Encoding Data from libraries.io for Topological Data Analytics

Jacob Miller, Brian Friend, Jonathan Anderson, and Kaixiang Wang

***Abstract*— Persistent homology and other forms of topological data analytics are still being developed at a research level, but their use in applications has still been limited. Part of this limitation is from the inability to encode data sets in meaningful ways to apply the techniques. This paper will examine data sets with a common theme, and will compare potential encodings.**

I. INTRODUCTION

Over the past several decades, new data analysis techniques have been developed using topological theory. By assuming that a data set comes from a distinct topological space, one can describe this data set in terms of a topological invariant. There are several popular invariants, such as a space's homopty, that are not always computationally feasible cite. For this reason, there have been many efforts to develop polynomial runtime algorithms for calculating different invariants.

A popular algorithm is the calculation of a simplicial complex's homology. The problem with this algorithm alone is that the data used to encode a simplicial complex may be incomplete or erroneous. For this reason, there are techniques used to examine slightly varying encodings of the data to see if the homology *persists* throughout. Appropriately named, these calculations are referred to as persistent homology.

Using the same invariant for the topological space of different data sets, one may be able to determine if the topological spaces are distinct. Such a distinction can be used as a way to classify differences in data sets. On the other hand, if two topological spaces share the same value for an invariant, more information would be necessary to determine whether or not they are identical. Since many applications also assume that different data sets come from distinct topological spaces, it is useful to encodings of the data that lead to distinct values for topological invariants. In the case of persistent homology, there have been clear cases where the type of filtrations reveal distinctions in resulting homologySamir's paper. For this study, we will maintain the same type of filtration between data sets, but instead look at how data is initially encoded at the start of the filtration process to see if there is a noticeable affect on persistent homology.

Before going further, some background will be given on the terms used throughout this proposal. In addition to defining the terms, an explanation of their interpretation will be provided.

*This work was not supported by any organization

II. BACKGROUND

A. Simplicial Complexes

Homology is often thought of as the "holes" present in a space. On the real number line, $\mathbb{R}$, a hole might be thought of as a missing point; in $\mathbb{R}^2$ it might be thought of as a circular gap; and in $\mathbb{R}^3$ it may be thought of as a three dimensional void in space. Through the theory of singular homology, these notions are generalized to arbitrarily large dimensions and for abstract spaces. Despite being defined on any space, it is only computable on discreet spaces. As we will see, simplicial complexes are discreet spaces which have easily calculated homology.

Definition 1. *An* n**-simplex**, Δ^n, *is the simplest geometric figure determined by a collection of* $n+1$ *points in the Euclidean space* $\mathbb{R}^n$. *That is, it is a complete graph of* $n+1$ *vertices in* n *dimensions.*

Definition 2. *An* n**-face** *of a simplex is a subset of that simplex's vertex set made of* $n+1$ *vertices.*

Definition 3. *A* **simplicial complex** $\mathcal{K}$ *is a finite set of simplices satisfying the following conditions:*

- *For each simplex* $A \in \mathcal{K}$, *every face of* A *is also a simplex of* $\mathcal{K}$; *and*
- *for* $A, B \in \mathcal{K}$, *either* $A \cap B$ *is empty or it is a shared face.*

So a simplicial complex is a space with nice features defined on a finite subset of the power set of a finite set of points. Because empirical data sets are finite, it is natural to use this set as the foundation for a simplicial complex. These points are often referred to as the *vertices* of the complex.

B. Simplicial Homology

Before being able to calculate simplicial homology, there must be an orientation on the vertices of a simplex. This can be thought of as an ordering of the vertices such that an equivalence relation is defined for all orderings that differ by an even permutation.

Definition 4. *A* **simplicial** k**-chain** *is given by a sum*

$$\sum_{i=0}^{k} c_i \sigma_i$$

where each c_i *is an integer and* σ_i *is an oriented* k*-simplex.*

Definition 5. *The* **gorup of** k**-chains** *on a simplicial complex, written* C_k, *is a free abelian group with a basis in one-to-one correspondence with the set of* k*-simplices in the complex.*

Definition 6. *The* **boundary operator** *is a function* $\delta_k : C_k \to C_{k-1}$ *given such that*

include actual homology def

C. *Filtrations*

Include more info on filterations
finish background

III. Goals

There are several desired outcomes expected from the completion of this study. Primarily, it is designed to expose the investigators to software libraries for using persistent homology, and to provide more comfort in using this form of data analytics.

The larger objective is to see if there is any underlying difference to the structure of software package managers. There will be obvious aspects of the data which will not agree, such as name and author, but these are not reasonable ways to track the philosophy or coding style of package managers. By philosophy, it is meant that some software is written in very fine, short code snippets while others are large projects with many hundred of code lines. Some software is meant to be self contained while others rely heavily on previously written software. Rather than asking every software author their philosophy and determining how closely their code matches, it is more practical to examine large collections of software written among a complete package manager.

By using topological data analytics on these large data sets, the study will determine if there is an appropriate way to encode data such that the differences in package managers and structural philosophy can be quantified. This will be especially interesting when comparing the results to the natural graph structure created from package dependencies. This information is included in the original data, but it is unknown how to best represent these dimensions so that our tools will be able to pick up on them.

IV. Methodology

There are many software libraries that exist for calculating persistent homology such as Perseus, Ripser, JavaPlex, and Dionysussite all. This project will use the R package called *TDA: Statistical Tools for Topological Data Analysis*. Besides introducing some original functionality, it has created an R interface for Dionysus and several other popular libraries.

The data itself will be very diverse information on every package available within each package manager. Much of the data will need to be polished so that unnecessary information can be removed. This will include variables that may suggest differences in data irrelevant to structure. Some of the obvious information that will be removed is package author and specific creation date.

It is important that the method for constructing simplicial homologies and the type of filtration act as the control variables for this experiment. First, a convenient filtration will need to be set up for the size of the data sets. Experiments will be carried out on how the data is encoded. Different dimensions of the original data will be included/excluded, and several values may be artificially calculated before being included as dimensions to the data. An example of a possible calculated dimension would be the out-degree from a package's dependency tree. It would be interesting if this value alone was enough to quantify the topological differences between data sets. So the study will test both the types of dimensions and the quantity of dimensions used for calculating persistence barcodes.

Upon calculating persistence barcodes from each data set, the bottleneck and/or Wasserstein distances may be calculated between them. If these distances are sufficient, then it may suggest that the encoding is appropriate for using the tool. Should some encodings induce greater pair-wise distances, we may examine the cause and determine which is the most appropriate for our data.

A. *Timetable*

Acknowledgment

Special thanks to my former mentor Facundo Mémoli, and to Audris Mockus for his patience in the preparation of materials.

include bib file from zotero once all sources collected