The gudhi stat doc.
Ver 1.0.

1 Idea

When it comes to the statistics and machine learning, one need only very limited number of operations to be performed on a data to get the results. Let us think of some representation of persistence, of a type $\mathcal{A}$. To perform most of the statistical and machine learning operations one need to be able to compute average of objects of type $\mathcal{A}$ (so that the averaged object is also of a type $\mathcal{A}$). One need to be able to compute distance between objects of a type $\mathcal{A}$, and to compute scalar product of objects of a type $\mathcal{A}$.

To put this statement into a context, let us assume we have two collections $c_1, ..., c_n$ and $d_1, ..., d_n$ of objects of a type $\mathcal{A}$. We want to verify if the average of those two collections are different by performing bootstrap. First of all, we compute averages of those two collections: C = average of $c_1, ..., c_n$ and D = average of $d_1, ..., d_n$. Note that both C and D are of a type $\mathcal{A}$. Then we compute $d(C, D)$, a distance between C and D. Later we put the two collections into one bin:

$$B = c_1, ..., c_n, d_1, ..., d_n$$

Then we shuffle B, and we divide the shuffled version of B into two classes: B_1 and B_2 (in this case, of the same cardinality). Note that again, B_1 and B_2 are of a type $\mathcal{A}$. We compute their distance $d(B_1, B_2)$. The procedure of shuffling and dividing the set B is repeated N times (where N is reasonably large number). Then the p-value of a statement that the averages of $c_1, ..., c_n$ and $d_1, ..., d_n$ is approximated by the number of times $d(B_1, B_2) > d(C, D)$ divided by N.

As one can see, this procedure can be performed for any type $\mathcal{A}$ which can be averaged, and which allows for computations of distances. The idea of Gudhi_stat is to take advantage of C++ (and later python) polymorphism and implement a few interfaces:

1. Interface of a representation of persistence that allows averaging (so that the average object is of the same type).
2. Interface of representation of persistence that allows computations of distances.
3. Interface of representation of persistence that allows computations of scalar product.

And then to implement various currently known representations based on those interfaces. At the moment we have the implementation of following classes available (further details of those representations will be discussed later):

1. Exact persistence landscapes (allow averaging, computation of distances and scalar products).
2. Persistence landscapes on a grid (allow averaging, computation of distances and scalar products).
3. Persistence heat maps various representations where one put some weighted or not Gaussian kernel for each point of diagram (allow averaging, computation of distances and scalar products).

4. Persistence vectors (allow averaging, computation of distances and scalar products).

5. Persistence diagrams / barcodes (allow computation of distances).

The main aim of this implementation is to be able to implement various statistical methods, both on the level of C++ and python, that operates on the interfaces that are required for that particular method (like, in the example above, the ability of averaging and computations of distances). Given those implementations of statistical methods, we are able to use any of representation that implement athe required collection of interfaces (that includes the future ones that has not been implemented yet). Doing so, we define a computational framework that joins topological methods that uses persistent homology with statistics and machine learning. This framework is very easy to being extend by new representations of persistence, and even more general, but any new type of representation.

Below we are discussing the representations which are currently implemented in Gudhi_stat:

2 Persistence Landscapes

Persistence landscapes were originally proposed by Bubenik in [1]. Efficient algorithms to compute them rigorously were proposed by Bubenik and Dłotko in [2]. The idea of persistence landscapes is shortly summarized in below.

To begin with, suppose we are given a point $(b,d) \in \mathbb{R}^2$ in a persistence diagram. With this point, we associate a piecewise linear function $f_{(b,d)} : \mathbb{R} \to [0,\infty)$, which is defined as

$$f_{(b,d)}(x) = \begin{cases} 0 & \text{if} \quad x \notin (b,d) \ , \\ x-b & \text{if} \quad x \in \left(b, \frac{b+d}{2}\right] \ , \\ d-x & \text{if} \quad x \in \left(\frac{b+d}{2}, d\right) \ . \end{cases} \tag{1}$$

A *persistence landscape* of the birth-death pairs (b_i,d_i), where $i = 1,\ldots,m$, which constitute the given persistence diagram is the sequence of functions $\lambda_k : \mathbb{R} \to [0,\infty)$ for $k \in \mathbb{N}$, where $\lambda_k(x)$ denotes the k^{th} largest value of the numbers $f_{(b_i,d_i)}(x)$, for $i = 1,\ldots,m$, and we define $\lambda_k(x) = 0$ if $k > m$. Equivalently, this sequence of functions can be combined into a single function $L : \mathbb{N} \times \mathbb{R} \to [0,\infty)$ of two variables, if we define $L(k,t) = \lambda_k(t)$.

The detailed description of algorithms used to compute persistence landscapes can be found in [2]. Note that this implementation provides exact representation of landscapes. That have many advantages, but also a few drawbacks. For instance, as discussed in [2], the exact representation of landscape may be of quadratic size with respect to the input persistence diagram. It may therefore happen that, for very large diagrams, using this representation may be memory–prohibitive. In such a case, there are two possible ways to proceed:

1. Use non exact representation on a grid described in the Section 3.

2. Compute just a number of initial nonzero landscapes. This option is available from C++ level.

3 Persistence Landscapes on a grid

This is an alternative representation of persistence landscapes defined in the Section 2. Unlike in the Section 2 we build a non–exact representation by sampling persistence landscape on a finite, equally distributed grid of points. Since, the persistence landscapes that originate from persistence diagrams have slope 1 or -1, we have an estimate of an error made in between grid points by such a sampling. We do not have a similar estimation when a landscape is obtained from averaging process.

Due to a lack of rigorous description of the algorithms in the literature, we are giving a short discussion of them in below. Persistence landscapes are being represented by vector of vectors of real numbers. Assume that i-th vector consist of n_i numbers sorted from larger to smaller. They represent the values of the functions $\lambda_1, \ldots, \lambda_{n_i}$ (λ_{n_i+1} and further are zero) on the i-th point of a grid.

When averaging two persistence landscapes represented by a grid, we simply compute point-wise averages of the entries of corresponding vectors[1]

Computations of distances between two persistence landscapes on a grid is not much different than in the rigorous case. In this case, we sum up the distances between the same levels of corresponding landscapes. For fixed level, we approximate the landscapes between the corresponding constitutive points of landscapes by linear functions, and compute the L^p distance between them.

Similarly as in case of distance, when computing the scalar product of two persistence landscapes on a grid, we sum up the scalar products of corresponding levels of landscapes. For each level, we assume that the persistence landscape on a grid between two grid points is approximated by linear function. Therefore to compute scalar product of two corresponding levels of landscapes, we sum up the integrals of products of line segments for every pair of constitutive grid points.

Note that for this representation we need to specify a few parameters:

1. Begin and end point of a grid (real numbers)
2. Number of points in a grid.

Note that the same representation is used in TDA R-package [3].

4 Persistence heat maps

This is a general class of discrete structures which are based on idea of placing a Gaussian kernel in the points of persistence diagrams. This idea appeared in work by many authors along the last 15 years. As far as we know this idea was firstly described in the work of Bologna group in [4] and [5]. Later it has been described by Colorado State University group in [6]. The presented paper in the first time provide a discussion of stability of the representation. Also, the same ideas are used in construction of two recent kernels used for machine learning: [7] and [8]. Both the kernel's construction uses interesting ideas to ensure stability of the representation with respect to Wasserstein metric. In the kernel presented in [7], a scaling function is used to multiply the Gaussian kernel in the way that the points close to diagonal got low weight and consequently do not have a big influence on the resulting total distribution. In [8] for every point (b, d) two Gaussian

[1]In this whole section we assume that if one vector is shorter than another, we extend the shorter one with zeros so that they have the same length.

kernels are added. First, with a weight one in a point (b, d), and the second, with the weight -1 for a point (b, d). In both cases, the representations are stable with respect to 1-Wasserstein distance.

In Gudhi_stat we currently implement all the structures described above. The base of this implementation is 2-dimensional arrays of pixels. Each pixel have assigned a real value which is a sum of values of distributions induced by each point of the persistence diagram.

The parameters of the structure are as follows:

1. Size of the image (we always assume that the images are square).
2. A filter: in practice a square matrix of a size $2k + 1 \times 2k + 1$. By default, this is a Gaussian kernel, but any other can be used instead.
3. The box $[x_0, x_1] \times [y_0, y_1]$ bounding the domain of the persistence image.
4. Scaling function (each Gaussian kernel at point (p, q) gets multiplied by the value of this function at the point (p, q).
5. A boolean value determining if the space below diagonal should be erased or not.

5 Persistence vectors

This is a representation of persistent homology in a form of a vector which was designed for an application in 3d graphic in [9]. Below we provide a short description of this representation.

Given a persistence diagram $D = \{(b_i, d_i)\}$, for every pair of birth–death points (b_1, d_1) and (b_2, d_2) we compute the following three distances:

1. $d((b_1, d_1), (b_2, d_2))$.
2. $d((b_1, d_1), (\frac{b_1,d_1}{2}, \frac{b_1,d_1}{2}))$.
3. $d((b_2, d_2), (\frac{b_2,d_2}{2}, \frac{b_2,d_2}{2}))$.

We pick the smallest of those and add it to a vector. The obtained vector of numbers is then sorted in decreasing order. This way we obtain a *persistence vector* representing the diagram.

Given two persistence vectors, the computation of distances, averages and scalar products is straightforward. Average is simply a coordinate-wise average of a collection of vectors. In this section we assume that the vectors are extended by zeros if they are of a different size. To compute distances we compute absolute value of differences between coordinates. A scalar product is a sum of products of values at the corresponding positions of two vectors.

References

[1] P. Bubenik, Statistical topological data analysis using persistence landscapes, Journal of Machine Learning Research, 16 (2015), 77102.

[2] P. Bubenik, P. Dlotko, A persistence landscapes toolbox for topological statistics. Journal of Symbolic Computation.

[3] B. Fasy, J. Kim, F. Lecci, C. Maria, Introduction to the R package TDA, arXiv:1411.1830.

[4] P. Donatini, P. Frosini, A. Lovato, Size functions for signature recognition, Proceedings of SPIE, Vision Geometry VII, vol. 3454 (1998).

[5] M. Ferri, P. Frosini, A. Lovato, C. Zambelli, Point selection: A new comparison scheme for size functions (With an application to monogram recognition), Proceedings Third Asian Conference on Computer Vision, Lecture Notes in Computer Science 1351.

[6] H. Adams, S. Chepushtanova, T. Emerson, E. Hanson, M. Kirby, F. Motta, R. Neville, C. Peterson, P. Shipman, L. Ziegelmeier, Persistence Images: A Stable Vector Representation of Persistent Homology, arXiv:1507.06217.

[7] G. Kusano, K. Fukumizu, Y. Hiraoka, Persistence weighted Gaussian kernel for topological data analysis, arXiv:1601.01741.

[8] J. Reininghaus, S. Huber, U. Bauer, R. Kwitt, A Stable Multi-Scale Kernel for Topological Machine Learning, arXiv:1412.6821.

[9] M. Carrire, S. Oudot, M. Ovsjanikov, Stable Topological Signatures for Points on 3D Shapes., Proc. Sympos. on Geometry Processing, July 2015.