TDAvec: Computing Vector Summaries of Persistence Diagrams for Topological Data Analysis in R and Python

2022-09-13

Summary

The theory of *persistent homology* is one of the popular tools in *topological data analysis* (TDA) to analyze data with underlying shape structure (Carlsson 2009; Edelsbrunner and Harer 2010; Chazal and Michel 2021). In this context, a single data observation could be a collection of points lying in a metric space, an image, a graph or a time series. The basic idea behind persistent homology is to build a nested sequence (or *filtration*) of *simplicial complexes* (indexed by a scale parameter) on top of data points and keep a record of the appearance and disappearance of various topological features at different scale values. Here, these topological features are “holes” of different dimensions – connected components, loops, voids, and their higher-dimensional versions whose emergence and subsequent disappearance are tracked using a concept of homology from algebraic topology. From a geometric point of view, simplicial complexes consist of vertices, edges, triangles, tetrahedra etc., glued together and serve as a means for recovering (at least partially) the underlying shape information which is lost during sampling (Nanda and Sazdanovic 2013).

A topological descriptor outputted by the persistent homology encoding the shape of data is called a *persistence diagram* (PD). Mathematically, a k-dimensional PD is a multi-set of points $D = \{(b_i, d_i)\}_{i=1}^{N}$, where each point (b_i, d_i) corresponds to a topological feature of homological dimension k (0 if a connected component, 1 if a loop, 2 if a void, etc) with the x-coordinate representing the scale at which this feature is born (or created), and the y-coordinate representing the scale at which it dies (or disappears). In practice, one is usually interested in applying a machine learning method to PDs to make further inferences from data. However, the fact that PDs do not form a Hilbert space, which is a feature (or an input) space for a wide class of machine learning methods, limits their direct of use in applications. To overcome this challenge, kernel methods and vectorization techniques are commonly used (Chung and Lawson 2022). The kernel approach involves defining a notion of similarity between pairs of PDs, whereas the vectorization methods aim to transform PDs into finite-dimensional feature vectors that can be used as input for many standard machine learning models. Such vector summaries of PDs are computed in two steps: first one constructs a respective summary function from a given PD and then vectorizes it using either one or two dimensional grid of scale values. In recent years, the kernel and vectorization approaches have proven successful and gained prominence in the applied TDA literature (see Hensel, Moor, and Rieck (2021) for a survey of applications of TDA in machine learning).

R tools for TDA

The computational tools for TDA in the `R` environment are provided through various packages[1] such as `TDA` (Fasy et al. 2021), `TDAstats` (R. R. Wadhwa et al. 2018), `kernelTDA` (Padellini and Palini 2020), `TDAmapper` (Pearson, Muellner, and Singh 2015), `TDAkit` (You and Yu 2021), `tdaunif` (Brunson, Demkowicz, and Choudhary 2024), `TDApplied` (Brown and Farivar 2024) and `ripserr` (R. Wadhwa, Piekenbrock, and Scott 2020) (see Table 1 for an overview of these packages in terms of their scope and areas of focus).

[1]In this overview, we only focus on `R` packages for TDA that are available on the CRAN repository.

	`TDA`	`TDApplied`	`TDAstats`	`TDAkit`	`kernelTDA`	`tdaunif`	`ripserr`	`TDAmapper`
Sampling methods	✓			✓		✓		
Density estimation	✓							
Alpha filtration	✓							
Alpha shape filtration	✓							
Vietoris-Rips filtration	✓		✓	✓			✓	
User-defined filtration	✓							
Cubical complex							✓	
Wasserstein distance	✓	✓	✓		✓			
Plotting persistence diagrams	✓	✓	✓	✓				
Statistical methods	✓	✓	✓	✓				
Vectorization methods	✓			✓	✓			
Kernel methods		✓		✓	✓			
Supervised learning methods		✓			✓			
Clustering methods	✓	✓		✓				
Visualization of high-dimensional data								✓
Dimension reduction		✓						

Table 1: `R` packages for TDA

The `TDA` package is the largest `R` package for TDA. The `TDA` package offers tools to compute PDs for commonly used types of filtrations such as *Vietoris-Rips*, *Alpha* and *Alpha shape*. It also allows to construct more general sublevel set filtrations and compute the corresponding PDs. Moreover, the `TDA` package provides implementations to plot PDs and compute *bottleneck and Wasserstein* distances between them. `TDAstats` offers a variety of tools for conducting statistical inference (such as hypothesis testing) on PDs. Compared to the `TDA` package, it computes PDs much faster for Vietoris-Rips filtrations based on the Ripser C++ library (Bauer 2021) and offers more aesthetic visualization of the diagrams using the `ggplot2` package (Wickham 2016). The `kernelTDA` package contains implementations of popular kernel-based methods for TDA such as *geodesic Gaussian kernel*, *geodesic Laplacian kernel*, *persistence Fisher kernel* and *persistence sliced Wasserstein kernel*. For computing the Wasserstein distance between a pair of PDs, unlike the `TDA` package, it uses an iterative procedure to reasonably approximate the exact distance which that leads to a considerable reduction in run-time cost. The `ripserr` package allows a fast computation of PDs for filtrations on Vietoris-Rips and cubical complexes using the Ripser C++ library. The `TDApplied` and `TDAkit` packages provides various tools to integrate topological features (PDs or their vector summaries) into machine and statistical learning settings. The `tdaunif` is a useful package if one needs to sample points from various manifolds such as a klein-bottle, an ellipse or a torus. `TDAmapper` offers tools to visualize high-dimensional data by constructing the so-called Mapper graphs that preserve its topological structure.

Python tools for TDA

Several `Python` libraries are available for TDA, such as `Giotto-tda` (Tauzin et al. 2021), `Ripser` (Christopher, Nathaniel, and Rann 2018), `Gudhi` (Rouvreau 2020), `Scikit-tda`, `Dionysus 2` (Morozov 2018), `Persim` (Saul 2019) and `KeplerMapper` (Van Veen et al. 2019). `Giotto-tda` is a powerful library that integrates with the popular machine learning library `scikit-learn`, offering tools for persistent homology and visualizations of persistence diagrams. `Ripser` focuses on fast computation of Vietoris-Rips complexes, especially for large datasets. `Gudhi` provides a wide range of topological tools for simplicial complexes, persistent homology, and topological signatures. `Scikit-tda` is another package that integrates with `scikit-learn`, simplifying the application of TDA to typical machine learning tasks. `Dionysus 2` offers fast computation of persistent homology and cohomology, with an emphasis on flexibility and efficiency. `Persim` focuses on tools for working with PDs. It contains implementations of commonly used vectorization and kernel methods for PDs. `KeplerMapper` implements the TDA Mapper algorithm to visualize high-dimensional data. For a more comprehensive list of `Python` libraries for TDA and their functionality, we refer the readers to ("Awesome-TDA" 2024).

Statement of need

The problem of transforming PDs into finite dimensional vectors for machine learning purposes has attracted considerable attention in the TDA research community over the past decade. While early vectors summaries of PDs such as *persistence landscape* (Bubenik 2015), *persistence silhouette* (Chazal et al. 2014), *Betti curve*[2] (Chazal and Michel 2021) and *persistence image* (Adams et al. 2017) have been implemented in both `Python` and `R` packages, there is no single package that systematically gathers them - the early ones as well as those introduced in recent years - in one place under unified syntax. Moreover, in `R`, all the code behind the existing vector implementations is written using standard `R` functions which may prove slow and inefficient for large-scale computations. The `TDAvec` `R` package (Islambekov and Luchinsky 2022) and its `Python` version aim to fill in some of these gaps. Its contributions can be summarized in the following three areas:

1. It expands the list of implemented vector summaries of PDs by providing vectorizations of eight functional summaries found in the TDA literature: *Betti function*, *persistence landscape function*, *persistence silhouette function*, *persistent entropy summary function* (Atienza, Gonzalez-Diaz, and Soriano-Trigueros 2020), *Euler characteristic curve* (Richardson and Werman 2014), *normalized life curve* (Chung and Lawson 2022), *persistence surface* (Adams et al. 2017) and *persistence block* (Chan et al. 2022) (see the Appendix for their definitions).

2. A univariate summary function f of a PD is typically vectorized by evaluating it at each point of a superimposed one dimensional grid and arranging the resulting values into a vector:

$$(f(t_1), f(t_2), \dots, f(t_n)) \in \mathbb{R}^n, \tag{1}$$

   where $t_1, t_2, \dots, t_n$ form an increasing sequence of scale values. For example, the `landscape()` and `silhouette()` functions of the `TDA` package compute vector summaries of persistence landscapes and silhouettes in this manner. The `TDAvec` package instead employs a different vectorization scheme which involves computing the average values of f between two consecutive scale values t_i and t_{i+1} using integration:

$$\Big(\frac{1}{\Delta t_1}\int_{t_1}^{t_2} f(t)dt, \frac{1}{\Delta t_2}\int_{t_2}^{t_3} f(t)dt, \dots, \frac{1}{\Delta t_{n-1}}\int_{t_{n-1}}^{t_n} f(t)dt\Big) \in \mathbb{R}^{n-1}, \tag{2}$$

   where $\Delta t_i = t_{i+1} - t_i$. Unlike (1), this vectorization method does not miss the behavior of f between neighboring scale points and applies to all univariate summary functions which are easy to integrate, namely persistence silhouette, persistent entropy summary function, Euler characteristic curve, normalized life curve and Betti function.

3. To achieve higher computational efficiency, all code behind the vector summaries of `TDAvec` is written in C++. For example, in `R`, computing the persistence landscape from a PD with the `TDAvec` package is more than 200 times faster than with the `TDA` package.

The `TDAvec` `R` package and a vignette showing its basic usage with examples are available on the CRAN repository[3]. For `Python` exmaples, we refer the readers to [@].

[2]also called *Betti function*

[3]https://cran.r-project.org/web/packages/TDAvec/index.html

Appendix: Definitions of the summary functions in `TDAvec`

Let $D = \{(b_i, d_i)\}_{i=1}^N$ be a persistence diagram.

1) *The kth order persistence landscape function* of D is defined as

$$\lambda_k(t) = k\max_{1\leq i\leq N}\Lambda_i(t), \quad k \in N,$$

where $k$max returns the kth largest value and

$$\Lambda_i(t) = \begin{cases} t - b_i & t \in [b_i, \frac{b_i+d_i}{2}] \\ d_i - t & t \in (\frac{b_i+d_i}{2}, d_i] \\ 0 & \text{otherwise} \end{cases}$$

2) *The pth power persistence silhouette function*:

$$\phi_p(t) = \frac{\sum_{i=1}^N |d_i - b_i|^p \Lambda_i(t)}{\sum_{i=1}^N |d_i - b_i|^p},$$

where

$$\Lambda_i(t) = \begin{cases} t - b_i & t \in [b_i, \frac{b_i+d_i}{2}] \\ d_i - t & t \in (\frac{b_i+d_i}{2}, d_i] \\ 0 & \text{otherwise} \end{cases}$$

3) *The persistent entropy summary function*:

$$S(t) = -\sum_{i=1}^N \frac{l_i}{L}\log_2(\frac{l_i}{L})\mathbf{1}_{[b_i,d_i]}(t),$$

where $l_i = d_i - b_i$ and $L = \sum_{i=1}^N l_i$.

4) *The Betti Curve*:

$$\beta(t) = \sum_{i=1}^N w(b_i, d_i)\mathbf{1}_{[b_i,d_i)}(t),$$

where the weight function $w(b, d) \equiv 1$.

5) *The Euler characteristic curve*:

$$\chi(t) = \sum_{k=0}^d (-1)^k \beta_k(t),$$

where $\beta_0, \beta_1, \dots, \beta_d$ are the Betti curves corresponding to persistence diagrams $D_0, D_1, \dots, D_d$ of dimensions $0, 1, \dots, d$ respectively, all computed from the same filtration.

6) *The normalized life curve*:

$$sl(t) = \sum_{i=1}^N \frac{d_i - b_i}{L}\mathbf{1}_{[b_i,d_i)}(t),$$

where $L = \sum_{i=1}^N (d_i - b_i)$.

7) *Persistence Surface*:

$$\rho(x, y) = \sum_{i=1}^N f(b_i, p_i)\phi_{(b_i,p_i)}(x, y),$$

where $\phi_{(b_i,p_i)}(x,y)$ is the Gaussian distribution with mean (b_i, p_i) and covariance matrix $\sigma^2 I_{2\times 2}$ and

$$f(b,p) = w(p) = \begin{cases} 0 & p \leq 0 \\ p/p_{max} & 0 < p < p_{max} \\ 1 & p \geq p_{max} \end{cases}$$

is the weighting function with p_{max} being the maximum persistence value among all persistence diagrams considered in the experiment.

8) *The persistence block*:

$$f(x,y) = \sum_{i=1}^{N} \mathbf{1}_{E(b_i,p_i)}(x,y),$$

where $E(b_i,p_i) = [b_i - \frac{\lambda_i}{2}, b_i + \frac{\lambda_i}{2}] \times [p_i - \frac{\lambda_i}{2}, p_i + \frac{\lambda_i}{2}]$ and $\lambda_i = 2\tau p_i$ with $\tau \in (0,1]$.

References

Adams, Henry, Tegan Emerson, Michael Kirby, Rachel Neville, Chris Peterson, Patrick Shipman, Sofya Chepushtanova, Eric Hanson, Francis Motta, and Lori Ziegelmeier. 2017. "Persistence Images: A Stable Vector Representation of Persistent Homology." *The Journal of Machine Learning Research* 18 (1): 218–52.

Atienza, Nieves, Rocio Gonzalez-Diaz, and Manuel Soriano-Trigueros. 2020. "On the Stability of Persistent Entropy and New Summary Functions for Topological Data Analysis." *Pattern Recognition* 107: 107509. https://doi.org/10.1016/j.patcog.2020.107509.

"Awesome-TDA." 2024. 2024. https://github.com/FatemehTarashi/awesome-tda.

Bauer, Ulrich. 2021. "Ripser: Efficient Computation of Vietoris-Rips Persistence Barcodes." *J. Appl. Comput. Topol.* 5 (3): 391–423. https://doi.org/10.1007/s41468-021-00071-5.

Brown, Shael, and Reza Farivar. 2024. *TDApplied: Machine Learning and Inference for Topological Data Analysis*. https://doi.org/10.32614/cran.package.tdapplied.

Brunson, Jason C., Brandon Demkowicz, and Sanmati Choudhary. 2024. *Tdaunif: Uniform Manifold Samplers for Topological Data Analysis*. https://doi.org/10.32614/CRAN.package.tdaunif.

Bubenik, Peter. 2015. "Statistical Topological Data Analysis Using Persistence Landscapes." *The Journal of Machine Learning Research* 16 (1): 77–102.

Carlsson, G. 2009. "Topology and Data." *Bulletin of the American Mathematical Society* 46 (2). https://doi.org/10.1090/S0273-0979-09-01249-X.

Chan, Kit C, Umar Islambekov, Alexey Luchinsky, and Rebecca Sanders. 2022. "A Computationally Efficient Framework for Vector Representation of Persistence Diagrams." *Journal of Machine Learning Research* 23 (268): 1–33.

Chazal, Frédéric, Brittany Terese Fasy, Fabrizio Lecci, Alessandro Rinaldo, and Larry Wasserman. 2014. "Stochastic Convergence of Persistence Landscapes and Silhouettes." In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, 474–83. https://doi.org/10.1145/2582112.2582128.

Chazal, Frédéric, and Bertrand Michel. 2021. "An Introduction to Topological Data Analysis: Fundamental and Practical Aspects for Data Scientists." *Frontiers in Artificial Intelligence* 4. https://doi.org/10.3389/frai.2021.667963.

Christopher, T, S Nathaniel, and BO Rann. 2018. "A Lean Persistent Homology Library for Python." *Open J* 925.

Chung, Yu-Min, and Austin Lawson. 2022. "Persistence Curves: A Canonical Framework for Summarizing Persistence Diagrams." *Advances in Computational Mathematics* 48 (1): 1–42. https://doi.org/10.1007/s10444-021-09893-4.

Edelsbrunner, Herbert, and John Harer. 2010. *Computational Topology: An Introduction*. American Mathematical Soc.

Fasy, Brittany T., Jisu Kim, Fabrizio Lecci, Clement Maria, David L. Millman, and Vincent Rouvreau. 2021. *TDA: Statistical Tools for Topological Data Analysis*. https://doi.org/10.32614/CRAN.package.TDA.

Hensel, Felix, Michael Moor, and Bastian Rieck. 2021. "A Survey of Topological Machine Learning Methods." *Frontiers in Artificial Intelligence* 4: 681108. https://doi.org/10.3389/frai.2021.681108.

Islambekov, Umar, and Alexey Luchinsky. 2022. *TDAvec: Vector Summaries of Persistence Diagrams.* https://doi.org/10.32614/CRAN.package.TDAvec.

Morozov, Dmitriy. 2018. "Dionysus 2 – Library for Computing Persistent Homology." 2018. https://github.com/mrzv/dionysus.

Nanda, Vidit, and Radmila Sazdanovic. 2013. "Simplicial Models and Topological Inference in Biological Systems." In *Discrete and Topological Models in Molecular Biology*, 109–41. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-40193-0_6.

Padellini, Tullia, and Francesco Palini. 2020. *kernelTDA: Statistical Learning with Kernel for Persistence Diagrams.* https://CRAN.R-project.org/package=kernelTDA.

Pearson, Paul, Daniel Muellner, and Gurjeet Singh. 2015. *TDAmapper: Topological Data Analysis Using Mapper.* https://github.com/paultpearson/TDAmapper.

Richardson, Eitan, and Michael Werman. 2014. "Efficient Classification Using the Euler Characteristic." *Pattern Recognition Letters* 49: 99–106. https://doi.org/10.1016/j.patrec.2014.07.001.

Rouvreau, V. 2020. "GUDHI User and Reference Manual." GUDHI Editorial Board.

Saul, Nathaniel. 2019. "Persim 0.3.7." 2019. https://persim.scikit-tda.org/en/latest/.

Tauzin, Guillaume, Umberto Lupo, Lewis Tunstall, Julian Burella Pérez, Matteo Caorsi, Anibal M Medina-Mardones, Alberto Dassatti, and Kathryn Hess. 2021. "Giotto-Tda:: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration." *Journal of Machine Learning Research* 22 (39): 1–6.

Van Veen, Hendrik Jacob, Nathaniel Saul, David Eargle, and Sam W Mangham. 2019. "Kepler Mapper: A Flexible Python Implementation of the Mapper Algorithm." *Journal of Open Source Software* 4 (42): 1315.

Wadhwa, Raoul R., Drew F. K. Williamson, Andrew Dhawan, and Jacob G. Scott. 2018. "TDAstats: R Pipeline for Computing Persistent Homology in Topological Data Analysis." *Journal of Open Source Software* 3 (28): 860. https://doi.org/10.21105/joss.00860.

Wadhwa, Raoul, Matt Piekenbrock, and Jacob Scott. 2020. *Ripserr: Calculate Persistent Homology with Ripser-Based Engines.* https://doi.org/10.32614/CRAN.package.ripserr.

Wickham, Hadley. 2016. *Ggplot2: Elegant Graphics for Data Analysis.* Springer-Verlag New York. https://ggplot2.tidyverse.org.

You, Kisung, and Byeongsu Yu. 2021. *TDAkit: Toolkit for Topological Data Analysis.* https://doi.org/10.32614/cran.package.tdakit.