Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Bendik Løvlie

Text Classification via Topological Data Analysis

Master's thesis in MLREAL
Supervisor: Øyvind Solberg
June 2023

NTNU
Norwegian University of
Science and Technology

Bendik Løvlie

Text Classification via Topological Data Analysis

Master's thesis in MLREAL
Supervisor: Øyvind Solberg
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

Abstract

Recent generative language models such as ChatGPT have demonstrated a remarkable capacity to generate texts that are difficult to distinguish from human-written texts. Topological Data Analysis (TDA) is concerned with the shape of data and can extract significant qualitative and quantitative information from complex datasets. This thesis provides an introduction to the mathematics underlying the main tool of TDA, persistent homology. We then present techniques from the field of Natural Language Processing (NLP) that can transform texts to collections of vectors whose shapes may be analyzed using TDA. Finally, we explore the application of TDA to tasks involving detection of machine-generated texts.

Preface

This thesis marks the completion of my Master's degree in the Natural Science with Teacher Education (MLREAL) programme at the Norwegian University of Science and Technology (NTNU). The work has been carried out during the spring of 2023 at the Department of Mathematical Sciences.

As a future science teacher, this work has been relevant for multiple reasons: Being able to explain complicated subject matter in an understandable way is an essential skill to have as a teacher. Secondly, this work has involved a lot of programming, which plays an increasingly important role in schools across the world. Lastly, this work deals with recent developments in the area of artificial text generation, which in an educational context provide many new opportunities and challenges.

I wish to express my gratitude to my supervisor Øyvind Solberg for his guidance and encouragement this semester. His feedback has been very helpful.

Contents

1 Introduction **1**

2 Theory **3**
2.1 Topological Data Analysis 3
2.1.1 Simplicial Complexes 3
2.1.2 Simplicial Homology 6
2.1.3 Induced Maps of Homology 9
2.1.4 Persistent Homology 12
2.1.5 The Persistence Algorithm 16
2.1.6 Persistence Images 18
2.2 Natural Language Processing 20
2.2.1 Text Vectorization 20
2.2.2 Text Preprocessing 24
2.2.3 TDA on Text 24
2.3 Related Work 27

3 Experiments **29**
3.1 Method 29
3.2 Experiment: Webtext & GPT-2 32
3.3 Experiment: ChatGPT Essays 37
3.4 Discussion 40

4 Conclusion **43**

A Implementation **45**
A.1 Text Preprocessing 45
A.2 Text Vectorization 47
A.3 Metric on Text Embeddings 49
A.4 SIF and SIFTS 50
A.5 Machine Learning Pipeline 51
A.6 TDA Pipeline Demonstration 55

Chapter 1

Introduction

Modern generative language models, such as ChatGPT, have an astounding capacity to generate texts that are difficult to differentiate from those produced by humans, and can be misused to generate fake news, cheat on exams and even produce abusive content. The rise of models replicating human language requires us to develop methods for detection of texts that have been artificially written.

This thesis explores the application of Topological Data Analysis (TDA) to machine-generated text classification tasks. The objective is to provide an accessible introduction to the fields of Topological Data Analysis and Natural Language Processing (NLP) while demonstrating an application that intersects these two fields within a machine learning context. To ensure transparency, we offer an easily accessible implementation of the code used in the experiments. It is our hope that this implementation will inspire and contribute to further experiments in the field.

The text is intended to be a self-contained introduction, and is written in a way that requires minimal prerequisites. In particular, little mathematical knowledge is required beyond basic linear algebra.

Chapter 2

Theory

2.1 Topological Data Analysis

Topological Data Analysis (TDA) is an area of applied mathematics that studies the "shape" of data using ideas from topology. It is based on the premise that topological features of datasets such as holes, voids, and connected components contain important information that may be used to expose the structure of complicated high-dimensional data. In this section we develop the theory of the main tool in TDA, persistent homology. The presented theory is based on books on persistent homology by Edelsbrunner & Harer [9], Virk [25], and on the lecture notes on Topological Data Analysis by Botnan [4].

2.1.1 Simplicial Complexes

Much of the data we would like to study takes the form of *point cloud data*, a finite number of discrete samples in some metric space. Because topological and geometric features most often are associated with continuous spaces, point clouds of data do not provide any topological information in and of themselves.

To be able to infer some topological structure of data, a natural technique is to "connect" data points that are sufficiently near each other to create continuous objects that may reveal structure underlying the data. This is commonly accomplished using relatively simple shapes called *simplicial complexes.* In this section, we will look at simplicial complexes in general and one specific construction of complexes that can be made from point cloud data: the Vietoris Rips complex. We begin by defining the basic building blocks of these simple shapes, called *simplices.*

Definition 2.1.1. A set of points $\{p_0, \ldots, p_n\} \subset \mathbb{R}^d$ is **geometrically independent** if $\{p_1 - p_0, \ldots, p_n - p_0\}$ is a linearly independent subset of the real vector space $\mathbb{R}^d$.

Definition 2.1.2. Let $\{p_0, \ldots, p_n\}$ be geometrically independent set in $\mathbb{R}^d$. The **n-simplex** σ spanned by the points p_i is the set of all points $x \in \mathbb{R}^d$ such that

$$x = \sum_{i=0}^{n} t_i p_i,$$

where $\sum_{i=0}^{n} t_i = 1$, and $t_i \geq 0$ for all i. This set is denoted by $\sigma = [p_0, p_1, \ldots, p_n]$ and the points $\{p_0, p_1, \ldots, p_n\}$ are called the **vertices** of σ.

By definition, an n-simplex is an n-dimensional version of a filled-in triangle. For example, a 0-simplex is a point, a 1-simplex is a closed interval, a 2-simplex is a triangle with the interior, a 3-simplex is a tetrahedron with the interior, and so on.

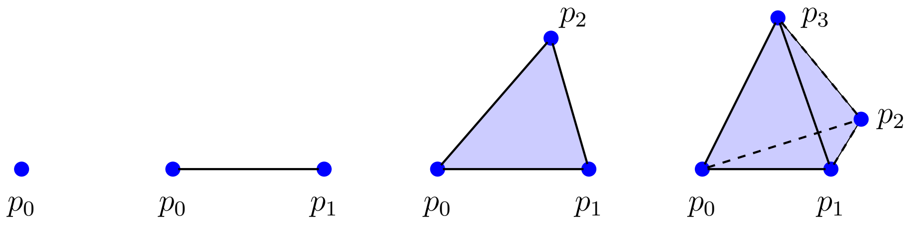


Figure 2.1: Examples of simplices of dimensions 0, 1, 2 and 3.

Next, we will fix some terminology. We say that a simplex τ is a **face** of σ if τ is spanned by a subset of the vertices of σ. This we denote by $\tau \subseteq \sigma$, or $\tau \subset \sigma$ if τ is a **proper face** of σ, meaning that $\tau \neq \sigma$. The **boundary** of σ is the union of all proper faces of σ and is denoted by $\mathrm{Bd}(\sigma)$.

Example 2.1.1. Consider the 2-simplex $\sigma = [p_0, p_1, p_2]$. Then the 1-simplex $\tau = [p_0, p_2]$ is an example of a proper face of σ. The boundary of σ is $\mathrm{Bd}(\sigma) = [p_0, p_1] \cup [p_1, p_2] \cup [p_0, p_2]$.

We are now ready to define simplicial complexes, which are collections of simplices glued together along their respective faces:

Definition 2.1.3. A **simplicial complex** K in $\mathbb{R}^d$ is a finite collection of simplices in $\mathbb{R}^d$ that has the following two properties:

1. Every face of a simplex of K is in K.
2. The non-empty intersection of any two simplices of K is a face of both simplices.

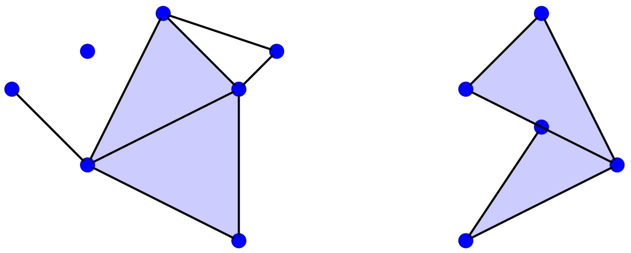


Figure 2.2: The left figure is a simplicial complex. The right figure is not a simplicial complex, since the intersection of the two 2-simplices only is a face of the lower 2-simplex, and not the upper.

As we can see in Definition 2.1.2, a simplex is defined by listing its vertices, and hence a simplicial complex may be defined as a list of points corresponding to its simplices. In many contexts, the precise shape of a simplicial complex is not of major importance, which motivates a simpler, purely combinatorial description of a simplicial complex:

Definition 2.1.4. An **abstract simplicial complex** is a finite collection K of finite non-empty sets such that if α is an element of K, then so is every nonempty subset of α.

The elements of K are called **simplices**. The vertices (0-simplices) of K are the one-point elements of K, and the **edges** (1-simplices) are the two-point elements of A. In general, a simplex $\sigma \in K$ is an n-simplex in K if the set σ consists of $n+1$ elements. Similarly to in the previous definition of simplicial complex, a simplex $\tau \in A$ is a face of $\sigma \in A$ if $\tau \subseteq \sigma$.

We say that $B \subseteq A$ is a **subcomplex** of A if B is an abstract simplicial complex. Moreover, we define the **dimension** of A to be the maximal dimension of its simplices.

Example 2.1.2. The following set determines an abstract simplicial complex:

$$\begin{aligned}\sigma &= [a,b,c] \cup [c,d] \\ &= \{\{a\},\{b\},\{c\},\{a,b\},\{a,c\},\{b,c\},\{a,b,c\}\} \cup \{\{c\},\{d\},\{c,d\}\} \\ &= \{\{a\},\{b\},\{c\},\{d\},\{a,b\},\{a,c\},\{b,c\},\{c,d\},\{a,b,c\}\}\end{aligned}$$

This simplicial complex is of dimension 2 because the largest simplex in σ is $\{a,b,c\}$, consisting of 3 elements, which thus corresponds to a 2-simplex. The following figure serves as a visual representation σ:

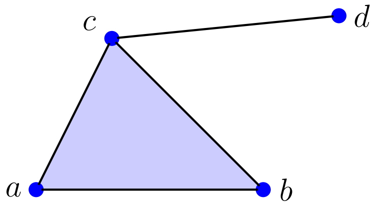


Figure 2.3: An illustration of the abstract simplicial complex σ.

Like suggested by the above example, every simplicial complex determines an abstract simplicial complex by identifying a geometric simplex with the vertices that span it. Due to their combinatorial nature, abstract simplicial complexes are better suited for calculations on a computer, which is why we will consider these in the theoretical constructions that follow later in the text.

Simplicial Complexes from Data

As previously mentioned, the datasets we want to analyze usually come in the form of a point cloud, a finite set of discrete points embedded in some metric space. There are a variety of ways to construct simplicial complexes from point cloud data, but among the more widely used is the following:

Definition 2.1.5. Let X be a pseudo-metric space with pseudo-distance D (i.e. D satisfies all the axioms of a formal distance, except that $D(x,y) = 0$ need not imply $x = y$). Then the **Vietoris Rips complex** of X at scale $\epsilon \geq 0$, denoted $\mathrm{VR}_\epsilon(X)$, is the abstract simplicial complex defined by the following condition [13]:

$$[x_0, \ldots, x_k] \in \mathrm{VR}_\epsilon(X) \text{ for } x_i \in X \iff D(x_i, x_j) \leq \epsilon \text{ for all } i,j \in \{0, \ldots, k\}\,.$$

In other terms, the Vietoris Rips complex is dependent only on the scale parameter ϵ. There is an edge between two points in the complex if and only if the distance between the locations is at most ϵ. There is a triangular face between any three points whose pairwise distances are at most ϵ, and so on. Figure 2.4 depicts four Vietoris Rips complexes constructed from the same base point cloud but with varying size parameters. For illustration purposes we have drawn balls of radius $\epsilon/2$ around each data point, implying that if the balls around a set of k points have pairwise nonempty intersections, these k points span a $(k-1)$-simplex.

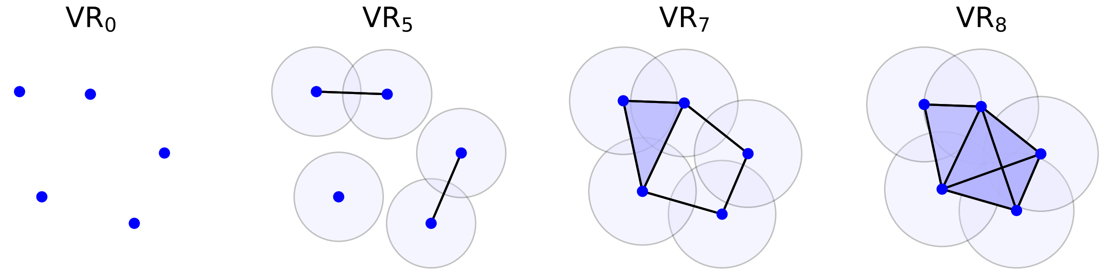


Figure 2.4: Four Vietoris Rips complexes built from a point cloud in $\mathbb{R}^2$, for varying values of the scale parameter ϵ. Here we have let the metric be the usual Euclidian distance.

The idea is that creating a Vietoris Rips complex from a dataset may somehow reveal details about the underlying shape of the data. A natural question that might arise, is what choice of scale ϵ will highlight the most essential features of a particular dataset? The insight of persistent homology, to be defined later in the text, is that it is not sufficient to consider any fixed value of ϵ. However, examining how topological features develop across a variety ϵ-values might uncover important details about the shape of a dataset.

2.1.2 Simplicial Homology

Having defined simplicial complexes, we now wish to develop a method that extracts geometric information from these objects. Here we introduce the concept of simplicial homology, which detects holes in a given simplicial complex. The idea of homology is that it captures holes in a space not by focusing on the holes directly, but by focusing on their surroundings. Defining this rigorously requires some algebraic machinery, that involves building vector spaces from simplicial complexes.

Definition 2.1.6. Let $n \in \mathbb{N}$ and let K be an abstract simplicial complex. The **vector space of n-chains** in K is the $\mathbb{Z}_2$-vector space $C_n(K)$ such that its basis elements are all n-simplices of K. Let K^n denote the set of all n-simplices of K. Then an arbitrary element of $C_n(K)$, an n-chain, is a linear combination of n-simplices, and can be uniquely written as

$$\sum_{\sigma \in K^n} a_\sigma \sigma, \text{ where } a_\sigma \in \mathbb{Z}_2.$$

We define the boundary of an n-simplex to be the formal sum of its $(n-1)$-dimensional faces. For an n-simplex $[p_0, \dots, p_n]$, let $[p_0, \dots, \hat{p}_i, \dots, p_n]$ denote the $(n-1)$-simplex obtained by omitting the vertex p_i. We may extend the idea of boundaries to n-chains in the following way:

Definition 2.1.7. The **boundary map** $\partial_n \colon C_n(K) \to C_{n-1}(K)$ is the linear transformation defined on simplices by

$$\partial_n\left([p_0, \dots, p_n]\right) = \sum_{i=0}^{n} [p_0, \dots, \hat{p}_i, \dots, p_n] .$$

Thus, we have defined the boundary of an n-chain $c = \sum a_\sigma \sigma$ to be $\partial_n(c) = \sum a_\sigma \partial_n(\sigma)$. An essential fact for the algebraic formulation of homology is that the composition of two boundary maps is zero. Intuitively, the following lemma states that a boundary does not have a boundary:

Lemma 2.1.1. *The composition $\partial_n \circ \partial_{n+1} = 0$ for all $n \geq 0$.*

Proof. By linearity it is sufficient to prove that $\partial_n \circ \partial_{n+1}(\sigma) = 0$ for an $n+1$-simplex $\sigma = [p_0, \dots p_{n+1}]$. From the definition of the boundary map we get that

$$\begin{aligned} \partial_n \circ \partial_{n+1}(\sigma) &= \partial_n \left(\sum_{i=0}^{n+1} [p_0, \dots, \hat{p}_i, \dots, p_{n+1}] \right) \\ &= \sum_{j=0, j\neq i}^{n+1} \sum_{i=0}^{n+1} [p_0, \dots, \hat{p}_i, \dots, \hat{p}_j, \dots, p_{n+1}] . \end{aligned}$$

For $j \neq i$, we see that the $n-1$-simplex $[p_0, \dots, \hat{p}_i, \dots, \hat{p}_j, \dots, p_{n+1}]$ appears exactly two times in the sum. Since our coefficients are in $\mathbb{Z}_2$, we conclude that $\partial_n \circ \partial_{n+1}(\sigma) = 0$. Thus $\partial_n \circ \partial_{n+1} = 0$ for all $n \geq 0$. □

Using the definition of the boundary map we now define two particularly interesting types of n-chains. We say that $c \in C_n(K)$ is an n-**cycle** if it has no boundary, i.e. if $\partial_n(c) = 0$. The set of n-cycles form a vector space, which is denoted by $Z_n(K) = \ker \partial_n$. An n-chain b is an n-**boundary** if it is the boundary of an $n+1$-chain, meaning that $b = \partial_{n+1}(d)$ for some $d \in C_{n+1}(K)$. The associated vector space of n-boundaries is denoted by $B_n(K) = \operatorname{im} \partial_{n+1}$. A consequence of the above lemma is that all boundaries are cycles.

Corollary 2.1.2. *All n-boundaries are n-cycles, i.e. $B_n(K) \subseteq Z_n(K)$ for all $n \geq 0$.*

Note that converse is untrue: not all cycles are boundaries. Since $B_n(K)$ is a subspace of $Z_n(K)$, we may form the quotient space $Z_n(K)/B_n(K)$ whose non-zero elements are cycles that are not boundaries. This is precisely the definition of simplicial homology:

Definition 2.1.8. The n**-th simplicial homology vector space** of a simplicial complex K is the quotient space

$$H_n(K) = \frac{Z_n(K)}{B_n(K)}.$$

Qualitatively, the basis elements of the homology vector spaces are cycles that are not boundaries. An interpretation of homology is that a basis element in $H_n(K)$ corresponds to an n-dimensional hole in the simplicial complex [25] (with a 0-dimensional hole representing a connected component). Do note that homology detects holes indirectly, by actually detecting a class of cycles that enclose a hole.

Example 2.1.3. Let us now calculate the homology vector spaces of a simple example. Let K be the abstract simplicial complex depicted in Figure 2.5. We want to calculate the homology vector spaces of K. We immediately conclude that $H_r(K) = \mathbf{0}$ for all $r < 0$ and $r > 2$, because the simplicial complex only consists of simplices of dimensions $0, 1$ and 2.

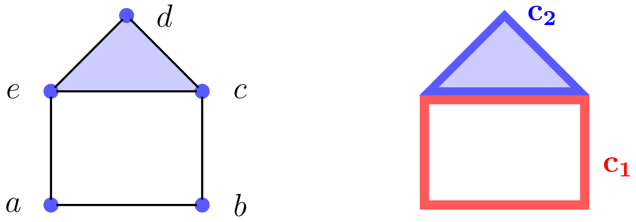


Figure 2.5: Left: The simplicial complex. Right: Two 1-cycles forming a basis for $Z_1(K)$.

The simplices of dimensions $0, 1$ and 2 form the basis elements for the vector spaces of chains $C_0(K)$, $C_1(K)$ and $C_2(K)$ respectively, and thus we have that

$$\begin{aligned} C_0(K) &= \operatorname{span}\{\{a\},\{b\},\{c\},\{d\},\{e\}\} && \cong \mathbb{Z}_2^5 \\ C_1(K) &= \operatorname{span}\{\{a,b\},\{b,c\},\{c,d\},\{d,e\},\{c,e\},\{a,e\}\} && \cong \mathbb{Z}_2^6 \\ C_2(K) &= \operatorname{span}\{\{c,d,e\}\} && \cong \mathbb{Z}_2 \end{aligned}$$

Let us first compute $H_1(K)$, which counts the number of holes in the simplicial complex. Since the complex has one hole, we expect $H_1(K)$ to be $\mathbb{Z}_2$. To find $Z_1(K) = \ker \partial_1$ we solve

$$\partial_1 \left(\lambda_1\{a,b\} + \lambda_2\{b,c\} + \lambda_3\{c,d\} + \lambda_4\{d,e\} + \lambda_5\{c,e\} + \lambda_6\{a,e\}\right) = 0$$

Evaluating the boundary map and rewriting in terms of the basis elements of $C_0(K)$, we get

$$(\lambda_6 + \lambda_1)\{a\} + (\lambda_1 + \lambda_2)\{b\} + (\lambda_2 + \lambda_3 + \lambda_5)\{c\} + (\lambda_3 + \lambda_4)\{d\} + (\lambda_4 + \lambda_5 + \lambda_6)\{e\} = 0$$

Because the basis elements $\{\{a\},\{b\},\{c\},\{d\},\{e\}\}$ are linearly independent, each of the coefficients must be zero, corresponding to the following system of equations:

$$\begin{cases} \lambda_1 + \lambda_6 & = 0 \\ \lambda_1 + \lambda_2 & = 0 \\ \lambda_2 + \lambda_3 + \lambda_5 & = 0 \\ \lambda_3 + \lambda_5 & = 0 \\ \lambda_4 + \lambda_5 + \lambda_6 & = 0 \end{cases} \iff \begin{cases} \lambda_1 = \lambda_2 = \lambda_6 \\ \lambda_3 = \lambda_4 \\ \lambda_5 = \lambda_1 + \lambda_3 \end{cases}.$$

From the equations we can see that there are two free variables, and thus that the solution space $Z_1(K)$ is spanned by $\{(1,1,0,0,1,1),(0,0,1,1,1,0)\}$. So $Z_1(K) \cong \mathbb{Z}_2^2$ is generated by the two 1-chains

$$\begin{aligned} c_1 &= \{a,b\} + \{b,c\} + \{c,e\} + \{a,e\}, \\ c_2 &= \{c,d\} + \{d,e\} + \{c,e\}. \end{aligned}$$

Next we compute $B_1(K)$ by calculating the image of ∂_2:

$$B_1(K) = \operatorname{im}(\partial_2) = \operatorname{span}\{\partial_2(\{c,d,e\}\} = \operatorname{span}\{\{c,d\} + \{d,e\} + \{c,e\}\} \cong \operatorname{span}\{c_2\}.$$

We have shown that $Z_1(K)$ is generated by the two cycles c_1 and c_2. The basis elements of $H_1(K)$ is however the class of cycles that are not also boundaries, and as we have just showed, c_2 is a boundary. Therefore, there is only one non-trivial class of cycles in $H_1(K)$, corresponding to the one hole in the simplicial complex. We get

$$H_1(K) = \frac{Z_1(K)}{B_1(K)} = \frac{\operatorname{span}\{\{c_1, c_2\}\}}{\operatorname{span}\{\{c_2\}\}} = \operatorname{span}\{[c_1]\} \cong \mathbb{Z}_2.$$

Let us now calculate $H_0(K)$. Right away we can see that $Z_0(K) = \ker \partial_0 \cong C_0(K)$ because the 0-boundary map sends each element to 0. Now we calculate the 0-boundaries:

$$\begin{aligned} B_0(K) &= \operatorname{im}(\partial_1) \\ &= \operatorname{span}\{\partial_1(\{a,b\}),\ \partial_1(\{b,c\}),\ \partial_1(\{c,d\}),\ \partial_1(\{d,e\}),\ \partial_1(\{c,e\}),\ \partial_1(\{a,e\})\} \\ &= \operatorname{span}\{\{a\}+\{b\},\ \{b\}+\{c\},\ \{c\}+\{d\},\ \{d\}+\{e\},\ \{c\}+\{e\},\ \{a\}+\{e\}\} \\ &= \operatorname{span}\{\{a\}+\{b\},\ \{b\}+\{c\},\ \{c\}+\{d\},\ \{d\}+\{e\}\} \\ &\cong \mathbb{Z}_2^4 \end{aligned}$$

We then get that

$$H_0(K) = \frac{Z_0(K)}{B_0(K)} = \frac{\operatorname{span}\{\{a\},\{b\},\{c\},\{d\},\{e\}\}}{\operatorname{span}\{\{a\}+\{b\},\{b\}+\{c\},\{c\}+\{d\},\{d\}+\{e\}\}} = \operatorname{span}\{[\{a\}]\} \cong \mathbb{Z}_2,$$

where now in homology $\{a\}$, {b}, {c}, {d}, and {e} all represent the same class of cycles because they differ by a boundary, and together correspond to one single connected component in the simplicial complex.

Lastly we calculate $H_2(K)$. Since there are no 3-simplices in the simplicial complex, there are no 3-chains, and so we get that $B_2(K) = \operatorname{im}(\partial_3) = \mathbf{0}$. Furthermore, the only basis element of $C_2(K)$ gets mapped to a non-zero element of $C_1(K)$, and thus $Z_2(K) = \ker \partial_2 = \mathbf{0}$. Thus the homology vector space of dimension 2 is $H_2(K) = Z_n(K)/B_n(K) = \mathbf{0}/\mathbf{0} = \mathbf{0}$. In conclusion, we have that

$$H_n(K) = \begin{cases} \mathbb{Z}_2 & n = 0, 1 \\ \mathbf{0} & \text{otherwise.} \end{cases}$$

Remark. It should be clear that performing a direct calculation of homology as the prior example will become increasingly difficult as the number of simplices grows. The vector spaces and basis elements of $H_n(K)$ can be calculated in a more systematic way in the form of a Gaussian matrix reduction of so-called "boundary matrices", but this too will quickly become too advanced to do by hand. In practice, we leave computation of simplicial homology to our computers.

2.1.3 Induced Maps of Homology

In the previous section we defined the simplicial homology vector spaces of a simplicial complex. We will now show that the idea of simplicial homology can be extended in a consistent manner to maps between simplicial complexes, which is an essential notion in the definition of persistent homology.

Definition 2.1.9. Let K and L be abstract simplicial complexes. A **simplicial map** $f\colon K \to L$ is a function such that for every simplex $[p_0, \dots, p_m]$ in K, $[f(p_0), \dots, f(p_m)]$ is a simplex in L.

A simplicial map $f\colon K \to L$ induces a linear map of n-chains in all dimensions:

$$\tilde{f}_n\colon C_n(K) \longrightarrow C_n(L), \qquad \tilde{f}_n(\sigma) = \begin{cases} f(\sigma) & \text{if } \dim f(\sigma) = n, \\ 0 & \text{otherwise.} \end{cases}$$

While the fact that a simplicial map induces a linear map between n-chains is straightforward, the fact that a simplicial map induces a well-defined map in homology is not immediately obvious. To prove this, we rely on the following lemma:

Lemma 2.1.3. *Let $f\colon K \to L$ be a simplicial map. Then the following diagram commutes:*

$$\begin{tikzcd}
C_n(K) \arrow[r, "\partial_n"] \arrow[d, "\tilde{f}_n"'] & C_{n-1}(K) \arrow[d, "\tilde{f}_{n-1}"] \\
C_n(L) \arrow[r, "\partial'_n"] & C_{n-1}(L)
\end{tikzcd}$$

That is, we have that $\partial'_n \circ \tilde{f}_n = \tilde{f}_{n-1} \circ \partial_n$, for all $n \geq 1$.

Proof. It is sufficient to prove the result for a single n-simplex $\sigma = [p_0, \dots, p_n]$, and the result will hold for n-chains by linearity. In the proof we have to be aware of the difference between the simplicial map f and the induced linear maps $\tilde{f}_k$. The specific calculations we have to perform to prove the commutativity of the diagram depends on the dimension of the simplex $f(\sigma)$, and we consider the following three cases:

Case 1: $\dim f(\sigma) = n$. This implies that the dimension of $f([p_0, \dots, \hat{p}_i, \dots, p_n])$ is $n-1$ for all i, and the following holds:

$$\begin{aligned}
\tilde{f}_{n-1} \circ \partial_n(\sigma) &= \sum_{i=0}^{n} \tilde{f}_{n-1}\left([p_0, \dots, \hat{p}_i, \dots, p_n]\right) \\
&= \sum_{i=0}^{n} [f(p_0), \dots, \widehat{f(p_i)}, \dots, f(p_n)] \\
&= \partial'_n\left([f(p_0), \dots, f(p_n)]\right) \\
&= \partial'_n \circ \tilde{f}_n\big([p_0, \dots, p_n]\big) \\
&= \partial'_n \circ \tilde{f}_n(\sigma).
\end{aligned}$$

Case 2: $\dim f(\sigma) \leq n-2$. It is immediate from the assumption that $\partial'_n \circ \tilde{f}_n(\sigma) = 0$, because $\tilde{f}_n(\sigma) = 0$. Furthermore, it follows from the assumption that

$$\dim\big(f([p_0, \dots, \hat{p}_i, \dots, p_n])\big) \leq \dim f(\sigma) = n-2 \quad \text{for all } i.$$

We thus get the following:

$$\tilde{f}_{n-1} \circ \partial_n(\sigma) = \sum_{i=0}^{n} \tilde{f}_{n-1}\left([p_0, \dots, \hat{p}_i, \dots, p_n]\right) = \sum_{i=0}^{n} 0 = 0 = \partial'_n \circ \tilde{f}_n(\sigma).$$

Case 3: $\dim f(\sigma) = n-1$. It is immediate from the assumption that $\partial'_n \circ \tilde{f}_n(\sigma) = 0$, because $\tilde{f}_n(\sigma) = 0$. Since the simplex σ has n vertices, while the simplex $f(\sigma)$ has $n-1$ vertices, there must be two vertices in σ that get mapped by f to the same vertex of $f(\sigma)$. We may therefore assume without loss of generality that $f(p_0) = f(p_1)$. It follows that

$$\dim\big(f([p_0, \dots, \widehat{p_i}, \dots, p_n])\big) \leq n-2 \quad \text{for } 2 \leq i \leq n$$

We thus get that

$$\begin{aligned}
\tilde{f}_{n-1} \circ \partial_n(\sigma) &= \sum_{i=0}^{n} \tilde{f}_{n-1}\left([p_0, \dots, \hat{p}_i, \dots, p_n]\right) \\
&= [f(p_0), f(p_2), \dots, f(p_n)] + [f(p_1), f(p_2), \dots, f(p_n)] \\
&= 0 \\
&= \partial'_n \circ \tilde{f}_n(\sigma).
\end{aligned}$$

We have now proved that $\partial'_n \circ \tilde{f}_n = \tilde{f}_{n-1} \circ \partial_n$ holds for any basis element σ of $C_n(K)$, and the result of the lemma follows. □

Having established the commutative property between the linear map of n-chains $\tilde{f}$ and the boundary map ∂_n, we are now ready to prove the existence of an induced map in homology.

Corollary 2.1.4. *A simplicial map $f\colon K \to L$ induces a well-defined map in homology,*

$$\begin{aligned}
f_n\colon H_n(K) &\to H_n(L), \\
[c] &\mapsto [\tilde{f}_n(c)].
\end{aligned}$$

Proof. Since $H_n(K) = Z_n(K) \,/\, B_n(K)$ and $H_n(L) = Z_n(L) \,/\, B_n(L)$ we need to show that $\tilde{f}_n(Z_n(K)) \subseteq Z_n(L)$ and $\tilde{f}_n(B_n(K)) \subseteq B_n(L)$. We begin by showing that $\tilde{f}_n(Z_n(K)) \subseteq Z_n(L)$. Let $z \in Z_n(K)$. Then by definition, $\partial_n(z) = 0$. By Lemma 2.1.3 we have that

$$\partial_n\big(\tilde{f}_n(z)\big) = \tilde{f}_{n-1}(\partial_n(z)) = \tilde{f}_{n-1}(0) = 0,$$

which proves the claim. Next, we show that $\tilde{f}_n(B_n(K)) \subseteq B_n(L)$. Let $b \in B_n(K)$. This implies that there exists some $n+1$-chain d such that $\partial_{n+1}(d) = b$. Then by Lemma 2.1.3,

$$\tilde{f}_n(b) = \tilde{f}_n(\partial_{n+1}(d)) = \partial_{n+1}\big(\tilde{f}_{n+1}(d)\big),$$

which shows that $\tilde{f}_n(b) \in B_n(L) = \operatorname{im} \partial_{n+1}$, and thus proves the claim.

Finally, we will verify explicitly that the map f_n is well-defined. Let $[c] = [d] \in H_n(K)$, which means that the two n-chains c and d are representatives of the same class in $H_n(K)$. Our goal is now to show that $f_n([c]) = f_n([d])$. Because $[c] = [d]$, there exists a boundary b in $B_n(K)$ such that $c = d + b$. Now, we have already proved that $b \in B_n(K)$ implies $\tilde{f}_n(b) \in B_n(L)$, and note that this implies that $[\tilde{f}_n(b)] = 0$. Now, inserting the definition of f_n yields

$$f_n([c]) = \big[\tilde{f}_n(c)\big] = \big[\tilde{f}_n(d+b)\big] = \big[\tilde{f}_n(d) + \tilde{f}_n(b)\big] = \big[\tilde{f}_n(d)\big] = f_n([d]),$$

which proves the well-definedness of the induced map in homology. □

The next theorem is crucial for the definition of persistent homology, because it asserts that if two simplicial complexes are isomorphic, then their homology vector spaces will also be isomorphic.

Theorem 2.1.5. *Suppose K, L and M are simplicial complexes. Let $f\colon K \to L$ and $g\colon L \to M$ be simplicial maps. Then*

$$(g \circ f)_n = g_n \circ f_n.$$

Proof. We begin by proving that $\widetilde{(g \circ f)}_n = \tilde{g}_n \circ \tilde{f}_n$. Let $c \in C_n(K)$. We have that

$$\widetilde{(g \circ f)}_n(c) = \begin{cases} (g \circ f)(c) & \text{if } \dim(g \circ f(c)) = n, \\ 0 & \text{otherwise.} \end{cases}$$

First we consider the case when $\dim(g \circ f(c)) = n$. Now since

$$n = \dim(c) \geq \dim(f(c)) \geq \dim(g \circ f(c)) = n,$$

we may deduce that $\dim(f(c)) = n$. Thus we get the following:

$$\widetilde{(g \circ f)}_n(c) = (g \circ f)(c) = g(f(c)) = g(\tilde{f}_n(c)) = \tilde{g}_n(\tilde{f}_n(c)) = (\tilde{g}_n \circ \tilde{f}_n)(c).$$

Now if $\dim(g \circ f(c)) \neq n$, we have that $\widetilde{(g \circ f)}_n(c) = 0$. We also know that $\tilde{g}_n(f(c)) = 0$, since $\dim(g \circ f(c)) \neq n$. Now since $\tilde{f}_n(c)$ is either equal to $f(c)$ or 0, this implies that $\tilde{g}_n(\tilde{f}_n(c)) = 0 = (\tilde{g}_n \circ \tilde{f}_n)(c)$, resulting in the following:

$$\widetilde{(g \circ f)}_n(c) = (\tilde{g}_n \circ \tilde{f}_n)(c) = 0.$$

We can therefore conclude that $\widetilde{(g \circ f)}_n = \tilde{g}_n \circ \tilde{f}_n$ for all $c \in C_n(K)$.

To prove the main result, suppose now that $[c] \in H_n(K)$. Then by definition,

$$\begin{aligned} (g \circ f)_n\big([c]\big) &= \Big[\widetilde{(g \circ f)}_n(c)\Big] \\ &= \Big[\big(\tilde{g}_n \circ \tilde{f}_n\big)(c)\Big] \\ &= \Big[\tilde{g}_n\big(\tilde{f}_n(c)\big)\Big] \\ &= g_n\big([\tilde{f}_n(c)]\big) \\ &= g_n\big(f_n([c])\big) \\ &= g_n \circ f_n([c]), \end{aligned}$$

which proves that $(g \circ f)_n = g_n \circ f_n$, and concludes the proof. □

2.1.4 Persistent Homology

Having proved that simplicial homology induces a map in homology between simplicial complexes in a consistent manner, we are close to being able to define persistent homology. The input in persistent homology takes the form of a nested sequence of simplicial complexes, defined as follows:

Definition 2.1.10. A **filtration** of a simplicial complex K is a collection of subcomplexes $\{K_i \subseteq K\}_{i=0}^{m}$ such that

$$K_0 \subseteq K_1 \subseteq \cdots \subseteq K_m = K.$$

Example 2.1.4. Given a point cloud X with pseudo-metric D, we can create a filtration of Vietoris Rips complexes for an increasing sequence of scales ϵ:

$$\mathrm{VR}_{\epsilon_0}(X) \subseteq \mathrm{VR}_{\epsilon_1}(X) \subseteq \cdots \subseteq \mathrm{VR}_{\epsilon_m}(X).$$

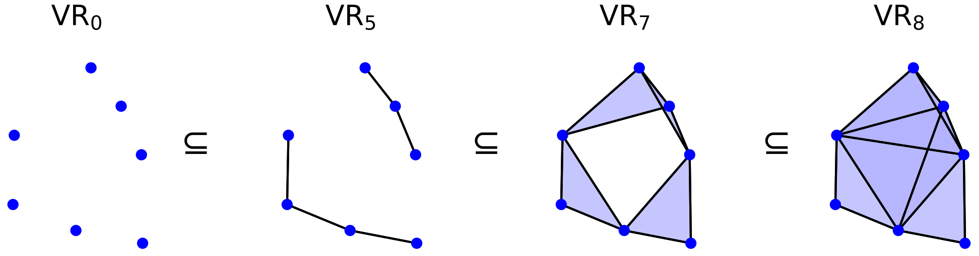


Figure 2.6: A Vietoris Rips filtration for a point cloud consisting of seven points in $\mathbb{R}^2$.

Remark. For any index i, the inclusion $K_i \subseteq K_{i+1}$ can equivalently be written as an inclusion map $K_i \hookrightarrow K_{i+1}$. Thus we have a sequence of inclusion maps of simplicial complexes, $K_0 \hookrightarrow K_1 \hookrightarrow \ldots \hookrightarrow K_m$. Note that these inclusions are simplicial maps. As a result, we have that for all $i \leq j$, the inclusion $K_i \subseteq K_j$ induces a map in homology

$$f_p^{i,j} \colon H_p(K_i) \to H_p(K_j)$$

for each dimension p. Furthermore, Theorem 2.1.5 implies that

$$f_p^{j,k} \circ f_p^{i,j} = f_p^{i,k} \text{ for all } i \leq j \leq k.$$

We are now ready to define persistent homology. Intuitively, persistent homology is a mathematical tool that keeps track of the homological features of the subcomplexes in a filtration, as the scale (or "time") parameter i increases in the filtration. It is defined formally in the following way:

Definition 2.1.11. The **p-th persistent homology vector spaces** of a filtration $K_0 \subseteq K_1 \subseteq \cdots \subseteq K_m = K$ are the images of the linear maps induced by inclusion,

$$H_p^{i,j} = \mathrm{im}\left(f_p^{i,j}\right) \text{ for all } 0 \leq i \leq j \leq m.$$

The associated **p-th persistent Betti numbers** are given by $\beta_p^{i,j} = \dim H_p^{i,j}$.

What makes persistent homology powerful, is that measuring the changes in homology over the course of a filtration might reveal details about the true "shape" of the input data. It recognizes p-dimensional holes in a data collection and records "when" these features occur and disappear. This we call the birth and death of a homological feature.

Definition 2.1.12. Let α be a nonzero homology class in $H_p(K_b)$.

- We say that α is **born** at b if $\alpha \notin H_p^{b-1,b}$.
- If α is born at b, then it **dies** at d if it is absorbed by another homology class when mapped from $H_p(K_{d-1})$ to $H_p(K_d)$. When two classes merge into one, which class that absorbs (or kills) the other is determined by which class was born first. This is called the *Elder rule*. Importantly, if no such d exists, then we say that α dies at infinity.
- The interval $[b, d)$ is the **persistence interval** associated to α, and the **persistence** of α is defined as the difference $d - b$.

The collection of all birth-death persistence-pairs is encoded within the persistent Betti numbers. If we unfold the definition of the persistent Betti number $\beta_p^{b,d}$, we see that it counts the number of homology classes that were born at or before b that are still alive at d [9]. By inclusion-exclusion, it is possible to calculate the number of linearly independent homology classes born at exactly b and dying at exactly d, with the formula

$$\mu_p^{b,d} = \left(\beta_p^{b,d-1} - \beta_p^{b,d}\right) - \left(\beta_p^{b-1,d-1} - \beta_p^{b-1,d}\right).$$

Indeed, we can see that the first difference on the right hand side counts the number of homological classes that die exactly at d (and born at or before b), while the second difference counts the number of homological classes born at or before $(b-1)$ that die at exactly d. The number of features that are born at b but never die can be expressed as

$$\mu_p^{b,\infty} = \beta_p^{b,m} - \beta_p^{b-1,m},$$

where m is the maximal filtration index.

Visualization of Persistent Homology

While the persistent Betti numbers encode information about the homological features of a filtration, for filtrations of increasing size and complexity, they will be hard to interpret with the naked eye. To visualize the output of persistent homology, we do not visually represent the Betti numbers directly, but instead opt to generate plots that contain the information about all birth-death persistence pairs (b, d). Usually these birth-death pairs are visualized in one of the following ways:

- The **persistence barcode** in dimension n is obtained by interpreting each $\mu_n^{b,d}$ as $\mu_n^{b,d}$ copies of the interval $[b, d)$. It is usually visualized by stacking all intervals on top of each other in the plane.
- The **persistence diagram** in dimension n is obtained by plotting the birth-death-pairs as points $(b, d) \in \mathbb{R}^2$ with multiplicity $\mu_n^{b,d}$. A point of the form (b, ∞) is usually plotted as a point (b, k), where $k > m$ is a value larger than the maximal filtration index.

The following lemma, called the Fundamental Lemma of Persistent Homology, establishes the fact that the information encoded in a persistence diagram or barcode, is exactly the same as the information encoded by the persistent Betti numbers. This ensures that the visualizations we have chosen represents persistent homology in a manner that is faithful to the original definition.

Lemma 2.1.6 (Fundamental Lemma of Persistent Homology)**.** *Let* $K_1 \subseteq \ldots \subseteq K_m = K$ *be a filtration. Then for every pair of indices* $0 \leq b \leq d \leq m$ *and every dimension* n*, we have that*

$$\beta_n^{b,d} = \sum_{i \leq b, j > d} \mu_n^{i,j}.$$

Proof. We prove the statement by showing that the right hand side of the equation is equal to the left hand side. We get that

$$\begin{aligned}
\sum_{i \leq b, j > d} \mu_n^{i,j} &= \sum_{i \leq b} \sum_{j \geq d+1} \mu_n^{i,j} \\
&= \sum_{i \leq b} \sum_{j \geq d+1} \left(\beta_n^{i,j-1} - \beta_n^{i,j}\right) - \left(\beta_n^{i-1,j-1} - \beta_n^{i-1,j}\right) \\
&= \sum_{i \leq b} \sum_{j \geq d+1} \left(\beta_n^{i-1,j} - \beta_n^{i,j}\right) - \left(\beta_n^{i-1,j-1} - \beta_n^{i,j-1}\right)
\end{aligned}$$

While it may not be immediately apparent due to the plethora of indices, the inner sum in the last formula is a telescoping sum of the form $\sum_{j=d+1}^{\infty}(a_j - a_{j-1}) = -a_d$. Thus it can be simplified to the following (a new telescoping sum):

$$\sum_{i \leq b, j > d} \mu_n^{i,j} = \sum_{i \leq b} \left(\beta_n^{i,d} - \beta_n^{i-1,d}\right) = \beta_n^{b,d},$$

which was exactly the left hand side of the equation in the lemma. Thus we have proven the result. ☐

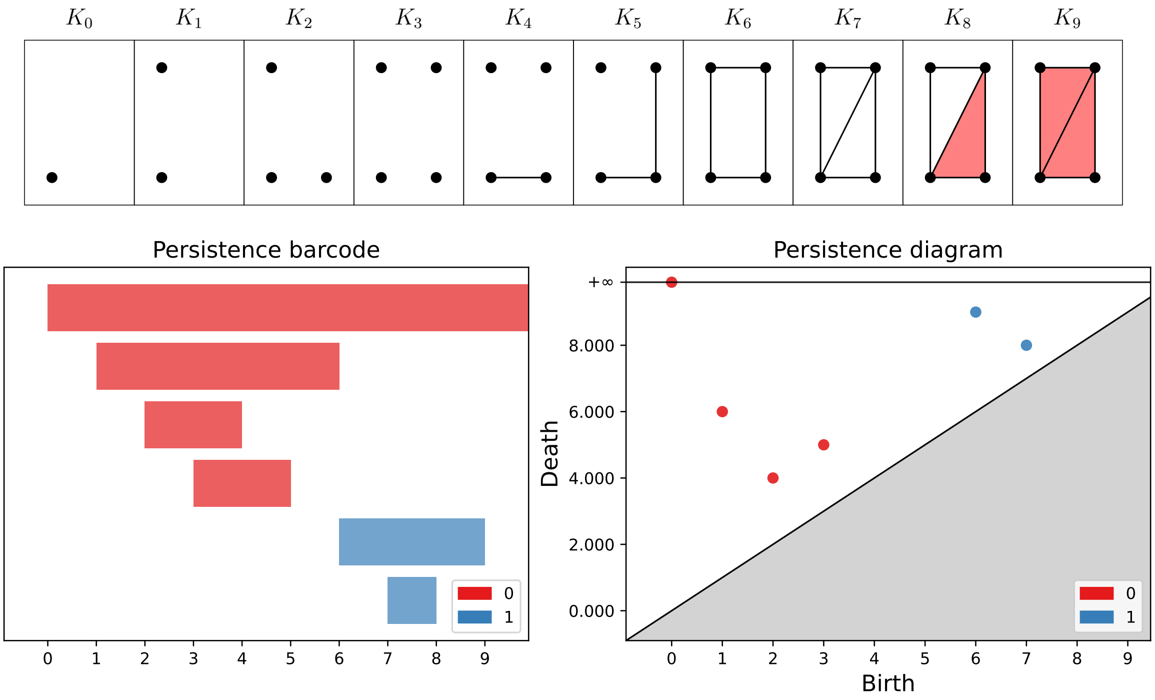


Figure 2.7: Above in the figure is a filtered simplicial complex, and below is its associated barcode and persistence diagram. This example is adapted from [4].

2.1.5 The Persistence Algorithm

Besides having a concise description in terms of persistence diagrams or barcodes, persistence can also be computed efficiently. In this section we introduce a relatively simple algorithm that computes the persistent homology of a given filtration. This section is based on [9] and [20].

Suppose like before that we have a filtration $K_0 \subseteq K_1 \subseteq \ldots \subseteq K_m = K$, and suppose that the simplicial complex K consists of a total number of N simplices. We now introduce some new notation. Define the **time of appearance** $t(\sigma)$ of a simplex $\sigma \in K$ to be the smallest integer i such that $\sigma \in K_i$. Based on the filtration of K, we create a **compatible ordering** $\{\sigma_1, \sigma_2, \ldots, \sigma_N\}$ of all the simplices of K, that satisfies the following two properties:

- if $t(\sigma_i) < t(\sigma_j)$, then $i < j$.
- if $t(\sigma_i) = t(\sigma_j)$ and σ_i is a face of σ_j, then $i < j$.

Aside from these constraints, the ordering of the simplices is arbitrary. From this ordered set of simplices, we create a $N \times N$ boundary matrix ∂ that sends a given simplex to its boundary by letting

$$\partial[i,j] = \begin{cases} 1 & \text{if } \sigma_i \text{ is a face of } \sigma_j \text{ and } \dim \sigma_i = \dim \sigma_j - 1, \\ 0 & \text{otherwise.} \end{cases}$$

Explained in words, the rows and columns of ∂ are ordered like the simplices in the compatible ordering, and the boundary of a simplex is noted in its corresponding column. The persistence algorithm now performs column operations to convert the matrix into a reduced matrix R, which encodes the information about the persistent homology of the filtration. Let $\operatorname{low}(j)$ represent the row number of the lowest 1 in the column j. If the entire column j is zero, then $\operatorname{low}(j)$ is undefined. The algorithm goes as follows:

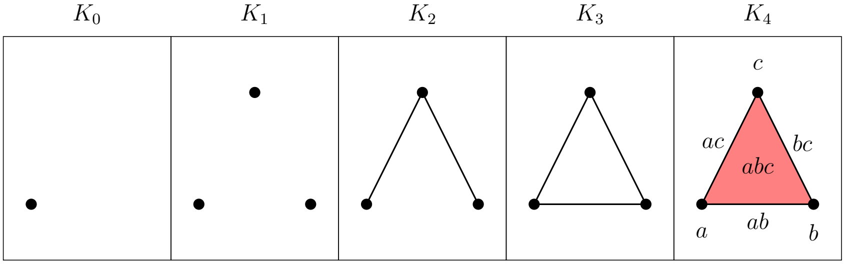
Algorithm 1 The Persistence Algorithm

```
for $j = 1$ to $m$:
    while there exists $i < j$ with $\operatorname{low}(i) = \operatorname{low}(j)$ do
        add column $i$ to column $j$
    end while
end for
```

The intervals of the persistence barcode may be read off the reduced boundary matrix R by pairing the simplices as follows:

- If $\operatorname{low}(j) = i$, then the simplex σ_j is paired with the simplex σ_i, and a homological feature of dimension $\dim \sigma_i$ is born at the entrance of σ_i that dies with the entrance of σ_j. This corresponds to a birth-death pair $(t(\sigma_i), t(\sigma_j))$.
- If $\operatorname{low}(j)$ is undefined, then the entrance of a simplex σ_j causes the birth of a homological feature of dimension $\dim \sigma_j$ at time $t(\sigma_j)$. If there exists a k such that $\operatorname{low}(k) = j$, then the homological feature dies at $t(\sigma_k)$. If no such k exists, the homological feature dies at ∞.

Example 2.1.5. We demonstrate the algorithm by calculating the persistent homology of the following filtration:



The first step is to find a compatible ordering of all the simplices in the filtration. There are in total 7 simplices in the final simplicial complex $K_4 = K$. Following the notation of the simplices as the figure in K_4, we calculate the time of appearance of each simplex in K to be the following:

$$\begin{aligned} 0 &= t(a) \\ 1 &= t(b) = t(c) \\ 2 &= t(ac) = t(bc) \\ 3 &= t(ab) \\ 4 &= t(abc) \end{aligned}$$

An example of a compatible ordering is then the ordered set $\{a, b, c, ac, bc, ab, abc\}$. From this ordered set of simplices, the corresponding 7×7 matrix ∂ is now given by

$$\partial = \begin{array}{c} \\ a \\ b \\ c \\ ac \\ bc \\ ab \\ abc \end{array} \begin{array}{c} \\ \left(\begin{array}{ccccccc} & & & 1 & & 1 & \\ & & & & 1 & 1 & \\ & & & 1 & 1 & & \\ & & & & & & 1 \\ & & & & & & 1 \\ & & & & & & 1 \\ & & & & & & \end{array} \right) \end{array},$$

where the columns are labelled $a \quad b \quad c \quad ac \quad bc \quad ab \quad abc$ in that order:

$$\partial = \begin{array}{c|ccccccc} & a & b & c & ac & bc & ab & abc \\ \hline a & & & & 1 & & 1 & \\ b & & & & & 1 & 1 & \\ c & & & & 1 & 1 & & \\ ac & & & & & & & 1 \\ bc & & & & & & & 1 \\ ab & & & & & & & 1 \\ abc & & & & & & & \end{array}$$

where the blank entries represent zeroes. We perform the persistence algorithm by first adding column ac to column bc, and then adding column bc to column ab. This yields the following reduced matrix:

$$R = \begin{array}{c|ccccccc} & a & b & c & ac & bc & ab & abc \\ \hline a & & & & 1 & 1 & & \\ b & & & & & \boxed{1} & & \\ c & & & & \boxed{1} & & & \\ ac & & & & & & & 1 \\ bc & & & & & & & 1 \\ ab & & & & & & & \boxed{1} \\ abc & & & & & & & \end{array}.$$

Note that we have drawn boxes around entries of interest in the reduced matrix R. We may now read off the persistence intervals from the reduced matrix:

- The simplex a is not paired with any other simplex, and its corresponding column is zero. This results in a persistence interval $[t(a), \infty) = [0, \infty)$ in dimension $\dim a = 0$.
- The simplices b and bc are paired, resulting in a persistence interval $[t(b), t(bc)) = [1, 2)$ in dimension $\dim b = 0$.
- The simplices c and ac are paired. This gives a persistence interval $[t(c), t(ac)) = [1, 2)$ in dimension $\dim c = 0$.
- The simplices ab and abc are paired, resulting in a persistence interval $[t(ab), t(abc)) = [3, 4)$ in dimension $\dim(ab) = 1$.

In conclusion, the barcode of the given filtration is $\{[0, \infty), [1, 2)^2\}$ in dimension 0, and $\{[3, 4)\}$ in dimension 1. The below figure shows the barcode and persistence diagram associated to the filtration.

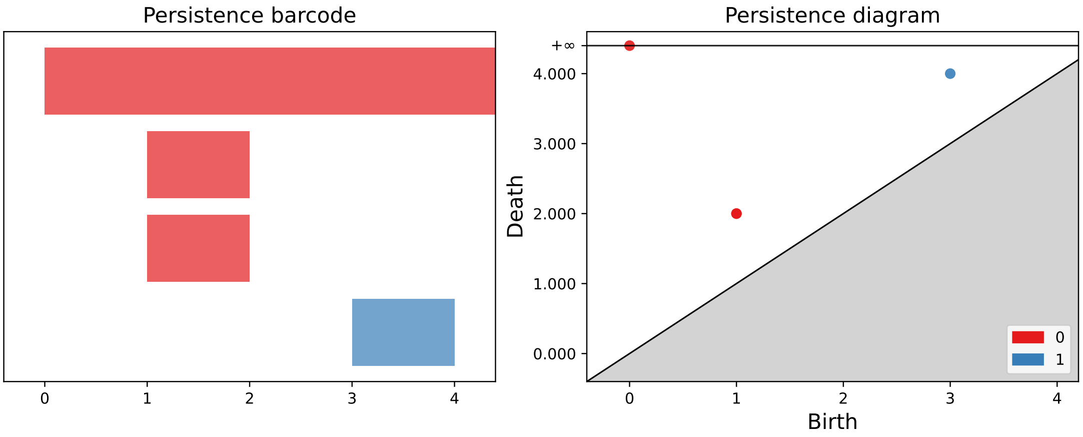


Figure 2.8: The barcode and persistence diagram of the filtration in the example.

We have now presented and demonstrated an algorithm that computes the persistent homology of a given filtration. The worst case running time of the persistence algorithm is cubic in the number of simplices [9]. Though there exists more efficient algorithms that are more often used in practical applications [20], we will not present any other algorithms here.

2.1.6 Persistence Images

Our aim is to use the data encoded in a persistence diagram to aid in text classification. However, it is difficult to apply persistence diagrams directly to downstream machine learning tasks, and we must translate the data from the persistence diagrams into some vector space to make the data compatible with machine learning classifiers. The TDA community has created ways of representing persistence diagrams as vectors which are stable under a suitable distance metric. One of these representations is the *persistence image*, introduced by Adams et al. in 2017 [1], which we will present here.

Assume we are given a persistence diagram B containing birth-death-coordinates. By applying the linear map $T\colon \mathbb{R}^2 \to \mathbb{R}^2$ defined by $T(x, y) = T(x, y - x)$ to B, the persistence diagram is transformed into a multiset $T(B)$ containing birth-persistence coordinates. The multiset $T(B)$ can be interpreted as a non-continuous function $f\colon \mathbb{R}^2 \to \mathbb{N}$ that counts the number of points in the diagram at the input location $u = (x, y)$. This function is now approximated in the continuous domain by means of Gaussian distributions:

Definition 2.1.13. The **persistence surface** of a persistence diagram B is the function

$$\rho_B(x, y) = \sum_{u \in T(B)} w(u) g_u(x, y),$$

where $g_u\colon \mathbb{R}^2 \to \mathbb{R}$ denotes the 2-dimensional Gaussian distribution with mean $u = (u_x, u_y)$ and variance σ^2, and where $w\colon \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$ is a non-negative weighting function that is 0 along the x-axis, continuous and piecewise differentiable.

The weighting function w might at first seem unnecessary, but it is essential for certain stability properties when transforming a diagram into a persistence image [1]. Continuing on, the persistence surface $\rho_B(x, y)$ is reduced into a finite-dimensional vector by integrating the persistence surface over cells of a grid of dimension (n_x, n_y).

Definition 2.1.14. Let B be a persistence diagram. Let R be a rectangular region of $\mathbb{R}^2$ and fix a grid of dimension (n_x, n_y) over R to divide R into a collection of equally sized pixels P. To each pixel P, assign the value

$$\mathrm{Im}_B(P) = \int_P \rho_B(x, y) dx dy$$

The integration value $\mathrm{Im}_B(P)$ of each pixel is stored in a $n_x \times n_y$ image, called the **persistence image** of B.

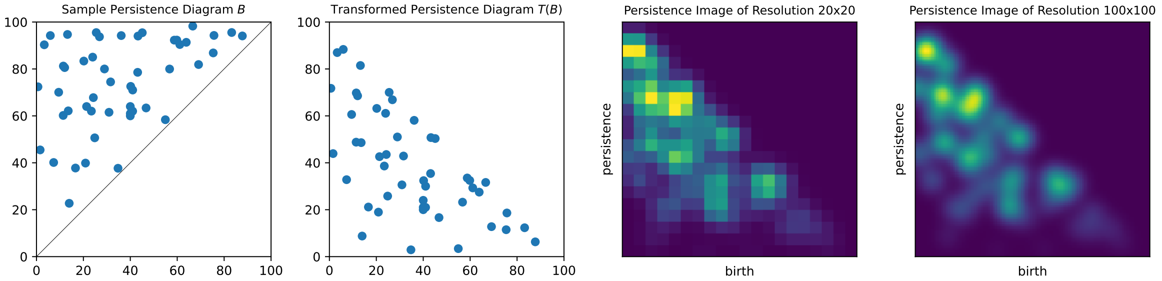


Figure 2.9: Persistence images of a randomly generated persistence diagram B.

Remark. To use the persistence images as input in machine learning algorithms, the images of resolution $n_x \times n_y$, which are matrices, are flattened and interpreted as vectors in $\mathbb{R}^{n_x \times n_y}$.

While persistence images have proven to be useful for classification tasks, they come with a disadvantage. When generating a persistence image, the user makes three nontrivial choices: the image resolution, the weighting function, and lastly, the standard deviation parameter σ in the Gaussian distribution. However, research suggest that the classification accuracy performance using persistence images is fairly robust to changes in image resolution, and has low sensitivity to changes in standard deviation σ in the Gaussian distribution [1].

2.2 Natural Language Processing

Natural Language Processing (NLP) is an interdisciplinary field of research that aims to provide computational tools and models for analyzing language [15]. Although TDA has been applied successfully in many studies involving high-dimensional numerical data, applying TDA to text is not a straightforward task [11]. Since TDA is used to extract the shape (or the topological signature) of point cloud data, to apply this in the case of natural language, the textual data must first be transformed into some geometric object to which persistent homology can be applied. It might not however be immediately clear how one should interpret the geometry of text.

The first step in the majority of NLP applications is to convert the natural language into some kind of numerical representation that can be processed by a computer more easily [19]. This representation is often in the form of a real-valued vector that tries to encapsulate the ”meaning” or ”relevance” of a word. Additionally, these vector embeddings are often designed so that words with similar meanings are supposed to be ”near” to one another in a suitable metric. As a result, the vector embeddings have an inherent geometry that can be analyzed using TDA.

In the following sections we lay the theoretical groundwork for how to apply TDA to text documents. We give an overview of vectorization methods that convert texts to point cloud data, present text preprocessing methods common in NLP tasks, and finally present a method of applying TDA to textual data proposed by Zhu [27].

2.2.1 Text Vectorization

Here we introduce four common methods of vectorizing text documents, and establish a notion of similarity between textual units by defining a metric on the vector spaces of text embeddings. First, we introduce some terminology relevant to this section: We define a **document** to be a single body of text, for example a sentence or paragraph (the choices may depend on the nature of the text). Next, define a **term** to be a single unit of meaningful text in a document. This may for example be a word, pairs of words, and so on. A **corpus** is a collection of documents.

Bag of Words (BoW)

The Bag-of-Words (BoW) model is one of the simplest methods to vectorize a given text. Given a corpus, the BoW model considers each term to be a basis element. A corpus consisting of d unique terms thus has d basis elements. The BoW model then assigns to each document a vector

$$\mathbf{x} = (c_1, c_2, \dots, c_d) \in \mathbb{R}^d,$$

where c_i counts the number of occurrences of the i-th term in a given document [26].

Example 2.2.1. Consider the following corpus, consisting of two documents:

1. my dog is a nice dog
2. your dog is not nice

In this example, each line is a document in the corpus. Furthermore, each word in the corpus is a term. Each unique term determines a basis element, and thus the following set determines the basis for the BoW vector space:

$$\{\text{"a"}, \text{"dog"}, \text{"is"}, \text{"my"}, \text{"nice"}, \text{"not"}, \text{"your"}\}.$$

Counting the occurences of each term in every document, we get the following two BoW-vectors in $\mathbb{R}^7$:

	a	dog	is	my	nice	not	your
$\mathbf{x_1}$	1	2	1	1	1	0	0
$\mathbf{x_2}$	0	1	1	0	1	1	1

Term Frequency - Inverse Document Frequency (TF-IDF)

An underlying assumption in the BoW-model is that the importance of a term is directly proportional to the number of times it appears in a document. This means that a document will be misrepresented by the model if the most important terms do not appear most frequently, which may often be the case.

The Term Frequency - Inverse Document Frequency (TF-IDF) model is a modification of the BoW-model that addresses the frequency assumptions by weighting a term based on its appearance in the document versus its frequency in the corpus. The intuition here is that terms important to the document appear more frequently, but this is offset by calculating whether the term appears in other documents, implying that it is less important to the specific document [19]. It is defined for a term in a given document as follows:

$$\text{tfidf}(\mathit{term}) = \text{tf}(\mathit{term}) \times \text{idf}(\mathit{term}),$$

where the **term frequency** tf is given by

$$\text{tf}(\mathit{term}) = \frac{\#\text{ of times term appears in document}}{\#\text{ of terms in document}}$$

and the **inverse document frequency**[1] idf is given by

$$\text{idf}(\mathit{term}) = 1 + \ln\left(\frac{1 + \#\text{ of documents}}{1 + \#\text{ of documents in corpus with term}}\right).$$

As mentioned previously, a term is defined to be a single unit of meaningful text in a document. While in some cases you may want a term to be a single word (also called **unigram**), there is no reason why a term cannot be two consecutive words (**bigrams**), n consecutive words (n-**grams**), or even combinations of different m-grams. This is commonly used in many TF-IDF applications.

---

[1] How the idf term is defined varies slightly across literature. Here we present the definition used by `TfidfVectorizer` in the Python-library `sklearn`, as this is used in our applications later.

Example 2.2.2. Let us again consider the corpus

1. my dog is a nice dog
2. your dog is not nice

Let us now calculate the TF-IDF vector of the first document, when we consider terms to be the single words (unigrams) of the corpus. There are in total 6 terms in the first document (counting "dog" twice), and we can calculate the term frequency of each of the terms to be:

$$\text{tf}(\text{"}dog\text{"}) = \frac{2}{6}, \quad \text{and } \text{tf}(\text{"}my\text{"}) = \text{tf}(\text{"}is\text{"}) = \text{tf}(\text{"}a\text{"}) = \text{tf}(\text{"}nice\text{"}) = \frac{1}{6}.$$

We now move on to calculate the inverse document frequencies. The words "dog", "is" and "nice" appear in both documents, and their idf-values are therefore

$$\text{idf}(\text{"}dog\text{"}) = \text{idf}(\text{"}is\text{"}) = \text{idf}(\text{"}nice\text{"}) = 1 + \ln\left(\frac{1+2}{1+2}\right) = 1.$$

The words "my" and "a" are unique to the first document, and hence have a inverse document frequency that differs from the other words, given by

$$\text{idf}(\text{"}my\text{"}) = \text{idf}(\text{"}a\text{"}) = 1 + \ln\left(\frac{1+2}{1+1}\right) \approx 1.405.$$

We obtain the TF-IDF-values of each term by multiplication of the tf and idf values:

$$\begin{aligned} \text{tfidf}(\text{"}a\text{"}) &\approx 0.234 \\ \text{tfidf}(\text{"}dog\text{"}) &\approx 0.333 \\ \text{tfidf}(\text{"}is\text{"}) &\approx 0.166 \\ \text{tfidf}(\text{"}my\text{"}) &\approx 0.234 \\ \text{tfidf}(\text{"}nice\text{"}) &\approx 0.166 \end{aligned}$$

The document vectors are usually normalized to avoid larger documents in the corpus (as in containing more terms) dominating in magnitude over smaller ones. Here we normalize by dividing by the Euclidian norm of the TF-IDF values of the terms in a given document, i.e. by dividing each TF-IDF value with

$$\sqrt{(\text{tfidf}(\text{"}a\text{"}))^2 + (\text{tfidf}(\text{"}is\text{"}))^2 + (\text{tfidf}(\text{"}dog\text{"}))^2 + (\text{tfidf}(\text{"}my\text{"}))^2 + (\text{tfidf}(\text{"}nice\text{"}))^2}.$$

The TF-IDF vector for the second document is obtained analagously, and we present the normalized TF-IDF vectors for the two documents as the rows in the table below.

	a	dog	is	my	nice	not	your
$\mathbf{x_1}$	0.45	0.63	0.32	0.45	0.32	0.00	0.00
$\mathbf{x_2}$	0.00	0.38	0.38	0.00	0.38	0.53	0.53

The benefits of the TF-IDF and BoW models are that they are simple and quite efficient to compute. One of the drawbacks of the BoW and TF-IDF-models is that they yield very large and sparse vector representations. Given a corpus with a vocabulary of d unique terms, a document will be embedded in $\mathbb{R}^d$, with mostly zero-entries and only some non-zero values in each vector. An additional drawback of these representations is that they do not take into account that semantically different words and sentences might have similar meanings.

We will now present two categories of vectorization models that are designed to embed semantically different words with similar meaning close to each other.

GloVe

The GloVe algorithm [21], introduced in 2014, embeds words as vectors in a fixed dimension $\mathbb{R}^n$, typically for n ranging between 25 and 300. It is an unsupervised learning algorithm that is trained on a (usually very large) text corpus, such that words that often co-occur together are embedded closely together in a chosen metric [10]. To produce sentence vectors (or in general document vectors) from GloVe word embeddings, a natural method is to take the average of the GloVe vectors of all the words in a given document.

BERT and Sentence-BERT

A notable example of a more advanced word embedding technique is BERT (Bidirectional Encoder Representations from Transformers), which produces context-dependent word embeddings [10]. Unlike the TF-IDF and GloVe models that provide a fixed embedding for a given word, BERT generates different embeddings of the same word based on neighbouring words (i.e. the linguistic context). For example, in different linguistic situations, the term ”bank” might have distinct meanings. It might function as either a noun or a verb, and it can mean different things, like financial institution or land near a river [10]. BERT will capture this context, and produce different words vectors of ”bank” accordingly.

Sentence-BERT is an extension of BERT, and is distinguished from the original BERT by being optimized such that sentences with similar meanings are mapped closely together in the resulting vector space of text embeddings [10].

Metric on Text Embeddings

We have now discussed possible ways of vectorizing textual data, letting us create point clouds from collections of texts. To apply persistent homology to these point clouds using Vietoris Rips complexes, it is required that we establish a metric on the vector space of text embeddings. While the standard choice for a metric in $\mathbb{R}^n$ is typically the Euclidian metric, it is not necessarily the best choice in NLP applications. To capture a notion of similarity we utilize the metric used by Zhu [27], the **angular distance**, defined on two vectors $\mathbf{u}$ and $\mathbf{v}$ by

$$\mathrm{D}(\mathbf{u},\mathbf{v}) = \cos^{-1}\left(\frac{\mathbf{u}\cdot\mathbf{v}}{\|\mathbf{u}\|\|\mathbf{v}\|}\right).$$

Notice that the angular distance simply measures the angle between two vectors $\mathbf{u}$ and $\mathbf{v}$

in $\mathbb{R}^n$. Often one makes use of a normalized variant of the angular distance, by dividing the above formula by either $\pi/2$ or π depending on if $\mathbf{u}$ and $\mathbf{v}$ contain only positive entries or if they can also be negative. This normalized metric is bounded in the interval $[0, 1]$. In the context of persistent homology of a Vietoris Rips filtration in this metric, this implies that all birth-death coordinates of persistent homological features are bounded within the box $[0, 1] \times [0, 1]$, which might not be the case in the Euclidian metric.

2.2.2 Text Preprocessing

There is an important step one usually performs on a dataset of text documents before generating any vectors, and this is to perform a cleaning procedure on each document. This is done to reduce the amount of unnecessary information from the original texts and to set the analysis up for success [2]. We use the following preprocessing procedure:

1. **Unitization and tokenization.** This step involves choosing what serves as a unit in the text (typically words, pairs of words, or sentences), and the text is separated into individual tokens corresponding to the choice of unit.
2. **Standardize & cleanse.** All letters are converted to lower-case, and all numbers, special characters, unnecessary whitespaces and punctuation are removed.
3. **Stop words removal.** This step involves the removal of filler words, or stop words, which add no value to the analysis. Examples of stop words are ”the”, ”and”, ”be”, ”to” and ”of”. While these words are of grammatical importance, they contribute very little to the meaning of the texts, and may be safely removed.
4. **Lemmatization.** Lemmatization is a technique for converting different words with the same word to a common base form. For instance, the words ”walked”, ”walking” and ”walks” are all considered as the same token ”walk”.

An example of the preprocessing procedure performed on a text document is provided in Appendix A.6.

2.2.3 TDA on Text

Once the textual data has been preprocessed and then mapped into some metric space, the implementation of TDA is relatively straightforward. The points of the metric space have an inherent geometry that allows TDA to extract topological properties. What these topological properties, such as holes and linked components, should represent about the original text might however be unclear. Ideally, the homological features should have an interpretation that relates back to the structure of the text.

In this section we will present two ways of extracting homological features of textual data, introduced in a 2013 paper by Zhu [27]. The strength of these methods, is that the 0th and 1st homology classes in persistent homology are given a clear and observable interpretation. For both methods, assume the following: A text has been divided into smaller units that have been assigned real valued vectors $\mathbf{x_1}, \ldots, \mathbf{x_n}$. These units may for example be sentences or paragraphs. In addition, we are given a metric $D(\mathbf{x_i}, \mathbf{x_j}) \geq 0$ (in our case, the angular distance) such that the distance between ”similar” text units is small.

Similarity Filtration (SIF)

The Similarity Filtration (SIF) algorithm calculates the persistent homology of a Vietoris Rips filtration over the points $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ with metric D for an increasing sequence of scales ϵ. As we progress through the filtration, the increasing diameter (or time parameter) ϵ corresponds to allowing looser and looser "tie-backs"[27]. In the Vietoris Rips complex, more and more dissimilar text units are linked together to form simplices. Note that the order of the units $\mathbf{x_1}, \ldots, \mathbf{x_n}$ is ignored in the SIF-algorithm.

Similarity Filtration with Time Skeleton (SIFTS)

The Similarity Filtration with Time Skeleton (SIFTS) is a modified version of the SIF-algorithm that injects an order between successive textual units. Without modification, the metric D will be oblivious to the ordering of $\mathbf{x_1}, \ldots \mathbf{x_n}$. To inject an order between vectors $\mathbf{x_i}$ and $\mathbf{x_{i+1}}$ representing consecutive units of text, we modify the metric D by requiring that $D(\mathbf{x_i}, \mathbf{x_{i+1}}) = 0$ for all i. Note that this modification renders D a pseudo-metric. The SIFTS-algorithm now applies persistent homology to a Vietoris Rips filtration over the points $\{\mathbf{x_1}, \ldots, \mathbf{x_n}\}$ with pseudo-metric D. Due to requiring $D(\mathbf{x_i}, \mathbf{x_{i+1}}) = 0$, there are now "time-edges" (1-simplices) $[x_i, x_{i+1}]$ for all i in all stages of the filtration. As a result, the complex in the filtration will always be connected by what Zhu calls a "time skeleton", resulting in a single trivial feature in 0th persistent homology.

An interpretation of 1-dimensional holes in SIFTS is that they represent semantic tie-backs [27]. For example, a well written essay may contain a conclusion paragraph that "links back" to the introduction paragraph. Thus the vectors representing the first and last sentence may be close in space, forming a loop since all the consecutive sentences between them are connected. On the other hand, a badly written essay might not form any loops at all.

Example 2.2.3. In this example we wish to make clear how the SIF and SIFTS algorithms work, and what sets them apart from each other, by considering a toy example. Suppose that a vectorizing a text consisting of 4 documents yields a point cloud $\{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4}\}$ consisting of four points with the following pairwise distances

$$\begin{aligned}
D(\mathbf{x_1}, \mathbf{x_2}) &= 1 \\
D(\mathbf{x_1}, \mathbf{x_3}) &= \sqrt{2} \approx 1.41 \\
D(\mathbf{x_1}, \mathbf{x_4}) &= 0.25 \\
D(\mathbf{x_2}, \mathbf{x_3}) &= 1 \\
D(\mathbf{x_2}, \mathbf{x_4}) &= 1.03 \\
D(\mathbf{x_3}, \mathbf{x_4}) &= 1.25
\end{aligned}$$

The points are constructed such that the first and the last text units $\mathbf{x_1}$ and $\mathbf{x_4}$ are closest to each other with the given distance function, and are depicted in Figure 2.10. An interpretation of this is that the first and last documents in the text, the "introduction" and "conclusion", are the most similar textual units. Applying SIF and SIFTS to the point cloud results in the filtrations depicted in Figure 2.11. Note that the SIFTS algorithm modifies the distance function, requiring that $D(\mathbf{x_1}, \mathbf{x_2}) = D(\mathbf{x_2}, \mathbf{x_3}) = D(\mathbf{x_3}, \mathbf{x_4}) = 0$. By injecting order between consecutive text units, the SIFTS algorithm captures a circular structure in the text that the SIF algorithm does not register.

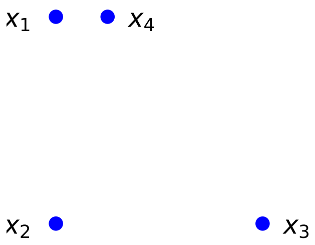


Figure 2.10: The point cloud considered in Example 2.2.3.

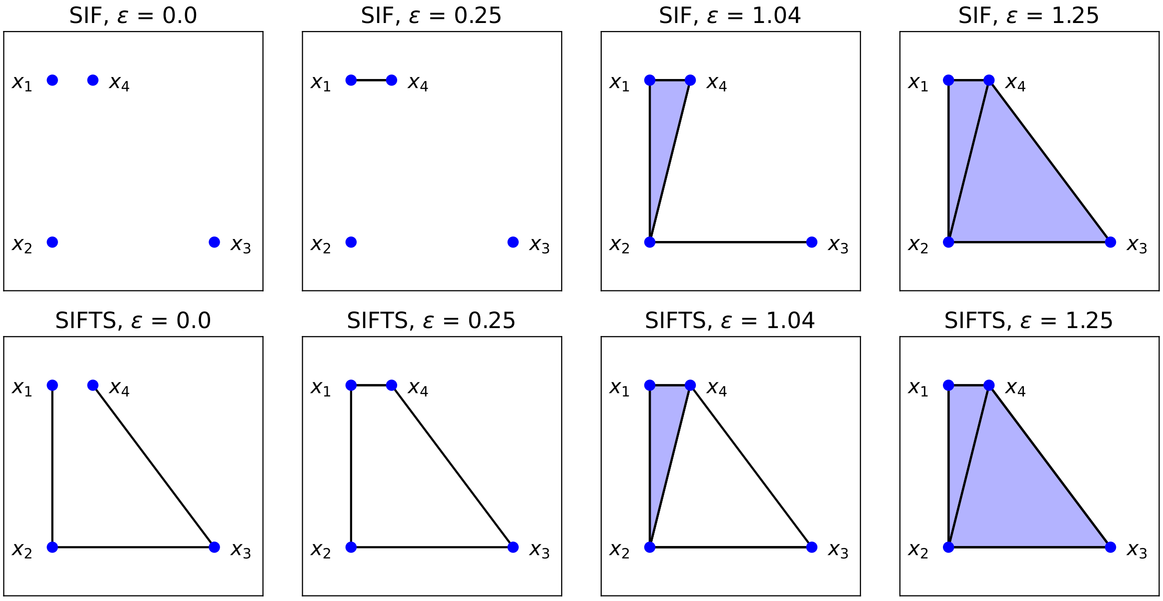


Figure 2.11: Above we see the parts of the Vietoris Rips filtration for the given point cloud used in the SIF algorithm, and below from SIFTS. The SIFTS algorithm manages to capture a circular structure in the text, representing a "tie-back", because there are time-edges connecting consecutive text units x_i and x_{i+1}.

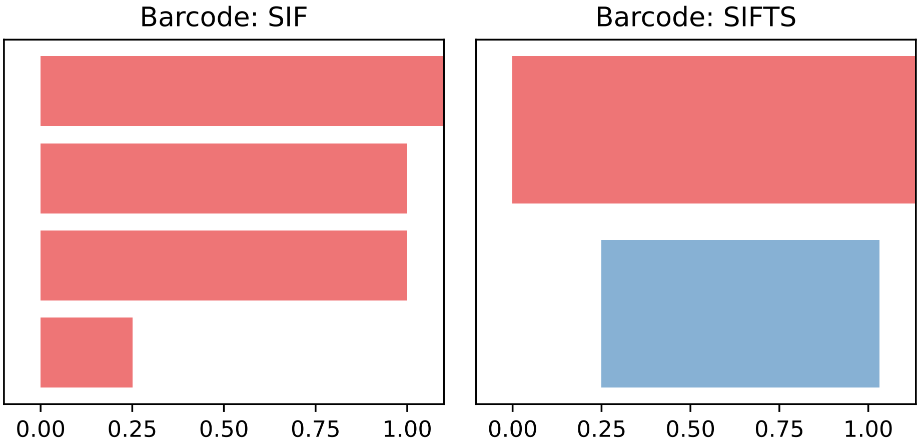


Figure 2.12: The persistence barcode for the SIF and SIFTS algorithms for the example.

2.3 Related Work

In this section we present a brief literature review of existing works in the field of text classification via topological data analysis, in addition to machine generated text detection in general.

One of the earliest works on TDA for natural language processing is the 2013 paper ”Persistent Homology: An Introduction and a New Text Representation for Natural Language Processing” by Zhu [27]. In this paper the SIF and SIFTS algorithms are introduced, and implementations of the algorithms in Matlab are published. Zhu illustrates the algorithms on a selection of nursery rhymes, as well as on a corpus consisting of child and adolescent texts. By employing the SIFTS algorithm, Zhu identifies statistically significant differences in the amount of semantic tie-backs (H_1-groups) between child and adolescent writing. Using persistent homology, Zhu concludes that children’s writing becomes structurally richer as they mature. However, he does not apply his algorithm to machine learning or text classification tasks.

There are several articles that investigate the use of TDA for classification of texts. Doshi and Zadrozny, in [8], use features generated by the SIFTS algorithm to predict movie genres based on movie plot summaries from IMDB. In [12], Gholizadeh et al. use topological data analysis to classify the authors of different 19th century novels. Deng and Duzhin [7] apply TDA in a fake news detection classification task. They compute persistent homology based on GloVe word embeddings in $\mathbb{R}^{50}$ with the Euclidian metric, and use persistent images as features for a binary classifier that predicts if a news article is real or fake. Their results demonstrate that combining deep learning models with topological data analysis improves accuracy, particularly for small training sets. Lastly, Das et al. [6] generate persistence images from sentence embeddings of transcripts from TED Talks and train a model to predict the rating label (e.g., ”beautiful,” ”confusing,” ”ok,” or ”jaw-dropping”) for each speech.

For the specific case of detection of artificially generated (or machine generated) texts, there are also some noteworthy articles. Kushnareva et al. [17] use TDA-based features as the basis for classifiers trained to distinguish between texts written by humans and by machines. In a recent non-TDA related work, Shijaku and Canahasi [23] train a machine learning model to differentiate between essays written by ChatGPT and humans. They use the previously mentioned TF-IDF model to generate features from the essays, which are then used in classification tasks.

Chapter 3

Experiments

Inspired by previous works in the field, the research objective of this thesis is to explore the usage of Topological Data Analysis for text classification, specifically in the case of machine-generated text detection. In this chapter we develop a pipeline for extracting topological features from text, similar to the methods of the fake news detection article [7] and the TED talk article [6]. These topological features are used to train machine learning classifiers to differentiate samples of human-written and machine-generated text.

We investigate the performance of the proposed TDA-based classifier in two experiments on machine-generated text detection. The first experiment considers the output of the language model GPT-2, which generates human-like web page documents, while the second experiment considers essays generated from ChatGPT. For comparison with the TDA-based classifier, we also evaluate the performance of non-TDA-based classifier on the same data. Lastly, in the second experiment we investigate the effects of assembling TDA-based and non-TDA-based classifiers.

3.1 Method

Both experiments conducted in this thesis follow the same general method, which we describe in this section. We are interested in training machine learning models to be able to distinguish between human-written and machine-generated texts. To this end, we train two types of classifiers: a TDA-based classifier and non-TDA-based classifier, which we present here. Furthermore, we elaborate on how these classifiers are implemented, and how their performance is evaluated in each experiment.

TDA Classifier Pipeline

Here we sketch the general pipeline for a TDA-based text classifier. To see the implementation in Python used for these experiments, we refer the reader to Appendix A. The steps performed on a dataset of text documents are as follows:

1. **Dataset Preparation.** Separate each document into sentences. On each sentence, perform the text preprocessing steps as previously described: lower-case and expand contractions, remove non-alphabetic characters, unnecessary white-space and stop words, and finally lemmatize each text.

2. **Text Vectorization.** For each text document, generate a document embedding matrix where the i-th row in the matrix is the sentence vector of the i-th sentence in a given document. Choosing to vectorize with either the TF-IDF model, averaging GloVe word embeddings or Sentence-BERT yields three distinct matrices to be used in different classifiers.

3. **Calculate Distance.** Compute a distance matrix from each sentence matrix in the previous step. Given a sentence matrix with k sentences, the resulting distance matrix is a $k \times k$ matrix in which the $[i, j]$ term of the matrix represents the distance between sentences i and j in a given metric D. As a result, the distance matrix is symmetric and has 0s along the diagonal.

4. **Persistent Homology.** Apply either the SIF or SIFTS algorithm to obtain persistence diagrams in dimensions 0 and 1. Convert the persistence diagrams in the dimension of your choice (0 or 1) to persistence images with resolution $n \times n$ to obtain vectors of dimension n^2 for each text.

5. **Classification.** Train and test a machine learning classifier (we use the Support Vector Classifier) using the persistence images as input.

Non-TDA Classifier: XGBoost

As an example of a non-TDA classifier, we follow a similar approach to that of [23]. The dataset is preprocessed exactly as in Step 1 of the TDA classifier pipeline. Considering terms to be both unigrams and bigrams, a single TF-IDF vector is then generated for each document (contrary to one vector per sentence for each document in the TDA-based classifier) and is used as input in the XGBoost classifier algorithm [5].

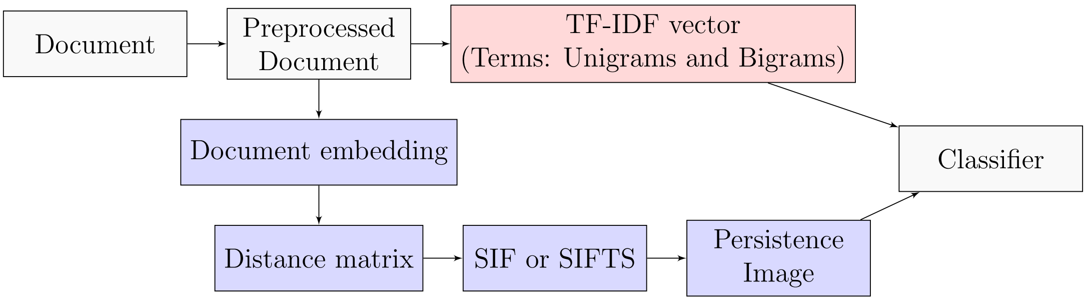


Figure 3.1: A graphic representation of the two pipelines. The above path in the flowchart shows the steps involved in the non-TDA-based classifier, while the path below shows the steps involved in the TDA-based classifier.

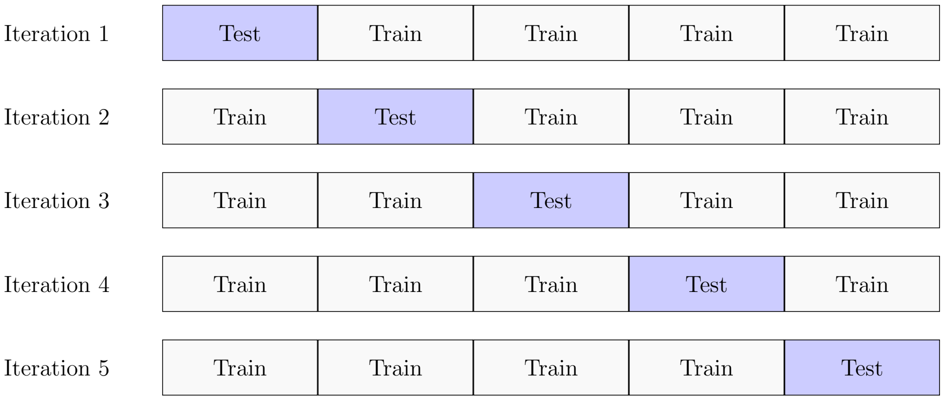
Implemenation

We implement the machine learning pipelines in Python. To preprocess and lemmatize the texts, we use the libraries `NLTK` and `re`. To vectorize texts, we use `TfidfVectorizer` in `sklearn` for TF-IDF embeddings, the library `sentence_transformers` for Sentence-BERT embeddings, and for the GloVe embeddings we directly implement an embedding algorithm using 50-dimensional pre-trained GloVe word embeddings [21].

To the author's knowledge, there are no publicly available implementations of the SIF and SIFTS algorithms in Python[1]. The SIF and SIFTS algorithms are therefore implemented manually, using the TDA-library `gudhi`. To create persistence images from the persistence diagrams resulting from these algorithms, we use the library `persim`. Finally, machine learning classifiers from the libraries `sklearn` and `xgb` are used to implement the pipelines and conduct the experiments. Appendix A contains more information regarding the implementation.

Evaluation

To evaluate the performance of the machine learning models, we use k-fold cross validation. The k-fold cross validation is a statistical technique for assessing the performance of a machine learning model on a given set of data. The dataset is randomly divided into k folds (or partitions), where k is a user-specified integer. The machine learning model is then trained on $k-1$ folds of data, and tested on the single remaining fold. The test accuracy is recorded, and the trained model is discarded. This procedure is performed a total number of k times, where each time, a different fold is treated as the testing set. The performance of the model is then assessed by taking the of average (and standard deviation) of the results over the k folds [14]. Associated to the choice of the integer k there is a bias-variance trade-off, and according to [14], the values $k = 5$ and $k = 10$ are standard choices. To evaluate the classification models in our experiments, we opt for $k = 5$, mainly because higher k means longer computation time.



Iteration 1	Test	Train	Train	Train	Train
Iteration 2	Train	Test	Train	Train	Train
Iteration 3	Train	Train	Test	Train	Train
Iteration 4	Train	Train	Train	Test	Train
Iteration 5	Train	Train	Train	Train	Test

Figure 3.2: A schematic overview of a 5-fold cross validation.

---

[1] However, an implementation in Matlab of the algorithms exists: `https://pages.cs.wisc.edu/~jerryzhu/publications.html`

3.2 Experiment: Webtext & GPT-2

For the first experiment we will consider text outputs from the text generation model GPT-2, released in 2018 [22]. The language model GPT-2 is trained to generate artificial texts that resemble the input training data and is fine-tuned with the objective to predict the next word, given all the previous words within some text.

The **GPT-2 Output Dataset** consists of outputs of GPT-2 models that are trained on the human-written **Webtext**, a dataset consisting of filtered and de-duplicated text documents from 8 million web pages. The dataset comprises outputs of multiple versions of the GPT-2-model of varying sizes and complexity. We consider multiple versions in the experiment, to get an overview of how classification performance varies with complexity: GPT-2 Small, GPT-2 Medium, GPT-2 Large and GPT-2 XL with pure sampling.

Method

Before conducting the experiments, we want to ensure that the machine generated texts and human-written texts compete on an equal footing. When vectorizing a text using the TDA method, the number of points produced is directly proportional to the number of sentences in a document, and hence somewhat dependent on the length of a given text. The amount of points influence the number of potential homological features produced from the texts, and these features are what we want to use to distinguish human and machine-written text. We therefore limit our experiment to entries of Webtext and GPT-2-output with a fixed length. Due to their abundance in the datasets, the choice falls naturally to texts of length 1024.

From the texts of length 1024, we randomly select and fix a set consisting of 250 texts from Webtext. For each of the variants of GPT-2 we also randomly select and fix sets consisting of 250 texts. The datasets we perform experiments on thus consist of 500 documents, each a combination of the fixed Webtext set and 250 texts from the GPT-2 variant.

Hyperparameter tuning

As previously mentioned, there are nontrivial choices involved when transforming persistence diagrams to persistence images to be used in the machine learning classifiers: The image resolution, weighting function and the standard deviation parameter σ. We select the default weighting function in `PersistenceImager`, the persistence weighted Gaussian kernel [16], and perform grid searches on the different TDA classifier models to determine the best parameters for the image resolution and the standard deviation σ. Performing a grid search on different persistence image resolutions,

$$\{25 \times 25, 50 \times 50, 75 \times 75, 100 \times 100\},$$

we find that a resolution of 100×100 always either outperforms or is equally good as the other alternatives. The standard deviation σ is then selected specifically for each TDA-based classifier to optimize performance, from the following set of values:

$$\{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005\}.$$

Furthermore, some experimentation with the TF-IDF vectorization method showed improved classification performance when defining terms to be both unigrams (single words) and bigrams (two consecutive words), compared with terms being defined as only unigrams. As a consequence, the former method was used in the TF-IDF vectorization.

We do preliminary tests on TDA classifiers with persistence images from SIF and SIFTS on three different vector embedding methods: GloVe, Sentence-BERT and TF-IDF. We limit the final experiments to the four TDA-based classifier models with the best performance, in addition to the non-TDA-based XGBoost classifier. The specifications for each model is presented in Table 3.1.

Model	**Description**
SIFTS TF-IDF	Support Vector Machine (SVM) classifier trained on persistence images with $\sigma = 0.0001$ from the persistence diagrams in dimension 1, generated by SIFTS on TF-IDF embeddings with terms defined as unigrams and bigrams.
SIFTS s-BERT	SVM classifier trained on persistence images with $\sigma = 0.01$ from the persistence diagrams in dimension 1, generated by SIFTS on sentence embeddings from the pretrained Sentence-BERT model 'all-MiniLM-L6-v2'.[2]
SIF s-BERT	SVM classifier trained on persistence images with $\sigma = 0.01$ from the persistence diagrams in dimension 0, generated by SIF on sentence embeddings from the pretrained Sentence-BERT model 'all-MiniLM-L6-v2'.
SIFTS GloVe	SVM classifier trained on persistence images with $\sigma = 0.005$ from the persistence diagrams in dimension 0, generated by SIF on 50-dimensional GloVe embeddings pretrained on Wikipedia 2014 and Gigaword 5[3].
XGBoost	The XGBoost classifier model with TF-IDF unigram and bigram features with minimum document frequency 5 as input.

Table 3.1: Model specifications in the experiments with Webtext and GPT-2.

---

[2] `https://www.sbert.net/docs/pretrained_models.html`

[3] `https://github.com/stanfordnlp/GloVe`

Results and discussion

The results from the experiments on Webtext & GPT-2 are presented in Table 3.2. The results clearly suggest that there are statistical differences in persistence images generated from text embeddings of Webtext and GPT-2, but the classification performances also indicate that the texts get increasingly more difficult to distinguish as the complexity of the GPT-2 model increases.

	GPT-2 Small	**GPT-2 Medium**	**GPT-2 Large**	**GPT-2 XL**
XGBoost	83.2(2.9)	86.2(3.4)	75.6(2.7)	67.4(4.6)
SIFTS TF-IDF	**93.0(1.7)**	93.6(3.3)	84.8(4.9)	**81.4(5.1)**
SIFTS s-BERT	92.0(2.2)	92.8(2.2)	84.8(2.8)	80.8(2.7)
SIF s-BERT	93.0(3.0)	**96.0(2.1)**	**87.2(2.8)**	81.0(4.8)
SIFTS GloVe	77.6(3.7)	83.4(2.3)	71.8(5.0)	70.6(5.4)

Table 3.2: Classification results from experiments on Webtext versus GPT-2. The performance is measured by the average accuracy score (%) from a 5-fold cross validation, and the standard deviation is reported in the brackets. The highest test accuracy for each experiment is bolded.

Continuing our analysis, we thought it would be both illustrative and enlightening to examine the persistent homology and persistence images produced from the different vector embeddings manually, to see if we could deduce any information about the homological structure of the texts, and how they might differ.

We first examined the the persistent homology from applying SIFTS on the TF-IDF embeddings on Webtext and the GPT-2 variants. We discovered that the total number of 1-st order homology classes (or simply, number of 1-dimensional holes), $|H_1|$, varied quite significantly across the different datasets. While the average $|H_1|$ of a Webtext document was 40.76, the average $|H_1|$ for a document across all the GPT-2 variants was 27.84. Recalling that the interpretation of Zhu [27] is that a 1-dimensional hole in the SIFTS algorithm represents a "semantic tie-back", our results suggest that on average, a human-written document from Webtext contains significantly more tie-backs than a text written by GPT-2.

Furthermore, examination of the persistence images from SIFTS TF-IDF and SIF s-BERT suggests that there are some qualitative differences in the persistence diagrams generated from Webtext and GPT-2, which helps explain the significant classification performance of the mentioned TDA-based classifier models. Figure 3.3 shows some examples of persistence images of Webtext and GPT-2 Small produced by SIFTS TF-IDF, while in Figure 3.4 we have plotted the average persistence image generated from documents from Webtext and GPT-2 small respectively, using SIFTS TF-IDF. Similarly, Figure 3.5 shows samples of persistence images of Webtext and GPT-2 Medium produced by SIF s-BERT, while the average persistence images from SIF s-BERT are plotted in Figure 3.6.

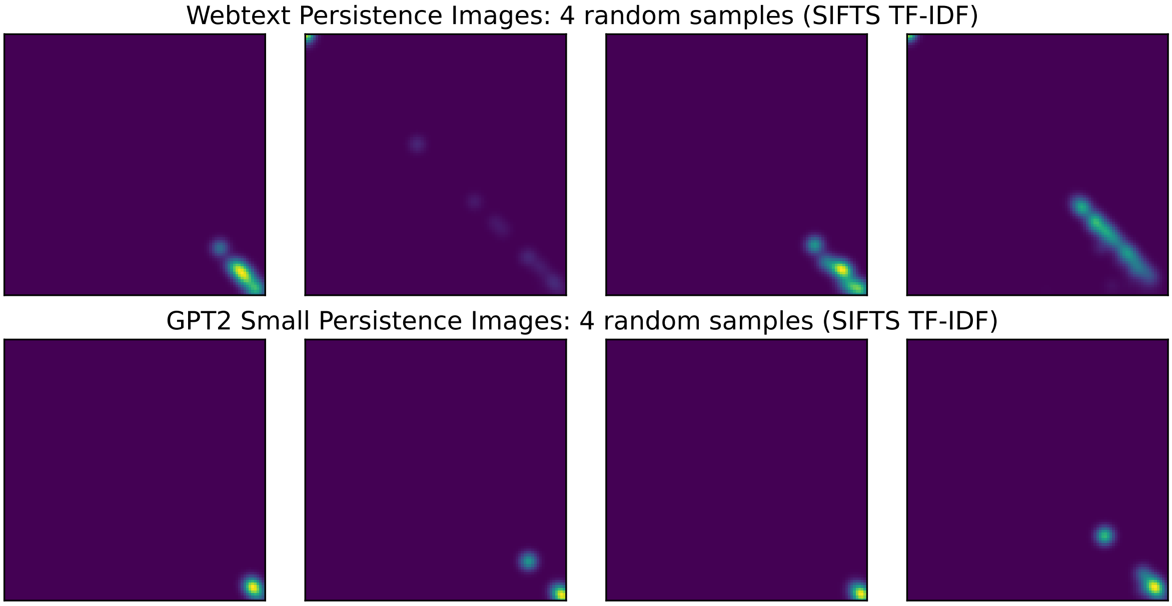


Figure 3.3: Examples of persistence images generated by applying SIFTS to the TF-IDF embeddings for a random selection of documents from Webtext and GPT-2 Small.

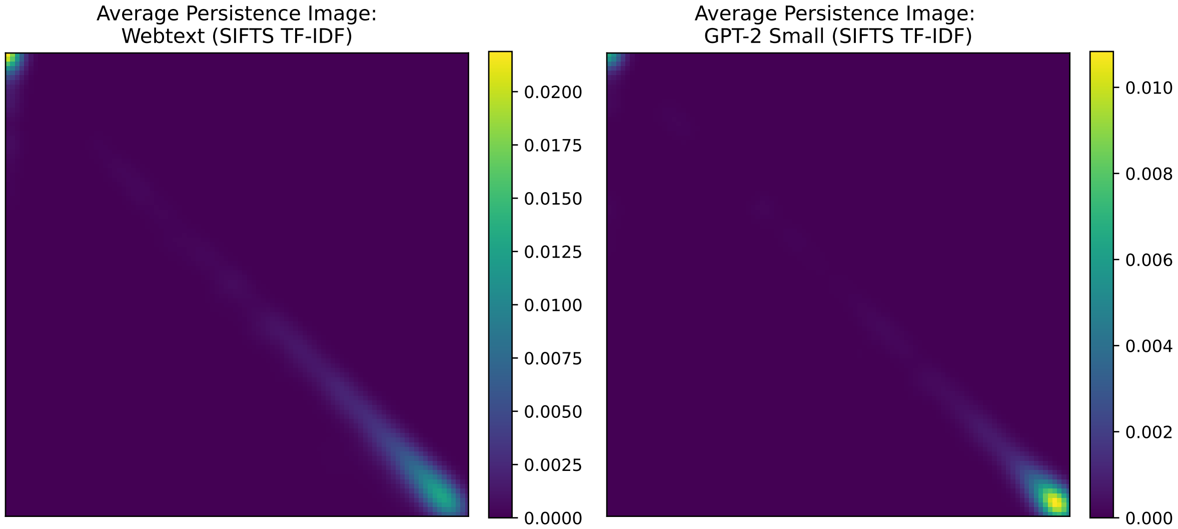


Figure 3.4: The average persistence images of Webtext documents and GPT-2 Small when applying SIFTS TF-IDF. Note that the maximum magnitude of the Webtext persistence image is about double as that of GPT-2 Small.

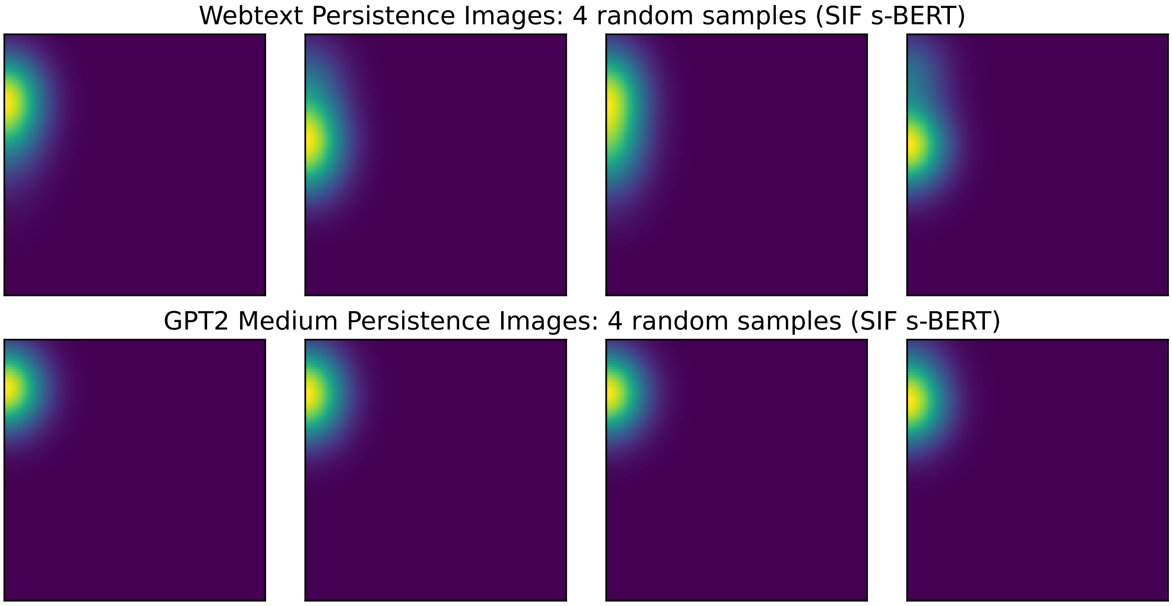


Figure 3.5: Examples of persistence images generated by applying SIF to the sentence BERT embeddings for a random selection of documents from Webtext and GPT-2 Medium.

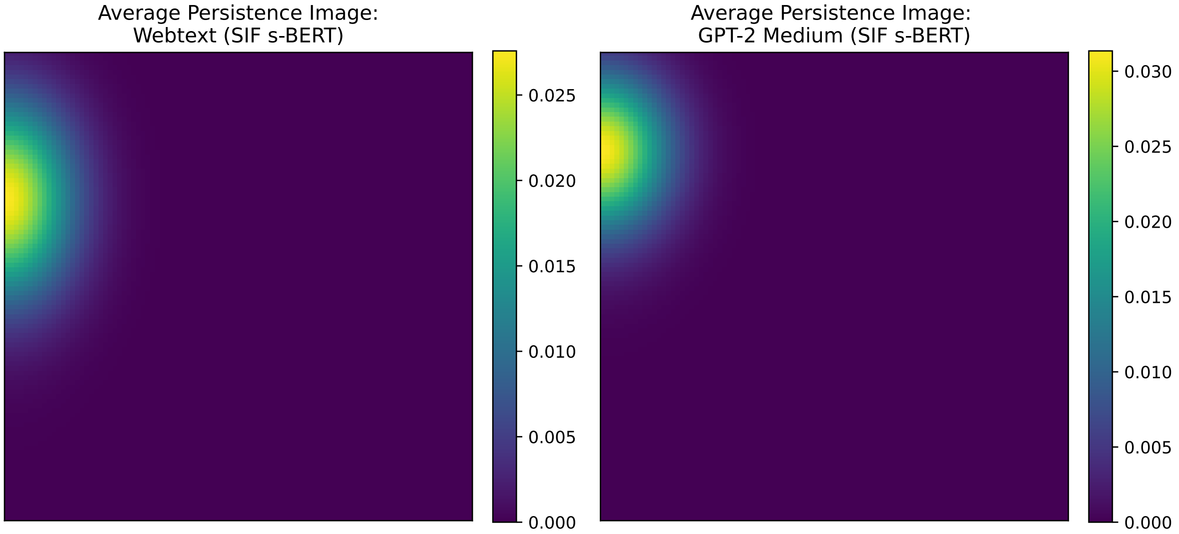


Figure 3.6: The average persistence images of Webtext documents and GPT-2 Medium when applying SIF s-BERT.

3.3 Experiment: ChatGPT Essays

The field of artificial text generation is undergoing frequent development, and a motivation for the next experiment is that we wished to try our TDA classifiers performance on outputs from a more recent generative text model.

Specifically, we wanted to test our TDA classifier on outputs of ChatGPT, a language model released in November 2022[4], that has recently garnered a lot of attention in media. To the author's knowledge, the creators of GPT-2 have not released official output datasets akin to the GPT-2 output dataset for ChatGPT. However, Shijaku & Canhasi [23] have compiled and published a dataset consisting of 126 essays written by humans and 126 essays generated by ChatGPT[5], and this is the dataset we consider in this experiment.

Table 3.3: An example of an essay topic and written responses by a human and ChatGPT.

Writer	**Question: Why do you think people attend college or university?**
ChatGPT	There are many reasons why people choose to attend college or university. Some of the most common reasons include the desire to gain new knowledge and skills, the opportunity to pursue a career in a particular field, and the chance to earn a higher salary. One of the most important reasons that people attend college or university is to gain new knowledge and skills. At college or university, students have the opportunity to learn about a wide range of subjects, from the sciences and humanities to the arts and social sciences...
Human	College is a place that the students can learn more and new knowledge and experience in it. Of course, different people have different reason to study in college. For example, some people want to be to go on a further study after they graduate from the college; some people hope to find a good job after their studying in the college and also some people wish to exchange their present situation through studying in the college. In my opinion, no matter what reason people study in the college for, studying in the college is just a preparation for their future' life...

Contrary to in the previous experiment, we do not filter out any texts in the dataset, but consider the entire dataset consisting of 252 documents. For the persistence images we again use a resolution of 100×100. The optimal values for standard deviation σ in the images are selected individually for each TDA-based classifier, and are shown in Table 3.4.

Model	σ-value
SIFTS TF-IDF	0.001
SIF GloVe	0.001
SIF s-BERT	0.005
SIFTS s-BERT	0.01

Table 3.4: Choices of σ for the different models in the ChatGPT essay experiment.

---

[4] `https://openai.com/blog/chatgpt`

[5] `https://github.com/rexshijaku/chatgpt-generated-text-detection-corpus`

Results and discussion

The results from the experiment on the human-written and ChatGPT-generated essays are presented in Table 3.5.

	Accuracy
XGBoost	**90.9(2.1)**
SIFTS TF-IDF	84.5 (3.6)
SIF GloVe	80.5 (6.4)
SIFTS s-BERT	85.3 (2.0)
SIF s-BERT	90.0 (4.4)

Table 3.5: Classification results from experiments on essays written by humans and ChatGPT. The performance is measured by the average accuracy score (%) from a 5-fold cross validation, and the standard deviation is reported in the brackets. The highest test accuracy for the experiment is bolded.

In this experiment, the non-TDA-based classifier XGBoost had the best performance, by a slight margin over SIF s-BERT. We went on to explore whether creating ensembles of the different classifier models would increase classification performance. This was implemented with sklearns `VotingClassifier`, with soft voting and equal weighting between the models. The results from the experiment with classifier ensembles is presented in Table 3.6.

	Accuracy
XGBOOST + SIFTS TF-IDF	94.1(2.8)
XGBOOST + SIF GloVe	94.1(2.2)
XGBOOST + SIFTS s-BERT	94.4(4.6)
XGBOOST + SIF s-BERT	94.8(3.7)
SIF GloVe + SIFTS TF-IDF	87.7(1.5)
XGBOOST + (SIF GloVe + SIFTS TF-IDF)	**95.2(1.6)**

Table 3.6: Accuracy tables for different ensembles of classifier models used in the experiment. The performance is measured by the average accuracy score (%) from a 5-fold cross validation, and the standard deviation is reported in the brackets. The highest test accuracy for the experiment is bolded.

The results indicate that an ensemble of XGBoost and a TDA-based classifier generally performs better than any single classifier, perhaps suggesting that there are some distinguishing features in the machine-written essays that are discovered by the TDA method that are not discovered by XGBoost, and vice versa.

Like in the previous experiment, we continued our analysis by examining the persistent homology and persistence images produced from the different vector embeddings manually. When applying SIFTS TF-IDF, we observed that the average number of H_1-features for the human-written essays were 15.12, while the average of the ChatGPT-essays was 8.65, suggesting that the average human-written essay contained more ”tie-backs” than the average ChatGPT-written essay. The persistence images also indicated qualitative differences between the two types of essays, and examples of persistence images are shown in Figure 3.7, and average persistence images are shown in Figure 3.8.

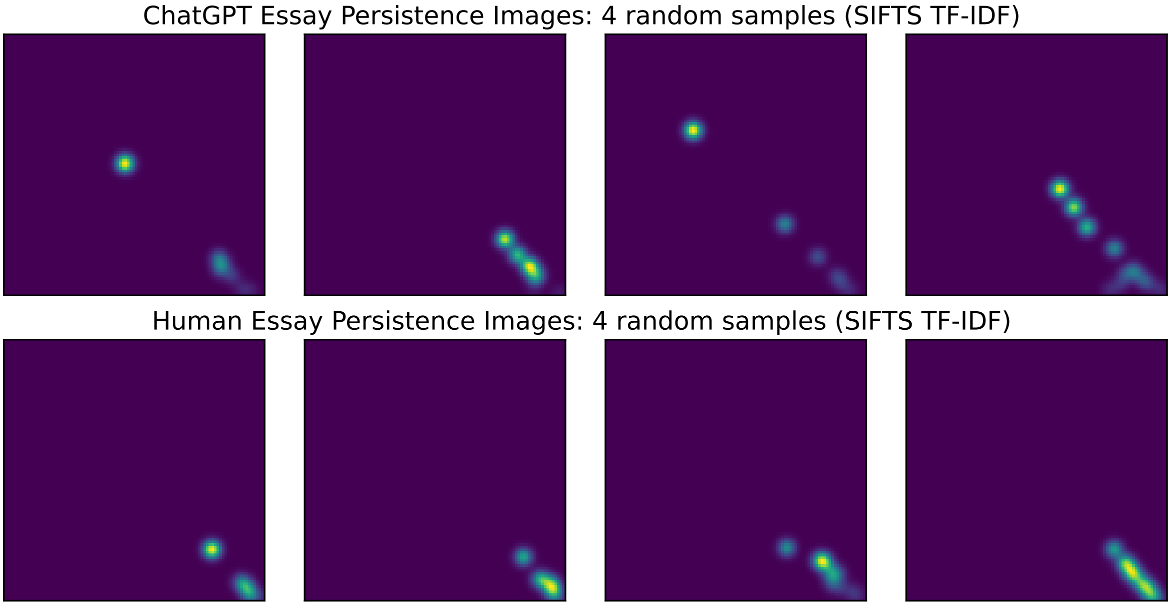


Figure 3.7: Examples of persistence images generated by applying SIFTS to the TF-IDF embeddings for a random selection of essays.

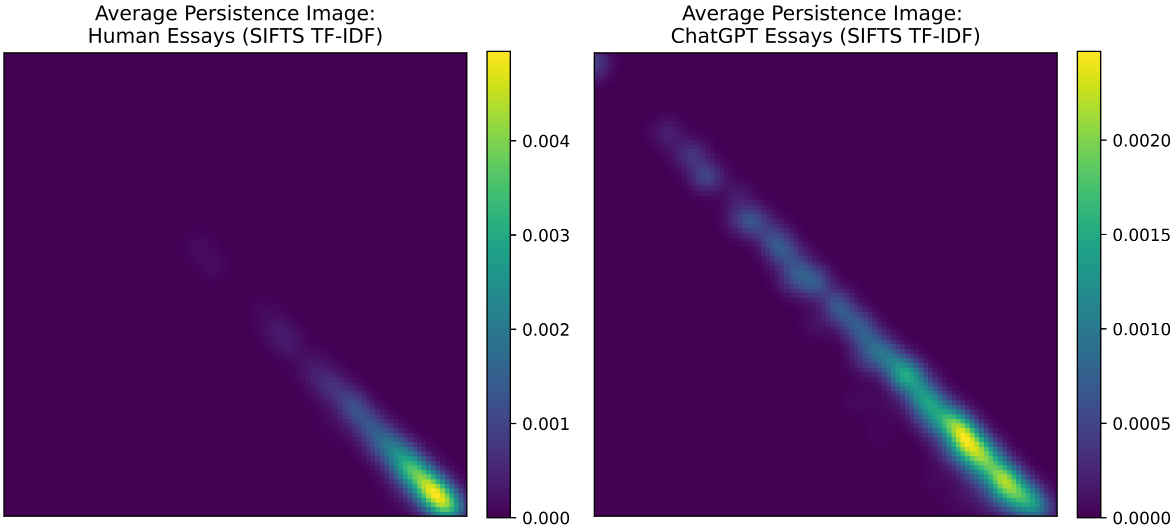


Figure 3.8: The average persistence images of the human-written and ChatGPT-written essays when applying SIFTS TF-IDF.

3.4 Discussion

In this section we discuss and summarize the results from both experiments.

Classification performance. The classification results from both experiments indicate that persistent homology effectively captures structural information that can differentiate between machine-written and human-written texts. These structural differences are clearly illustrated by Figures 3.3 - 3.8, which show that the persistence images obtained from machine-generated and human-written texts are often visibly different, at the very least on average.

Number of tie-backs. In addition to the classification results, a significant finding was the differences in the number of 1-dimensional holes generated by the human-written and machine-generated texts, when we applied the SIFTS algorithm on the TF-IDF vectors. In the Webtext/GPT-2 experiment, the average $|H_1|$ for human-written texts was 40.76, while the GPT-2 variants averaged at 27.84. As for the essays, the average $|H_1|$ of a human-written essay was 15.12, whereas a ChatGPT-generated essay had an average of 8.65. According to Zhu [27], a 1-dimensional hole can be interpreted as a semantic tie-back, and our results suggest that human-written texts contain significantly more tie-backs compared to machine-generated texts in both experiments. This finding is analog with Zhu's research [27], which demonstrated that texts authored by adolescents exhibited more 1-dimensional holes than those written by children. It is possible that this indicates humans often write in a more self-referential and coherent manner compared to current generative language models.

Vectorization methods. We created TDA-based machine learning classifiers with three different types of vectorization methods: the TF-IDF model, the GloVe model, and the Sentence-BERT model. Among these vectorization methods, Sentence-BERT demonstrated the best performance. This outcome is not surprising since Sentence-BERT is the most sophisticated embedding model. The Sentence-BERT embeddings take into account the semantic similarity of words and sentences, resulting in vectors that preserve more information about the structure of the original sentences. In contrast, the TF-IDF vectorization model considers two similar sentences with different words as completely different.

Perhaps surprisingly, the TDA classifiers based on the GloVe embedding method demonstrated the poorest performance. On paper, the GloVe embeddings should contain more information about the original texts than the TF-IDF embeddings, as the GloVe embeddings are designed to map semantically different words with similar meanings closely together, unlike TF-IDF. However, the TF-IDF classifier models outperformed the GloVe-based models. One possible explanation for this could be that the GloVe vectors we used had too low an embedding dimension for our specific application. We used 50-dimensional GloVe vectors, while pre-trained GloVe vectors of dimensions up to 300 were available and could have been used instead. Some limited testing with 100-dimensional GloVe vectors on the ChatGPT essay dataset showed a performance of 82.9% ± 5.7% using the SIF GloVe method, and with 300-dimensional GloVe vectors, a performance of 82.5% ± 3.9% (compared to 80.5%±5.6% for the 50-dimensional embeddings). Although not conclusive evidence, this suggests that choosing higher-dimensional GloVe embeddings somewhat improves classification results.

While the Sentence-BERT-based models often outperformed models based on other vector embeddings, the retrieval of Sentence-BERT vectors was significantly slower in our implementation. To illustrate this, generating the Sentence-BERT document matrices for 500 documents took approximately 15 minutes, whereas generating all the GloVe and TF-IDF document matrices on the computer used for the experiments only took a couple of seconds. The slow embedding speed of Sentence-BERT was a significant limiting factor in determining the size of the Webtext/GPT-2 datasets, leading us to conclude that TF-IDF vector embeddings were the best overall choice for the TDA-based classifier.

Choice of Classifier. Our TDA-based models generated persistence images, which were used as input for a classifier model of our choice. To prioritize the quality of the training data (i.e. the persistence images) rather than focusing on the capabilities of a specific state-of-the-art classifier model, we selected a well-established and relatively standard classifier model: the Support Vector Machine (SVM) for all of our TDA-based classifiers. It is possible that our decision to use the SVM classifier may have influenced the performance of our TDA-based classifiers, and further improvements could potentially be achieved by experimenting with other more advanced classifier models while utilizing the same persistence images as inputs.

As pointed out by Deng et al. [7], this study shares a similar limitation with many others that utilize TDA for classification: we invest significantly more effort in fine-tuning the TDA classifier methods compared to non-TDA approaches. In both experiments, we used the same XGBoost classifier model with TF-IDF unigram and bigram features as input. The model was fine-tuned by the authors of [23] for the essay experiment, and it is possible that we could have obtained better results in the Webtext/GPT-2 experiment by selecting a different classifier. For reference, OpenAI themselves have published a detection model with detection rates of around 95% on the GPT-2 XL versus Webtext dataset [24]. Nonetheless, the XGBoost model served as a useful non-TDA reference in our classifier experiments, as it was efficient and easy to implement. Furthermore, our lack of consideration for the choice of non-TDA model may be partly justified by the fact that this text primarily focuses on the application of TDA in classification problems.

Further research. There are many ways of vectorizing text not considered in this thesis, and it could be interesting to research the TDA-based classifier further with other text vectorization models. Related to vectorization techniques, is the choice of what to vectorize in the first place. While we separated our documents into sentences that were vectorized and used as a basis for persistent homology, experimenting with other ways of dividing documents into textual units is also a possibility that could be researched further. An example of already existing works on this is [11], where authors instead divide any document into 10 consecutive blocks of text. Furthermore, experimenting with metrics other than the angular distance that we used in the SIF and SIFTS algorithm might yield interesting results, although our findings suggested that the Euclidian metric was inferior.

In our experiments, we varied the method for generating persistence images (SIF and SIFTS), and used various embedding methods while keeping the final classifier constant. A natural extension to these experiments would be to vary the classifier model instead, while keeping the method of generating persistence images constant. For instance, one could explore using the SIFTS TF-IDF method, which demonstrated fast computation times and promising classification results. This approach could provide a better understanding of the performance of different classifier models and potentially improve the TDA classification method.

Another natural variation of the conducted experiments involves investigating the classification performance of TDA-based text classifiers using alternative vector representations of persistence diagrams. While we utilized persistence images, several other methods exist for transforming persistence diagrams into machine learning-friendly inputs. A summary of many of these methods can be found in [3].

Further work could be done in the area of assembling TDA-based and non-TDA-based classifiers. In our second experiment, we combined different TDA-based classifiers with the non-TDA-based classifier XGBoost, and saw classification performance that exceeded that of any single classifier. Deng and Duzhin [7] achieved promising results when assembling TDA-based classifiers with state-of-the-art deep learning models for natural language processing in a fake news detection experiment with little training data. It would be interesting to conduct a similar experiment focusing on machine-generated text detection.

Although our TDA-based methods demonstrated promising classification results for our selected datasets of machine-written and human-written texts, their performance on human/machine classification tasks for other text types remains unknown. As a side note, it would be interesting to study the performance of the TDA-based classifier method on texts that have been modified by humans, rather than focusing on texts generated solely by machines.

Chapter 4

Conclusion

The goal of the thesis was to provide an accessible introduction to the fields of Topological Data Analysis (TDA) and Natural Language Processing (NLP). Additionally, we explored the application of TDA for detecting machine-generated text.

Building upon existing research in the field, we developed a TDA-based text classification model that demonstrated promising results in distinguishing between human-written and machine-generated text in our experiments. The classification accuracy ranged from 80% to 96% depending on the text type and complexity of the generative language model. However, it remains uncertain whether this solid performance will extend to other datasets of machine-generated and human-written texts, as well as to the detection of documents co-written by humans and machines. Furthermore, as artificial text generation models advance, it is uncertain how the topological structure of their text outputs will evolve. It is possible that their topological signatures will eventually become indistinguishable from those of humans.

During our discussion, we made several suggestions for further research with TDA-based text classifiers. These include conducting experiments with alternative vectorization methods, exploring different classification algorithms, and investigating alternative approaches for transforming persistence diagrams into a format compatible with machine learning methods. To facilitate further exploration in this area, we have published the code developed for this thesis on GitHub [18]. Furthermore, a detailed explanation of the code can be found in Appendix A. We consider the publication of this code to be one of the key contributions of this work.

Appendix A

Implementation

In this chapter we present an implementation in Python of the machine learning pipelines described in Chapter 3. The full repository, including the datasets of text and a demonstration of the experiments, is available on GitHub [18].

A.1 Text Preprocessing

First we import libraries involved in the processing of strings in Python:

```
import nltk              # Natural Language Toolkit
import re                # Regular expressions operations
import contractions      # Contractions
```

We then define functions for removing stop words (common words like ”the”, ”a”, etc.) and expanding contractions (i.e. changing ”don’t” to ”do not”) for a given text document.

```
# Generate a list of english stop words using NLTK.
sw_nltk = nltk.corpus.stopwords.words('english')

# A function that removes stop words and expands contractions in a single line.
def remove_stopwords_line(line):
    # Remove contractions in the string (i.e. switching 'don't' to 'do not'.)
    line = ' '.join([contractions.fix(expanded_word)
                     for expanded_word in line.split()])
    # Remove stop words by removing the words that are in sw_nltk.
    line = ' '.join([word for word in line.split() if word not in sw_nltk])
    return line

# A function that removes stopwords and expands contractions
# in document consisting of multiple lines.
def remove_stopwords_document(document):
    # Create a list consisting of each line in the document.
    lines = list(filter(None, document.splitlines()))
    # Perform stop word removal and contraction expanding on each line
    # and join the modified strings back together.
    return'\n'.join([remove_stopwords_line(line) for line in lines])
```

Now we are ready to define a function that performs the steps in the preprocessing pipeline (except lemmatization, which we perform separately). The operations are performed on a `pandas` dataframe consisting of text documents, on the column named ’text’.

```
import pandas as pd

# Performs preprocessing steps on a dataset consisting of text documents
def preprocess_df(dataframe):
    # Create a copy of the input dataframe
    df = dataframe.copy()

    # Split the texts into sentences into a format of one sentence per row
    df['text'] = df['text'].apply(lambda x: '\n'.join(nltk.sent_tokenize(x)))

    # Convert all letters to lower-case.
    df['text'] = df['text'].str.lower()

    # Expand contractions (don't --> do not)
    # and remove all stop words from each text document
    df['text'] = df['text'].apply(remove_stopwords_document)

    # Remove punctuation, numbers and special characters
    df['text'] = df['text'].apply(lambda x: x.replace('-',' '))
    df['text'] = df['text'].apply(lambda x: re.sub(r'[^ \nA-Za-z.?!]+', '', x))
    df['text'] = df['text'].apply(lambda x: re.sub(r'[.?!]+', '', x))

    # Remove any excess white-space like spaces and newlines
    df['text'] = df['text'].apply(lambda x: re.sub(' +', ' ', x))
    df['text'] = df['text'].apply(lambda x: re.sub('\n+', '\n', x))
    df['text'] = df['text'].apply(lambda x: '\n'.join([line.strip()
                                                       for line in x.splitlines()]))

    return df
```

Finally, we define a function that lemmatizes each text in a given dataset:

```
# Load the lemmatizer from the NLTK-library.
lemma = nltk.WordNetLemmatizer()

# Define a function that performs lemmatization on a text document
def lemmatize(document):
    # Create a list consisting of each line in the document
    lines = document.splitlines()
    # Make an empty list that will be filled with
    # the lemmatized versions of the lines in the document.
    lemmatized_lines = []

    # perform lemmatization on each line in the document separately,
    # and rejoin them together
    for line in lines:
        lemmatized_lines.append(" ".join([lemma.lemmatize(item)
                                          for item in line.split()]))
    return "\n".join(lemmatized_lines)
```

Given a dataset of text documents with directory `"dataset.csv"`, we may now perform the preprocessing pipeline in Chapter 2.2.2 with the following three lines of code:

```
df = pd.read_csv("dataset.csv")
df["text"] = preprocess_df(df)
df["text_lemmatized"] = df["text"].apply(lemmatize)
```

The column ”text_lemmatized” may now be used as input in a machine learning pipeline.

A.2 Text Vectorization

After the text documents have been preprocessed, lemmatized and split into sentences, they are ready for vectorization. Here we implement functions whose input is a document, and whose output is a document embedding matrix, where the i-th row in the matrix is the sentence vector of the i-th sentence in a given document. We define document embedding functions for three vectorization models: TF-IDF, Sentence-BERT and GloVe.

TF-IDF embedding

```
# Import the TF-IDF vectorizer from the sklearn library
from sklearn.feature_extraction.text import TfidfVectorizer

def tfidf_matrix(document):
    # Initialize the TF-IDF vectorizer. Define terms to be unigrams and bigrams.
    vectorizer = TfidfVectorizer(ngram_range=(1,2))

    # Split the document into lines (which represent sentences)
    lines = document.splitlines()

    # Produce a TF-IDF vector corresponding to each line and stack them in a matrix
    matrix = vectorizer.fit_transform(lines).toarray()

    return matrix
```

Sentence-BERT embedding

```
# Load a pre-trained sentence-BERT model from sentence_transformers
from sentence_transformers import SentenceTransformer
sbert_model = SentenceTransformer('all-MiniLM-L6-v2')

def sbert_matrix(document):
    # Split the document into lines (which represent sentences)
    lines = document.splitlines()

    # Generate an s-BERT embedding for each line and stack them in a matrix
    matrix = sbert_model.encode(lines)

    return matrix
```

GloVe embedding

To generate GloVe sentence embeddings, we first download a text-file containing pre-trained GloVe word embeddings from [21]. We then load these word embeddings into a Python dictionary whose keys are words, and whose values are their respective embeddings.

```python
glove_file = "D:\\glove_file_directory"

def load_glove_model(glove_file):
    with open(glove_file, encoding="utf8" ) as f:
        content = f.readlines()

    model = {}

    for line in content:
        split_line = line.split()
        word = split_line[0]
        vector = np.array([float(val) for val in split_line[1:]])
        model[word] = vector

    return model

# Load the GloVe model from the given directory
glove = load_glove_model(glove_file)
```

Next, we create a function that returns the GloVe embedding of a given word if it exists, and a 50-dimensional zero vector if not.

```python
def glove_vectorize(word):
    try:
        vector = glove[word]
        return vector
    except KeyError:
        return np.zeros(50)
```

Finally, we define a function that returns the GloVe sentence embeddings of a document:

```python
def glove_matrix(document):
    lines = document.splitlines()
    matrix = []

    for line in lines:
        if not line == "":
            words = nltk.word_tokenize(line)
            word_vectors = [glove_vectorize(word) for word in words]
            sentence_vector = sum(word_vectors) / len(word_vectors)

            # Only include the sentence-vector if it is non-zero
            if np.count_nonzero(sentence_vector) > 1:
                matrix.append(sentence_vector)

    return np.array(matrix)
```

A.3 Metric on Text Embeddings

Here we define a function that calculates the pairwise angular distances between all sentence vectors in a document embedding matrix. The input of the function is a document embedding matrix A, whose i-th row in the matrix is the sentence vector $\mathbf{x_i}$ of the i-th sentence in a document. The output of the function is a matrix D whose $[i, j]$-term is $D(\mathbf{x_i}, \mathbf{x_j})$ in the normalized angular distance metric.

```
import numpy as np

def angular_distance_matrix(A):
    if A.ndim == 1:
        return np.array([0])

    n_row, n_col = A.shape

    # The matrix where the (i,j)-th entry is the dot product
    #between row vector x_i and row vector x_j of A
    K = A @ A.transpose()

    # The array where the i-th entry is the
    #norm of the i-th row vector of A
    normA = np.sqrt(np.diag(K))

    # Smooth out by making zero-entries non-zero
    normA[normA == 0] = 1e-10

    # Tile the norms of A such that X_i * X_j is the matrix where the
    #(i,j)-th entry is ||x_i||*||x_j||
    X_i = np.tile(normA, (n_row, 1))
    X_j = X_i.transpose()

    # The (i,j)-th entry is the cosine similarity of x_i and x_j
    cs = K / (X_i * X_j)

    # Avoid potential numerical issues when using the arccos function
    cs[cs>1] = 1
    cs[cs<0] = 0

    # Calculates the normalized angular distance matrix of A
    D = 2 * np.arccos(cs) / np.pi

    # Ensure that the diagonal is exactly zero
    for i in range(len(D[0])):
        D[i,i] = 0

    return D
```

A distance matrix D contains all information about the pairwise distances in a point cloud representing a text, and therefore contains the necessary information to generate a filtration of Vietoris Rips complexes, and calculate its persistent homology. Moving on, these distance matrices will be used as input when applying TDA on text.

A.4 SIF and SIFTS

Here we utilize the TDA-library `gudhi` to implement algorithms SIF and SIFTS introduced by Zhu [27]. The input in these functions is a distance matrix D, and the output is a list of persistence intervals representing the persistent homology of a given text.

```
import gudhi as gd

def SIF(D):
    # Create a Vietoris Rips filtration for a distance matrix D
    # with simplices of maximum dimension 2.
    VR_complex = gd.RipsComplex(distance_matrix = D)
    filtration = VR_complex.create_simplex_tree(max_dimension = 2)

    #Calculate the persistent homology in dimensions 0 and 1
    persistence_intervals = filtration.persistence()

    return persistence_intervals


def SIFTS(D):
    # Create a copy of the distance matrix D
    D_SIFTS = D.copy()

    # Modify the the distance matrix by requiring that D(x_i,x_{i+1}) = 0
    for i in range(len(D_SIFTS[0])-1):

        D_SIFTS[i,i+1]=0
        D_SIFTS[i+1,i]=0

    # Create a Vietoris Rips filtration for a distance matrix D_SIFTS
    # with simplices of maximum dimension 2.
    VR_complex = gd.RipsComplex(distance_matrix = D_SIFTS)
    filtration = VR_complex.create_simplex_tree(max_dimension = 2)

    #Calculate the persistent homology in dimensions 0 and 1
    persistence_intervals = filtration.persistence()

    return persistence_intervals
```

A.5 Machine Learning Pipeline

We have now defined the SIF and SIFTS functions that return the persistence diagrams in dimensions 0 and 1 for a given distance matrix D. The next step in the TDA based classifier is to convert these persistence diagrams in a fixed dimension to persistence images, which we implement using `PersistenceImager` in the library `persim`.

We begin by writing a function that restricts a list of persistence intervals to a specific dimension, and converts these intervals to a format compatible with `PersistenceImager`. In particular, we convert a list of Python tuples of the form $(\text{dimension}, (\text{birth}, \text{death}))$ to a list with elements of the form $[\text{birth}, \text{death}]$ in a fixed dimension:

```
from persim import PersistenceImager

# Returns a list of the persistence intervals in a fixed dimension n
def intervals_in_dimension(n, persistence_intervals):
    list_bd = []
    for item in persistence_intervals:

        if item[0] == n:

            # Append elements of form [birth,death] to the new list
            # Exclude potential intervals with infinite persistence in dim 0,
            # because they are incompatible with persistence images
            if item[1][1] != np.inf:

                list_bd.append(list(item[1]))
    return list_bd
```

For flexibility and generality in the final TDA classifier pipeline, we now define a function `text_to_matrix` that returns the text vectorization function of a desired vectorization model, ”tfidf”, ”sbert” or ”glove”:

```
def text_to_matrix(embedding):
    if embedding == "tfidf":
        return tfidf_matrix
    elif embedding == "sbert":
        return sbert_matrix
    else:
        return glove_matrix
```

We are now ready to define the function `text_to_image_transformer` that transforms an entire dataset of (lemmatized and preprocessed) text documents into a dataset of persistence images, that will be used as input in a machine learning classifier. To support experimentation with the parameters (such as image resoluton and vector embedding model) in the TDA based classifiers, the function takes in multiple arguments, that are explained in the comments of the code.

```
def text_to_image_transformer(algorithm, embedding, dimension, sigma, grid_size):
    '''
    Returns a function transformer compatible with machine learning pipelines
    in sklearn, that transforms an array of preprocessed text documents to an
    array of corresponding persistence images, with user specified parameters
    for a given embedding method.

    PARAMETERS:

    algorithm : {"SIF", "SIFTS"}
    Choose which TDA-algorithm to apply on the text embeddings.

    embedding : {"tfidf", "sbert", "glove"}
    Selects which vector embedding method to use on the texts.

    dimension : {0, 1}
    The dimension of the persistence diagram to generate a persistence image from.

    sigma     : float
    The parameter sigma in the Gaussian distribution used in persistence images.

    grid_size : int
    Select persistence image resolution, which will be (grid_size * grid_size).
    '''

    # Define a function that transforms a text to a
    # persistence image with desired parameters.
    def text_to_image(text):
        A = text_to_matrix(embedding)(text)
        D = angular_distance_matrix(A)

        if algorithm == "SIFTS":
            intervals = intervals_in_dimension(dimension,SIFTS(D))
        else:
            intervals = intervals_in_dimension(dimension,SIF(D))

        pimgr = PersistenceImager(pixel_size    = 1 / grid_size,
                                  birth_range   = (0, 1),
                                  pers_range    = (0, 1),
                                  kernel_params = {'sigma': sigma})

        imgs = pimgr.transform(intervals)
        imgs_array = imgs.flatten()
        return imgs_array

    # Define a function transforms an array of texts
    # into an array of corresponding persistence images
    def corpus_to_images(array):
        return np.array([text_to_image(item) for item in array])

    return FunctionTransformer(corpus_to_images)
```

Conducting an experiment on the ChatGPT essay dataset may now look like the following. For more details, we refer the reader to [18].

```python
import xgboost as xgb
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.pipeline import Pipeline
from sklearn import svm
from sklearn.ensemble import VotingClassifier
from sklearn.model_selection import cross_val_score

# Load the essay dataset
df = pd.read_csv("datasets/essays.csv")

# The training/testing inputs in the pipelines consists of lemmatized texts
X = df["text_lemmatized"].values

# Corresponding labels for each text. 1 for machine-generated, 0 for human-written.
y = df["label"].values

# Create TF-IDF and XGBoost based machine learning classifiers + an ensemble:

tfidf_transformer = text_to_image_transformer(embedding  ="tfidf",
                                              algorithm  ="SIFTS",
                                              dimension  = 1,
                                              sigma      = .001,
                                              grid_size  = 100)

# Make a TDA classifier on persistence images from TF-IDF embeddings
tfidf_model = Pipeline([("persistence images", tfidf_transformer),
                        ("classifier",  svm.SVC(probability=True))])

# Make a non-TDA classifier with XGBoost on TF-IDF unigrams and bigram features
xgboost_model = Pipeline([('tfidf features',
                            TfidfVectorizer(ngram_range=(1,2),min_df=5)),
                          ('classifier',
                            xgb.XGBClassifier())])

# Make an ensemble of the two previous classifier models
ensemble = Pipeline([['ensemble',
                    VotingClassifier(voting="soft",
                                    estimators=[("tfidf",tfidf_model),
                                                ("xgboost", xgboost_model)])]])


models = [tfidf_model, xgboost_model, ensemble]
names = ["TF-IDF", "XGBoost", "TF-IDF/XGBoost Ensemble"]

# Evaluate the machine learning models using 5-fold cross valuation
for i in range(len(models)):
    model, name = models[i], names[i]
    print(name)
    scores = cross_val_score(model, X=X, y=y, cv=5)
    print("Scores:", scores.round(3))
    print('Cross Validation accuracy: %.3f +/- %.3f \n'
          % (np.mean(scores),np.std(scores)))
```

Running the code yields the following output:

```
TF-IDF
Scores: [0.804 0.863 0.88  0.8   0.88 ]
Cross Validation accuracy: 0.845 +/- 0.036

XGBoost
Scores: [0.902 0.941 0.9   0.88  0.92 ]
Cross Validation accuracy: 0.909 +/- 0.021

TF-IDF/XGBoost Ensemble
Scores: [0.902 0.941 0.94  0.92  1.   ]
Cross Validation accuracy: 0.941 +/- 0.033
```

A.6 TDA Pipeline Demonstration

Here we demonstrate the TDA pipeline on a document from the ChatGPT essay dataset.

Orignal Text:

In my opinion an important skill that a person should learn in order to be successful in today's world is to keep him updated in every field. It is said that knowledge is power. Keeping oneself updated in this high speed, changing world is very important. Knowledge is like a small drop in this vast ocean and there is no end for it. The more knowledge you gain, the more you need.

One of the advantages of keeping oneself aware is that nobody can mislead you in anyway. For example if you know the price of a car, you want to buy, would help you negotiate more rather than just agree to the dealers price. Keeping ourselves updated also boosts our confidence and also keeps us ahead in this competitive world.

We should develop this skill by listening to news, reading articles, learning new technologies. In short we should always keep on increasing our knowledge. If we do this then rest of the thing like wealth, fame, comforts will automatically come to us. Hence keeping yourself updated is very important skill everybody should acquire in today's world.

Preprocessed and Lemmatized Text:

opinion important skill person learn order successful today world keep updated every field said knowledge power
keeping oneself updated high speed changing world important
knowledge like small drop vast ocean end it
knowledge gain need
one advantage keeping oneself aware nobody mislead anyway
example know price car want buy would help negotiate rather agree dealer price
keeping updated also boost confidence also keep u ahead competitive world
develop skill listening news reading article learning new technology
short always keep increasing knowledge
rest thing like wealth fame comfort automatically come u
hence keeping updated important skill everybody acquire today world

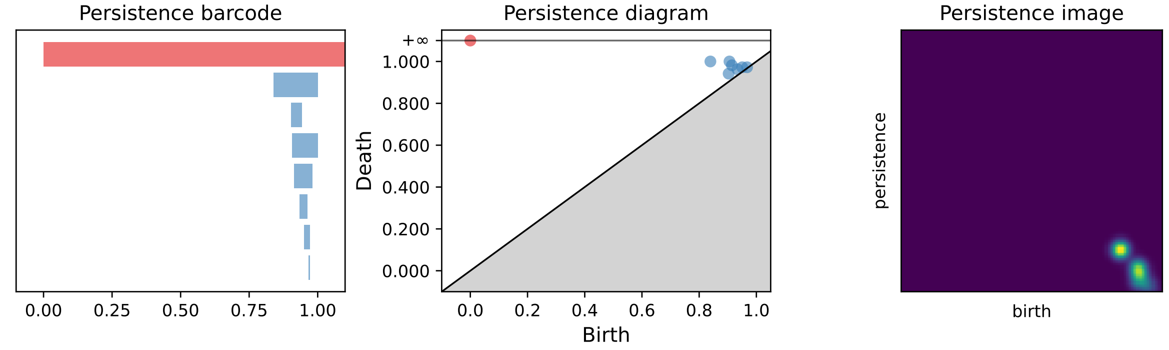


Figure A.1: Output from applying SIFTS TF-IDF on the proprocessed lemmatized text.

Bibliography

[1] H. Adams, "Persistence images: A stable vector representation of persistent homology," *Journal of Machine Learning Research, 18, 135*, 2017.

[2] M. Anandarajan, C. Hill, and T. Nolan, "Text preprocessing," in *Practical Text Analytics: Maximizing the Value of Text Data.* Cham: Springer International Publishing, 2019, pp. 45–59, ISBN: 978-3-319-95663-3. DOI: `10.1007/978-3-319-95663-3_4`. [Online]. Available: `https://doi.org/10.1007/978-3-319-95663-3_4`.

[3] D. Barnes, L. Polanco, and J. A. Perea, "A comparative study of machine learning methods for persistence diagrams," *Frontiers in Artificial Intelligence*, vol. 4, 2021, ISSN: 2624-8212. DOI: `10.3389/frai.2021.681174`. [Online]. Available: `https://www.frontiersin.org/articles/10.3389/frai.2021.681174`.

[4] M. Botnan, *Topological Data Analysis.* 2022. [Online]. Available: `https://www.few.vu.nl/~botnan/lecture_notes.pdf`.

[5] T. Chen and C. Guestrin, "XGBoost," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, Aug. 2016. DOI: `10.1145/2939672.2939785`.

[6] S. Das, S. A. Haque, and M. I. Tanveer, *Persistence homology of tedtalk: Do sentence embeddings have a topological shape?* 2021. arXiv: `2103.14131 [cs.LG]`.

[7] R. Deng and F. Duzhin, "Topological data analysis helps to improve accuracy of deep learning models for fake news detection trained on very small training sets," *Big Data Cogn. Comput.*, 2022. DOI: `10.3390/bdcc6030074`.

[8] P. Doshi and W. Zadrozny, "Movie genre detection using topological data analysis: 6th international conference, slsp 2018, mons, belgium, october 15–16, 2018, proceedings," in Jan. 2018, pp. 117–128, ISBN: 978-3-030-00809-3. DOI: `10.1007/978-3-030-00810-9_11`.

[9] H. Edelsbrunner and J. Harer, *Computational Topology, An Introduction.* The American Mathematical Society, 2010.

[10] Q. Fang, D. Nguyen, and D. L. Oberski, "Evaluating the construct validity of text embeddings with application to survey questions," vol. 11, no. 1, 2022. DOI: `10.1140/epjds/s13688-022-00353-7`.

[11] S. Gholizadeh, K. Savle, A. Seyeditabari, and W. Zadrozny, *Topological data analysis in text classification: Extracting features with additive information*, 2020. arXiv: `2003.13138 [cs.LG]`.

[12] S. Gholizadeh, A. Seyeditabari, and W. Zadrozny, "Topological signature of 19th century novelists: Persistent homology in text mining," *Big Data and Cognitive Computing*, vol. 2, p. 33, Oct. 2018. DOI: `10.3390/bdcc2040033`.

[13] J.-C. Hausmann, "On the vietoris-rips complexes and a cohomology theory for metric spaces," in *Proceedings of a Conference in Honor of William Browder. (AM-*

138), F. Quinn, Ed. Princeton: Princeton University Press, 1996, pp. 175–188, ISBN: 9781400882588. DOI: `doi:10.1515/9781400882588-013`.

[14] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013. [Online]. Available: `https://hastie.su.domains/ISLR2/ISLRv2_website.pdf`.

[15] A. J. Joshi, "Natural language processing," *Science*, vol. 253, no. 5025, pp. 1242–1249, 1991, ISSN: 00368075, 10959203. [Online]. Available: `http://www.jstor.org/stable/2879169` (visited on 04/14/2023).

[16] G. Kusano, K. Fukumizu, and Y. Hiraoka, *Persistence weighted gaussian kernel for topological data analysis*, 2016. arXiv: `1601.01741`.

[17] L. Kushnareva, D. Cherniavskii, V. Mikhailov, *et al.*, "Artificial text detection via examining the topology of attention maps," Jan. 2021, pp. 635–649. DOI: `10.18653/v1/2021.emnlp-main.50`.

[18] B. Løvlie, *Text Classification via Topological Data Analysis.* [Online]. Available: `https://github.com/bendilo/TDA_text_classification`.

[19] D. Munteau, "Vector space model for document representation in information retrieval," *Annals of Dunarea de Jos*, 2017.

[20] N. Otter, M. A. Porter, U. Tillmann, P. Grindrod, and H. A. Harrington, "A roadmap for the computation of persistent homology," *EPJ Data Science*, vol. 6, no. 1, Aug. 2017. DOI: `10.1140/epjds/s13688-017-0109-5`.

[21] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: `http://www.aclweb.org/anthology/D14-1162`.

[22] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2018. [Online]. Available: `https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf`.

[23] R. Shijaku and E. Canhasi, "Chatgpt generated text detection," 2023. DOI: `10.13140/RG.2.2.21317.52960`.

[24] I. Solaiman, M. Brundage, J. Clark, *et al.*, *Release strategies and the social impacts of language models*, 2019. arXiv: `1908.09203 [cs.CL]`.

[25] Z. Virk, *Introduction to Persistent Homology.* 2022. DOI: `10.51939/0002`.

[26] R. Zhao and K. Mao, "Fuzzy bag-of-words model for document representation," *IEEE Transactions on Fuzzy Systems*, 2017.

[27] X. Zhu, "Persistent homology: An introduction and a new text representation for natural language processing," *Proceedings of the 23rd International Joint Conference on Artificial Intelligence*, 2013. [Online]. Available: `https://www.ijcai.org/Proceedings/13/Papers/288.pdf`.

NTNU

Norwegian University of
Science and Technology