TDAvectorize: Python library for vectorization of TDA methods

Aleksei Luchinsky[1¶] **and Umar Islambekov**[1*]

1 Bowling Green State University ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

1) Summary

aaa

2) Statement of Need

As it was mentioned in the summary section, persistence diagrams in the original format do not form a vector space and cannot be easily used in ML algorithms. To solve this problem vectorization transformation is performed, when from each persistence diagram a fixed number of predictors is extracted and the resulting data set is used in the analysis.

There are lot of different vector summary types available in TDA literature. Good up-to-date review of the current status of the TDA vectorization methods and libraries can be found in the page (*Awesome-TDA*, 2024). One can list, for example, classical Euler Characteristic Curve (Richardson & Werman, 2014), Persistence Landscape (Bubenik & others, 2015), (Chazal et al., 2014), or Persistence Surface (Adams et al., 2017). Later such summaries as Persistence Entropy Summary (Atienza et al., 2020), Betti Curves (also known as Persistence Block) (Chazal & Michel, 2021), (Chung & Lawson, 2022), (Chan et al., 2022), Normalized Life Curves (Chung & Lawson, 2022) were introduced. Finally, methods of Persistence Images (Adams et al., 2017) and Vectorized Persistence Blocks (Chan et al., 2022) were added. The last two methods work with birth-death space as whole and produces 2-dimensional vector as a result. You can find more details about listed vectorization summaries in the appendix.

For computational analysis we need to perform some computer simulations. are listed above vectorization summary methods were implemented in a series of libraries, both Python (see, for example, (*Giotto-TDA*, 2024), (*Geometry Understaing in High Dimesions*, 2024), (*Persim 0.3.7 Documentation*, 2024)) and R (see, for example, (*TDA*, 2024), (*TDAstats*, 2024), (*TDAvec*, 2022)). As you can see from the table below, however, different methods are scattered among these libraries, there is no python package that implements all of them. In the proposed library we are filling this gap.

Method	this	Giotto-TDA	GUDHI	Persim	TDA	TDAvec
BC	✓	✓	✓			✓
ECC	✓					✓
NLC	✓					✓
PES	✓		✓			✓
PS	✓	✓	✓		✓	✓
PL	✓	✓	✓	✓	✓	✓
PI	✓	✓	✓	✓		✓
VPB	✓					✓

3) Software Details and Program Workflow

Considered in this paper package follows usual to TDA approach workflow: we start with some original data (collection of the data clouds, for example), convert them to the persistence diagrams, and then vectorize them using implemented in the library functions. Resulting vectors are then used as predictors for some ML method, like linear regression, logistic regression, classification, etc.

In the current section we will demonstrate the work of the proposed library on a simple example of 2-dimensional ellipse-like point clouds. Using included in the package function **createEllipse()** some set of ellipses with different proportions of axis is generated:

```
> clouds = []
> ratList = np.random.uniform(-0.5, 0.5, 10**3)
> for ratio in ratList:
>    clouds = clouds + [createEllipse(a=1-ratio, b=1, eps=0.1)]
```

You can see some typical results in the figure below:

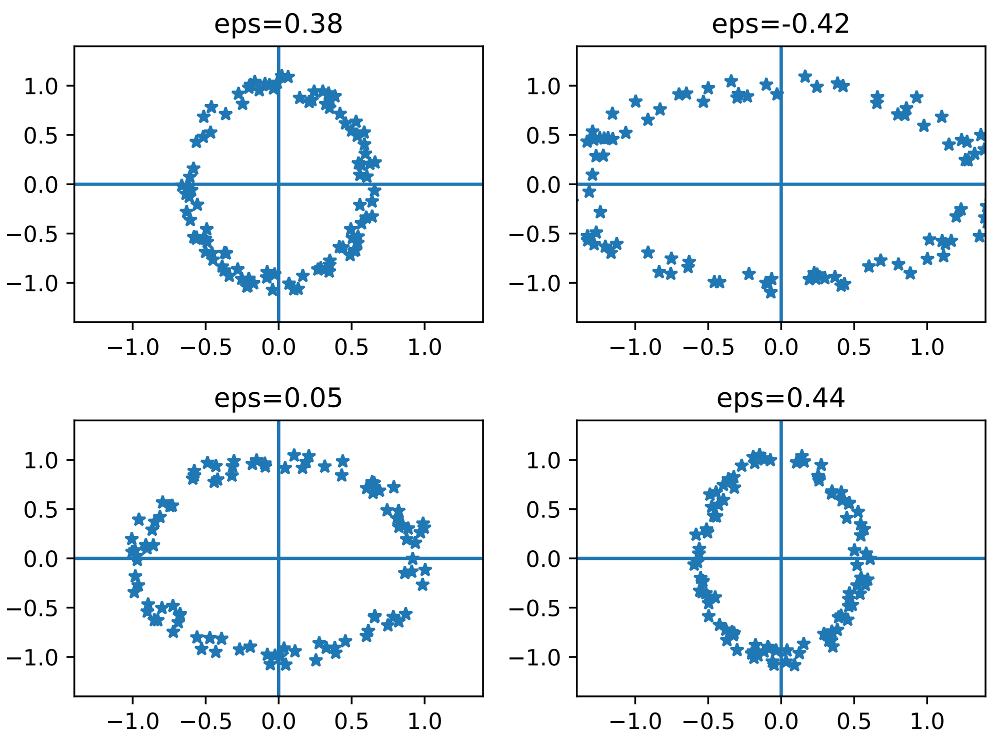


Figure 1: Sample Point Clouds

The most simples way to generate persistence diagrams from these point clouds is to use supplied with use main package class **TDAvectorizer**:

```
> vectorizer = TDAvectorizer()
> vectorizer.fit(clouds)
```

In the figure below you can see some example of persistence diagrams:

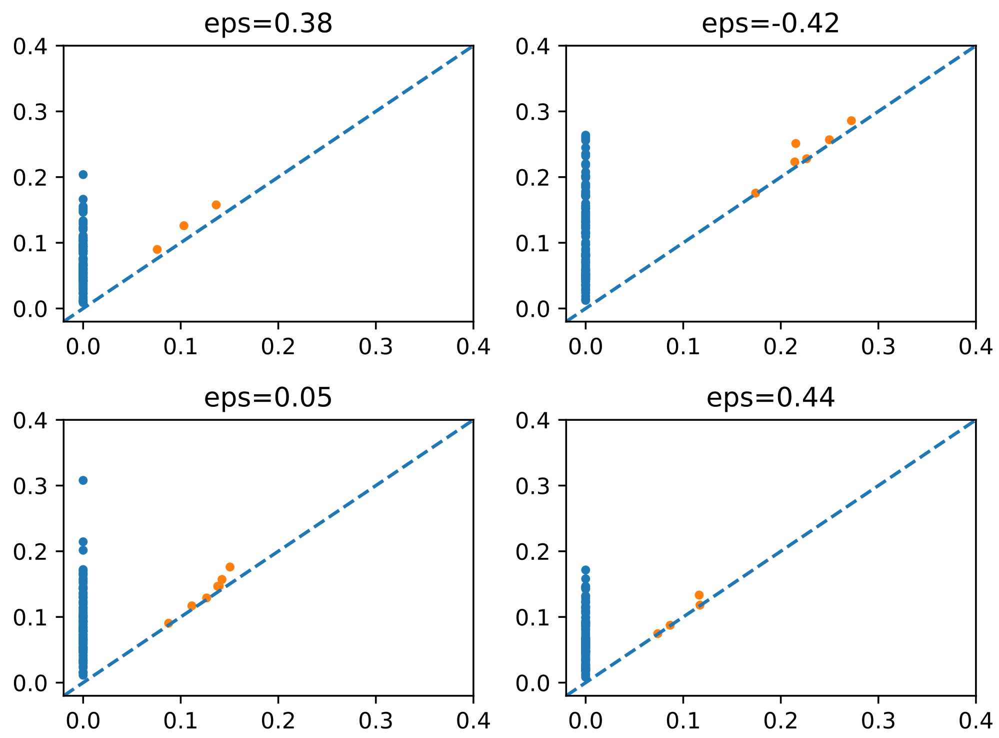


Figure 2: Sample Persistence Diagrams

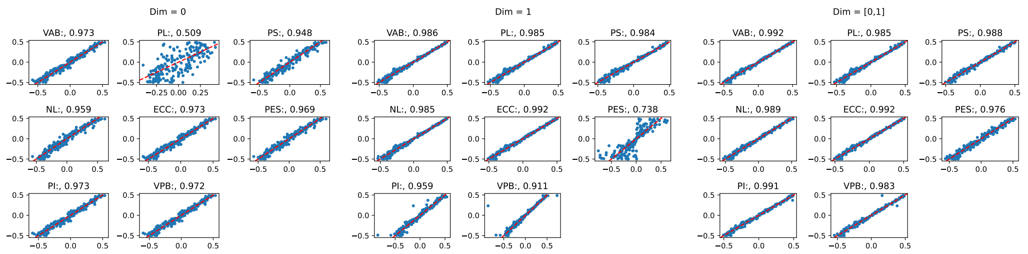
Once the TDAvectorizer is fitted, you can transform these persistence diagrams into vector representation using the `transform` method. The resulting vectors are then used as predictors:

```
> X = vectorizer.transform(homDim = 1, output = "PS")
> X_train, X_test, y_train, y_test = \
>     train_test_split(X, ratList, train_size=0.8, random_state=42)
> model = LinearRegression().fit(X_train, y_train)
> score = model.score(X_test, y_test)
> print("score = {:.3f}".format(score))
score = 0.979
```

You can see from this example, that when calling the `transform` method you can choose different vectorization methods and dimension of homology dimension. It is also possible to specify such parameters as vector grid, etc. The same parameters can also be stored as object properties by calling the `setParams` method:

```
> vectorizer.setParams({"homDim": 1, "output": "PS"})
```

Described approach makes it convenient to perform a systematic analysis and compare performance of different vector representations. In the figure below you can see some correlations between vectors and their correlation coefficients:



To get more detailed information you should perform such simulation several times, so that you can see the general pattern, determine mean values and spread of each methods' score. You can see in the table below the results of the analysis:

method	0	1	[0, 1]
ECC	0.97 ± 0.004	0.992 ± 0.001	0.968 ± 0.107
NL	0.955 ± 0.006	0.985 ± 0.003	0.985 ± 0.014
PES	0.966 ± 0.004	0.768 ± 0.016	0.974 ± 0.004
PI	0.97 ± 0.004	0.714 ± 0.223	0.934 ± 0.085
PL	0.491 ± 0.053	0.985 ± 0.002	0.985 ± 0.002
PS	0.943 ± 0.006	0.964 ± 0.094	0.977 ± 0.036
VAB	0.969 ± 0.004	0.964 ± 0.067	0.991 ± 0.003
VPB	0.969 ± 0.004	0.851 ± 0.246	0.755 ± 0.283

From this table it is clear that almost all results are comparable with each other, but ECC method with homDim = 1 is the best solution for the current data set. Surprisingly, for some of the feature extraction methods (PES with homDim = 1 and PL with homDim = 0) the results were very bad and the simulation score turned out to be negative.

It should be mentioned also that TDAvectorizer class is not the only way to perform the vectorization of the persistence diagram. In addition to described above approach you can also call directly provided with the package vectorization functions, like

```
> TDAvectorizer.computeECC(pd, 0, np.linspace(0,2,20))
```

The format of such functions is:

- computePL(PD, homDim, xSeq, k=1)
- computePS(PD, homDim, xSeq, p=1)
- computeNL(PD, homDim, xSeq)
- computeVAB(D, homDim, xSeq)
- computeECC(D, homDim, xSeq)
- computePES(D, homDim, xSeq)
- computePI(PD, homDim, xSeq, ySeq, sigma)
- computeVPB(PD, homDim, xSeq, ySeq, tau=0.3)

where *PD* is the persistence diagram we are analyzing, homDim is the homology dimension, xSeq and ySeq are grid vectors in x and (x, y) space, while other parameters are specific for each vectorization function.

5) Conclusion

6) Acknowledgements

Appendix: Vectorization Summaries Calculation Details

All defined above functions are now elements of some vector spaces and can be used in theoretical statistical analysis. In practical calculations, however, it is useful to digitize them and consider the values on some discrete 1-dimensional or 2-dimensional grids.

As it was noticed in the previous section, lots of different vectorization methods of the can be found in the literature. For a given persistence diagram

$$PD = \{(b_i, d_i\}_{i=1}^N$$

we can consider such quantities as

1) **Betti Curves**, when each value of dimension d corresponds to function

$$\beta_d(t) = \sum_{i=1}^{N} I_{[b_i,d_i)}(t),$$

where $I_{[b,d)}(t)$ stands for the indicator function, which is equal to unity on the region $b \le t < d$ and vanishes outside. In the following we will refer to this vectorization as **BC**.

2) **Euler Characteristic Curve**, which is a linear combination of the Betti Curves

$$\chi(t) = \sum_d (-1)^d \beta_d(t)$$

In the following it will be referred to as **EEC**.

3) **Normalized Line Curve**, where for each dimension d we have

$$s_d(t) = \sum_{i=1}^{N} \frac{d_i - b_i}{L} I_{[b_i,d_i]}(t),$$

where

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

In the following it will be referred to as **NLC**.

4) **Persistence Entropy Summary** function

$$S_d(t) = \sum_{i=1}^{N} \frac{d_i - b_i}{L} \log_2 \left(\frac{d_i - b_i}{L}\right) I_{[b_i,d_i)}(t)$$

In the following it will be referred to as **PES**.

5) **Persistence Silhouette** function

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

where p is a hyper-parameter of the model and a block triangle function Λ is defined as

$$\Lambda_i(t) = \begin{cases} t - b_i & b_i \le t \le (b_i + d_i)/2, \\ d_i - t & (b_i + d_i)/2 \le t \le d_i, \\ 0 & \text{otherwise} \end{cases}$$

In the following it will be referred to as **PS**.

6) **Persistence Landscape** function (PL in the following)

$$\lambda_k(t) = \arg\max_{1 \le i \le N} \Lambda_i(t)$$

7) **Persistence Image** function (PI in the following)

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

where

$$\phi_{(b_i,d_i)}(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(x - b_i)^2 + (y - p_i)^2}{2\sigma^2}$$

is a Gauss distribution centered at the point $(b_i, p_i = d_i - b_i)$ and

$$f(b,p) = w(p) = \begin{cases} 0 & p \le 0 \\ p/p_{max} & 0 \le p \le p_{max}, \\ 1 & p \ge p_{max}. \end{cases}$$

8) **Vectorized Persistence Block** (VPB in the following)

$$V(x,y) = \sum_{i=1}^{N} I_{E(b_i,p_i)}(x,y),$$

where the indicator function is different from zero on the rectangle

$$E(b_i,p_i) = \left[b_i - \frac{\lambda_i}{2}; b_i + \frac{\lambda_i}{2}\right] \times \left[p_i - \frac{\lambda_i}{2}; p_i + \frac{\lambda_i}{2}\right]$$

References

Adams, H., Emerson, T., Kirby, M., Neville, R., Peterson, C., Shipman, P., Chepushtanova, S., Hanson, E., Motta, F., & Ziegelmeier, L. (2017). Persistence images: A stable vector representation of persistent homology. *Journal of Machine Learning Research*, *18*(8), 1–35.

Atienza, N., González-Díaz, R., & Soriano-Trigueros, M. (2020). On the stability of persistent entropy and new summary functions for topological data analysis. *Pattern Recognition*, *107*, 107509.

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

Bubenik, P., & others. (2015). Statistical topological data analysis using persistence landscapes. *J. Mach. Learn. Res.*, *16*(1), 77–102.

Chan, K. C., Islambekov, U., Luchinsky, A., & Sanders, R. (2022). A computationally efficient framework for vector representation of persistence diagrams. *Journal of Machine Learning Research*, *23*(268), 1–33.

Chazal, F., Fasy, B. T., Lecci, F., Rinaldo, A., & Wasserman, L. (2014). Stochastic convergence of persistence landscapes and silhouettes. *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, 474–483.

Chazal, F., & Michel, B. (2021). An introduction to topological data analysis: Fundamental and practical aspects for data scientists. *Frontiers in Artificial Intelligence*, *4*, 667963.

Chung, Y.-M., & Lawson, A. (2022). Persistence curves: A canonical framework for summarizing persistence diagrams. *Advances in Computational Mathematics*, *48*(1), 6.

Geometry understaing in high dimesions. (2024). https://gudhi.inria.fr/

Giotto-TDA. (2024). https://giotto-ai.github.io/gtda-docs/0.5.1/index.html

Persim 0.3.7 documentation. (2024). https://persim.scikit-tda.org/en/latest/

Richardson, E., & Werman, M. (2014). Efficient classification using the euler characteristic. *Pattern Recognition Letters*, *49*, 99–106.

TDA: Statistical tools for topological data analysis. (2024). https://cran.r-project.org/web/packages/TDA/

TDAstats: Topological data analysis in r. (2024). https://github.com/rrrlw/TDAstats

TDAvec: Vector summaries of persistence diagrams. (2022). https://cran.r-project.org/web/packages/TDAvec/index.html