Spectral Clustering: Implementation and Performance Analysis

June 2023

Nicola Orecchini
Politecnico di Torino
s303998@studenti.polito.it

Leonardo Moraglia
Politecnico di Torino
s309345@studenti.polito.it

Contents

I. Introduction

This report addresses the solving of the Homework about Spectral Clustering for the final exam of the course "Computational Linear Algebra for Large Scale Problems", held during the academic year 2022-2023. In this homework we will implement, in the programming language Python, the Spectral Clustering method, an Unsupervised Learning technique to classify unlabelled data.

The questions we empirically aim to ask within this study is how the Spectral Clustering algorithm performs versus other clustering algorithms (e.g. k-Means) in grouping an unlabelled set of points, and also what factors influence the performance on the Spectral Clustering.

In this report, we provide a detailed explanation of the work: in II, we describe how we tackled different tasks; in III we present our experimental setup, made of two different datasets and by an additional clustering technique (k-Means) for comparison; in IV, we analyse the behaviour of Spectral Clustering technique among the different datasets and how does it compare to the k-Means algorithm; finally, in V we wrap up our findings.

II. Methodology

Spectral Clustering (SC) is a technique originally designed for graphs. SC allows us to separate the nodes of a graph in what are known as its *connected components*: a connected component is a group of nodes in a graph where every node has a path to any other node within the group. If all nodes of a graph are connected, then there only is a single, big connected component. SC aims to find connected components that only have a "weak" connection between them, so that we can approximate the graph by separating those two "not-perfectly connected components" but rather "approximated connected components". SC achieves this by leveraging a fundamental property, that is that the smallest eigenvectors of the Laplacian matrix associated to the graph are able to find a splitting of the graph into connected components. The SC can be also applied to a set of n-dimensional points, provided that a representation of the points as a graph is available.

We move on to explaining how we implemented the full SC algorithm in the programming language Python by presenting the steps we followed, according to the instructions provided in the Homework assignment.

1) The first step consists in, starting from the set of points, constructing the k-nearest neighborhood similarity graph associated with the points. To do this, we define the nodes of the graph as the points in the set, and their edges (i.e., their connections) as numbers based on the value of the similarity between the points, calculated as follows:

$$s_{i,j} = \exp\left(-\frac{||X_i - X_j||^2}{2\sigma^2}\right), \text{with } \sigma = 1 \qquad (1)$$

where X_i, X_j are two points, and $s_{i,j}$ is the weight of the edge between node i and node j. Note that values of $s_{i,j}$ close to 1 indicate higher similarity between the points, while values closer to 0 indicate higher dissimilarity. Actually, we would like to work with sparse matrices, as datasets of points can be arbitrarily large, and for this reason we introduce a simplification: we do not take all the $s_{i,j}$, but rather the more "significant" values. To accomplish this, we only take, for each point i, its

k-nearest neighbours based on the specified similarity function, filtering as follows:

$$s_{i,j} = \begin{cases} s_{i,j} & \text{, if } s_{i,j} \in \text{k-highest values} \\ & \text{among } \{s_{i,1}, s_{i,2}, \dots s_{i,n}\} \\ 0 & \text{, otherwise} \end{cases}$$

A further simplification is introduced:

$$s_{i,j} = s_{j,i}, \forall i, j$$

i.e. the graph is undirected.
To conclude this point, we must compute the adjacency matrix, which contains all zeros along the diagonal, whereas outside it contains the $s_{i,j}$. To accomplish this task, we implemented the following two functtons:

- **calculate_pairwise_distances(X, sigma=1)**: This function calculates pairwise distances between a set of points using the given formula. It takes the following arguments:
  - X (numpy.ndarray): Array representing a set of points. Shape: (N, D), where N is the number of points and D is the dimensionality of each point.
  - sigma (float): Value of sigma (default value is 1).

  The function first calculates the squared pairwise distances between all pairs of points using np.square(np.linalg.norm(X[:, None] - X, axis=2)). Then, it applies the exponential function to the negative squared distances divided by 2 * sigma**2. It returns the pairwise distances as a numpy array of shape (N, N), where N is the number of points in X.
- **calculate_adjacency_matrix(df, sigma, k)**: This function calculates the adjacency matrix based on pairwise distances between points in a DataFrame, considering only the k-nearest neighbors for each point. It takes the following arguments:
  - df (pandas.DataFrame): DataFrame containing points' coordinates. Rows represent points, and columns represent coordinates
  - sigma (float): Value of sigma.
  - k (int): Number of nearest neighbors to consider.

  The function first converts the DataFrame to a numpy array (X = df.values). Then, it calculates the pairwise similarities between points using the **calculate_pairwise_distances** function. Next, it finds the k points with the highest similarity for each point by sorting the similarities and selecting the k highest indices. Finally, it constructs the adjacency matrix as a sparse lil_matrix where the (i, j)-th element is set to the corresponding pairwise distance if j is one of the k-nearest neighbors of i; otherwise, it is set to 0. The function returns the adjacency matrix.

2) Next, we calculated the Degree Matrix and the Laplacian Matrix through the following functions:
   - **calculate_diagonal_matrix(A)**: This function calculates the degree matrix D using the adjacency matrix A. It takes the following arguments:
     - A (scipy.sparse.lil_matrix): Adjacency matrix.

     The function first calculates the sum of weights (or similarities) for each point by summing the elements in each row of the adjacency matrix using np.array(A.sum(axis=1)).flatten(). Then, it constructs the diagonal matrix D as a sparse lil_matrix where the (i, i)-th element is set to the sum of weights for the i-th point using D.setdiag(row_sums). Finally, it returns the diagonal matrix D.
   - **calculate_laplacian_matrix(A, D))**: This function calculates the Laplacian matrix L by subtracting the adjacency matrix A from the diagonal matrix D. It takes the following arguments:
     - A (scipy.sparse.lil_matrix): Adjacency matrix.
     - D (scipy.sparse.lil_matrix): Diagonal matrix.

     The function simply subtracts the adjacency matrix A from the diagonal matrix D using the expression L = D - A. It then returns the Laplacian matrix L.
3) To compute the number of the connected components, we simply calculated the first smallest eigenvalues of the Laplacian matrix. From the theory we know in fact that the number of connected components is approximately equal to the number of eigenvalues of the Laplacian matrix having a value near zero. Specific results regarding the number of connected components of the different datasets are presented in Section III.
4) To compute the smallest eigenvalues of the Laplacian matrix, we used both the *scipy.sparse.linalg.eigs()* [2] function and our custom implemented Deflation Method, whose details are reported in the other homework "Deflation Method with Inverse Power Iteration for Computing the Smallest Eigenvalues of Symmetric Matrices: Analysis and Implementation". Then, the value of M for the Spectral Clustering algorithm has then been chosen: as we know from the theory, the eigenvalues of the Laplacian that are 0 or very close to 0 indicate that the graph can be split in connected components, whose number is given exactly by the number of near to zero eigenvalues of the Laplacian. So, what we did is to define a function that calculates first the average of the 10 smallest eigenvalues, and then sets a threshold at the value of the average divided by a constant value, to be empirically determined, which we set at 2.5:

$$ths = \frac{\hat{\lambda_1} + \dots + \hat{\lambda_{10}}}{10} * \frac{1}{2.5}$$

Then, each eigenvalue under the threshold is considered zero.
5) Next step is to calculate the corresponding eigenvectors of the Laplacian: they will indicate us the "cuts" that have to be done in the initial graph in order to separate it

in its M connected components. We calculate the eigenvectors associated with the smallest eigenvalues again both with our custom implemented Deflation Method and with the *scipy.sparse.linalg.eigs()* function [2]. In this step, we also construct the matrix U, whose columns are given by the M found eigenvectors. Note that the matrix U has N rows, with N number of points in the initial dataset and M rows, where M is the number of found connected components.

6) We can think of the process we did up to now as a pre-processing of our initial data, as we transformed the starting dataset, which was of dimension NxD, with $D = 2$ number of dimensions of the data, to a dataset of dimension NxM, with M number of connected components of the graph associated to it. Here, we simply apply any clustering algorithm to the rows of this new matrix (we apply the KMeans function of the sklearn.cluster library for Python).
7) Then, we take the column containing the labels obtained from this clustering and simply copy-paste it into the initial dataset, to find clusters of the original points.
8) We plot the results of the clustering, along with other diagnostics figures in IV.
9) Finally, we apply the k-Means algorithm directly to the original set of points to compare and inspect the differences with the results obtained with Spectral Clustering.

III. Experimental Setup

We present here the components of our experimental setup:

1) We have two different datasets to cluster: "Circle" and "Spiral". Both are 2-D datasets, so we can easily visualize them in Fig. 1. The first is made of 900 points, the second 312. The first does not have the correct clusters indicated, while the second does.
2) We start by constructing the corresponding k-nearest neighborhood similarity graph for our two datasets, trying out different values for k (number of considered neighbours): 10, 20 and 40. We can visualize the results in the first columns of the graphs in Fig. 2 for the Circle dataset and in Fig. 3 for the Spiral. We can immediately get a sense of what constructing the k-nearest neighborhood similarity graph means: we are drawing edges between the points, based on whether they are similar to each other, according to Equation (1). We can also see how increasing the value of k leads to having more connections in the graph.
3) Then, we calculate the Laplacian matrix and plot its 10 smallest eigenvalues in the second columns of Figures 2 and 3. We do this using both our custom implemented Deflation Method and the scipy function, which we indicate as "theoretical" on the graphs. We also plot, in these graphs, two vertical lines (one for each method we used for computing the eigenvalues) corresponding to the last eigenvalue that is comprised within the threshold defined. In other words, the vertical lines indicate the number of connected components in the graph.
4) Continuing in our experimental setup, we plot, in the third and fourth columns of Figures 2 and 3, the clusters obtained by Spectral Clustering with our Deflation Method and with Scipy functions to compute eigenvalues and eigenvectors. In fact, the usage of a specific method may lead to taking different decisions, so we want to assess the main "long-term" effects of using a method with respect to another.
5) To conclude our experimental setup, we perform the clustering of the initial dataset using a k-Means method directly to the set of points, and plot the results for comparison. The results are contained in second column of graphs of Figure 4, while in the first column we include the "Elbow Plot", which is a useful tool to decide the optimal number of clusters.

IV. Discussion of results

We summarize in this section our findings and interpretation of the results, by answering the following questions.

- **What are the effects on Spectral Clustering of using different methods for the computation of eigenvalues?** From the graphs in the second columns of Figures 2 and 3, we can assess that using different methods to compute the smallest eigenvalues can lead to different decisions regarding the number of connected components in the graph: not always the red and the blue vertical lines are aligned. However, we must report that in 4 graph out of 6 the difference between the two vertical lines is 1 or 0, while in the other two cases is 4. We also note that, in both the two cases presenting a larger difference, the red line (theoretical number of connected components) is the smallest of the two. We could then think that there could be a misalignment in the threshold: when using different methods to compute the eigenvalues, different thresholds could have to be set, by empirically determining them. In any case, both methods could be practically used but taking into account that the threshold parameter has to be set accordingly. For what concerns the effects on the final clustering, let's compare the third and fourth columns of Figure 2 and 3: in general, we do not see significant differences in the clustering behaviour: both methods seem to find the same clusters. The only difference lies on the fact that sometimes the detected number of clusters is not appropriate, for example in the $k = 10$ case of 2 using Scipy functions: only 1 eigenvalue is considerable negligible, and thus only 1 cluster is generated. Of course, this behaviour confirms our recommendation on using different thresholds when using different methods.
- **What are the effects on Spectral Clustering of using different values of k-nearest neighbours?** In Figures 2

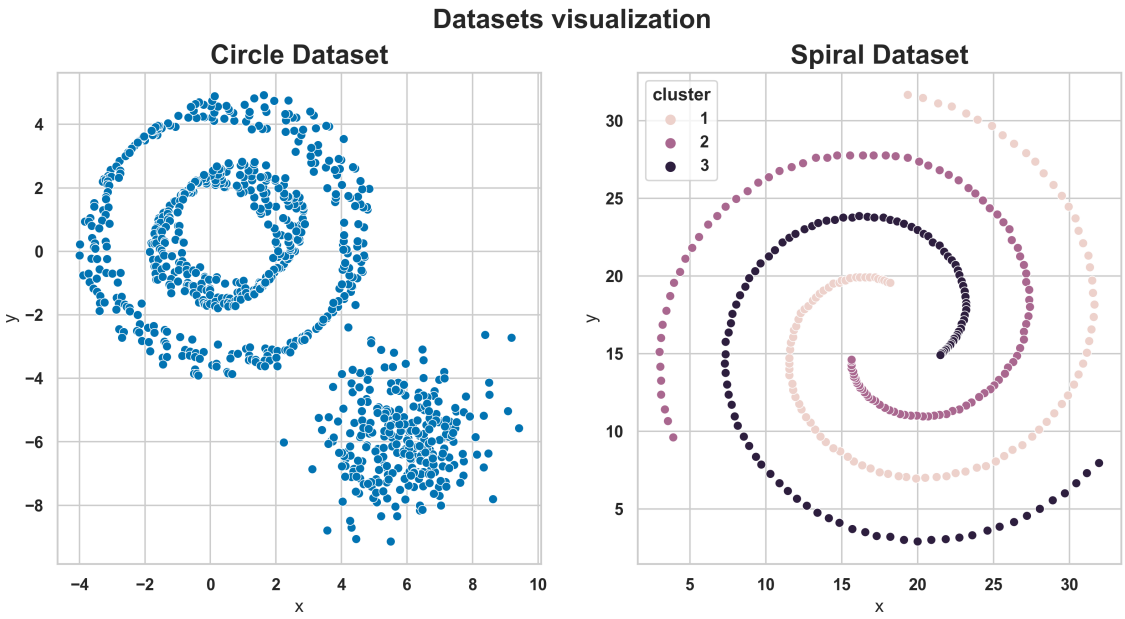

Fig. 1. Visualization of the two datasets: Circle and Spiral.

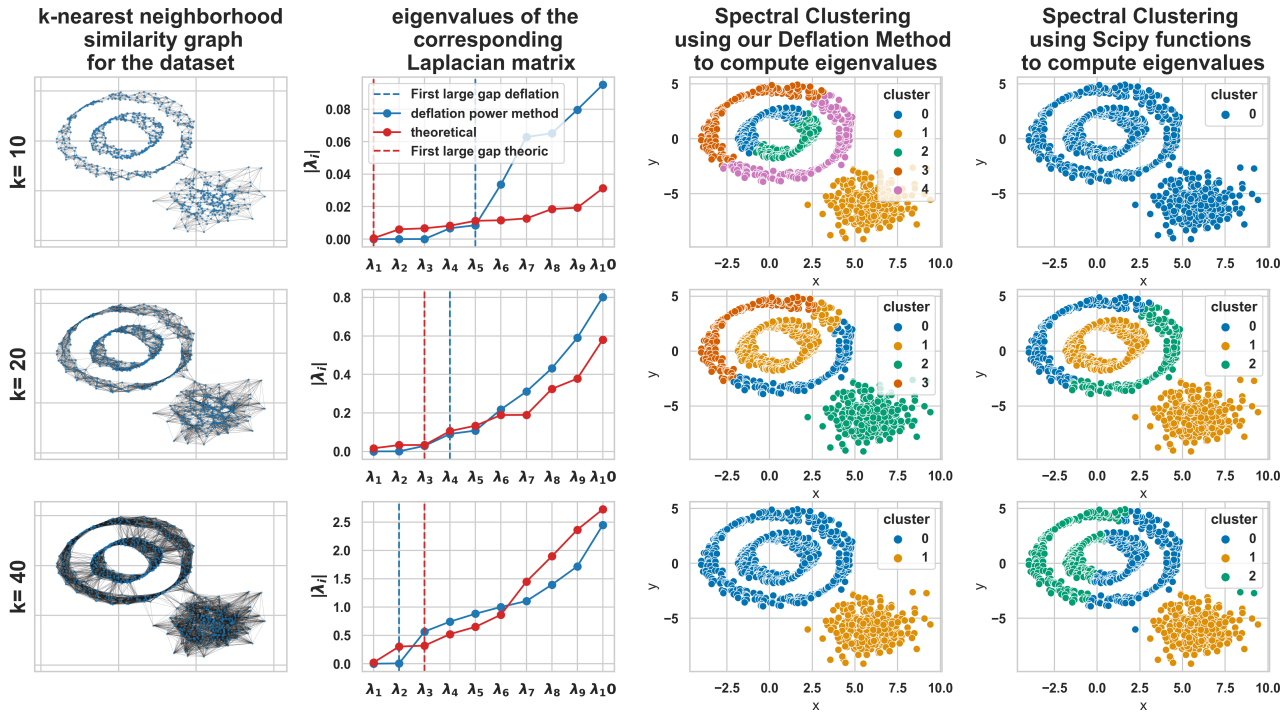

Fig. 2. Spectral Clustering workflow for the Circle dataset: k-nearest neighborhood similarity graph, plot of the eigenvalues (obtained both with our Deflation method, in blue, and with the Scipy builtin method, in red), and final clustering of the points using both results from our deflation method and from the Scipy one.

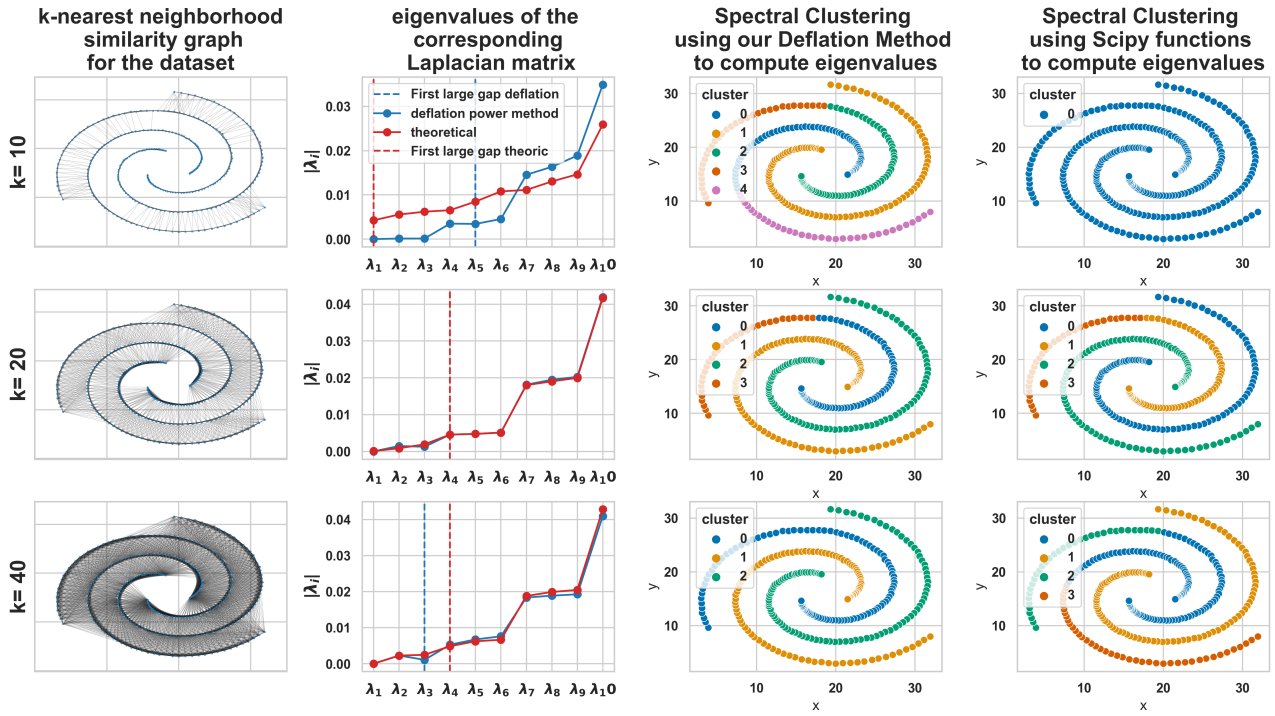

Fig. 3. Spectral Clustering workflow for the Spiral dataset: k-nearest neighborhood similarity graph, plot of the eigenvalues (obtained both with our Deflation method, in blue, and with the Scipy builtin method, in red), and final clustering of the points using both results from our deflation method and from the Scipy one.

and 3, each row is referred to a certain value of k, where k is the number of neighbours taken in consideration. Starting from the Circle dataset, in 2, let's inspect the clustering obtained with our Deflation Method (third column). Let us identify, in a human way, three groups of points in this dataset, that we here refer to as "inner circumference", "outer circumference" and "small circle on bottom right". The most visually significant clustering seems to be that obtained by selecting $k = 20$ neighbours, as the other two values of k lead to too much or too few clusters. How can we explain the fact that the outer circumference is labelled $\approx 40\%$ as cluster 0, $\approx 40\%$ as cluster 3 and $\approx 20\%$ as cluster 1? Let us take a look at its corresponding graph: we see that some connections are present in the area around the points of the outer circumference that were classified as cluster 1, and in fact they get more visible when $k = 40$. Also, the two halves of the outer circumference do not seem "heavily connected" and this can be confirmed from the graph for $k = 10$. These behaviours may influence the Laplacian matrix of the graph, even slightly: we can in fact see in the corresponding eigenvalues plot that the blue line starts increasing from zero around the value of λ_3 and λ_4. Our automatic threshold has detected the point of λ_4 as the first eigenvalue significantly different from 0, but one could argue that this point should be λ_3, which is in fact already slightly upper than λ_2, and thus one could choose 3 as the number of connected components, and thus clusters. In that case, the $\approx 20\%$ of the points of the outer circumference labelled as cluster 1 may be incorporated by the inner circumference, and the two halves of the outer circumference may be connected into one single cluster, as their light connection is still heavier that the connection there is bewtween the outer circumference and the small circle in the bottom right of the graph.

Regarding Figure 3, with similar reasoning we could conclude that, using our deflation method, the value of $k = 40$ is the optimal, as it seems to correctly detect all the three clusters present in the dataset.

- **What are the effects on Spectral Clustering of initial dataset distribution?** Here we want to address possible optimalities of the Spectral Clustering algorithm with some particular dataset. By comparing the results in Figures 2 and 3, we notice that the algorithm seems to correctly identify clusters in both cases, provided that a suitable value of k is selected. In fact, for not suitable values of k, both methods tend to separate the points in a wrong way, but we remark how the choice of the value of k is part of the algorithm, and constitutes what is in the Data Science field often referred to as a "hyperparameter". Thus, we do not report evident optimalities of the algorithm for a specific dataset, suggesting us that the

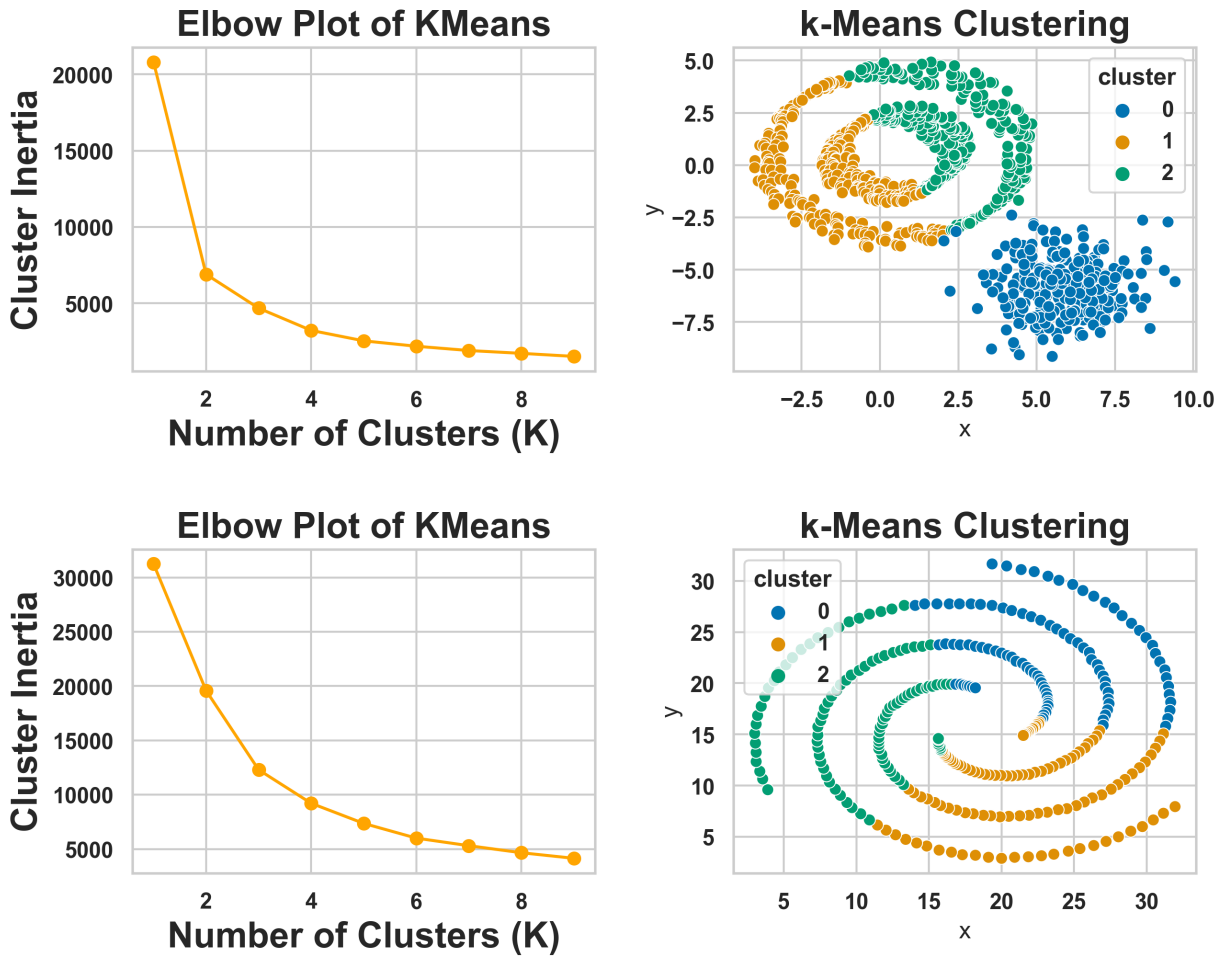

Fig. 4. kMeans Clustering of the two datasets.

algorithm may be suitable even for different datasets.

- **How does Spectral Clustering compare to k-Means Clustering?** Finally, we compare the results with that of a k-Means algorithm directly applied to the initial dataset. We can visualize the results in Figure 4, and immediately asses that, once chosen the right number of clusters visually ($k = 3$ in both cases), the performed clustering fails to get the pattern of the points for the Spiral case, while seems to get the small circle in the Circle dataset, but still failing to get the separation between the inner and the outer circle, that instead the Spectral Clustering was able to find. Furthermore, we gave a "help" to k-Means in selecting the right number of clusters by manually setting it to 3 for both datasets. It is worth noticing that, if one had to make a choice about this solely based on the Elbow Plot, different number of clusters could be set (e.g. 2, for the Circle dataset), leading to even more imprecise clustering.

V. Conclusion

In conclusion, we implemented the Spectral Clustering algorithm and performed an analysis about its performance and the factors that are most influencing it. We found out that the number of neighbours for constructing the corresponding graph has definitely an effect on the final clustering, as well as using different implementations of methods to find the smallest eigenvalues of a matrix. The performance of the algorithm seems not to be affected by the dataset distribution, but some further experimenting with other datasets could be made. Finally, we assessed the better performance, at least in these two datasets, of the Spectral Clustering with respect to the k-Means clustering algorithm.

REFERENCES

[1] William Fleshman, Towards Data Science Spectral Clustering: Foundation and Application, [Accessed 21-May-2023]; https://towardsdatascience.com/spectral-clustering-aba2640c0d5b.

[2] Scipy Documentation scipy.sparse.linalg.eigs, [Accessed 21-May-2023]; https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.eigs.html.