Ripser.py: A Lean Persistent Homology Library for Python

Christopher Tralie¹, Nathaniel Saul², and Rann Bar-On¹

¹ Department of Mathematics, Duke University ² Department of Mathematics and Statistics, Washington State University

Summary

Topological data analysis (TDA) (Edelsbrunner & Harer, 2010), (Carlsson, 2009) is a field focused on understanding the shape and structure of data by computing topological descriptors that summarize features as connected components, loops, and voids. TDA has found wide applications across nonlinear time series analysis (Perea & Harer, 2015), computer vision (Perea & Carlsson, 2014), computational neuroscience (Giusti, Pastalkova, Curto, & Itskov, 2015), (Bendich, Marron, Miller, Pieloch, & Skwerer, 2016), computational biology (Iyer-Pascuzzi et al., 2010), (Wu et al., 2017), and materials science (Kramar, Goullet, Kondic, & Mishaikow, 2013), to name a few of the many areas of impact in recent years.

Persistent homology (Edelsbrunner & Harer, 2010) is the main workhorse of TDA, and it computes a data structure known as the persistence diagram to summarize the space of stable topological features. The most commonly used scheme for generating persistence diagrams is the Vietoris Rips filtration (VR) since it is easily defined for any point cloud. In its naive implementation, VR is prohibitively slow, but recently a C++ library known as Ripser (Bauer, 2017) has been devised to aggregate all known computational speedups of the VR filtration into one concise implementation. Because of the unprecedented speed of Ripser, it has created a large user base for both research and applications. However, the library as it stands is only a command line tool and as a result, multiple efforts have been made to wrap the C++ library for use in other languages, often via clunky system calls to the command line.

In this work, we develop an intuitive interface for VR filtrations with Ripser at its core via Cython. We have gone through extensive testing via continuous integration frameworks to ensure it works across all platforms and as a result, Ripser.py is currently as easy to setup as `pip install ripser`. We see this package as particularly useful for mathematicians with little programming experience who would like to use TDA as an entry point into data science, or conversely for researchers with little understanding of Algebraic Topology who would like to apply TDA to their problem domain. To aid this, we have created a large set of Jupyter notebooks to showcase some of the many applications that are possible with this library.

Library Details

Ripser.py supplies two interfaces: one lightweight and functional interface, as well as an object-oriented interface designed to fit within the Scikit-Learn transformer paradigm (Pedregosa et al., 2011). We have merged together multiple branches of the original Ripser library (“sparse-distance-matrix,” “representative-cocycles”) to expose some lesser
known but incredibly useful features hidden in Ripser. Below we detail some of the special features made easy with our library.

Sparse filtrations

Ripser.py can accommodate sparse distance matrices using the `scipy.sparse` library. In an accompanying notebook, we demonstrate how Ripser.py can be used in conjunction with “sparse filtration” approximation algorithms (Cavanna, Jahanseir, & Sheehy, 2015) to add further computational speedups to Ripser. Additionally, sparse matrices allow us to easily define a “sublevelset filtration,” or the “watershed algorithm,” for quantifying critical points in grid data such as time series and images, and Ripser.py includes a helper function to make this easy for image data. One of the notebooks demonstrates how this can be used to identify cells in an image, for instance.

Coefficient Fields

Ripser.py naturally allows the specification of arbitrary field coefficients. Most applications that use TDA use binary coefficients in the VR filtration, and as a result, most existing TDA software can only handle binary coefficients. Using other coefficient fields \( \mathbb{Z}/p\mathbb{Z} \) for any prime \( p \), makes it possible to detect “twists” in point cloud data. We have included an example notebook that shows how this can differentiate a point cloud sampled from the boundary of a Möbius strip from a point cloud sampled from an ordinary, untwisted loop.

Applications of this surprisingly appears in some important real world scenarios such as periodic time series analysis (Perea & Harer, 2015) and image analysis (Perea & Carlsson, 2014). This feature of Ripser.py has been used in a pipeline to synthesize slow motion videos (Christopher J. Tralie & Berger, 2018) and is currently being used to quantify periodicities in repetitive motions with children with autism spectrum disorder (Christopher J. Tralie, Matthew, & Sapiro, 2018).

Representative Cocycles

TDA is generally used to quantify topological features, but there has been some research on localizing topological features back in the point cloud. Ripser.py can return “representative cocycles” associated to different homology classes (topological features), for sparse and dense filtrations over any field. Use of this feature can be viewed as topological non-linear dimension reduction (NLDR). Examples of this can be seen in mapping the point cloud to a circle (Silva, Morozov, & Vejdemo-Johansson, 2011), which is useful for parameterizing periodic data, or in mapping the point cloud to the projective plane (Perea, 2018), which shows up in analysis of image patches.

Higher Order Homology

Because of long standing speed concerns with TDA, most applications have focused on 0-dimensional (connected components) and 1-dimensional (loops) homology features from VR filtrations. However, Ripser.py is fast enough to compute 2-dimensional homology (voids e.g. empty space in a basketball) for modest point clouds. This feature has been used to quantify quasiperiodic phenomena in videos of patients with vocal fold disorders (Christopher J. Tralie & Perea, 2018), and we anticipate more researchers will use the feature for other problems now that it is computationally accessible.
Source Code

The source code for Ripser.py is available on Github through the Scikit-TDA organization https://github.com/scikit-tda/Ripser.py. The original Ripser library can be found at https://github.com/Ripser/Ripser/

Acknowledgements

Christopher T ralie and Rann Bar-On were supported by an NSF big data grant DKA-1447491. Nathaniel Saul was partially supported by NSF DBI-1661348 and by Washington NASA Space Grant Consortium, NASA Grant #NNX15AJ98H. We thank Ulrich Bauer for the original Ripser library and for valuable feedback during development of Ripser.py. We also thank Jose Perea, William Guss, and Matija Čufar for helpful feedback and bug fixes. Finally, we thank the students of the “Topological Data Analysis and Persistent Homology” workshop in Levico, Italy for beta testing the code.

References


https://doi.org/10.21105/joss.00925


