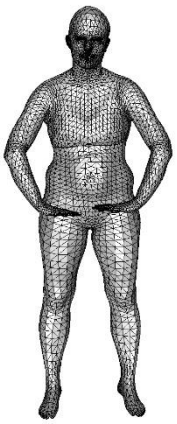
Topological Persistence for Data Clustering

Topological Data Analysis

Topological Data Analysis (TDA) is an emerging trend in exploratory data analysis and data mining. It has known a growing interest and some notable successes (such as the identification of a new type of breast cancer, or the classification of NBA players) in the recent years. Indeed, with the explosion in the amount and variety of available data, identifying, extracting and exploiting their underlying structure has become a problem of fundamental importance. Many such data come in the form of point clouds, sitting in potentially high-dimensional spaces, yet concentrated around low-dimensional geometric structures that need to be uncovered. The non-trivial topology of these structures is challenging for classical exploration techniques such as dimensionality reduction.

Introduction



In this project, we will present algorithmic solutions to extract meaningful topological features from point clouds. We only consider the feature extraction process, not the data clustering from said features which is studied in a future project. The data we will study are cloud of points which represent human bodies. The meshes will be converted into a filtration of simplexes. The evolution of the resulting homology groups will yield intervals of “births” and “deaths” that provide a topological identity of the point cloud (referred to as “barcode”). These barcodes can then be vectorized and used as features for data clustering.

The theory behind said barcodes is quite old (Ghrist, Robert) but it is only recently that new vectorizations techniques have been developed which significantly improve current state-of-the-art data clustering performances. My professor, Steve Oudot, is at the forefront of such developments in topological data analysis research. His works are available here : https://geometrica.saclay.inria.fr/team/Steve.Oudot/ .

We provide all codes in Python 3, requirements are indicated in the associated file. All files can be called independently from the command line by specifying the name of the file that should be processed, see the Pipeline section for details.

The pipeline

The analysis process is the following : point clouds are converted in simplicial filtrations which are then processed to compute the topological persistence intervals of the geometry behind the point clouds. These intervals are then vectorized to provide a descriptor which can be used in data clustering tasks.

1. Computing the lower star filtration

filtration_from_off.py converts the point clouds (in format .off) that are contained in the "off_files" directory into simplicial filtrations using the save_filtration function. This function takes as argument the complete name of the point cloud (tr_reg_000.off for instance) and writes the lower star filtration in the "filtrations" directory under the same name (tr_reg_000.txt for instance).

The lower star filtration is the following : vertices are assigned their height as value and every simplex is assigned the maximum value of its vertices' values. This procedure invariably yields a valid filtration : all simplices are inserted after all of its sub-simplices.

2. Computing the topological persistence intervals

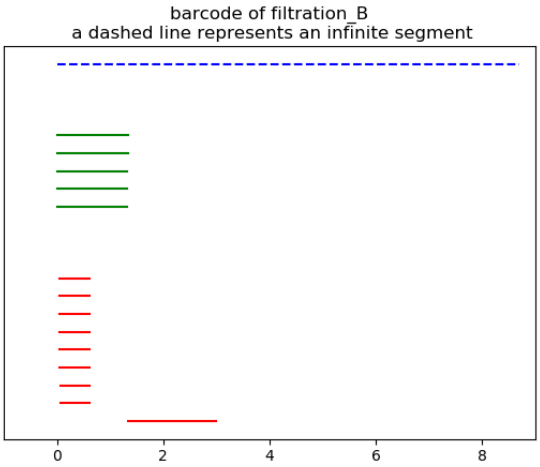
compute_barcode.py takes a filtration and computes the topological persistence intervals using the save_filtration function : the matrix of the boundary operator is calculated in sparse representation and then reduced with an efficient gaussian pivot; we use hash-maps to speed up the search of simplices. This function takes as argument the complete name of the filtration (tr_ger_000.txt for instance) and writes the resulting intervals in the "bar_codes" directory under the same name (tr_reg_000.txt for instance).

We have studied the performances of this key step on four different filtrations and observed a satisfactory linear complexity.

	Filtration A	Filtration B	Filtration C	Filtration D
Time	25.88 seconds	5.50 seconds	25.80 seconds	259.75 seconds
Size of filtration	428643	108161	180347	2716431
Ratio	$6\ e^{-5}$	$5\ e^{-5}$	$1.5\ e^{-4}$	$9\ e^{-5}$

3. (Optional) Plotting the barcodes

plot_barcode.py plots barcodes that have been computed. Its function print_bar_code only needs to be given the complete name of the barcode (tr_reg_000.txt for instance) and will produce a plot using matplotlib :



4. Extracting feature vectors from barcodes

extract_features.py finally takes a barcode and extracts topological features using the mapping function. Mapping takes a complete filtration name (it will actually look for a bare-code but since all name are kept equal we still refer to the names as filtration names) and two extra parameters n and d and returns the matrix feature vector. The exact method used is detailed in the annex 1.

Testing the pipeline

The main.py file has a compute-all-features function that iterates through all previously exposed steps (excluding the optional plotting of the barcodes) for all 100 off files. This is quite time-consuming so the results have already been computed and written in the features.txt file.

Full.py will perform both steps sequentially and rewrite the features.txt file in the process.

Conclusion

The proposed pipeline provides an efficient and comprehensive feature extraction process for .off point clouds and can be directly incorporated in any feature-based machine learning process to potentially increase performances.

Formats specifications

Off files

Structure of the file :

OFF
nv nf 0
x_0 z_0 y_0
...
x_ (nv-1) z_ (nv-1) y_ (nv-1)
3 i_0 j_0 k_0
...
3 i_ (nf-1) j_ (nf-1) k_ (nf-1)

where:

- OFF is the standard signature for .off files,
- nv and nf are respectively the numbers of vertices and triangles in the mesh,
- x_n z_n y_n are the coordinates of the nth vertex (beware that the z coordinate is second, not third),
- i_m j_m k_m are the IDs of the three vertices representing the mth triangle. For instance, 3 14 0 5 denotes the triangle that is given by the 14th, 0th and 5th vertices.

Filtrations

A “valid” filtration is in the format : f(sigma) dim(sigma) v˙0 ... v˙–dim(sigma)˝

where:

- f(sigma) is the function value of sigma (its ”time of appearance” in the filtration),
- dim(sigma) is the dimension of sigma,
- v˙0 ... v˙–dim(sigma)˝ are the IDs (integers) of the vertices of sigma.

Barcodes

A bar-code interval in the format : dimension of the homology group, the time of birth, the time of death (inf for an infinite interval)

Annex 1 : the feature extraction procedure

The procedure takes in a barcode and two parameters:

- d, which is the maximal desired homological dimension,
- n, which is the maximal desired number of barcode intervals.

Then, for each homology dimension k up to d, the procedure builds a feature vector as follows:

- First, it selects the n longest intervals in the subset of the barcode corresponding to dimension k, and it sets the first n coordinates of the feature vector to be half the lengths of these intervals, completing with zeros in case there are not enough intervals.

- Then, it selects the n(n-1)/2 highest entries in the upper triangle of the modified distance matrix (the one in which the entry (i,j) corresponds to the minimum of the distance between the i-th and j-th diagram points and their respective distances to the diagonal in the plane), and it sets them to be the next n(n-1)/2 coordinates of the feature vector, completing with zeros if necessary. Thus, the feature vector has n(n+1)/2 coordinates in total.

Finally, the feature vectors for all homological dimensions are concatenated into a single vector of total dimension (d+1)n(n+1)/2.

Bibliography

Ghrist, Robert (2007) : Barcodes : The persistent topology of data. In Bulletin of the American mathematical society. url: https://www.ams.org/journals/bull/2008-45-01/S0273-0979-07-01191-3/ last checked on 12/08/18.