TP APPC : Clustering

Stéphane Canu

27 Novembre 2023, ITI, INSA Rouen

The purpose of this notebook is compare different clustering algorithms on four different toy data sets to determine

- what are the best ones ?
- what are the scalable ones ?

The code is adapted from :

- `https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html`
- `https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/`
- `https://saskeli.github.io/data-analysis-with-python-summer-2019/clustering.html`.

FIGURE 1 : Raw data and example of expected results.

Ex. 1 — Clustering

1. Data preparation.
   a) Load some relevant API

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn import cluster, datasets, mixture
```

   b) Generate some toy data and explain why they could be relevant. Generate your own fifth data set.

```
np.random.seed(42)

# mixture of 4 gaussian
clust1 = np.random.normal(5, 2, (500,2))
clust2 = np.random.normal(15, 3, (1000,2))
clust3 = np.random.multivariate_normal([17,3], [[1,0],[0,1]], 500)
clust4 = np.random.multivariate_normal([9,16], [[1,0],[0,1]], 1000)
X1 = np.concatenate((clust1, clust2, clust3, clust4))
plt.figure(figsize=(9 * 2 + 3, 5))
plt.subplot(1,4,1)
plt.plot(X1[:, 0],X1[:, 1],'o')

# 2 circles
plt.subplot(1,3,2)
n_samples = 1500
noisy_circles = datasets.make_circles(n_samples=n_samples, factor=.5, noise=.05)
X2,y2 = noisy_circles
plt.subplot(1,4,2)
plt.plot(X2[:, 0],X2[:, 1],'o')

# 2 moons + noise
noisy_moons = datasets.make_moons(n_samples=n_samples, noise=.05)
```

```
X3,y3 = noisy_moons
Xn = np.random.uniform(-1.25,2.,(30,2))
X3 = np.concatenate((X3, Xn))
plt.subplot(1,4,3)
plt.plot(X3[:, 0],X3[:, 1],'o')

# Anisotropicly distributed data
random_state = 170
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X4 = np.dot(X, transformation)
aniso = (X4, y)
plt.subplot(1,4,4)
plt.plot(X4[:, 0],X4[:, 1],'o')
plt.show()
```

2. the k-means
   a) begin with the k-means

```
kmeans_1 = cluster.KMeans(n_clusters=4).fit_predict(X1)
kmeans_2 = cluster.KMeans(n_clusters=2).fit_predict(X2)
kmeans_3 = cluster.KMeans(n_clusters=2).fit_predict(X3)
kmeans_4 = cluster.KMeans(n_clusters=3).fit_predict(X4)

plt.figure(figsize=(9 * 2 + 3, 5))
plt.subplot(1,4,1)
plt.scatter(X1[:, 0],X1[:, 1], s = 1.5, c=kmeans_1, cmap='rainbow')
plt.subplot(1,4,2)
plt.scatter(X2[:, 0],X2[:, 1], s = 1.5, c=kmeans_2, cmap='rainbow')
plt.subplot(1,4,3)
plt.scatter(X3[:, 0],X3[:, 1], s = 1.5, c=kmeans_3, cmap='rainbow')
plt.subplot(1,4,4)
plt.scatter(X4[:, 0],X4[:, 1], s = 1.5, c=kmeans_4, cmap='rainbow')
plt.show()
```

   b) factorize the code.
   Managing lists : what is the effect of the ”enumerate” an ”zip” instructions ?
   How to use them to factorize the code above ?

```
lstdata=[X1,X2,X3,X4]
lst_cluster = [4, 2, 2, 3]

kmeans = [cluster.KMeans(n_clusters=nb).fit_predict(X) for X,nb in zip(lstdata,
    lst_cluster)]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdata,kmeans)):
  plt.subplot(1,4,1+i)
  plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

3. Try the clustering with a mixture of 3 Gaussian together with Expectation Maximisation (EM) algorithm.

```
gauss_cluster = [mixture.GaussianMixture(n_components=nb, covariance_type='full').
    fit_predict(X) for X,nb in zip(lstdata,lst_cluster)]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdata,gauss_cluster)):
  plt.subplot(1,4,1+i)
  plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

4. Hierarchical Clustering

```
from sklearn.neighbors import kneighbors_graph

# connectivity matrix for structured Ward
c1 = kneighbors_graph(X1, n_neighbors=2, include_self=False)
c2 = kneighbors_graph(X2, n_neighbors=20, include_self=False)
c3 = kneighbors_graph(X3, n_neighbors=2, include_self=False)
c4 = kneighbors_graph(X4, n_neighbors=2, include_self=False)

# make connectivity symmetric
#c1 = 0.5 * (c1 + c1.T)
#c2 = 0.5 * (c2 + c2.T)
#c3 = 0.5 * (c3 + c3.T)
#c4 = 0.5 * (c4 + c4.T)
connectivity = [c1,c2,c3,c4]

agglo_cluster = [cluster.AgglomerativeClustering(linkage="average", affinity="euclidean
    ",
n_clusters=nb, connectivity=connec).fit_predict(X) for X,nb,connec in zip(lstdata,
    lst_cluster,connectivity)]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdata,agglo_cluster)):
  plt.subplot(1,4,1+i)
  plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

a) visualize the hierarchy graph for the first data set

```
import scipy.cluster.hierarchy as sch

# create dendrogram
plt.figure(figsize=(9 * 2 + 3, 5))
dendrogram = sch.dendrogram(sch.linkage(X1, method='centroid'))
plt.show()
```

5. try now with Affinity Propagation [1].

```
from sklearn.preprocessing import StandardScaler

lstdataN = [StandardScaler().fit_transform(X) for X in lstdata]
affinity_cluster = [cluster.AffinityPropagation(damping=.9, preference=-200).fit_predict
    (X) for X in lstdataN]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdataN,affinity_cluster)):
  plt.subplot(1,4,1+i)
  plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

6. Spectral clustering

```
spectral_cluster = [cluster.SpectralClustering(n_clusters=nb, eigen_solver='arpack',
affinity="nearest_neighbors").fit_predict(X) for X,nb in zip(lstdata,lst_cluster)]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdata,spectral_cluster)):
  plt.subplot(1,4,1+i)
  plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

[1]

7. try now with Density-based spatial clustering of applications with noise (DBSCAN)[2]. DBSCAN is a density based clustering algorithm that can neatly handle noise (the clue is in the name). Clusters are considered zones that are sufficiently dense. DBSCAN identifies clusters and then expands those clusters by scanning the neighbourhoods of the assigned points. The user must define :
   - the minimum number of observations (min_samples=5) that form a cluster
   - the maximum distance between two samples (eps) for one to be considered as in the neighborhood of the other.

```
dbscan_cluster = [cluster.DBSCAN(eps=.15).fit_predict(X) for X in lstdataN]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdataN,dbscan_cluster)):
  plt.subplot(1,4,1+i)
  plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

   a) try now with tuned parameter. What is the improvement ?

```
list_eps = [1., .1, .1, .3]
dbscan_cluster = [cluster.DBSCAN(eps=val_eps).fit_predict(X) for X,val_eps in zip(
    lstdata,list_eps)]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdata,dbscan_cluster)):
plt.subplot(1,4,1+i)
plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

8. Ordering points to identify the clustering structure (Optics)[3].

```
dbscan_cluster = [cluster.OPTICS(min_samples=20,xi=0.05,min_cluster_size=0.1).
    fit_predict(X) for X in lstdata]

plt.figure(figsize=(9 * 2 + 3, 5))
for i,(X,labels) in enumerate(zip(lstdata,dbscan_cluster)):
  plt.subplot(1,4,1+i)
  plt.scatter(X[:, 0],X[:, 1], s = 5, c=labels, cmap='rainbow')
plt.show()
```

9. HDBSCAN [4]
   you may need the install hdbscan first by using
   `pip install hdbscan`

```
import hdbscan
hdbscan_cluster1 = hdbscan.HDBSCAN(min_cluster_size=15, gen_min_span_tree=True).fit(X1)
hdbscan_cluster2 = hdbscan.HDBSCAN(min_cluster_size=5, gen_min_span_tree=True).fit(X2)
hdbscan_cluster3 = hdbscan.HDBSCAN(min_cluster_size=5, gen_min_span_tree=True).fit(X3)
hdbscan_cluster4 = hdbscan.HDBSCAN(min_cluster_size=25, gen_min_span_tree=True).fit(X4)

plt.figure(figsize=(9 * 2 + 3, 5))
plt.subplot(1,4,1)
plt.scatter(X1[:, 0],X1[:, 1], s = 5, c=hdbscan_cluster1.labels_, cmap='rainbow')
plt.subplot(1,4,2)
plt.scatter(X2[:, 0],X2[:, 1], s = 5, c=hdbscan_cluster2.labels_, cmap='rainbow')
plt.subplot(1,4,3)
plt.scatter(X3[:, 0],X3[:, 1], s = 5, c=hdbscan_cluster3.labels_, cmap='rainbow')
plt.subplot(1,4,4)
plt.scatter(X4[:, 0],X4[:, 1], s = 5, c=hdbscan_cluster4.labels_, cmap='rainbow')
plt.show()
```

[2] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
[3]
[4] https://hdbscan.readthedocs.io/en/latest/

a) Visualize the minimum spanning tree

```
hdbscan_cluster3.minimum_spanning_tree_.plot(edge_cmap='viridis',
                      edge_alpha=0.6,
                      node_size=80,
                      edge_linewidth=2
                      )
plt.show()
```

10. **Your turn :**
    a) What are the list of hyperparameter of each of these methods ?
    What are their influences ?
    b) find out another clustering method
    c) According to your experience, which is the best method and why ?
    d) Try to compare these methods on the MNIST data set on a pipline together with a dimensionality reduction method by doing
        1. load the MNIST data
        2. preprocess the data
        3. perform dimensionality reduction
        4. cluster the data in the low dimensional space