Lab 11 Solution - Extended Exercises on Time Series Clustering

You are the Senior Data Scientist in a learning platform called LernTime. Your data science team built a data frame in which each row contains the aggregated features per student (calculated over the first 5 weeks of interactions) and the feature `dropout` indicates whether the student stopped using the platform (1) or not (0) before week 10.

The dataframe is in the file `lerntime.csv` and contains the following features:

- `video_time`: total video time (in minutes)
- `num_sessions` total number of sessions
- `num_quizzes`: total number of quizzes attempts
- `reading_time`: total theory reading time
- `previous_knowledge`: standardized previous knowledge
- `browser_speed`: standardized browser speed
- `device`: whether the student logged in using a smartphone (1) or a computer (-1)
- `topics`: the topics covered by the user
- `education`: current level of education (0: middle school, 1: high school, 2: bachelor, 3: master, 4: Ph.D.).
- `dropout`: whether the student stopped using the platform (1) or not (0) before week 5.

```python
import pandas as pd

import numpy as np
from scipy import linalg

from sklearn.metrics import silhouette_score
from sklearn.neighbors import kneighbors_graph
from sklearn.metrics.pairwise import pairwise_kernels
from sklearn.manifold import spectral_embedding
from scipy.sparse.csgraph import laplacian
from sklearn.cluster import KMeans
from scipy.spatial.distance import pdist, squareform

# Data directory
DATA_DIR = "./../../data/"

df = pd.read_csv(f'{DATA_DIR}/lerntime_dropout.csv')

df.head()
```

```
   video_time  num_sessions  num_quizzes  reading_time
previous_knowledge  \
0   45.793303          99.0         36.0     48.186562
1.675972
1   51.331242          57.0         12.0     49.945810
```

```
0.700522
2   87.414834          52.0          7.0     20.611978
1.836716
3   58.556388          47.0         31.0     33.785805
0.209577
4   74.822362          58.0         37.0     38.907983
0.265678

   browser_speed  device
topics  \
0      -0.294704     1.0  ['Locke', 'Descartes', 'Socrates', 'Kant',
'Ni...
1       1.253694     1.0  ['Nietzche', 'Locke', 'Confucius',
'Aristotle'...
2      -1.171352     1.0  ['Plato', 'Locke', 'Nietzche', 'Socrates',
'De...
3      -2.043047     1.0  ['Aristotle', 'Socrates', 'Plato',
'Confucius'...
4      -0.754559     1.0  ['Kant', 'Aristotle', 'Confucius', 'Locke',
'P...

   education  dropout
0        2.0        0
1        3.0        0
2        4.0        0
3        3.0        0
4        4.0        0
```

You decide to explore the different type of users. You want to use your knowledge from your ML4BD course and decide to cluster using Spectral Clustering. In the course, you learnt different ways of constructing the similarity graph, yielding the adjacency matrix serving as an input to the Spectral Clustering. Based on your in-depth exploration of the data, you decide to construct the similarity graph as a *k-nearest neighbor graph*.

Your tasks are to:

a) Write a function to compute the k-nearest neighbor graph.

b) Cluster the users using Spectral Clustering and your k-nearest neighbor graph function (use 4 neighbors). Use only the features *reading_time* and *topics*. You can assume that optimal number of clusters is 2.

a) Computation of the k-nearest neighbor graph

Unfortunately, there is no k-nearest neighbor graph implementation available in scikit-learn and you therefore have to implement the function yourself.

The function `'k_nearest_neighbor_graph'` takes a similarity matrix `S` as well as the number of neighbors `k` as an input an returns the adjacency matrix `W`.

Note that we will not evaluate the coding efficiency of your function.

```
def k_nearest_neighbor_graph(S, k):
    # S: similarity matrix
    # k: number of neighbors

    S = np.array(S)
    # k+1 because include_self. -S to pass from similarity to 
distance, +translation to avoid negative values
    G = kneighbors_graph(-S + S.max(), k+1, metric='precomputed', 
mode='connectivity', include_self=True).toarray()
    W = (G + G.T).astype(bool) * S

    return W

k = 2
# Please run this cell for evaluation purposes
S = [[1, 0.2, 0.7, 0.1],
     [0.2, 1, 0.8, 0.4],
     [0.7, 0.8, 1, 0.6],
     [0.1, 0.4, 0.6, 1]]

k_nearest_neighbor_graph(S, k)
```

```
array([[1. , 0.2, 0.7, 0. ],
       [0.2, 1. , 0.8, 0.4],
       [0.7, 0.8, 1. , 0.6],
       [0. , 0.4, 0.6, 1. ]])
```

```
# Please run this cell for evaluation purposes
S = [[1, 0.3, 0.01, 0.1],
     [0.3, 1, 0.8, 0.9],
     [0.01, 0.8, 1, 0.6],
     [0.1, 0.9, 0.6, 1]]

k_nearest_neighbor_graph(S, k)
```

```
array([[1. , 0.3, 0. , 0.1],
       [0.3, 1. , 0.8, 0.9],
       [0. , 0.8, 1. , 0.6],
       [0.1, 0.9, 0.6, 1. ]])
```

b) Spectral Clustering

Perform a spectral clustering using a k-nearest neighbor graph (with 4 neighbors).

Use the two features `reading_time` and `topics` only.

If you did not manage to solve task a), use a *fully connected graph* as similarity graph to obtain the adjacency matrix `W`.

You can assume that the optimal number of clusters is 2.

Print the obtained cluster labels.

```
# Function for doing spectral clustering
def spectral_clustering(W, n_clusters, random_state=111):
    """
    Spectral clustering
    :param W: np array of adjacency matrix
    :param n_clusters: number of clusters
    :param normed: normalized or unnormalized Laplacian
    :return: tuple (kmeans, proj_X, eigenvals_sorted)
        WHERE
        kmeans scikit learn clustering object
        proj_X is np array of transformed data points
        eigenvals_sorted is np array with ordered eigenvalues

    """
    # Compute eigengap heuristic
    L = laplacian(W, normed=True)
    eigenvals, _ = linalg.eig(L)
    eigenvals = np.real(eigenvals)
    eigenvals_sorted = eigenvals[np.argsort(eigenvals)]

    # Create embedding
    random_state = np.random.RandomState(random_state)
    proj_X = spectral_embedding(W, n_components=n_clusters,
                                random_state=random_state,
                                drop_first=False)

    # Cluster the points using k-means clustering
    kmeans = KMeans(n_clusters=n_clusters, random_state =
random_state)
    kmeans.fit(proj_X)

    return kmeans, proj_X, eigenvals_sorted

time =df[['reading_time']]
S1 = pairwise_kernels(time, metric='rbf', gamma=1)

topics = df[['topics']].apply(lambda x: set(eval(x.topics)),
axis=1).to_numpy().reshape(-1, 1)
S2 = squareform(pdist(topics, metric=lambda x, y:
float(len(x[0].intersection(y[0])) / len(x[0].union(y[0])))))

# Set diagonal to 1
gen = tuple([i for i in range(S2.shape[0])])
S2[gen, gen] = 1

S = (S1 + S2) / 2
```

```python
# Compute W
k = 4
W = k_nearest_neighbor_graph(S, 4)

# Call the spectral clustering function and print out the labels
clusters =2
kmeans, proj_X, eigenvals_sorted = spectral_clustering(W, clusters)
y_pred = kmeans.labels_
print(y_pred)
```

```
[0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0
0 1
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0
0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1
1 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0]
```