giotto-tda: A Topological Data Analysis Toolkit for Machine Learning and Data Exploration

Guillaume Tauzin[1, 2]
gtauzin@protonmail.com

Umberto Lupo[3,4]
umberto.lupo@epfl.ch

Lewis Tunstall[4]
lewis.c.tunstall@gmail.com

Julian Burella Pérez[5]
julian.burellaperez@heig-vd.ch

Matteo Caorsi[4]
m.caorsi@l2f.ch

Wojciech Reise[6]
reisewojciech@gmail.com

Anibal M. Medina-Mardones[2]
anibal.medinamardones@epfl.ch

Alberto Dassatti[5]
alberto.dassatti@heig-vd.ch

Kathryn Hess[2]
kathryn.hess@epfl.ch

[1]INAIT SA
[2]Laboratory for Topology and Neuroscience, EPFL
[3]Laboratory of Computational Biology and Theoretical Biophysics, EPFL
[4]L2F SA
[5]School of Management and Engineering Vaud, HES-SO,
University of Applied Sciences Western Switzerland
[6] DataShape, Inria Saclay – Île-de-France

Abstract

We introduce *giotto-tda*, a `Python` library that integrates high-performance topological data analysis with machine learning via a *scikit-learn*–compatible API and state-of-the-art `C++` implementations. The library's ability to handle various types of data is rooted in a wide range of preprocessing techniques, and its strong focus on data exploration and interpretability is aided by an intuitive plotting API. Source code, binaries, examples, and documentation can be found at https://github.com/giotto-ai/giotto-tda.

1 Introduction

Topological Data Analysis (TDA) uses tools from algebraic and combinatorial topology to extract features that capture the shape of data [1]. In recent years, algorithms based on topology have proven very useful in the study of a wide range of problems. In particular, *persistent homology* has had significant impact on data intensive challenges including the classification of porous materials [2], the study of structures in the weight space of CNNs [3], and the discovery of links between structure and function in the brain [4]. The *Mapper* algorithm has also received considerable attention after its use in the identification of a highly treatable subgroup of breast cancers [5].

Despite its power and versatility, TDA has remained outside the toolbox of most Machine Learning (ML) practitioners, largely because current implementations are developed for research purposes and not in high-level languages. The aim of *giotto-tda* is to fill this gap by making TDA accessible to the `Python` data science community, while supporting research. To this end, *giotto-tda* inherits the flexibility of *scikit-learn*, the most popular all-purpose ML framework [6], and extends it with

Topological Data Analysis and Beyond Workshop at the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

TDA capabilities including a wide range of persistent homology and Mapper-type algorithms. It enables TDA to be applied to univariate and multivariate time series, images, graphs, and their higher dimensional analogues, simplicial complexes. This makes *giotto-tda* the most comprehensive `Python` library for topological *machine learning* and data exploration to date.

2 Architecture

To use topological features in machine learning effectively, techniques such as hyperparameter search and feature selection need to be applied at a large scale. Facilitating these processes is one of the reasons why *giotto-tda* maintains and extends compatibility with the *scikit-learn* API. *giotto-tda* provides users with full flexibility in the design of TDA pipelines via modular `estimators`, and the highly visual nature of topological signatures is harnessed via a plotting API based on *plotly*. This exposes a set of external functions and class methods to plot and interact with intermediate results represented as standard *NumPy* arrays [7].

To combine TDA methods with the many time-delay embedding techniques used frequently in time series prediction [8; 9], one must allow `transformers` extra flexibility not present in the basic architecture of *scikit-learn*. To support this task, *giotto-tda* provides a novel `TransformerResamplerMixin` class, as well as an extended version of *scikit-learn*'s `Pipeline`. [1]

Through *scikit-learn*–based wrapper libraries for *PyTorch* [10] such as *skorch* [11] and the *scikit-learn* interface offered in *TensorFlow* [12], it is also possible to use deep learning models as final estimators in a *giotto-tda* `Pipeline`.

3 Persistent homology

Persistent homology is one of the main tools in TDA. It extracts and summarises, in so-called persistence diagrams, multi-scale relational information in a manner similar to hierarchical clustering, but also considering higher-order connectivity. It is a very powerful and versatile technique. To fully take advantage of it in ML and data exploration tasks, *giotto-tda* offers *scikit-learn*–compatible components that enable the user to a) transform a wide variety of data input types into forms suitable for computing persistent homology, b) compute persistence diagrams according to a large selection of algorithms, and c) extract a rich set of features from persistence diagrams. The result is a framework for constructing end-to-end `Pipeline` objects to generate carefully crafted topological features from each sample in an input raw data collection. At a more technical level, features are often extracted from persistence diagrams by first representing them as curves or images, or by defining kernels. Each method for doing so typically comes with a set of hyperparameters that must be tuned to the problem at hand. *giotto-tda* exposes a large selection of such algorithms and, by tightly integrating with the *scikit-learn* API for hyperparameter search, cross-validation and feature selection, allows for simple data-driven tuning of the many hyperparameters involved.

In Figure 1, we present some of the many possible feature-generation workflows that are made available by *giotto-tda*, starting with a sample in the input raw data collection.

A comparison between *giotto-tda* and other `Python` persistent homology libraries is shown in Table 1. A highlight of this comparison is the presence of directed persistent homology [4; 13], a viewpoint that emphasises the non-symmetric nature of many real-world interactions. *giotto-tda* provides preprocessing `transformers` to make use of it for a wide range of input data types.

Our library matches the code and documentation standards set by *scikit-learn*, and relies on state-of-the-art external `C++` libraries [24; 25; 26; 13] using new performance-oriented bindings based on *pybind11* [27]. In the case of `ripser` [25], bindings from `ripser.py` [28] were adapted. In the case of `flagser` [13], no `Python` API was available prior to *giotto-tda*'s sibling project *pyflagser*.[4] As concerns the computation of Vietoris–Rips barcodes, *giotto-tda* improves on the state-of-the-art runtimes achieved in [15] (and now part of *GUDHI*'s C++ codebase) by combining their edge collapse algorithm with *ripser*. Furthermore, the *joblib* package is used throughout to parallelize computations

[1] The interested reader is referred to https://giotto-ai.github.io/gtda-docs/0.3.1/notebooks/time_series_forecasting.html for a tutorial on these concepts and features.

[4] Source code available at https://github.com/giotto-ai/pyflagser.

		giotto-tda v0.3.1	*GUDHI* v3.3.0	*scikit-tda*	*Dionysus* 2
time series	sliding window	Yes	-	-	-
	Takens' embedding	Yes	Yes	-	-
	Pearson dissimilarity	Yes	-	-	-
point clouds & metric spaces	consistent rescaling [14]	Yes	-	-	-
	k-nearest neighbors	Yes	Yes	-	-
	subsampling	-	Yes	-	-
	density	-	Yes	-	-
	Gromov–Hausdorff distance	-	-	Yes	-
	distance to measure	-	Yes	-	-
images	binarizer	Yes	-	-	-
	image to point cloud	Yes	-	-	-
	height filtration	Yes	-	-	-
graphs	transition graph	Yes	-	-	-
	geodesic distance	Yes	-	-	-
	flag filtrations	Yes (flagser)	-	-	-
undirected simplicial persistent homology	Vietoris–Rips	Yes	Yes	Yes	Yes
	sparse Rips	Yes	Yes	-	-
	weighted Rips	-	Yes	-	-
	edge collapse [15]	Yes	`C++` only	-	-
	Čech	Yes	`C++` only	Yes	-
	alpha	Yes (weak [16])	Yes	Yes	-
	witness	-	Yes	-	-
	tangential	-	Yes	-	-
	extended	-	Yes	Yes	-
	zigzag	-	-	-	Yes
	lower star	-	Yes	Yes	Yes
other persistent homology	directed simplicial	Yes	-	-	-
	cubical	Yes	Yes	-	-
diagram representations	persistence landscape	Yes	Yes	-	-
	Betti curves	Yes	Yes	-	-
	silhouette	Yes	Yes	-	-
	heat representation [17]	Yes	-	-	-
	persistent image	Yes	Yes	Yes	-
diagram distances and kernels	bottleneck distance	Yes	Yes	Yes	Yes
	Wasserstein distance	Yes	Yes	-	Yes
	persistent Fisher [18]	-	Yes	-	-
	heat [17]	Yes	Yes	Yes	-
	persistent weighted Gaussian [19]	-	Yes	-	-
	sliced Wasserstein [20]	-	Yes	Yes	-
	L^p distance between representations	Yes	-	-	-
diagram features	prominent points	-	Yes	-	-
	ATOL [21]	-	Yes	-	-
	persistence entropy	Yes	Yes	Yes	-
	number of points	Yes	-	-	-
	complex polynomial [22]	Yes	Yes	-	-
	topological vector [23]	-	Yes	-	-
	amplitude	Yes	-	-	-
	curve features	Yes	-	-	-
plotting	time series	Yes	-	-	-
	point cloud	Yes	-	-	-
	image	Yes	-	-	-
	graph	Yes	-	-	-
	diagram	Yes	Yes	Yes	Yes
	diagram density	-	Yes	-	Yes
	representation	Yes	-	-	-

Table 1: Snapshot of the feature support present on the main `Python` open source libraries with persistent homology capabilities. [3]

across batches of data. Whenever possible, we contributed with enhancements and bug fixes to some of *giotto-tda*'s `C++` and `Python` dependencies.

4 Mapper

Mapper is a representation technique of high-dimensional data that, combining the application of filter functions and partial clustering, creates a simple and topologically meaningful description of the input as an unweighted graph (or, more generally, as a simplicial complex). It is primarily used as a data visualization tool to explore substructures of interest in data. In *giotto-tda*, this algorithm is realised as a sequence of steps in a *scikit-learn* `Pipeline`, where the clustering step can be parallelized. The resulting graph is visualized through an interactive plotting API. This design choice provides a great

[4] *GUDHI* [24], *scikit-tda* [29], *Dionysus 2* [30].

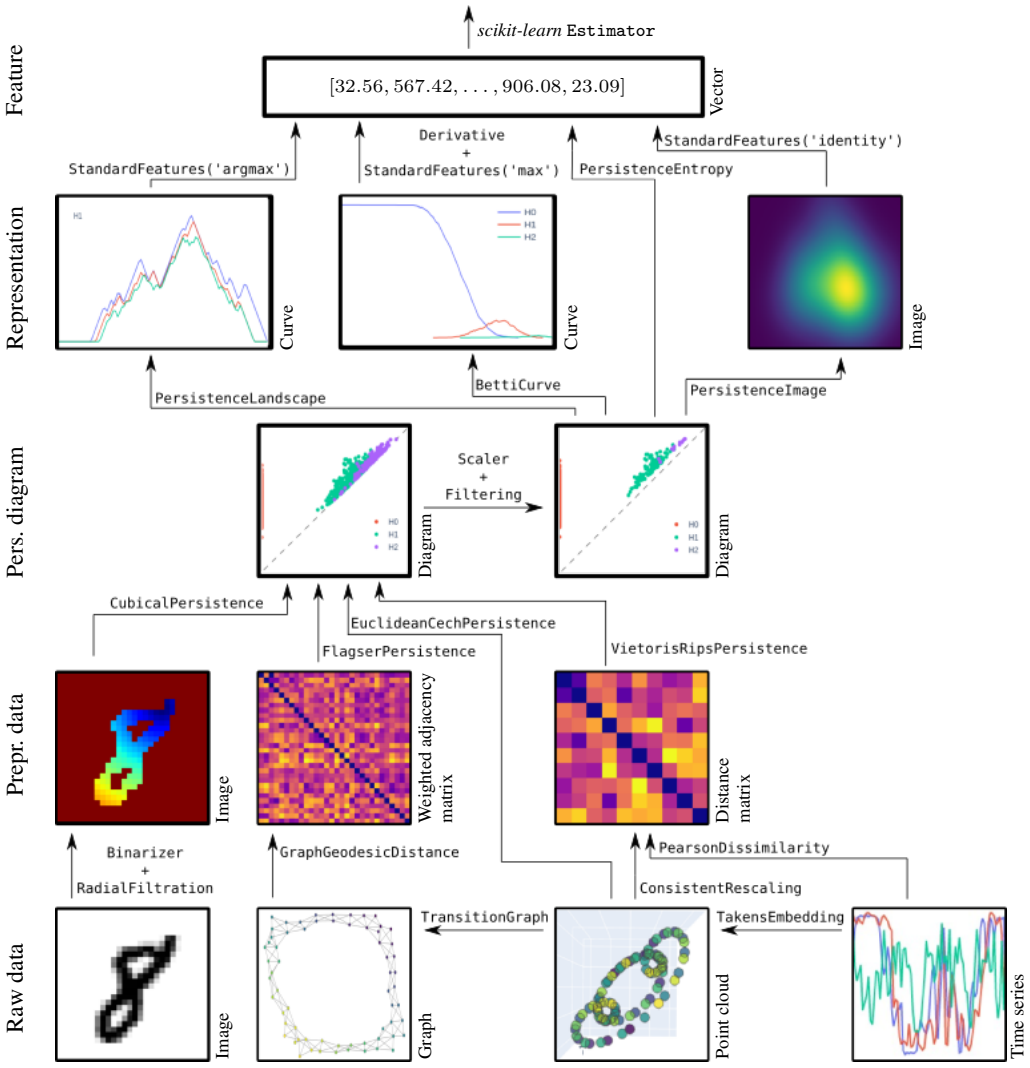

Figure 1: Non-exhaustive depiction of *giotto-tda* capabilities. Arrows represent operations available as `transformers` and paths potential `pipelines`.

deal of interoperability and computational efficiency, allowing users to a) realize relevant steps of the Mapper algorithm through any *scikit-learn* `Estimator`, b) integrate Mapper pipelines as part of a larger ML workflow, and c) make use of memory caching to avoid unnecessary re-computations. Memory caching is especially useful for interactive plotting, where *giotto-tda* allows users to tune Mapper's hyperparameters and observe how the resulting graph changes in real time. An example of a mapper skeletonization adapted from [31] is shown in Fig. 2.

To the best of our knowledge, *KeplerMapper* [32] is the only alternative open-source implementation of Mapper in `Python` that provides general-purpose functionality. Although *KeplerMapper* also provides the flexibility to use *scikit-learn* `estimators` to generate Mapper graphs, it does not implement all steps of the algorithm in a single class and is only partially compatible with *scikit-learn* `pipelines`. Moreover, it does not implement memory caching or provide real-time hyperparameter interactivity in the visualization.

5 Project management

Easy installation: Binary packages are available for all major operating systems on the PyPI package repository and can be installed easily by running `python -m pip install -U giotto-tda`.

Figure 2: Mapper graph generated by *giotto-tda* based on the height of a 3D model.

Code quality: The code is unit-tested throughout using *pytest* and *hypothesis* and, as of v0.3.1, test coverage is at 98%. The code follows `PEP8` standards and adheres to the `Python` coding guideline and *NumPy*-style documentation. CI/CD best practices are in place via Azure Pipelines.

Community-based development: We base *giotto-tda*'s development on collaborative tools such as Git, GitHub, and Slack. Contributions are encouraged, and we actively make use of GitHub's issue tracker to provide support and discuss ideas. The library is distributed under the GNU AGPLv3 license.

Documentation and learning resources: A detailed API reference is provided using *sphinx*.[5] To lower the entry barrier, we provide a theory glossary and a wide range of tutorials and examples that help new users explore how TDA-based ML pipelines can be applied to datasets of various sorts.

Project relevance: At the time of writing, the GitHub repository has attracted over 300 stars and between 500 and 1000 visits per week. The PyPI package is downloaded 350 times per month. The library appears in *scikit-learn*'s curated list of related projects.

6 Concluding remarks

The very active research field of TDA provides algorithms that can be used at any step of a ML pipeline. *giotto-tda* aims to make these algorithms available in a form that is useful to both the research and data science communities, thus allowing them to use TDA as a part of large-scale ML tasks. We have written *giotto-tda* under the code and documentation standards of *scikit-learn* and, alongside further performance optimization of the existing `C++` code, future developments will include the first implementation of novel TDA algorithms such as persistence Steenrod diagrams [33].

Acknowledgements

We thank Roman Yurchak, Philippe Nguyen, and Philipp Weiler for their numerous ideas and contributions. Support from Innosuisse (grant number 32875.1 lP-ICT) is gratefully acknowledged.

References

[1] Gunnar Carlsson. Topology and data. *Bull. Amer. Math. Soc. (N.S.)*, 46(2):255–308, 2009.

[2] Yongjin Lee, Senja D Barthel, Paweł Dłotko, et al. High-Throughput Screening Approach for Nanoporous Materials Genome Using Topological Data Analysis: Application to Zeolites. *Journal of chemical theory and computation*, 14(8):4427–4437, August 2018.

[3] Rickard Brüel Gabrielsson and Gunnar Carlsson. Exposition and interpretation of the topology of neural networks. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1069–1076. IEEE, 2019.

[5] Currently hosted at https://giotto-ai.github.io/gtda-docs/latest/modules/index.html.

[4] Michael W. Reimann, Max Nolte, Martina Scolamiero, et al. Cliques of neurons bound into cavities provide a missing link between structure and function. *Frontiers in Computational Neuroscience*, 11:48, 2017.

[5] Monica Nicolau, Arnold J. Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, 108(17):7265–7270, 2011.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, et al. Array programming with NumPy. *Nature*, 585:357–362, 2020.

[8] Jose A. Perea. Topological times series analysis. *Notices Amer. Math. Soc.*, 66(5):686–694, 2019.

[9] Audun Myers, Elizabeth Munch, and Firas A. Khasawneh. Persistent homology of complex networks for dynamic state detection. *Phys. Rev. E*, 100:022314, Aug 2019.

[10] Adam Paszke, Sam Gross, Francisco Massa, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

[11] Marian Tietz, Thomas J. Fan, Daniel Nouri, Benjamin Bossan, and skorch Developers. *skorch: A scikit-learn compatible neural network library that wraps PyTorch*, 2017.

[12] Martín Abadi, Ashish Agarwal, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[13] Daniel Lütgehetmann, Dejan Govc, Jason P. Smith, et al. Computing persistent homology of directed flag complexes. *Algorithms*, 13(1):19, 2020.

[14] Tyrus Berry and Timothy Sauer. Consistent manifold representation for topological data analysis. *Foundations of Data Science*, 1(1):1, 2019.

[15] Jean-Daniel Boissonnat and Siddharth Pritam. Edge Collapse and Persistence of Flag Complexes. In *36th International Symposium on Computational Geometry (SoCG 2020)*, volume 164, pages 19:1–19:15, 2020.

[16] Rickard Brüel Gabrielsson, Bradley J Nelson, Anjan Dwaraknath, and Primoz Skraba. A topology layer for machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1553–1563, 2020.

[17] J. Reininghaus, S. Huber, U. Bauer, and R. Kwitt. A stable multi-scale kernel for topological machine learning. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4741–4748, 2015.

[18] Tam Le and Makoto Yamada. Persistence Fisher kernel: A Riemannian manifold kernel for persistence diagrams. In *Advances in Neural Information Processing Systems*, volume 31, pages 10007–10018, 2018.

[19] Genki Kusano, Yasuaki Hiraoka, and Kenji Fukumizu. Persistence weighted Gaussian kernel for topological data analysis. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48, pages 2004–2013, 2016.

[20] Mathieu Carrière, Marco Cuturi, and Steve Oudot. Sliced Wasserstein kernel for persistence diagrams. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 664–673, 2017.

[21] Martin Royer, Frédéric Chazal, Clément Levrard, Umeda Yuhei, and Ike Yuichi. ATOL: Measure vectorization for automatic topologically-oriented learning. *arXiv:1909.13472*, 2020.

[22] Barbara Di Fabio and Massimo Ferri. Comparing persistence diagrams through complex vectors. In *International Conference on Image Analysis and Processing*, pages 294–305. Springer, 2015.

[23] Mathieu Carrière, Steve Y. Oudot, and Maks Ovsjanikov. Stable topological signatures for points on 3d shapes. *Computer Graphics Forum*, 2015.

[24] The GUDHI Project. *GUDHI User and Reference Manual*. GUDHI Editorial Board, 3.3.0 edition, 2020.

[25] Ulrich Bauer. Ripser: efficient computation of vietoris-rips persistence barcodes. *arXiv preprint arXiv:1908.02518*, 2019.

[26] Michael Kerber, Dmitriy Morozov, and Arnur Nigmetov. Geometry helps to compare persistence diagrams. *Journal of Experimental Algorithmics*, 22:1–20, 09 2017.

[27] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – seamless operability between C++11 and Python, 2017.

[28] Christopher Tralie, Nathaniel Saul, and Rann Bar-On. Ripser.py: A lean persistent homology library for Python. *Journal of Open Source Software*, 3(29):925, 2018.

[29] Nathaniel Saul and Chris Tralie. Scikit-TDA: Topological data analysis for Python, 2019.

[30] Dmitriy Morozov. Dionysus 2 – library for computing persistent homology, 2018.

[31] Jeff Murugan and Duncan Robertson. An introduction to topological data analysis for physicists: From LGM to FRBs. *arXiv preprint arXiv:1904.11044*, 2019.

[32] Hendrik van Veen, Nathaniel Saul, David Eargle, et al. Kepler Mapper: A flexible Python implementation of the Mapper algorithm. *Journal of Open Source Software*, 4(42):1315, 2019.

[33] Anibal M. Medina-Mardones. Persistence Steenrod modules. *arXiv:1812.05031*, 2018.