TP1: Spectral Clustering
Graphs in ML

AL HOUCEINE KILANI

October 20, 2019

1 Graph construction

1.1- What is the purpose of the option parameter in worst_case_blob (in the file generate_data.py)?

Solution: This parameter is used to produce an outlier dot (vertex) placed on the X-axis. Its distance from the farthest vertex from the center of the graph is the *gen_param*.

1.2- What happens when you change the generating parameter of worst_case_blob in how_to_choose_espilon and run the function? What if the parameter is very large?

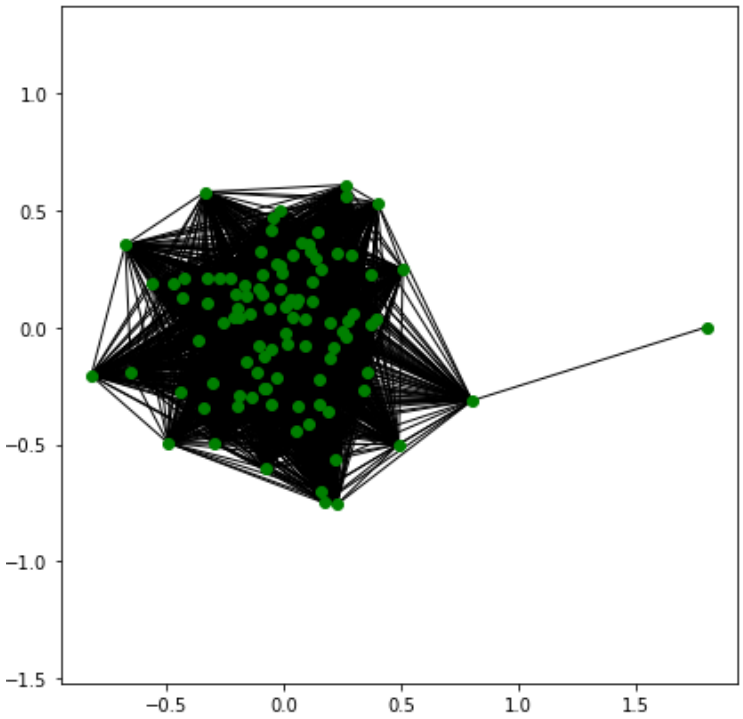

Figure 1: ϵ graph of the worst case blob — gen_param = 1

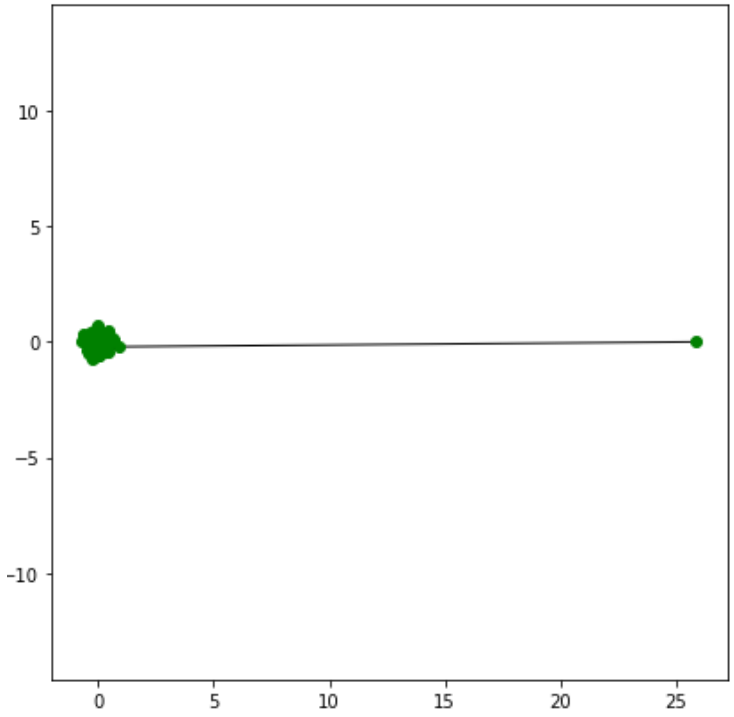

Figure 2: ϵ graph of the worst case blob — gen_param = 25

Solution: Figure 1 displays a worst case blob graph generated with parameter = 1, same for figure 1 with a parameter = 25. We see that the graph is fully connected because we took an adequate ϵ value using the min spanning tree. But the problem with this method of choice, is that it is not robust towards outliers, thus, when the outlier is far distant (when *gen_param* is very large), the ϵ value becomes smaller thus creating a vey dense graph where our outlier is connected to only one of the other non-outliers data points.

1.3- Using plot_similarity_graph and one of the datasets, compare knn to graphs. When is it easier to build a connected graph using k-nn ? When using graphs?

Solution:
Building connected graphs is easier using k-nn method for graphs that have quite different norms and scales of distances between verteces. Indeed for the 4 blobs data and two moons, I managed to get a fully connected graph easily using knn than using ϵ method and that for the reason told before : when we have large distances between the verteces, ϵ becomes smaller and thus we get dense graphs regions. So my conclusion would be using K-nn to build graphs if our data points have a large scale of distances whereas using ϵ graphs would be for the data points that are have inter-distances approximately in the same range and scale.

2 Spectral clustering

Build a graph starting from the data generated in two_blobs_clustering, and remember to keep the graph connected. Motivate your choice on which eigenvectors to use and how you computed

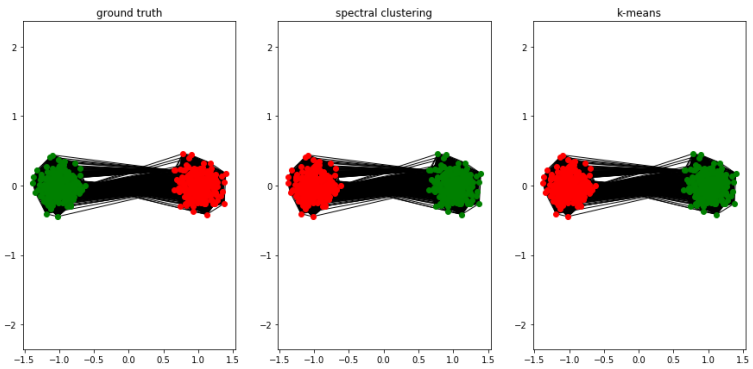

Figure 3: K-NN graph K = 300

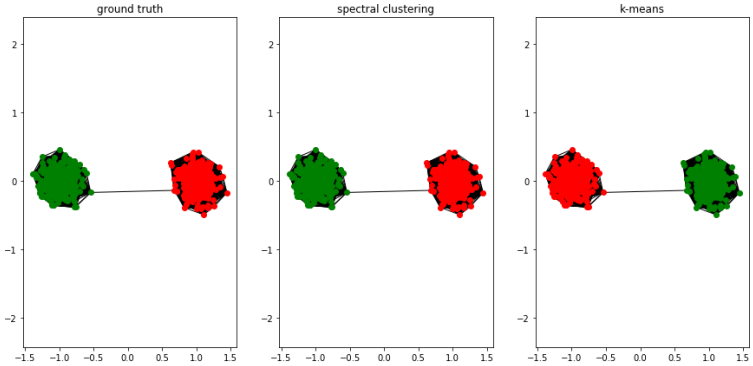

Figure 4: ϵ-graph

the clustering assignments from the eigenvectors. Now compute a similar clustering using the built-in k-means and compare the results.

Solution:
When using k-nn to build the connected graph, we have to choose a big value for K (Figure 3). Or we can use an ϵ-graph (using the minimum spanning tree heuristic) that works well too and is more adequate (Figure 4) (because, as we said before, the distances between the generated dots are more or less in the same scale)

Having a connected graph here means that there is 1 connected component, which means that we have 0 as an eigenvalue with multiplicity equal to 1. So after sorting the eigenvectors according to the eigenvalues, we can limit ourselves to considering the first eigenvector (corresponding then to the null eigenvalue) to perform the data points embedding (chosen_eig_indices = [0]) and applying a K-means on it.

This method performs the same as applying K-means directly on the data points.

Build a graph starting from the data generated in two_blobs_clustering, but this time make it so that the two components are separate. How do you choose which eigenvectors to use in this case? Motivate your answer.

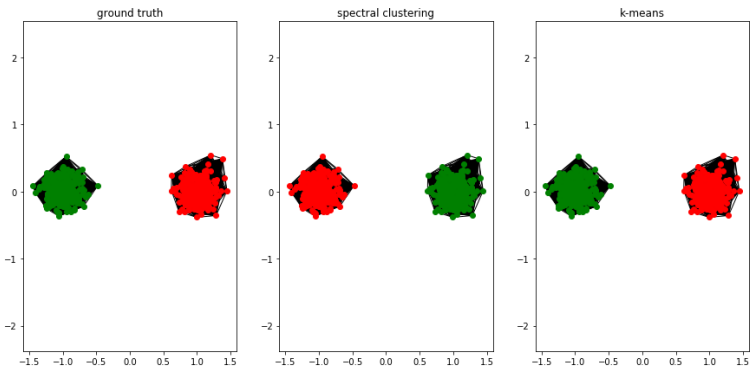

Figure 5: ϵ-graph

Solution:
By taking a bigger value for ϵ we built a non connected graph (Figure 5).
Now that we have two connected components, we will have a multiplicity 2 for the eigenvalue 0. Which means that to perform the clustering, we must take first and second eigenvectors (chosen_eig_indices = [0,1]).

K-means directly on the data points performs the same also in this case.

Look at the function find_the_bend. Generate a dataset with 4 blobs and $\sigma^2 = 0.03$. Build a graph out of it and plot the first 15 eigenvalues of the Laplacian. Complete the function choose_eig_function to automatically choose the number of eigenvectors to include. The decision rule must adapt to the actual eigenvalues of the problem.

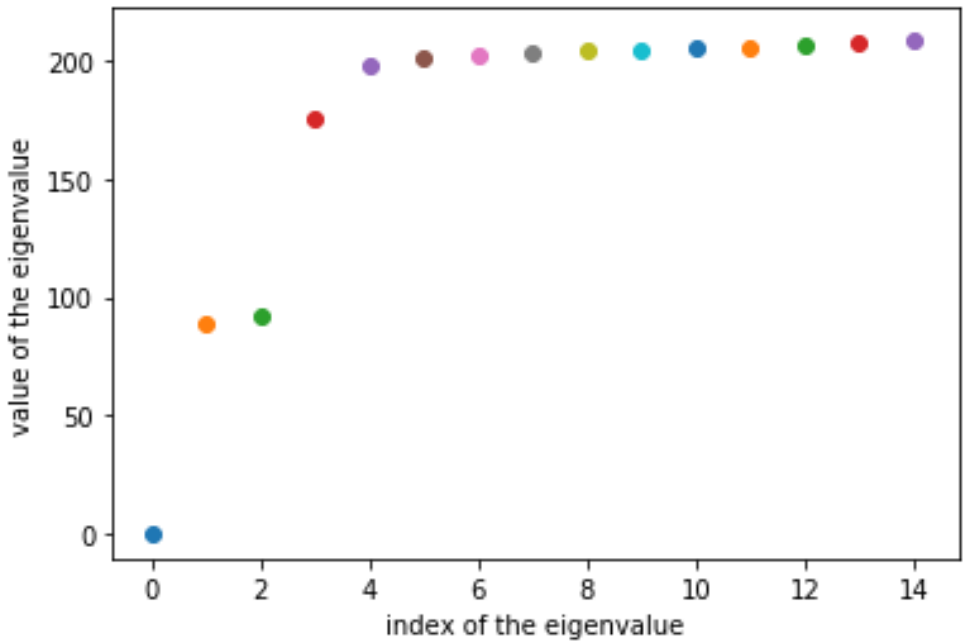

Figure 6: First 15 eigenvalues for the 4-blobs connected graph

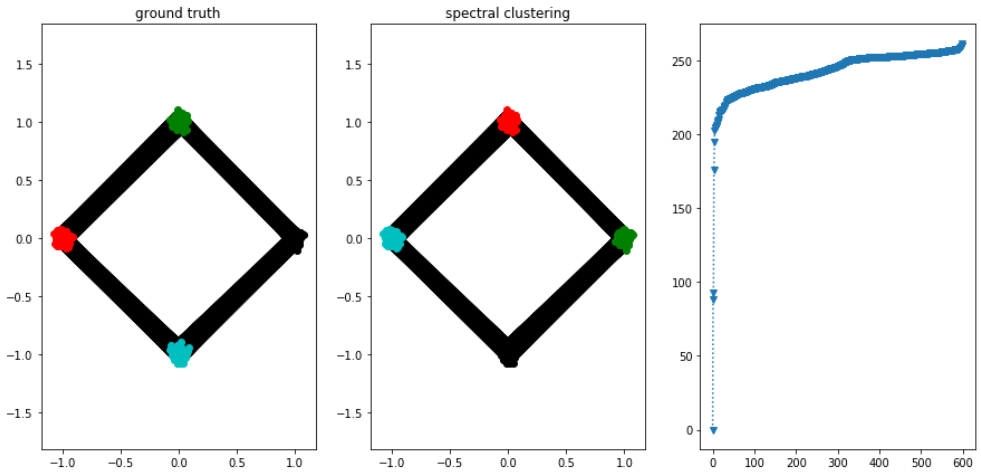

Figure 7: Clustering on the 4 blobs data using the adaptive method for the choixe of vectors

Solution:
We see that (Figure 6) there is a big gap between two eigenvalues. So to choose automatically which eigenvectors to include, we have to spot automatically this gap, and exactly : the index of the last eigenvector before that sudden jump in values.

We will simply compute a first derivative of the eigenvalues array to identify this gap (where the value of the derivative will be the higher value). This method is not that robust (especially in the case where there will be multiple consequent gaps before the one that will be spotted) but it is a first approach that gave good results (Figure 7).

The interesting thing here is that our adaptive method just returned the 3 first eigenvectors (indices 0, 1 and 2) where we would fix manually the four first eigenvectors (which is a drawback of this heuristic and can be improved by computing second order derivatives or using histograms)

Now increase the variance of the Blobs to $\sigma 2 = 0.20$ as you keep plot- ting the eigenvalues. Use choose_eig_function.m. Do you see any difference?

Solution:
When increasing the variance, we have larger blobs and thus the ranges of distances between become closer. Which makes building an ϵ-graph easier. (K-nn Graph works too when choosing a large enough value for K, but the graph will be dense like in the previous question).

After few reruns of the code, I see no difference and the adaptive spectral clustering works well.

When you built the cluster assignment, did you use thresholding, k-means or both? Do you have any opinion on when to use each?

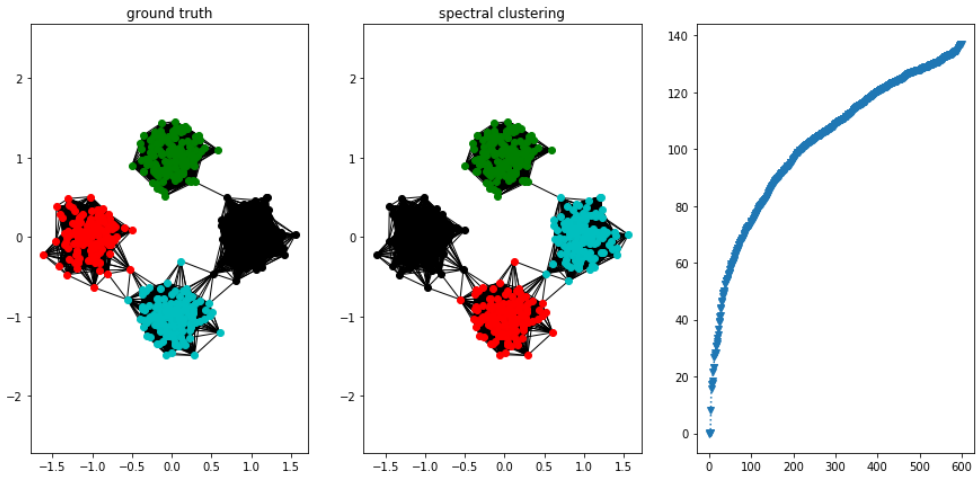

Figure 8: Clustering on the 4 blobs data using the adaptive method for the choixe of vectors, $\sigma^2 = 0.2$

Solution: I used K-means for the cluster assignements. Manual thresholding may be interesting to overcome to random initialisation of the centeroïds of the KMeans algorithm. But if the dataset becomes larger and non conventionnal, it is, in my opinion, safer to use an algorithm and rerun it multiple times instead of forcing borders and clusters manually and risk to commit mistakes.

What is another use that looking at the distribution of the eigenvalues can have during clustering, beside choosing which eigenvectors to include?

Solution:
If we get to manage to build a graph that is not too dense, the number of eigenvalues that take small values is an indication of how many cluster there are in the dataset.

Plot your results using spectral clustering and k-means in two_moons_clustering and compare the results. Do you notice any difference? Taking into consideration the graph structure, can you explain them?

Solution: I found it really hard to get some good clustering in this case with spectral clustering. Indeed it is hard to find the right K to build a KNN graph (since the ranges of distances between the dots is large). So I decided to use an ϵ-graph instead. But there seems to be an issue with the distance used to build the graph : it is not adequate for such structures of data (dense in some parts and sparse in the center and in the regions separating). The best result I managed to get is the following.

Same comment for K-means, it is not adequate for this type of structures since the euclidean distance is not appropriate here as a metric of similarity.

It may be interesting to find some sort of manifold that we can use to transform the data points so as they can be more incline to be clustered using the euclidean distance or the

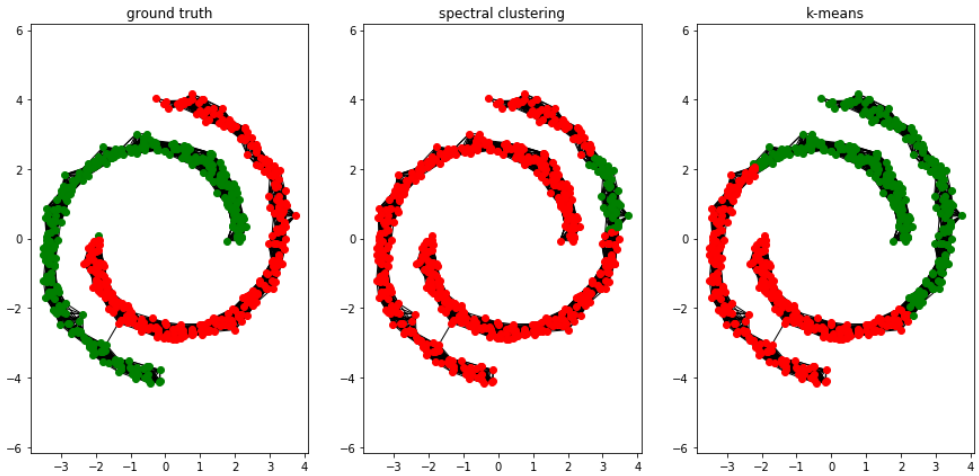

Figure 9: Comparison of clustering methods for and epsilon graph

exponential similarity defined in the TP. (We may also learn some metric specifically for this kind of data structure)

In the function point_and_circle_clustering, compare spectral clustering using the normal laplacian L and the random-walk regularized Laplacian Lrw. Do you notice any difference? Taking into consideration the graph structure, can you explain them?

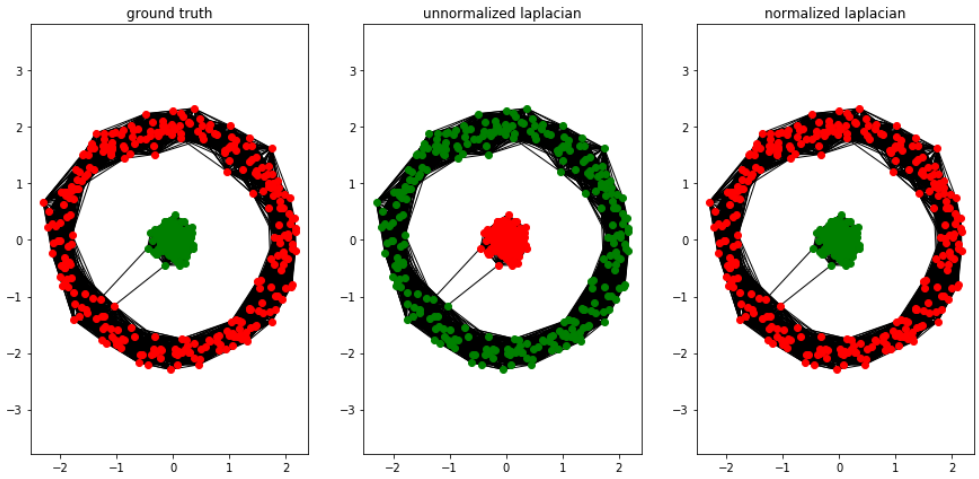

Figure 10: Clustering using different L matrices

Solution: Built a connected graph using K-nn (K = 50), both laplacians managed to give good clustering results. But the problem here again was finding the adequate value for K to build the graph.

Complete the function parameter_sensitivity, and generate a plot of the ARI index while varying one of the parameters in the graph construction or k. Comment on the stability of spectral clustering

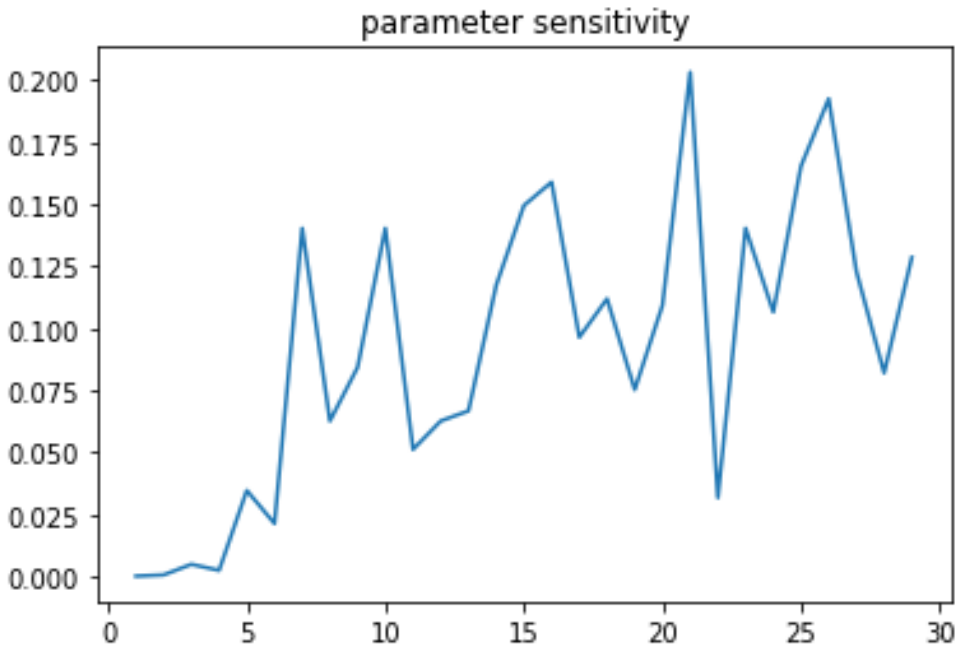

Figure 11: Spectral clustering on two moons graph built with different values for K

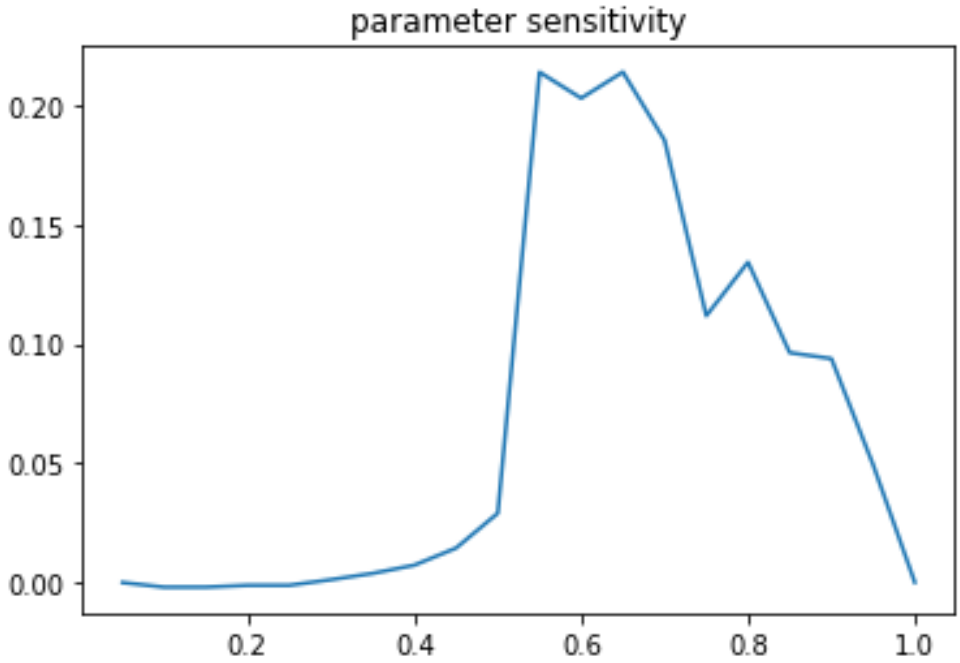

Figure 12: Spectral clustering on two moons graph built with different values for epsilon

Solution: For the two moons dataset, we see that there is an area of stability of the assigned clusters for epsilon less than 0.5, where as it is hard to have stable results if we run clustering on a KNN built graph. This rejoins the question **2.7** where we found it hard to build a

connected graph with the KNN method for the two moons.

If we did not have access to true labels how could we evaluate the clustering result (or what should we not use as evaluation)?

Solution:
Depending on the context of the data, we can analyse the quality of the clusters with the domain specialists.

We may also try to look for similar structured datasets that have known labels, apply our clustering on it, tune it so as it performs well on it and then cluster the real data with this tuned model.

3 Image segmentation

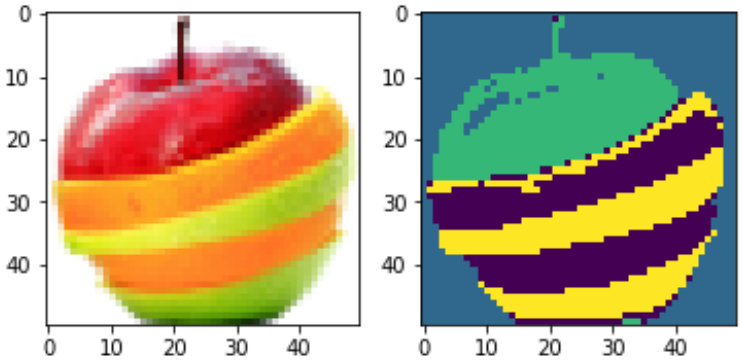

Figure 13: Fruit salad image color segmentation

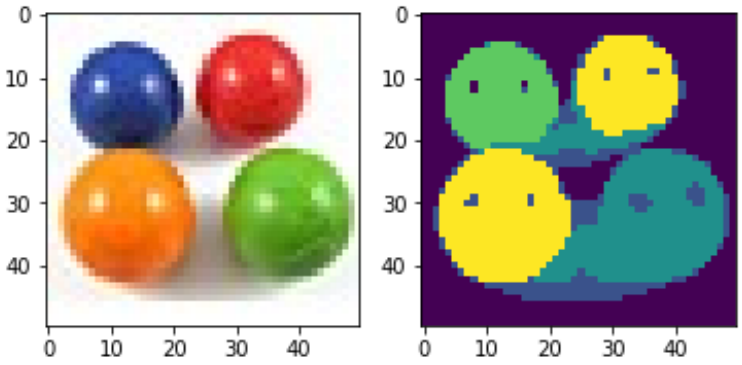

Figure 14: Baloons image color segmentation — Orange and red colors assigned o the same cluster

The first documentation is the code. If your code is well written and well commented, that is enough. If you think you need to better express some of the choices you made in the imple-

mentation, you can use this question in the report. Remember to include all the related code in the submission.

Solution: I hope that the code is well documented.
I want to precise something concerning the spectral clustering function and the spectral decomposition functionIf :

if we want to make things run faster, we can use the eigs function, but in this case we will have to make a choice on the number of eigenvectors to include beforehand. Whereas if we want to use an adaptive version of the spectral clustering (where the eigenvectors to include are automatically picked), we cannot use the eigs function.
A problem that I had was that I had a eingenvector matrix containing complex numbers (very small but non null) so I have to modify the functions in spectral_clustering.py to only consider the real part of the numbers.

A full graph built between the pixels of a 50 50 image corresponds to 50**2 nodes. Solving the full eigenvalue problem in this case would scale in the order of 2**34. Even on weak hardware (i.e. iPhone) this takes only seconds to minutes. Segmenting a Full HD picture of 1920 1080 would scale in the order of 2**64 (about a month on a decent machine i.e. not an iPhone). Beyond that, the large picture would require to store in memory a graph over millions of nodes. A full graph on that scale requires about 1TB of memory. Can you think two simple techniques to reduce the computational and occupational cost of Spectral Clustering?

Solution: We may play on the W matrix, the method of building the graph (which influences the Laplacian) or the method for finding the eigenvalues.
I found out that Low-rank matrix approximations methods (Nyström approximation) can find approximations of the eigenvectors and values only using subsets of the matrix (here the Laplacian).
We can use the "eigs" function istead of "eig" function to find the eigenvalues/vectors.

Did you use eig or eigs to extract the final eigenvectors? Shortly, what is the difference between the two? How do they scale to large graphs (order of complexity)?

Solution: I used eig even if eigs is more adequate in our case (L is a sparse matrix because the graph is not connected). The difference between eig and eigs is that we do not compute all the eigenvalues/vectors when using eigs by specifying how many we want.
The computational complexity of eig is $O(N^3)$ where as eigs varies with the sparcity of the matrix

Did you use eig or eigs to extract the final eigenvectors? Shortly, what is the difference between the two? How do they scale to large graphs (order of complexity)?

Solution: I used eig but eigs in a much better choice for our application. Indeed it is much more efficient in our case as it only return the first n eigenvalue, which is what we want, we don't need all eigenvalue. Furthermore eigs is optimized for sparse matrix which is our case as the graph is not fully connected.
In terms of complexity, *eig* has a complexity of $\mathcal{O}(n^3)$, but I don't know the complexity of *eigs*, it seems to vary with the sparsity of the matrix.