An introduction to Topological Data Analysis with Gudhi
TP 2: The Vietoris-Rips and α-complex filtrations
MVA 2017-18

Frédéric Chazal

December 3, 2017

Abstract

To download this file and get the code samples (available on the course webpage soon):
`http://geometrica.saclay.inria.fr/team/Fred.Chazal/MVA2017.html`
Documentation for the Python interface of Gudhi:
`http://gudhi.gforge.inria.fr/python/latest/`

1 Basic instructions to build Vietoris-Rips and α-complex filtrations and compute their persistence

```
#Create a random point cloud in 3D
nb_pts =100
pt_cloud = np.random.rand(nb_pts,3)

#Built Rips-Vietoris filtration and compute its persistence diagram
rips_complex = gd.RipsComplex(pt_cloud,max_edge_length=0.5)
simplex_tree = rips_complex.create_simplex_tree(max_dimension=3)
print("Number of simplices in the V-R complex: ",simplex_tree.num_simplices())
diag = simplex_tree.persistence(homology_coeff_field=2, min_persistence=0)
gd.plot_persistence_diagram(diag)


#Compute Rips-Vietoris filtration and compute its persistence diagram from
#a pairwise distance matrix
dist_mat = []
for i in range(nb_pts):
    ld = []
    for j in range(i):
        ld.append(np.linalg.norm(pt_cloud[i,:]-pt_cloud[j,:]))
    dist_mat.append(ld)
rips_complex2 = gd.RipsComplex(distance_matrix=dist_mat,max_edge_length=0.5)
simplex_tree2 = rips_complex2.create_simplex_tree(max_dimension=3)
diag2 = simplex_tree2.persistence(homology_coeff_field=2, min_persistence=0)
gd.plot_persistence_diagram(diag2)

#Compute the alpha-complex filtration and compute its persistence
```

```
alpha_complex = gd.AlphaComplex(points=pt_cloud)
simplex_tree3 = alpha_complex.create_simplex_tree(max_alpha_square=60.0)
print("Number of simplices in the alpha-complex: ",simplex_tree3.num_simplices())
diag3 = simplex_tree3.persistence(homology_coeff_field=2, min_persistence=0)
gd.plot_persistence_diagram(diag3)
```

Exercise 1.
a) Illustrate the stability theorem for persistence diagrams of Vietoris-Rips and α-complex filtrations.
b) What does happen to Vietoris-Rips and α-complex filtrations when the size of the point cloud increases? When the ambient dimension increases?

2 Sensor data

Download the data at the following address and save it as a file named data_acc.dat:
`http://geometrica.saclay.inria.fr/team/Fred.Chazal/slides/data_acc.dat`
and load it using the pickle module:

```
import numpy as np
import pickle as pickle
import gudhi as gd
from mpl_toolkits.mplot3d import Axes3D

f = open("data_acc.dat","rb")
data = pickle.load(f,encoding="latin1")
f.close()

data_A = data[0]
data_B = data[1]
data_C = data[2]
label = data[3]
```

The walk of 3 persons A, B and C, has been recorded using the accelerometer sensor of a smartphone in their pocket, giving rise to 3 multivariate time series in $\mathbb{R}^3$: each time series represents the 3 coordinates of the acceleration of the corresponding person in a coordinate system attached to the sensor (take care that as, the smartphone was carried in a possibly different position for each person, these time series cannot be compared coordinates by coordinates). Using a sliding window, each serie have been splitted in a list of 100 times series made of 200 consecutive points, that are now stored in data_A, data_B and data_C.

1. Plot a few of the time series to get an idea of the corresponding point clouds in $\mathbb{R}^3$. For example:

```
data_A_sample = data_A[0]
plt.gca(projection='3d')
plt.plot(data_A_sample [:,0],data_A_sample [:,1],data_A_sample [:,2] )
```

2. Compute and plot the persistence diagrams of the Vietoris-Rips and the alpha-complex filtrations, for a few examples of the time series.

3. Compute the 0-dimensional and 1-dimensional persistence diagrams (α-shape or Rips-Vietoris filtration) of all the time series. Compute the matrix of pairwise distances between the diagrams (as this is may take a while, you can just select a subset of all the diagrams where each of the 3 classes A, B and C are represented). Visualize the pairwise distances via Multidimensional Scaling (use a different color for each class). You can use sklearn for that:

```
from sklearn import manifold

# B is the pairwise distance matrix between 0 or 1-dim dgms
#label_color contains the colors corresponding to the class of each dgm
mds = manifold.MDS(n_components=3, max_iter=3000, eps=1e-9,
                   dissimilarity="precomputed", n_jobs=1)
pos1 = mds.fit(B1).embedding_
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(pos1[:,0], pos1[:, 1], pos1[:,2], marker = 'o', color=label_color)
```

4. Use the function below to embed the data in dimension $3 \times 3 = 9$ with a delay equal to 2 (time-delay embedding) and do the same experiments as previously, using the Vietoris-Rips filtration this time.

```
def sliding_window_data(x,edim,delay=1):
    """time delay embedding of a d-dim times series into R^{d*edim}
    the time series is assumed to be periodic
    parameters:
        + x: a list of d lists of same length L or a dxL numpy array
        + edim: the number of points taken to build the embedding in
          R^{d*edim}
        + delay: embeeding given by (x[i],x[i+delay],...,
          x[i + (edim-1)*delay])
          Default value for delay is 1
    """
    ts = np.asarray(x)
    if len(np.shape(ts)) == 1:
        ts = np.reshape(ts,(1,ts.shape[0]))
    ts_d = ts.shape[0]
    ts_length = ts.shape[1]
    #output = zeros((edim*ts_d,nb_pt))
    output = ts
    for i in range(edim-1):
        output = np.concatenate((output,np.roll(ts,-(i+1)*delay,axis=1)),axis=0)
    return output
```