Q3.The power of the graph perspective in clustering

Dependencies

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

from scipy.linalg import svd
from scipy.spatial import distance
from sklearn.preprocessing import normalize
```

Helper Functions

In [ ]:
```
def show_data_results(X, num_plot=2, y_pred=None, cmap='jet'):
    """
    Plots the input data and additionally predicted clusters if provided.
    Set num_plot=1 to plot only input.
    Set num_plot=2 along with a y_pred to plot predicted clusters.
    """
    if num_plot==1:
        plt.scatter(X[:, 0], X[:, 1])
        plt.title ("input data")
    elif num_plot==2:
        try:
            assert y_pred is not None
            fig = plt.figure(figsize=(10, 3))
            ax1 = fig.add_subplot(121)
            ax1.scatter(X[:, 0], X[:, 1])
            ax1.set(xticks=[],yticks=[],title ="input data")

            ax2 = fig.add_subplot(122)
            ax2.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=cmap)
            ax2.set(xticks=[], yticks=[], title ="clustered data")
        except:
            print('y_pred is required for 2 plots')
```

K-Means Clustering

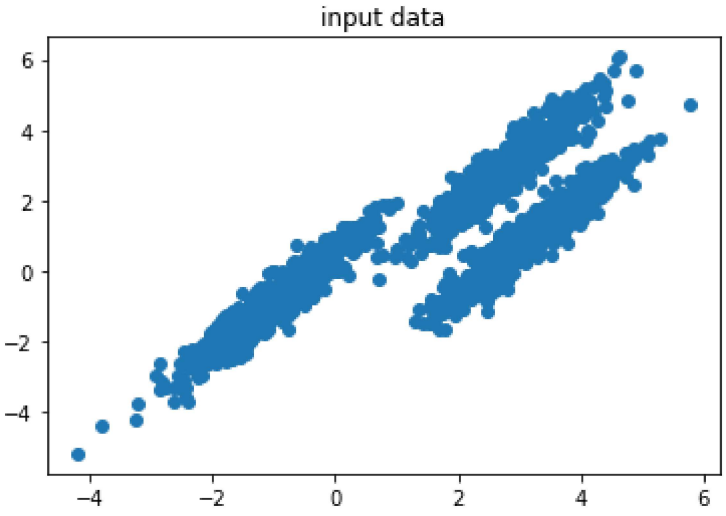
Please read https://en.wikipedia.org/wiki/K-means_clustering (https://en.wikipedia.org/wiki/K-means_clustering) about the Kmeans algorithm.

In this problem, we will show how interperting a dataset as a graph may result is obtaining an elegant clustering solution. We have an input dataset that we wish to cluster in 3 apparent classes.

We provide the synthetic dataset of 2000 points described below where the T_matrix is just a 2D transformation matrix:

In [ ]:
```
T_matrix, seed = [[-0.60834549, -0.63667341], [0.40887718, 0.85253229]], 170
X_orig, y_orig = make_blobs(n_samples=2000, random_state=seed)
X = np.dot(X_orig, T_matrix)
```

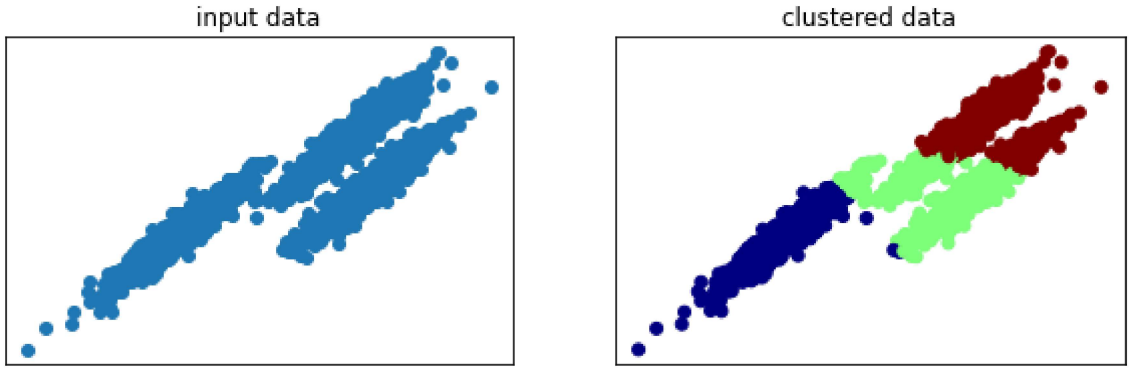
In [ ]:
```
show_data_results(X, num_plot=1)
```



Let's use the kmeans algorithm implementation of sklearn, to cluster this dataset into 3 classes in one line of code.

In [ ]:
```
y_pred = KMeans(n_clusters=3, random_state=seed).fit_predict(X)
```

In [ ]:
```
show_data_results(X, 2, y_pred)
```



Q.1

Comment on the output the the KMeans algorithm? Did it work? If so explain why, if not, explain why not.

Solution

The dataset has been clustered into 3 classes. It works since those data points that are closed to each other are classified into the same class.

Q.2.

Let's now interprete every single point in the provided dataset as a node in a graph. Our goal is to find a way to relate every node in the graph is such way that they points that closer together maintain that relationship while points that are far are explicitely identified.

lots of points points are closed top each other and kmaeans is missing it. representing as the graph unveils the relation.

One way to capture such relationship between points (nodes) in a graph is through the Adjacency matrix. Typically, a simple adjacency matrix between nodes of and indiredted graph is given by:

$$A_{i,j} = \begin{cases} 1 : \text{if there is an edge between node i and node j,} \\ 0 : \text{otherwise.} \end{cases}$$

In this probem, we will use the weighted distances betweeen points instead as a similary measure. Write a function that takes in the input dataset and some coeficient gamma which returns the adjacency matrix A.

$$A_{i,j} = e^{-\gamma ||x_i - x_j||^2}$$

where x_i and x_j represent each point in the provided dataset. You may find the $dictance$ module from $scipy.spatial$ useful.

Solution

In [ ]:
```
def get_adjacency_matrix(gamma, X):
    # TODO - fill in your code here
    # print(X.shape)
    adjacency_matrix = np.exp(-gamma * distance.cdist(X, X, metric='sqeuclidean')).res
    return adjacency_matrix
```

Q. 3.

The degree matrix of an undirected graph is a diagonal matrix which contains information about the degree of each vertex. In other word, it contains the number of edges attached to each vertex and it is given by:

$$D_{i,j} = \begin{cases} deg(v_i) : \text{if i} == \text{j}, \\ 0 : \text{otherwise.} \end{cases}$$

where the degree $\deg(v_i)$ of a vertex counts the number of times an edge terminates at that vertex. Note that in the traditional definition of the adjacency matrix, this boils down to the diagonal matrix in which element along the diagonals are column-wise sum of the adjacency matrix. Using the same idea, write a function that takes in the adjacency matrix as argument and returns the inverse square root of degree matrix.

Solution

In [ ]:
```
def get_degree_matrix(adjacency_matrix):
    # TODO - fill in your code here
    # print(adjacency_matrix.shape)
    degree_matrix = np.diag(np.sum(adjacency_matrix, axis=0))
    return degree_matrix
```

Type *Markdown* and LaTeX: α^2

Q. 4.

Using γ = 7.5, compute the symmetrically normalized adjacency matrix A, degree matrix D and the matrix $M = D^{-1/2}AD^{-1/2}$

Solution

In [ ]:
```
adjacency_matrix = get_adjacency_matrix(7.5, X)
degree_matrix = get_degree_matrix(adjacency_matrix)
tmp = np.linalg.inv(np.sqrt(degree_matrix))
M = tmp @ adjacency_matrix @ tmp
```

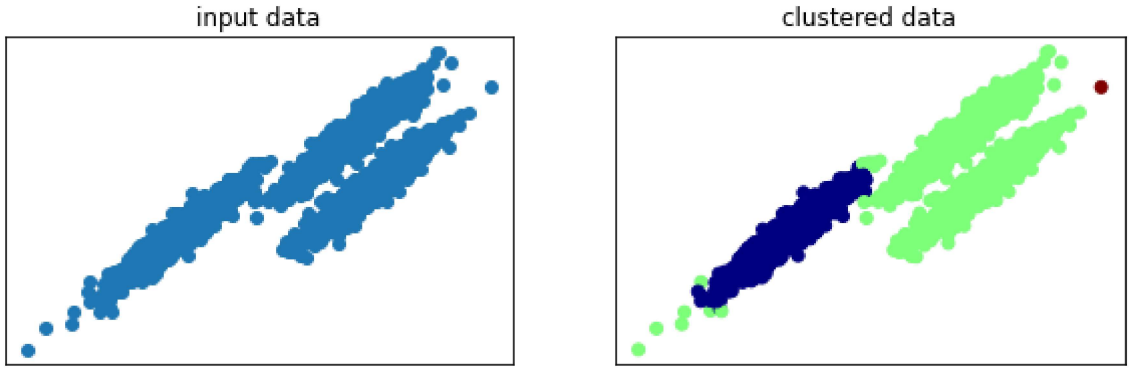
Q. 5.

Applying SVD decomposition on M, select the first 3 vectors in the matrix U and perform the same KMeans clustering used above on them. What do you observe? Did it work? If so explain why, if not, explain not not.

Solution

In [ ]:
```
u, s, vh = np.linalg.svd(M)
```

In [ ]:
```
y_pred_svd = KMeans(n_clusters=3, random_state=seed).fit_predict(u[:, :3])
show_data_results(X, 2, y_pred_svd)
```



It does not work, because the amount of data points in each cluster varies greatly, unable to minimize the mean.

Q.6.

Now let's think of the symmetrically normalized adjacency matrix obtained above as the transition matrix in of a Markov Chain. That is, it represents the probability of jumping from one node to another. In order to fully interpret M in such way, A needs to be a proper stochastic matrix which means that the sum of the elements in each column must add up to 1.

Write a function that takes in the matrix M and returns M_stochastic, the stochastic version of M. Use this to compute the stochastic matrix

Solution

In [ ]:
```
def stochastic_matrix_converter(M):
    # TODO - fill in your code here
    M_stoch = M / np.sum(M, axis=0)
    return M_stoch
```

In [ ]:
```
M_stoch = stochastic_matrix_converter(M)
```

Using the method from Q.5 above, compute the top 3 U vectors on `M_stoch` , perform K Means clustering on it and plot your answers. What do you observe?

In [ ]:
```
u, s, vh = np.linalg.svd(M_stoch)
```

In [ ]:
```
# TODO - compute y_pred_spectral_stoch below
y_pred_spectral_stoch = KMeans(n_clusters=3, random_state=seed).fit_predict(u[:, :3])
show_data_results(X, 2, y_pred_spectral_stoch)
```