THESIS

APPLICATIONS OF TOPOLOGICAL DATA ANALYSIS TO NATURAL LANGUAGE PROCESSING AND COMPUTER VISION

Submitted by

Jason S. Garcia

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2022

ABSTRACT

APPLICATIONS OF TOPOLOGICAL DATA ANALYSIS TO NATURAL LANGUAGE PROCESSING AND COMPUTER VISION

Topological Data Analysis (TDA) uses ideas from topology to study the "shape" of data. It provides a set of tools to extract features, such as holes, voids, and connected components, from complex high-dimensional data.

This thesis presents an introductory exposition of the mathematics underlying the two main tools of TDA: Persistent Homology and the MAPPER algorithm. Persistent Homology detects topological features that persist over a range of resolutions, capturing both local and global geometric information. The MAPPER algorithm is a visualization tool that provides a type of dimensional reduction that preserves topological properties of the data by projecting them onto lower dimensional simplicial complexes.

Furthermore, this thesis explores recent applications of these tools to natural language processing and computer vision. These applications are divided into two main approaches: In the first approach, TDA is used to extract features from data that is then used as input for a variety of machine learning tasks, like image classification or visualizing the semantic structure of text documents. The second approach, applies the tools of TDA to the machine learning algorithms themselves. For example, using MAPPER to study how structure emerges in the weights of a trained neural network.

Finally, the results of several experiments are presented. These include using Persistent Homology for image classification, and using MAPPER to visual the global structure of these data sets. Most notably, the MAPPER algorithm is used to visualize vector representations of contextualized word embeddings as they move through the encoding layers of the BERT-base transformer model.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Nikhil Krishnaswamy, for his advice, guidance, and encouragement throughout this process. Our many conversations and discussions were an invaluable resource. This thesis would not have been possible without his insight and patience.

I'm also indebted to Francisco Ortega, who brought me to CSU and selflessly encouraged me to pursue my dreams. His kindness and support helped me persevere through some difficult times.

I would also like to thank my graduate committee members, Henry Adams and Ross Beveridge, for their feedback and insightful suggestions. I'm grateful for their comments and constructive criticism.

I am also grateful for my colleagues in the SIGNAL Lab for fostering a fertile environment to discuss and explore new ideas. Many of these conversations help shape the direction of my research.

I would be remiss if i didn't acknowledge my friends in the NUI Lab for their camaraderie and hospitality. During my time here at CSU, they made feel at home.

Finally, I would like to thank my father, Israel Garcia, for his unwavering support and encouragement.

DEDICATION

This work is dedicated to Lori Elaine Elliott and Israel Alonzo Garcia, to whom I owe everything.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

Topological Data Analysis (TDA) is a branch of applied mathematics that uses ideas from topology to study the "shape" of data. It is predicated on the idea that topological features of data sets, such as holes, voids, and connected components, contain useful information that can be exploited to reveal the structure of complex high-dimensional data. The set of tools provided by TDA extracts these topological features in such a way that is robust to noise and small perturbations.

While TDA has been applied to a wide range of scientific disciplines, like Chemistry, Biology, and Physics, in this thesis, I restrict the focus to applications in machine learning. Specifically, I concentrate on the fields of Natural Language Processing (NLP) and Computer Vision.

In Chapter 2, I discuss some notions from topology that will inform our understanding about exactly what topological features TDA extracts from data. The material presented in this chapter provides the mathematical foundation for understanding the primary tools and methods that fall under the TDA umbrella.

Chapter 3 formally introduces the field of Topological Data Analysis and briefly describes its two main tools: Persistent Homology and the MAPPER algorithm. Persistent Homology is essentially an adaptation of traditional homology that handles point cloud data. The MAPPER algorithm is a visualization tool that combines a type of dimensional reduction with a method of *partial clustering*.

Chapter 4 contains a discussion of a few recent papers that cover applications of TDA to Natural Language Processing (NLP) and Computer Vision. While results from these papers are mentioned, the primary reason that these papers were selected is their unique implementation of TDA methods.

In Chapter 5, I detail a couple of experiments where I apply tools from TDA to address some questions in NLP and computer vision. One such experiment involves using MAPPER to visualize the hidden representations of polysemous words in the BERT transformer model. The remaining experiments explore using persistent homology to classify images of hand written digits from the MNIST database.

Finally, Chapter 6 briefly summarizes the main points of this thesis and provides several directions for future work. In particular, it details possible extensions of my experiment which used the MAPPER algorithm to explore vector representations in the BERT transformer model.

Chapter 2

Topological Preliminaries

The purpose of *this* chapter is twofold. First, I will introduce some notions from *algebraic topology* that will be useful for informing our intuition about what is meant by the "shape" of data. Specifically, I will discuss the concept of an n*-dimensional hole*, a topological feature of a space that *persistent homology* explicitly exploits. Second, I will introduce a type of topological space that naturally arises from data and show how information about n*-dimensional holes* is extracted from these spaces.

2.1 N-Dimensional Holes

As in any category in mathematics, determining which objects are to be considered equivalent is of paramount importance. Often, fundamental topological properties of a space, like *continuity*, *compactness*, or *connectedness*, can be used to distinguish between topological spaces. For example, if we remove a point from the real line $\mathbb{R}$, we are left with a disconnected space. On the other hand, removing a single point from the unit circle S^1 still leaves a single connected component. This implies there is some fundamental difference in the geometric properties of these two spaces.

However, sometimes these simple topological properties are not enough. As a motivating example, consider the unit circle S^1 and the sphere S^2. While we intuitively understand that these spaces are different, removing a single point will not help us distinguish between the two[1].

The topological feature I am trying to motivate is the notion of a *hole* or *void*. As it turns out, the presence of holes is a powerful tool for classifying topological spaces. In fact, this is one of the primary concerns of *algebraic topology*. Broadly speaking, the principal conceit underlying *algebraic topology* is to take a topological space X and associate an algebraic object to it, like a *group* or *vector space*. This association is constructed in a such a manner that the algebraic

[1]The astute reader will immediately recognize that removing two points will differentiate the spaces. But what about S^2 and S^3?

object contains topological information about X in its structure. In doing so, questions about the topological space X can now be formulated as questions about the structure of the algebraic object.

As illustrated in Figure 2.1, it is useful to think of this map from X to some algebraic object as transporting information. For our purposes, the type of maps we are concerned with will be transporting information about the n*-dimensional holes* present in X.

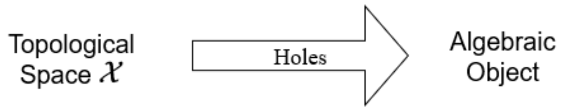


Figure 2.1: Map transporting information about the n*-dimensional holes* present in X.

Avoiding the philosophical hurdles of describing something that isn't there, mathematicians simply define the dimension of a hole (or void) to be the dimension of its boundary. For example, as seen in Figure 2.2, the space enclosed by the unit circle S^1 is a 1-dimensional hole since the boundary of this space, S^1 itself, is a 1-dimensional object. Similarly, the void enclosed by the sphere S^2 is a 2-dimensional hole since its boundary is a 2-dimensional surface.

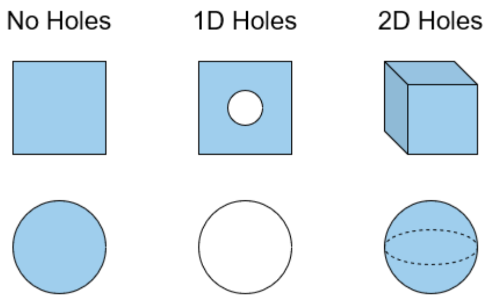


Figure 2.2: The dimension of a hole is defined to be the dimension of its boundary.

In general, computing these algebraic objects for a given topological space is highly non-trivial. Fortunately for us, we will be restricting our attention to a special class of well behaved topological spaces. In the following sections, I describe these topological spaces (*simplicial complexes*) and how to construct the algebraic objects (*homology groups*) whose structures carry information about n-dimensional holes.

2.2 Simplicial Complexes

I previously described *simplicial complexes* as well behaved. Amongst several compelling reasons, perhaps the most relevant feature that embodies this characterization is that these spaces admit a description in terms of some fundamental components. Analogous to the relationship primes have to the integers, or the relationship between the periodic table of elements and chemical compounds, *simplicial complexes* are constructed by composing a finite collection of these fundamental components. We begin by defining the basic building blocks of these topological spaces.

Definition 2.2.1 (k-simplex)**.** For an affinely independent set of $k + 1$ points $v_0, v_1, ..., v_k \in \mathbb{R}^k$, a k-simplex is the set of points

$$\sum_{i=0}^{k} \lambda_i v_i \quad \text{such that} \quad \sum_{i=0}^{k} \lambda_i = 1 \quad \text{and} \quad \lambda_i \geq 0$$

By an affinely independent set of $k + 1$ points, we mean that $v_0, v_1, ..., v_k$ do not lie in some hyperplane of dimension less than k. This is equivalent to requiring that the k vectors $v_1 - v_0, v_2 - v_0, ..., v_k - v_0$ are linearly independent.

Geometrically, a k-simplex is just the convex hull of these $k + 1$ points. As such, a k-simplex can be thought of as a k-dimensional generalization of a triangle. In Figure 2.3 we see that a 0-simplex is just a point, a 1-simplex is a line segment, a 2-simplex is a triangle, and a 3-simplex is a tetrahedron. In this geometric context, the condition of affine independence avoids the ambiguity of degenerative cases. For example, we disallow three colinear points from defining a triangle (2-simplex), or four coplanar points from defining a tetrahedron (3-simplex).

The points v_i are referred to as the *vertices* of the simplex. Observe that by removing one of the vertices, the remaining k vertices span a $(k - 1)$-simplex we call a *face* of the original k-simplex. Specifying two points of the k-simplex is often referred to as an *edge*.

Having defined these fundamental components, we now turn our attention to how these simplicies are put together to form *simplicial complexes*. Intuitively, this accomplished by "gluing" the

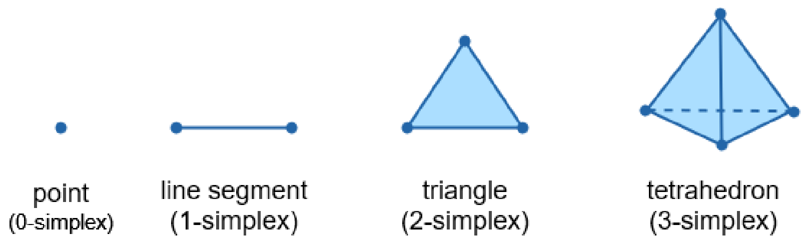


Figure 2.3: A point (0-simplex), a line segment (1-simplex), a triangle (2-simplex), and a tetrahedron (3-simplex),

simplicies in very specific way. The requirement is that the simplicies have to be "glued" together along entire faces. Otherwise, the simplicies are considered disjoint. Observe that if we include a k-simplex in some simplicial complex, then necessarily, all of its subsimplexes are also included in the complex. This intuition is formalized in the following definition.

Definition 2.2.2 (Simplicial Complex)**.** A simplicial complex $\mathcal{K}$ is a set of simplices such that:

- Every face of a simplex from $\mathcal{K}$ is also in $\mathcal{K}$.
- The non-empty intersection of any two simplices $\sigma_1, \sigma_2 \in \mathcal{K}$ is a face of both σ_1 and σ_2.

The dimension of a simplicial complex is defined to be the dimension of its largest simplex. An example of a simplicial complex is shown in Figure 2.2

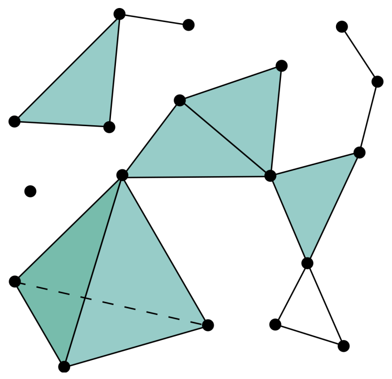


Figure 2.4: Exmaple of a simplicial complex

2.3 Simplicial Homology

Given a simplicial complex $\mathcal{K}$, our job now is to compute its homology- a sequence of groups that carry information about the n-dimensional holes present in $\mathcal{K}$. To that end, we introduce the notion of an oriented k-simplex.

For a k-simplex σ, we define an *orientation* on σ by specifying an order on its set of vertices. A consequence of this ordering, say $\sigma = [v_0, v_1, ..., v_k]$, is that it determines an orientation of the edges $[v_i, v_j]$. As shown in Figure 2.5, the orientation of an edge is prescribed by increasing subscripts. Naturally, the orientation of any face or subsimplex *must* be consistent with the orientation of the larger simplex.

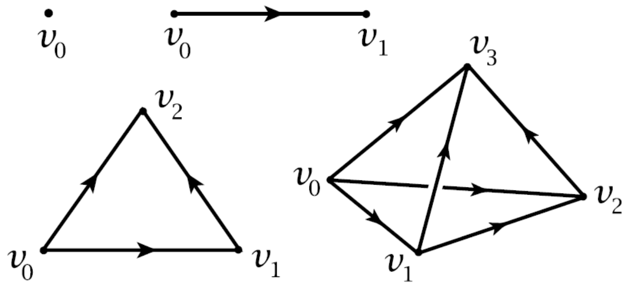


Figure 2.5: Exmaples of oriented simplicies
Image courtesy of Hatcher [1].

As a general rule, we consider two orientations to be equivalent if and only if their orderings differ by an even permutation. Effectively, this means every simplex has exactly two orientations. Observe that this allows us to define the "negative" of an oriented simplex to be the simplex with the opposite orientation. For example, given the 1-simplex $[v_i, v_j]$, we may write

$$[v_i, v_j] = -[v_j, v_i]$$

Algebraically, this observation hints at some form of an additive structure, which we shall exploit in the following sections.

2.3.1 The k-chain Groups C_k

Given an oriented simplicial complex $\mathcal{K}$, let $\mathcal{B}_k$ be the set of all k-dimensional simplices in $\mathcal{K}$. The group C_k is defined to be the *free abelian group* with $\mathcal{B}_k$ as its basis. Basically, one can think of a *free abelian group* as being "constructed" from its basis elements. That is, every element of the group can be uniquely expressed as an integer combination of finitely many basis elements. As such, elements of C_k are finite formal sums referred to as k-chains. A typical k-chain $c \in C_k$ is given by,

$$c = \sum_{i=0}^{N} a_i \sigma_i$$

where $a_i \in \mathbb{Z}$ and σ_i is an oriented simplex of dimension k. Essentially, the group C_k is the collection of all possible k-chains.

2.3.2 The Boundary Homomorphism ∂_k

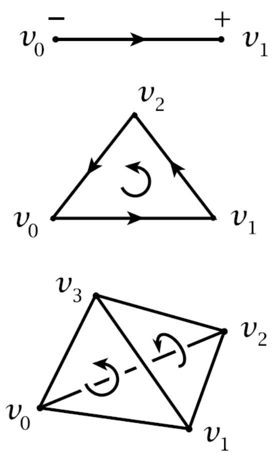
In this section, we introduce the notion of the *boundary* of a simplex. Intuitively, we can think of the boundary of a k-simplex $[v_0, v_1, ..., v_k]$, to be the collection of its (k-1)-dimensional faces. We can extend this notion of a *boundary* to define a map between k-chain groups by specifying its values on the basis elements.

Definition 2.3.1 (k^{th} Boundary Map ∂_k)**.** For an oriented k-simplex $\sigma \in C_k$, say $\sigma = (v_0, \ldots, v_k)$, the k^{th} boundary map $\partial_k : C_k \longrightarrow C_{k-1}$ is given by

$$\partial_k(\sigma) = \sum_{i=0}^{k} (-1)^i (v_0, \ldots, \hat{v}_i, \ldots, v_k)$$

where the oriented simplex $(v_0, \ldots, \hat{v}_i, \ldots, v_k)$ is the i^{th} face of σ.

As it turns out, this boundary map ∂_k actually defines a homomorphism of groups. Notice that the alternating sum in the above definition utilises the additive structure that was hinted at when we introduced the idea of an *orientation*. So that the "negative" of an *orientated simplex* is well defined. Figure 2.6 shows examples of the boundary map applied to some oriented simplicies.



$$\partial[v_0, v_1] = [v_1] - [v_0]$$

$$\partial[v_0, v_1, v_2] = [v_1, v_2] - [v_0, v_2] + [v_0, v_1]$$

$$\begin{aligned}\partial[v_0, v_1, v_2, v_3] = [v_1, v_2, v_3] - [v_0, v_2, v_3] \\ + [v_0, v_1, v_3] - [v_0, v_1, v_2]\end{aligned}$$

Figure 2.6: Boundaries of some oriented simplicies.
Image courtesy of Hatcher [1].

2.3.3 Chain Complex and the Homology Groups $H_k(\mathcal{K})$

A straightforward computation reveals that the composition $\partial_k \circ \partial_{k+1} : C_{k+1} \longrightarrow C_{k-1}$ is the trivial homomorphism. In words, the boundary of a boundary is zero. This property of the boundary homomorphism, and the k-chain groups themselves, form a *chain complex*, a sequence of abelian groups and homomorphisms that are written as follows:

$$\cdots \xrightarrow{\partial_{k+2}} C_{k+1} \xrightarrow{\partial_{k+1}} C_k \xrightarrow{\partial_k} C_{k-1} \xrightarrow{\partial_{k-1}} \cdots \xrightarrow{\partial_2} C_1 \xrightarrow{\partial_1} C_0 \xrightarrow{\partial_0} 0$$

The property that the composition of any two consecutive boundary maps is the zero map, implies that the image of the $k+1$ boundary map *must* be contained in the kernel of the k^{th} boundary map.

$$\partial_k \circ \partial_{k+1} = 0 \implies Im(\partial_{k+1}) \subset Ker(\partial_k)$$

The elements of $B_k := Im(\partial_{k+1})$ are referred to as *boundaries* while elements of $Z_k := Ker(\partial_k)$ are called *cycles*. The containment $B_k \subset Z_k$ is depicted in Figure 2.7.

We are now in a position to define the homology groups of the simplicial complex $\mathcal{K}$.

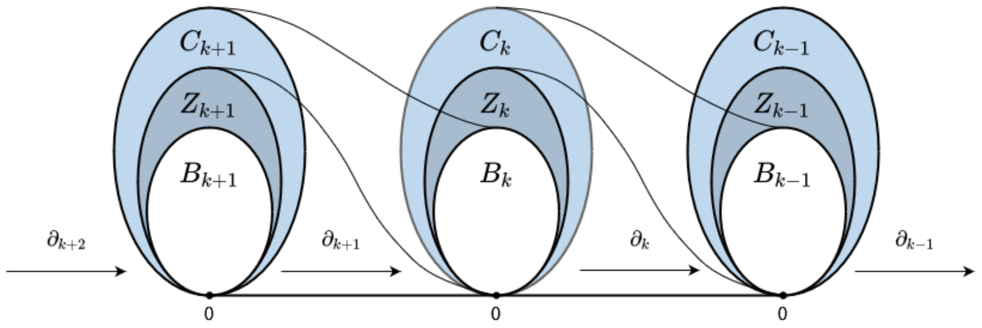


Figure 2.7: Chain complex with boundary maps.

Definition 2.3.2 (k^{th} Homology Group $H_k(\mathcal{K})$)**.** For an oriented simplicial complex $\mathcal{K}$, the k^{th} homology group of $\mathcal{K}$ is defined to be the quotient group

$$H_k(\mathcal{K}) = Z_k/B_k$$

Observe that since the original chain groups C_k were abelian, the subgroups Z_k and B_k are abelian as well. Therefore, the condition of normality ($B_k \trianglelefteq Z_k$) is immediate and the quotient Z_k/B_k is well defined. Also, note that the k^{th} homology group is nonzero precisely when there are k-cyles in $\mathcal{K}$ that did not arise as the boundary of $k+1$ simplex. Formally, this is what mathematicians mean when they say there is a k-dimensional hole present in the simplicial complex $\mathcal{K}$.

The intuition is that the k^{th} homology group detects the presence of k-dimensional holes. For instance, the homology group $H_1(\mathcal{K})$ will contain information about the 1-dimensional holes in $\mathcal{K}$, while $H_2(\mathcal{K})$ will contain information about the 2-dimensional holes in $\mathcal{K}$. In the case $k = 0$, the homology group $H_0(\mathcal{K})$ will reveal the presence of connected components or "0-dimensional" holes.

Furthermore, the kth homology group not only detects the presence of k-dimensional holes, it also carries information about *how many* k-dimensional holes there are. The idea *here* is that every k-dimensional hole in $\mathcal{K}$ will correspond to a generator of of the homology group $H_k(\mathcal{K})$.

Definition 2.3.3 (k^{th} Betti number β_k)**.** For an oriented simplicial complex $\mathcal{K}$, the k^{th} Betti number β_k is defined to be the rank of the homology group $H_k(\mathcal{K})$

$$\beta_k = rank(H_k(\mathcal{K}))$$

Essentially, the k^{th} Betti number β_k counts the number of k-dimensional holes that exist in the simplcial complex $\mathcal{K}$. The first few Betti numbers for some simple topological spaces are shown in Figure 2.8.

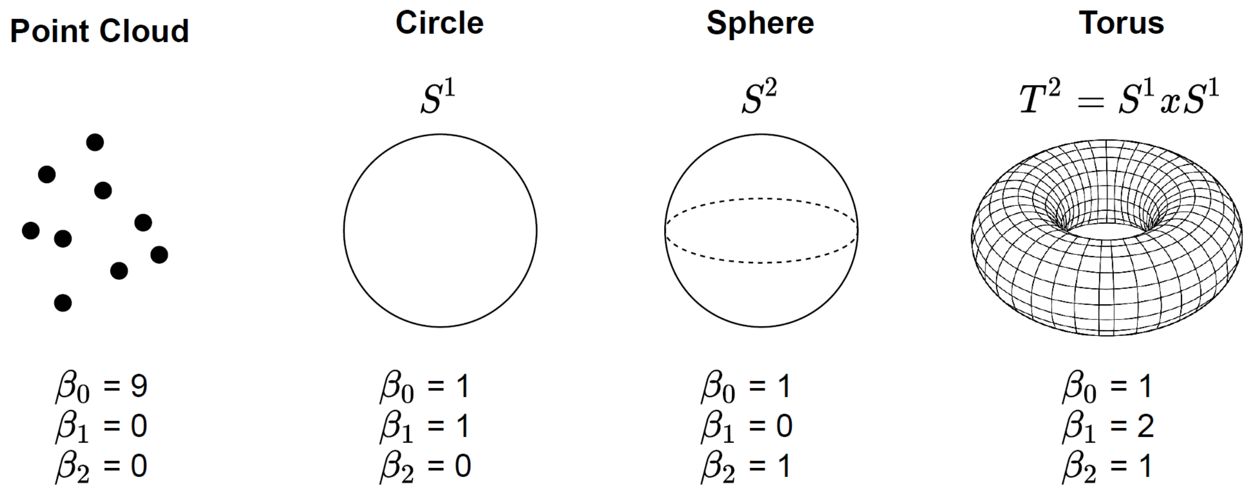


Figure 2.8: The first few Betti numbers for some common topological spaces. Recall that β_0 counts the number of connected components, β_1 counts the number of of 1-dimensional holes, and β_2 counts the number of of 2-dimensional holes.

Chapter 3

Topological Data Analysis

As previously stated, Topological Data Analysis (TDA) is concerned with the "shape" of data. It is an extremely powerful tool for extracting topological information from high dimensional data sets that may be noisy or incomplete. It is premised on the idea that topological properties of a data set, like connectedness and the presence of holes, contain useful information.

This chapter briefly introduces the two main tools of TDA: Persistent Homology[2] and the MAPPER algorithm.

3.1 Persistent Homology

Persistent homology is a tool for computing topological features of a space over a range of different scales. It detects the presence of n-dimensional holes in a data set and keeps track of "when" these features appear and disappear.

3.1.1 Filtrations and the Vietoris-Rips Complex

In section 2.3, I introduced the idea of *simplicial homology*. Given a single simplicial complex $\mathcal{K}$, a sequence of groups were constructed that contained information about the number of n-dimensional holes. However, data sets ususaly come in the form of a point cloud: a set of discrete points embedded in some metric space. How do we get a simplicial complex from a point cloud? One such possibility is the Vietoris-Rips construction, which we now define.

Definition 3.1.1 (Vietoris-Rips Complex)**.** A Vietoris-Rips Complex of diameter $\epsilon \geq 0$, is defined as

$$VR(\epsilon) = \{\sigma |\, diam(\sigma) \leq \epsilon\}$$

where $diam(\sigma)$ is the largest distance between any two vertices of the simplex σ.

[2]The exposition here is modeled after Zhu [2]. For a more complete treatment see Zomorodian and Carlsson [3].

The natural question that arises is, what choice of ϵ will reveal the most important features of a given data set? The insight of Persistent Homology is that it examines a range of epsilons. Initially, when $\epsilon = 0$, our simplicial complex $VR(0)$ is simply just the points in the original cloud, a collection of disjoint 0-simplices. As the value of ϵ increases, more simplicies are added to the complex.

For an increasing sequence of ϵ's, we produce what is called a *filtration*, a sequence of increasing simplicial complexes

$$VR(\epsilon_0) \subset VR(\epsilon_1) \subset ...$$

Persistent homology keeps track of the homology groups as we move through the filtration. It records the values of ϵ where n-dimension holes appear, and also records the values of ϵ where these holes disappear. That is, it detects how long these features persist.

3.1.2 Bar Codes and Persistence Diagrams

As we have seen, persistent homology begins with a point cloud. As it moves through a filtration of this cloud, a nested sequence of simplicial complexes, it records when a homology class (hole) first appears and when it disappears. The result is that for each homology class (hole) detected, we have a pair of real numbers, say ϵ_1 and ϵ_2, that correspond to the birth and death of that feature in the filtration. There are two common ways to represent these pairs.

One way to visualize these pairs, the output of persistent homology, is a *bar code*. A *bar code* is a graphical representation of the homology classes (holes) as a collection of horizontal line segments where the x-axis is given by ϵ. Each horizontal bar corresponds to the birth and death of a separate topological feature. The intuition is that longer bars, ones persisting over a greater range of ϵ's, correlate to more robust topological features and are more representative of the global structure present in the data.

Another way to visualize these pairs, is a *persistence diagram*. Basically, a *persistence diagram* is 2D plot of the birth and death points for each homology class (hole), allowing for multiplicities. It maybe the case that separate features share the same birth and coordinates. The birth of a feature is plotted on the x-axis and the death of a feature is plotted on the y-axis. Observe that since ϵ_1 is necessarily less than ϵ_2 , a feature can not die before it is born, all the points in a persistent diagram lie above the $y = x$ line. The intuition is that points lying farther away from the $y = x$ line, ones that "lived" longer, correspond to features that are more characteristic of the global structure of the data. These features persisted over a wider range of scales. Points lying close to the $y = x$ line didn't last very long and are considered as less important to the structure of the data. These points are typically considered to be noise.

Examples of bar codes and persistent diagrams can be found in Figure 3.1.

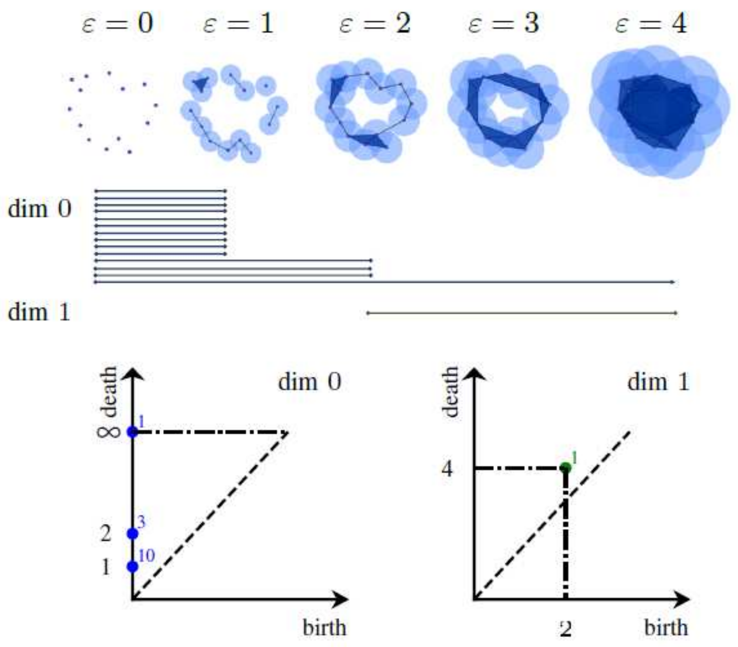


Figure 3.1: Bar Codes and Persistent Diagrams for a four step filtration of a point cloud.
Image courtesy of Garin et al. [4].

3.2 The MAPPER Algorithm

The MAPPER algorithm was first introduced in 2007 by Gurjeet Singh et al. [5]. At its core, MAPPER is a visualization tool that combines a type of dimensionality reduction with a method of *partial clustering*. The intuition is that MAPPER preserves the topological properties of a high dimensional point cloud when it projects the data onto lower dimensional simplicial complexes. Typically, these lower dimensional simplicial complexes are 1-dimensional; i.e., a vertex-edge graph.

Given a high dimensional point cloud X, the first step of the MAPPER algorithm is to project the cloud onto a lower dimension parameter space Y.

$$f : X \longrightarrow Y$$

This function f is referred to as a *filter function* or *data lens*. While there are many possible choices for the parameter space Y, the filter function f is often chosen to be a real-valued function. In the special case where the parameter space is chosen to be $\mathbb{R}$, the MAPPER algorithm is essentially a discrete version of a Reeb graph (see [6]) associated to the filter function f. [5]. Two very common choices of f include PCA projections and t-distributed stochastic neighbor embedding (t-SNE). To simplify our exposition in what follows, we will impose the assumption that the parameter space $Y = \mathbb{R}$. That is,

$$f : X \longrightarrow \mathbb{R}$$

After the point cloud has been mapped into the parameter space, the next step in the MAPPER algorithm is to construct a finite open cover for $Im(f)$ consisting of overlapping intervals $\{U_i\}_{i \in I}$.

$$Im(f) \subset \bigcup_{i \in I} U_i \quad \text{and} \quad U_i \bigcap U_{i+1} \neq \emptyset$$

This construction of this cover for $Im(f)$ gives us two parameters to choose, namely the length of the intervals and the percentage overlap between them. Collectively, these two parameters control a property referred to as *resolution*.

The main idea behind creating a cover for the image of f, is to provide a guide with which to partition, or "bin", the points of the original point cloud. The idea is to apply some sort of clustering method to the pull back $f^{-1}(U_i)$ for each open set U_i in the cover.

The final step in the MAPPER algorithm produces a vertex-edge graph. For every cluster that is found in a single pull back, a node is created. If two nodes from different "bins" have a non-empty intersection, an edge is added between the two nodes.

3.2.1 MAPPER Example

The paper that introduced MAPPER, Singh et al. [5], used 3D object recognition to motivate the MAPPER algorithm. In keeping with this motivating application, I chose the following example from Lum et al. [7] to illustrate the MAPPER algorithm.

This example begins with a point cloud $X \in \mathbb{R}^3$ sampled from a 3-dimensional hand as seen Figure 3.2. The filter function will be a simple projection onto the x-coordinate. That is, let $f : X \longrightarrow \mathbb{R}$ be given by:

$$f(x, y, z) \mapsto x$$

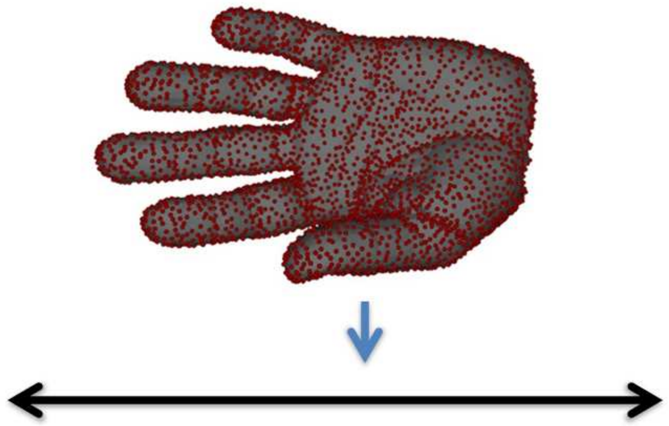


Figure 3.2: Points sampled from a 3-dimensional hand projected onto the x-axis.
Image courtesy of Lum et al. [7].

We now chose an open cover for the $Im(f)$ consisting of six over lapping intervals. In Figure 3.3, these six intervals have been marked and colored on the x-axis. Observe that the coloring highlights the overlap of the pullbacks for each interval, allowing us to see which points in the original point cloud belong to which interval.

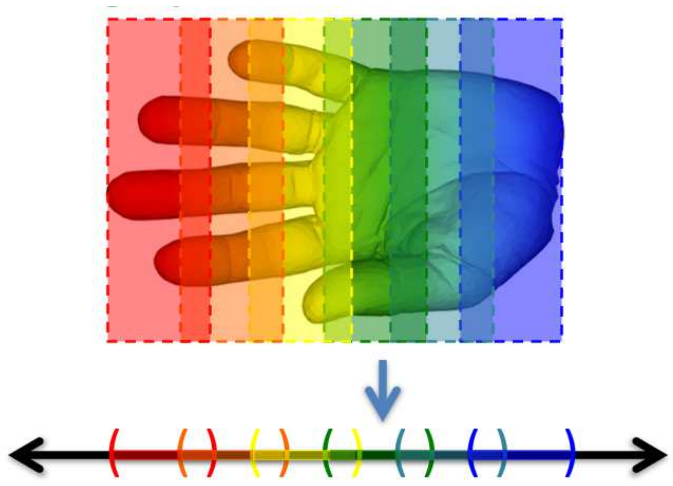


Figure 3.3: Open cover for the $Im(f)$ consisting of six over lapping intervals. Both the intervals and their pre-images under f have been colored to highlight their overlap.
Image courtesy of Lum et al. [7].

Now that the points in the original cloud are "binned" by the intervals, the next step in MAPPER is to apply a clustering algorithm to the points in each bin. For each cluster found in a bin, a node of the resulting graph is created. Figure 3.4 shows the original hand, partitioned by the intervals, and a node corresponding to each cluster found.

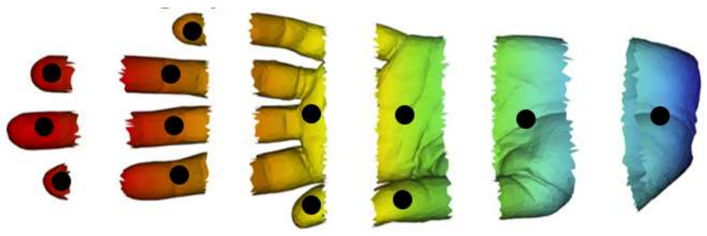


Figure 3.4: For each cluster in a partition, a node of the final graph is created.
Image courtesy of Lum et al. [7].

After clustering has been performed, the last step in the MAPPER algorithm is to create the edges for the final graph. If two nodes, from separate bins, have a non-empty intersection, an edge is drawn between these two nodes. Observe that this intersection is a result of the overlap in intervals for the original cover. The resulting edges for our example are displayed in Figure 3.5.

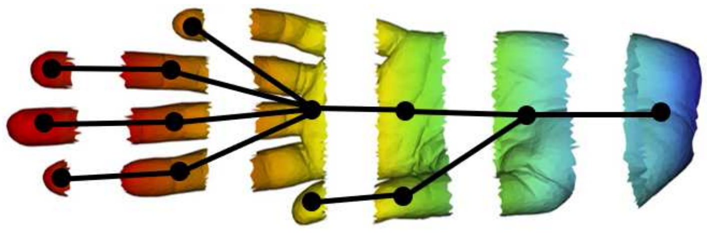


Figure 3.5:
Image courtesy of Lum et al. [7].

The result of the MAPPER algorithm is a vertex-edge graph that has managed to capture some topological properties of the original point cloud. As the data was sampled from a 3-dimensional hand, MAPPER was a able to distill how the various components of a hand are connected.

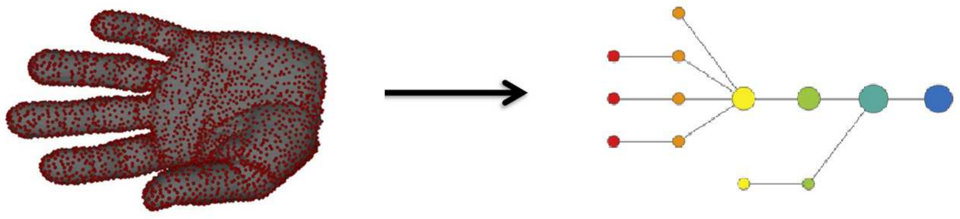


Figure 3.6: MAPPER has taken a 3D point cloud sampled from a hand and produced a vertex-edge graph that captures the topology of a hand.
Image courtesy of Lum et al. [7].

As we can see, MAPPER was able to detect the topology of a hand in 3-space and represent these relationships in a 1-dimensional simplicial complex. This map from the original point cloud to the resulting graph is illustrated in Figure 3.6.

Chapter 4

TDA and Machine Learning

This chapter explores recent applications of TDA to natural language processing and computer vision. These applications can be divided into two main approaches: In the first approach, TDA is used to extract features from data that is then used as input for a variety of machine learning tasks, like image classification or visualizing the semantic structure of text documents. The second approach, applies the tools of TDA to the machine learning algorithms themselves. For example, using MAPPER to study how structure emerges in the weights of a trained neural network.

4.1 Natural Language Processing

While applications of TDA abound in many areas of science and data analysis, applying TDA to Natural Language Processing is still challenging [8]. As TDA exploits the shape of data, it is not exactly clear how one should interpret the geometry of language.

Most NLP tasks begin by transforming natural language into some sort of numerical representation. The idea is to map textual data into a space more amenable to computation. Typically, this representation takes the form of a real-valued vector that attempts to encode the "meaning" or "relevance" of a word. Furthermore, this embedding is usually constructed in such a manner that words with similar meaning are expected to be "close" together.

Once textual data has been mapped in some metric space, the application of TDA is quite natural. The points in this space have an intrinsic geometry that allows TDA to extract topological features. However, what these topological features, like holes and connected components, should represent in the original text is often ambiguous.

This was the motivating factor in my decision to discuss the paper by Zhu [2]. In this paper, Zhu presents a novel way to interpret topological features of text. Through this interpretation, Zhu assigns a concrete meaning to connected components and 1-dimension holes.

Zhu suggests that holes can be interpreted as semantic “tie-backs” in a text document [2]. For example, a well written essay might have a concluding paragraph that “ties back” to its introduction. This premise is formalized in the following two algorithms introduced in the paper.

Similarity Filtration: SIF

In what Zhu calls *Similarity Filtration* (SIF), suppose that a document or corpus has been divided into smaller units which have been assigned real-valued vectors , say $x_1, \ldots, x_n$. These units may be sentences, paragraphs, or even whole documents themselves in the case of a corpus. Furthermore, suppose we have defined some notion of distance where the metric encodes information about similarity, so that similar units are considered "close". For instance, we may have embedded these units into a vector space using term frequency–inverse document frequency (tf-idf). In which case the cosine metric would be a natural choice.

SIF now applies persistent homology to the Vietoris-Rips complex over the points $x_1, \ldots, x_n$. As we move through the filtration, the increasing diameter of the neighborhoods corresponds to allowing looser and looser "tiebacks" [2].

Similarity Filtration with Time Skeleton: SIFTS

In what Zhu calls *Similarity Filtration with Time Skeleton* (SIFTS), the basic idea behind SIF is extended to incorporate where a unit appears in the text. The idea is to capture the "flow" of the document. Observe that in SIF, the order that the units $x_1, \ldots, x_n$ occur is immaterial when applying persistent homology.

This extension is accomplished by adding “time edges” to the simplicial complex before any similarity filtration is performed. Specifically, 1-simplices are added between consecutive units x_i and x_{i+1}. These edges form a “time skeleton” by connecting units in document order [2]. By adding these edges, we are forcing consecutive units to be considered as "close".

To illustrate the differences in SIF and SIFTS, Zhu constructs the following toy data set of 4 points in $\mathbb{R}^2$:

$$x_1 = (0,0) \quad x_2 = (1,0) \quad x_3 = (2,0) \quad x_4 = (0, -\frac{1}{2})$$

Now consider the Vietoris-Rips complex of diameter $\epsilon = \frac{1}{2}$. This complex, denoted as $VR(\frac{1}{2})$, is shown in figure 4.1 for both the SIF algorithm and the SIFTS algorithm.

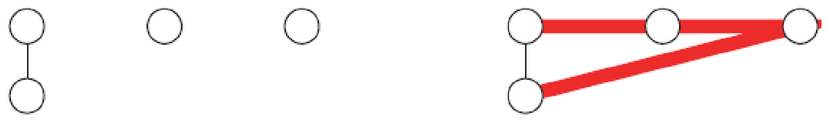


Figure 4.1: $VR(\frac{1}{2})$ for the points x_1, x_2, x_3, and x_4. (Left) SIF (Right) SIFTS with time skeleton in red. *Image courtesy of Zhu [2].*

SIF sees the complex $VR(\frac{1}{2})$ as four vertices and an edge between $x_1 = (0,0)$ and $x_4 = (0, -\frac{1}{2})$. Although SIF considers this edge as a tie-back between the first and last units, no 1-dimensional hole has appeared in $VR(\frac{1}{2})$. In contrast, SIFTS also sees the edge between $x_1 = (0,0)$ and $x_4 = (0, -\frac{1}{2})$, but it includes the time skeleton (in red) constructed from the unit order. As a result of combining the similarity edge and the time skeleton, SIFTS detects the presence of a 1-dimensional hole in the complex $VR(\frac{1}{2})$.

After introducing these two algorithms, SIF and SIFTS, Zhu illustrates their application to a few well known nursery rhymes. Zhu posits that since nursery rhymes are repetitive, they are ideal for persistent homology [2]. Overall, Zhu finds that SIFTS consistently detects more holes and detects them earlier than SIF, highlighting the effectiveness of the time skeleton.

As a more real world example, Zhu uses SIFTS to qualitatively analyze the complexity of children's writing. Zhu's working hypothesis is that older writers will produce text that is structurally richer than younger writers and will therefore contain more "tie backs", or more 1-dimensional holes. Essentially, Zhu is using persistent homology as a feature to measure the complexity of the writing.

Zhu performs this analysis on the LUCY corpus [9], a collection of child and adolescent writing. While Zhu divides this corpus into two sets, child and adolescent, it should be noted that the adolescent set actually consists of 48 undergraduate essays.

Table 4.1: Statistics for Child vs. Adolescent writing. Entries that Zhu found "statistically different" from *Child* are marked by *.

	Child	Adolescent	Adolescent (truncated)
Holes	87%	$100\%^*$	$98\%^*$
$\|H_1\|$	3.0 (± 0.2)	17.6(± 0.9)*	3.9(± 0.2)*
ϵ^*	1.35 (± 0.2)	1.27(± 0.2)*	1.38($\pm .01$)

After applying the SIFTS algorithm to each sample in the LUCY corpus, Zhu extracted two main statistics. The first statistic was the total number of 1-dimensional holes found over all values of ϵ, denoted $|H_1|$. The second statistic, which Zhu denotes as ϵ^*, is the smallest ϵ where the first 1-dimensional hole appeared. If no 1-dimensional holes were found, then ϵ^* was set to $\epsilon^* = \frac{\pi}{2}$, the largest possible angular distance. The results of this experiment can be found in Table 4.1.

The first two columns in Table 4.1 show the difference between child and adolescent writing. Only 87% of child essays produced 1-dimensional holes while every adolescent essay had 1-dimensional holes. The average child essay only contained 3 holes while adolescent essays contained an average of 17.6 holes. Also, observe that the first 1-dimensional hole appears earlier in adolescent writing.

Zhu suggests that there might be reason to suspect that this difference in homology is due to the fact that adolescent essays were about twice as long [2]. In an attempt to correct for this suspicion, Zhu creates a third data set, "Adolescent Truncated". This data set consisted of the first 11 sentences from the adolescent essays. While this removed a large number of "tie-backs" from the adolescent writing, the third column in Table 4.1 still shows a "significant difference" in the total number of 1-dimensional holes.

Zhu concludes that persistent homology was able detect "significant differences" between child and adolescent writing using only structural features [2]. However, the main point of this experiment is not that this classification task necessarily requires such "sophisticated machinery" [2]. Zhu suggests that simpler features such as word usage would probably be sufficient. Rather, the main point of this experiment is that homology contains useful information about the structure of texts [2]. Finally, Zhu posits that incorporating this type of information into existing text representation can potentially improve the performance for many NLP tasks.

4.2 Computer Vision

The first paper we explore in computer vision is by Garin et al. [4]. In this paper, the authors uses persistent homology to extract features from images and propose a data pipeline for the classification of images.

While this paper does inspire an experiment I detail in Chapter 5, my interest in discussing it here stems from the authors' implementation of persistent homology for images. Normally, persistent homology uses simplicial complexes, which are made up of "triangluar" simplices. Since images consist of pixels arranged in a rectangular grid, the authors posit that the more natural construction of a complex from an image should be based on "cubical" simplices [4]. Examples of these cubical simplcies are shown in figure 4.2.

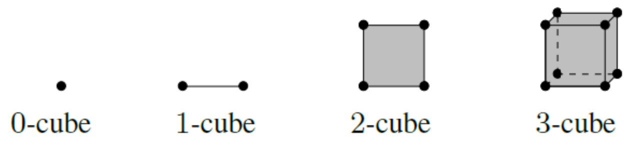


Figure 4.2: Examples of low-dimension cubical simplicies.
Image courtesy of Garin et al. [4]

Analogous to the construction of traditional complexes, these *cubical complexes* are composed from a finite collection of these cubical simplcies. And, as it turns out, persistent homology can be applied to filtrations of these cubical complexes [4]. While using cubical complexes to represent images is not new, the authors cite Allili et al. [10], the authors extend this work to persistent homology.

Along with using these novel representations, the authors also discuss some unique filtrations specifically designed for images. These filtrations exploit the natural geometry of pixels and the pixel values themselves.

For example, one of the filtrations that exploits the geometry of the pixels is called a *Height Filtration*. It selects a direction in the form of a vector and performs a "sweep" of the image, adding simplicies as it pans across the image. Another similar filtration is called a *Radial Filtration*. This filtration selects a single pixel to serve as a center point and adds simplicies to the complex based on the distance from this central pixel.

Another interesting filtration exploits the value of the pixels in a gray scale image. The authors state that gray scale images come with a natural filtration embedded in the values of its pixels. Here, the sublevel sets are indexed by increasing pixel values. An example of this filtration and the resulting bar codes can be found in figure 4.3.

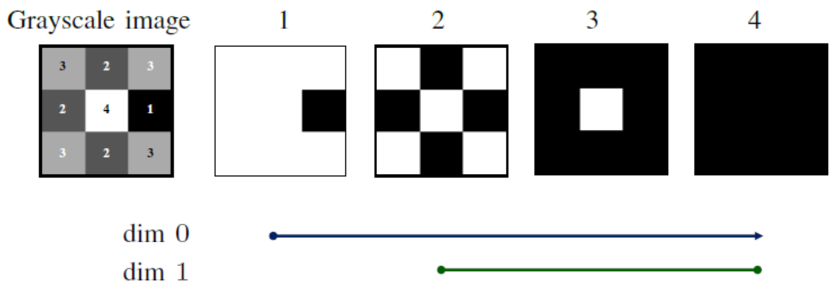


Figure 4.3: Example of persistent homology applied to a gray scale image. The image is represented as cubical complex and the filtration is based on pixel values. The corresponding bar codes are displayed below. *Image courtesy of Garin et al. [4]*

The next paper we examine was published in 2020 by Gabella [11]. This paper uses the MAPPER algorithm to examine the weights of a Neural Network. It follows a similar line of thinking presented in a previous paper by Gabriel and Carlsson in 2019 [12]. In this previous paper, Gabriel and Carlsson use MAPPER to visualize the space of spatial filters (3x3 kernels) for a Convolutional Neural Network (CNN).

In the Gabella paper [11], a simple feed forward neural networks, with a single hidden layer of 100 neurons, was trained on the MNIST dataset of handwritten digits. The hidden layer used a sigmoid activation function and the output layer used softmax. Cross-entropy was used for the loss function.

The main idea was to monitor the evolution of the weights in both the hidden and output layer during training. After each training step (a fixed number of minibatches), the weights for each neuron were extracted. To simplify the situation, the biases were turned off so that only the columns of the weight matrix were considered as parameters.

This procedure produced a point cloud in a high-dimensional vector space. The next step was to analyze the topology of this cloud with the MAPPER algorithm. The assumption was that the resulting graph would encode the topological structure of the weights and represent their evolution during training. The results of the MAPPER algorithm, along with PCA projections, for both the hidden and output layer can be found in figure 4.4

Gabella found that the evolution of the weights during training produced a trees, where subsets of weights branched off from each other. While this branching behavior can be seen by looking at the PCA projections, the topological structure of the trees is more readily discernable in the MAPPER graph [11].

Observe that for the output layer, the number of leaves in the tree coincided with the number of neurons in the output layer, one for each digit. Perhaps the most notable result of this experiment is that each branching point triggered an increase in the model's accuracy. This suggests that the performance of the model was strongly correlated to the branching of the weights [11].

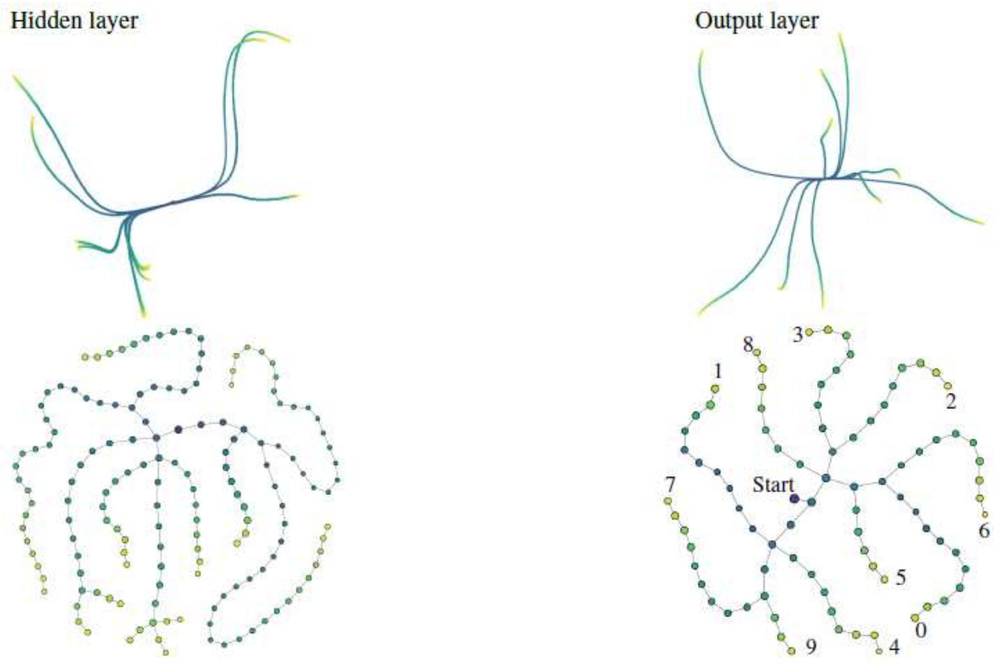


Figure 4.4: Evolution of weights for a neural network with one hidden layer, initialized at zero (colored by training step). Top: PCA projections. Bottom: MAPPER graphs. Left: Hidden layer. Right: Output layer (digits are indicated). *Image courtesy of Gabella [11].*

Chapter 5

Experiments

In this Chapter, I present a couple of experiments that explore applications of TDA to Natural Language Processing and Computer Vision. The first NLP experiment attempts to use persistent homology to extract topological features and classify text data. The second experiment uses MAPPER to explores a popular network architecture (BERT) used in many NLP tasks. The remaining experiments explore the MNIST database with the MAPPER algorithm and use persistent homology to classify the MNIST images.

5.1 20 Newsgroup Data Set

The idea of this first experiment is to investigate the possibility of using persistent homology to classify text. In particular, I wanted to use persistent homology to extract topological features from the 20 Newsgroup Data set and use them for classification. This data set, first assembled by Lang [13], consists of approximately 19,000 newsgroup posts. It contains 20 different newsgroups, each representing a different topic. The data set is also partitioned into training and test sets. A breakdown on the 20 Newsgroup data set by topic and set can be found in Table 5.1.

The 20 Newsgroups collection has become a popular choice for applications of machine learning techniques to text. For example, it is provided by Scikit-learn [14] as a sample data set for tasks such as text classification and text clustering. This was one of the reason for choosing the 20 Newsgroup data set, it has bench-marked results for various machine learning techniques with which to compare performance. As we shall soon see, the 20 Newsgroup data set turned out to be bad choice.

The first step in this experiment was to pre-process the data. The entire corpus, both training and testing sets, were partitioned into individual sentences. Each sentence was then vectorized using term frequency-inverse document frequency (tf-idf). The total vocabulary of the corpus consisted of 129,791 words. Therefore each sentence was mapped to a vector in $\mathbb{R}^{129,791}$.

The vector representations for each sentence were then regrouped into their corresponding samples from both the training and testing sets. Then, the homology classes for each sample corresponding to a post was computed.

Table 5.1: Breakdown of the 20 Newsgroup data set .

Newsgroup	**Train**	**Test**
comp.graphics	584	389
comp.os.ms-windows.misc	591	394
comp.sys.ibm.pc.hardware	590	392
comp.sys.mac.hardware	578	385
comp.windows.x	593	395
sci.crypt	595	396
sci.electronics	591	393
sci.med	594	396
sci.space	593	394
rec.autos	594	396
rec.motorcycles	598	398
rec.sport.baseball	597	397
rec.sport.hockey	600	399
talk.politics.misc	465	310
talk.politics.guns	546	364
talk.politics.mideast	564	376
talk.religion.misc	377	251
alt.atheism	480	319
soc.religion.christian	599	398
misc.forsale	585	390
Total	11 314	7 532

Here is where the first obstacle was encountered. I had originally intended to use the first homology group, 1-dimensional holes, as the primary feature for classification. As it turns out, of the 11,314 samples in the training set, less than 3,000 samples contained 1-dimensional holes. Thinking this may be an artifact of my original choice to use the cosine distance for a metric, I recomputed the homology groups using the standard Euclidean metric. I still found that only a small fraction of the posts contained 1-dimensional holes.

After a cursory examination, it appeared that the presence of 1-dimensional holes was correlated to the length of the post. Posts containing a greater number of sentences, and therefore a greater number of points for computing homology, were more likely to produce 1-dimensional holes. Unable to find a balanced subset of posts containing 1-dimensional holes, I was forced to attempt the classification only using connected components, the zeroth homology groups. This greatly reduced my expectations for classification. It is not exactly clear how the presence of connected components, and when they appear, would produce any distinguishing features.

With only the information from the zeroth homology groups as input, I used a fully connected Neural Network with a single hidden layer of 40 nodes with ReLU activation. The varying number of connected components found in each sampled forced me to pad the input layer with zeros, to compensate for samples with only a few connected components. The output layer consisted of 20 nodes, one for each topic, with softmax activation. Sparse Categorical Cross entropy was used for the loss function.

As expected. the results were abysmal. The over all accuracy of the model was 10%. This is only slightly better than the performance one would expect from random guessing (5%). I suspect the network was able to use the number of connected components in each sample to slightly improve the classification, but not by much. This conjecture would require further investigation to confirm. One thing is for certain, the information extracted from the zeroth homology groups, the connected components, was not enough to successfully distinguish between topics.

5.2 The BERT Transformer Model

The Bidirectional Encoder Representations from Transformers model (BERT), was introduced in 2018 by Devlin et al. from Google in [15]. It utilizes the original transformer model architecture introduced by Vaswani et al. from Google Brain in [16], which in the field of sequence to sequence learning, has become the predominate model for many NLP tasks. It has all but replaced earlier architectures such as Recurrent Neural Networks (RNNs) and Long Term Short Term Memory (LSTMs) [17].

In the context of NLP, the transformer model utilizes the mechanism of self-attention to compute a weighted representation for each word in a sentence. This allows the model to attend to long range dependencies in text. Essentially, the BERT model is a stack of transformer encoding layers, with the output of one layer feeding into the next.

While BERT has produced state-of-the-art results for many NLP benchmarks, the reasons for its success are not well understood [18]. Since BERT's introduction, a substantial portion of the literature has been devoted to understanding its performance. [18]. As an infinitesimal step in this direction, I used TDA to study how BERT expresses the semantics of ambiguous words.

Experiment

In this experiment, I explored how BERT handles representations of polysemous words as they move through its encoding layers. The goal of this experiment was to examine the geometry of these representation using MAPPER and explore how the meaning of a word (expressed in each layer) effects the "shape" of these representations.

The specific model I examine in this experiment is the $BERT_{BASE}$ model, which was also introduced in [15]. It consists of 12 encoder layers, a hidden size of 768, and 12 bidirectional self-attention heads. The model was trained on two self supervised tasks, Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The training corpus consisted of the BooksCorpus (800M words) introduced by Zhu et al. in [19], and English Wikipedia (2,500M words).

For text samples, I constructed ten simple English sentences containing the word "bank". Half of these sentences use the word "bank" to reference a financial institutions and the other half refer to a body of water. These sentences can be found in Table 5.2.

Table 5.2: Sample Sentences with "Bank"

Sample Sentences with "Bank"	
Financial	Water
"The bank teller handed him money."	"She sat down on the bank of the river."
"She cashed the check at the bank."	"The boat was parked on the bank."
"The bank vault was closed."	"He swam to the other bank of the river."
"He stole money from the bank."	"They ate lunch on the left bank of the Seine."
"She transferred money to a different bank."	"He skipped rocks from the river bank."

First, each sentence was tokenized using the BERT Tokenizer, which utilizes the wordpiece model introduced in [20]. Then, with $BERT_{BASE}$ set to feed-forward operation, each tokenized sentence was fed as input into the model. Finally, for each sentence, the hidden state of the word "bank" in each encoding layer was extracted from the model. Since $BERT_{BASE}$ has a hidden size of 768, these hidden representations were vectors in $\mathbb{R}^{768}$.

There were 10 sentences, each with a single use of the word "bank". For each sentence, we obtained 13 vector representations, 1 representation for each of the 12 encoding layers plus an initial BERT token embedding. This produced a data set with 130 samples in $\mathbb{R}^{768}$.

Using this as the initial point cloud, I applied the MAPPER algorithm with the following choice of parameters:

- Filter Function - The point cloud was projected onto the first 2 principal components (2D PCA)
- Cover- The image of this projection was partitioned into 10 hyper-cubes with 50% overlap.
- Clustering Method- DBSCAN (metric = Euclidean, $\epsilon = 10$,min_samples = 2)

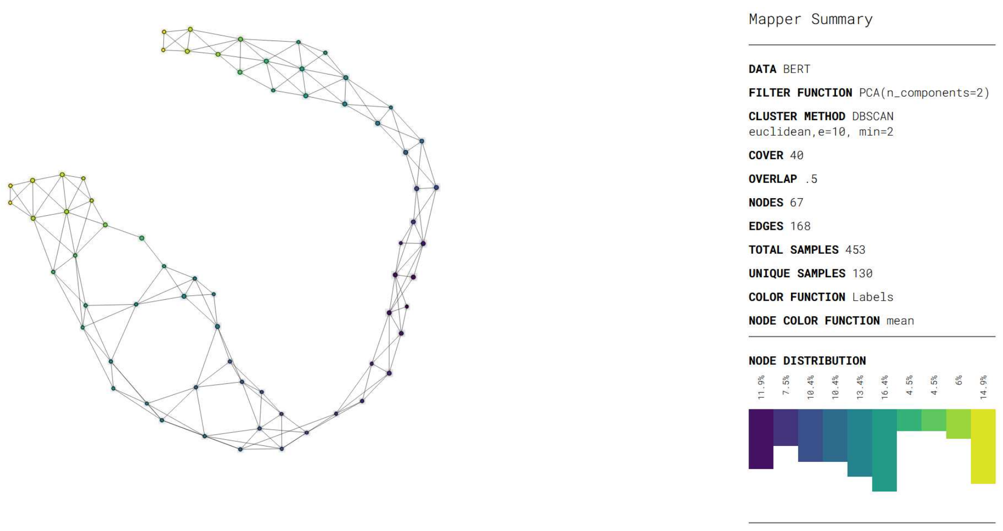
The graph produced by the MAPPER algorithm can see in Figure 5.1.

Analysis

Exploring the MAPPER graph, I observed that in early encoder layers, the representations for the two meanings of the word "bank" were present in the same node. This implies that the initial vector embeddings for both meanings were close[3]. However, the representations for the two meanings quickly diverged as they moved through the layers. In fact, by the third layer, no nodes contained samples from both meanings.

The rest of the layers formed two "flares", one flare referencing the financial institution and one flare relating to water. Moving through the nodes of the flares, I observed the samples progressing

[3]By close, we mean the standard Euclidean metric on $\mathbb{R}^{768}$



Mapper Summary

DATA BERT

FILTER FUNCTION PCA(n_components=2)

CLUSTER METHOD DBSCAN
euclidean,e=10, min=2

COVER 40

OVERLAP .5

NODES 67

EDGES 168

TOTAL SAMPLES 453

UNIQUE SAMPLES 130

COLOR FUNCTION Labels

NODE COLOR FUNCTION mean

NODE DISTRIBUTION

Figure 5.1: MAPPER image of vector embeddings for the word "Bank". Nodes are colored by encoding layer, with purple representing the initial layer and yellow for the last layer.

through the encoding layers. The nodes at the end of each flare strictly contained samples from the final two layers.

Ostensibly, MAPPER was able to visualize the bifurcation of the two meanings for the word "bank". It was able to express the fact that the initial embeddings are close, and diverge as the representations move through the layers. The topological features uncovered by MAPPER revealed a small insight into how BERT handles representations for ambiguous words.

As confirmation of how BERT's representations for ambiguous words diverge, I constructed an analogous visualization that highlights this same property. Similar to the filter function I used for MAPPER, I projected all 130 points onto the first three principal components. As it turns out, even in this low dimension visualization that suffers from projection loss, this divergence property of BERT was still observed.

Each point in figure 5.2 represents a hidden representation of the word "bank" projected into 3D space. Colored by sentence, the points from consecutive layers have been connected with a line. Observe how the points from the initial layer coincide. After the initial layer, the points diverge along two paths, one path for water, and one path for finance.

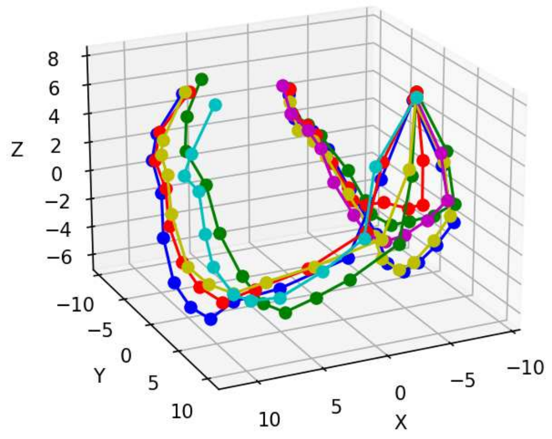


Figure 5.2: 3D PCA for "Bank" - The left "swoop" contains sentences referring to *water* and the right "swoop" contains sentences referring to *financial institutions.*

To further validate these findings, I decided to investigate the representations for another ambiguous word. To do so, I constructed another set of ten simple sentences containing the word "bat". Half of these sentences use the word "bat" to reference the animal and the other half refer to baseball. These sentences can be found in Table 5.3.

Table 5.3: Sample Sentences with "Bat"

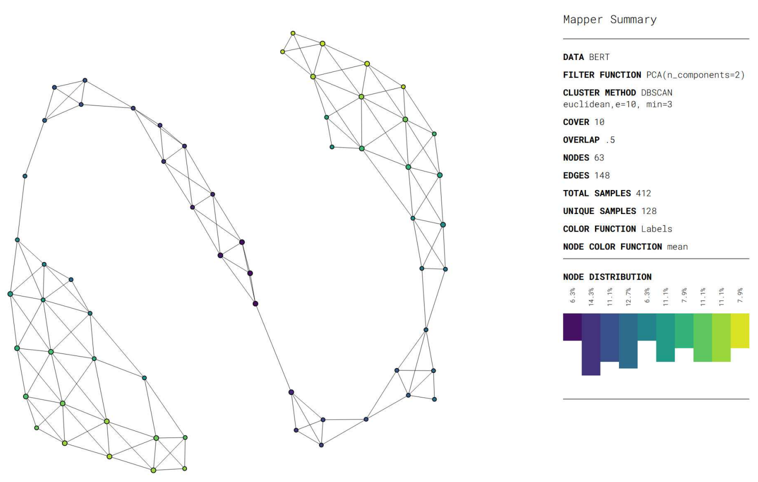
Sample Sentences with "Bat"	
Animal	Baseball
"The bat left the cave."	"She swung the baseball bat."
"Bats feed on insects and fruit."	"It was his second time at bat."
"Bats navigate by using sonar."	"This baseball bat is made of wood."
"The bat slept upside down."	"He preferred an aluminum bat."
"Bats are social animals."	"The player got in trouble for corking his bat."

The pre-processing and data pipelines for these new sentences containing the word "bat" were identical to the previous set of sentences containing the word "bank". First, each sentence was tokenized using the BERT Tokenizer. Then, each tokenized sentence was fed to the $BERT_{BASE}$

model as input. For each sentence, the hidden state of the word "bat" in each encoding layer was extracted. Again, Since $BERT_{BASE}$ has a hidden size of 768, these representations were vectors in $\mathbb{R}^{768}$. For each sentence, we obtained 13 vector representations, 1 representation for each of the 12 encoding layers plus an initial BERT token embedding. This again, produced a data set with 130 samples in $\mathbb{R}^{768}$.

Using this new set of points as input for MAPPER, I chose almost identical parameters as I used for the previous point cloud associated to the word "bank". The only difference involved a slight change in the clustering algorithm. I set the minimum number of samples needed to form a cluster from 2 to 3. This removed an unconnected component consisting of only two samples from the "animal" set in the very last encoding layers. My choice of parameters for the MAPPER algorithm are listed below:

- Filter Function - The point cloud was projected onto the first 2 principal components (2D PCA)
- Cover- The image of this projection was partitioned into 10 hyper-cubes with 50% overlap.
- Clustering Method- DBSCAN (metric = Euclidean, $\epsilon = 10$,min_samples = 3)



Mapper Summary

DATA BERT
FILTER FUNCTION PCA(n_components=2)
CLUSTER METHOD DBSCAN
euclidean,e=10, min=3
COVER 10
OVERLAP .5
NODES 63
EDGES 148
TOTAL SAMPLES 412
UNIQUE SAMPLES 128
COLOR FUNCTION Labels
NODE COLOR FUNCTION mean

NODE DISTRIBUTION

Figure 5.3: MAPPER image of vector embeddings for the word "Bat". Nodes are colored by encoding layer, with purple representing the initial layers and yellow for the last layers.

As we can see in Figure 5.3, MAPPER was able to detect the same behavior in the embedding layers of $BERT_{BASE}$ as before. The algorithm was able to visualize the bifurcation of the two meanings for the word "bat". Analogous to what we saw with "bank", the nodes representing early embedding layers (0-3) contained samples from both meanings of the word "Bat". Nodes containing samples from the later layers were only associated with a specific meaning of the word, either the animal or baseball.

In keeping with the analysis performed for the word "Bank", I constructed an analogous visualization to see if this behaviour could be observed with a PCA projection. Like before, I projected all 130 points onto the first three principal components. Again, even in this low dimension visualization that suffers from projection loss, this divergence property of BERT was still observed.

As seen in Figure 5.4, each point represents a hidden representation of the word "bat" projected into 3D space. Colored by sentence, the points from consecutive layers have been connected with a line. Observe how the points from the initial layer coincide. After the initial layer, the points diverge along two paths, one path for animal, and one path for baseball.

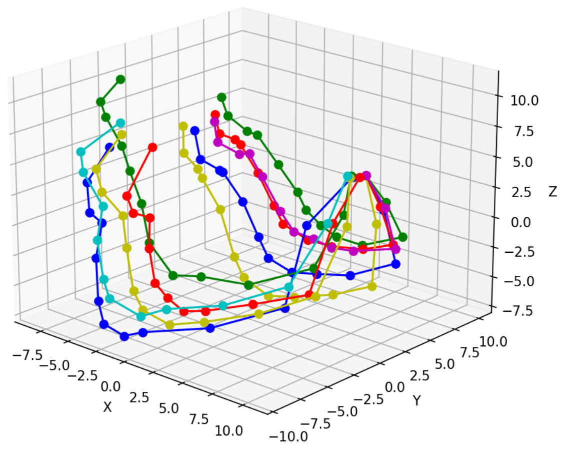


Figure 5.4: 3D PCA for "Bat" - The left "swoop" contains sentences referring to the *animal* and the right "swoop" contains sentences referring to *Baseball.*

5.3 MNIST Data set

Inspired by Garin et al. [4], I decided to explore the possibilities of applying TDA to the MNIST data set. While the authors of [4] used unique representations for the images, my implementation here is considerably more tame. That is, I use standard simplicial complexes and the Vietoris-Rips filtration, an approach more closely aligned with a traditional application of persistent homology.

The MNIST database contains 70,000 samples of hand written digits. The database is divided into two sets, 60,000 training images and 10,000 testing images. Each sample is a grey scale 28x28 pixel image representing a hand drawn digit. A breakdown of the MNIST dataset by digit can be found in table 5.4.

Table 5.4: Breakdown of the MNIST dataset.

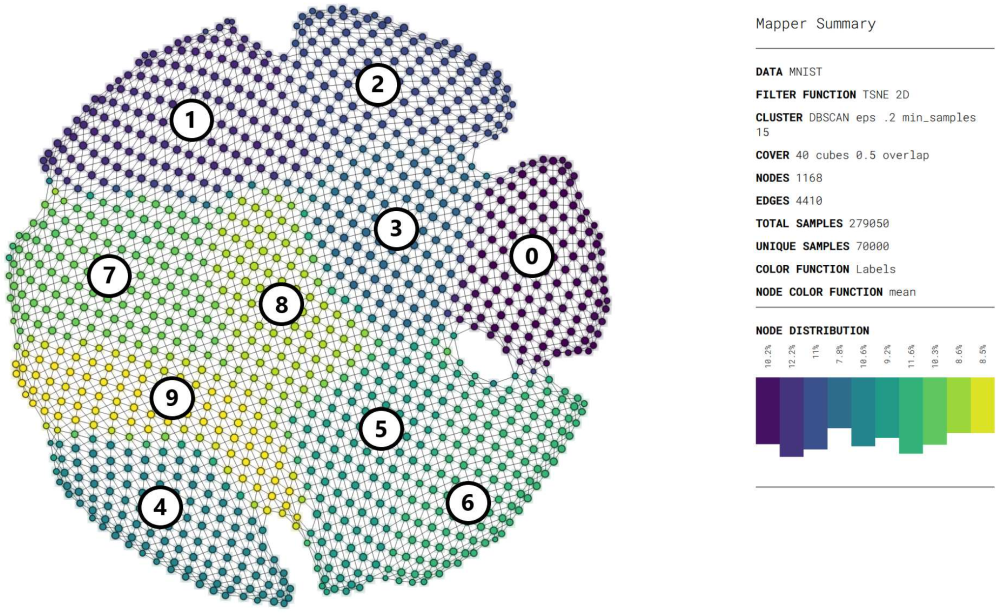
Digit	0	1	2	3	4	5	6	7	8	9	Total
Train	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949	60000
Test	980	1135	1032	1010	982	892	958	1028	974	1009	10000

Experiment: MAPPER

As a first step, I used MAPPER to explore the full MNIST data set. I combined both the training and testing sets into a single collection containing 70,000 samples. Each image was flattened by row into a single vector. Since the original samples were 28x28 pixel images, the resulting point cloud consisted of 70,000 points in $\mathbb{R}^{784}$. I applied the MAPPER algorithm with the following choice of parameters:

- Filter Function - 2D t-distributed stochastic neighbor embedding (t-SNE)
- Cover- The image of this projection was partitioned into 40 hyper-cubes with 50% overlap.
- Clustering Method- DBSCAN (metric = Euclidean, $\epsilon = .2$, min_samples = 15)

The graph produced by the MAPPER algorithm can see in figure 5.5.



Mapper Summary

DATA MNIST

FILTER FUNCTION TSNE 2D

CLUSTER DBSCAN eps .2 min_samples 15

COVER 40 cubes 0.5 overlap

NODES 1168

EDGES 4410

TOTAL SAMPLES 279050

UNIQUE SAMPLES 70000

COLOR FUNCTION Labels

NODE COLOR FUNCTION mean

NODE DISTRIBUTION

Figure 5.5: MAPPER algorithm applied to the full MNIST dataset. Nodes are colored by digit.

Experiment: Zeros and Eights

The initial idea for this experiment was to use persistent homology to classify the hand-written digits in the MNIST datset. Specifically, I sought to use the presence of 1-dimensional holes to distinguish between digits. However, unsure of exactly how well this feature would differentiate the digits in practice, I decided to begin with a specific subset of the digits.

Table 5.5: Breakdown of the MNIST dataset : zeros and eights.

Digit	0	8	Total
Train	5923	5851	11774
Test	980	974	1954

I chose to perform a binary classification of zeros and eights. Observe that both 8s and 0s have 1-dimension holes: a zero has a single 1-dimension hole and an eight has two 1-dimension holes. A breakdown of the zeros and eights in the MNIST dataset can be found in table 5.5.

My prediction was that these 1-dimensional holes would persist long enough to be a noticeable in a persistent diagram and the number of these holes would be enough for classification.

Pipeline

The raw gray scale images were first transformed into a binary image via a threshold value of 70%. The binary images were then embedded into $\mathbb{R}^2$ equipped with the standard Euclidean metric. From the resulting point cloud representations, I used persistent homology to computed the 2-dimensional points associated with the birth and death of 1-dimensional holes. Figure 5.6 shows an "8" moving through the pipeline where the final 2D points are displayed as a persistent diagram.

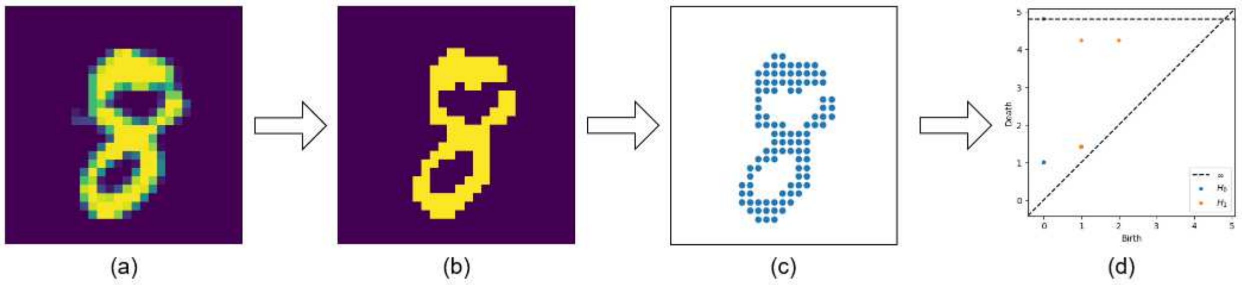


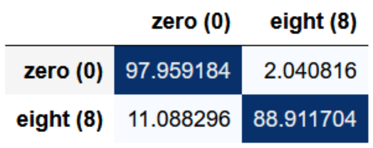
Figure 5.6: MNIST Pipeline (a) Raw Image (b) Binary Image (70% threshold) (c) Point Cloud image (d) Persistence Diagram.

The 1-dimensional holes extracted from the persistent diagram were then used as input for a fully connected Neural Network. As the number of 1-dimensional holes for a particular sample is variable, we “naively” found the sample containing the most holes (255) and padded the rest of the samples with zeros.

This resulted in a size of (255,2) for the input layer. This input layer was flattened and followed by a fully connected layer of 40 nodes. This was followed by a drop out layer with a value of 20%. Finally, the output layer consisted of a soft max layer with 2 nodes, one for each class of digit. ReLU was used as the activation function for the two hidden layers and the loss function used was Sparse Categorical Cross entropy. The model was trained for 50 epochs with a batch size of 1875.

Results: Zeros and Eights

As it turns out, using the birth and death of the 1-dimensional holes was particularly well suited for this binary classification problem. The model achieved an accuracy of 93% on the test data. The confusion matrix for this classification task is shown in figure 5.7.



	zero (0)	eight (8)
zero (0)	97.959184	2.040816
eight (8)	11.088296	88.911704

Figure 5.7: Confusion Matrix for Zeros and Eights

Experiment: Full MNIST data set

For this next experiment, I decide to expand my previous results and investigate the possibility of using persistent homology (1-dimensional holes) for the full multinomial classification of the MNIST data set (0-9).

As I alluded to earlier, I expected some difficulty in classifying certain digits. For example, would the model confuse the 1-dimensional hole present in a "6" or "9" for the 1-dimensional hole in a "0"? Furthermore, I expected the model to have a particularly difficult time distinguishing between a "6" and "9". These two digits are topologically equivalent and persistent homology is invariant under rotations. How would the model handle digits without 1-dimensional holes, like a "3" or a "7"?

Pipeline

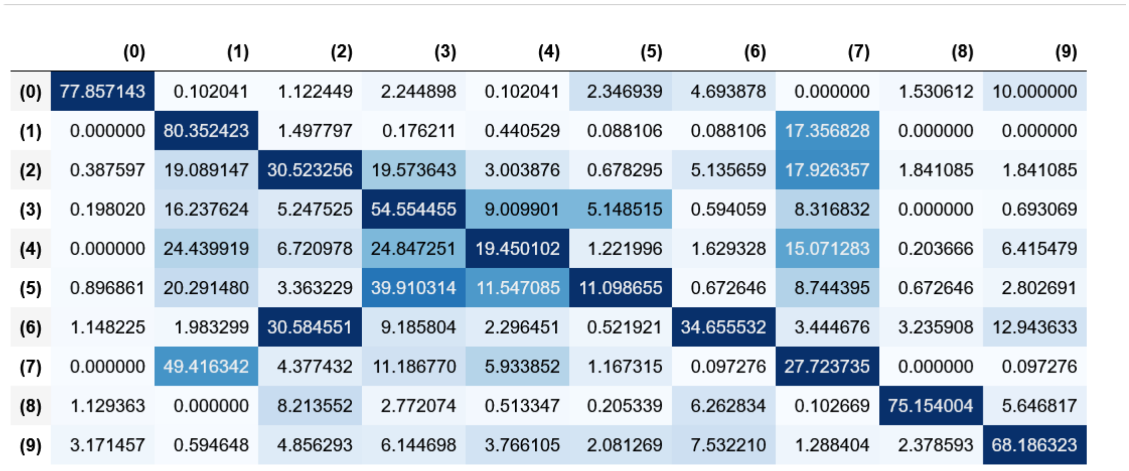
For consistency, the pipeline for this experiment was exactly the same as I used for the classification of eights and zeros. The raw images from the MNIST dataset were first transformed into a binary images via a threshold value of 70%. The binary images were then embedded into $\mathbb{R}^2$ to form a point cloud image. Persistent Homology was then applied to this embedding to detect topological features (1-dimensional holes). The birth and death points for the 1-dimensional holes were then used as input for a fully connected Neural Network.

To keep the input size consistent, the data was again padded with zeros. As a result, the input layer's size was (255,2). The input layer was then flattened and followed by a fully connected layer of 40 nodes. This was followed by a drop out layer with a value of 20%. Finally, the output layer consisted of 10 nodes, one for each class of digit, with softmax activation. ReLU was used as the activation function for the two hidden layers and the loss function used was Sparse Categorical Cross entropy. The model was trained for 50 epochs with a batch size of 1875.

Results: Full MNIST data set

Surprisingly, the model achieved an over all accuracy of 50.55% on the test data. The confusion matrix for this multinomial classification task is shown in figure 5.8. Observe the strong main diagonal in the confusion matrix. This was a better than expected result.

To start, the model was very effective at classifying 8's and 0's. This was highly anticipated considering the original motivating experiment (binary classification). As it turns out, using 1-dimensional holes as features provided enough of a distinctive signature (amongst the entire data set) for the models to classify these two digits.



	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(0)	77.857143	0.102041	1.122449	2.244898	0.102041	2.346939	4.693878	0.000000	1.530612	10.000000
(1)	0.000000	80.352423	1.497797	0.176211	0.440529	0.088106	0.088106	17.356828	0.000000	0.000000
(2)	0.387597	19.089147	30.523256	19.573643	3.003876	0.678295	5.135659	17.926357	1.841085	1.841085
(3)	0.198020	16.237624	5.247525	54.554455	9.009901	5.148515	0.594059	8.316832	0.000000	0.693069
(4)	0.000000	24.439919	6.720978	24.847251	19.450102	1.221996	1.629328	15.071283	0.203666	6.415479
(5)	0.896861	20.291480	3.363229	39.910314	11.547085	11.098655	0.672646	8.744395	0.672646	2.802691
(6)	1.148225	1.983299	30.584551	9.185804	2.296451	0.521921	34.655532	3.444676	3.235908	12.943633
(7)	0.000000	49.416342	4.377432	11.186770	5.933852	1.167315	0.097276	27.723735	0.000000	0.097276
(8)	1.129363	0.000000	8.213552	2.772074	0.513347	0.205339	6.262834	0.102669	75.154004	5.646817
(9)	3.171457	0.594648	4.856293	6.144698	3.766105	2.081269	7.532210	1.288404	2.378593	68.186323

Figure 5.8: Confusion matrix for the full MNIST data set

Before this experiment, it was not obvious whether the single 1-dimensional hole present in a "6" or a "9" might be a source of confusion for classifying 0's. Clearly, the model seldomly mistook 0's for a 6 or a 9. I suspect that the model was able to detect the size of the 1-dimensional hole with respect to the overall size of the image; The 1-dimensional hole in a 0 is much larger, respectively, than a 6 or a 9. That is, the holes/features in a 6 or a 9 would disappear quicker and not persist as long. Here, the "persistence" of the feature over a range of distances captured enough of the "global" structure for the models to learn. This is precisely the property of persistent homology we were expecting to exploit.

The model did better than expected when it came to the case of a 6 or 9. This is in spite of the fact that these digits are topologically equivalent and persistent homology is invariant under rotations. Nevertheless, the highest probability of misclassification for a 9 was a 6. Oddly enough, the highest probability of misclassification for a 6 was a 2.

The somewhat muddled results for 3,4, and 5 were expected. 3's and 5's do not have a "complete" 1-dimensional hole, and depending on how the 4 is drawn, it may or may not contain a 1-dimensional hole. Observe that at some distance in the persistent homology algorithm, these "almost" holes will be completed and show up as features. While these holes will quickly fill in, I suspect the model was able to learn a few distinguishing "local" features for 3,4, and 5. Although, these features were obviously not distinctive enough to produce a good classification.

Surprisingly, the model did very well in classifying 1's, even better than it did for 8's and 0's. However, the model was somewhat prone to confusing 1's and 7's. At first, it might appear that these digits should not contain any 1-dimensional holes. But, because of the way we embedded these images into R^2, holes did arise. Under our embedding map, pixels that are vertically or horizontally adjacent are mapped to points 1 unit apart. Pixels that are diagonally adjacent (common corner) are mapped to a distance of $\sqrt{2}$. As a consequence of this embedding, the first homology group for both 1's and 7's contained multiple features (1-dimensional holes) being born at a distance of 1 and dying at a distance of $\sqrt{2}$. In fact, the persistent diagrams for 1's and 7's mostly contained the point $(1, \sqrt{2})$ repeated multiple times.

Chapter 6

Conclusion

Topological Data Analysis is primarily concerned with the "shape" of data. The underlying conceit of TDA is that topological properties of a data set, like connectedness and the presence of holes, contain useful information. TDA uses this conceit to reveal structure in high dimensional data sets. In this thesis, I explored applications of TDA to Natural Language Processing and Computer Vision.

I discussed some notions from *algebraic topology* that provided a foundation for a working understanding of the tools and methods of TDA. In particular, I detailed the fundamental concept of n-dimensional holes and how *simplicial homology* detects their presence. Furthermore, I discussed how persistent homology expands this notion to handle point cloud data and extract features from multiple scales, from local to global.

In Chapter 4, I presented a few papers that applied the two main tools of TDA to problems in Natural Language Processing and computer vision. I selected these papers for their unique approaches and novel interpretations. For example, in the paper by Zhu [2], the features extracted by persistent homology were able to reveal semantic structure in text. In particular, semantic "tie-backs" were correlated to the presence of 1-dimensional holes.

This thesis has demonstrated that the topological features extracted from data sets can be used as input for traditional machine learning pipelines like image classification. Additionally, I've shown that the tools of TDA can be used to understand the machine algorithms themselves; contributing, for example, to the explainability of Neural Networks.

6.1 Future Work

The directions for future investigations that I would like to highlight here are related to my experiment where I used the MAPPER algorithm to explore how the BERT transformer model handles representations of polysemous words as they move through its encoding layers. This work has several natural extensions.

First, this experiment was preformed on the pre-trained $BERT_{BASE}$ model introduced in [15]. The most obvious extension to this work, similar to [11], would be to start with an untrained model and explore how the representations in the encoding layers evolve from scratch. However, training a BERT transformer model turns out to computationally expensive. So much so, that it might be prohibitive to any person or team without the resources of a large company like Google.

As a comprise, perhaps one could explore the effects of fine tuning a BERT model, comparing the pre-trained model representations to the ones produced after fine tuning BERT for a particular task.

Another future direction to explore is the evolution of the attention mechanisms. Probing the BERT model to see what contextual clues are contributing the most to the meaning of ambiguous words. My experiment only examined the hidden representations in the encoders, not the attention heads themselves.

As for my experiment that used persistent homolgy to classify MNIST images, I believe using filtrations similar to those found in [4] would greatly improve performance. For instance, a filtration constructed from a top to bottom "sweep" would ostensibly be able to detect the difference between 6 and a 9. In this vertical "sweep", the 1-dimensional hole present in a 9 would appear earlier. While the 1-dimensional hole present in a 6 would not appear until the sweep reached the bottom of the digit. I also suspect that a horizontal "sweep" might help differentiate a 1 from a 7.

Bibliography

[1] Allen Hatcher. *Algebraic topology*. Cambridge Univ. Press, Cambridge, 2002.

[2] Xiaojin Zhu. Persistent homology: An introduction and a new text representation for natural language processing. pages 1953–1959, 08 2013.

[3] Afra Zomorodian and Gunnar Carlsson. Computing persistent homology. *Discrete Comput. Geom.*, 33(2):249–274, feb 2005.

[4] Adélie Garin and Guillaume Tauzin. A topological "reading" lesson: Classification of mnist using tda, 2019.

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

[6] Georges Reeb. Sur les points singuliers d'une forme de pfaff completement integrable ou d'une fonction numerique [on the singular points of a completely integrable pfaff form or of a numerical function]. *Comptes Rendus Acad. Sciences Paris*, 222:847–849, 1946.

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

[8] Shafie Gholizadeh, Ketki Savle, Armin Seyeditabari, and Wlodek Zadrozny. Topological data analysis in text classification: Extracting features with additive information. *ArXiv*, abs/2003.13138, 2020.

[9] Geoffrey Sampson. The structure of children's writing: moving from spoken to adult written norms. *Language and Computers*, pages 177–193, 10 2003.

[10] Madjid Allili, K. Mischaikow, and Allen Tannenbaum. Cubical homology and the topological classification of 2d and 3d imagery. volume 2, pages 173 – 176 vol.2, 11 2001.

[11] Maxime Gabella. Topology of learning in feedforward neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(8):3588–3592, 2021.

[12] Rickard Gabrielsson and Gunnar Carlsson. Exposition and interpretation of the topology of neural networks. pages 1069–1076, 12 2019.

[13] Ken Lang. Newsweeder: Learning to filter netnews. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 331–339, 1995.

[14] 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.

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[17] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[18] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.

[19] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[20] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.