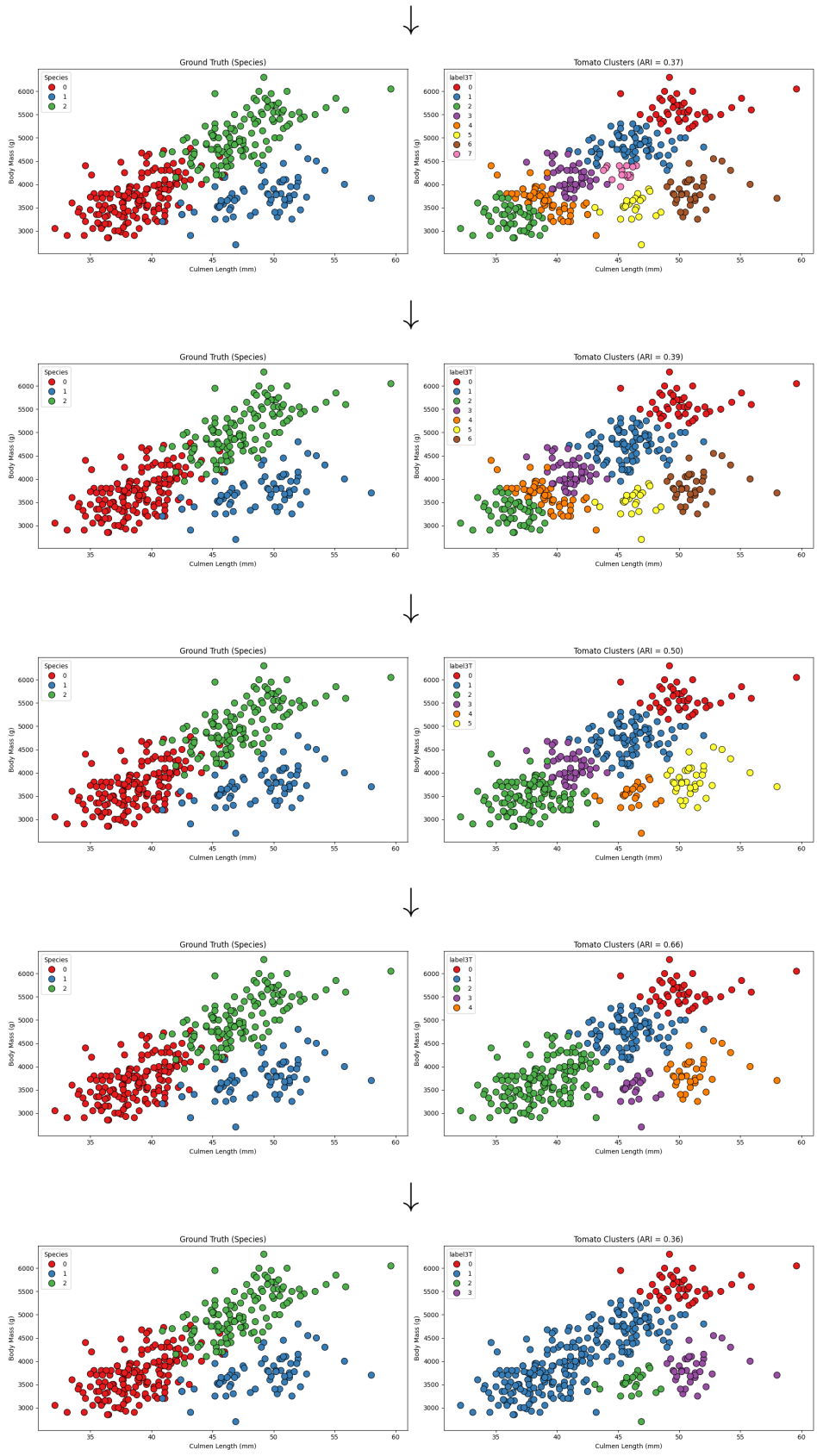


Figure 3.9: TDA does not perform good low ARI of 0.30

To dissect this problem we initialized the Tomato cluster without passing in the n_clusters parameter, and then we reduce the n_cluster eventually reaching 3 The results are as follows




Ground Truth (Species)
Species
0
1
2
Tomato Clusters (ARI = 0.37)
label3T
0
1
2
3
4
5
6
7
Tomato Clusters (ARI = 0.39)
label3T
0
1
2
3
4
5
6
Tomato Clusters (ARI = 0.50)
label3T
0
1
2
3
4
5
Tomato Clusters (ARI = 0.66)
label3T
0
1
2
3
4
Tomato Clusters (ARI = 0.36)
label3T
0
1
2
3
Body Mass (g)
Culmen Length (mm)

3.4 Testing on High-Dimensional Dataset

We evaluate the performance of our approach on the MNIST digit dataset, a widely used benchmark dataset in machine learning and deep learning research. The dataset consists of 70,000 grayscale images of handwritten digits, each of size 28×28 pixels, resulting in a 784-dimensional input feature vector per sample.

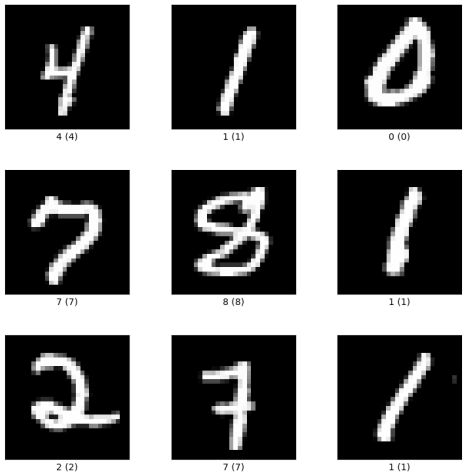


Figure 3.10: Sample MNIST Digits

3.4.1 Dimensionality Reduction

Due to the high dimensionality of the original feature space and the associated computational complexity, we first apply dimensionality reduction using a feedforward neural network. The architecture of the network is as follows:

$$784 \to 128 \to \mathbf{32} \to 10$$

where:

- The input layer consists of 784 neurons, corresponding to the flattened pixel values.
- A hidden layer with 128 neurons applies a non-linear activation function (ReLU).
- A bottleneck layer with 32 neurons serves as a low-dimensional representation of the data.
- The output layer consists of 10 neurons, corresponding to the 10 digit classes.

The network is trained using categorical cross-entropy loss and optimized with the Adam optimizer. The 32-dimensional embeddings obtained from the bottleneck layer serve as the feature representations for further processing.

3.4.2 Clustering Using the ToMATo Algorithm

After obtaining the 32-dimensional embeddings, we apply the ToMATo algorithm for clustering. The algorithm is initialized using a k-NN graph with $k = 200$ and $k = 300$. For visualization, we utilize t-SNE. The results are summarized below.

Table 3.1: Adjusted Rand Index (ARI) for Different Methods

Method	ARI
ToMATo ($k = 200$)	0.7844
ToMATo ($k = 300$)	0.7432
K-Means	0.5203

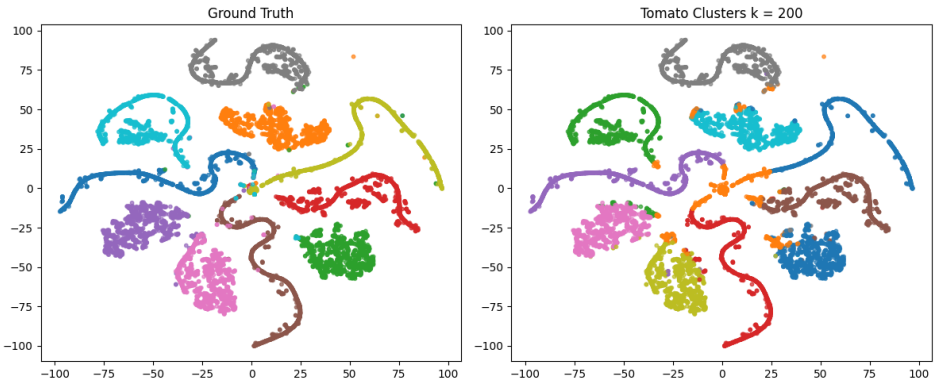


Figure 3.11: Clustering results using ToMATo ($k = 200$).

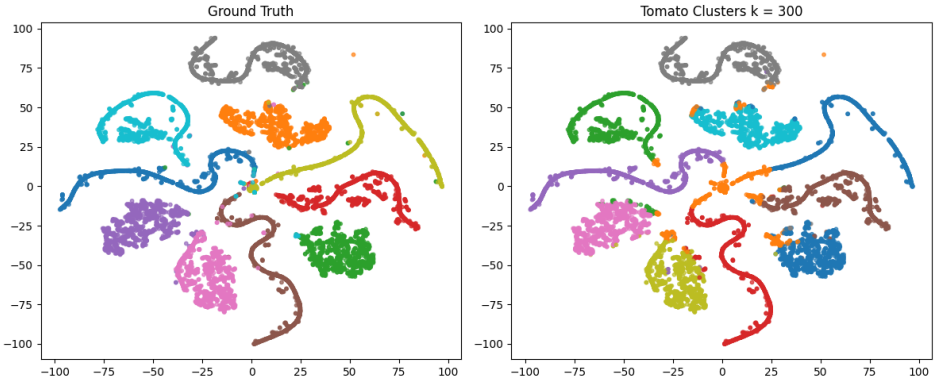


Figure 3.12: Clustering results using ToMATo ($k = 300$).

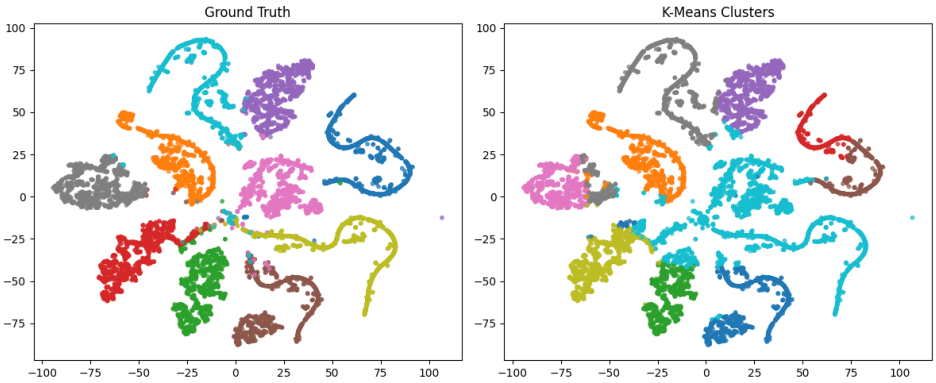


Figure 3.13: Clustering results using K-Means.

Class-Wise Accuracies

Table 3.2: Class-Wise Accuracies for ToMATo ($k = 200$) and K-Means

Class	ToMATo ($k = 200$)	K-Means
0	0.9450	0.9160
1	0.9090	0.9160
2	0.8730	0.5150
3	0.8970	0.8530
4	0.8820	0.0000
5	0.9180	0.8460
6	0.9110	0.9140
7	0.9720	0.5210
8	0.2020	1.0000
9	0.9500	0.0000
Overall	0.8459	0.6481

K-Means completely fails on classes 4 and 9 and underperforms on classes 2 and 7. ToMATo underperforms on class 8 but provides better overall performance.

Cluster Analysis

Since this is an unsupervised task, we analyze the most frequent predicted cluster for each true class.

Table 3.3: Most Frequent Predicted Clusters for Each True Class

True Class	K-Means	ToMATo
0	5 (916)	4 (945)
1	8 (916)	9 (909)
2	7 (515)	0 (873)
3	4 (853)	5 (897)
4	0 (615)	6 (882)
5	2 (846)	3 (918)
6	0 (914)	8 (911)
7	9 (521)	7 (972)
8	6 (1000)	0 (798)
9	6 (1000)	2 (950)

K-Means groups classes 4 and 6 together as class 0, and classes 9 and 8 together. In contrast, ToMATo misclassifies class 8 and class 2 together but provides a more balanced clustering overall.

3.5 Limitations

Parameter Sensitivity and Tuning

ToMaTo algorithm is highly sensitive to parameters as it requires two free parameters: neighborhood scale (δ)/number of neighbours(k) and persistence threshold. Small variations in them can alter merging outcomes.

Example 1: Consider the following point cloud data.

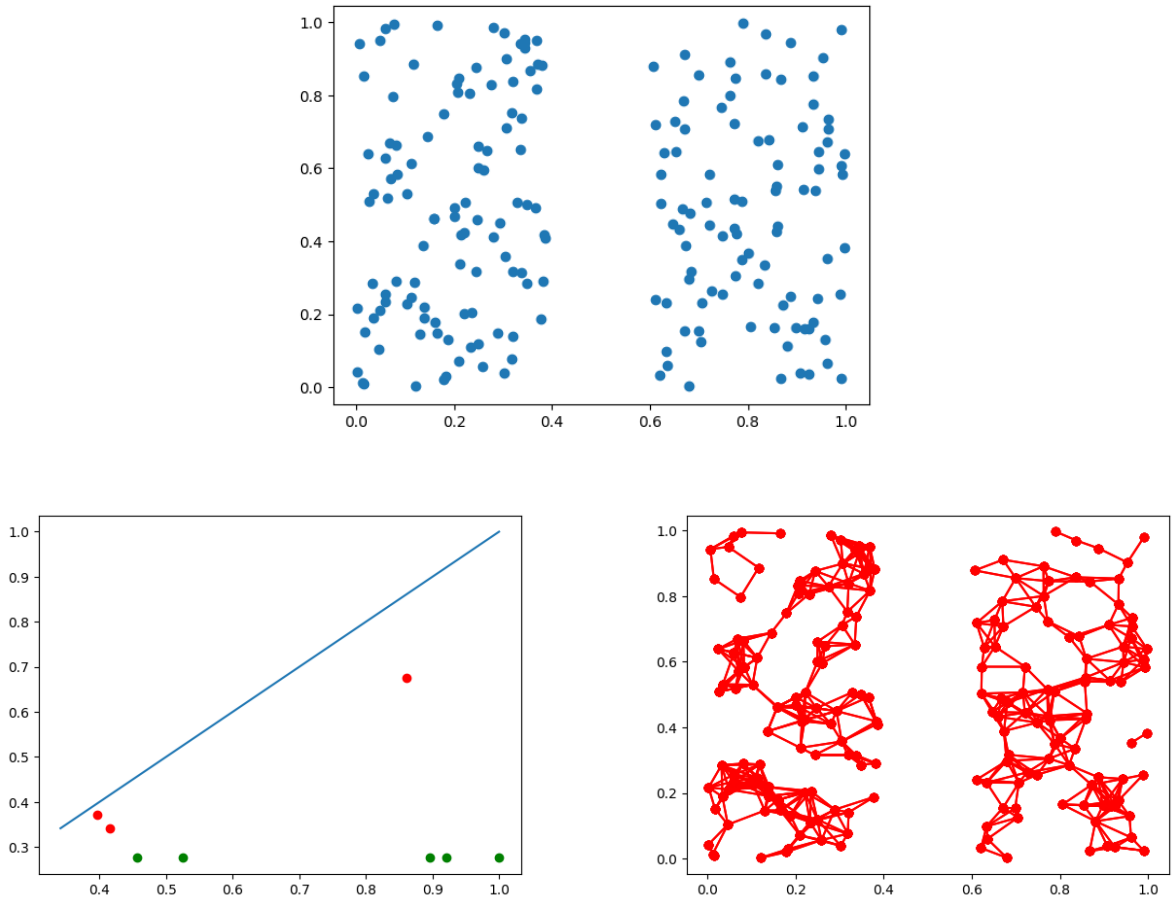


Figure 3.14: When we initialize the graph with an epsilon-ball neighborhood for $r = 0.1$, we observe that the algorithm 'naturally' detects 8 clusters, which can be later merged using persistence.

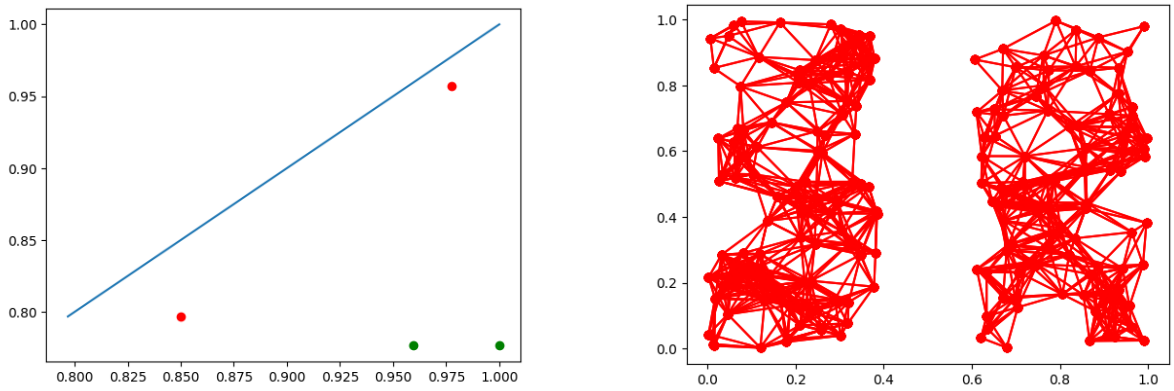


Figure 3.15: When we initialize the graph with an epsilon-ball neighborhood for $r = 0.13$, we observe that the algorithm 'naturally' detects 4 clusters, which can be later merged using persistence.

Example 2: While clustering MNIST digits we used kNN for initial graph:

- for k = 5 we get *The number of clusters required 10 is smaller than the number of connected components 46*
- for k = 200 we get the ARI score of 0.7844
- for k = 300 we get the ARI score of 0.7432

- for k = 1000 we get *The number of clusters required 10 is larger than the number of mini-clusters 2*

Dependence on Density Estimation

ToMATo relies on accurate estimation of the density function f (e.g., via kernel methods). Misestimation of f can lead to incorrect clustering.

High-Dimensional Limitations

Persistent homology via Rips complexes is practical only in moderate dimensions. High-dimensional data increases sparsity and computational challenges which may lead to unreliable clustering outcomes.

References

- Baris Coskunuzer and Cüneyt Gürcan Akçora. *Topological Methods in Machine Learning: A Tutorial for Practitioners*. University of Texas at Dallas, USA; University of Central Florida, USA, 2023. Available at: https://arxiv.org/abs/2409.02901
- Frédéric Chazal, Leonidas J. Guibas, Steve Y. Oudot, and Primoz Skraba. *Persistence-Based Clustering in Riemannian Manifolds*. 2013. Available at: https://doi.org/10.1145/2535927
- Kara Combs and Trevor J. Bihl. *Clustering and Topological Data Analysis: Comparison and Application*. 2023. Available at: https://aisel.aisnet.org/hicss-56/da/big_data_and_analytics/4/
- Bastian Rieck. *TDA for ML - Lec1*. 2020. Available at: https://bastian.rieck.me/talks/ECML_PKDD_2020_Lecture_1.pdf
- Guide ToMATo - Jupyter Notebook. *ToMATo: Topological Mode Analysis Tool*. Available at: https://github.com/xetaxe/ToMATo-Notebook/blob/master/Guide%20ToMATo.ipynb