TP 1 : Spectral Clustering

Raphaël Chekroun

1 Question 1

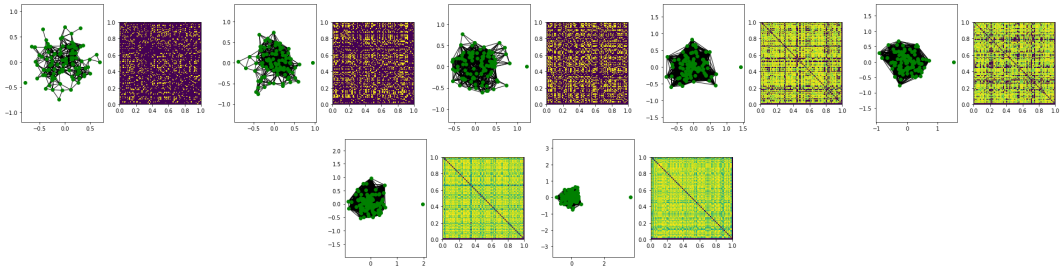
1. The optionnal parameter **gen_pam** correspond to the distance to which the first data element of the Gaussian blobs generated by **skd.make_blobs** is shifted on the abscissa axis from the previous far right element.

Thus, it create a singular blob in the distribution.

2. If **gen_pam** is big enough, the graph goes from a one connected component graph to a two connected component graph, as the worst blob now is way more far than the other blobs from the center of the Gaussian.

Moreover, as ϵ is chosen as a function of the maximal value of distance between two linked vertexes whose edge is present in the minimal spanning tree of the fully connected graph, the higher this parameter goes, the more dense is the graph matrix. It seems to reach a density saturation around when **gen_pam**$> \sigma^2$, which makes sense knowing than when the blob is farther from all of the other blobs from σ^2, it shouldn't be a neighbor of any blobs anymore.

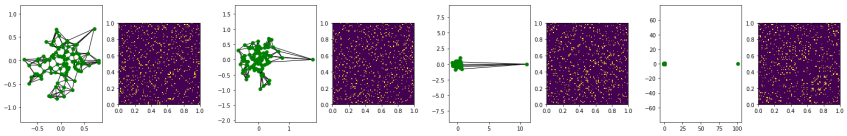
We can observe this in the next figures, for **gen_pam** $\in [0, 0.2, 0.4, 0.6, 0.8, 1, 3]$:



3. Let's consider we're working on the blob dataset.

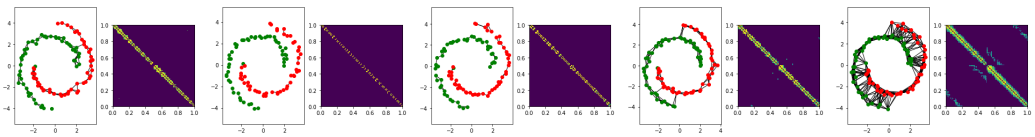
When the data is clean, so when **gen_pam** is very low or null, an ϵ-graph is the easiest to compute and is relevent enough. When **gen_pam** is high, the ϵ parameter isn't relevant anymore because of our criteria of selection. Thereby, in this case, constructing a $k - nn$-graph is best, as the results are still relevant.

We can observe this in the next figures, for **gen_pam** $\in [0, 1, 10, 100]$ and $k = 5$:



If we're working with the two moon dataset, we notice that the clustering seems to work well for $k-nn$ graph as well as for ϵ-graph, as shown in the followinf images.

This is an ϵ-graph followed by $k - nn$-graph for $k \in [1, 2, 5, 10]$



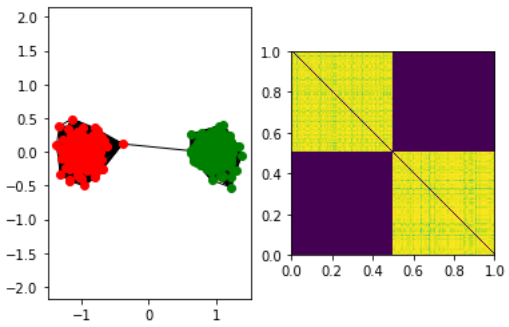
It's caused by the fact that the graph density never gets too high, due to well defined shape of the distribution and the fact than distances between vertices varies a lot.

As a conclusion, $k - nn$ graph is a better choice than ϵ-graph when distances between data points have different scale among space regions.

2 Question 2

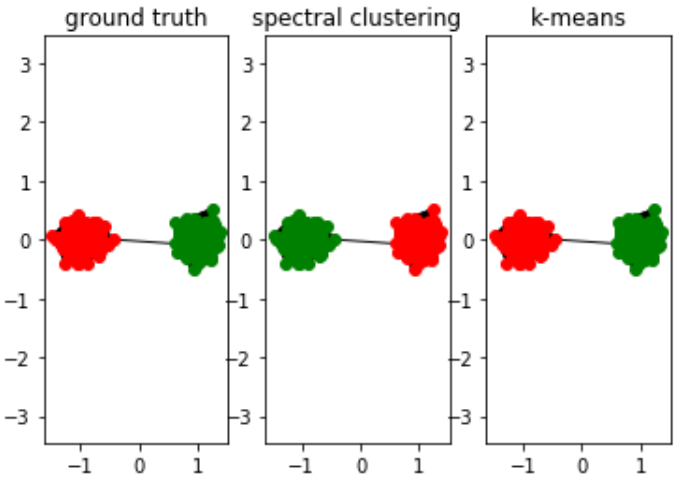
1. Let's consider an ϵ-graph.

As we want to have a connected graph we chose ϵ as the maximum weight of the minimal span tree. This way, the graph is connected, but only by a single edge, as the following picture shows us:

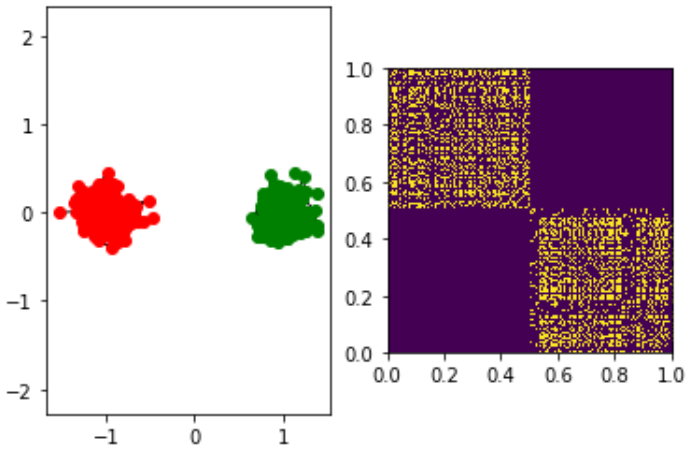


Thus, the first eigenvalue is 0, as there's only one connected component. The second eigenvalue is 1, as we're only one cut away to go to one to two connected component. The eigenvector correponding to this value will give us the edge correponding to the bridge between the two blobs by defining in which subset (left or right) each vertex belongs to. Thus, we can cluster our two blobs.

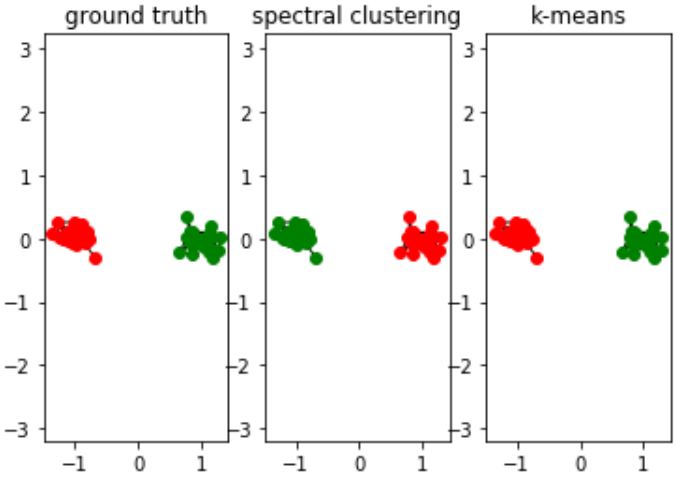
The integrated Kmeans, used with two classes also works well and give the same result as the spectral clusterisation, as tht next figure shows us:



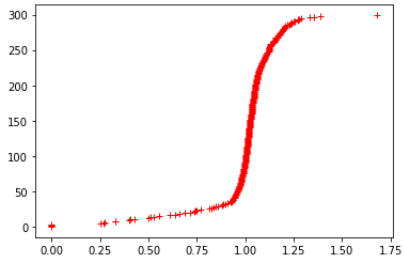
2. This time, we chose ϵ as the second maximum weight of the minimal span tree, thus allowing the two set of blobs dense and highly connected. We thereby have two well define components, as the following figure shows us:



As the graph is no more connected but has two connected components, the eigenvalue 0 has multiplicity of 2, and we need to two first eigenvectors to bring enough information for clustering the dataset. Once again results for kmeans and spectral clustering are good, as highlighted by the following figure:

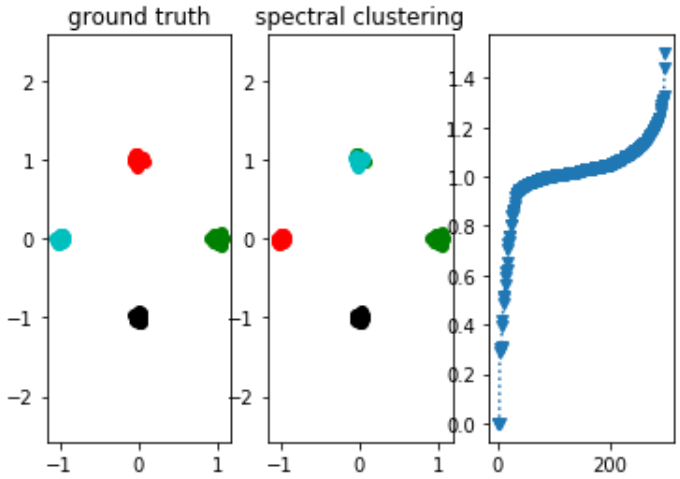


3. Indeed, Looking at the plot of eigenvalue, for instance in the following figure, we guess that we have to find the last index before a "significant" increase in eigenvalue.



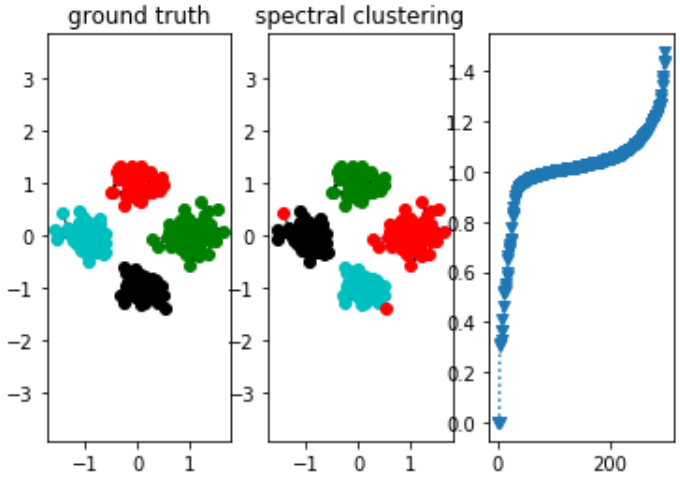
To do so, my solution is in the piece of code of the function **chose_eigenvalues**: we keep all the eigenvalues, indexed by $i = \{1..n\}$ until $2 * eig[i] < eig[i-1] + eig[i+1]$.

In order to make **find_the_bench** work, we build an ϵ graph with ϵ being the **num_classes**-th biggest weight in the minimal spanning tree. This way, we have a grave composed of **num_classes** connected component, as shown in the following figure where we have 4 classes:



Thus, the **num_classes** following eigenvectors gives us all the information we need on the clustering.

4. With $\sigma^2 = 0.20$, the dataset is more difficult to cluster. The method explained in the last question, allows us to find a clustering but with a few errors, as shown in the next figure:



5. I used k-means for the cluster assignment.

I believe k-means allows to find more details in the data by giving a more subtle clustering delimitation. Nonetheless, using k-means also brings some instability due to its initialization. Using thresholding is more stable and would give more rectiligne, geometrical cluster delimitation, by construction.

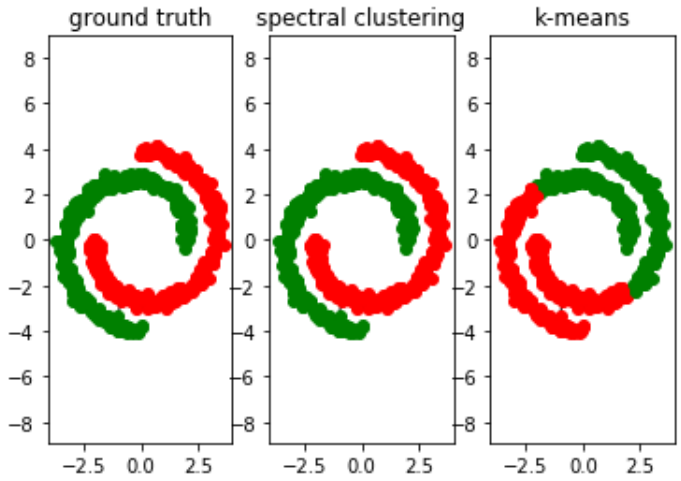
Thus, k-means would be more efficient to clusterize clusters far from each other but with a large variance, while quality thresholding clustering would be more efficient to custerize close clusters. Indeed, quality thresholding allows to chose a maximal radius, so if the clusters are close and kind of overlap, the clusterinzation can take place by finding the center of every cluster and the variance of the distribution.

Nonetheless, k-means needs to know the number of clusters while quality thresholding needs to know the threshold and the minimal amount of blob in a cluster. This parameters also takes place in the choice of clustering algorithm.

6. The distribution of the eigenvalues can also gives us information on how many clusters compose the dataset. Indeed, when clustering we don't usually know the value of k for the k-means algortihm, i.e. the number of clusters. If the graph is connected but neither too dense nor too sparse, the number of cluster in the dataset will be the number of 'small' eigenvalues.

7. I chose to build an ϵ-graph. The value of ϵ making it work is the same as before: I chose ϵ as being the distance correspondong to the greatest weight in the minimal spanning tree of the distance matrix.

The eigenvalues are automatically chosen by the previously designed function.

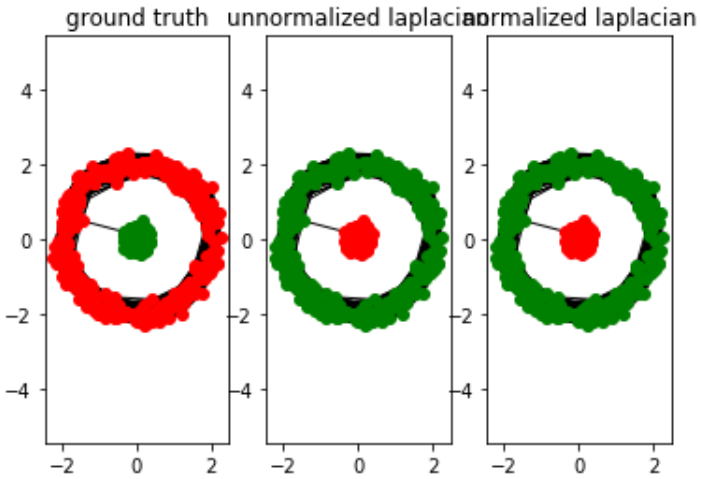


One again, this method works because chosing ϵ like this allows to draw a few bridges between the clusters while still having one connected component. Then, the first eigenvectors says which cluster is on which side of each bridge, giving us the wanted answer.

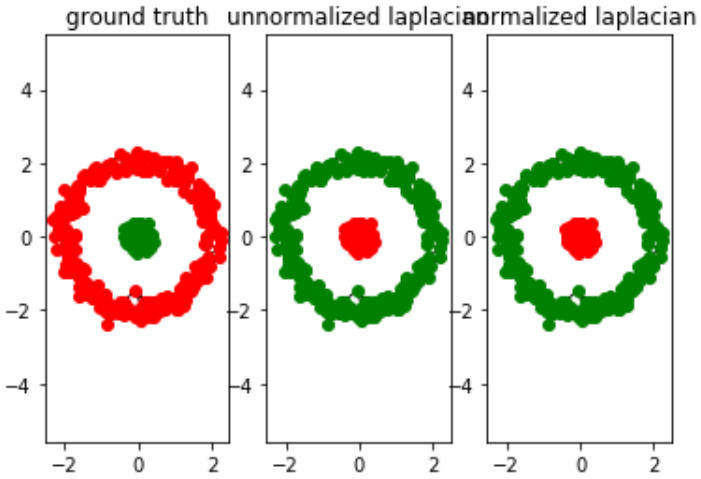
Nonetheless, things aren't that good for k-means, whom just cuted both clusters in two parts: as it's searching the average of blobs to clusterize according to the distance of means and blobs, it can only design clusters with a convex shape. The two moons dataset, being composed of two non-convex shape, can't be well clustered by the k-means.

8. I once again chose to build an ϵ-graph. The value of ϵ making it work is the same as before: the distance correspondong to the greatest weight in the minimal spanning tree of the distance matrix.

Nonetheless, this time, the automatic choice of eigenvectors gave a fake result. I thus chose wisely the eigenidexes, who are 0 and 1. Indeed, thanks to the choice of ϵ, there's only one connection between the two clusters, and we only need the 0-th eigenvalue to know that there's juste one connected component, and the 1-st eigenvector to find out which vertex is on which side of the unique bridge.



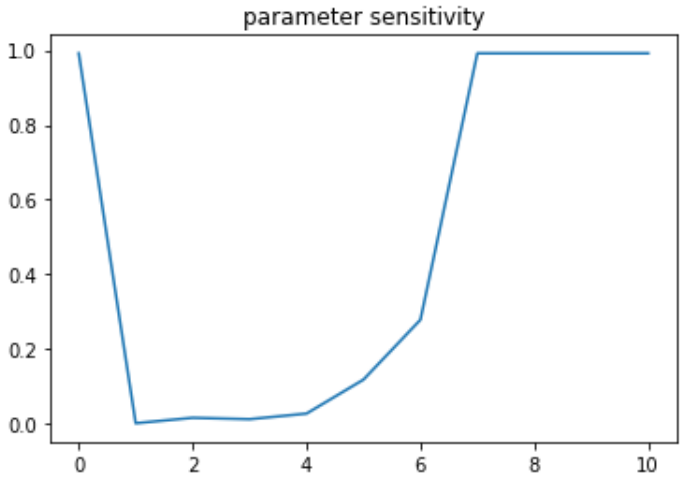
Nonetheless, this doesn't allow to make any observations on the difference between the two kind of laplacians. I thereby chose to test k-nn-graphs. It happens that starting $k = 5$ and with the same choice as chosen eigen indexes as before, the resultats are still good, as shown in the next figure for $k = 6$:



Thus let's have a theoritical approach on the difference between the two laplacians.

Unnormalized Laplacian solves the RatioCut strategy (who maximizes the cardinal of each cluster), while Normalized Laplacian solves the NCut strategy (who maximizes the sum of the weight in the cluster). The point and circle dataset have a very dense and close blob of point in the middle, a little gap and then around it a denser "ring" of points. If we want to cut the graph in two, making a cutting in the blob (or the ring) will penalize a lot the NCut object function because weight are high within the blob (or the ring). But it won't penalize as much the RatioCut as it won't impact the cardinality. Thus, the Normalized Laplacian is a better choice for this case than the Unnormalized Laplacian.

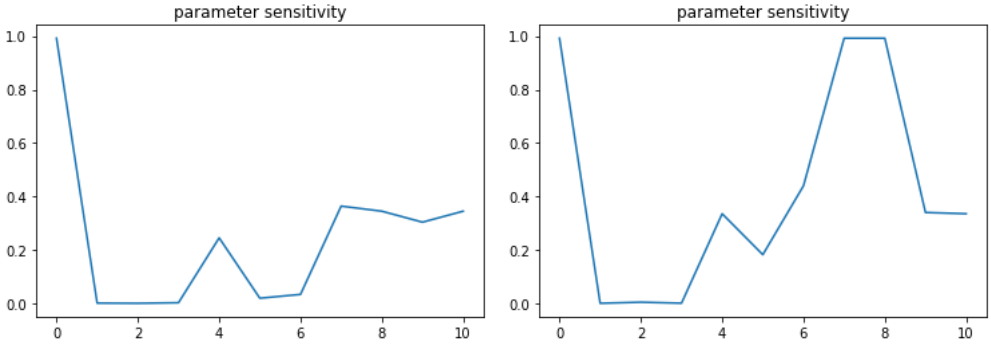
9. For $k \in \{0..10\}$, we have the following evolution of the ARI:



Thus, we can see that, for an ϵ-graph where ϵ is chosen as before, we have a perfect clustering.

The clustering seems to start from very low for $k = 1$ and reach 1 for $k > 6$.

I repeted the test a few times and the observation is often the same, but sometimes the ARI is lower than expected for $k > 0$, or with an unexpected evolution, as shown in the next figure:



Thus, we can conclude than the spectral clustering is unstable, particularly when computing it over a k-nn-graph.

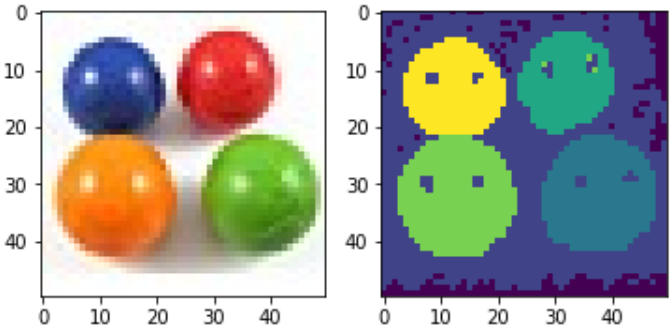
10. First of all, one souldn't use a clustering algorithm to evaluate another one. Indeed, as we're not able to evaluate any clustering whatsoever, it seems like a bad idea.

One thing that can be done, even if it doesn't guarantee than the clustering on every dataset is good, is to test the algorithm on a given, known and evaluable dataset. This way, we can have the first clue of if the algorithm can be efficient in some cases. However, it's impossible to guess for the general case.

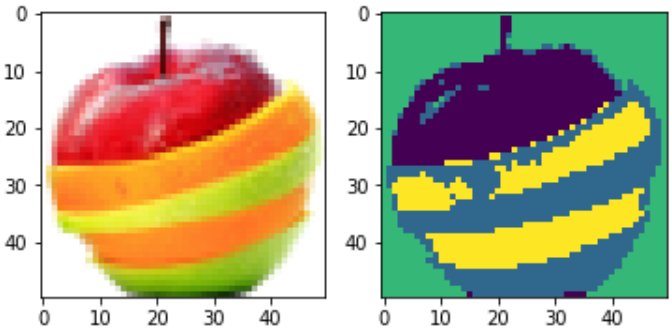
3 Question 3

After some tests, I found the best parameters to be $k = 45$ and $var = 5$. The eigen values indices are chosen by the function implemented in a previous question. I found the optimal number of class for each segmentation to be the number of saved indices, which makes sense knowing that the clustering information is in all the first len(number of clusters) eigenvectors.

For the 'four_elements.bmp' image, we have the following result:



For the 'fruit_salad.bmp' image, we have the following result:



1. According to the litterature, we can optimize the computational and occupational cost of spectral clustering by:

- Quantizing the Laplacian to reduce the dimension. This can be done by some algorithms such as K-means or graph embedding using neural network.
- Finding eigenvalue without storing all the Laplacians with the Nyström approximation. Nonetheless, this method only give approximations of eigen objects.

2. I used eig but it happens $eigs$ would have been a better choice in our case. Indeed, $eigs$ allows to compute just the n first eignavalues instead of all of them - and this is what we want - and is optimized for sparse matrix - and this is the case in our situation, the graphs being not totally connected. The complexity of $eigs$ is dependant of the sparsity of the matrix, while the complexity of eig is in $O(n^3)$.