TOPOLOGICAL DATA ANALYSIS DRIVEN FEATURE GENERATION IN MACHINE LEARNING MODELS

By

Danielle Barnes

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computational Mathematics, Science, and Engineering—Doctor of Philosophy

2024

ABSTRACT

Topological data analysis (TDA) is an emerging field in data science, with origins in algebraic topology. I focus on two main disciplines of topological data analysis, mapper and persistent homology. Mapper is an algorithm to construct a graph that is similar to a Reeb graph [1], allowing for the abstraction of shape from data. Persistent homology is way to measure features in a dataset, and returns a set of points (a persistence diagram) that represents the structure of the dataset. While both mapper and persistent homology are effective tools in their own right, a significant area of research includes using features created from these topological tools in machine learning algorithms. In this dissertation, I focus on advancing both theoretical and computational methods that allow the use of topological data analysis in machine learning algorithms.

I have developed an extension to the mapper algorithm, named predictive mapper, that uses the eigenvectors of the graph Laplacian of the geometric realization of a mapper graph, allowing features to be created from a linear combination of the eigenvector values for use in machine learning. I have also contributed to *teaspoon* [2], an open source package for topological signal processing by including new datasets and expanded functionality for featurization methods. Lastly, I have started the development of *ceREEBerus*, a python package for working with Reeb graphs while implementing a standardized software development framework for the Munch Lab.

ACKNOWLEDGEMENTS

Many people provided the support, guidance, and humor required to complete a dissertation. Many thanks to my committee members, Dr. Brian O'Shea and Dr. H. Metin Atkulga, and special thanks to my chairs Dr. Liz Munch and Dr. Jose Perea. Special thanks to my running group, which has taken many iterations over the years for keeping me sane by very insanely hopping in a van (sometimes with strangers) and running 200 miles across the country. And the most thanks to my husband, Adi, for always laughing and taking care of our dogs over the years when I was working, either at work or on my degree.

Chapter 4 was originally co-authored by Dr. Luis Polanco and Dr. Jose Perea. Chapter 4 and 5 work was supported in part by Michigan State University through computational resources provided by the Institute for Cyber-Enabled Research, with assistance from Dr. Elizabeth Munch for providing background information for computing persistence diagrams using a directional transform. Chapter 5 work was also supported and collaborative with many others in the Munch Lab and at Michigan State including Dr. Luis Polanco, Dr. Jose Perea, Elena Wang, Pat Bills, Doug Krum, Morgan Patterson, Max Chumley, Sunia Tanweer, and Dr. Firas Khasawneh.

Thank you to those outside of my committee who also read my dissertation and provided critical feedback, Dr. Tara Kilbride and Dr. Dakota Crisp. Additionally, the completion of this dissertation would not have been possible without the support of my supervisor, Jonathan Prantner. And finally, special thanks to the unsung heros of this dissertation - Keurig, Nespresso, Starbucks, and my favorite coffee maker, Spinn.

TABLE OF CONTENTS

LIST OF SYMBOLS

X	Dataset, $X \in \mathbb{R}^{pxm}$
$X(v_i)$	Subset of data X in each node $v_i \in V$
$H(M, Y)$	Measure of optimal
$\|X(v_i)\|$	Number of points in $X(v_i)$
M	A mapper graph
$V(M)$	Set of nodes in mapper graph M
$\bar{X}$	Training set, $\bar{X} \subset X$
$\hat{X}$	Testing set, $\hat{X} \subset X, \bar{X} \cap \hat{X} = \emptyset$
$\bar{X}(v_i)$	Subset of training data $\bar{X}$ in each node $v_i \in V$
$\bar{M}$	Optimized mapper graph
$V(\bar{M})$	Set of nodes in optimized mapper graph $\bar{M}$
$A(\bar{M})$	Adjacency matrix of $\bar{M}$
$\bar{v}_i \in V(\bar{M})$	Node i of optimized mapper graph M
$\bar{X}(\bar{v}_i)$	Test dataset points in node $\bar{v}_i$
$\|\bar{X}(\bar{v}_i) \cap \bar{X}(\bar{v}_j)\|$	Number of data points in $\bar{X}(\bar{v}_i) \cap \bar{X}(\bar{v}_j$
$\mathbf{y}$	Eigenvectors of $L\mathbf{y} = \lambda D\mathbf{y}$
$\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$	Finite open covering of X
$N(\mathcal{U})$	Nerve of $\mathcal{U}$
$f : X \to \mathbb{R}$	Continuous map from X to $\mathbb{R}$
$f^{-1}(U_\alpha)$	Pullback cover of U_α
$\bar{\mathcal{U}}$	Refined pullback cover of U_α
$N(\bar{\mathcal{U}})$	Nerve of $\bar{\mathcal{U}}$

$\lvert N(\bar{\mathcal{U}})\rvert$	Geometric realization of the nerve of $\bar{\mathcal{U}}$
$\rho(x)$	Map from $X \to \lvert N(\mathcal{U})\rvert$. Also refers to map from $X \to \lvert\bar{M}\rvert$
Φ	Family of gaussian distributions
β	Normalization constant
$\bar{\Phi}$	Family of gaussian distributions, normalized and a partition of unity
$\{v_0, v_1, ..., v_k\} = V$	Set of vertices of $N(\bar{\mathcal{U}})$
$(\bar{\varphi}_0(x), \bar{\varphi}_1(x), ..., \bar{\varphi}_k(x))$	Barycentric coordinates defined by $\bar{\Phi}$
$\lvert\bar{M}\rvert$	Geometric realization of the nerve of $\bar{M}$

CHAPTER 1

INTRODUCTION

This dissertation contributes to outstanding areas of Topological Data Analysis by extending theory for mapper [1], providing additional examples and datasets for use in persistent homology, and contributes to open source software to enable collaborative research.

For mapper, the algorithmic result is an abstract simplicial complex with no direct translation into a real-valued vector space, which is a requirement for use in machine learning algorithms. I propose an extension to the current mapper algorithm, that allows features to be computed from it, extending from an abstract graph into a real-valued vector space.

For persistent homology, translation needs to similarly be done from the space of persistence diagrams into a real-valued vector space, and I focus on computational aspects of existing methods. Some methods can be costly from a computational perspective, limiting the ability to easily analyze large or complex datasets. To address this, I have extended the `teaspoon` [2] python software package to include vectorized and parallel options for specific machine learning features. This work was published as part of a review paper [3] that provided additional code for researchers to use for feature creation from persistence diagrams and example datasets for analysis.

Researchers commonly access these methods through open source software, so developing and maintaining easy-to-use and purpose built software is an important challenge to support more widespread adoption of useful topological methods. I address these issues through contributions to open source software, and the development of additional packages and machine learning pipelines focused on using topological data analysis. This work includes moving to a standardized software development framework, with documentation and tutorials for collaborators to contribute.

Through this dissertation, I have made a meaningful contribution to topological meth-

ods, computational efficiency, and ability for researchers to implement topological methods across different disciplines. All code is available as open source, and many principles of standardized software development are applied, allowing for ease of maintenance and appropriate documentation, vital to researchers continuing to make contributions and improvements in topological methods.

CHAPTER 2

BACKGROUND

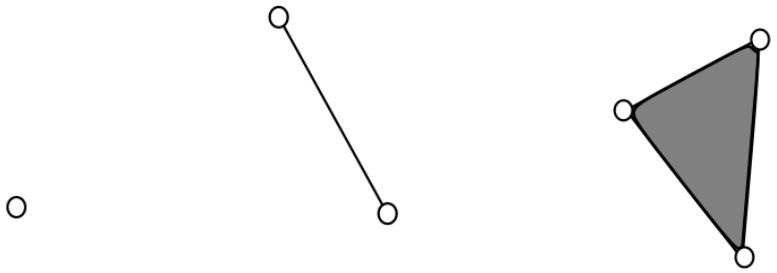
Intuitively, topology is the study of shape, and of specific interest is identifying various properties such as holes and connected components. These features allow differentiation between item types. For example, in the widely used MNIST [4] dataset, the number "8" has different topological features than the number "0". Specificially the number "8" has one connected component, and two holes; while the number "0" has one connected component, and one hole. These correspond to homology groups, and specifically persistence of these homology groups through filtrations gives a way to identify relevant topological features.

2.1 Homology

Homology concerns itself with useful properties of a simplicial complex and can provide information about the underlying space. We follow [5] in this section. Firstly we define **simplices**. Let $u_0, u_1, ..., u_k$ be points in $\mathbb{R}^d$. A point $x = \sum_{i=0}^{k} \lambda_i u_i$, with each $\lambda_i \in \mathbb{R}$, is an **affine combination** of the u_i if the $\sum_{i=0}^{k} \lambda_i = 1$. The **affine hull** is the set of affine combinations. It is a k**-plane** if the $k+1$ points are **affinely independent**, by which we mean that any two affine combinations, $x = \sum \lambda_i u_i$ and $y = \sum \delta_i u_i$, are the same iff $\lambda_i = \delta_i$ for all i. The $k+1$ points are affinely independent iff the k vectors $u_i - u_0$, for $1 \leq i \leq k$, are linearly independent. In $\mathbb{R}^d$ we can have at most d linearly independent vectors and therefore at most $d+1$ affinely dependent points.

An affine combination, $x = \sum \lambda_i u_i$ is a **convex combination** if all λ_i are non-negative. The **convex hull** is the set of convex combinations. A k**-simplex** is the convex hull of $k+1$ affinely independent points, $\sigma = \text{conv}\{u_0, u_1, ..., u_k\}$. Example simplices are shown in Figure 2.1.

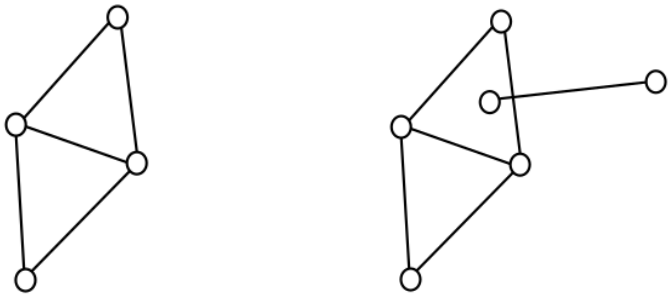
A **face** of σ is the convex hull of a non-empty subset of the u_i, and is denoted as τ with $\tau \leq \sigma$ meaning τ is a face of σ. Using this we next define a **simplicial complex** as a finite collection of simplices K such that $\sigma \in K$ and $\tau \leq \sigma$ implies $\tau \in K$ and $\sigma, \sigma_0 \in K$



(a) 0-simplex. (b) 1-simplex. (c) 2-simplex.

Figure 2.1 Illustration of simplices.

implies $\sigma \cap \sigma_0$ is either empty or a face of both. An example simplicial complex is in Figure 2.2. The former definitions refer to a geometric simplicial complex, which is a simplicial complex in Euclidean space. This definition can be generalized to an **abstract simplicial complex**, which is a finite collection of sets A such that $\alpha \in A$ and $\beta \subseteq \alpha$ implies $\beta \in A$.



(a) Simplicial complex. (b) Not a simplicial complex.

Figure 2.2 Example of a simplicial complex (left) and not a simplicial complex (right).

To understand similarities and differences of shapes, we define some additional terminology to understand homology groups, still following [5]. Let K be a simplicial complex and p a dimension. A **p-chain** is a formal sum of p-simplices in K, denoted c with coefficients a_i and p-simplices σ_i, represented as $c = \sum a_i \sigma_i$. The p-chains, along with addition, form **chain groups** denoted C_p. The **boundary** of a p-simplex is the sum of its $p - 1$

dimensional faces, so that $\delta_p\sigma = \sum_{j=0}^{p}[u_o, ..., \hat{u}_j, ..., u_p]$, with $\sigma = [u_0, u_1, ..., u_p]$ and the hat denoting that u_j is omitted. This definition extends to a p-chain so that $\delta_p c = \sum a_i \delta_p \sigma_i$. This is called the **boundary map**, $\delta_p : C_p \to C_{p-1}$. A **chain complex** is a sequence of chain groups with boundary maps: ... $\xrightarrow{\delta_{p+2}} C_{p+1} \xrightarrow{\delta_{p+1}} C_p \xrightarrow{\delta_p} C_{p-1} \xrightarrow{\delta_{p-1}}$. A **cycle** is a p-chain with an empty boundary, $\delta_p c = 0$. The set of cycles is the kernel of δ_p, $\ker\delta_p = Z_p$. A **boundary** is a p-chain that is a boundary of a $p+1$-chain, and the boundary group is denoted B_p. Finally, the $\boldsymbol{p^{th}}$ **homology group**, denoted H_p, provides useful information in describing topological features. Each p corresponds to a dimension, with H_0 representing the number of connected components, and H_1 representing 1-dimensional holes. Formally the $\boldsymbol{p^{th}}$ **homology group** is the quotient group $H_p = \text{Ker } \delta_p / \text{ Im } \delta_{p+1}$. The rank of H_p is called the p^{th} **Betti number**, β_p, and provides the number of features in the p^{th} homology group.

This is important because homeomorphic spaces have the same Betti numbers. For example, the number "8" can be identified as the same type with different handwriting since $\beta_0 = 1$ and $\beta_1 = 2$. An illustration of equivalent homology groups is in Figure 2.3.

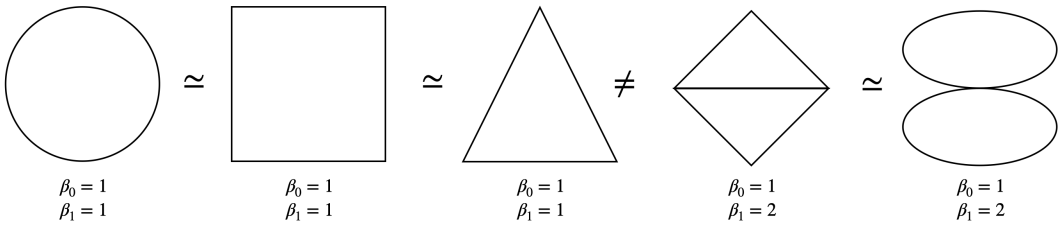


Figure 2.3 Homology groups H_0 and H_1 across different spaces.

2.2 Persistent Homology

Extending the idea from a singular simplicial complex to an ordered sequence of simplicial complexes provides a way to quantify how prominent, or persistent, specific features are for a given data set. This provides motivation for the concept of **persistent homology**, which takes a function defined on a simplicial complex, and quantifies the changes in ho-

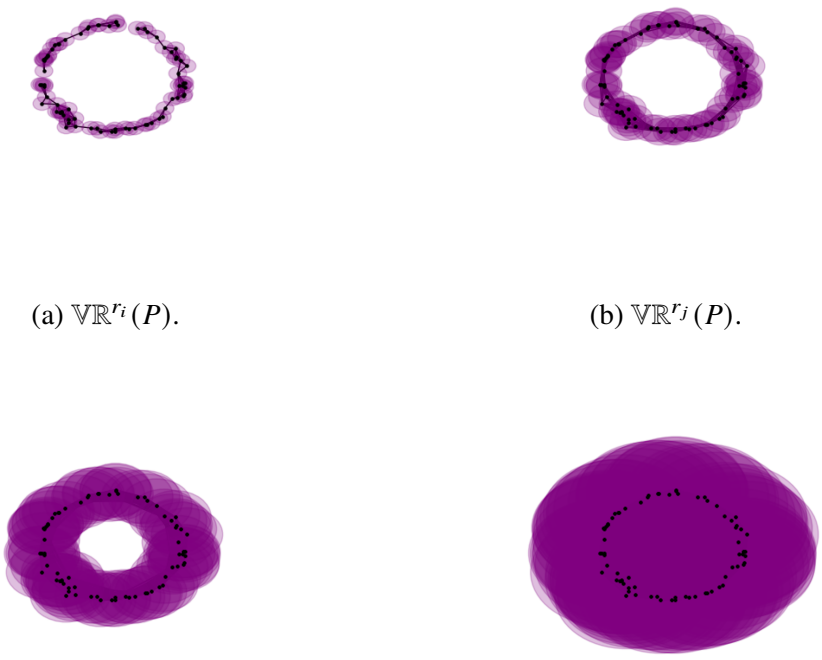
mology classes as the sublevel sets grow with increasing value of the function [6]. Features that are more persistent are more likely to be attributed to features of the underlying space from which the data was sampled.

For this section, we follow [6]. To measure the persistence of features, we start with a filtration. Using Definition 3.1, a **filtration** $\mathbb{F} = \mathbb{F}(K)$ of a simplicial complex K is a nested sequence of its subcomplexes $\mathbb{F} : \emptyset = K_0 \subseteq K_1 \subseteq ... \subseteq K_n = K$. I focus on creating this filtration on real data sets in two ways. From Definition 2.10, let (P, d) be a finite metric space. Given a real $r > 0$, the **Vietoris-Rips complex** is the abstract simplicial complex $\mathbb{VR}^r(P)$ where a simplex $\sigma \in \mathbb{VR}^r(P)$ if and only if $d(p, q) \leq 2r$ for every pair of vertices of σ. This is illustrated in Figure 2.4 where r increases through the image.

Another way to construct a filtration is through **sublevel set filtration**, where if X is a topological space and $f : X \to \mathbb{R}$ is a function, then the **sublevel sets**, $X_a = f^{-1}(-\infty, a]$, $a \in \mathbb{R}$. Then for $a_i \leq a_j \leq a_k \leq a_l$, the **sublevel set filtration** would be $X_0 \subseteq .... \subseteq X_{a_i} \subseteq X_{a_j} \subseteq X_{a_k} \subseteq X_{a_l} \subseteq ... \subseteq X$. This type of filtration is commonly used on image data, and an example is in Figure 2.5.

Following [5], we start with a filtration $\mathbb{F}(K)$ with $\emptyset = K_0 \subseteq K_1 \subseteq ... \subseteq K_n = K$. For every $i \leq j$ there is an inclusion map from the underlying space of K_i to that of K_j, hence an induced homomorphism, $f_p^{i,j} : H_p(K_i) \to H_p(K_j)$, for each dimension p. This filtration corresponds to a sequence of homology groups connected by homomorphisms, $0 = H_p(K_0) \to H_p(K_1) \to ... \to H_p(K_n) = H_p(K)$.

The **p-th persistent homology groups** are the images of the homomorphisms, $H_p^{i,j} = \text{im} f_p^{i,j}$ for $0 \leq i \leq j \leq n$. The ranks of these groups are the **p-th persistent Betti numbers**, $\beta_p^{i,j} = \text{rank} H_p^{i,j}$. The collection of persistent Betti numbers is then represented as a persistence diagram, $D(f) \subset \mathbb{R}^2$, where $D(f) \subset \mathbb{R}^2$ of f is the set of points (a_i, a_j), $a_i < a_j$, counted with multiplicity $u_p^{i,j}$ for $0 \leq i < j \leq n$. The time at which the feature appears, a_i, is known as the **birth**, and the time at which the feature disappears, a_j, is the



(a) $\mathbb{VR}^{r_i}(P)$. (b) $\mathbb{VR}^{r_j}(P)$.

(c) $\mathbb{VR}^{r_k}(P)$. (d) $\mathbb{VR}^{r_l}(P)$.

Figure 2.4 Given $i \leq j \leq k \leq l$, the Vietoris-Rips Complex is using the filtration $\mathbb{VR}^{r_i}(P) \subseteq \mathbb{VR}^{r_j}(P) \subseteq \mathbb{VR}^{r_k}(P) \subseteq \mathbb{VR}^{r_l}(P)$ is demonstrated in this image. Note that in the last filtration $\mathbb{VR}^{r_l}(P)$ the hole in H_1 no longer persists.

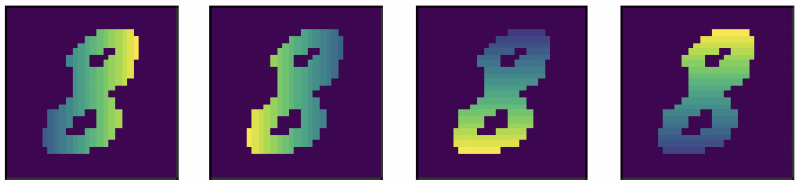


Figure 2.5 Example MNIST observation after applying directional transforms provides an application for sublevel set persistence.

death. The assumed coordinate representation for a persistence diagram is in **birth-death coordinates**, which is the ordered pair (a_i, a_j). Each feature has a corresponding coordinate

pair, with Figure 2.6 showing an example persistence diagram. Finally, by the Fundamental Lemma of Persistent Homology, the persistence diagrams encode the necessary information about persistent homology groups.

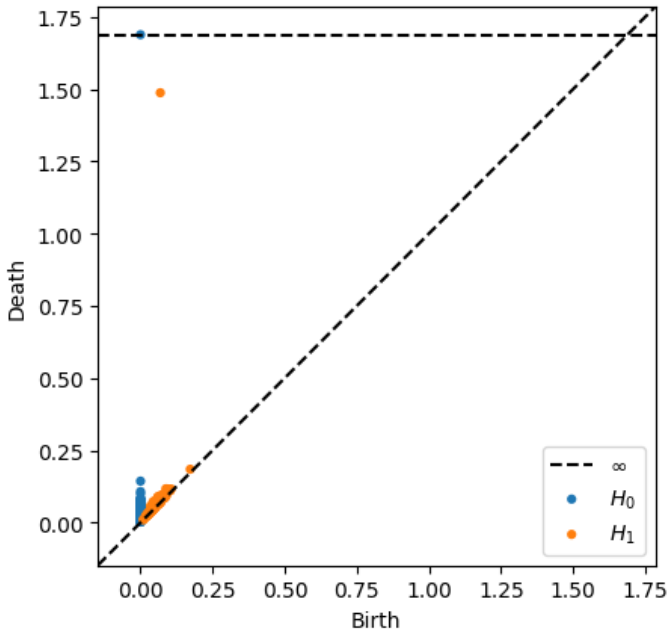


Figure 2.6 The Rips persistence diagrams in dimensions 0 (blue) and 1 (orange), for a point cloud sampled around the unit circle.

Lemma 1 (Fundamental Lemma of Persistent Homology) *Let* $\emptyset = K_0 \subseteq K_1 \subseteq \ldots \subseteq K_n = K$ *be a filtration. For every pair of indices* $0 \leq k \leq l \leq n$ *and every dimension* p*, the* p*-th persistent Betti number is* $\beta_p^{k,l} = \sum_{i \leq k} \sum_{j > l} u_p^{i,j}$

The sequence of homomorphisms can also be generalized to vector spaces, and as adapted from [7]:

Definition 2.2.1 (Persistence modules) *Let* $\mathbb{V}$ *denote a sequence of vector spaces and linear maps, of length* n*:* $V_1 \xrightarrow{p_1} V_2 \xrightarrow{p_2} \ldots \xrightarrow{p_{n-1}} V_n$*. Each* $\xrightarrow{p_i}$ *represents a map. The object* $\mathbb{V}$ *is called a persistence diagram of vector spaces, or simply a persistence module.*

We can also define a persistence diagram for a persistence module, analogous to the above definition. Then a given persistence module decomposes uniquely into interval

modules when the index set is finite. Specifically Proposition 2 (Gabriel's Theorem) from [6] guarantees a unique representation of persistence modules as birth-death pairs.

Proposition 2 (Gabriel's Theorem) *Any persistence module over a finite index set decomposes uniquely up to isomorphism into closed-closed interval modules.*

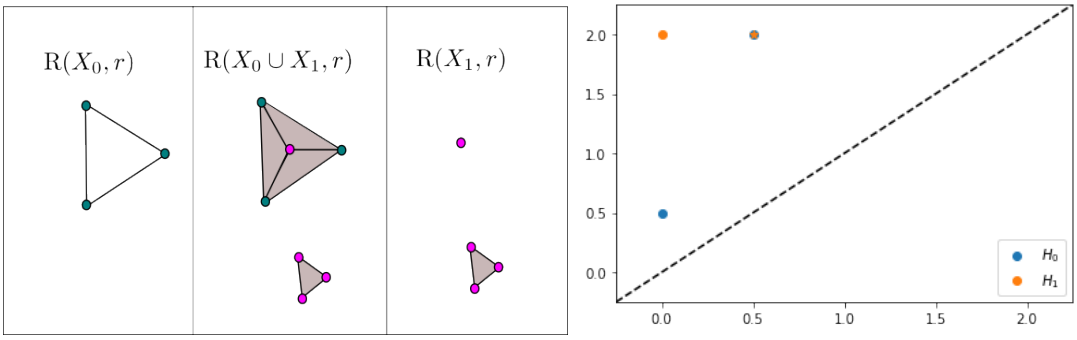
In the context of features for use in machine learning models, the types of features required need to identify relevant differences and similarities between data points, and persistence diagrams provide a way to do so. These diagrams are a set of points in the plane, allowing many different computations and transformations into the required data structures for use in machine learning applications. Many featurization methods exist for working with persistence diagrams as noted in [3].

Lastly, we follow [7] for zigzag persistence. Zigzag persistence differs from sublevel set persistence in that instead of using a monotonically increasing sequence to create a sublevel set filtration, $X_0 \subseteq .... \subseteq X_{a_i} \subseteq X_{a_j} \subseteq X_{a_k} \subseteq X_{a_l} \subseteq ... \subseteq X$, we consider filtrations that 'zigzag' to create zigzag modules.

Assuming $K_j \in K$ is a sequence of simplicial complexes, the input to zigzag persistence would be $K_0 \leftrightarrow K_1 \leftrightarrow ... \leftrightarrow K_n$, with $\leftrightarrow$ being an inclusion to either the left or right. An example of this is in Figure 2.7a along with the corresponding persistence diagram in Figure 2.7b. The persistence diagram is computed using the Vietoris-Rips complex for a fixed radius, r, and differing in zigzag filtrations.

Definition 2.2.2 (Zigzag modules) *Let $\mathbb{V}$ denote a sequence of vector spaces and linear maps, of length n: $V_1 \overset{p_1}{\leftrightarrow} V_2 \overset{p_2}{\leftrightarrow} ... \overset{p_{n-1}}{\leftrightarrow} V_n$. Each $\overset{p_i}{\leftrightarrow}$ represents either a forward map $\overset{f_i}{\rightarrow}$ or a backward map $\overset{g_i}{\leftarrow}$. The object $\mathbb{V}$ is called a zigzag diagram of vector spaces, or simply a zigzag module.*

Similar to Proposition 2 for persistence modules, the following theorem guarantees a unique decomposition into birth-death coordinates of a zigzag modules. A zigzag persis-



(a) Zigzag filtration. (b) Persistence Diagram.

Figure 2.7 Example zigzag filtration with the corresponding persistence diagram.

tence module is a **quiver representation**, denoted $\mathbb{V}(Q)$ with $Q = (N, E)$ a **quiver**, or directed graph. When the graph is finite and linear shaped (like a zigzag module) it is known as A_n-type.

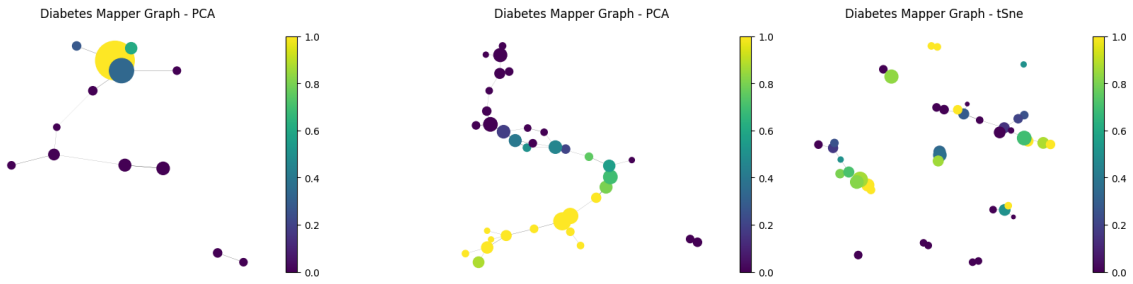
Proposition 3 *Every quiver representation* $(\mathbb{V}(Q)$ *for an* A_n*-type quiver* Q *has an interval decomposition. Furthermore, this decomposition is unique up to isomorphism and permutation of the intervals.*

This then guarantees the unique existence of birth-death pairs for zigzag persistence modules.

2.3 Mapper

Mapper, first introduced in 2007 by Singh, Memoli, and Carlsson [1], is another tool for Topological Data Analysis, commonly used for visualization and data exploration. This original paper identified diabetes subtypes using data from the Miller-Reaven diabetes study as a proof of concept. The Miller-Reaven diabetes study used a dimensionality reduction method (projection pursuit [8]) to identify juvenile and adult onset diabetes as essentially different diseases, and using Mapper a similar result was observed, see Figure 2.8. The mapper algorithm has since been used widely as a data visualization tool across various domains, including for clustering and feature selection. The identification of loops and flares can be used to identify interesting clusters and help select variables that stratify data

[9]. Some applications include breast cancer [10], voting [11], and sports [11]. While Mapper provides a way to explore, visualize and reveal a lower dimensional structure of the data, there is no obvious and direct way to use information from Mapper in machine learning models.



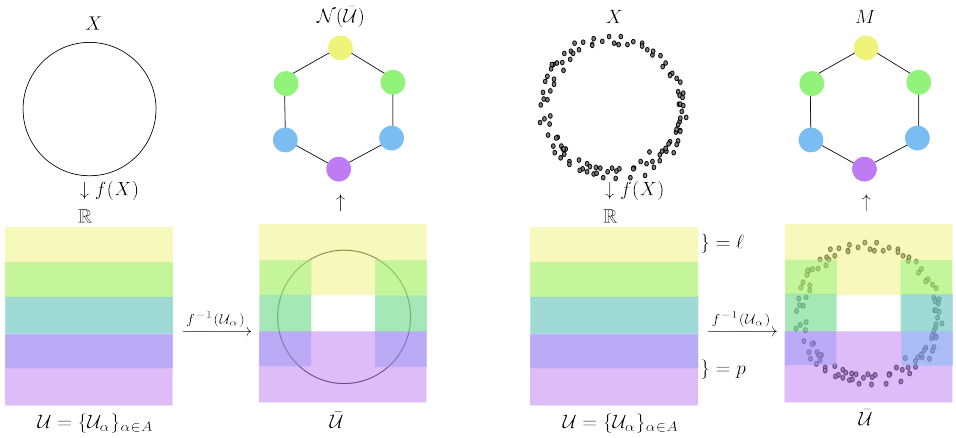
(a) (b) (c)

Figure 2.8 Diabetes data visualized using different mapper filters and different resolutions. a) Diabetes data from the Miller-Reaven study visualized using the first principle components as a filter. (b) Diabetes data from the Miller-Reaven study visualized using the first principle components as a filter at a more granular resolution. (c) Diabetes data from the Miller-Reaven study visualized using the tSne as a filter.

Next we review the topological motivation and theoretical framework for Mapper from [1]. Intuitively, the goal is for mapper to produce a graph that resembles the original space. For example, applying the mapper algorithm to a point cloud sampled from an annulus should result in a graph that is circular.

Assume, given a continuous map $f : X \to \mathbb{R}$, a finite covering $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$, we note $f^{-1}(U_\alpha)$ forms an open cover of X. We then decompose $f^{-1}(U_\alpha)$ into path connected components as a covering of X, referred to as $\bar{\mathcal{U}}$. **Mapper** is then defined as the nerve of this cover, $\mathcal{N}(\bar{\mathcal{U}})$ [12]. This is illustrated in Figure 2.9a, and discussed further in Section 3.4.

For mapper to be applied to real data, we use a statistical construction that is analogous to the topologically-motivated algorithm. First, we assume we have a sample of n data points X from a metric space, and a function $f : X \to \mathbb{R}$, known for all n points, and a distance



(a) The topological version of mapper. (b) The statistical version of mapper.

Figure 2.9 The Topological vs. Statistical Version of Mapper.

matrix D. The function $f : X \to \mathbb{R}$ is called a **filter**, and is analogous to the function from the topological version. After applying the filter function, a covering is determined on $f(x) = \mathcal{U}$. Intervals are then computed on the covering, specified by parameters ℓ and p, where ℓ is the length, and p is the percentage overlap of the intervals of $\mathbb{R}$. We then can construct a covering, $\mathcal{U} = \{U_\alpha\}_{\alpha\in A}$. For each $U_\alpha \in \mathcal{U}$, we take the set $\{f^{-1}(U_\alpha)\}_{\alpha\in A}$, which defines a cover of X. We then have subsets of X such that $X_\alpha \in f^{-1}(U_\alpha)$ and subsets of D, denoted D_α, corresponding to each X_α.

Once the data and distance matrix is partitioned into subsets, a clustering algorithm, using each D_α, is used locally to determine the number of clusters. The desired clustering algorithm also does not require specifying the number of clusters [1]. Some examples of appropriate clustering algorithms are single-linkage clustering [13] and HDBScan [14]. These clusters construct a new covering, a refined pull-back cover, denoted $\bar{\mathcal{U}}$. A node is created for each cluster, and edges are added between nodes which share a data point in their respective clusters. The statistical version of mapper is then the nerve of the refined pullback cover. The algorithm to construct the statistical version of mapper is then as follows:

1: Given a dataset X, filter function f, distance matrix D, interval length l, partition

overlap p, and local clustering algorithm a

2: Take $f(X) \to \mathbb{R}$ and using l and p create a covering of $f(X)$, $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$

3: Take the pull-back cover, $\{f^{-1}(U_\alpha)\}_{\alpha \in A}$ to get $X_\alpha \in f^{-1}(U_\alpha)$ and subsets of D, denoted D_α, corresponding to each X_α

4: Using D_α, perform local clustering using a to create a refined pullback cover, $\bar{\mathcal{U}}$

5: For each cluster, create a vertex (0-dimensional simplex), which is designated as a node

6: For each node, if the intersection of the data points in neighboring nodes is non-empty, add an edge (1-simplex) between that node and the neighboring node

The statistical construction takes point cloud data as input, and returns a graph that shows the shape and features of the underlying data with respect to the filter function. Mapper also extends to a d-dimensional version, with details available in the original paper [1]. We refer to the statistical version of Mapper as $M(X)$. This is illustrated in Figure 2.9b.

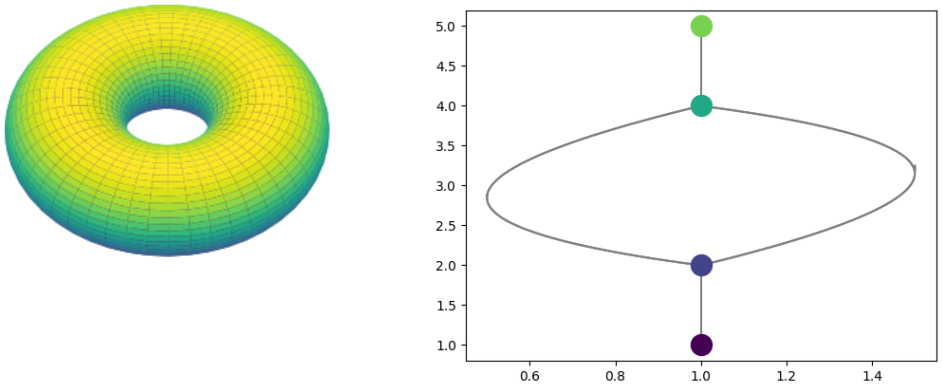
2.4 Reeb Graphs

Another important tool for topological data analysis is the Reeb graph, which provides a simplified view of potentially complex shapes while retaining important information on connected components. The use of Reeb graphs has been popularized in computer science [15] and provides utility for 3D shape estimation. Additionally, mapper is a discretized version of the Reeb graph. With the existence of metrics that calculate distances between Reeb graphs, such as bottleneck distance and interleaving distance [16], similarity of Reeb graphs and hence underlying spaces can be quantified.

To better understand the representation Reeb graphs provide, consider the torus, shown in Figure 2.10a and a Reeb graph of the height function on the torus shown in Figure 2.10b.

Following [6], the Reeb graph is defined formally as follows with an example in Figure 2.11.

Definition 2.4.1 (Reeb Graph) *Let X be a topological space with a function $f : X \to \mathbb{R}$.*



(a) Torus plotted in 3 dimensions. (b) Reeb graph of the torus.

Figure 2.10 Reeb graph representation of a Torus.

Define an equivalence relation $\sim$ *on* X *by asserting* $x \sim y$ *iff (i)* $f(x) = f(y) = \alpha$ *and (ii)* x *and* y *belong to the same connected component of the level set* $f^{-1}(\alpha)$. *Let* $[x]$ *denote the equivalence class of* $x \in X$. *The Reeb graph* R_f *of* $f : X \to \mathbb{R}$ *is the quotient space* $X/\sim$, *i.e. the set of equivalent classes equipped with the quotient topology. Let* $\Phi : X \to R_f, x \mapsto [x]$ *denote the quotient map.*

Through this construction, the Reeb graph preserves relationships between the connected components of level sets [17]. The Mapper algorithm is a generalization of the Reeb graph construction for real-valued datasets, however, algorithms for their construction have only been available recently [18], allowing additional applications for Reeb graphs and computations on Reeb graphs. In this dissertation, I focus on basics of working with Reeb graphs, and assume the Reeb graph has been provided as input.

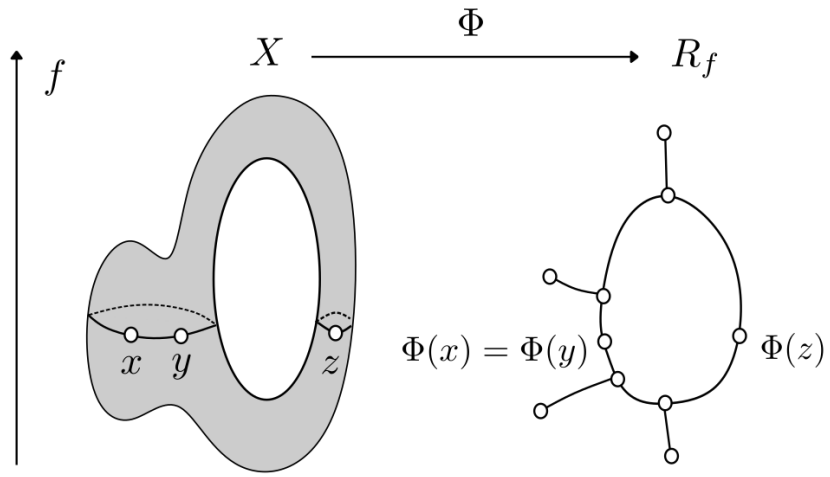


Figure 2.11 Reeb graph R_f of the function $f : X \to \mathbb{R}$. Figure inspired by [6].

CHAPTER 3

PREDICTIVE MAPPER

A primary area of research for my dissertation is work on the mapper algorithm, both in extending available software and the definitions and development of predictive mapper. Predictive mapper extends the mapper algorithm to create geometrically-motivated features for use in machine learning models. For a review of the original mapper algorithm, refer to Section 2.3.

When creating the predictive mapper algorithm, there are multiple items that must be developed to create useful features for machine learning. The mapper graph needs to be structured to represent different classes to be predicted, and must be optimal in a sense to be defined later. There needs to be a way to create real-valued vectors from the mapper graph, and finally a mapping must exist from the original dataset to the mapper graph. Using predictive mapper as an extension to the original mapper algorithm, each of these areas is addressed allowing for prediction using information from mapper output.

Before I discuss the algorithm, I review some spectral theory and motivation. Next I discuss each of the three components of the algorithm, and end with two examples on datasets and a discussion.

3.1 Spectral theory and motivation

To provide a theoretical justification for predictive mapper, I will first review a proposition and fact from spectral graph theory, and then discuss prior use of a related algorithm, Lapalacian eigenmaps. This background will be important when I discuss the mapper graph optimization and mapper embedding in Sections 3.2 and 3.3 respectively.

For this, I follow [19] until noted otherwise and start with notation required for this section. Let $G = (V, E)$, a similarity graph with vertex set $V = \{v_1, ..., v_n\}$ and edge set E with weights $w_{ij} \geq 0$ representing the similarity between the vertices connected by the edge. The weighted adjacency matrix of the graph is $W = (w_{ij})_{i,j=1,...,n}$. Assume G is

undirected so that $w_{ij} = w_{ji}$. The degree of a vertex $v_i \in V$ is

$$d_i = \sum_{j=1}^{n} w_{ij}. \tag{3.1}$$

The degree matrix D is the diagonal matrix with degrees d_i on the diagonal. Given a subset of vertices $A \subset V$, the complement $V \setminus A = \bar{A}$. Also one can measure the size of a A in two ways:

$$|A| \coloneqq \text{the number of vertices in } A \tag{3.2}$$

$$\text{vol}(A) \coloneqq \sum_{i \in A} d_i. \tag{3.3}$$

3.1.1 Spectral Theory

Moving into the spectral theory, disconnected and weakly connected components of a graph G can be separated by the eigenvectors of the graph Laplacian. The Laplacian matrix is defined as $L = D - W$, where D is the degree matrix, and W is the weighted adjacency matrix.

Proposition 4 (Number of connected components and the spectrum of L) *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of the Laplacian L equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$ of those components.*

From the proof of Proposition 4, the structure of the Laplacian matrix, L, has block diagonal form. Assume there are k connected components in G, then the Laplacian matrix L is:

$$L = \begin{pmatrix} L_1 & & & \\ & L_2 & & \\ & & \ddots & \\ & & & L_k \end{pmatrix}. \tag{3.4}$$

Each of the blocks of L_i is a graph Laplacian corresponding to the subgraph of the i-th connected component. The Laplacian L as a whole provides the indicator functions for connected components in a graph, using the first eigenvector in each $L_i \in L$.

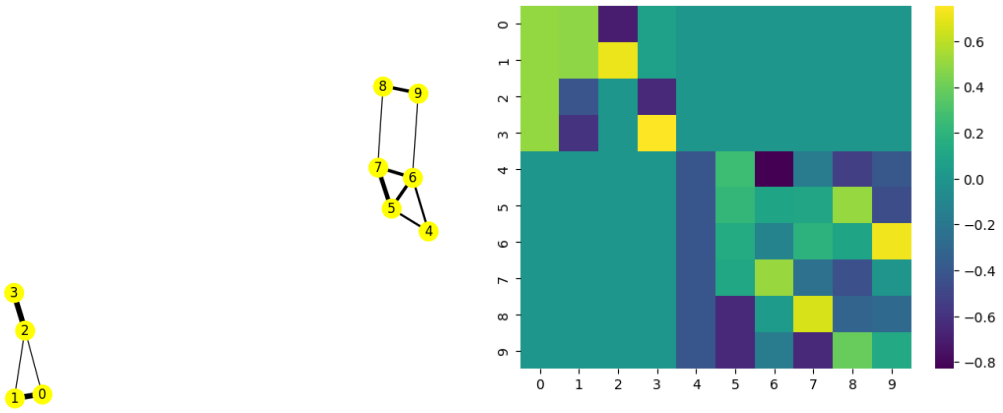
Beyond the Laplacian matrix $L = D - W$, there are other variants of this matrix called normalized graph Laplacians with the former referred to as the unnormalized graph Laplacian. For our purposes, the random walk Laplacian is

$$L_{rw} = D^{-1}L. \tag{3.5}$$

For completeness, there is another normalized Laplacian, called L_{sym} with details available in [19]. A property of L_{rw} I use is λ is an eigenvalue of L_{rw} with eigenvector u if and only if λ and u solve the generalized eigenvalue problem $Lu = \lambda Du$. It is notable the authors of [19] provide a few arguments for using the random walk Laplacian, and it is the Laplacian used throughout this paper. From the graph partitioning point of view (mincut problem) using L_{rw} maximizes within cluster similarity in contrast to the other Laplacians. Additionally, the normalized Laplacians do not have issues with convergence that are seen in the unnormalized Laplacians. There is also a proposition analogous to Proposition 4.

Proposition 5 (Number of connected components and the spectrum of L_{rw}) *Let G be an undirected graph with non-negative weights. Then the multiplicity k of the eigenvalue 0 of L_{rw} equals the number of connected components $A_1, ..., A_k$ in the graph. The eigenspace of eigenvalue 0 is spanned by the indicator vectors $\mathbb{1}_{A_1}, ..., \mathbb{1}_{A_k}$ of those components.*

To demonstrate this, in Figure 3.1 note a graph with 2 connected components, and the corresponding eigenvectors of the graph Laplacian. This shows the correspondence to the block structure of the eigenvectors of the Graph Laplacian and how these vectors can be used an indicator functions on the graph.



(a) Graph with 2 components.

(b) Eigenvectors of L corresponding to Figure 3.1a.

Figure 3.1 The eigenvalues of the graph Laplacian provide indicator functions for the 3 different components of the graph.

3.1.2 Laplacian eigenmaps

We see related work in [20]. The authors use a Laplacian eigenmap embedding as a lower dimensional representation of the original dataset. A graph G is constructed from the original dataset using either ϵ-neighborhoods or n nearest neighbors. Given points $x_1, ..., x_k \in \mathbb{R}^l$, an ϵ-neighborhood connects nodes i, j when $||x_i - x_j||^2 < \epsilon, \epsilon \in \mathbb{R}$. Alternatively, using n nearest neighbors, nodes i, j are connected by an edge if i is among the $n \in \mathbb{N}$ nearest neighbors of j (or j is among n nearest neighbors of i).

An edge weight is selected, for example from either the heat kernel, $W_{ij} = e^{\frac{||x_i - x_j||^2}{t}}$ or setting $W_{ij} = 1$ if vertices i and j are connected. Then for each connected component of G the eigenvalues and eigenvectors for $L\mathbf{y} = \lambda D\mathbf{y}$ are computed where D is the diagonal weight matrix, $D_{ii} = \sum_j W_{ji}$ and $L = D - W$ is the Laplacian matrix. The data is then embedded into a lower dimensional space given by $(\mathbf{y}_1(i), ..., \mathbf{y}_m(i))$ where each $\mathbf{y}_j$ is an

eigenvector, ordered by eigenvalue, with $\mathbf{y}_1$ having the smallest eigenvalue.

3.1.3 Mincut Problem

Additionally, the eigenvectors of each L_i provide a solution to the mincut problem. The mincut problem is a way to partition a graph into separate components by removing edge(s) with the minimum weight. Given a graph with adjacency matrix $W(A, B) \coloneqq \sum_{i \in A, j \in B} w_{ij}$ and $\bar{A}$ the complement of A. For subsets k, the mincut problem chooses a partition $A_1, .., A_k$ to minimize

$$\text{cut}(A_1, ..A_k) \coloneqq \frac{1}{2} \sum_{i=1}^{k} W(A_i, \bar{A}_i). \tag{3.6}$$

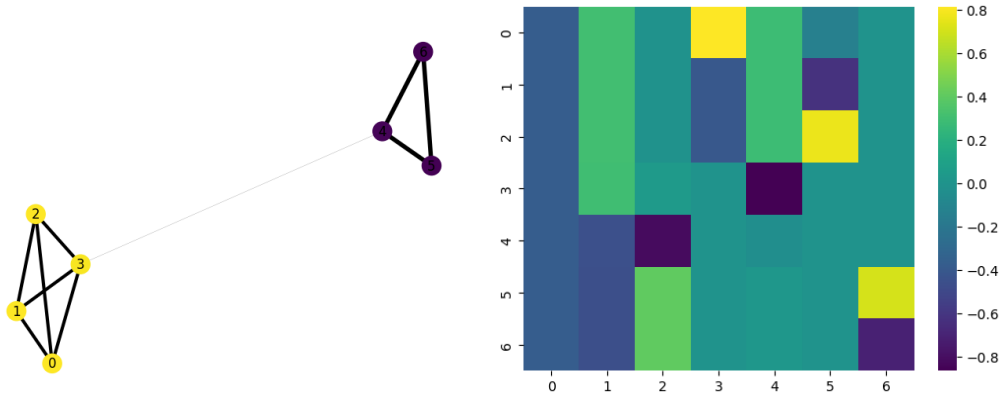
This can also be modified for other problems, with the Ncut problem having a direct relationship with the random walk Laplacian. The Ncut problem minimizes

$$\text{Ncut}(A_1, ..A_k) \coloneqq \frac{1}{2} \sum_{i=1}^{k} \frac{W(A_i, \bar{A}_i)}{\text{vol}(A_i)} = \sum_{i=1}^{k} \frac{\text{cut}(A_i, \bar{A}_i)}{\text{vol}(A_i)}. \tag{3.7}$$

When $k = 2$, the second eigenvector of L_{rw} provides a solution to a relaxed version of the Ncut problem. When $k > 2$, the first k eigenvectors provide a solution. In Figure 3.2 note the second eigenvector separates the graph into two distinct regions, where each region is strongly connected to other notes in the region and only weakly connected to the other region. This also translates to Figure 3.3, where the second eigenvector for the block matrix of each connected component differentiates areas of that component of the graph with the minimal edge weights.

3.2 Optimized mapper graph

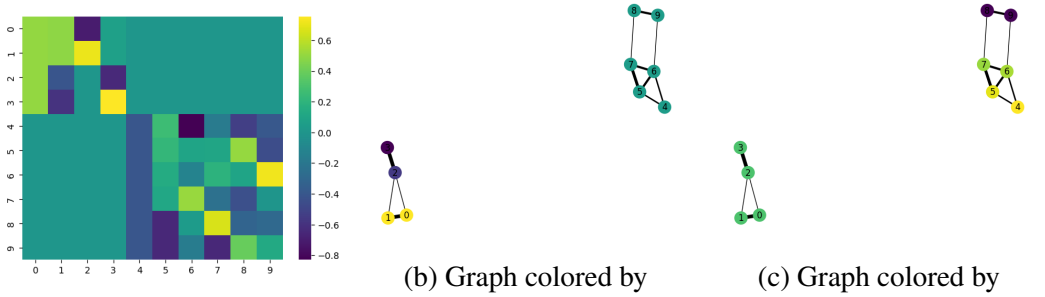
The overall goal of optimizing the mapper object as defined earlier is to have weak connections between areas of the graph that represent different classes, and strong connections between areas of heavy overlap. Very recent related work to optimize a mapper graph is



(a) Connected graph colored by the second eigenvector of the graph Laplacian.

(b) Eigenvalues of L corresponding to Figure 3.2a.

Figure 3.2 The eigenvalues of the graph Laplacian demonstrate the solution to the mincut problem as the second eigenvector separates the graph by minimal edge weight.



(a) Eigenvalues of L for Figure 3.3b.

(b) Graph colored by eigenvector 1 of L in Figure 3.3a.

(c) Graph colored by eigenvector 5 of L in Figure 3.3a.

Figure 3.3 The eigenvalues of the graph Laplacian demonstrate the solution to the mincut problem as the eigenvectors separate based on minimum edge weights.

available in [21], where the authors use an algorithm named G-mapper to determine an optimal cover of a mapper graph. This variant is a bit different and is as follows:

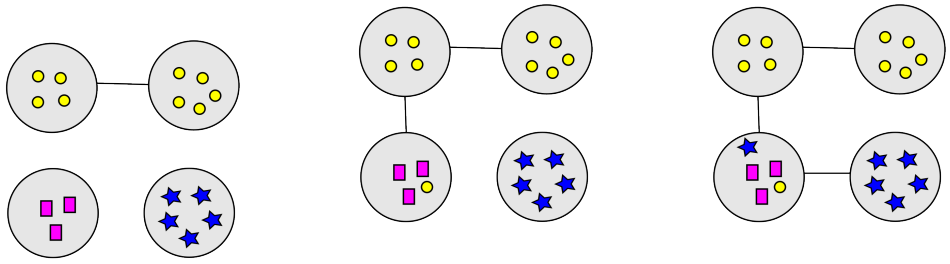
Definition 3.2.1 *Let $X(v_i)$ be the points associated with node $v_i \in V(M)$, with $V(M)$ the set of nodes in mapper graph M. Let $|X(v_i)_y|$ be the number of data points in v_i with label y. Then $f(v_i, y) = |X(v_i)_y|/|X(v_i)|$. Let N_y be the number of nodes in M which contain points with label y. Then*

$$H(M,Y) = \sum_{v \in V(M)} \sum_{y \in Y} \frac{f(v,y)}{N_y} \tag{3.8}$$

provides a measure on a mapper graph and the mapper graph is considered ***optimal*** *when* $H(M, Y) = |Y|$.

In the prior definition, if each node $v_i \in V(M)$ only contains one label, then $H(M, Y)$ is optimal. $H(M, Y)$ can be thought of as an overall quality measure on M, representing the percentage of outcomes in each node.

Consider Figure 3.4 as an example. Each color and shape corresponds to a different outcome variable, $y \in Y$. In both graphs, $|Y| = 3$, so to check if each graph is optimal, compute $H(M, Y)$. For Figure 3.4a compute $\frac{1}{2} + \frac{1}{2} + \frac{1}{1} + \frac{1}{1} = 3$, so this is an optimal graph. For Figure 3.4b compute $\frac{1}{3} + \frac{1}{3} + \frac{.25}{3} + \frac{.75}{1} + \frac{1}{1} = 2.5$, so this graph is not optimal. Lastly from Figure 3.4c $H(M, Y) = \frac{1}{3} + \frac{1}{3} + \frac{.2}{3} + \frac{.6}{1} + \frac{.2}{2} + \frac{1}{2} = 1.93$. Visually it is possible to distinguish that Figure 3.4c is less optimal than Figure 3.4b and that information is reflected in the metric $H(M, Y)$.



(a) Optimal mapper graph.

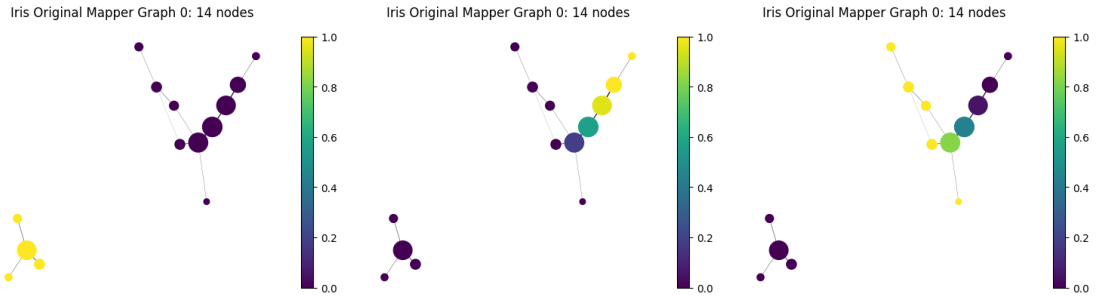
(b) Suboptimal mapper graph.

(c) Suboptimal mapper graph.

Figure 3.4 Example of (a) an optimal mapper graph where $H(M, Y) = |Y|$ and (b-c) two suboptimal mapper graphs. For (a): $H(M, Y) = \frac{1}{2} + \frac{1}{2} + \frac{1}{1} + \frac{1}{1} = 3$. For (b): $H(M, Y) = \frac{1}{3} + \frac{1}{3} + \frac{.25}{3} + \frac{.75}{1} + \frac{1}{1} = 2.5$. For (c):$H(M, Y) = \frac{1}{3} + \frac{1}{3} + \frac{.2}{3} + \frac{.6}{1} + \frac{.2}{2} + \frac{1}{2} = 1.93$.

To create an optimal mapper graph in practice, construction relies on various parameter choices during the initial mapper construction, and $H(M, Y)$ was computed on the mapper graph before optimization and after optimization as a comparison. This could be used as a metric during initial mapper graph creation, however I did not use it at the part of

the pipeline. Figure 3.5 shows the mapper algorithm applied to the Iris dataset [22] and highlighted by the proportion of each species of iris in the dataset. In Figure 3.5a there is complete separation of the Setosa class from the others, but the separation between Versicolor and Virginia is not as as obvious.



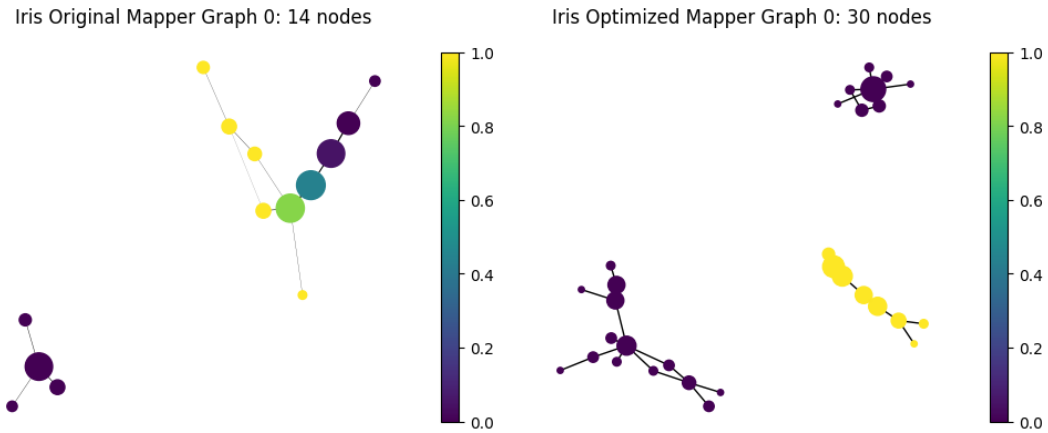
(a) Setosa percentage. (b) Versicolor percentage. (c) Virginia percentage.

Figure 3.5 Example mapper graph using the iris dataset with percentage of classifications in each node highlighted for Setosa, Versicolor, and Virginica. $H(M, Y) = 2.53$.

To create an optimal mapper graph, and for utility in a predictive pipeline, I first build a mapper graph, M, using $\bar{X} \subset X \in \mathbb{R}^{pxm}$, with X a dataset with p observations and m features. I also split X into a training and test set, denoted $\bar{X}$ and $\hat{X}$ respectively.

Then using mapper graph M, the set of nodes, $V(M)$, and the set of datasets associated with each node, $\bar{X}(v_i)$ for $v_i \in V(M)$, a clustering algorithm is applied to the subset of the data associated with a node. A new mapper object, $\bar{M}$ is then constructed using the new clusters to construct a larger node set, $V(\bar{M})$. The adjacency matrix, $A(\bar{M})$, is constructed by forming an edge between two nodes when points are in the intersection, that is for nodes $\bar{v}_i, \bar{v}_j \in V(\bar{M})$ with $i \neq j$, $A_{i,j}(\bar{M}) = |\bar{X}(\bar{v}_i) \cap \bar{X}(\bar{v}_j)|$. This provides a weighted adjacency matrix with weights equal to the size of the intersection of points.

What this means is after building a mapper graph through the normal pipeline, better separation can be achieved via localized clustering to target larger nodes with more variation in data distributions, and specifically variation in a desired outcome variable. Using the iris dataset as an example, this is illustrated in Figure 3.6.

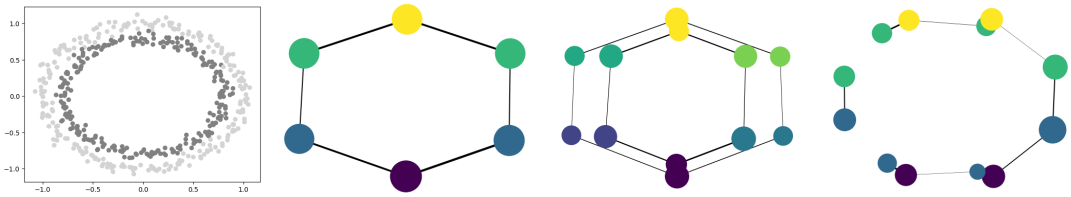


(a) Percent of versicolor nodes before additional clustering, $H(M,Y) = 2.53$.

(b) Percentage of versicolor nodes after additional clustering, $H(M,Y) = 3$.

Figure 3.6 Before and after of applying localized clustering to a mapper graph to achieve better separation between classes.

While this type of result may be achievable through the use of various mapper parameters, this additional way to cluster allows for more targeted post-processing of a mapper graph when the usual pipeline is not yielding a desireable result. An example of how the how the usual mapper process may fail is in Figure 3.7.



(a) (b) (c) (d)

Figure 3.7 This example shows (a) Point cloud sampled from 2 annuli (b) mapper graph using original pipeline (c) Mapper graph after optimizing (d) mapper graph with additional clusters in the original pipeline to create more separation. Note the two annuli were not recovered in the original mapper pipeline.

It should be noted that while using different filter functions may recover a desirable mapper graph, many trials of mapper graph creation with the Iris did not result in separation between the Versicolor and Virginica classes. The outcome variable was also used in the original mapper graph creation in an attempt to create additional separation without much

success.

3.3 Mapper embedding

Using both spectral theory and Laplacian eigenmaps as motivation for predictive mapper, I construct the mapper embedding portion of the algorithm. Through optimization of the mapper graph, the goal is to have nodes in the mapper graph all have the same classes, and be disconnected or weakly connected to nodes of different classes. Then the eigenvectors of the graph Laplacian of the optimized mapper graph represent the different components, and hence different classes.

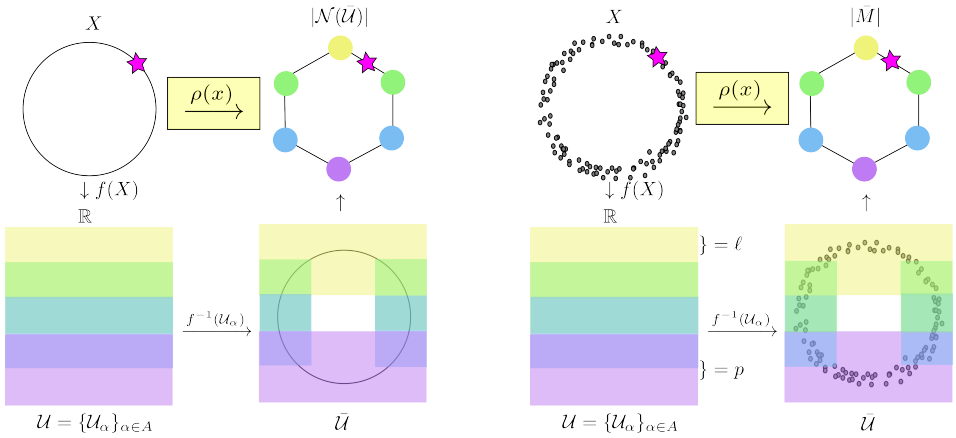
Using the optimized mapper graph as $\bar{M}$, the general eigenvalue problem $L\mathbf{y} = \lambda D\mathbf{y}$ is solved for $\mathbf{y}$, resulting in an embedding of $\bar{M} = [\mathbf{y}_o, ..., \mathbf{y}_n]$ This embedding represents the classes through connected components and the separation of weakly connected components.

3.4 Mapping from X to $|\bar{M}|$

The last item needed is a mapping from the dataset to the mapper graph so that these eigenvectors can be used as an embedding for both $\bar{X}$ and $\hat{X}$. The goal of this construction is shown in Figure 3.8.

To understand the following construction, I quickly review [1] for the topological motivation, and then construct the statistical version. The **nerve** of an open covering $\mathcal{U}$ of a space X, denoted $N(\mathcal{U})$, as an abstract simplicial complex with vertices $\in \mathcal{U}$ and its simplices are the finite subcollections $\{U_1, ..., U_n\}$ of $\mathcal{U}$ such that $U_1 \cap U_2 \cap ... \cap U_n \neq \emptyset$.

Topological Motivation Given a finite covering $\mathcal{U} = \{U_\alpha\}_{\alpha \in A}$ of a space X, a continuous map $f : X \to \mathbb{R}$, and the refinement of the pullback cover $f^{-1}(U_\alpha)$, which is path connected components as a covering of X, referred to as $\bar{\mathcal{U}}$. Mapper is then defined as the nerve of this cover, $\mathcal{N}(\bar{\mathcal{U}})$. Also recall a **simplicial complex** from Section 2.1. The nerve, $N(\bar{\mathcal{U}})$ can be embedded in Euclidean space and this is the **geometric realization** of this nerve, denoted $|N(\bar{\mathcal{U}})|$. I will construct a map $\rho : X \to |N(\bar{\mathcal{U}})|$.



(a) The topological version of mapper with an additional mapping $\rho(x)$.

(b) The statistical version of mapper with an additional mapping $\rho(x)$.

Figure 3.8 Topological vs. Statistical Version of Mapper with an additional mapping required for predictive mapper.

Definition 3.4.1 (Partition of unity) *A **partition of unity** subordinate to the finite open covering $\mathcal{U}$ is a family of real valued functions $\{\varphi_\alpha : X \to \mathbb{R}\}_{\alpha\in A}$ with the following properties:*

- $0 \leq \varphi_\alpha(x) \leq 1$ *for all* $\alpha \in A$ *and* $x \in X$.
- $\sum_{\alpha\in A} \varphi_\alpha(x) = 1$ *for all* $x \in X$.
- *The closure of the set* $\{x \in X \mid \varphi_\alpha(x) > 0\}$ *is contained in the open set* U_α.

An example partition of unity is shown in Figure 3.9b. A family of probability density functions could be used to construct to be a partition of unity. Consider the Gaussian probability distribution function $\phi_{\mu,\sigma}(x) = \frac{1}{\beta}\exp(-\frac{1}{2}(\frac{x-\mu}{\sigma})^2)$, with mean μ, standard deviation σ, and β a normalization constant. Next define a family of functions $\Phi = \{\varphi_0(x), \varphi_1(x), \varphi_2(x), \varphi_3(x)\}$ with

$$\varphi_0(x) = \exp\left(\frac{1}{2}\left(\frac{-4-x}{1}\right)^2\right) \tag{3.9}$$

$$\varphi_1(x) = \exp\left(\frac{1}{2}\left(\frac{2-x}{2}\right)^2\right) \tag{3.10}$$

$$\varphi_2(x) = \exp\left(\frac{1}{2}\left(\frac{8-x}{2}\right)^2\right) \tag{3.11}$$

$$\varphi_3(x) = \exp\left(\frac{1}{2}\left(\frac{-10-x}{2}\right)^2\right) \tag{3.12}$$

$$\tag{3.13}$$

Let $\beta = \varphi_0(x) + \varphi_1(x) + \varphi_2(x) + \varphi_3(x)$ and set $\mathcal{U} = (-18, 16)$ with $U_0 = (-8, 0), U_1 = (-6, 10)$, $U_2 = (0, 16)$ and $U_3 = (-18, 2)$. Then the family of functions $\bar{\Phi} =$

$$\bar{\varphi_0} = \frac{\varphi_0(x)}{\beta} \tag{3.14}$$

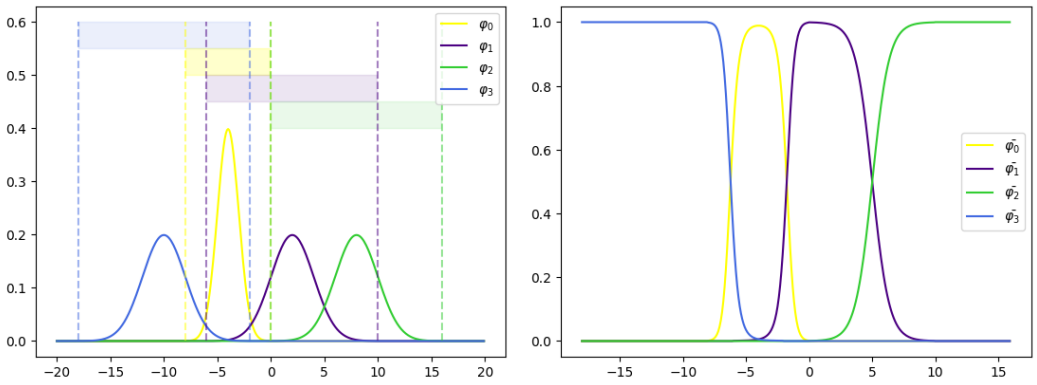
$$\bar{\varphi_1} = \frac{\varphi_1(x)}{\beta} \tag{3.15}$$

$$\bar{\varphi_2} = \frac{\varphi_2(x)}{\beta} \tag{3.16}$$

$$\bar{\varphi_3} = \frac{\varphi_3(x)}{\beta} \tag{3.17}$$

is a partition of unity on $\mathcal{U} = (-18, 16)$. It can be checked that this construction satisfies the definition of a partition of unity.

Using $\{v_0, v_1, ..., v_k\} = V$ to denote the vertices of $N(\bar{\mathcal{U}})$, there exists the **barycentric coordinization**, which is a bijection of the points v in the geometric simplex to a set of ordered k-tuples $(r_0, r_1, ..., r_k) \in \mathbb{R}^{k+1}$ such that $0 \leq r_i \leq 1$ and $\sum_{i=0}^{k} r_i = 1$. Each r_i is a **barycentric coordinate**. For a given partition of unity $\bar{\Phi}$, define $\rho(x) \in |\mathcal{N}(\mathcal{U})|$ as the point in the simplex spanned by the vertices V, with barycentric coordinates $(\bar{\varphi_0}(x), \bar{\varphi_1}(x), ..., \bar{\varphi_k}(x))$. This $\rho(x)$ is the mapping required from $X \to |\mathcal{N}(\bar{\mathcal{U}})|$.



(a) A family of functions Φ. (b) A family of functions $\bar{\Phi}$.

Figure 3.9 A family of functions Φ before and after normalization to create a partition of unity on the range [-17.99,15.9].

Statistical Construction Using this, I construct a partition of unity to build the map $P : X \to |\bar{M}|$, where $|\bar{M}|$ is the geometric realization of $\bar{M}$. Given $\bar{X}(v_i)$ the subset of data $\bar{X}$ in each node $v_i \in V(\bar{M})$ with $0 \leq i \leq k$, summary statistics are then computed on each set $\bar{X}(\bar{v}_i)$. Define Let $\Phi = \{\varphi_0, \varphi_1, ..., \varphi_k\}$ be a family of probability distributions with

$$\varphi_i(x) = \exp(-\frac{1}{2}(x - \mu_{\bar{v}_i})^T \Sigma_{\bar{v}_i}^{-1}(x - \mu_{\bar{v}_i})) \tag{3.18}$$

$$\beta = \sum_i \varphi_i(x) \tag{3.19}$$

$$\bar{\varphi}_i(x) = \frac{\varphi_i(x)}{\beta} \tag{3.20}$$

with $\mu_{\bar{v}_i}$ as the mean of $\bar{X}(\bar{v}_i)$ and $\Sigma_{\bar{v}_i}$ the covariance of $\bar{X}(\bar{v}_i)$. Then $\bar{\Phi} = \{\bar{\varphi_0}, \bar{\varphi_1}, ..., \bar{\phi_k}\}$ is a partition of unity that can provide a barycentric coordinization of points to the mapper graph, i.e. a point $\hat{x} \in \hat{X}$ has a barycentric coordinatization in $|\bar{M}| = (\bar{\varphi_0}(\hat{x}), \bar{\varphi_1}(\hat{x}), ..., \bar{\varphi_k}(\hat{x}))$.

Finally, using this barycentric coordinatization together with the eigenvectors $\mathbf{y}$, features can be created that are a linear combination of the values for each eigenvector. Taking a single eigenvector, $\mathbf{y}_j \in \mathbb{R}^k$, the eigenvector feature for $\hat{x}$ is

$$\sum_{i=0}^{k} \mathbf{y}_{ji} \bar{\varphi}_i(\hat{x}). \tag{3.21}$$

This is the eigenvector value associated with each node, multiplied by the probability of inclusion for $\hat{x}$ in each node. This has the effect that for nodes with small or 0 probability of inclusion for $\hat{x}$, the value of the eigenvector at that node provides minimal contribution to the feature, whereas higher probability nodes provide higher contributions to the feature.

Note that $\bar{X}$ is a random sample of X, and X is a sample from some underlying distribution. One can approximate the probability distribution function with any of our random samples: $X, \bar{X}, \hat{X}$. Error bounds will differ, but this construction then holds for any random samples of X.

3.5 Predictive Mapper Pipeline

The optimized mapper graph, mapper embedding, and mapping from X to $\bar{M}$ provide the predictive mapper pipeline which I summarize here. The following inputs are required to create a mapper graph and consequently the predictive mapper pipeline: **training data**: $\bar{X}$, **testing data**: $\hat{X}$, **filter function**: f, and **distance matrix or metric**: d. Additionally the following are parameters required: **clustering algorithm**, **partition length:** ℓ, **percent overlap in partition:** p, and a **clustering algorithm for mapper optimization**. The predictive mapper construction is then as follows:

- Construct M, a mapper graph using the $\hat{X}$, f, d, **clustering algorithm**, p, l
- Optimize M with localized clustering using the **clustering algorithm for mapper optimization** on nodes to so that if $\bar{M}$ is an optimized version of M, then $H(M, Y) < H(\bar{M}, Y)$
- Construct the weighted adjacency matrix of $\bar{M}$ such that $\bar{v}_i, \bar{v}_j$ with $i \neq j$, $A_{i,j}(\bar{M}) = |\bar{X}(\bar{v}_i) \cap \bar{X}(\bar{v}_j)|$
- Compute the eigenvectors $\mathbf{y}$ of the generalized eigenvalue problem, $L\mathbf{y} = \lambda D\mathbf{y}$

- Taking $\bar{X}(v_i)$ the subset of data $\bar{X}$ in each node $v_i \in V$, compute summary statistics and the partition of unity $\bar{\Phi}$ functions
- For each $x \in \bar{X}$ and $x \in \hat{X}$, compute the probability of inclusion in each node using $\bar{\Phi}$
- Create features as a linear combination of eigenvectors and the partition of unity, i.e. for each j, $\sum_{i=0}^{k} \mathbf{y}_{j_i} \bar{\varphi}_i(\hat{x})$

This pipeline provides an approach to creating features from a mapper graph that can be used in machine learning models. An example of the full pipeline is shown in Figure 3.10.

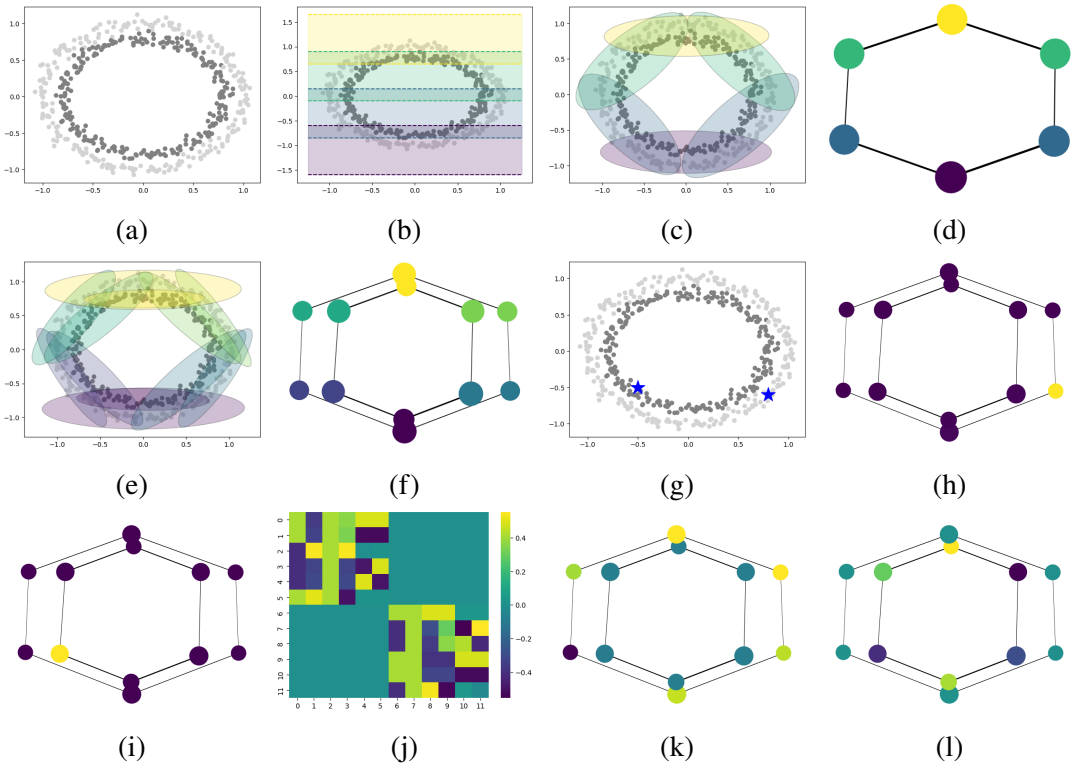


Figure 3.10 A simulated example of the predictive mapper pipeline. (a) Point cloud sampled from two annuli. (b) Construction of pullback cover. (c) Refined pullback cover. (d) mapper graph. (e) Cover from mapper optimization step. (f) Optimized mapper graph. (g) Two test points in X. (h) Probability of inclusion in each node for point on the right. (i) Probability of inclusion in each node for the point on the left. (j) Eigenvectors of $\bar{M}$. (k) Eigenvector features of the point on the right represented on the mapper graph. (l) Eigenvector features of the point on the left represented on the mapper graph.

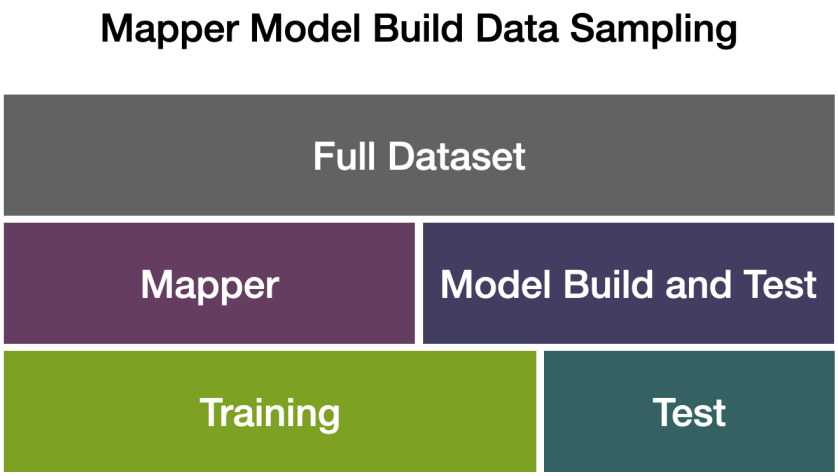


Figure 3.11 Mapper Experiment Design.

3.6 Methods and Results

Using two datasets, the famous iris [23] dataset and a student performance [24] dataset, features created from the predictive mapper algorithm were assessed for accuracy in different machine learning models. The iris dataset was used as a familiar dataset during algorithm development and to identify potential areas to test, while the student performance dataset was used to explore potential use cases for predictive mapper features.

To create a comparison for performance of the predictive mapper features, the same machine learning models were evaluated a feature set created from predictive mapper and the feature set from the original data. To create a true comparison for performance, 50% of the data was used to build a mapper model, and 50% was used as a testing set to validate model performance. The dataset structure for building the mapper model and building and testing the model is shown in Figure 3.11.

For both datasets, the clustering algorithm used during the initial mapper fit was HDBScan [14] and the chosen clustering algorithm to optimize mapper was Bayesian gaussian mixture modeling [25]. HDBScan was chosen as it meets the requirements of [1] and Bayesian Gaussian mixture modeling was chosen as computation of means and standard deviations is part of the algorithm, the distributions are assumed normal, and the number

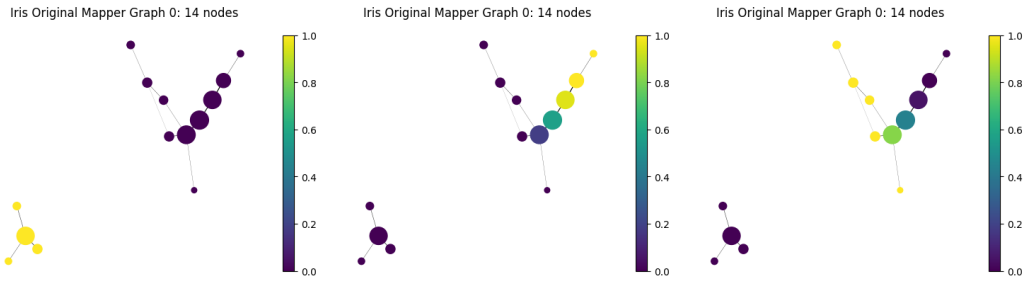
of clusters can be inferred from the data.

Both datasets also report performance in terms of original data, eigenvector features, and augmented features. The augmented data is using both the original data and eigenvector features to assess model performance.

3.6.1 Iris

The iris [23] dataset is one of the earliest known datasets using in machine learning models, and can be traced back to scientist R.A. Fisher in 1936. The dataset has 4 features that measure properties of the iris plant, with a classification of plant type. There are three classes with 50 instances each, making the dataset perfectly balanced for testing machine learning tasks. One of the classes is also easily separable from the other two, making iris an excellent candidate for evaluation and constructing the predictive mapper algorithm.

The predictive mapper algorithm was run on the iris dataset using three different mapper graphs, all with the same parameters in mapper graph creation using a filter of principle components analysis. The original mapper graph is in Figure 3.12 and colored by percentage of outcome variable, and complete separation between Setosa and the other two classes is observed in the original graph.



(a) Iris mapper graph color by percentage of node in Setosa class.

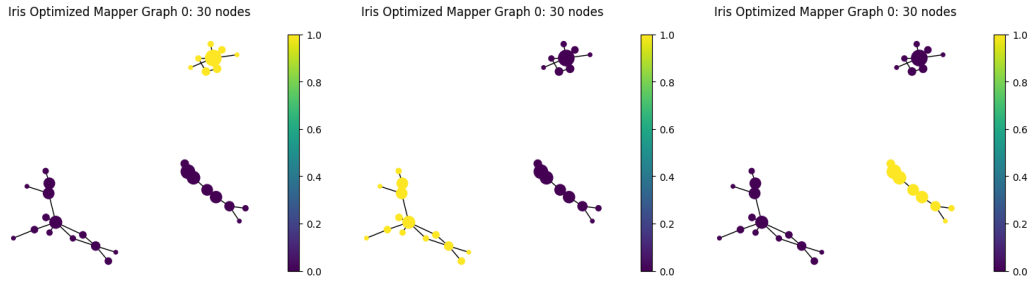
(b) Iris mapper graph color by percentage of node in Versicolor class.

(c) Iris mapper graph color by percentage of node in Virginia class.

Figure 3.12 Original iris mapper graphs for mapper graph 0.

To optimize the mapper graph, Bayesian Gaussian mixture modeling was applied to the original mapper graph, and the mapper graphs after optimization are shown in Figure 3.13,

and each class is associated with a different connected component.



(a) Iris mapper graph color by percentage of node in Setosa class.

(b) Iris mapper graph color by percentage of node in Versicolor class.

(c) Iris mapper graph color by percentage of node in Virginia class.

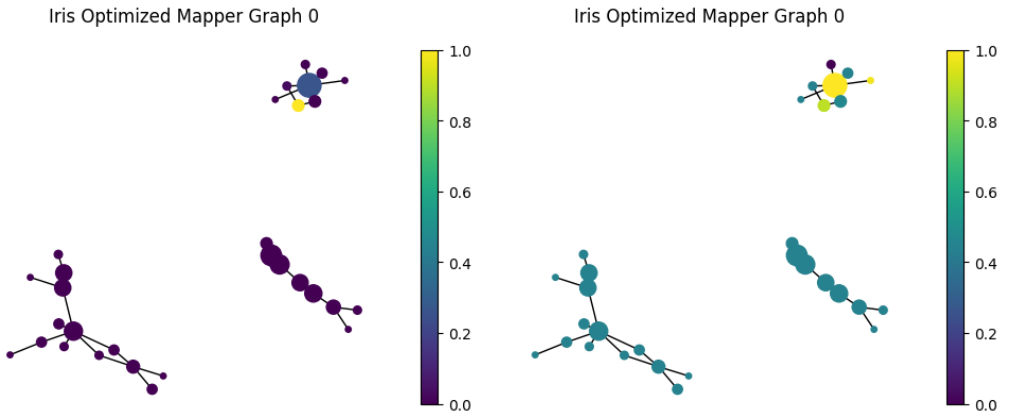
Figure 3.13 Optimized iris mapper graphs for mapper graph 0.

The probability of inclusion in each node for all data points was computed, and then the features were created using the partition of unity and eigenvector basis. Example graphs of the probability of inclusion and eigenvector features are shown for a selected point in Figure 3.14. For both images, the graph on the left highlights three nodes with a non-zero probability of inclusion for a sample data point. This is then reflected in the eigenvector features, with variation in the same nodes and the nodes in the same connected component. Components with a zero probability of includes for the data point do not show variation in the eigenvector features.

Three different optimized mapper graphs were assessed with the results are included in Table 3.2. $H(M,Y)$ was measure before and after for each and reported in Table 3.1. In each of the three trials a more optimal mapper graph was created through the predictive mapper pipeline.

Trial	$\lvert V(M)\rvert$	$H(M,Y)$	$\lvert V(\bar{M})\rvert$	$H(\bar{M},Y)$
Mapper Graph 0	14	2.53	30	3.00
Mapper Graph 1	16	2.55	33	3.00
Mapper Graph 2	15	2.43	33	3.00

Table 3.1 Iris mapper graph comparison before and after optimization.



(a) Example point in iris dataset colored by probability of inclusion in each node.

(b) Example point in iris dataset colored by eigenvector features.

Figure 3.14 An example point from the iris dataset shown with the probability of node inclusion on the left and the associated eigenvector feature on the right.

With all three graphs, ridge classification using the eigenvector features outperformed classification using the original dataset. The random forest model outperformed the original data in 2 of the 3 trials, and trial 1 had 4 of the 8 models where the eigenvector features outperformed the original data.

While these results are not broad enough to be generally conclusive, there are a few observations to note. The eigenvector features seem to perform well for ridge regression, and may be useful in applications where that machine learning model is preferred. The mapper graph and subsequent optimized mapper graph impact results, with variation among the three different graphs. Finally, as the Iris dataset is a small dataset, sampling may have a larger impact on the outcome.

3.6.2 Student Performance

The dataset used to evaluate predictive mapper features is called Student Performance [24] and hosted on the UC Irvine Machine Learning Repository. The dataset has 30 features and 649 observations comprised of grades, demographic, social, and school factors collected by a combination of school information and questionnaires. The dataset was originally evaluated in [26] where the original authors predicted three different outcomes

Iris Mapper Object 0			
Model	Original Data	Eigenvector	Augmented
Random Forest (n=5)	92.5%	93.5%	**93.6%**
SVM (linear, c=1)	**95.6**	93.5	93.8
SVM (linear, c=10)	**95.8**	92.9	93.5
SVM (rbf)	**95.4**	93.3	93.3
KNN (n=5)	**94.7**	93.5	93.5
Ridge (alpha=1)	82.9%	93.1%	**93.3%**
Ridge (alpha=10)	80.8%	93.2%	**93.4%**
Logistic Regression	**95.9%**	93.1%	93.2%
Iris Mapper Object 1			
Random Forest (n=5)	94.3%	94.1%	**94.7%**
SVM (linear, c=1)	**96.6**	93.6	93.6
SVM (linear, c=10)	**95.1**	93.1	92.2
SVM (rbf)	**95.7**	92.4	92.8
KNN (n=5)	**95.2**	84.0%	92.2%
Ridge (alpha=1)	82.4%	94.0%	**94.1%**
Ridge (alpha=10)	83.6%	**93.9%**	93.9%
Logistic Regression	93.5%	**94.1%**	93.6%
Iris Mapper Object 2			
Model	Original Data	Eigenvector	Augmented
Random Forest (n=5)	**96.2%**	94.8%	95.1%
SVM (linear, c=1)	**97.3**	94.7	94.6
SVM (linear, c=10)	**97.3**	94.5%	94.7%
SVM (rbf)	**97.4%**	96.5%	96.5%
KNN (n=5)	**97.2%**	89.7%	95.5%
Ridge (alpha=1)	79.7%	**94.6%**	94.6%
Ridge (alpha=10)	74.9%	**94.8%**	94.8%
Logistic Regression	**97.1%**	94.4%	94.5%

Table 3.2 Accuracy from Iris mapper models.

using a variation of machine learning methods. The variables are in Table A.1 taken from [26] and important to understanding both the original classification methods used and those explored in this paper.

The original analysis predicted G3 using all variables, using all variables except G2, and using all variables except G1 and G2. The third problem, predicting the final grade (G3) without first (G1) and second (G2) period grades was much harder, and the focus of the predictive mapper analysis. Data was prepared for analysis by creating indicator

variables for each of the binary outcomes, creating a binary indicator for pass or fail for G3, and oversampling the failed class. In the original dataset of 649 observations, 100 of the observations were classified as fail for G3, or 15.4 % of the sample having failed. Data was split into a training and testing of 50% each, and then the failed class was oversampled in the training set to create a balanced dependent variable. The testing set was not modified.

The 50% of the data used as the training set was used to build a mapper object and was used to train various models. To create some variability, data from the held out testing set was included to train the model (50%) and not used to report on model accuracy, and the average of 100 runs is reported. To also create variation in the mapper graphs, there were 12 different mapper graphs generated and evaluated, all with different datasets. For direct comparisons, mapper graphs were created with the same parameters from different data. The parameters of each experiment is outlined in Table 3.3. Full results are included here for trials 0-5.

Trial	Filter	Partitions	Overlap	Original Nodes	Optimized Nodes
0	PCA	8	50%	20 (H = 1.29)	28 (H = 1.25)
1	PCA	8	50%	16 (H = 1.14)	26 (H = 1.09)
2	PCA	8	50%	20 (H = 1.28)	26 (H = 1.22)
3	Eccentricity (5)	8	25%	20 (H = 1.27)	28 (H = 1.26)
4	Eccentricity (5)	8	25%	21 (H = 1.32)	29 (H = 1.26)
5	Eccentricity (5)	8	25%	22 (H = 1.18)	28 (H = 1.14)
6	tsneX (5)	8	50%	24 (H = 1.34)	38 (H = 1.30)
7	tsneX (5)	8	50%	22(H = 1.24)	33 (H = 1.21)
8	tsneX (5)	8	50%	18 (H = 1.26)	25 (H = 1.21)
9	tsneX (5)	12	30%	20 (H = 1.07)	49 (H = 1.20)
10	tsneX (5)	12	30%	14 (H = 1.03)	37 (H = 1.10)
11	tsneX (5)	12	30%	16 (H = 1.02)	44 (H = 1.08)

Table 3.3 Parameters for student performance data mapper graph comparison.

The standard predictive mapper pipeline was followed for the student performance dataset, and how optimal $H(M, Y)$ the graphs were was measured. The same optimization process followed for the Iris dataset was used, and did not result in a more optimal mapper

graph for many of the graphs. To create a direct comparison to the Iris dataset, one can normalize by the number of classes. The Iris mapper graphs all had an optimal score of 1. The optimal score for the student performance dataset in any mapper graph, optimized or not, was not close to that range. The minimum normalized score was .5 and the maximum normalized score was .67. It is also doubtful there is a relevant difference in features created from the original or optimized mapper graphs for this dataset based on the normalized optimal scores.

Using the optimized mapper graphs, a total of 8 machine learning models were run, and the partition of unity matrix before the eigenvector change of basis was also used as a feature set. "Eigenvector +" denotes the original dataset augmented with eigenvector features, and "Partition +" denotes the original dataset augmented with the features from the partition of unity. While evaluating this dataset initially, the eigenvector features performed intermittently in terms of accuracy, but very consistently predicted all students passed. When this was observed, the partition of unity features were also evaluated to better understand if they could be useful in prediction.

Based on the lack of performance from the eigenvector features for this dataset, different parameters in how the partition of unity was computed were modified to see if there were ways to modify features for better performance. Given this, the student dataset results should absolutely be treated as exploratory in nature.

Trials 0-1 For the first three trials (trials 0, 1, and 2) the first principle component was used as a filter for creation of the mapper object. Results for these trials are shown in Table 3.4. In two of the three mapper graphs, some combination involving the eigenvector features or partition of unity features outperformed the original dataset features alone for all models.

In the results table, * denotes no recall for the failed class in the dataset ($G3 = 1$). I see this specifically happening with the eigenvector features. This suggests there may be

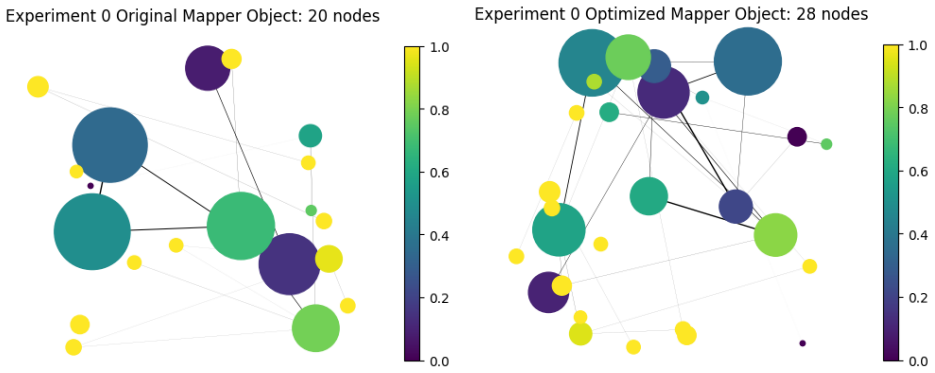
Student performance Mapper Graph 0					
Model	Original Data	Eigenvector	Partition	Eigenvector+	Partition+
Random Forest (n=5)	82.7%	78.3%	81.0%	83.1%	**83.5%**
Random Forest (n=50)	84.1%	79.0%	81.5%	84.4%	**84.5%**
Random Forest (n=100)	84.3%	80.0%	81.4%	84.7%	**84.5%**
SVM (rbf)	83.2%	84.6%*	**85.6%**	82.3%	85.1%
KNN (n=5)	82.7%	78.3%	81.0%	83.1%	**83.5%**
Ridge (alpha=1)	81.4%	84.6%*	**85.6%**	81.5%	82.5%
Ridge (alpha=10)	81.5%	84.6%*	**85.5%**	81.6%	82.6%
Logistic Regression	81.7%	84.6%*	**85.3%**	81.7%	81.1%
Student performance Mapper Graph 1					
Random Forest (n=5)	83.4%	83.5%	83.0%	**84.5%**	84.3%
Random Forest (n=50)	**85.7%**	83.6%	83.1%	85.2%	85.4%
Random Forest (n=100)	**85.8%**	83.7%	83.2%	85.0%	85.3%
SVM (rbf)	**88.0%**	84.6%*	84.6%*	85.5%	85.4%
KNN (n=5)	83.4%	83.6%	83.0%	**84.5%**	84.3%
Ridge (alpha=1)	81.2%	**85.0**	84.9%	84.8%	84.8%
Ridge (alpha=10)	81.4%	85.0%	84.9%	**85.2%**	85.2%
Logistic Regression	81.4%	**84.9%**	84.9%	84.2%	84.2%
Student performance Mapper Graph 2					
Random Forest (n=5)	81.7%	80.6%	80.9%	**83.3%**	83.3%
Random Forest (n=50)	84.1%	80.9%	81.2%	84.3%	**84.5%**
Random Forest (n=100)	84.3%	80.8%	81.2%	84.3%	**84.5%**
SVM (rbf)	83.5%	82.8%	83.1%	**85.0%**	85.0%
KNN (n=5)	81.7%	80.6%	80.9%	**83.3%**	83.3%
Ridge (alpha=1)	77.4%	84.3	84.3%	85.4%	**85.4%**
Ridge (alpha=10)	77.4%	84.3%	84.3%	85.4%	**85.4%**
Logistic Regression	77.9%	**84.3%**	84.3%	83.6%	83.6%

Table 3.4 Results from trials 0, 1, and 2 for the student performance dataset models using mapper.

information loss in the predictive mapper pipeline that is useful for this machine learning task.

Each of the mapper graphs had slightly different features, and the original mapper graph for a representative graph along with the optimized mapper graphs is shown in Figure 3.15, with the other two for this trial in the Appendix as Figure A.1 and Figure A.2.

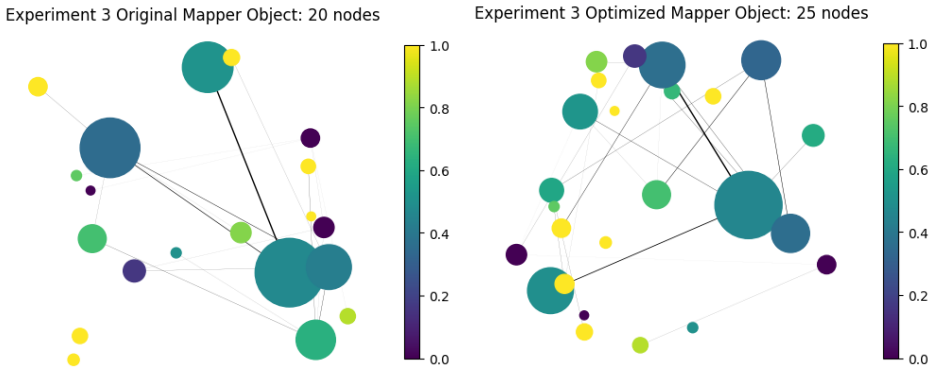
Trial 3 - 5 For the next trials, eccentricity was used as a filter for creation of the mapper



(a) Original mapper graph. (b) Optimized mapper graph.

Figure 3.15 Mapper graphs for Experiment 0 before and after optimization. For this example, the typical optimization work flow was not successful.

object. A representative optimized mapper graph is shown in Figure 3.16, with the other two in the Appendix as Figure A.3 and Figure A.4.



(a) Original mapper graph. (b) Optimized mapper graph.

Figure 3.16 Mapper graphs for Experiment 3 before and after optimization. For this example, the typical optimization work flow was not successful.

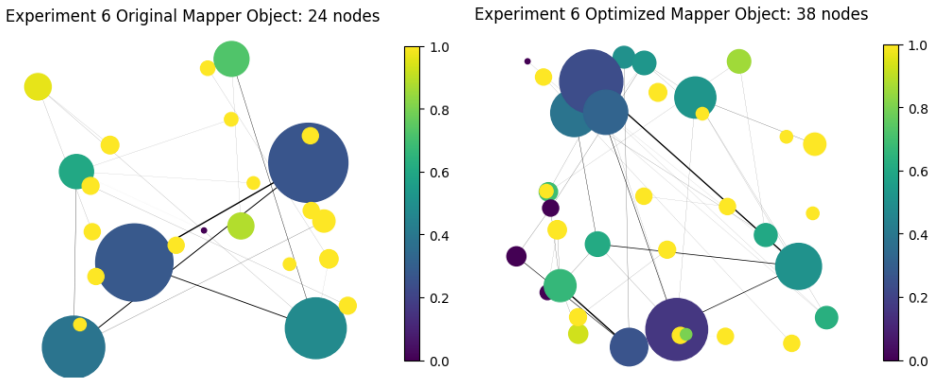
Results are shown in Table 3.5. Note the * designated denotes the feature set and model were not able to identify any failed observations in the data. For the third mapper graph random forest models, the partition+ features seem to perform better than the original dataset. This may be due to the original dataset overfitting the models, and the partition features underfitting the models. I also observed the eigenvector features for the third mapper graph is not able to recall any of the failed class, further supporting this graph may

be underfitting the data.

Student performance Mapper Graph 3					
Model	Original Data	Eigenvector	Partition	Eigenvector+	Partition+
Random Forest (n=5)	**82.7%**	84.5%*	71.6%	82.7%	82.7%
Random Forest (n=50)	83.8%	84.5%*	71.3%	83.8%	**84.2%**
Random Forest (n=100)	83.9%	84.5%*	71.3%	83.8%	**84.3%**
SVM (rbf)	82.7%	84.6%*	81.2%	81.8%	**84.1%**
KNN (n=5)	82.1%	84.6%*	71.6%	**82.7%**	**82.7%**
Ridge (alpha=1)	78.2%	84.6%*	81.1%	80.5%	**84.1%**
Ridge (alpha=10)	78.1%	84.6%*	81.1%	80.5%	**84.0%**
Logistic Regression	78.9%	84.6%*	80.6%	81.2%	**82.2%**
Student performance Mapper Graph 4					
Random Forest (n=5)	83.4%	82.7%	81.3%	83.4%	**84.1%**
Random Forest (n=50)	85.9%	83.2%	85.8%	85.2%	**86.0%**
Random Forest (n=100)	**86.0%**	83.4%	81.4%	86.0%	86.0%
SVM (rbf)	88.1%	82.7%	75.7%	**88.6%**	88.5%
KNN (n=5)	83.4%	83.6%	81.3%	83.4%	**84.1%**
Ridge (alpha=1)	81.3%	84.6%*	**83.4%**	83.2%	83.4%
Ridge (alpha=10)	81.4%	84.6%*	83.4%	83.2%	**84.3%**
Logistic Regression	81.4%	84.6%*	**83.5%**	82.0%	83.4%
Student performance Mapper Graph 5					
Random Forest (n=5)	82.7%	84.6%*	81.5%	83.2%	**83.3%**
Random Forest (n=50)	**85.6%**	84.6%*	81.5%	85.6%	85.2%
Random Forest (n=100)	85.9%	84.6%*	81.4%	85.8%	85.3%
SVM (rbf)	85.4%	84.6%*	84.6%*	**85.5%**	85.4%
KNN (n=5)	82.7%	84.6%*	81.5%	83.2%	**83.3%**
Ridge (alpha=1)	80.0%	84.6%*	70.2%	81.6%	**84.5%**
Ridge (alpha=10)	80.0%	84.6%*	84.6%*	81.2%	**84.2%**
Logistic Regression	78.6%	84.6%*	84.4%	81.4%	**84.2%**

Table 3.5 Results from trials 3, 4, and 5 for the student performance dataset models using mapper.

Trial 6 - 12 Additional models were run with this set of mapper graphs to allow for more extensive evaluation of performance. The mapper graphs were created using tSne as a filter, and the focus was on creating nodes in the mapper graph that are more heavily weighted to either passing or failing. Before and after optimization of an example mapper graph is shown in Figure 3.17.



(a) Original mapper graph. (b) Optimized mapper graph.

Figure 3.17 Mapper graphs for Experiment 6 before and after optimization. For this example, the typical optimization work flow was not successful.

Additional figures (Figure A.5 and Figure A.6 and results from these trials can be found in the Appendix. Figures from mapper graphs 9, 10, 11 are available as Figure A.7, Figure A.8 and Figure A.9 respectively. These results do not differ from the mapper graphs created using other filters, and the eigenvector features are unable to identify the failing class. Results for these are not formally reported as they do not change the interpretation or outcome.

3.7 Computation

Work for predictive mapper was completed in python. This was done through a combination of custom functions for prediction and an extension of Kepler mapper [27]. Mapper graphs and notebooks are available for models run for the Iris dataset. Future work could include the incorporation of these functions into the Kepler mapper software.

3.8 Discussion and Limitations

Results of predictive mapper for the Iris dataset show promise, and further research is needed into similar use cases where predictive mapper may have usefulness. Based on the student performance dataset, predictive mapper may not be useful with unbalanced datasets and machine learning models where recall is a difficult task. The student performance dataset may also require specialized filters that extract the relevant information from the

dataset. Even with significant modification of features in an attempt to extract better performance, predictive overall using eigenvector or partition of unity features was not useful.

It is important to note that creating an apples to apples comparison for model performance of predictive mapper features to the original dataset features has the potential for bias based on the way the mapper models are created and the need to have a similar amount of information available for each model. To minimize this issue multiple mapper models were built using the same parameters for each dataset from different random samples of data. Model parameters were not optimized during model fitting beyond running with a few different parameters, and accuracy for each model is reported using the same parameters for both the original dataset and the predictive mapper features. This could significantly impact the performance of both the original and predictive mapper models.

A main area where there is significant impact to the performance of the predictive mapper models is in the creation of the mapper graph used to create features and parameters for feature creation. For the development of each mapper graph used to assess model performance subjective decisions were made to optimize model performance, with the end of goal of developing useful features for machine learning. To create predictive features decisions on mapper graph parameters and optimization and a suitable way to map test observations onto the mapper object must be made and are subjective and heavily depend on the dataset. An average or stable mapper graph could be a potential avenue to explore to mitigate current issues with individual mapper graphs in predictive pipelines.

Related work related to the Laplacian eigenvectors was done in [28]. This work crafts an algorithm for eigenvector cascading, and shows that important dataset features persist over different scales. The idea would need to be adapted across samples of data and not necessarily scale, although that could be a relevant feature.

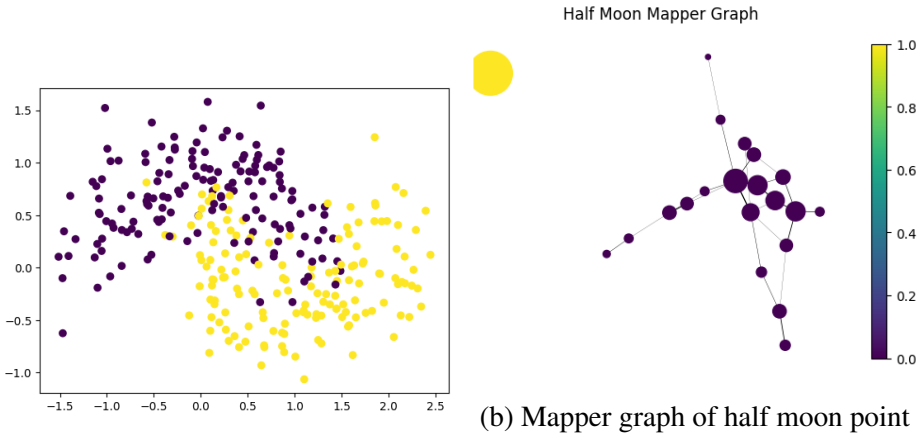
Outstanding work in the predictive mapper algorithm includes additional work on cre-

ating an optimal mapper graph through filters and additional localized clustering, and exploration of performance with different probability distribution functions and additional data sets. The ideal goal is to find specific areas where mapper features can provide useful predictive information that available in the original feature set, and high dimensional datasets may benefit from an additional feature reduction technique. Based on prior work with Laplacian eigenmaps, this may be in machine learning problems requiring a reduced feature set where predictive mapper would be a good solution. Further research into how optimal a mapper graph can be before overfitting becomes an issue for modeling is also needed. A formal assessment on how well a particular mapping from $X \rightarrow |\bar{M}|$ performs for datasets would be a final area of investigation to increase utility of predictive mapper.

Further notes for another potential application are below, and this is an addition that was discovered near the completion of this dissertation.

3.8.1 Potential for Clustering Applications

An additional, barely explored avenue for predictive mapper could be as an algorithm for clustering and assigning new data points to existing clusters. Although in a very preliminary stage, predictive mapper seems to outperform spectral clustering in identifying classes for noisy half-moon shapes. An example is shown in Figure 3.18 of both the original data and the mapper graph with complete separation. Using spectral clustering (nearest neighbors = 10) 81% of the points in Figure 3.18a were identified correctly, while using predictive mapper with 50% data for mapper graph creation and an additional 25% data as training data provided accuracy of 84% (depending on the method used). It should be noted this work is very preliminary, and it is not assumed that different parameters for clustering would not yield better results than predictive mapper. Additional work is needed here and could be of interest.



(a) Point cloud of half moon.

(b) Mapper graph of half moon point cloud.

Figure 3.18 Example of half moon.

3.9 Conclusion

Some areas and applications of predictive mapper may be useful for use in machine learning models, and in the not-yet-explored avenue of an extension to clustering algorithms. The development of the predictive mapper pipeline for use with the statistical construction of mapper provides another avenue for researchers to explore in topological data analysis, and may be improved by other work in mapper graph optimization.

CHAPTER 4

FEATURE GENERATION FOR MACHINE LEARNING

Many and varied methods currently exist for featurization, which is the process of mapping persistence diagrams to Euclidean space, with the goal of maximally preserving structure. However, and to our knowledge, there are presently no methodical comparisons of existing approaches, nor a standardized collection of test data sets. This chapter provides a comparative study of several such methods, and is taken largely from my paper, A Comparative Study of Machine Learning Methods for Persistence Diagrams [3]. This paper was co-authored by Dr. Luis Polanco and Dr. Jose Perea.

In this chapter we review, evaluate and compare the stable multi-scale kernel, persistence landscapes, persistence images, Carlsson Coordinates - the ring of algebraic functions, template functions, and adaptive template systems. Using these approaches for feature extraction, we apply and compare popular machine learning methods on five data sets: MNIST, Shape retrieval of non-rigid 3D Human Models (SHREC14), extracts from the Protein Classification Benchmark Collection (Protein), MPEG7 shape matching, and HAM10000 skin lesion data set. These data sets are commonly used in the above methods for featurization, and we use them to evaluate predictive utility in real-world applications.

4.1 Introduction

Persistence diagrams are a popular tool in Topological Data Analysis, with a recent interest in feature creation for machine learning tasks, with success in several applications to science and engineering. Many methods exist for the vectorization of persistence diagrams, but a comprehensive evaluation of these methods is largely unexplored. Our goal here is to contribute to the comparative analysis of machine learning methods with persistence diagrams.

Starting with topological descriptors of datasets, in the form of persistence diagrams, we provide examples and methodology to create features from these diagrams to be used

in machine learning algorithms. We provide the necessary background and mathematical justification for six different methods (in chronological order): the Multi-Scale Kernel, Persistence Landscapes, Persistence Images, Adcock-Carlsson Coordinates, Template Systems, and Adaptive Template Systems. To thoroughly evaluate these methods, we have reviewed five different data sets along with the relevant methods to compute persistence diagrams from them. The datasets, persistence diagrams and code to compute the persistence diagrams is readily available for academic use.

As part of this review, we also provide a user guide for these methods, including comparisons and evaluations across the different types of datasets. After computing the six types of features, we compared the predictive accuracy of a ridge regression, random forest, and support vector machine model to assess the type of featurization that is most useful in predictive models. The code developed for this analysis is available, with some functions developed specifically for use in machine learning applications, and easy-to-use jupyter notebooks showing examples of each function with multiple dataset types.

Of these methods, Persistence Landscapes, Adcock-Carlsson Coordinates, and Template Systems are quite accurate and create features for large datasets quickly. Adaptive Template Systems and Persistence Images took somewhat longer to run, however, the Adaptive Template Systems featurization method did improve accuracy over other methods. The Multi-Scale Kernel was the most computationally intensive, and during our evaluation we did not observe instances of it outperforming other methods.

4.2 Background

In order to develop the mathematical foundations needed for doing machine learning with persistence diagrams, it has been informative to first study the structure of the space they form. Additional background pertinent to this chapter and important for understanding properties of persistence methods is included here. For this background we follow [29].

Definition 4.2.1 (Wasserstein distance) *The pth Wasserstein distance between two persistence diagrams, d_1 and d_2 is defined as*

$$W_p(d_1, d_2) = (\inf_{\gamma} \sum_{x \in d_1} ||x - \gamma(x)||_\infty^p)^{\frac{1}{p}} \tag{4.1}$$

where γ ranges over all bijections from d_1 to d_2. The set of bijections is nonempty because of the diagonal.

A related distance is the bottleneck distance, which is the Wasserstein distance with $p = \infty$. Then the space of persistence diagrams also needs to be complete to be appropriate for statistical inference, so the following definitions are key.

Definition 4.2.2 (Persistence Diagram) *A generalized persistence diagram is a countable multiset of points in $\mathbb{R}^2$ along with the diagonal $\triangle = \{(x, y) \in \mathbb{R}^2 | x = y\}$*

Definition 4.2.3 (Total persistence) *The degree-p total persistence diagram d is defined as*

$$Pers_p(d) = \sum_{x \in d} (pers(x))^p \tag{4.2}$$

Definition 4.2.4 (Space of persistence diagrams) *We define the space of persistence diagrams as*

$$D_p = \{d | W_p(d, d_\emptyset) < \infty\} = \{d | Pers_p(d) < \infty\} \tag{4.3}$$

The space of persistence diagrams is complete and separable. Two key theorems also provide stability for both sublevel set persistence and persistence computed using Ripser.

The *stability* theorem of [30] for sublevel set persistence contends that if $\mathbb{X}$ is a finitely triangulated space and $f, g : \mathbb{X} \longrightarrow \mathbb{R}$ are *tame* and continuous, then

$$d_B(\mathsf{dgm}_n(f), \mathsf{dgm}_n(g)) \leq \|f - g\|_\infty$$

for every integer $n \geq 0$. We note that the theorem is still true if continuous is replaced by piecewise linear. Similarly, if $(X, \mathbf{d}_X)$ and $(Y, \mathbf{d}_Y)$ are finite metric spaces, then the stability of Rips persistent homology [31, Theorem 5.2] says that

$$d_B(\mathsf{dgm}_n^{\mathcal{R}}(X), \mathsf{dgm}_n^{\mathcal{R}}(Y)) \leq 2d_{GH}(X, Y)$$

where $d_{GH}(\cdot, \cdot)$ denotes the Gromov-Hausdorff distance [32].
When we refer to featurization methods as stable, this is usually in regards to the Wasserstein distance or bottleneck distance.

Doing statistics and machine learning directly on the space of persistence diagrams turns out to be quite difficult. Indeed, $(\mathcal{D}, d_B)$ does not have unique geodesics, and thus the Fréchet mean of general collections of persistence diagrams is not unique [33]. Since computing averages, and in general, doing linear algebra on persistence diagrams is not available, then several authors have proposed mapping $(\mathcal{D}, d_B)$ to topological vector spaces where further analysis can be done. These methods are the main focus of this review. The theory of vectorization of persistence diagrams is an active area of research, with recent results showing the impossibility of full embeddability. Indeed, even though the space of persistence diagrams with exactly n points can be coarsely embedded in a Hilbert space [34], this ceases to be true if the number of points is allowed to vary (see [35] and [36]). That said, partial featurization is still useful as we will demonstrate here.

4.3 Featurization Methods

For each of the methods below, we start with a collection of persistence diagrams. A persistence diagram can be represented in either the birth-death plane or birth-lifetime plane—some methods will require birth-death coordinates and others will require birth-lifetime coordinates. The **birth-death plane** is the representation pair (x, y) where x is the time of birth, and y is the time of death of the feature in the persistence diagram. The **birth-lifetime plane** can be defined as the collection of points $(x, y - x)$, where (x, y) is

in birth-death coordinates. In this manner, we define lifetime as the persistence $y - x$ of a feature (x, y). The persistence diagrams of a particular geometric object can be calculated in a variety of ways, which will be made explicit for each dataset at time of evaluation.

4.3.1 Multi-Scale Kernel

The Multi-Scale Kernel of [37] defines a Kernel over the space of persistence diagrams, which can then be used in various types of kernel learning methods. In general, a kernel k is by definition a symmetric and positive definite function of two variables. Mathematically, from [37], given a set X, a function $k : X \times X \to \mathbb{R}$ is a **kernel** if there exists a Hilbert space H, called the **feature space**, and a map $\Phi : X \to H$, called the **feature map**, such that $k(x, y) = \langle \Phi(x), \Phi(y) \rangle_H$ for all $x, y \in X$. The kernel induces a distance on X defined as

$$d_k(x, y) = (k(x, x) + k(y, y) - 2k(x, y))^{\frac{1}{2}} = \|\Phi(x) - \Phi(y)\|_H.$$

The authors of [37] propose a multi-scale kernel on $\mathcal{D}$ as follows. Given $F, G \in \mathcal{D}$, the persistence scale space kernel k_σ is

$$k_\sigma(F, G) = \langle \Phi_\sigma(F), \Phi_\sigma(G) \rangle_{L^2(\Omega)} \tag{4.4}$$

where $\Phi_\sigma : \mathcal{D} \to L^2(\Omega)$ is the associated feature map, and $\Omega \subset \mathbb{R}^2$ is the closed half-plane above the diagonal. Deriving the solution of a distribution-analogue of the Heat equation with boundary conditions in Definition 1 of [37], the closed form expression of the multi-scale kernel is:

$$k_\sigma(F, G) = \frac{1}{8\pi\sigma} \sum_{p \in F, q \in G} e^{-\frac{\|p-q\|^2}{8\sigma}} - e^{-\frac{\|p-\bar{q}\|^2}{8\sigma}}$$

where if $q = (a, b)$, then $\bar{q} = (b, a)$.

The multi-scale kernel is shown to be stable w.r.t the 1-Wasserstein distance by Theorem 2 of [37], which is a desirable property for classification algorithms. However, by Theorem 3 of [37], the multi-scale kernel is not stable in the Wasserstein sense for $1 < p \leq \infty$.

4.3.2 Persistence Landscapes

Persistence landscapes are a mapping of persistence diagrams into a function space that is either a Banach space or Hilbert space ([38]). Advantages of persistence landscapes are that they are invertible, stable, parameter-free, and nonlinear. Persistence landscapes can be computed from a persistence diagram as follows.

From [38], for a persistence diagram $D = (a_i, b_i)_{i \in I}$, and for $a < b$, let

$$f_{(a,b)}(t) = \max(0, \min(a + t, b - t)) \tag{4.5}$$

and

$$\lambda_k(t) = \text{kmax} \left\{ f_{(a_i,b_i)}(t) \right\}_{i \in I} \tag{4.6}$$

with kmax as the kth largest element.

The **persistence landscape** is the sequence of piecewise linear functions, $\lambda_1, \lambda_2, \ldots : \mathbb{R} \to \mathbb{R}$. Bubenik shows desirable properties for working with persistence landscapes in statistical modeling, in particular that even if unique means do not exist in the set of persistence diagrams, persistence landscapes do have unique means and the mean landscape converges to the expected persistence landscape. Figure 4.1 shows an example of persistence landscapes from the MPEG7 dataset, described in the data section.

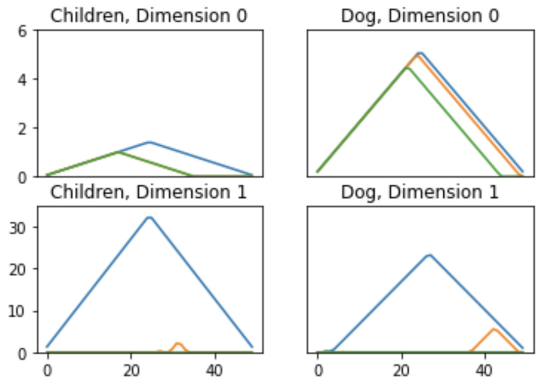


Figure 4.1 Persistence Landscapes from the MPEG7 dataset to show differences in features. Each color corresponds to a different landscape, i.e. λ_k for $k = 1, 2, 3$.

4.3.3 Persistence Images

From [39], persistence images are a mapping sending a persistence diagram to an integrable function, called a persistence surface. Fixing a grid on $\mathbb{R}^2$, the integral over this grid yields pixel values forming the persistence image. Advantages of persistence images include a representation in $\mathbb{R}^n$, stability, and ease of computation. When calculating the persistence image, a resolution, a distribution, and a weighting function are required as parameters. It is worth noting that the resolution (i.e., number of pixels) determines the number of features computed by the persistence image.

More explicitly, let D be a persistence diagram in birth-lifetime coordinates. We take $\phi_u : \mathbb{R}^2 \to \mathbb{R}$ to be a differentiable probability distribution. Using, for instance, the Gaussian Distribution with mean u and variance σ^2 we have

$$\phi_u(x, y) = \frac{1}{2\pi\sigma^2} e^{-[(x-u_x)^2+(y-u_y)^2]/2\sigma^2}$$

The persistence surface $\rho_D : \mathbb{R}^2 \to \mathbb{R}$ is the function

$$\rho_D(z) = \sum_{u=(x,y-x)\in D} f(u)\phi_u(z)$$

with $f : \mathbb{R}^2 \to \mathbb{R}$, a nonnegative weighting function that is zero along the horizontal axis, continuous, and piecewise differentiable. For example, $f(x) = y$ would satisfy this requirement and is used frequently as $f(x)$. The **persistence image** is then $I(\rho_D)p = \int\int_p \rho_D dydx$, where integration is over the fixed grid on $\mathbb{R}^2$. This creates an image depicting high and low density areas in the defined grid, that are represented as a high-dimensional vector for use in machine learning algorithms. An example is shown in Figure 4.2 taken from the MNIST dataset.

4.3.4 Adcock-Carlsson Coordinates: The ring of algebraic functions on persistence diagrams

This method is explored by [40], where the authors highlight the fact that any persistence diagram with exactly n points can be described by a vector of the form $(x_1, y_1, x_2, y_2, \ldots, x_n, y_n)$

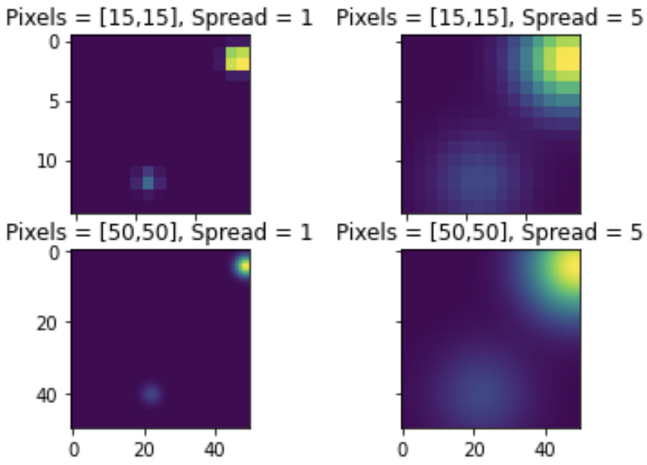


Figure 4.2 Persistence Images of a 5 from the MNIST set in dimension 0.

where x_i denotes the birth of the i-th class and y_i the corresponding death time. This featurization method also appears in other applications, such as brain imaging [41], chatter identification [42], image classification [40], and also led to related work with additional stability properties [43].

From [40] the general form below is used for feature creation for **Adcock-Carlsson coordinates**.

$$p_{a,b} = \sum_i (x_i + y_i)^a (y_i - x_i)^b$$

for integers $a \geq 0$ and $b \geq 1$.

Adcock-Carlsson coordinates that both emphasized more persistent features, and took into account all types of features were desired. Based on this, the following features for both the 0-dimensional and 1-dimensional persistence diagrams meet this criteria, as suggested

in [40]:

$$\sum_i x_i(y_i - x_i) \tag{4.7}$$

$$\sum_i (y_{max} - y_i)(y_i - x_i) \tag{4.8}$$

$$\sum_i x_i^2(y_i - x_i)^4 \tag{4.9}$$

$$\sum_i (y_{max} - y_i)^2(y_i - x_i)^4. \tag{4.10}$$

The theoretical motivation for Adcock-Carlsson Coordinates is as follows. Define $Sp^n(\mathbb{R}^2)$ as the n-fold symmetric product of $\mathbb{R}^2$ (unordered n-tuples). We note the set of persistence diagrams with exactly n points of the form $\{(x_1, y_1), ..., (x_n, y_n)\}$ is a subset of $Sp^n(\mathbb{R}^2)$. The inclusions $Sp^n(\mathbb{R}^2) \hookrightarrow Sp^{n+1}(\mathbb{R}^2)$ thus produce an inverse system of affine coordinate rings

$$\cdots \to A[Sp^{n+1}(\mathbb{R}^2)] \to A[Sp^n(\mathbb{R}^2)] \to \cdots$$

which provide the basis for studying algebraic functions on the space of persistence diagrams.

With this setting in mind, the main goal of [40] is to determine free generating sets for the subalgebra of $A[Sp^\infty(\mathbb{R}^2)]$ comprised of elements which are invariant under adjoining a point of zero persistence to a persistence diagram. The following theorem is an answer to this question (see Theorem 1 [40]).

Theorem 6 *The subalgebra of 2-multisymmetric functions invariant under adding points with zero persistence, is freely generated over* $\mathbb{R}$ *by the set of elements of the form*

$$p_{a,b} = \sum_i (x_i + y_i)^a (y_i - x_i)^b$$

for integers $a \geq 0$ *and* $b \geq 1$.

These are the features we call **Adcock-Carlsson coordinates**.

4.3.5 Template Systems

As with other methods, the goal of template systems as a featurization method is to embed the space of persistence diagrams into a real-valued vector space since persistence diagrams cannot be used directly in machine learning models. Template systems additionally separate the space of persistence diagrams, which is desired for classification accuracy.

Template systems first start with a **template function**, which is any function on $\mathbb{R}^2$ that is continuous, and has compact support contained within the upper half plane, $\mathbb{W} := \mathbb{R} \times \mathbb{R}_{>0}$. Given a template function $f : \mathbb{W} \to \mathbb{R}$ and a persistence diagram, D, the template function becomes a function on persistence diagrams [44]:

$$\nu_f(D) := \sum_{\mathbf{x} \in D} f(\mathbf{x}).$$

A **template system** is a collection of template functions that separates points on the persistence diagrams. The specific template functions I used for the review are **tent functions**, where given a point $\mathbf{a} = (a, b)$, and a radius δ, define

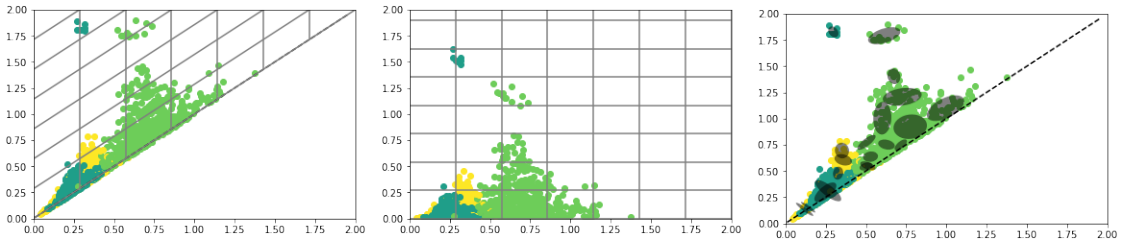
$$g_{\mathbf{a},\delta}(x, y) = |1 - \tfrac{1}{\delta}\max\{|x - a|, |y - b|\}|_+$$

so that with a persistence diagram $D = \mathbf{x} = (x_i, y_i)$, the value of the tent function is

$$G_{\mathbf{a},\delta}(D) = \sum_{\mathbf{x} \in D} g_{\mathbf{a},\delta}(\mathbf{x}).$$

To compute the values, a grid size and associated parameters are chosen so that g is compactly supported in $\mathbb{W}$. This is illustrated in Figure 4.3.

The advantage of working with these template systems is that they can be used to approximate real-valued functions on the space of persistence diagrams as proven by the following theorem (see Theorem 29 [45]). Denote $C_c(\mathbb{W})$ to be the set of compactly supported continuous functions, $\mathcal{D}$ the space of persistence diagrams and $C(\mathcal{D}, \mathbb{R})$ the set of continuous functions from $\mathcal{D}$ to $\mathbb{R}$.



(a) Birth-Death Coordinates. (b) Birth-Persistence. (c) Adaptive Template Systems.

Figure 4.3 Figures 4.3a and 4.3b show the grid required for the computation of tent functions in different coordinates, and Figure 4.3c shows adaptive template systems.

Theorem 7 *Let $\mathcal{T} \subset C_c(\mathbb{W})$ be a template system for $\mathcal{D}$, let $C \subset \mathcal{D}$ be compact, and let $F : C \to \mathbb{R}$ be continuous. Then for every $\epsilon > 0$ there exist $N \in \mathbb{N}$, a polynomial $p \in \mathbb{R}[x_1, ..., x_N]$ and template functions $f_1, \ldots, f_N \in \mathcal{T}$ so that*

$$\left| p\big(\nu(D, f_1), \ldots, \nu(D, f_N)\big) - F(D) \right| < \epsilon$$

for every $D \in C$.

That is, the collection of functions of the form $D \to p(\nu(D, f_1), \ldots, \nu(D, f_N))$, is dense in $C(\mathcal{D}, \mathbb{R})$ with respect to the compact-open topology.

4.3.6 Adaptive Template Systems

The Adaptive Template Systems methodology of [46] concerns itself with improving and furthering some of the work presented in [45] and [47]. The goal is to produce template systems that are adaptive to the input data set and the supervised classification problem at hand. One shortcoming of template systems, like tent functions, when applied to Theorem 7 is that without prior knowledge about the compact set $C \subset \mathcal{D}$, the number of template functions that carry no information relevant to the problem can be high. By reducing this overhead, adaptive templates improve the computation times and accuracy in some specific problems.

The relationship between template systems and adaptive template systems is demonstrated in Figure 4.3, showing the adaptive template systems depend on density of data. To do

so, given a compact set $C \subset \mathcal{D}$ we consider the set $S = \bigcup_{D \in C} D \subset \mathbb{W}$ along with different algorithms such as Gaussian mixture models (GMM) ([48]), Hierarchical density-based spatial clustering of applications with noise (HDBSCAN) ([14]) and Cover-Tree Entropy Reduction (CDER) ([49]) to define a family of ellipsoidal domains $\left\{\mathbf{z} \in \mathbb{R}^2 : (\mathbf{z}-\mathbf{x})^* A(\mathbf{z}-\mathbf{x}) \leq 1\right\}$ in $\mathbb{W}$, fitting the density distribution of S. Here A is a 2×2 symmetric matrix and $\mathbf{x} \in \mathbb{R}^2$.

Once this family of ellipsoidal domains is computed, we use them to define the following **adaptive template functions**

$$f_A(\mathbf{z}) = \begin{cases} 1 - (\mathbf{z}-\mathbf{x})^* A(\mathbf{z}-\mathbf{x}) & \text{if } (\mathbf{z}-\mathbf{x})^* A(\mathbf{z}-\mathbf{x}) < 1 \\ \\ 0 & \text{if } (\mathbf{z}-\mathbf{x})^* A(\mathbf{z}-\mathbf{x}) \geq 1. \end{cases}$$

4.4 Datasets

The five different datasets considered in this work were chosen from a collection of experiments presented in the literature of topological methods for machine learning. We acknowledge that this selection is inherently biased towards datasets with favorable performance with regards to specific topological methods. Nevertheless, we counterbalance this by applying all the evaluated featurization methods to all the data sets here considered and compare the classification results across all the presented methodologies. This comparative work showcases how the variation between methods results in the need for the user to find a suitable combination of featurization methods and parameter tuning to obtain optimal results in a given dataset. As such, readers should view this as a resource for their own analysis, and not as a recommendation for specific techniques.

For all datasets and methods, parameter tuning was done using a grid search method on a subset of data that was not used to report final results, and parameters were chosen based on performance of a ridge regression model, a random forest and a support-vector machine (SVM) model. It is worth noting a weakness of the analysis in that the same parameters were used in the feature set calculation for all reported models, and run with a single split.

This was due to time required for feature calculation.

The ridge regression and random forest classifier were run with default parameters, and the support-vector machine was run using the radial basis function (RBF) with some tuning on the cost parameter (C). The exception is for the Multi-Scale Kernel feature set - we only fit a support-vector machine model, and was specific to feature computation for the Multi-Scale Kernel. Due to computation times for the Multi-Scale Kernel and the direct use of the features in a kernel support-vector machine model, this was the most appropriate choice for model fitting. Preliminary results showed that ridge regression did not provide additional accuracy for kernel methods as well. Each dataset was sampled to create training and testing sets for a 10 or 100 trials depending on size, with the exception of the Protein Classification Dataset, which included indices for machine learning tasks.

Random forest classifiers as presented in [50] are used to solve the same classification problems presented for each data set. Parameters such as number of trees in each forest and the size of each tree are chosen based on performance and tuned on the testing set.

4.4.1 MNIST

The MNIST dataset from [4] is a database of 70,000 handwritten single digits of numbers 0 to 9. An example image from the MNIST database is shown in Figure 4.4.

The calculation of persistence diagrams for the MNIST dataset is as in [40]. The persistence diagrams are calculated using a "sweeping" motion in one of four directions: left to right, right to left, top to bottom, or bottom to top, corresponding to the 0-dimensional and 1-dimensional persistence diagrams. This method creates a base of 8 different persistence diagrams to use in the application of methods. To compute this filtration, pixels are converted to a binary response with intensity calculated based on position. This has the effect that depending on the direction of sweep, features will have different birth and death times, providing distinct information for each direction. Figure 4.4 shows the various calculations of persistence diagrams for an example number eight. Both 0-dimensional and

1-dimensional persistence diagrams were used for the MNIST dataset, noting that some observations did not have 1-dimensional persistence diagrams, so these observations were filled with a single diagram of birth-death coordinate of [0,.01].

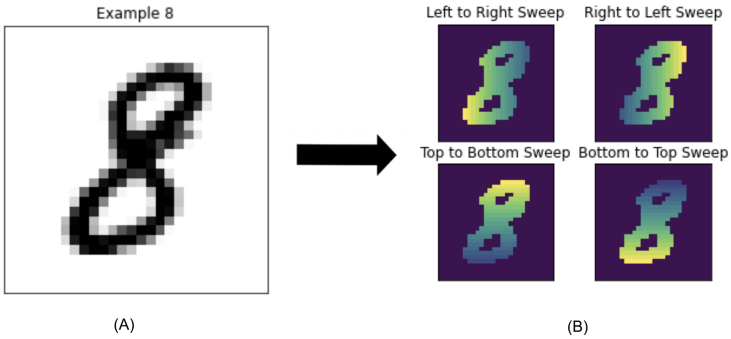


Figure 4.4 A) Example number 8 from the MNIST dataset. B) The same number 8 after computing each of the 4 types of coordinate transforms to compute the persistence diagrams.

The number of topological features available for model fitting is dependent on the method. For the Persistent Images, Persistence Landscapes, and Template Systems there are 8 features each. The Multi-Scale Kernel produces 8 different kernel matrices, and for Adcock-Carlsson Coordinates, 32 different features were computed from these persistence diagrams.

For the MNIST dataset, a random sample of 1000 images was used to tune parameters, with 80% used for the training portion, and 20% used for the testing portion. We used the set of 60,000 images corresponding to the training set of MNIST to create our own training and testing sets for model fitting and evaluation. For this set of 60,000, 10 trials were run with an 80% training and 20% testing split to determine model performance.

4.4.2 SHREC14

We evaluated the SHREC 2014 dataset ([51]) in the same manner as the authors of [46]. To compute the topological features, the authors of [37] describe using a heat kernel signature to compute persistence diagrams for both the 0-dimensional and 1-dimensional

persistence diagrams. The kernel is computed, and then the persistence diagrams were computed using Dipha [52], which can directly handle meshes. We were provided the dataset as a set of persistence diagrams. The dataset consists of 15 different labels, corresponding to 5 different poses for the three classes of male, female, and child figures. As noted in [46], parameters in the dataset define different machine learning tasks due to a different calculation of the heat kernel signature, and for this evaluation we focused problem 6, which has the highest accuracy ranging from 89% - 92% as reported in[46].

For the SHREC14 dataset, a random sample of 90 images (30% of the data) was used to tune the model and determine appropriate model parameters. The remaining 210 observations were split into 80% training and 20% testing for 100 trials to report final model fit, which is listed below. Persistence diagrams for 0-dimensional homology and 1-dimensional homology were computed for this dataset. Table 4.3 shows complete results for the SHREC 2014 dataset.

4.4.3 Protein Classification

We use the Protein Classification Benchmark dataset PCB00019 [53] as another type of data to evaluate the topological methods above. This specific set contains information for 1357 proteins corresponding to 55 classification problems, and we reported on 54 of the problems using one to tune parameters. The training and testing index were provided, and the mean accuracy was reported for both training and testing sets using these indices. Table 4.4 shows results from our experiments using the training and testing indices provided in the original dataset.

Persistence diagrams for this dataset were computed for each protein by considering the 3-D structure (provided in [54]) as a point cloud in $\mathbb{R}^3$. This point cloud was built using the x, y and z position of each atom in the molecule at hand. With this information the persistent 0-dimensional and 1-dimensional homology is computed using Ripser [55].

4.4.4 MPEG7

The mpeg-7 dataset from [56] is a database of object shapes in black and white, with 1400 shapes in 70 classes. An example from the original dataset is shown in Figure 4.5 along with the contour as described below.

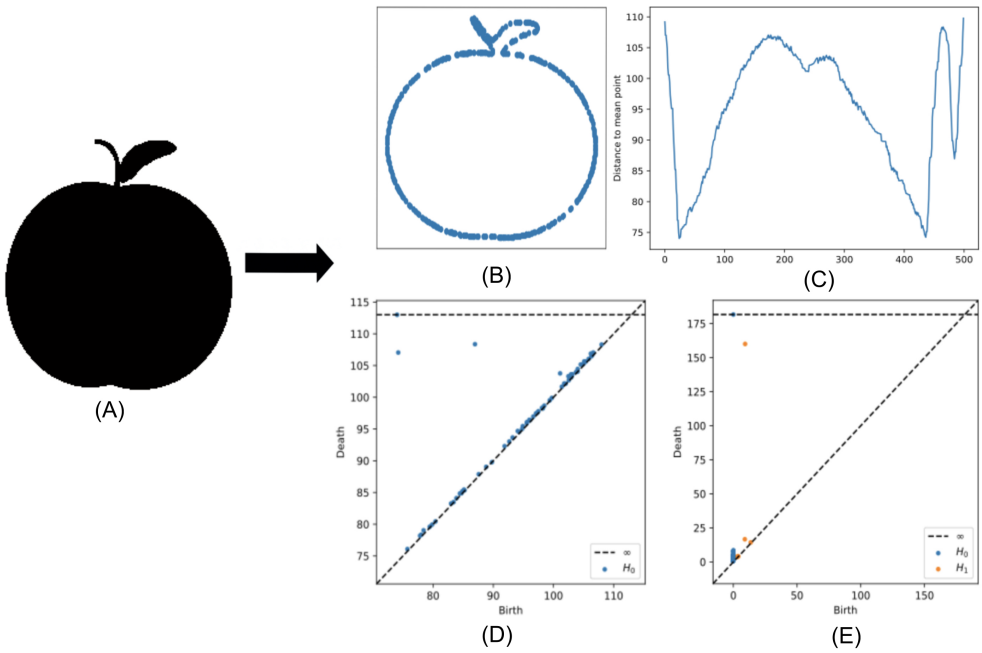


Figure 4.5 (A) An example apple from MPEG7 data. (B) An example image contour used for MPEG7. (C) The distance to mean point calculation used for sublevel set persistence. (D) Persistence diagrams from lower star persistence. (E) Persistence diagrams from the contour.

To compute persistence diagrams, first the image contour is computed by placing observations from the point cloud into a sequence. The distance curve is computed as the distance from the center of the sequence. Sublevel set persistence is taken using the computed distance curves as point cloud data. Persistence diagrams for both 0-dimensional and 1-dimensional homology were computed for this dataset.

An example notebook of MPEG7 is provided using only 4 shapes - apple, children, dog, and bone. This approach is due to the initial difficulty in getting accurate models for the full

dataset. Due to the small number of samples (80 total) and lack of repeated sampling, the estimates provided for this dataset are not stable and are not reported. We used this dataset for a timing comparison of featurization methods from persistence diagrams. We do not report on the results of this dataset.

4.4.5 HAM10000

The HAM10000 dataset provided by [57] is a collection of 10000 images of skin lesions with one 7 potential classifications: (i) Actinic Keratoses and Intraepithelial Carcionma, (ii) Basal cell carcinoma, (iii) Benign keratosis, (iv) Dermatofibroma, (v) Melanocytic nevi, (vi) Melanoma, (vii) Vascular skin lesions. A total of 18 persistence diagrams for this set were calculated using the methods outlined in [58], 9 corresponding to the 0-dimensional homology and 9 corresponding to the 1-dimensional homology.

To obtain such diagrams, first a mask is computed by implementing the methodology proposed in [58]. The mask encodes the image with a binary response of white if that region (pixel) of the image if part of the lesion, and black if it is part of the healthy tissue. Once the mask is computed it is applied to the original image to identify the lesion. The area recognized as the lesion is converted into 3 different color models: RGB, HSV and XYZ. Each color model is split into their corresponding channels, and for each channel we use sublevel set filtration to obtain 0-dimensional and 1-dimensional persistence diagrams. In total, for each image on the data set we obtain 18 persistence diagrams, 9 in homological dimension 0 and 9 for homological dimension 1. An example image and this process is shown in Figure 4.6.

To tune the models, a random sample of 250 images were taken and a ridge regression, random forest and support vector machine model were fit to determine parameters. The remaining 9750 images were split into an 80% training and 20% testing set to report final results.

To evaluate the HAM10000 dataset, due to the large number of birth and death pairs in

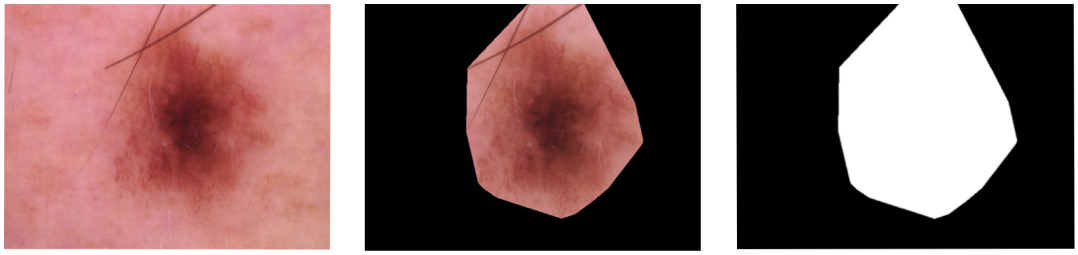


Figure 4.6 (A) Example of skin lesion in HAM10000 (B) Skin lesion with mask (C) Mask only dataset.

each persistence diagram, subsampling of persistent features was required. Each observation in a data set, for example an image, will yield 18 persistence diagrams corresponding to homological features in that observation. In the HAM10000 dataset, there was an average of 5,783 birth-death pairs in each persistence diagram. This was an issue to complete computation for the vectorization methods, even for adaptive templates, so each persistence diagram was subsampled as follows.

The method of subsampling is two steps: Highly persistent features were always included, and a uniform random sampling method (without replacement) was used to sample the remaining points. The threshold for feature lifetime and number of points to sample was determined by using parameters that preserve the distribution of points in each persistence diagram. As a result, features in each persistence diagram with a lifetime of 5 or more were automatically included, and 5% of the rest of the points were also included. This resulted in sampled persistence diagrams with an average of 290 points each (see Table 4.1).

4.5 User Guide

4.5.1 Available functions

As part of the available code, a function for each method is included. Each function requires two sets of persistence diagrams, a training set and a testing set, and parameters specific to the function. The function returns two feature sets for that method, corresponding to the training and test set respectively. Each function also prints the time in seconds taken

at the end of each run. In this section of the user guide each function is described, along with the required parameters for the function.

The **Multi-Scale Kernel** feature matrix can be computed using the function *kernel_features* or *fast_kernel_features*. It is recommended to use *fast_kernel_features* due to computation time. Both functions require a parameter *sigma*, denoted as *s* in the function with a default value of 4. In [37] this parameter is referred to as the scale parameter. From the definition, we note that as *sigma*, σ, increases the function decreases. Increasing *sigma* results in a less diffuse kernel matrix, while decreasing *sigma* results in a more diffuse kernel matrix.

Due to time required for the Multi-Scale Kernel, there are two additional sets of functions that use Numba ([59]) for significantly faster computation. In the current implementation, these are not able to be combined with multi-core processing (MPI for example), and have a different format than the other functions included. These functions are provided in the github repository for this project, and were used to compute results for the Multi-Scale Kernel for the MNIST dataset.

The **Persistence Landscapes** features can be computed using *landscape_features*. The Multi-Scale Kernel function, *landscape_features* requires two parameters: the number of landscapes, *n* and resolution, *r*. The number of landscapes parameter, *n*, controls the number of landscapes used, and the resolution, *r*, controls the number of samples used to compute the landscapes. The default parameters for *n* is 5 and *r* is 100.

The **Persistence Images** can be computed using the function *persistence_image_features*. The *persistence_image_features* function requires two parameters, pixels and spread. The pixels, *p* is a pair that defines the number of pixels along each axis. The spread, *s*, is the standard deviation of the Gaussian kernel used to generate the persistence image. It is worth noting that the implementation here uses the Gaussian kernel, however, other distributions could be chosen so that *s* would correspond to parameters specific to the chosen distribution.

Additionally, the weighting function is constant for this implementation. Increasing spread increases the variance for each distribution, resulting in larger "hot spots". Increasing pixels provides a smoother distribution, whereas decreasing pixels yields a less smooth distribution. Note that increasing pixels increases computation time. This is demonstrated in Figure 4.2 in the methods section.

The **Adcock-Carlsson Coordinates** features can be computed using the function *carlsson_coordinates*, does not require any parameters. This function returns four different features for every type of persistence diagram provided. So for datasets that have persistence diagrams corresponding to 0-dimensional and 1-dimensional homology, 8 features are returned for machine learning. The features returned correspond to the four coordinates calculated in [40].

The **Template Systems** features can be computed using the function *tent_features*, and has a choice of two parameters: *d*, which defines the number of bins in each axis and *padding*, which controls the padding around the collection of persistence diagrams. This function returns a training and testing set. This function computes the tent features from [45].

The **Adaptive Template Systems** features can be called with the function *adaptive_features*, and requires the labels for the training set. Users can choose three different types of Adaptive Templates: Gaussian Mixture Modeling (GMM), Cover-Tree Entropy Reduction (CDER), and Hierarchical density-based clustering of applications with noise (HDBSCAN). The parameter *d* refers to the number of components when using the GMM model type. This would be minimally the number of classes in your data, and ideally represents closer to the number of distributions in the data that correspond to each observation. Details on these methods can be found in [46], as well as the original references linked in the methods section. During this evaluation, we evaluated adaptive templates using both GMM and CDER methods, but did not formally evaluate HDBSCAN. HDBSCAN was

difficult to formally assess as we had difficulty with completion of the algorithm for some datasets. For those datasets we were able to complete, we did not notice an improvement over other adaptive methods.

4.6 Results

One consideration we must make before analysing the results comes from the computation of **Multi-Scale Kernel** features. As explained for each dataset in Section 4.4, more often than not we will compute multiple persistent diagrams per data point in a given data set. Since this multi-scale kernel provides a notion of similarity between persistent diagrams (see [37]) we require it to be computed between diagrams corresponding to the same dimension homology and method type. For example, the kernel matrix that corresponds to the 0-dimensional homology of a data set is computed using the persistence scale space kernel between two sets of persistence diagrams that represent the 0-dimensional homology. This means that for a dataset that has sets of 0-dimensional homology persistence diagrams and 1-dimensional homology persistence diagrams, two kernel matrices were returned (one per each dimension).

The kernel matrix used in our models is the sum of available kernels, and differs based on the persistence diagrams available for each dataset. While this does improve accuracy significantly over individual kernel matrices, other methods of combining kernel features were not explored in this dissertation, but is available in [60] for the interested reader. The available parameter, `sigma`, is consistent across all types of diagrams for our evaluation.

For each of the other methods, **Persistence Landscapes**, **Persistence Images**, **Adcock-Carlsson Coordinates**, **Template Systems**, and **Adaptive Template Systems**, each feature matrix was constructed for the relevant set of diagrams, and all topological types were used in fitting the same model.

The datasets used in this analysis were of varying size, both in terms of observations and the size of sets of persistence diagrams. As noted in the descriptions of data, the types of

Dataset Characteristics				
Dataset	Observations	Diagrams	Average Pairs	Min/Max Pairs
MNIST	70,000	8	1.15	0/7
SHREC14	300	2	14	1/29
Protein	1357	2	346	3/500
MPEG7	1400	2	205	1/500
HAM10000	10,000	18	5783	13/32610

Table 4.1 Characteristics of each dataset. The column headings can be explained as such: Observations - number of observations in the dataset, Diagrams - the number of homological types used to compute persistence diagrams, Average Pairs - the average number of birth/death pairs across the set of persistence diagrams for a single observation in the original dataset, and Min/Max Pairs - the minimum and maximum number of birth/death pairs across the set of persistence diagrams for a single observation in the original dataset.

persistence diagrams calculated also differs. A summary of characteristics for each dataset is included in Table 4.1.

4.6.1 MNIST

The Multi-Scale Kernel features calculated yielded 8 different kernel matrices, and the final kernel matrix was calculated using the unweighted summation of these kernels as in [60]. Due to the time needed for computation of the Multi-Scale Kernel, a smaller set of 12,000 observations was used to report final results.

Table 4.2 shows complete results for the MNIST analysis. Four different methods (highlighted on the table) provided similar results for the MNIST dataset, and we note the SVM model had higher accuracy in each case. This table, and all subsequent results tables, include the method used to construct topological features, training and test accuracy, and model and parameters used for evaluation.

4.6.2 SHREC14

Results are reported in Table 4.3. Adaptive Template Systems and Persistence Landscapes were the two methods with highest classification accuracy on the test dataset, with Template Systems and the Multi-Scale Kernel performing nearly as well.

Full Results for MNIST Dataset			
Topological Method	Train	Test	Model
Multi-Scale Kernel (sum of Kernels for 12,000 observations)	.6895 ± .0035	.6932 ± .0117	SVM
Persistence Landscapes	.8844 ± .0004	.8786 ± .0019	Ridge Regression
	.9231 ± .0004	.9180 ± .0018	SVM(RBF)
	.5814 ± .0098	.5828 ± .0098	Random Forest
Persistent Images	.8997 ± .0005	.8934 ± .0021	Ridge Regression
	.9368 ± .0004	.9199 ± .0023	SVM (RBF)
	.6889 ± .0036	.6953 ± .0123	Random Forest
Adcock-Carlsson Coordinates	.8590 ± .0010	.8547 ± .0030	Ridge Regression
	.9525 ± .0004	.9356 ± .0018	SVM (RBF)
	.7214 ± .0092	.7170 ± .0097	Random Forest
Template Systems	.896 ± .0005	.8959 ± .0017	Ridge Regression
	.9638 ± .0003	.9477 ± .0015	SVM(RBF)
	.6967 ± .0035	.6973 ± .0031	Random Forest
Adaptive Template Systems	.8819 ± .0016	.8817 ± .0027	Ridge Regression (GMM)
	.9515 ± .0021	.9363 ± .0021	SVM(RBF) (GMM)
	.6914 ± .0188	.6932 ± .0209	Random Forest

Table 4.2 Results from the MNIST Dataset using the average model classification accuracy ± standard deviation over 10 trials.

4.6.3 Protein Classification

Nearly all of the topological methods in this thesis provided similar classification accuracy for this dataset. We observe the testing accuracy as higher than the training accuracy for this dataset, and the results are similar to those in [61]. The Multi-Scale Kernel though did not perform as well and as shown in Figure 4.7 is the most computationally intensive. Results are reported in Table 4.4.

Full Results for SHREC14 Dataset			
Method	Train	Test	Model
Multi-Scale Kernel (sigma = .5, sum of kernels)	.8942 ± .0142	.8938 ± .0464	Kernel SVM
Persistence Landscapes (n = 5, r = 200)	.9968 ± .0037	.9312 ± .0336	Ridge Regression
	.9302 ± .0098	.9186 ± .0417	SVM(RBF, c = 10)
	.9739 ± .0190	.9114 ± .0441	Random Forest
Persistent Images (p = 40, s = .5)	.7243 ± .0387	.7048 ± .0588	Ridge Regression
	.9067 ± .0147	.8876 ± .0479	SVM (RBF, c = 1)
	.9855 ± .0092	.865 ± .0764	Random Forest
Adcock-Carlsson Coordinates	.85 ± .0199	.7124 ± .0814	Ridge Regression
	.8671 ± .0183	.6928 ± .0599	SVM (RBF, c = 50)
	.9147 ± .0299	.6976 ± .0899	Random Forest
Template Systems (d = 12, p = 1.1)	.9442 ± .0087	.9100 ± .0405	Ridge Regression
	.9350 ± .0079	.9159 ± .0383	SVM(RBF, c = 1)
	..9483 ± .0214	.8874 ± .0481	Random Forest
Adaptive Template Systems (CDER)	.9937 ± .0078	.9169 ± .0395	Ridge Regression
	.9929 ± .0083	.9064 ± .0397	SVM(RBF, c = 10)
	.9729 ± .0200	.9164 ± .0422	Random Forest

Table 4.3 Results from the Shrec14 Dataset using the average model classification accuracy ± standard deviation over 100 trials.

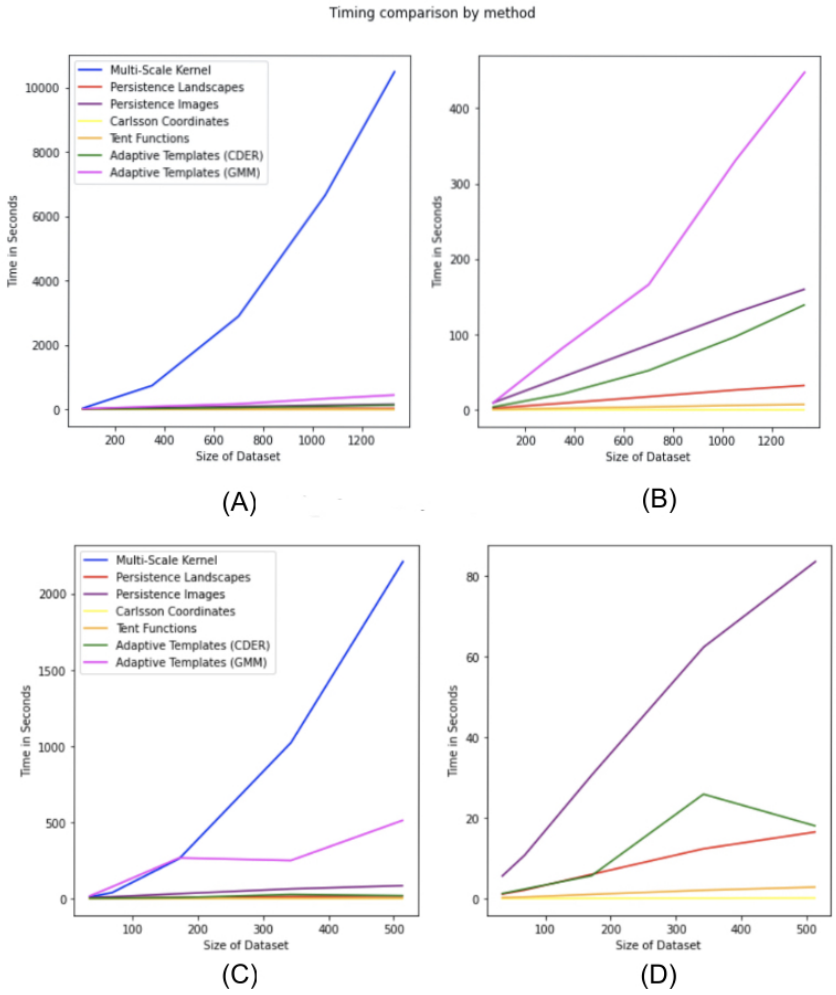


Figure 4.7 Comparison of timing by method. The legend is the same for all plots. The x-axis represents the size of the dataset, and the y-axis represents the time in seconds required for calculation of all of the persistence diagrams associated with the dataset of the given size. A) Timings for the MPEG7 Dataset including the Multi-Scale Kernel. B) Timings for the MPEG7 Dataset excluding the Multi-Scale Kernel. C) Timings for the Protein Dataset including all features. D) Timings for the Protein Dataset excluding Multi-Scale Kernel and Adaptive Templates (GMM).

4.6.4 HAM10000

Due to run time for the large number of points in each persistence diagram, even after subsampling, results were not reported for the Multi-Scale Kernel or Adaptive Template Systems.

Full Results for Protein Dataset			
Method	Train	Test	Model
Multi-Scale Kernel (sum of kernels)	.8294 ± .1063	.8803 ± .0702	Kernel SVM
Persistence Landscapes	.9108 ± .0615	.9620 ± .0204	Ridge Regression
	.9012 ± .0682	.9782 ± .0151	SVM(RBF)
	.9011 ± .0686	.9782 ± .0152	Random Forest
Persistent Images	.9011 ± .0682	.9758 ± .0165	Ridge Regression
	.9007 ± .0684	.9782 ± .0151	SVM (RBF)
	.9008 ± .0685	.9782 ± .0151	Random Forest
Adcock-Carlsson Coordinates	.9008 ± .0685	.9780 ± .0151	Ridge Regression
	.9009 ± .0685	.9782 ± .0151	SVM (RBF)
	.9015 ± .0677	.9779 ± .0151	Random Forest
Template Systems	.9008 ± .0684	.9780 ± .0151	Ridge Regression
	.9020 ± .0678	.9782 ± .0151	SVM(RBF)
	.9016 ± .0678	.9775 ± .0152	Random Forest
Adaptive Template Systems	.9008 ± .0685	.9782 ± .0151	Ridge Regression (CDER)
	.9007 ± .0684	.9782 ± .0151	SVM(CDER) (HDB)
	.9100 ± .0685	.9800 ± .0151	Random Forest

Table 4.4 Results from the Protein Dataset using the average model classification accuracy ± standard deviation over 54 trials corresponding to the predefined indices of the dataset.

Results are listed in Table 4.5. The HAM10000 dataset presented the largest computational challenge during this review, and is a continued area of research.

4.7 Computation Time of Features

Formal timings were captured for all features for the 0-dimensional persistence diagrams for the MPEG7 and Protein Datasets. A comparison of timings is in Figure 4.7. The timing reported is for the generation of features from one type of persistence diagram for a dataset of that size. This means when computing a training feature set and testing feature set for multiple types of persistence diagrams, the expected time to generate features can be significantly longer. For example, in the MNIST dataset we compute 4 different types

Results for HAM10000 Dataset			
Topological Method	Train	Test	Model
Multi-Scale Kernel	Did Not Run		
Persistence Landscapes	.8347 ± .0022 .6695 ± 0 .6695 ± 0	.6881 ± .0074 .6692 ± $1.2e-16$ .6692 ± $1.2e-16$	Ridge Regression SVM(RBF, c = 1) Random Forest
Persistence Images (pixels = 20, spread = 1)	.7417 ± .0017 .7122 ± .0012 .6695 ± 0	.6371 ± .0671 .6895 ± .0031 .6692 ± $1.2e-16$	Ridge Regression SVM (RBF, c= 1) Random Forest
Adcock-Carlsson Coordinates	.6719 ± .0007 .6801 ± .0009 .6695 ± 0	.6696 ± .0025 .6710 ± .0019 .6692 ± $1.12e-16$	Ridge Regression SVM (RBF) Random Forest
Template Systems (d = 10, p = 1.5)	.7193 ± .0015 .7830 ± .0024 .6695 ± 0	.6987 ± .0041 .7303 ± .0054 .6692 ± $1.2e-16$	Ridge Regression SVM(RBF, c = 5) Random Forest
Adaptive Template Systems	Did Not Run		

Table 4.5 Results from the HAM10000 Dataset using the average model classification accuracy ± standard deviation over 10 trials.

of persistence diagrams with both 0-dimensional and 1-dimensional homology, giving 8 sets of features that can be generated for the sets of persistence diagrams for that dataset. Specific to the multi-scale kernel method, the timing reported is for a symmetric feature matrix that is nxn, where n is the number of observations in the dataset. This means the training feature set requires less computation time than a testing feature set of comparable size.

Additionally, during the review of these methods, we did not encounter significant issues with model fitting, hence formal timings were not recorded for this portion of the analysis. Conclusions from these timings are addressed in the discussion section.

Data Availability

The datasets, persistence diagrams (or code to compute the diagrams), and all other associated code for this study can be found in the machine learning methods for persistence diagrams github repository https://github.com/barnesd8/machine_learning_for_persistence.

For each of the five datasets, the following code is available:

- A jupyter notebook that loads and formats the persistence diagrams including images and does a preliminary model fitting on a subset of the data
- A python script that calculates the persistence diagrams from the original dataset - some of these are written using MPI depending on the size of the dataset
- A python script that fits models for random samples of the data to get mean estimates of accuracy for both the training and test dataset

These scripts reference modules included in the github repository, including a persistence methods script that calculates features from persistence diagrams for a training and test dataset. This uses a combination of other available functions and functions written specifically for this thesis.

The **Template Systems** and **Adaptive Template Systems** methods use functions from https://github.com/lucho8908/adaptive_template_systems, which is the corresponding code repository for [46]. The available methods in our extension include Tent Functions and Adaptive Template Functions (GMM, CDER, and HDBScan methods).

The **Adcock-Carlsson Coordinates** method is a function developed specifically for this thesis, and includes the calculation of the 4 different features used in our analysis. The **Persistence Landscape** method uses the persistence landscape calculation from the Gudhi Package [62]. The **Multi-Scale Kernel Method** has two included implementations, one is from [63] and is slower to compute, while the other is a faster implementation that can be used on larger datasets. All of the results in this thesis were reported using the implementation written specifically for this thesis. The **Persistence Images** features are also from [63]. Additionally, many functions from [64] are used throughout.

4.8 Discussion

Adcock-Carlsson Coordinates, Tent Functions, and Persistence Landscapes scale well, and perform well even for large datasets. It is of note though that parameter choice will affect computation time. This was especially notable in the Template Features (Tent Functions) computation time. As the number of tent functions is increased, the time to compute features also increases. We observed a superlinear increase, however, even with this increase computation time was not a barrier for analysis.

Persistence Images and Adaptive Template Functions do not scale or perform as well, however, they do provide good featurizations for accurate models and should be considered depending on the dataset. Specifically, the Adaptive Template Functions was not completed for the full HAM10000 dataset due to computation time.

When using these methods, it should be of note that the Multi-Scale Kernel method is computationally intensive, and does not scale well. Additionally, the accuracy achieved is not better than other methods for the datasets in this thesis.

4.9 Conclusion

This chapter reviews 6 methods for topological feature extraction for use in machine-learning algorithms. Persistence Landscapes, Adcock-Carlsson Coordinates, Template Systems, and Adaptive Template Systems perform consistently with minimal differences between datasets and types of persistence diagrams. These methods are also less expensive in terms of execution time. A main contribution of this chapter is the availability of datasets, persistence diagrams, and code for others to use and contribute to the research community.

CHAPTER 5

OPEN SOURCE SOFTWARE CONTRIBUTIONS

5.1 Open Source Software Development Framework

Topological data analysis is a multi-disciplinary analysis approach, with applications in an extensive number of domains. To enable the use of TDA tools, well-documented and maintained open source software is needed. Multiple software packages are available as part of this dissertation, either developed collaboratively as part of Liz Munch's lab, or as part of my individual contributions to topological data analysis.

Three different open source tools are included in this section, `teaspoon` [2], `ceREEBerus`, and an extension to the kepler mapper [65] software. `teaspoon` is a python package developed for topological signal processing, with modules for specific applications including parameter selection, persistent homology, and machine learning. I have contributed to the machine learning module by optimizing already available feature creation functions and contributing an additional method from prior work. Additionally, I have implemented a software development process for automated releases to pypi for `teaspoon`.

This standardized software development framework was used when creating `ceREEBerus`, a python package for working with Reeb graphs. Users can load Reeb graphs, customize plots, compute merge trees, and perform basic distance calculations using `ceREEBerus`. `ceREEBerus` includes code from collaborators in the Munch lab, and I wrote the base functions and created the overall structure and initial package for `ceREEBerus`.

This section focuses on the available open source software, and begins with the software development process implemented to promote collaboration.

5.1.1 Broader Impact

While much of the work in this section focuses on implementation of already existing methods, a significant motivation for this work is to enable collaboration and multidis-

ciplinary research with topological tools. Each of the sections in this chapter outlines a contribution that heavily contributes to the broader impact of toplogical data analysis.

Through the standardization and automation of the development pipeline of `teaspoon` and `ceREEBerus` student and collaborators from disciplines outside of computer science and software engineering can readily contribute and learn standards along the way. I have also provided training to other graduate students as well as led a discussion-based responsible conduct of research seminar focusing on version control software. Additionally having automation for deployment and testing provides a process where individuals just learning software development can feel more comfortable with the process of contributing to software packages.

The focus of `teaspoon` has been very similar, with the additional endeavor of simplifying the user experience and providing additional examples for individuals wishing to learn and use topological methods for data analysis through `teaspoon`. To this end, I simplified the install process and integrated notebooks, datasets and code from [3] in `teaspoon`. This further simplifies the process for users to get started as persistence diagrams are readily available from familiar datasets for use in featurization methods and subsequently machine learning models.

Finally through my work on `ceREEBerus` I initialized a software package and process for collaborators in the Munch Lab (and more broadly) to be used as they continue to expand on both theory and computation of Reeb graphs. The structure and examples I provided through this initial work set a solid foundation for collaboration of Reeb graph computation.

5.1.2 Automation of development pipeline

A major component of the software development framework was standardizing and automating software releases for any existing packages in the MunchLab, and providing guidelines for future packages for the MunchLab. This provides a framework for packages developed primarily in python, and released via PyPI. This framework helps support a

consistent development practice and allows for maintenance by graduate students in limited terms. The process consists of two main components: 1) Process for merging, testing releases, and approval to the master/main branch and 2) Automation of deployment to PyPI.

To support an automated deployment with multiple contributors, it is imperative to first establish a collaborative process that supports updates from multiple developers that are tested and reviewed. This process was designed and documented, and provided via GitHub and as workshops to graduate students contributing to software development projects. As such, the CMSE value to collaborate across disciplines was supported and enabled. The process is visualized in Figure 5.1, and consists of creating a feature branch from the test release branch, implementing required features, merging into the test release branch with appropriate review, merging into the master/main branch after review and testing.

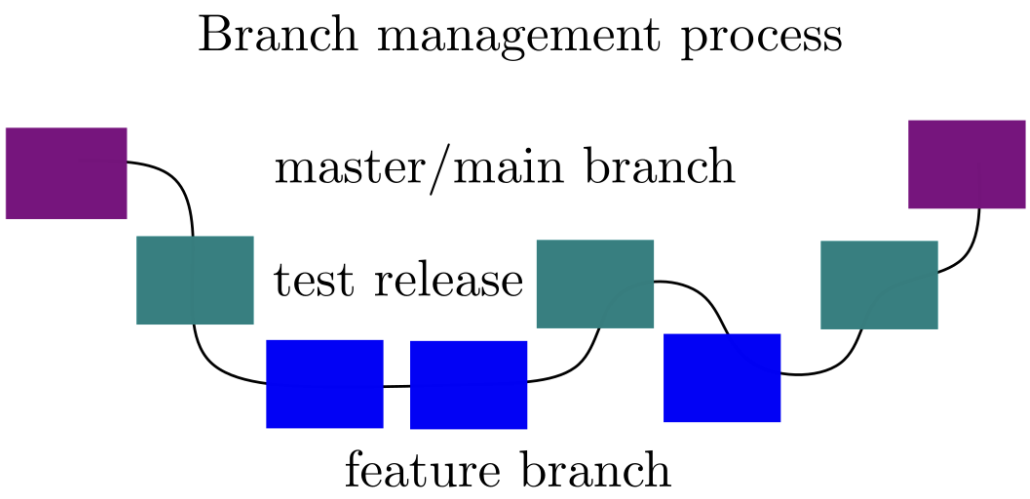


Figure 5.1 Illustration of the collaborative release process implemented.

To further support the development pipeline, workflows via GitHub were configured to allow for automated releases and include criteria that must be met for the automated releases. Upon approval of the release, the following workflow is executed: 1) Create a container (ubuntu), 2) Install python, 3) Install the package, 4) Run unit tests, 5) Build the package, and 6) Publish to PyPI. Workflow monitoring is enabled, and failure at any point

in the workflow will halt the process and will not publish to PyPI. The set of standards for a successful build are provided in further detail.

Must have passed a successful deployment to TestPyPI The next steps are implement into a release process on the test release branch, which upon successful completion deploys to the TestPyPI site. This is the first check required for code to be merged into the master/main branch for release.

Must have passed a code review Before a feature branch can be integrated into a release a code review is required by a project collaborator. The collaborator can approve the changes or suggest changes to further refine the code.

Must pass unit tests Each module has a series of unit tests that are required to be passed before deployment. Once the code is reviewed, and the merge to the main/master branch is accepted, the automated deployment process runs the unit tests.

Must have package dependencies in pyproject.toml For a successful install, build, and deployment, package dependencies must be listed in the pyproject.toml file in the dependencies section.

Must have successful package build Before the version can be automatically deployed it must build successfully.

Must have the version incremented for release When the release is pushed to PyPI the version must be incremented.

5.2 Teaspoon

The information in this section on `teaspoon` follows [2]. `teaspoon` is a comprehensive python package for topological signal processing combining techniques from topological

data analysis to create tools for signal processing that use shape for inference. `teaspoon` is open source with extensive documentation and follows the collaborative development process outlined in Section 5.1.2 making the code both easy to use and easy to contribute.

`teaspoon` is comprised of five main modules: dynamical systems, machine learning, complex networks, information, and parameter selection. I focus on the machine learning module, which historically include the featurization and classification modules, and have been extended to include both a datasets module and notebook examples for machine learning pipelines. Additionally, some functions in the featurization module have been added or updated for computational efficiency. Source code and notebook examples are adapted from [3].

As part of the focus on usability for teaspoon, a considerable number of users continued to have difficulties with package installation due to system dependencies CMake [66] and Boost [67]. Installation issues were limited to those related to zigzag persistence. Currently computation of zigzag persistence has limited implementations, all reliant on underlying C/C++ libraries to ensure reasonable computation times. As such, these `teaspoon` modules have been moved to their own repository, **spork**.

The main goals of my work with `teaspoon` has been to improve ease of use, enable additional collaboration and expand the machine learning modules by including work done in [3] to allow for easy distribution. Since featurization of persistence diagrams is an active area of research, providing example datasets and efficient code for feature computation provides a contribution to the topological data analysis community.

5.2.1 spork

spork is the companion to `teaspoon` and was developed to create a simpler install for the majority of users utilizing the `teaspoon` package. As additional software for TDA has become available, `teaspoon` has expanded to include additional features for method applications, many of which are used for specific purposes. Many users would

not necessarily need all functions when first using `teaspoon`, so to improve the install experience a subset of functions were moved to a companion package, **spork**.

The additional install currently required for **spork**, is Dionysus 2 [68] and includes features required to compute zigzag persistence. The install is available on the Python Package Index (PyPI), and builds locally, requiring a compiler for the C++ to python bindings, as well as other system dependencies.

5.2.2 Featurization

The featurization module in `teaspoon` originally contained six different featurization methods: persistence landscapes, persistence images, Carlsson Coordinates, template functions, kernel method, and signature of paths. As part of my work I have extended the featurization module to include adaptive template systems as an additional feature, and included a faster version of the kernel method and the persistence images method. Initial code for adaptive template systems was provided by Luis Polanco and I have included a subset of these functions.

Each feature type was optimized in a different way specific to the method and required libraries. The two types of optimization were explored during this analysis, **parallelization** and **vectorization**. The discussion on parallelization follows [69]. Parallelization can refer to either **functional parallelism** or **data parallelism**. Functional parallelism is running multiple sets of instructions concurrently, and data parallelism is running the same instructions on different sets of data concurrently. Data parallelism was used to optimize the persistence images code and an illustration of this is in Figure 5.2.

Vectorization can refer to a few things: the process of turning persistence diagrams into features, a type of data parallelism where computation is done simultaneously on elements of a vector, or the process of restructuring code to remove loops and offloading the a lower level language (C) for better performance [70]. When I refer to vectorization, I mean the last definition.

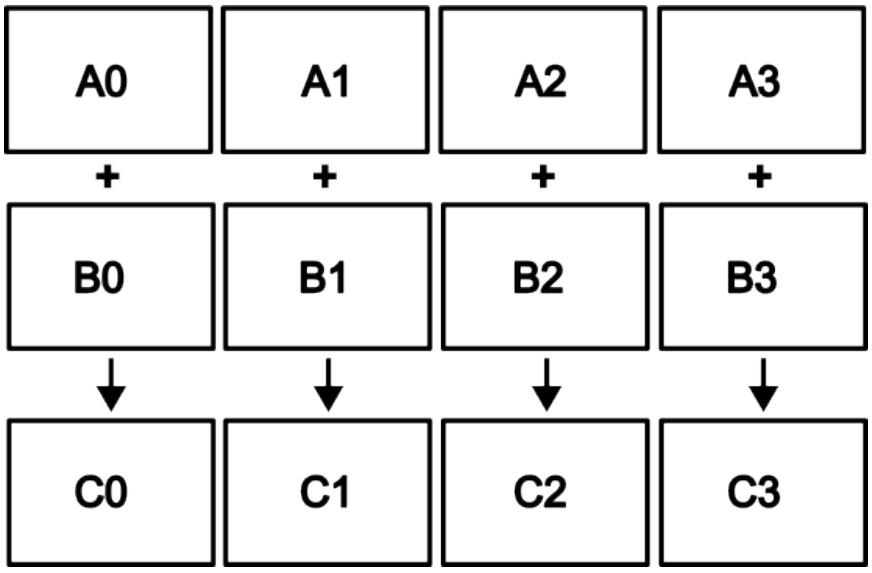


Figure 5.2 Illustration of data parallelism where the addition operation is performed on blocks of a dataset at a time concurrently.

To determine the most appropriate implementation in the `teaspoon` software, a number of methods were developed and timed using a simulated dataset. Development and timing were performed on an iMac using the Apple M1 chip, 16 GB of Ram with 8 total cores. Initial timing was run locally as many users of `teaspoon` will be working on personal computers. Timing was also assessed on Michigan State's High Performance Computing Center.

5.2.3 Kernel Method

The first method discussed is the Multi-scale Heat Kernel Method from [37]. To optimize the kernel method, code was developed and assessed using Numba [59], which translates Python to optimized machine code and compiles upon the first run. Numba provides many different ways to improve performance, including automatic just in time compiling, automatic parallelization, and the creation of NumPy universal functions [70].

Numba's just-in-time compiling (jit) is simply a decorator applied to a Python function that directs Numba to automatically optimize and generate machine code with performance similar to C, C++ and Fortran. This same decorate can also be designated to compile code

to run in parallel. While this is simple to use, initial benchmarking tests showed much better performance using guvectorize.

Guvectorize (and vectorize) is a way to create NumPy universal functions (ufuncs) with Numba, avoiding the traditional process required to create a NumPy ufunc. NumPy ufuncs operate element-by-element and is a wrapper for C functions. Functions can be written using guvectorize in Numba with cpu, parallel, and gpu targets and also broadcast returning different size outputs based on input size. Any loops inside the function do not parallelize, but any additional dimensions will parallelize.

To better understand this mechanism, consider a dataset comprised of persistence diagrams. For each observation in the dataset, there is a set of persistence diagrams. Each diagram is comprised of birth and death pairs. Consider the example below, with 4 sets of length 3 birth and death pairs. From the python array perspective, the shape of this set would be $(4, 3, 2)$. To compute features using the kernel method, each data point is compared to each other. Writing a function then to take a set of persistence diagrams and compare them to a single persistence diagram, so an array of $(4, 3, 2)$ and an array of $(3, 2)$ will then broadcast across the first dimension for the calculation.

```
df = [[[0,1], [0,2], [0,4]],
      [[1,3], [0,4], [0,6]],
      [[2,4], [3,5], [0,7]],
      [[0,1], [2,4], [3,7]]]
```

To allow for core-based parallelization or gpu processing, writing a function using guvectorize to compare an entire array to a single observation will parallelize the additional dimension. In this case, providing two arrays of size $(4, 3, 2)$ will parallelize along the first dimension for one of the arrays using a scatter operation, sending observations of the second array to different processes to run in parallel. The implicit assumption in this example is

that each persistence diagram is the same length, i.e. the second dimension will always be 3. That is not the case, so first restructuring is required. Any timing reported includes the time required to restructure the data.

The code example below shows both the before and after of the restructuring function that fills in the array so create the same second dimension to enable both the creation of a NumPy ufunc using guvectorize in Numba.

```
df = [[[0,1], [0,2]],
      [[1,3], [0,4], [0,6]],
      [[2,4], [3,5], [0,7]],
      [[0,1]]]
df = reshape_to_array(df)
print(df)
[[[0,1], [0,2], [-1,-1]],
[[1,3], [0,4], [0,6]],
[[2,4], [3,5], [0,7]],
[[0,1], [-1,-1],[-1,-1]]]
```

Results of the local run are shown in Figure 5.3 and results of the HPCC run are shown in Figure 5.4. For both note a speed-up that scales well as additional processors are used. For the kernel computation the level of complexity is: $O(N^2|F| \cdot |G|)$ where $|F|$ and $|G|$ are the cardinality of the multisets F and G and N is the number of multisets. In other words, for a dataset with N observations, a computation needs to happen for each set of birth-death pairs for each $n_i, n_j \in N$. Further optimization could be possible through a fast kernel implementation [71], but the implementation would need to be adapted to account for the specific structure of persistence diagrams.

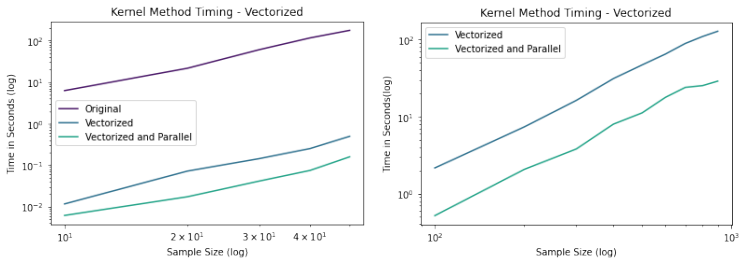


Figure 5.3 Comparison of timing for Multiscale Heat Kernel using a simulated dataset running locally.

Timing for the multiscale kernel method was run on MSU's HPCC with amd20-v100 20 Intel(R) Xeon(R) Platinum 8260 CPU's with clock speed 2.40 GHz. The gpu used was an NVIDIA v100 with 32768 MB memory.

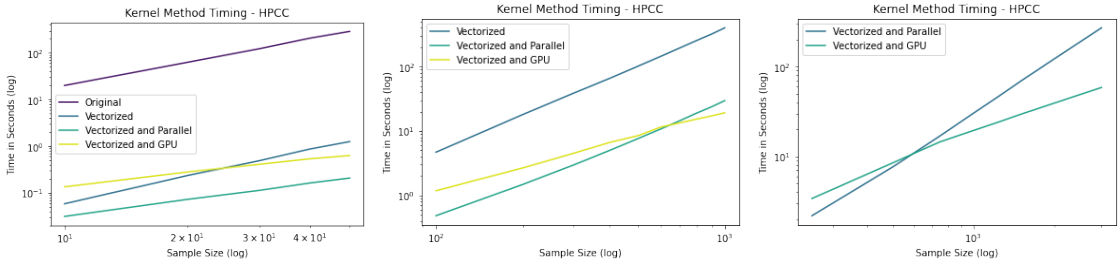


Figure 5.4 Comparison of timing for Multiscale Heat Kernel using a simulated dataset running MSU's HPCC with 16 cores.

5.2.4 Persistence Images

Similar methods were explored for the optimization of persistence images. Ultimately the implementation from the persim [72] package was used for the availability of the parallel option. Due to the need for integration over a grid of a gaussian density function, the error function from the SciPy [73] library is required and not supported the Numba features. At attempt to recreate the integration in NumPy for use with Numba was attempted, but ultimately not as efficient nor accurate. Timing from running locally is shown in Figure 5.5 and timing from using HPCC is shown in Figure 5.6.

Timing run on MSU's HPCC for persistence images was on a intel18-v100 node with Intel(R) Xeon(R) Gold 6148 CPU's with clockspeed of 2.40 GHz using 16 cores.

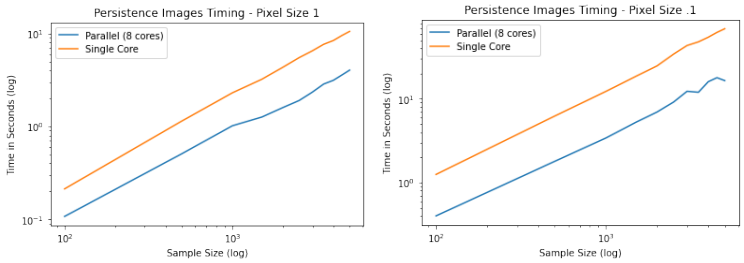


Figure 5.5 Comparison of timing for persistence images using a simulated dataset running locally.

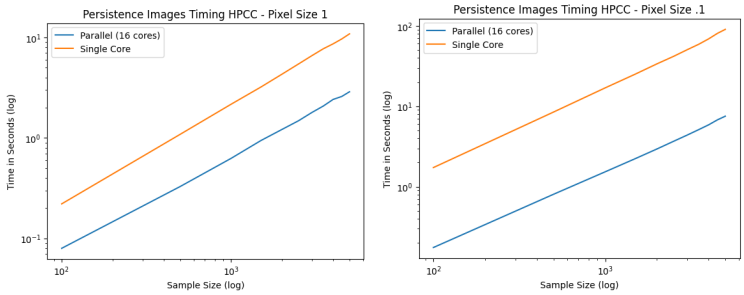


Figure 5.6 Comparison of timing for persistence images using a simulated dataset running on MSU's HPCC.

5.2.5 Datasets

Datasets used in [3] were integrated into `teaspoon` as part of a new module within the machine learning section, *load_datasets*. This module includes the persistence diagrams computed from the MNIST dataset and the MPEG7 Dataset. The following describes how to load each dataset for use.

The **mnist** persistence diagrams can be loaded using the function *load_datasets.mnist()*. No parameters are required to load this function. This dataset includes the persistence diagrams for the 60,000 observations considered the training set for the MNIST dataset. The dataset is loaded as a pandas dataframe, and has 9 columns, 8 which correspond to persistence diagrams and one column of dataset labels. The 8 columns of persistence diagrams can be defined as:

1. **zero_dim_rtl**: persistence diagram for 0 dimensional features computed using the right to left directional transform

2. **zero_dim_ltr**: persistence diagram for 0 dimensional features computed using the left to right directional transform

3. **zero_dim_btt**: persistence diagram for 0 dimensional features computed using the bottom to top directional transform

4. **zero_dim_ttb**: persistence diagram for 0 dimensional features computed using the top to bottom directional transform

5. **one_dim_rtl**: persistence diagram for 1 dimensional features computed using the right to left directional transform

6. **one_dim_ltr**: persistence diagram for 1 dimensional features computed using the left to right directional transform

7. **one_dim_btt**: persistence diagram for 1 dimensional features computed using the bottom to top directional transform

8. **one_dim_ttb**: persistence diagram for 1 dimensional features computed using the top to bottom directional transform

Each of the persistence diagrams was computed first using a directional transform on pixel values, and then sublevel set persistence. This is demonstrated in Figure 5.7 with additional details available in [3] for mnist, and all other datasets listed here.

The **mpeg** persistence diagrams can be loaded using the function *load_datasets.mpeg7()*. No parameters are required to load this function. This dataset includes the persistence diagrams for the 1400 observations in the mpeg7 dataset. The dataset is loaded as a pandas dataframe, and has 5 columns, 4 which correspond to persistence diagrams or related computation and one column of dataset labels. The columns of the mpeg7 persistence diagrams dataset can be defined as:

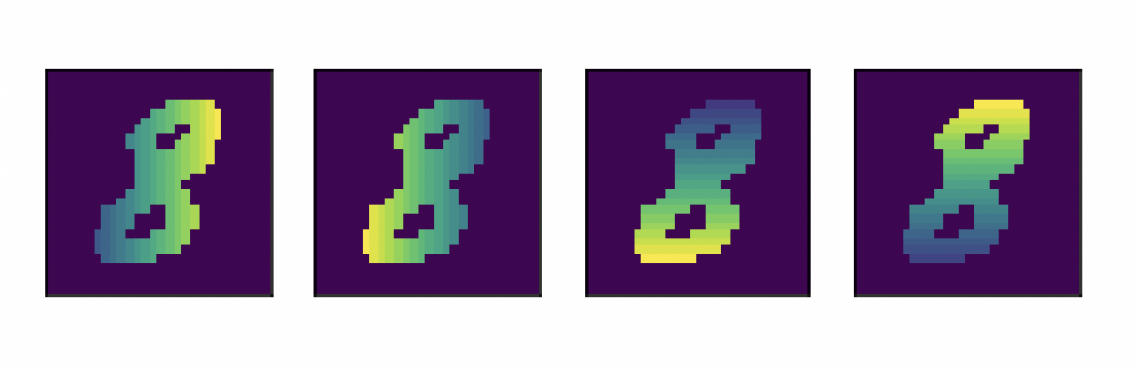


Figure 5.7 The number 8 after computing each of the 4 types of coordinate transforms to compute the persistence diagrams.

1. **Name**: label of dataset
2. **Outline**: outline of the image represented as a 2-dimensional point cloud dataset
3. **DistCurve**: distance from center of outline
4. **persistence_outline**: persistence diagrams for 0 dimensional features and 1 dimensional features computed from the outline (contour)
5. **lower_star_persistence**: persistence diagram for 0 dimensional features computed using sublevel set persistence of the distance curve

Persistence diagrams were computed using both the shape outline and distance curve. Ripser was used to compute the 0-dimensional and 1-dimensional features from the outline as well as the 0-dimension sublevel set persistence from the distance curve. The available data for the outline and distance curve is plotted in Figure 5.8.

The shape retrieval of non-rigid 3D Human Models, **SHREC14**, persistence diagrams can be loaded using the function *load_datasets.shrec14()*. No parameters are required to load this function. This dataset includes the persistence diagrams for the 3000 observations in the SHREC14 dataset. The dataset is loaded as a pandas dataframe, and has 6 columns, 3 which correspond to persistence diagrams or related computation and one column of dataset

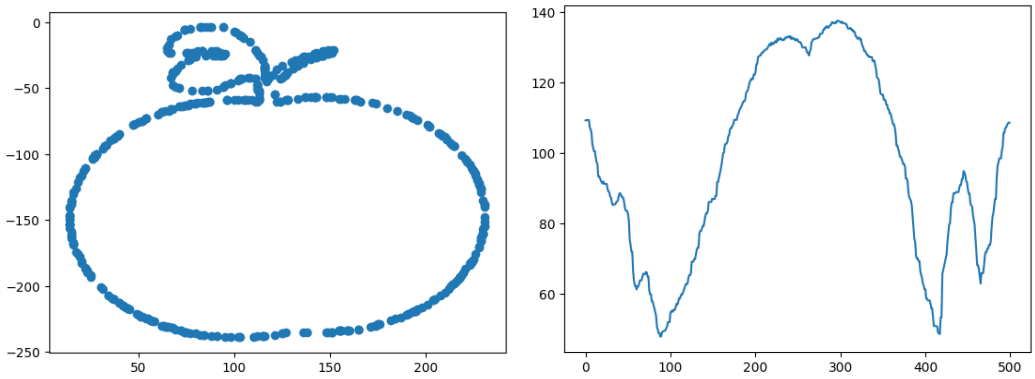


Figure 5.8 Example output from mpeg7 dataset for columns Outline and DistCurve.

labels. Persistence diagrams were computed using the Heat Kernel Signature with a range of parameter values (corresponding to trial) [74] on 300 images, each corresponding to a specific trial value in the dataset. The columns of this persistence diagrams dataset can be defined as:

1. **freq**: index of machine learning problem (300 observations per problem)
2. **trial**: corresponds to a machine learning problem (10 total, corresponds to a parameter used in computing the persistence diagram)
3. **CollectedDgm**: set of all persistence diagrams
4. **Dgm1**: persistence diagrams for 1 dimensional features
5. **Dgm0**: persistence diagram for 0 dimensional features
6. **TrainingLabel**: One of 15 labels corresponding to five different poses for the three classes of male, female, and child figures

5.3 CeREEBerus

Another area of focus for open source software development for topological data analysis is a packaged called `ceREEBerus`, which focuses on computation related to Reeb graphs. Working with Reeb graphs is an active area of research, with provably correct and efficient

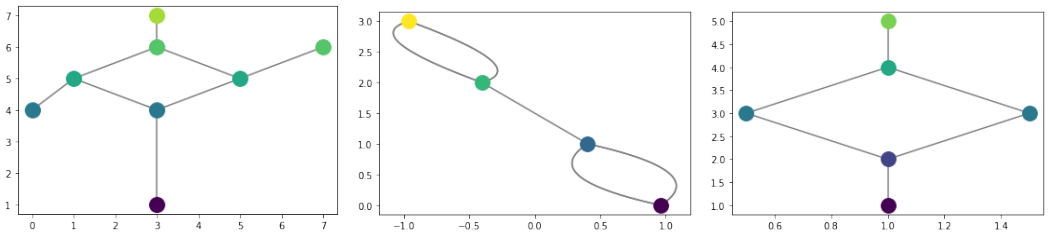
algorithms for computing Reeb graphs available [18] in recent years. This provides a renewed interest in the use of Reeb graphs, along with the availability of metrics defined on Reeb graphs [16]. The goal then of `ceREEBerus` is to start with Reeb graphs and basic functions on Reeb graphs and build a comprehensive package to compute a wide variety of metrics on Reeb graphs.

The initial and current utility of `ceREEBerus` allows users to load example Reeb graphs, generate random merge trees, plot graphs and merge trees, generate merge trees from Reeb graphs, and compute distances between merge trees. This package is available via PyPI and follows the open source software framework implemented for the Munch Lab.

5.3.1 Available functions

As part of the package, extensive user documentation is available at the `ceREEBerus` site, and a summary is included here. The `ceREEBerus` package is comprised of three modules: example data and graphs (*data*), Reeb graphs (*reeb*), and compute (*compute*).

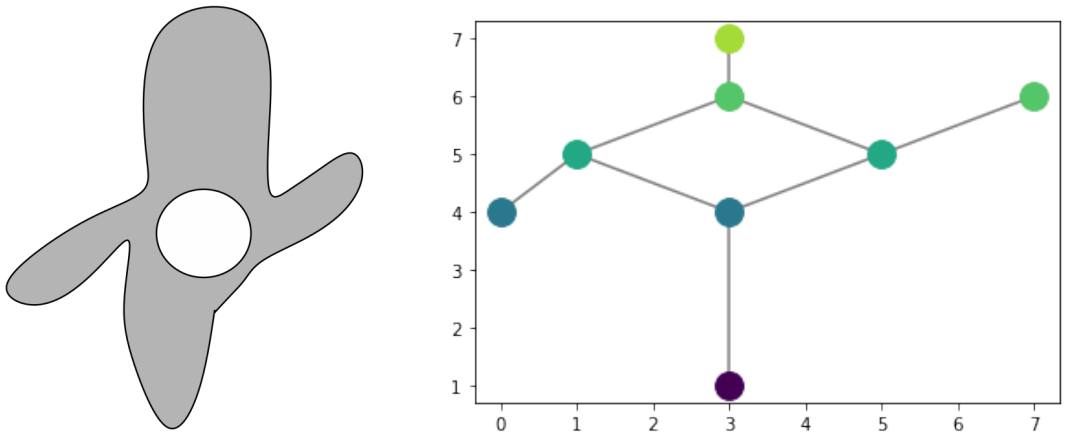
The sample data and graphs module provides both networkx and Reeb graphs for multiple example graphs including the dancing man, simple loops, and torus graphs. Each of these is plotted in Figure 5.9 using the plotting function available in `ceREEBerus`. An example Reeb graph is in Figure 5.10a and corresponding `ceREEBerus` plot is in Figure 5.10b.



(a) Dancing Man. (b) Simple Loops. (c) Torus.

Figure 5.9 Output from `ceREEBerus` plotting capabilities of dancing man, simple loops, and torus.

There are two main ways to load the example graphs, either as a networkx graph or a



(a) A Reeb graph of dancing man.

(b) Output from `ceREEBerus` - a graph of dancing man.

Figure 5.10 Cereeberus output from on the right of a Reeb graph on the left.

Reeb graph, which is a custom class created for `ceREEBerus`. These can be loaded and accessed by using *from cereeberus.data import graphs* for the networkx version, or *from cereeberus.data import reeb* for the Reeb graph version. An example code block for loading the relevant libraries, loading the Reeb graph version of the torus, and plotting is shown below. Relevant output for this code is shown in Figure 5.9c.

```
from cereeberus.data import reeb
reeb_torus = reeb.torus()
reeb_torus.plot_reeb()
```

When plotting Reeb graph objects, the use of additional parameters allows you to control position and display of the plots. The available parameters with listed default values and definitions are:

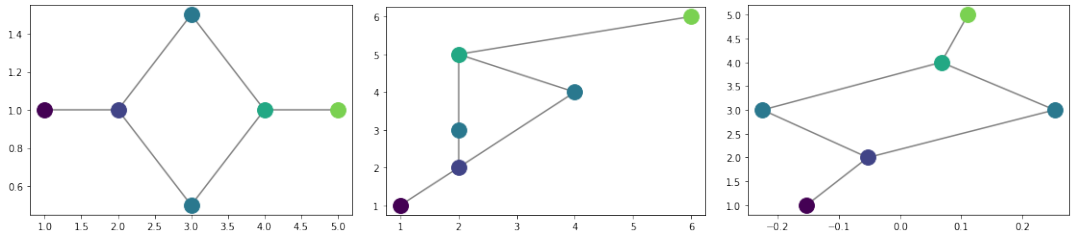
- position = {}: an optional argument to update positions of nodes for plotting
- resetSpring = False: a boolean flag that when true assigns new node positions for the underlying Reeb graph using the spring layout from networkx

- horizontalDrawing = False: a boolean flag that when true changes the x and y axis for plotting and does not update node position
- verbose = False: a boolean flag that when true provides logging output
- cpx = .1: float that is only used for graphs where the degree of connectivity between two nodes is two and controls the radius of the curve used for plotting relevant to the x position
- cpy = .1: float that is only used for graphs where the degree of connectivity between two nodes is two and controls the radius of the curve used for plotting relevant to the y position

An example of horizontal drawing, updating position, and reset spring are shown in the following code block, with example output in Figure 5.11.

```
### Import relevant module
from cereeberus.data import reeb
reeb_torus = reeb.torus()
### Plotting the Reeb graph using horizontalDrawing = True
reeb_torus.plot_reeb(horizontalDrawing=True)
### Plotting the Reeb graph with an updated position
reeb_torus.plot_reeb(position = {0:(1,1), 1:(2,2), 2:(2,3),
3:(4,4), 4:(2,5), 5:(6,6)}, horizontalDrawing = False,
verbose = False)
### Plotting the Reeb graph with the resetSpring parameter = True
reeb_torus.plot_reeb(resetSpring = True)
```

Lastly as part of the plotting function, the ability to refine how multiple lines between two nodes is plotted is shown below with the corresponding output in Figure 5.12.



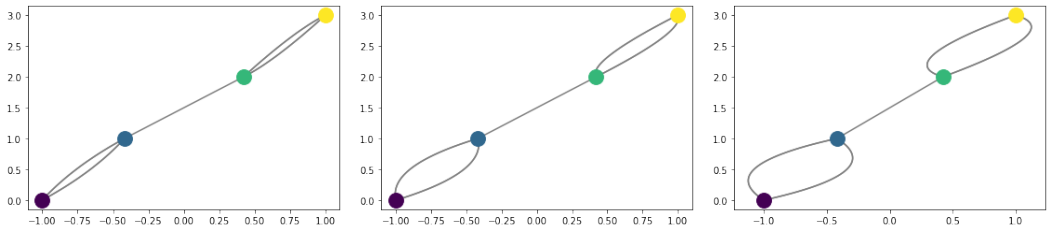
(a) Example torus plotting horizontally.

(b) Example torus with updated positions.

(c) Example torus using spring layout.

Figure 5.11 Output from `ceREEBerus` plotting capabilities of torus using different parameters.

```
### Import relevant module

from cereeberus.data import reeb

sl = reeb.simple_loops()

### Plotting the simple loops graph with cpx = .1

sl.plot_reeb(cpx=.1)

### Plotting the simple loops graph with cpx = .5

sl.plot_reeb(cpx=.5)

### Plotting the simple loops graph with cpx = 1

sl.plot_reeb(cpx=1)
```



(a) Example simple loops with cpx = .1.

(b) Example simple loops with cpx = .5.

(c) Example simple loops with cpx = 1.

Figure 5.12 Output from `ceREEBerus` plotting capabilities of simple loops using different cpx parameters.

Another capability in the compute.degree module is adding nodes, removing isolates, and removing any unnecessary nodes to create a minimal Reeb graph. Recalling the

definition from Section 2.4, the nodes with both an up degree and down degree equal to 1 can be removed to compute the minimal Reeb graph. I use the following algorithm to construct the minimal Reeb graph:

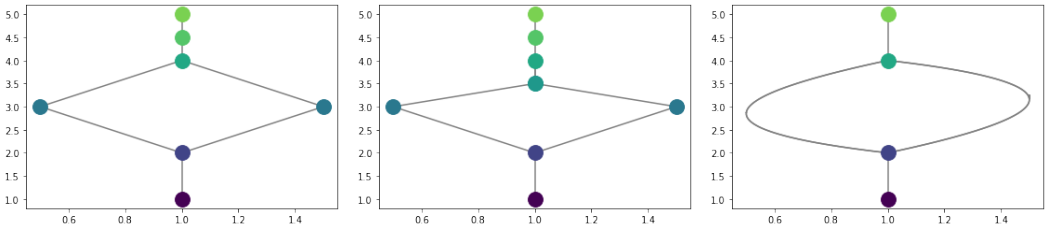
Require: R, a Reeb graph

```
n = length(nodes)
if i ≤ n then
    if up degree(R) == down degree(R) == 1 then
        add edge(up node, down node)
        remove edge(up node, i)
        remove edge(down node, i)
        remove node(i)
    end if
end if
```

The following code example shows first adding nodes to the torus Reeb graph, and then constructing the minimal Reeb graph of the torus. Plots of each are available in Figure 5.13.

```
import cereeberus.compute.degree as degree
import networkx as nx
reeb_torus = reeb.torus()
### Add node and plot
reeb_torus_add = degree.add_nodes(reeb_torus, fx=4.5, x = 1)
reeb_torus_add.plot_reeb()
### Add another node and plot
reeb_torus_add = degree.add_nodes(reeb_torus_add, fx=3.5, x=1)
reeb_torus_add.plot_reeb()
### Compute minimal Reeb graph
```

```
reeb_torus_min = degree.minimal_reeb(reeb_torus_add)
reeb_torus_min.plot_reeb(cpx=1)
```



(a) Torus Reeb graph after adding one node.

(b) Torus Reeb graph after adding two nodes.

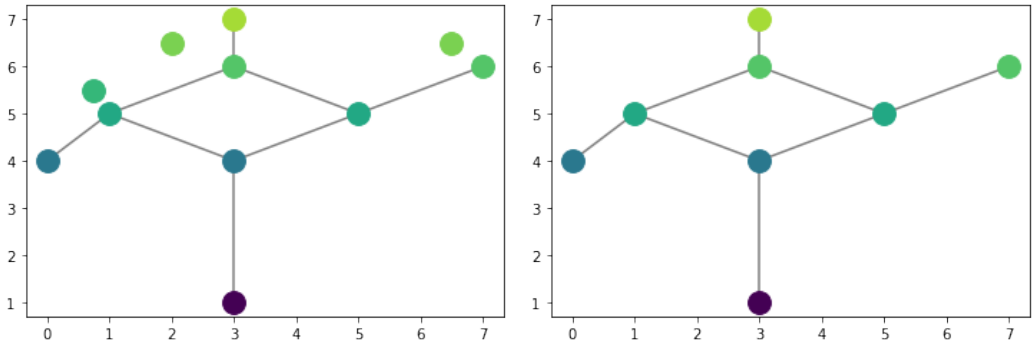
(c) Minimal Reeb graph of torus.

Figure 5.13 Output from `ceREEBerus` of degree functions.

As part of the compute.degree module one can compute distances between graphs and remove isolates, or nodes with up degree and down degree equal to zero. The remove isolates function is a necessary step in the union find algorithm implemented for computing merge trees. Code is shown below and images are available in Figure 5.14. Interested readers are directed to the `ceREEBerus` documentation for details on distance as graph edit distance and directed merge tree distance are both available.

```
from cereeberus.data import reeb
import cereeberus.compute.degree as degree
jm = reeb.juggling_man()
jm.plot_reeb()
### Remove isolates and plot
jm_im = degree.remove_isolates(jm)
jm_im.plot_reeb()
```

The final and core functionality of `ceREEBerus` is the ability to compute merge trees from Reeb graphs. The main code was provided by Elena Wang, and I structured and implemented it into the overall python package. The *computeMergeTree* function uses an



(a) Reeb graph with isolates. (b) Removed isolates.

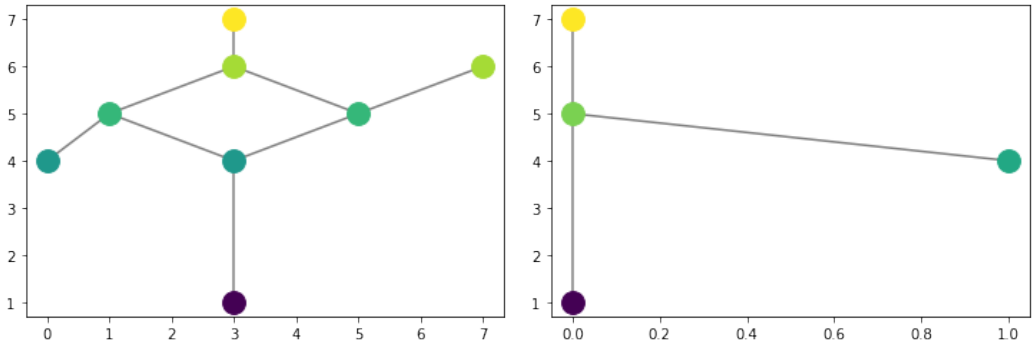
Figure 5.14 Plots from Reeb graphs before and after removing isolates.

implementation of the union find algorithm. Example code and the associated output are below and shown in Figure 5.15.

```
from cereeberus.data import reeb
import cereeberus.compute.degree as degree
jm = reeb.juggling_man()
### Remove isolates and plot
jm_im = degree.remove_isolates(jm)
jm_im.plot_reeb()
### Import merge tree, compute, and plot
from cereeberus.compute.merge import computeMergeTree
cmt = computeMergeTree(jm_im, verbose=False, precision=1, filter=False)
cmt.plot_reeb(position = {7:(0,1), 5:(1,4), 2:(0,5), 'inf':(0,7)})
```

5.3.2 Predictive Mapper

Through a seed grant from Center for Business and Social Analytics at Michigan State University, I was able to work with a software developer, Doug Krum, to take a prototype for an interactive Mapper application built by Pat Bills and transition the work to python, taking advantage of the full suite of functions in scikitlearn [64], including parallelism. We started



(a) Dancing man before merge tree. (b) Merge tree of dancing man.

Figure 5.15 Plot of Reeb graph and associated merge tree.

with the mapper implementation from KeplerMapper [27] and [65], and expanded the package to include additional features that allowed the development of predictive mapper. This variant included a web-interface to allow users to interactively create a mapper graph and choose specific parameters for the mapper graph creation. During the development of predictive mapper this tool was used, and additional functions were created to enable the predictive pipeline.

Based on the final predictive mapper pipeline, the associated release is still a work in progress, requiring additional code and data restructuring needed to support integration into the already existing Kepler mapper. As mentioned in Section 3.4, example notebooks for the work done on the iris dataset are available along with the mapper graphs generated. Similarly an example notebook used to create images for the predictive mapper portion of this dissertation is available and is an independent mapper pipeline.

5.3.3 Conclusion

As part of my work in the Computational Mathematics, Science, and Engineering department I have contributed to many computational projects supporting work in topological data analysis. These contributions include a standardized process and framework for collaboration in the Munch Lab, the inclusion of additional and optimized methods in `teaspoon`, and the reorganization of `teaspoon` to have a companion package spork. I have addition-

ally used this framework for the creation of `ceREEBerus`. As part of a review paper I have additionally contributed a set of notebooks and datasets for machine learning use of persistence diagrams.

CHAPTER 6

CONCLUSION

Topological data analysis is a powerful tool for extracting the shape of data. Two primary tools, persistent homology and mapper, require additional transformations to be used directly in machine learning models. In this dissertation, I have contributed to the discipline of TDA by developing another method for feature creation and providing additional computational tools for working with persistence diagrams. I have also contributed to the available open source software in TDA through package development and the implementation of standards for the MunchLab.

To extend the use of mapper, I have developed predictive mapper as a way to create an optimal version of a mapper graph, compute probabilities of inclusion of data points in mapper nodes, and then using the eigenvectors of the graph Laplacian of the mapper graph, represent each data point as a linear combination of the probability of inclusion in each node and the eigenvector values. This featurization provides real-valued vectors for use in machine learning models that represent the connectivity of the mapper graph weighted by a meaningful probability for each data point.

In the field of persistent homology, I have contributed additional datasets consisting of persistence diagrams for some popular datasets, along with the inclusion of a far more computationally efficient function to compute the Multi-Scale Heat Kernel [37]. Example notebooks of 6 different featurization methods are available for academic use, furthering the availability of TDA resources.

I have also contributed meaningfully to available tools for researchers in TDA, with the creation of `ceREEBerus` to work with Reeb graphs, additional contributions to `teaspoon`, and the additional code required for the predictive mapper pipeline. This code is available as open source with a standardized development process to enable collaboration.

Through this dissertation I have contributed both to the field of Topological Data

Analysis, and under taken a combination of theoretical and applied work in the spirit of the CMSE program.

BIBLIOGRAPHY

[1] Gurjeet Singh, Facundo Memoli, and Gunnar Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *Eurographics Symposium on Point-Based Graphics*, 2007.

[2] Audun D Myers, Melih Yesilli, Sarah Tymochko, Firas Khasawneh, and Elizabeth Munch. Teaspoon: A comprehensive python package for topological signal processing. In *NeurIPS 2020 Workshop on Topological Data Analysis and Beyond*, 2020.

[3] Danielle Barnes, Luis Polanco, and Jose A. Perea. A comparative study of machine learning methods for persistence diagrams. *Frontiers in Artificial Intelligence*, 4:91, 2021.

[4] Y. LeCun and C. Cortes. The mnist database, 1999.

[5] Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*, volume 69. American Mathematical Soc., 2010.

[6] Tamal Krishna Day and Yusu Wang. *Computational Topology for Data Analysis*. Cambridge Univ. Press, Cambridge, 2006-2021.

[7] Gunnar E. Carlsson and Vin de Silva. Zigzag persistence. *CoRR*, abs/0812.0197, 2008.

[8] J. H. Friedman and J. W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C(23):881=890, September 1974.

[9] Frédéric Chazal and Bertrand Michel. An introduction to topological data analysis: Fundamental and practical aspects for data scientists. *Frontiers in Artificial Intelligence*, 4:108, 2021.

[10] Monica Nicolau, Arnold J. Levin, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences of the Unites States of America*, pages 7265–7270, 2011.

[11] P.Y. Lum, G. Singh, A. Lehman, T. Ishkanov, M. Vejdemo-Johansson, M. Alagappan, J. Carlsson, and G. Carlsson. Extracting insights from the shape of complex data using topology. *Scientific Reports*, 3:1236, 2013.

[12] Elizabeth Munch and Bei Wang. Convergence between categorical representations of reeb space and mapper. *32nd International Symposium on Computational Geometry*,

53:53:1–53:15, 2016.

[13] Stephen C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.

[14] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[15] Silvia Biasotti, Daniela Giorgi, Michela Spagnuolo, and Bianca Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392:5–22, 02 2008.

[16] Brian Bollen, Erin W. Chambers, Joshua A. Levine, and Elizabeth Munch. Reeb graph metrics from the ground up. *CoRR*, abs/2110.05631, 2021.

[17] Ulrich Bauer, Elizabeth Munch, and Yusu Wang. Strong Equivalence of the Interleaving and Functional Distortion Metrics for Reeb Graphs. In Lars Arge and János Pach, editors, *31st International Symposium on Computational Geometry (SoCG 2015)*, volume 34 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 461–475, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[18] Harish Doraiswamy and Vijay Natarajan. Efficient algorithms for computing reeb graphs. *Computational Geometry*, 42(6):606–616, 2009.

[19] Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4), 2007.

[20] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering, 2001.

[21] Enrique Alvarado, Robin Belton, Emily Fischer, Kang-Ju Lee, Sourabh Palande, Sarah Percival, and Emilie Purvine. g-mapper: Learning a cover in the mapper construction, 2024.

[22] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[23] R. A. Fisher. Iris. UCI Machine Learning Repository, 1988. DOI: https://doi.org/10.24432/C56C76.

[24] Paulo Cortez. Student Performance. UCI Machine Learning Repository, 2014. DOI:

https://doi.org/10.24432/C5TG7T.

[25] S.J. Roberts, D. Husmeier, I. Rezek, and W. Penny. Bayesian approaches to gaussian mixture modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(11):1133–1142, 1998.

[26] P. Cortez and A. M. Gonçalves Silva. Using data mining to predict secondary school student performance. 2008.

[27] Hendrik Jacob van Veen, Nathaniel Saul, David Eargle, and Sam W. Mangham. Kepler mapper: A flexible python implementation of the mapper algorithm. *Journal of Open Source Software*, 4(42):1315, 2019.

[28] Joshua L. Mike and Jose A. Perea. Geometric data analysis across scales via laplacian eigenvector cascading, 2019.

[29] Yuriy Mileyko, Sayan Mukherjee, and John Harer. Probability measures on the space of persistence diagrams. *Inverse Problems*, 27(12):124007, 2011.

[30] David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & computational geometry*, 37(1):103–120, 2007.

[31] Frédéric Chazal, Vin De Silva, and Steve Oudot. Persistence stability for geometric complexes. *Geometriae Dedicata*, 173(1):193–214, 2014.

[32] Mikhail Gromov. *Metric structures for Riemannian and non-Riemannian spaces*. Creative Commons, 2007.

[33] Katharine Turner, Yuriy Mileyko, Sayan Mukherjee, and John Harer. Fréchet means for distributions of persistence diagrams. *Discrete & Computational Geometry*, 52(1):44–70, 2014.

[34] Atish Mitra and Žiga Virk. The space of persistence diagrams on n points coarsely embeds into hilbert space. *Proceedings of the American Mathematical Society*, 149(6):2693–2703, 2021.

[35] Alexander Wagner. Nonembeddability of persistence diagrams with $p > 2$ wasserstein metric. *arXiv preprint arXiv:1910.13935*, 2019.

[36] Peter Bubenik and Alexander Wagner. Embeddings of persistence diagrams into hilbert spaces. *Journal of Applied and Computational Topology*, 4(3):339–351, 2020.

[37] Jan Reininghaus, Stefan Huber, Ulrich Bauer, and Roland Kwitt. A stable multi-scale

kernel for topological machine learning. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4741–4748, 2015.

[38] Peter Bubenik. The persistence landscape and some of its properties. *Topological Data Analysis*, 15:77–102, 2020.

[39] Henry Adams, Sofya Chepushtanova, Tegan Emerson, Eric Hanson, Michael Kirby, Francis Motta, Rachel Neville, Chris Peterson, Patrick Shipman, and Lori Ziegelmeier. Persistence images: A stable vector representation of persistent homology, 2015.

[40] Aaron Adcock, Erik Carlsson, and Gunnar Carlsson. The ring of algebraic functions on persistence bar codes. *Homology, Homotopy and Applications*, 18(1):381–402, 2016.

[41] Luigi Caputi, Anna Pidnebesna, and Jaroslav Hlinka. Promises and pitfalls of topological data analysis for brain connectivity analysis. *NeuroImage*, 238:118245, 2021.

[42] Firas A. Khasawneh, Elizabeth Munch, and Jose A. Perea. Chatter classification in turning using machine learning and topological data analysis∗∗this material is based upon work supported by the national science foundation under grant nos. cmmi-1759823 and dms-1759824 with pi fak, and cmmi-1800466 and dms-1800446 with pi em. jap acknowledges the support of the nsf under grant dms-1622301 and darpa under grant hr0011-16-2-003. *IFAC-PapersOnLine*, 51(14):195–200, 2018. 14th IFAC Workshop on Time Delay Systems TDS 2018.

[43] Sara Kališnik. Tropical coordinates on the space of persistence barcodes. *Foundations of Computational Mathematics*, 19:101–129, 2019.

[44] Sarah Tymochko, Elizabeth Munch, and Firas A. Khasawneh. Adaptive partitioning for template functions on persistence diagrams. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. IEEE, dec 2019.

[45] J. A. Perea, A. Munch, and F. A. Khasawneh. Approximating continuous functions on persistence diagrams using template functions. *CoRR*, abs/1902.07190, 2019.

[46] Luis Polanco and J. A. Perea. Adaptive template systems: Data-driven feature selection for learning with persistence diagrams. *IEEE ICMLA*, 2019.

[47] Sarah Tymochko, Elizabeth Munch, and Firas A. Khasawneh. Adaptive partitioning for template functions on persistence diagrams. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1227–1234, 2019.

[48] Douglas Reynolds. *Gaussian Mixture Models*, pages 659–663. Springer US, Boston, MA, 2009.

[49] Abraham Smith, Paul Bendich, John Harer, and Jay Hineman. Supervised learning of labeled pointcloud differences via cover-tree entropy reduction. *CoRR*, abs/1702.07959, 2017.

[50] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001.

[51] D. Pickup, X. Sun, P. L. Rosin, R. R. Martin, Z. Cheng, Z. Lian, M. Aono, A. Ben Hamza, A. Bronstein, M. Bronstein, S. Bu, U. Castellani, S. Cheng, V. Garro, A. Giachetti, A. Godil, J. Han, H. Johan, L. Lai, B. Li, C. Li, H. Li, R. Litman, X. Liu, Z. Liu, Y. Lu, A. Tatsuma, and J. Ye. SHREC'14 track: Shape retrieval of non-rigid 3d human models. In *Proceedings of the 7th Eurographics workshop on 3D Object Retrieval*, EG 3DOR'14, pages 1–10. Eurographics Association, 2014.

[52] Ulrich Bauer, Michael Kerber, and Jan Reininghaus. Distributed computation of persistent homology, 2013.

[53] Paolo Sonego, Mircea Pacurar, Somdutta Dhir, Attila Kertész-Farkas, András Kocsor, Zoltán Gáspári, Jack A. M. Leunissen, and Sándor Pongor. A protein classification benchmark collection for machine learning. *Nucleic Acids Research*, 35:D232 – D236, 2007.

[54] wwPDB consortium. Protein Data Bank: the single global archive for 3D macromolecular structure data. *Nucleic Acids Research*, 47(D1):D520–D528, 10 2018.

[55] Christopher Tralie, Nathaniel Saul, and Rann Bar-On. Ripser.py: A lean persistent homology library for python. *The Journal of Open Source Software*, 3(29):925, Sep 2018.

[56] Xiang Bai, Xingwei Yang, Longin Jan Latecki, Wenyu Liu, and Zhuowen Tu. Learning context sensitive shape similarity by graph transduction. *IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)*, 2009.

[57] Philipp Tschandl, Cliff Rosendahl, and Harald Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Sci Data*, 5(180161), 2018.

[58] Y. Chung, C. Hu, A. Lawson, and C. Smyth. Topological approaches to skin disease image analysis. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 100–105, 2018.

[59] Antoine Pitrou Siu Kwan Lam and Stanley Seibert. Numba: a llvm-based python jit compiler. In *LLVM '15: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, Nov 2015.

[60] Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of Machine Learning Research*, (12):2211–2268, 2011.

[61] L. Polanco and J. A Perea. Coordinatizing data with lens spaces and persistent cohomology. *Proceedings of the 31 st Canadian Conference on Computational Geometry (CCCG)*, pages 49–57, 2019.

[62] Pawel Dlotko. Persistence representations. In *GUDHI User and Reference Manual*. GUDHI Editorial Board, 3.4.1 edition, 2021.

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

[64] 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.

[65] Hendrik Jacob van Veen, Nathaniel Saul, David Eargle, and Sam W. Mangham. Kepler Mapper: A flexible Python implementation of the Mapper algorithm, October 2020.

[66] Bill Hoffman. Cmake.

[67] Rene Rivera Beman Dawes, David Abrahams. Boost.

[68] Dmitry Morozov. Dionysus 2, 2023. https://www.mrzv.org/software/dionysus2/.

[69] Edmond Chow Victor Eijkhout and Robert van de Geijn. *Introduction to High Performance Scientific Computing*. Springer Science & Business Media, 2020.

[70] 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.

[71] John Paul Ryan, Sebastian Ament, Carla P. Gomes, and Anil Damle. The fast kernel

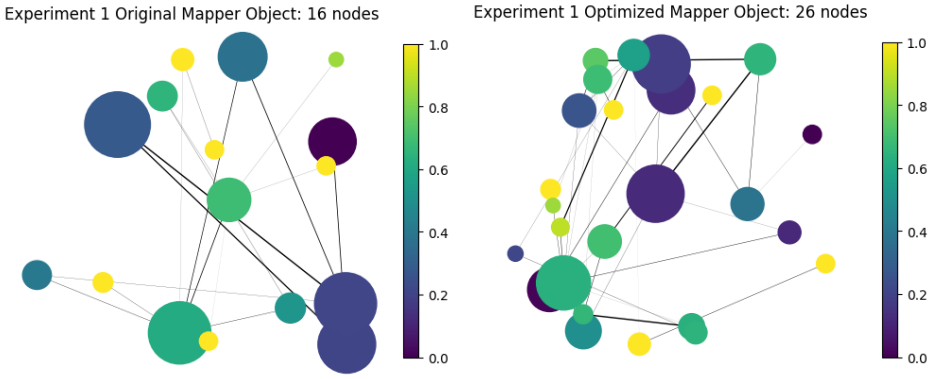
transform. *CoRR*, abs/2106.04487, 2021.

[72] Nathaniel Saul. Persim 0.3.6: a scikit-tda project, 2019.

[73] 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.

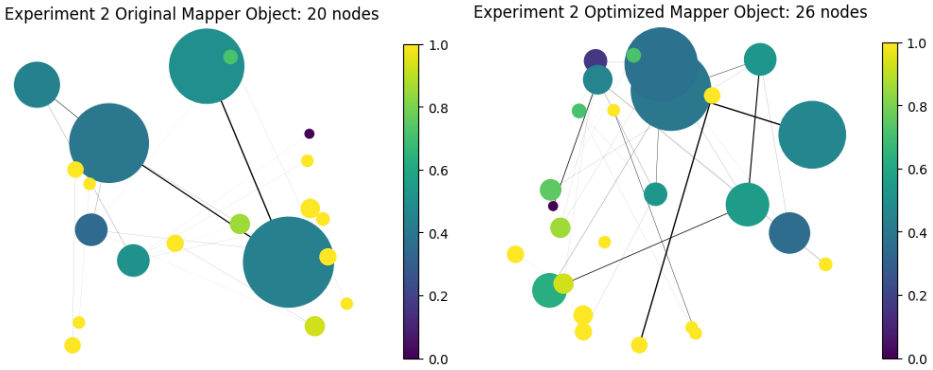
[74] Leonidas Guibas Jian Sun, Maks Ovsjanikov. A concise and provably informative multi-scale signature based on heat diffusion. *Eurographics Symposium on Geometry Processing*, 28(5), 2009.

APPENDIX



(a) Original mapper graph (b) Optimized mapper graph

Figure A.1 Mapper graphs for the grades dataset for Experiment 1 before and after optimization. For this example, the typical optimization work flow was not successful.

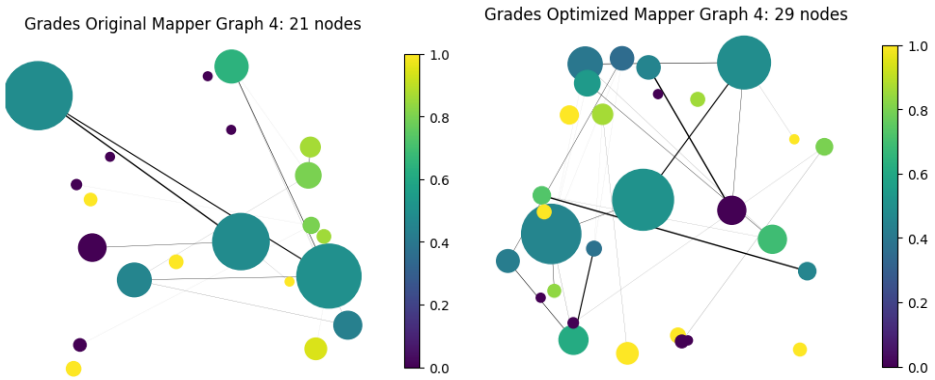


(a) Original mapper graph (b) Optimized mapper graph

Figure A.2 Mapper graphs for the grades dataset for Experiment 2 before and after optimization. For this example, the typical optimization work flow was not successful.

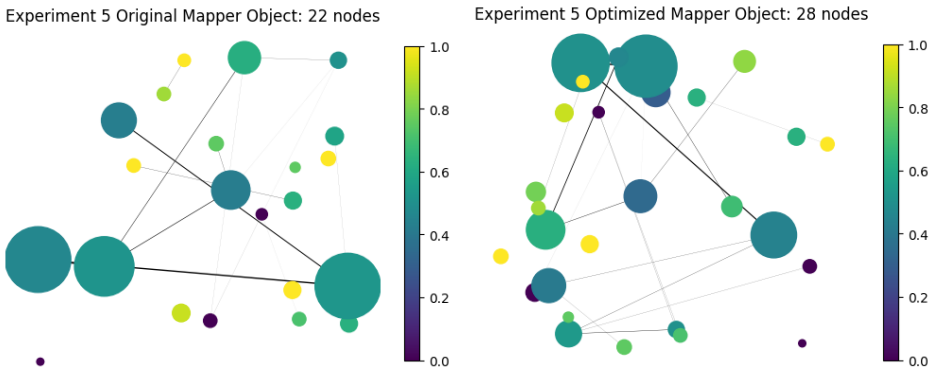
Attribute	Description (Domain)
sex	student's sex (binary: female or male)
age	student's age (numeric: from 15 to 22)
school	student's school (binary: Gabriel Pereira or Mousinho da Silveira)
address	student's home address type (binary: urban or rural)
Pstatus	parent's cohabitation status (binary: living together or apart)
Medu	mother's education (numeric: from 0 to 4)
Mjob	mother's job (nominal)
Fedu	father's education (numeric: from 0 to 4)
Fjob	father's job (nominal)
guardian	student's guardian (nominal: mother, father or other)
famsize	family size (binary: ≤ 3 or > 3)
famrel	quality of family relationships (numeric: from 1 – very bad to 5 – excellent)
reason	reason to choose this school (nominal: close to home, school reputation, course preference or other)
traveltime	home to school travel time (numeric: 1 – < 15min, 2 – 15 to 30 min, 3 – 30min to 1 hour or 4 – > 1 hour)
studytime	weekly study time (numeric: 1 – < 2 hours, 2 – 2 to 5 hours, 3 – 5 to 10 hours or 4 – > 10 hours)
failures	number of past class failures (numeric: n if $1 \leq n < 3$, else 4)
schoolsup	extra educational school support (binary: yes or no)
famsup	family educational support (binary: yes or no)
activities	extra-curricular activities (binary: yes or no)
paidclass	extra paid classes (binary: yes or no)
internet	Internet access at home (binary: yes or no)
nursery	attended nursery school (binary: yes or no)
higher	wants to take higher education (binary: yes or no)
romantic	with a romantic relationship (binary: yes or no)
freetime	free time after school (numeric: from 1 – very low to 5 – very high)
goout	going out with friends (numeric: from 1 – very low to 5 – very high)
Walc	weekend alcohol consumption (numeric: from 1 – very low to 5 – very high)
Dalc	workday alcohol consumption (numeric: from 1 – very low to 5 – very high)
health	current health status (numeric: from 1 – very bad to 5 – very good)
absences	number of school absences (numeric: from 0 to 93)
G1	first period grade (numeric: from 0 to 20)
G2	second period grade (numeric: from 0 to 20)
G3	final grade (numeric: from 0 to 20)

Table A.1 Student related variables from the grades dataset [24].



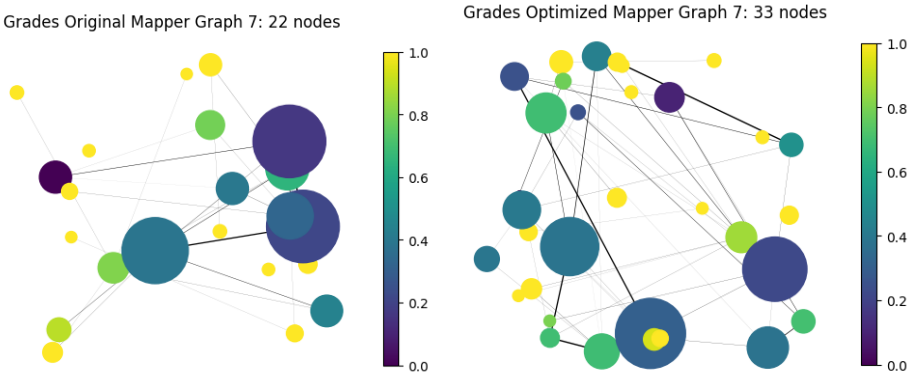
(a) Original mapper graph (b) Optimized mapper graph

Figure A.3 Mapper graphs for the grades dataset for Experiment 4 before and after optimization. For this example, the typical optimization work flow was not successful.



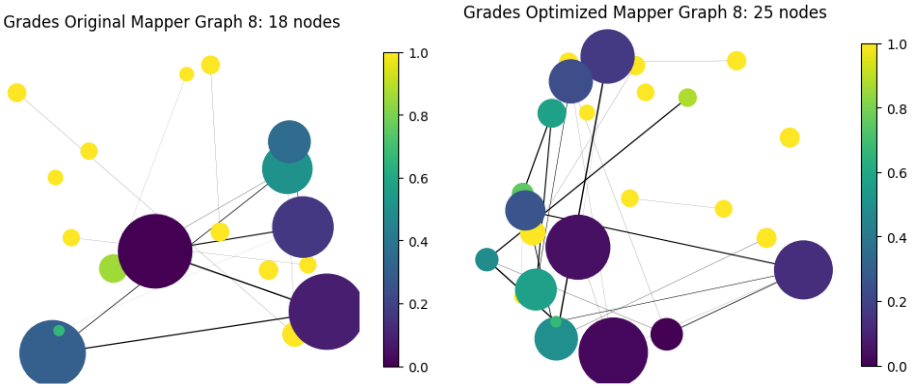
(a) Original mapper graph (b) Optimized mapper graph

Figure A.4 Mapper graphs for the grades dataset for Experiment 5 before and after optimization. For this example, the typical optimization work flow was not successful.



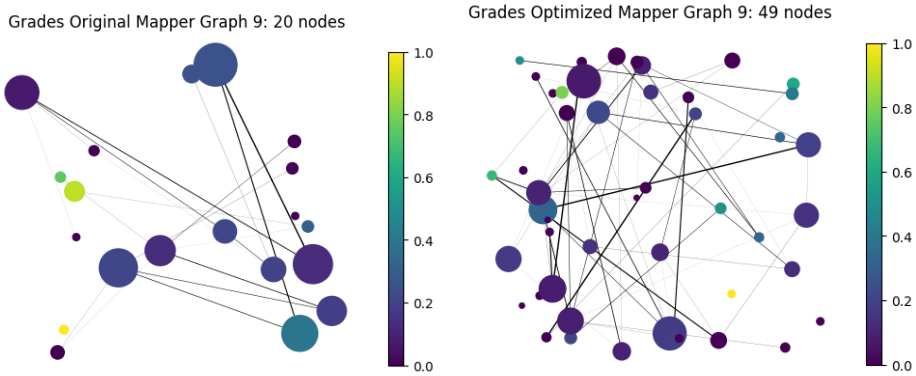
(a) Original mapper graph (b) Optimized mapper graph

Figure A.5 Mapper graphs for Experiment 7 before and after optimization. For this example, the typical optimization work flow was not successful.



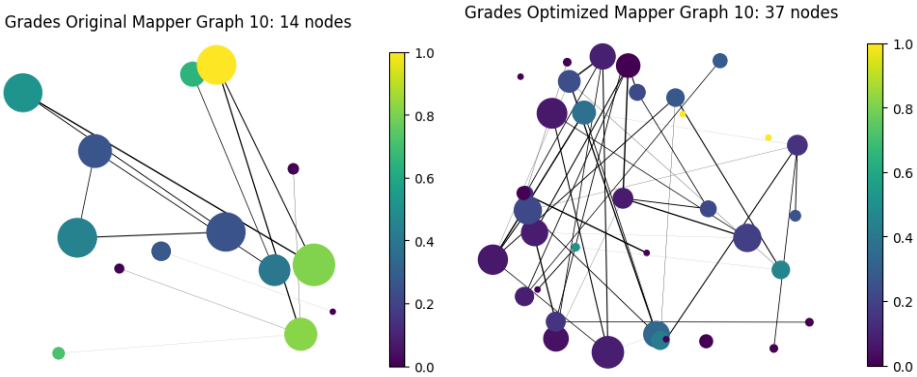
(a) Original mapper graph (b) Optimized mapper graph

Figure A.6 Mapper graphs for Experiment 8 before and after optimization. For this example, the typical optimization work flow was not successful.



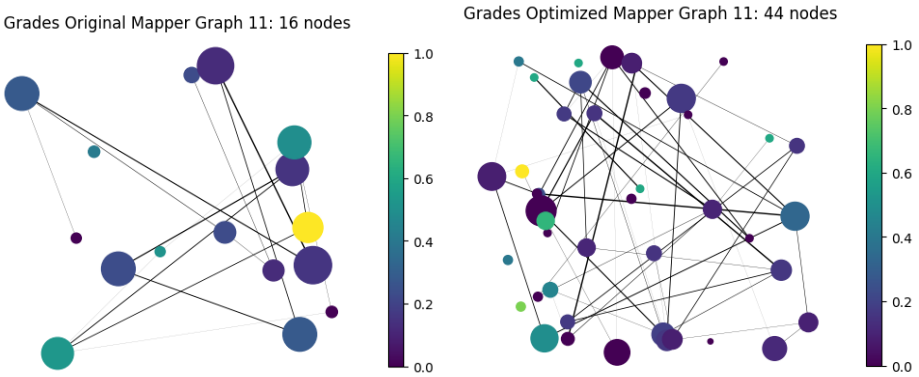
(a) Original mapper graph (b) Optimized mapper graph

Figure A.7 Mapper graphs for Experiment 9 before and after optimization. For this example, the typical optimization work flow was not successful.



(a) Original mapper graph (b) Optimized mapper graph

Figure A.8 Mapper graphs for Experiment 10 before and after optimization. For this example, the typical optimization work flow was not successful.



(a) Original mapper graph (b) Optimized mapper graph

Figure A.9 Mapper graphs for Experiment 11 before and after optimization. For this example, the typical optimization work flow was not successful.