Geometric Machine Learning Algorithms

Problem Set 2

The purpose of this homework is

Problem 1: implement BFS_graph_connected_componentes

Writing a BFS algorithm that takes as input a *graph G* and returns back the *connected components of G*.

You need to use the networkx library you installed last week to implement this function. Compare your results to the result obtained here and make sure they are identical (up to ordering).

Problem 2 implement : Euclidian minimum spanning tree EMST

Write a function that takes as an input a collection of points $X = [p_1, \dots, p_n]$ and returns the Euclidian minimum spanning tree of X. This algorithm can be implemented as follows :

1- Compute the complete graph on the points $p_1, \dots, p_n$. Use the distance between the points as the weight of the edges of the complete graph.
2- Run the Kruskal's algorithm you implemented in problem set 1 on the complete graph you obtained in step 1.

Problem 3 Zahn's algorithm zahn_algorithm

Write a function that takes as an input a collection of points $X = [p_1, \dots, p_n]$ and a number of clusters k and returns back k clusters of the X using Zahn's clustering algorithm.

The points p_i can be in any Euclidian space R^d. The steps of the algorithm are described in lecture 4 but the outline are the following:

1. Compute the EMST of X.
2. Sort the edges of EMST(X) in decreasing order.
3. Delete the longest k-1 edges of EMST(X).
4. Return the k connected components (clusters) of the resulting forest.

Problem 4: ε-neighborhood graph clustering

Part 1 : implement epsilon_graph

Write a function that takes as an input a collection of points $X = [p_1, \dots, p_n]$ and a positive real number $\varepsilon \geq 0$ and returns the ε-neighborhood graph. This can be done in number of ways (choose either 1 or 2 –you do not have to do both only one):

1-Use the definition of the ε-neighborhood graph : you create a graph with nodes at the points of X and edges between x and y whenever $d(x, y) \leq \varepsilon$. You will also need to insert the weight on the edges being the distance between the points.

2-Use the adjacency matrix of the radius graph available at sklearn here. If you want to choose this option make sure to choose **mode='distance'** so that the output adjacency matrix gives back the correct weight between the edges.

Part 2 implement epsilon_graph_clusters

Write a function that takes as an input a collection of points $X = [p_1, \ldots, p_n]$ and a positive real number $\varepsilon \geq 0$ and returns the clusters induced by the connected components of the ε-neighborhood graph you computed in part 1. This can be done in the following two steps :

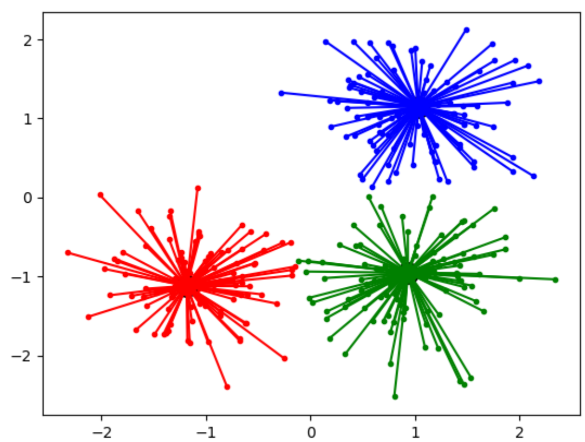
1- Use the function you implemented in part 1 to compute the ε neighborhood graph G.
2- Use the function you implemented in problem 1 to compute the connected components of G. You can also use the networkx function available here.

Remarks: Having your algorithm tested on a point cloud is essential. For this reason you have many option :

1- Create your own point cloud input : create a point cloud consists of a few points in the plane and run your input on it as you are testing your algorithms
2- Use sklearn point cloud generator functions : for instance the following two lines

```
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=300, centers=centers, cluster_std=0.5,
                            random_state=0)
```

produce three blobs centered at (1,1), (-1,1) and (1,-1) as shown below. For our purpose you need to ignore the labels and just focus on the set X.



you can use X above as your input for testing your algorithms. See here for the sklearn example.