Topological Data Analysis Python, scikit-tda

1. Ripser package (ripser.py grinds to a halt even on H1 if the number of points exceeds around 1000.)

   Installation

   Pip install cython
   Pip install ripser

   Ripser is used for 1. computing persistence cohomology of sparse and dense data sets, 2. visualizing persistence diagrams, 3. computing lowerstar filtrations on images, and 4. computing representative cochains.

   I. ripser.Rips - sklearn interface
      A. rips = Rips(maxdim=n, thresh=m,n_perm=k , verbose=True)
         1. n_perm is for subsampling from the data cloud
         2. Verbose is to print the data as constructed
      B. rips.fit_transform(data, distance_matrix = False, metric = 'cosine') where data.shape = (N,2)
      C. Set the metric by rips.metric_ = 'euclidean', 'manhattan' or 'cosine'. Alternatively, if metric is a callable function, it is called on each pair of instances (rows) and the resulting value recorded. The callable should take two arrays from X as input and return a value indicating the distance between them
      D. rips.dperm2all_, rips.num_edges_, rips.idx_perm_, rips.r_cover_:
         1. dperm2all_ : the distance matrix used if n_perm is used. Otherwise, the distance from all points in the permutation to all points in the dataset
         2. num_edges_ = number of edges added during the computation
         3. idx_perm_ = index of the subsample in the original point cloud.
         4. r_cover = covering radius of the subsample

   II. ripser.ripser -
      A. Dgms = ripser(data, maxdim = n, thresh = m, distance_matrix = True, metric = 'cosine', n_perm = k)['dgms']
         1. Maxdim computes the n-dim homology persistent diagram

2. Thresh is the threshold value of the radius for rips filtration (VR complex)

B. Same methods as Rips, however, you need to consider 'dgms' column.

III. Plotting options:

1. plot_diagrams(dgms, xy_range = [-1,20,-5,15], colormap = 'seaborn', lifetime = True)
   xy_range is the range of both x and y axes.
   Colormap is the usual colormap from matplotlib
   Lifetime - Boolean value
2. plot(diagrams, plot_only = [0], title='asf', xy_range = [-1,20,-5,15], colormap = 'seaborn', lifetime = True, labels = 'x', size = 20)
   Plot_only plots certain subsection(particular homology class) of the diagram
   Diagrams = rips.fit(data) or rips.fit_transform(data)
   labels : to replace H0, H1 with something else
   Size is the size of pixel of each plotted point.

ApproximateSparse Filtrations

2000 data points. When you create a sparse filter and use it as the distance matrix, the computation is much faster.

```
D = pairwise_distances(X, metric='euclidean')
lambdas = getGreedyPerm(D) #list of insertion radii
DSparse = getApproxSparseDM(lambdas, 0.1, D)
resultsparse = ripser(DSparse, distance_matrix=True)
```

Time Series visualization

For a time series model, with time discrete, the persistence diagram says about the maxima and minima of the time series model. But we need to use the sparse matrix. Even with noisy data with some trend, maxima and minima, we can get the actual points and the trend.

Lower Star Image Filtration

One can find the local minimum and maximum of the data cloud in an image. It's very helpful in understanding the critical points of the image.

ripser.lower_star_img(image): returns `ripser(sparseDM, distance_matrix=True, maxdim=0)["dgms"][0]`

2. Persim Package

I. persim.bottleneck
   A. bottleneck(diagram1,diagram2,matching = False)
      1. Performs bottleneck distance matching between persistent diagrams. Tells which points correspond to the bottleneck distance.
      2. It can be used to see the distance between a noisy structure and a clear structure.
   B. Returns distance, (matching,D) if matching=True Otherwise returns just distance.

II. persim.sliced_wasserstein
   A. sliced_wasserstein(persistentdiag1, persdiag2,M=n)
      1. It tries to approximate wasserstein distance by computing the distance in 1 dimension repeatedly (n iterations in this case).
      2. Wasserstein distance is used to measure the distance between probability distribution on a metric space.
   B. Returns sliced wasserstein distance.

III. persim.heat
   A.

IV. persim.gromov_hausdorff

V. persim.PersImage

VI. persim.plot_diagrams

VII. persim.bottleneck_matching

VIII. persim.wasserstein_matching

IX. persim.persistent_entropy