Intro_TDA_with_GUDHI_Part2

December 2, 2018

0.1 MVA 2018-19

To download this notebook or its pdf version:
http://geometrica.saclay.inria.fr/team/Fred.Chazal/MVA2018.html
Documentation for the latest version of Gudhi:
http://gudhi.gforge.inria.fr/python/latest/

1 Sensor data

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

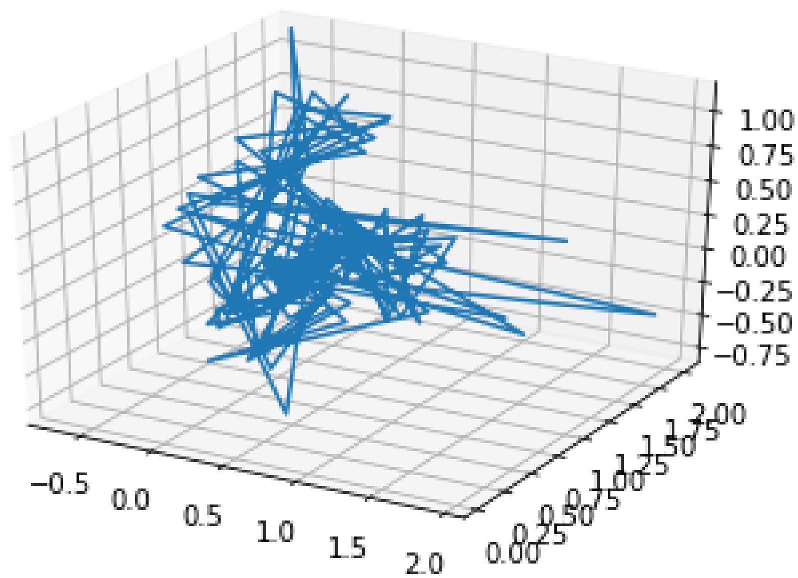
```
In [4]: import numpy as np
        import pickle as pickle
        import gudhi as gd
        import matplotlib.pyplot as plt
        from mpl_toolkits.mplot3d import Axes3D
        from sklearn import manifold
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import confusion_matrix
        from sklearn.cluster import KMeans

        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]

        %matplotlib inline
```

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 series has been split in a list of 100 time series made of 200 consecutive points, that are now stored in data_A, data_B and data_C.

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

```
In [5]: data_A_sample = data_A[0]
        plt.gca(projection='3d')
        plt.plot(data_A_sample [:,0],data_A_sample [:,1],data_A_sample [:,2])
```

```
Out[5]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x21330e82860>]
```



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

- Compute the 0-dimensional and 1-dimensional persistence diagrams (-shape or Rips-Vietoris filtration) of all the time series. Compute the matrix of pairwise distances between the diagrams (as this 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 Multi-dimensional Scaling (use a different color for each class). You can use sklearn for that:

```
In [6]: # 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"
        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)
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-6-9c995588e20e> in <module>()
  2 #label_color contains the colors corresponding to the class of each dgm
  3 mds = manifold.MDS(n_components=3, max_iter=3000, eps=1e-9, dissimilarity="precompu
----> 4 pos1 = mds.fit(B1).embedding_
  5 fig = plt.figure()
  6 ax = fig.add_subplot(111, projection='3d')


NameError: name 'B1' is not defined
```

- Use the function below to embed the data in dimension 3Œ3 = 9 with a delay equal to 2 (time-delay embedding) and do the same experiments as previously, using the Vietoris-Rips filtration this time.

In [ ]:
```
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
```

2 Persistence landscapes

Landscape construction is currently only available in the C++ version of Gudhi. Here is a simple python implementation you can use for this class.

In [ ]:
```
def landscapes_approx(diag,p_dim,x_min,x_max,nb_nodes,nb_ld):
    """Compute a dicretization of the first nb_ld landscape of a
    p_dim-dimensional persistence diagram on a regular grid on the
```

```
    interval [x_min,x_max]. The output is a nb_ld x nb_nodes numpy
    array
    + diag: a persistence diagram (in the Gudhi format)
    + p_dim: the dimension in homology to consider
    """
    landscape = np.zeros((nb_ld,nb_nodes))
    diag_dim = []
    for pair in diag: #get persistence points for homology in dimension dim
        if (pair[0] == p_dim):
            diag_dim.append(pair[1])

    step = (x_max - x_min) / (nb_nodes - 1)
    #Warning: naive and not the most efficient way to proceed!!!!!
    for i in range(nb_nodes):
        x = x_min + i * step
        t = x / np.sqrt(2)
        event_list = []
        for pair in diag_dim:
            b = pair[0]
            d = pair[1]
            if b <= t <= d:
                if t >= (d+b)/2:
                    event_list.append((d-t)*np.sqrt(2))
                else:
                    event_list.append((t-b)*np.sqrt(2))
        event_list.sort(reverse=True)
        event_list = np.asarray(event_list)
        for j in range(nb_ld):
            if(j<len(event_list)):
                landscape[j,i]=event_list[j]

    return landscape
```

- Test the function on a few examples of diagrams and plot the resulting landscapes.
- Compute and store the persistence landscapes of the accelerometer time series. Use the obtained landscapes to experiment with supervised and non supervised classification on this data.

In [ ]:
```
# Example of parameters, you don't have to use those
nb_ld = 5 # number of Landscapes
nb_nodes = 500
length_max = 1.0
```

3 Bootstrap and confidence bands for lanscapes

The goal of this exercise is to implement the bootstrap algorithm below from [F. Chazal, B.T. Fasy, F. Lecci, A. Rinaldo, L. Wasserman. *Stochastic Convergence of Persistence Landscapes and Silhouettes.*

in Journal of Computational Geometry, 6(2), 140-161, 2015] to compute confidence bands for landscapes. As an example compute confidence bands for the expected landscapes for each of the 3 classes in the accelerometer data set.

3.1 The multiplier bootstrap algorithm.

Input: landscapes $\lambda_1, \ldots, \lambda_n$; confidence level $1-\alpha$; number of bootstrap samples B
Output: confidence functions $\ell_n, u_n \colon \mathbb{R} \to \mathbb{R}$ 1. Compute the average $\overline{\lambda}_n(t) = \frac{1}{n}\sum_{i=1}^n \lambda_i(t)$, for all t 1. For $j = 1$ to B: 1. Generate $\xi_1, \ldots, \xi_n \sim N(0,1)$ 1. Set $\tilde{\theta}_j = \sup_t n^{-1/2} |\sum_{i=1}^n \xi_i \, (\lambda_i(t) - \overline{\lambda}_n(t))|$ 1. End for 1. Define $\tilde{Z}(\alpha) = \inf\{z : \frac{1}{B}\sum_{j=1}^B I(\tilde{\theta}_j > z) \leq \alpha\}$ 1. Set $\ell_n(t) = \overline{\lambda}_n(t) - \frac{\tilde{Z}(\alpha)}{\sqrt{n}}$ and $u_n(t) = \overline{\lambda}_n(t) + \frac{\tilde{Z}(\alpha)}{\sqrt{n}}$ 1. Return $\ell_n(t), u_n(t)$