ANALYZING STATE OF THE UNION ADDRESSES USING TOPOLOGY

DILLON KOUGH, MARIA NEUZIL, AND CHEYANNE SIMPSON
ADVISOR: REBECCA GLOVER

Abstract. In this paper, we investigate the implementation of Topological Data Analysis (TDA) with Natural Language Processing (NLP). We introduce the mathematical foundations of persistent homology, the main tool of TDA, and provide the theoretical framework for its application in determining the topological structure of a text. This includes a discussion on how NLP is used to prepare a text for such analysis. We then apply these tools to a collection of the State of the Union addresses. We discuss the meaning of our results and propose additional inquiries to help establish the role of TDA with NLP.

1. Introduction

Topological Data Analysis (TDA) is a relatively recent development in applied mathematics. It takes large, noisy data sets and uses a tool called *persistent homology* to identify their topological structures: points forming clusters, points forming loops, points enclosing a three-dimensional void, and so on. This powerful technique is more general than traditional modeling tools, such as cluster finders, because it can identify all of these topological features with one process. This burgeoning new field has already been applied to a wide breadth of topics: economics, medicine, neuroscience, and machine learning, to name a few [1].

Recently, there has been some work applying TDA to textual data sets [2, 3, 4]. The goal is to identify the structure of a text, as characterized by its word usage throughout the document. For example, in [2], repetitive nursery rhymes are examined by finding "loops" in the text as similar themes are returned to over and over again. In [3], full novels are examined topologically by comparing how main characters are discussed in each novel using semantic analysis. We are interested in exploring this application further to see if TDA can meaningfully characterize the structure of speeches, as there is a natural flow of speech patterns. This includes circling back to previous ideas, creating these topological "loops."

In this project, we apply TDA and NLP techniques to a selection of the State of the Union addresses to see if we can characterize their topological structure. The State of the Union is the president's annual address to a joint session of Congress, reviewing the previous year and enumerating the coming agenda. These addresses provide many opportunities for comparisons, such as comparing the structure of a president's speeches against his other speeches or comparing different presidents' speeches to each other. We examine these comparisons in the hopes of determining whether TDA could be useful in differentiating between the speech patterns of different presidents.

This paper is organized as follows: in Section 2, we present the background information necessary to understand our project. This includes a mathematical discussion of group theory (Section 2.1), general topology (Section 2.2), and homology (Section 2.3). We also discuss the foundations of NLP in Section 2.5 and show how it can be used with TDA in Section 2.6. Our methods are explained in Section 3, and we present our results and conclusion in Sections 4 and 5, respectively. We finish with a discussion of possible future work that could be done to extend this project in Section 6.

2. Background

2.1. **Group Theory.** Before we discuss the main tool of topological data analysis, persistent homolgy, we introduce some basic ideas from group theory. This will provide the theoretical foundations that will help us explain how we can determine the topological structure of an object. For the mathematical formalism, we follow the conventions presented in [2].

Date: August 31, 2019.

Definition 2.1. An *abelian group* $\langle G, * \rangle$ is a set G with a binary operation $*$ such that

(1) Associativity: $a * (b * c) = (a * b) * c \quad \forall\, a, b, c \in G$
(2) Identity: $\exists\, e \in G$ such that $e * a = a * e = a \quad \forall\, a \in G$
(3) Inverse: $\forall\, a \in G$, $\exists\, a^{-1}$ such that $a * a^{-1} = a^{-1} * a = e$
(4) Commutativity: $a * b = b * a \quad \forall\, a, b \in G$

Note that properties $(1) - (3)$ in Definition 2.1 define a group; property (4) makes it what is called *abelian*. We often drop the notation $\langle G, * \rangle$ and just refer to G as a group when the operation is understood.

Example 2.2. Some familiar abelian groups include:

- $\langle \mathbb{Z}, + \rangle$, where $+$ is regular addition, the identity is 0 and the inverse of $z \in \mathbb{Z}$ is $-z$.
- $\langle \mathbb{R} \backslash \{0\}, * \rangle$, where $*$ is regular multiplication, the identity is 1 and the inverse of $r \in \mathbb{R} \backslash \{0\}$ is $1/r$.

Another example of an abelian group, which will be useful later, involves what is called a *power set*. Given a set A, its power set is the set containing all possible subsets of A. If we take this power set and define our operation to be the symmetric difference, we form a group. The identity is $\emptyset$ and the inverse of $S \subseteq A$ is S.

Some groups can be divided into pieces, called *cosets*, which will be used when calculating homology in Section 2.3.

Definition 2.3. A subset $H \subseteq G$ of a group $\langle G, * \rangle$ is a *subgroup* of G if $\langle H, * \rangle$ is also a group.

Definition 2.4. If H is a subgroup of an abelian group G, for a given $a \in G$ the set $a * H = \{a * h \,|\, h \in H\}$ is called the *coset* of H represented by a.

Example 2.5. Consider again the group $\langle \mathbb{R} \backslash \{0\}, * \rangle$. If we let $\mathbb{R}_+$ ($\mathbb{R}_-$) denote the positive (negative) real numbers, then $\langle \mathbb{R}_+, * \rangle$ is a subgroup of $\langle \mathbb{R} \backslash \{0\}, * \rangle$. If we pick any positive real number r, then $r * \mathbb{R}_+ = \mathbb{R}_+$ itself is the coset of $\mathbb{R}_+$ represented by r.

On the other hand, $\langle \mathbb{R}_-, * \rangle$ is not a subgroup of $\mathbb{R} \backslash \{0\}$: $\mathbb{R}_-$ is not closed under $*$. But, $\mathbb{R}_-$ is the coset of $\mathbb{R}_+$ represented by any negative real number $-r$: $-r * \mathbb{R}_+ = \mathbb{R}_-$.

Note that these two cosets, $\mathbb{R}_-$ and $\mathbb{R}_+$, form a partition for $\mathbb{R} \backslash \{0\}$. This will be important when calculating homology groups later.

We can use a subgroup and its cosets to define a new kind of group. Consider the binary operation $\star$ that acts on the cosets of H: $(a * H) \star (b * H) = (a * b) * H \quad \forall\, a, b \in G$.

Definition 2.6. The cosets of H with the operation $\star$ form a group called the *quotient group*, denoted G/H.

The quotient group will be useful later when trying to count the topological features of an object, like connected components and holes. Before we can connect group theory to topology, we need to introduce a few more definitions.

Definition 2.7. Given $S \subset G$, the *subgroup generated by* S, denoted $\langle S \rangle$, is the group formed by all elements of G that can be expressed as finite operations of elements of S and their inverses.

Example 2.8. Consider the group $\langle \mathbb{Z}, + \rangle$, and let $S = \{2\}$. Here performing "finite operations" of elements of S and their inverses just means we can take 2 or -2 and add as many of them as we please. This will form the set $\{2z \,|\, z \in \mathbb{Z}\}$, which is in fact a group under addition.

The subgroup generated by S could be equal to G itself. If such an S is minimal, it provides us with an important classification of G, its *rank*.

Definition 2.9. The *rank* of G is defined to be the size of the smallest subset that generates G.

Example 2.10. The rank of $\langle \mathbb{Z}, + \rangle$ is 1, with $S = \{1\}$. Any integer can be expressed as a finite addition of 1's or -1's.

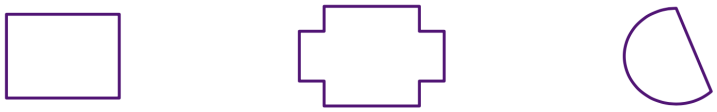
We have now encountered the concepts from group theory that will be most relevant for our paper. Rank, in particular, will be an important tool for counting topological features like connected components and holes. Now, we introduce topology and then the way group theory arises in the shapes we study.

2.2. **General Topology.** Topology is a branch of math that studies spaces under continuous transformations, called homeomorphisms. We adopt a more flexible perspective than the traditional geometric one: we don't think of geometric shapes as being fixed, unchangeable objects. Rather, we can stretch and squish shapes without changing some of their fundamental properties. Topology is sometimes called "rubber sheet geometry:" we can imagine that our shapes are made out of rubber, so we can freely morph them in many ways.

Definition 2.11. A *homeomorphism* is a function that is bijective, continuous, and has a continuous inverse.

Qualitatively, a homeomorphism allows us to smoothly transform back and forth between two spaces. In terms of geometric shapes, under a homeomorphism an object can be stretched, squished, bent, twisted, and so forth; it cannot be punctured, torn, glued, or have pieces deleted or added. Two objects are considered *topologically equivalent* if there exists a homeomorphism between them.

Example 2.12. The following shapes are topologically equivalent as any one can be continuously transformed into another. Note that they share some properties, despite being geometrically different: for example, they all have one connected component, and they all have one hole. These properties are important for topological data analysis.



Using topology, given an n-dimensional geometric shape we can create an object called a *simplicial complex* that is topologically equivalent to that shape. Simplicial complexes are composite structures made of basic building blocks called p-simplices, where p denotes the dimension.

Definition 2.13. A *p-simplex* σ is the convex hull of $p+1$ affinely independent points $x_0, x_1, \ldots, x_p$. A *face* τ of σ is the convex hull of some proper subset of $x_0, x_1, \ldots, x_p$ and is denoted $\tau \leq \sigma$.

Intuitively, a simplicial complex breaks up an n-dimensional topological object into simple geometric shapes that fit together to form the object. The first four p-simplices are shown in Figure 1. A 0-simplex is a vertex; a 1-simplex, and edge; a 2-simplex, a triangle; a 3-simplex, a tetrahedron. The faces of a given p-simplex are the lower dimensional simplex constituents.

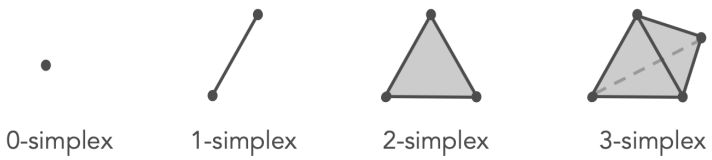
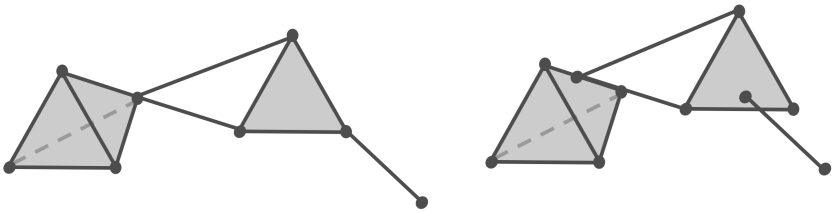


FIGURE 1. The first four p-simplices.

Definition 2.14. A *simplicial complex* K is a finite collection of simplices such that $\sigma \in K$ and $\tau \leq \sigma$ implies $\tau \in K$, and $\sigma_0, \sigma_1 \in K$ implies $\sigma_0 \cap \sigma_1$ is either empty or a face of both.

An example and a non-example of a simplicial complex are shown in Figure 2. The simplex components must be connected by their faces, and all of their faces must belong to the complex. However, a simplicial complex need not be connected.



(A) A simplicial complex. (B) Not a simplicial complex.

FIGURE 2. p-simplices must be connected at their faces to form a simplicial complex.

Certain subsets of a simplicial complex will give us information about its shape, including the number of connected components and the number of holes. The first subset that we use is called a *p-chain*.

Definition 2.15. A *p-chain* of a simplicial complex is a subset of its p-simplices.

Despite the name "chain," a p-chain need not be connected. Given a simplicial complex K, a 0-chain is just a subset of its vertices; a 1-chain, its edges; a 2-chain, its triangles, and so on. For example, the orange edges in Figure 3 form a 1-chain.

For a given p, all the subsets of p-simplices form its power set. Recall from Example 2.2 that a power set with the finite difference operation forms a group.

Definition 2.16. Given a simplicial complex K, its set of p-chains forms a *p-chain group* denoted C_p.

This is the first of a number of groups that can arise from a simplicial complex. We need a few more definitions before they can be introduced.

Definition 2.17. The *boundary* of a p-simplex is the collection of its $(p-1)$-simplex faces. The boundary of a p-chain is the symmetric difference of the boundaries of its p-simplices.

In Figure 3, the boundary of the orange 1-chain contains the two green vertices. Notice that some 1-chains will have an empty boundary: for example, the 1-chain that borders the triangle on the right. These p-chains with empty boundaries form our next group.

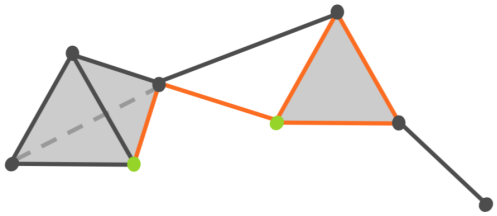


FIGURE 3. The orange edges form a 1-chain of this simplicial complex. The boundary of this 1-chain is the collection of green vertices.

Definition 2.18. The set of p-chains with empty boundaries form a group called the *p-cycle group*, denoted Z_p.

Figure 4 highlights all of the 1-chains with empty boundaries, the 1-cycles, in a particular simplicial complex. Notice that the pink cycle itself is the boundary of the 2-simplex. The orange cycles are not the boundary of any such 2-chain because of the empty rectangular hole. This brings us to our next group.

Definition 2.19. A *p-boundary-cycle* is a p-cycle that is the boundary of some $(p+1)$-chain. The p-boundary-cycles form a group, denoted B_p.

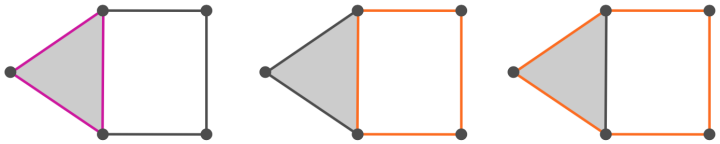


FIGURE 4. The pink and orange 1-cycles are the three 1-chains with empty boundaries in this simplicial complex. The pink cycle is unique because it is the boundary of the 2-simplex, while the orange cycles are not boundaries of any 2-chains. Thus the only element of B_1 is the pink cycle.

We have now connected ideas from topology and group theory. This forms the bulk of the theory required to understand how we can determine the shape of an object. Finally, we are ready to discuss homology.

2.3. **Homology.** The homology of an object is a measure of shape: it accounts for connected components, the loops or holes, the 3-dimensional cavities, and in general higher dimensional voids.

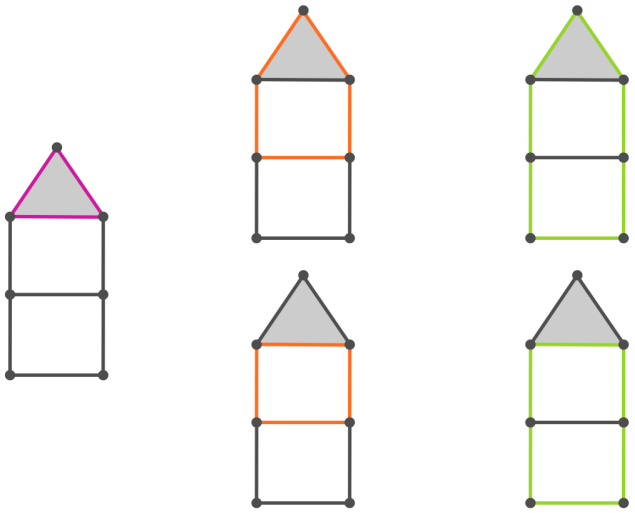
These features can be found by calculating the homology groups of a simplicial complex. Notice that B_p from Definition 2.19 is a subgroup of Z_p from Definition 2.18. This means that we can take the quotient, as in Definition 2.6.

Definition 2.20. The quotient group $H_p = Z_p/B_p$ is called the p^{th} *homology group*. The p^{th} *Betti number* β_p is the rank of H_p.

The Betti numbers are our end product when calculating homology. They tell us what we seek: the number of "features" in a particular dimension p. The 0^{th} Betti number β_0 tells us the number of connected components; β_1, the loops; β_2, the 3-dimensional cavities, and so on.

Example 2.21. Consider the following simplicial complex. We will investigate β_1, which should tell us that there are two holes.

The five 1-cycles, which make up Z_1, are highlighted. The pink cycle is the only 1-boundary-cycle and thus is the only element of B_1. The three resulting cosets of B_1 are the pink, orange, and green classes. (Imagine taking the set difference of the pink and one orange cycle: you get the other orange cycle. Likewise for green.)

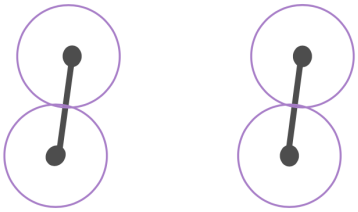


H_1 is thus a group with three elements: the pink, orange, and green classes. The rank of H_1 is two: the subset containing the orange and green classes generates the entire group. Indeed, there are two holes.

2.4. **Topological Data Analysis.** Topological Data Analysis (TDA) is a relatively new branch of mathematics, seeing the intersection of work between topologists, computer scientists, and data scientists. The primary tool in TDA is *persistent homology*, a specific kind of homology. The essence of persistent homology is to build a sequence of growing simplicial complexes from a point cloud (generally plotted in some high-dimensional space), namely, a Vietoris-Rips complex, and discern how the topological features of the complex change at different points in the sequence by examining its homology. Persistent homology tracks the number of connected components and holes that occur throughout the sequence. The process of this computation is best explained via example.

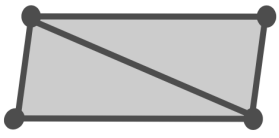
Example 2.22. Suppose we have the following point cloud:

We will create a Vietoris-Rips complex from this point cloud and explain how to calculate its persistent homology. We can build a simplicial complex starting with these four points as our 0-simplices. Around each of these 0-simplices, we create a ball of continuously increasing radius. Note that since this particular point cloud can be plotted in 2-D, these balls are simply disks; however, in higher dimensions they are high-dimensional n-spheres. As soon as two balls have grown large enough to intersect, the corresponding 0-simplices are connected by a 1-simplex, like so:



These balls will continue to grow, and eventually our data points will connect horizontally. This will form a hole in the simplical complex.

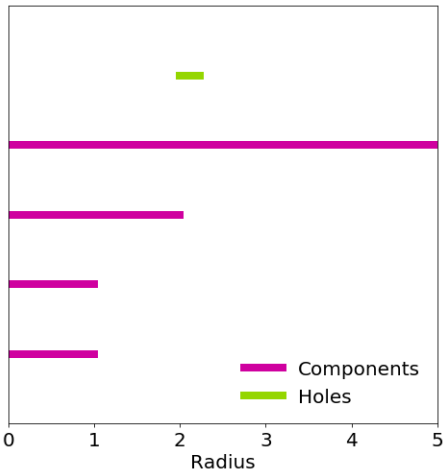
If three 0-simplices are all connected to form a triangle, the hollow triangle is automatically filled in to form a 2-simplex. In this example, our first 2-simplices will form with the addition of the shorter diagonal:



Similarly, $(n+1)$-simplices are automatically filled in when enough n-simplices connect.

We have now created a sequence of simplicial complexes, each one a subset of the next. Now we can use persistent homology to track this sequence of complexes. We look at "snapshots" in the sequence of complexes and determine the 0- and 1-dimensional homology groups. We record the number of connected components and loops at a given radius, and take note when these features are formed ("born") and when they close up ("die").

This series of topological transformations can be clearly recorded on a *barcode*, a modeling tool showing the birth and death of topological features. Although you can track higher-dimensional topological features using persistent homology, we will only focus on 0-dimensional (number of components) and 1-dimensional (number of holes) barcodes. The barcode for this example is shown below. Note that one pink bar continues indefinitely, as we are left with one connected component that will never die.

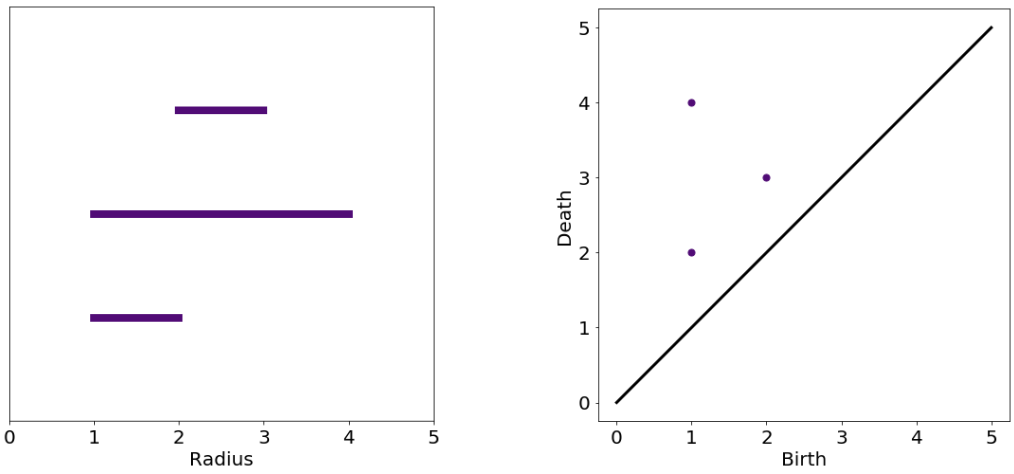


Once we have calculated the persistent homology of a given data set, we want to be able to compare the results to other data sets. This is done by comparing their *persistence diagrams*, which is another graphical representation of the information contained in a barcode.

A persistence diagram plots each bar from the barcode as a point. The x-coordinate is the radius at which the bar (feature) is born, and the y-coordinate is the radius at which it dies.

Example 2.23. The following barcode and persistence diagram represent the same data set. Each bar corresponds to a point. As a bar must be born before it can die, note that the x-coordinate will always be smaller than the y-coordinate. Thus, all points will lie above the diagonal line $y = x$.

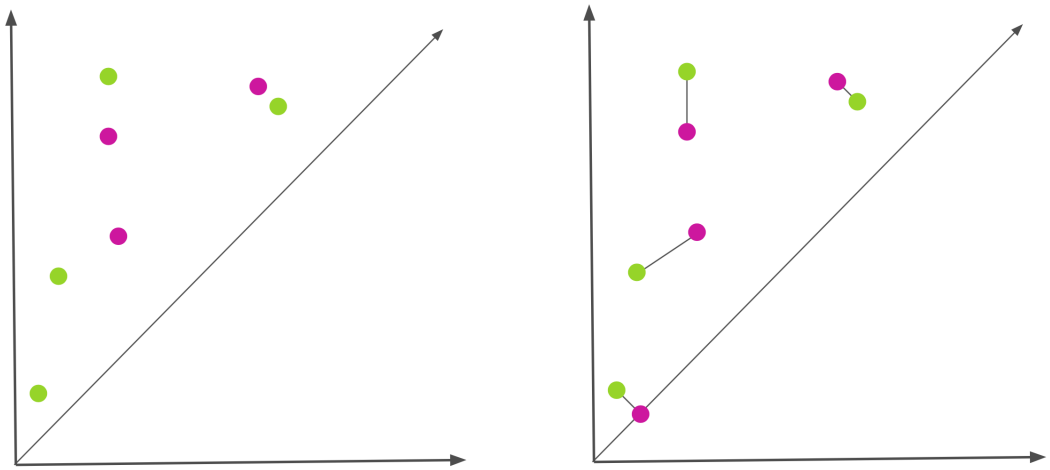
Qualitatively, a shorter bar will have more similar x and y coordinates and will thus lie closer to the diagonal. A more persistent feature – in other words, a longer bar – will lie farther from the diagonal.



Once we have two persistence diagrams, we can measure the "distance" between them, which helps us compare the two data sets.

Example 2.24. Consider the two persistence diagrams, the pink and the green, overlaid on the left plot below. We wish to know how similar these two diagrams are. This is determined by pairing points in the green diagram with points in the pink diagram, as shown in the plot on the right. Points are added to the diagonal as needed so that every point has a partner. These diagonal points aren't actual topological features; they would correspond to bars that are born and die simultaneously.

There are many possible pairings between the two diagrams. We choose the one that minimizes the total length of all the line segments between each pairing.



Once the optimal matching between the two diagrams has been determined, we compute two values. Suppose we have m pairs, and let l_i represent the length of the i^{th} line segment. We define the *Wasserstein distance* as

$$WD = \sum_{i=1}^{m} l_i$$

This simply sums up the total length of all of the line segments. We define the *bottleneck distance* as

$$BD = \max\{l_i\}_{i=1}^{m}$$

The bottleneck distance just measures the largest line segment. Qualitatively, the Wasserstein distance takes into account more detailed information about the two diagrams because it uses every single pairing. However, it is more sensitive to noise as a large number of unimportant features close to the diagonal can significantly increase this distance. Bottleneck, on the other hand, is less sensitive to noise but only takes into account the global structure of the diagram. For example, it might tell us the distance between the biggest feature from each diagram.

Now that we have described the basics of TDA, we can discuss the application chosen for this project: Natural Language Processing. Later in this report we will explicitly connect NLP to TDA, but first, we present a brief introduction to the key ideas in Natural Language Processing.

2.5. **Natural Language Processing.** Natural Language Processing (NLP) is the programming of computers to organize and analyze large amounts of data from ordinary language. The process itself can be found in such features as spell check, auto-complete, and programs like Siri and Alexa. The foundation of using NLP is converting a string of words into a vector.

A series of *terms*, or words in a text, makes up a *document*, a unit of text such as a sentence, paragraph, or chapter. The original text is split into documents, creating a *corpus*, which is a collection of Natural Language Text documents constructed with a specific purpose. From the corpus, we construct a *Document Term Matrix* (DTM) which is an $m \times n$ matrix where m is the number of documents in a corpus and n is the number of terms. Each position within the DTM counts the number of times a given term appears in a given document.

Example 2.25. Consider the following example converting a series of documents (a corpus) into its DTM.

Document 1: "There she was."
Document 2: "There he was not."
Document 3: "There he was, there she was not."

This corpus can be represented by the following 3×5 DTM:

$$DTM = \begin{array}{c} \\ \text{Doc. 1} \\ \text{Doc. 2} \\ \text{Doc. 3} \end{array} \begin{array}{c} \begin{array}{ccccc} \text{There} & \text{She} & \text{He} & \text{Was} & \text{Not} \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 2 & 1 & 1 & 2 & 1 \end{pmatrix} \end{array}$$

Now that we have defined and expanded upon some of the standard NLP conventions, we can discuss the applications of TDA to NLP.

2.6. **NLP with TDA.** In order to combine TDA with NLP techniques, we must understand how to begin the process in Example 2.22. In other words, where do these data points come from, and what do they mean? With a DTM, each row becomes a data point mapped onto a n-dimensional space for n terms in the corpus. The documents from the corpus make up the 0-simplices, or vertices. A 1-simplex between two vertices indicates a connection between two documents, based on similar word usage. If a hole emerges, this indicates that there is a loop in the text. This could mean the documents return to similar word use throughout the text, but not necessarily in every document. The goal of using TDA with NLP is to search for these loops.

For persistent homology, we want to know when the balls surrounding our data points will intersect, like in Example 2.22. In order to do this, we must know the distance between each pair of data points. For this purpose, a *distance matrix* is built.

A distance matrix, D, is an $m \times m$ matrix representing the pairwise distance between all rows in the DTM. Note that the diagonal should be zero, considering the distance of a vector from itself is zero. Likewise, the values should be symmetric across the diagonal because the distance from row 1 to row 2 is the same as the distance from row 2 to row 1.

$$D = \begin{bmatrix} D_{11} & D_{12} & D_{13} & \cdots & D_{1m} \\ D_{21} & D_{22} & D_{23} & \cdots & D_{2m} \\ D_{31} & D_{32} & D_{33} & \cdots & D_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{m1} & D_{m2} & D_{m3} & \cdots & D_{mm} \end{bmatrix} = \begin{bmatrix} 0 & D_{12} & D_{13} & \cdots & D_{1m} \\ D_{21} & 0 & D_{23} & \cdots & D_{2m} \\ D_{31} & D_{32} & 0 & \cdots & D_{3m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{m1} & D_{m2} & D_{m3} & \cdots & 0 \end{bmatrix}$$

There are many methods for calculating the distance between points in n-space; the following demonstrate three possible techniques for calculating D_{ij}, the distance between row i, $\mathbf{x}_i$, and row j, $\mathbf{x}_j$.

2.6.1. *Euclidean.* Euclidean distance is the straight line distance between two points. We can find the values of D through the following equation:

$$D_{ij} = \sqrt{(\mathbf{x}_i - \mathbf{x}_j) \cdot (\mathbf{x}_i - \mathbf{x}_j)}$$

2.6.2. *Taxicab.* Taxicab distance is the sum of the absolute value of the difference between two Cartesian coordinates. With this distance type, distance is measured along the gridlines of the space the points reside. Now we find the values of matrix D for a DTM with n columns as follows:

$$D_{ij} = \sum_{k=1}^{n} |x_{i_k} - x_{j_k}|$$

2.6.3. *Angular.* Angular distance measures the angle between two vectors. To do this, we create a vector between the origin and the point representing each document. We can then use the following formula to calculate the angle between the i^{th} and j^{th} row:

$$D_{ij} = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{||\mathbf{x}_i||\,||\mathbf{x}_j||}$$

We will now demonstrate the result of each distance type performed on our DTM from Example 2.25.

Example 2.26. From Example 2.25, the three types of pairwise distances between the three rows gives us:

$$D_{\text{Euclidean}} = \begin{bmatrix} 0 & 1.732 & 2 \\ 1.732 & 0 & 1.732 \\ 2 & 1.732 & 0 \end{bmatrix}$$

$$D_{\text{Taxicab}} = \begin{bmatrix} 0 & 3 & 4 \\ 3 & 0 & 3 \\ 4 & 3 & 0 \end{bmatrix}$$

$$D_{\text{Angular}} = \begin{bmatrix} 0 & 0.955 & 0.515 \\ 0.955 & 0 & 0.441 \\ 0.515 & 0.441 & 0 \end{bmatrix}$$

2.7. **Optional Processes.** In addition to distance calculations, there are many NLP techniques to process textual data. Many of these techniques are specific to language usage, such as considering the passage of time, the quantity of specific words being used, and so on. In the coming sections these methods will be discussed.

2.7.1. *Term Frequency-Inverse Document Frequency (TFIDF).* In the optional process of TF-IDF, which occurs after the DTM is created, each of the positions in the matrix has a weight applied to it. The TF-IDF weight is two calculations multiplied together. It is applied to each position in the matrix and replaces the original value of the DTM. This process is used to limit the influence of words which may connect documents despite containing little of the document's meaning.

Definition 2.27. The *Inverse Document Frequency* determines the term's importance to the entire text. This statistical interpretation originated from Karen Spärck Jones in 1972 [5].

$$IDF = \log_{\text{e}} \frac{Total\ Number\ of\ Documents}{Number\ of\ Documents\ with\ Term\ t\ in\ it}$$

There are three types of Term Frequency we considered: Standard Term Frequency (STF), Log Normalization (LN), and Augmented Normalization (AN) [6].

Definition 2.28. The *Term Frequency* measures the term's importance to a given document.

$$STF = \frac{Number\ of\ Times\ Term\ t\ Appears\ in\ a\ Document}{Total\ Number\ of\ Terms\ in\ the\ Document}$$

$$LN = \log_{10}(Term\ t's\ Frequency\ in\ a\ Document\ + 1)$$

$$AN = \frac{Term\ t's\ Frequency\ in\ a\ Document}{Frequency\ of\ Most\ Often\ Occurring\ Term\ in\ the\ Document}$$

This process is used to limit the influence of words which may connect documents despite containing little of the document's meaning.

2.7.2. *Similarity Filtration with Time Skeleton (SIFTS).* SIFTS is an optional modification to the distance matrix. This process arbitrarily connects each point to the one immediately preceding and following it. This takes into account the order of the text when searching for loops. This changes the distance matrix by converting the super-diagonal and sub-diagonal to zeros.

Example 2.29. If we apply SIFTS to our distance matrices from our Example 2.26, we get the following:

$$D_{\text{Euclidean, SIFTS}} = \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

$$D_{\text{Taxicab, SIFTS}} = \begin{bmatrix} 0 & 0 & 4 \\ 0 & 0 & 0 \\ 4 & 0 & 0 \end{bmatrix}$$

$$D_{\text{Angular, SIFTS}} = \begin{bmatrix} 0 & 0 & 0.515 \\ 0 & 0 & 0 \\ 0.515 & 0 & 0 \end{bmatrix}$$

These ideas provide the foundation for where we began our specific project. Our contributions will be described in the following section.

3. Methods

We now describe our methods for choosing a data set and using NLP with TDA to search for loops in the text. We will describe the existing R packages we used, our own creations, and the decisions we made to arrive at our results.

3.1. **Data set.** We used the State of the Union Addresses from the years 1901 to 2019. We also included the speeches of George Washington and John Adams. For consistency, we only included addresses that were delivered as speeches and not as written messages. These files, available at [7], were converted to .txt files manually before being input into RStudio using functions from the `utils` package [8]. It is worth mentioning that this method would likely be burdensome for larger data sets, so an automated method would be needed.

3.2. **Text Functions.** In our code we created our own functions, which combined many existing functions in RStudio and allowed us to make choices about which NLP techniques we would like to focus on. This section will discuss the NLP preparations prior to applying TDA.

3.2.1. *Delimiters.* One of the inputs of this function is the user's choice of how to delimit, or break up, the original text either by sentence or paragraph. The function `read.delim()` separates sentences by recognizing a period and paragraphs by a new line. These create column vectors, where each row is a document delimited by our choice. The sentence method has issues differentiating between end of sentence periods and those used in other instances, such as decimal points or in abbreviations such as "Mr." or "Dr." The State of the Union Addresses are naturally split into paragraphs, so this method was selected.

3.2.2. *Some NLP techniques.* Some techniques were always performed on our files. These functions come from the `textclean` package [9]. These functions replaced common contractions, ordinals, and certain symbols with their long forms. We also added a function which would add an extra space after commas, without which occasionally caused comma-separated words to merge to one in later processing.

3.2.3. *Creating a Corpus.* To convert our text matrix to a corpus we use functions from the `tm` package [10]. Once the corpus is created, there are several more NLP techniques to be performed. In every case, the text was entirely converted to lowercase, since the word "An" and "an" are just the same word with different formatting. At this point, the code removes punctuation and extra spaces. Another option we had, but did not use, was to remove numeric characters. We also added the option to remove *stop words* from the corpus before moving forward. Stop words are words which appear frequently in a language which carry little meaning compared to those around them. Examples include words such as to, an, for, of, the, am. The `tm` package [10] provides a vector containing common stopwords in the english language . Different sources would consider alternate lists, which is up to the user to determine. For our data set, we do not to remove stop words as we feel TF-IDF adequately addresses commonly used words.

3.2.4. *Document Term Matrix.* From the corpus, the `DocumentTermMatrix()` function in the `tm` package [10] can create a DTM . One input is the minimum or maximum word length to be included in the DTM; for our data set, we included all words, regardless of length. The function creates a dataframe, which then must be converted to a matrix using the `as.matrix()` function.

3.2.5. *TF-IDF.* We chose to apply TF-IDF with the first method from Definition 2.28. The function we created is capable of performing all three options. We chose to apply TF-IDF instead of removing stop words because the TF-IDF weighting is more dynamic. The weighting can take into account words that would not traditionally be considered stop words, but distort the connections.

3.3. **Distance Matrices.** The function we created can calculate any of the three distance types discussed in Section 2.6, and we added the option to apply SIFTS to the distance matrix. We chose to use angular distance for our data set because it is conventionally used in NLP [4].

3.4. **Creating and Comparing Persistence Diagrams.** The final steps are to use the distance matrix to find the barcodes of the features. The function `calculate_homology()` in the `TDAstats` package [11] takes the distance matrix and produces the birth and death radii of each bar. We save these values to a CSV file and use the `plot_barcode()` function to visualize the barcodes. We then can use the the `wasserstein()` and `bottleneck()` functions in the `TDA` package [12] to compare the persistent homology of two texts with Wasserstein and bottleneck distance. We can look at these distances to determine the similarity of two texts.

4. Results

In this section we will discuss the speeches we analyzed to try to determine whether TDA effectively differentiates between presidents' speeches. We will examine the barcodes and persistence diagrams of Richard Nixon's addresses, from 1970-1974, and Gerald Ford's, from 1975-1977.

4.1. **Investigating individual speeches.** Our first step is to look at an individual speech to determine what topological features we can discern. The code discussed above outputs a barcode which represents the persistence of each feature from a text. When a hole emerges, a bar representing the loop is born, then dies when all points are connected to each other. Recall that holes can be thought of as "circling back" to previous ideas in the text.

Figure 5 shows the output from Nixon's 1971 address. The dimension 1 features, or holes, are colored in blue and dimension 0, or individual components, in red. We see that holes begin appearing only as many of the individual components connect, around a radius of 1.4. Note that this is close to the maximal angular separation of any two of our vectors, $\pi/2$. Looking at the persistence diagram, we see that many of the holes are near the 45 degree line. Being close to the line means that a component's birth radius is very close to its death radius, so the feature is less persistent. Holes often appeared in this position for many of the speeches we looked at. This indicates that many of the holes from our texts do not persist, and thus may not represent meaningful looping patterns in the text.

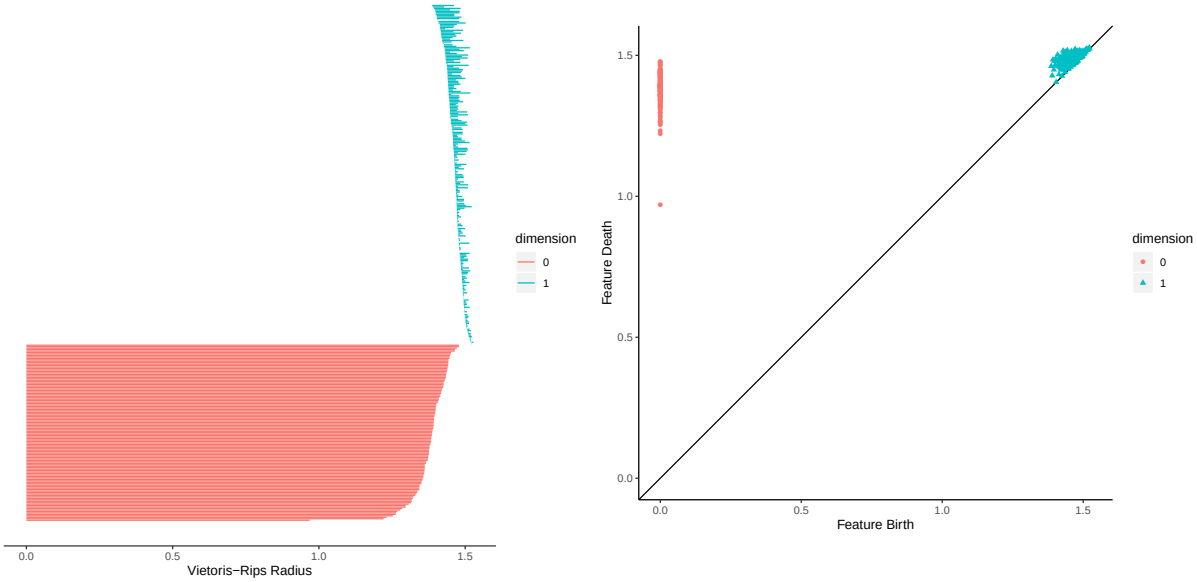
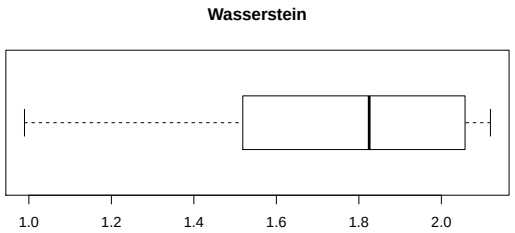


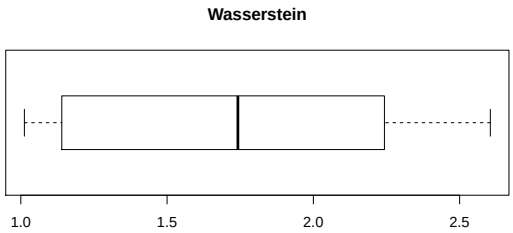
FIGURE 5. The barcode and persistence diagram from Nixon's 1971 address. Both dimension 0 and dimension 1 components are shown. Because the dimension 1 points lie close to the diagonal on the persistence diagram, they may not represent significant topological features.

4.2. **Comparing Pairs of Speeches.** Our next step is to compare between speeches to see how similar they are. In this section we look at all of Richard Nixon's speeches compared to each other, contrasted with all of Nixon's compared to all of Gerald Ford's. We use the Wasserstein and bottleneck distances of the dimension 1 components to determine the similarity of two speeches. Figure 6 shows the range of Wasserstein and bottleneck distances for our two types of comparisons. As might be expected, the distances between Nixon's speeches were smaller than those between Nixon's and Ford's. A possible interpretation is that Nixon's addresses were more similar to each other than to those of Ford.

We also chose to apply the SIFTS method from Section 2.7.2 to see if accounting for the flow of the document would affect the distances. Shown in Figure 7, the Wasserstein and bottleneck distances increased for both types of comparison. This increase shows that as we account for the order of the documents in the text, they become more dissimilar. Again, Nixon's speeches are "closer" to each other than they are to Ford's, possibly indicating more topological similarity.

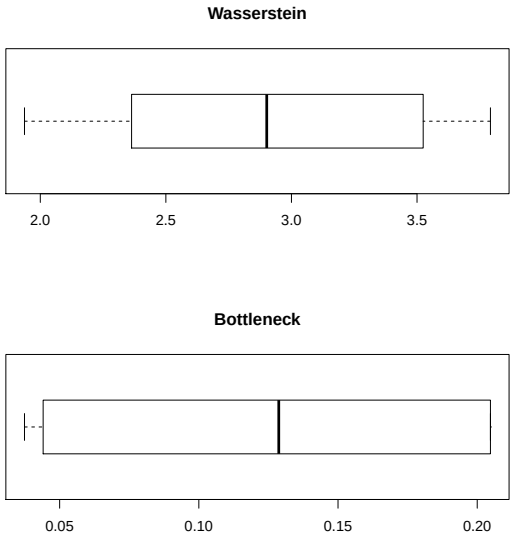


(A) The pairwise distances between Nixon's speeches.

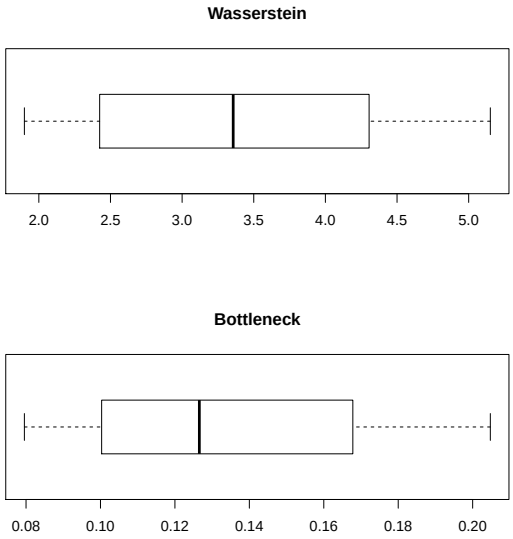


(B) The pairwise distances between Nixon's speeches and Ford's speeches.

FIGURE 6. The Wasserstein and bottleneck distances between the dimension 1 components of the speeches. On the left, Nixon's speeches have been compared to each other. On the right, Nixon's speeches have been compared to Ford's. A preliminary interpretation is that Nixon's speeches are more similar to each other than they are to Ford's.



(A) The pairwise distances between Nixon's speeches.



(B) The pairwise distances between Nixon's speeches and Ford's speeches.

FIGURE 7. The same as in Figure 6, but with SIFTS applied. Again, a possible interpretation is that Nixon's speeches are more similar to each other than they are to Ford's.

4.3. **Investigating the Influence of Noise.** We want to determine the significance of these results. Does the smaller median distance within Nixon's speeches really mean that they are more similar to each other than they are to Ford's? Or is our data too noisy to make such definitive conclusions?

We developed a method to investigate the influence of noise. We took each speech and scrambled its words, only retaining the number and size of each paragraph. This should theoretically destroy any meaningful topological features of the speech. We then calculated the Wasserstein and bottleneck distances between each scrambled speech and the real speeches for that particular president. Then, for each president, we compared each of their real speeches against their other real speeches. The results of all of these pairwise comparisons are shown in Figure 8. As you can see, these distributions are similar. Our interpretation is that the scrambled speeches are roughly as similar to the real speeches as the real speeches are to each other. At this point, we cannot definitively say that the topological features we measure meaningfully capture the structure of the speeches.

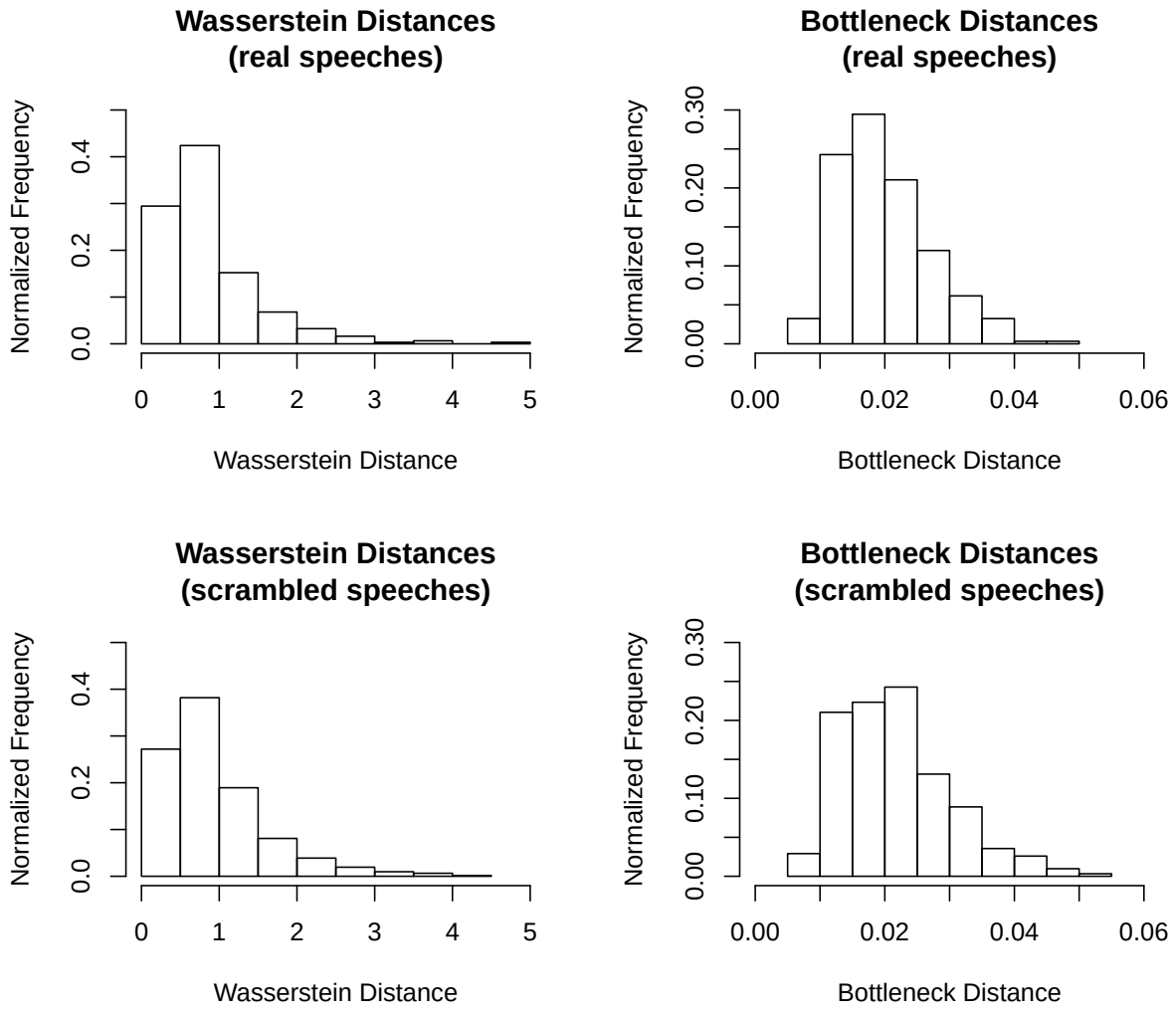


FIGURE 8. The distributions of Wasserstein and bottleneck distances for real speeches compared to real speeches and scrambled speeches compared to real speeches. The distances shift slightly upward for the scrambled speeches, but overall are quite similar to the real speeches. We interpret this as evidence that the topological features we measure do not necessarily give a meaningful representation of the structure of the speeches.

5. Conclusion

Over the course of this project we were able to successfully implement Topological Data Analysis to the State of the Union addresses by using the many Natural Language Processing techniques mentioned above. Through a mixture of previously existing packages in R and our own contributions, we were able to efficiently run texts though our code and output a variety of results such as barcodes and persistence diagrams, as well as compare these via Wasserstein and bottleneck distance calculations. The results we attained remain open to interpretation due to the likelihood of noise impacting our data. This issue may be specific to our data set. It is possible that other text documents could produce meaningful results. There is much potential for future work when we look at applying TDA to NLP, as it is a relatively recent application in the field of applied mathmatics.

6. Future Work

There are a few follow-up inquiries we can pursue with this project. As noise seems to be a relevant factor for the State of the Union addresses, we want to develop a precise statistical method to determine the significance of our topological features. We could create a set of sampling distributions for hypothesis testing. This may involve the scrambled speeches from our work, or a new approach to account for the possibility of noise. One problem that may arise when using traditional statistical methods on our data set is that the State of the Union addresses are not independent since multiple are written by the same president. And, the sample size of speeches is relatively small since written messages were the preferred form of communication for many years. Another option would be to explore work that has been done to use statistics combined with TDA, using a method involving persistence landscapes.

If we establish a robust statistical method to determine the importance of our topological features, we can then expand our analysis to different data sets. This could help clarify the role of TDA applied to NLP in general.

References

[1] Nina Otter, Mason A. Porter, Ulrike Tillmann, Peter Grindrod, and Heather A. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science*, 6(1):17, 2017.

[2] Xiaojin Zhu. Persistent homology: An introduction and a new text representation for natural language processing. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI '13, pages 1953–1959. AAAI Press, 2013.

[3] Shafie Gholizadeh, Armin Seyeditabari, and Wlodek Zadrozny. Topological signature of 19th century novelists: Persistence homology in context-free text mining. 09 2018.

[4] Hubert Wagner, Pawel Dlotko, and Marian Mrozek. Computational topology in text mining. In *CTIC*, 2012.

[5] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[6] Christopher Manning. *Introduction to information retrieval*. Cambridge University Press, New York, 2008.

[7] The american presidency project. `https://www.presidency.ucsb.edu/documents/presidential-documents-archive-guidebook/annual-messages-congress-the-state-the-union`.

[8] R Core Team. *utils-package: The R Utils Package*. R package version 2.6.1.

[9] Tim Rinker. *textclean: Text Cleaning Tools*, 2018. R package version 0.9.3.

[10] Ingo Feinerer, Kurt Hornig, and Inc Artifex Software. *tm: Text Mining Package*, 2018. R package version 0.7-6.

[11] Raoul R. Wadhwa, Drew F.K. Williamson, Andrew Dhawan, and Jacob G. Scott. TDAstats: R pipeline for computing persistent homology in topological data analysis. *Journal of Open Source Software*, 3(28):860, 2018.

[12] Brittany Fasy, Jisu Kim, Fabrizio Lecci, Clement Maria, David Millman, and Vincent Rouvreau. *TDA: Statistical Tools for Topological Data Analysis*, 2019. R package version 1.6.5.