**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\(^5\)
jburellaperez@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](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...
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.

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

Through scikit-learn–based wrapper libraries for PyTorch such as skorch and the scikit-learn interface offered in TensorFlow, 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, 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 using new performance-oriented bindings based on pybind11. In the case of ripser, bindings from ripser.py were adapted. In the case of flagser, no Python API was available prior to giotto-tda’s sibling project pyflagser. 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](https://giotto-ai.github.io/gtda-docs/0.3.1/notebooks/time_series_forecasting.html) for a tutorial on these concepts and features.

2 Source code available at [https://github.com/giotto-ai/pyflagser](https://github.com/giotto-ai/pyflagser)
<table>
<thead>
<tr>
<th></th>
<th>giotto-tda</th>
<th>GUDHI v3.3.0</th>
<th>scikit-tda</th>
<th>Dionysus 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>time series</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sliding window</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Takens’ embedding</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pearson dissimilarity</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>point clouds &amp; metric spaces</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>consistent rescaling</td>
<td>[14]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$k$-nearest neighbors</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>subsampling</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>density</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gromov–Hausdorff distance</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>distance to measure</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>images</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>binarizer</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>image to point cloud</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>height filtration</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>graphs</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>transition graph</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>geodesic distance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>flag filtrations</td>
<td></td>
<td></td>
<td></td>
<td>Yes (flagser)</td>
</tr>
<tr>
<td><strong>undirected simplicial persistent homology</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vietoris–Rips</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>sparse Rips</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>weighted Rips</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>edge collapse [15]</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Čech</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>witness</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>tangential</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>extended</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>zigzag</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>lower star</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>other persistent homology</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>directed simplicial</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>cubical</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td><strong>diagram representations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>persistence landscape</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Betti curves</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>silhouette</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>heat representation [17]</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>persistent image</td>
<td>Yes</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td><strong>diagram distances and kernels</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bottleneck distance</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Wasserstein distance</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>persistent Fisher [15]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>heat [15]</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>persistent weighted Gaussian [17]</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>sliced Wasserstein [20]</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>$L^1$ distance between representations</td>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>diagram features</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>prominent points</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>ATOL [21]</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>persistence entropy</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>number of points</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complex polynomial [22]</td>
<td>Yes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>topological vector [23]</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>amplitude</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>curve features</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>plotting</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time series</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>point cloud</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>image</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>graph</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>diagram</td>
<td></td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>diagram density</td>
<td>-</td>
<td></td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>representation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

\(^3\)GUDHI [24]. scikit-tda [29]. Dionysus 2 [30].
deal of interoperability and computational efficiency, allowing users to a) realize relevant steps of the Mapper algorithm through any *scikitlearn* 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 *scikitlearn* estimators to generate Mapper graphs, it does not implement all steps of the algorithm in a single class and is only partially compatible with *scikitlearn* 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`. 

![Diagram](image)
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`. 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 IP-ICT) is gratefully acknowledged.

### References


