Spectral Clustering From Scratch

Manthan Bagade, Anay

July 2025*

1 Introduction

Spectral clustering is one of the unsupervised algorithms to group datapoints. In recent years, spectral clustering has become one of the most popular modern clustering algorithms. It is simple to implement, can be solved efficiently by standard linear algebra software, and very often outperforms traditional clustering algorithms, such as the k-means algorithm.
Here we will implement a variant of this algorithm from scratch(using only numpy). We will go through various components of the algorithm and understand them one by one. We have implemented the algorithm with the help of Python classes. The notebook - `https://github.com/manthanbagade/SpectralClustering` contains the code, comparisons and visualizations of the algorithm.

2 The need for spectral clustering

Spectral clustering is a powerful technique for clustering data that is not linearly separable. It helps us overcome two major problems in clustering: one being the shape of the cluster and the other being determining the cluster centroid. K-means algorithm generally assumes that the clusters are spherical or round, i.e. within k-radius from the cluster centroid. In K-means, the algorithm requires many iterations to determine the cluster centroid.
In spectral clustering, the clusters do not adhere to a fixed shape or pattern. The algorithm assigns points that are far apart but connected to the same cluster, while points that are close to each other could belong to different clusters if they are not connected. This suggests that the algorithm may be effective for data with varying shapes and sizes. These properties make spectral clustering a powerful tool for clustering data that is not linearly separable, and it has been shown to outperform traditional clustering algorithms in many cases. It is more efficient than k-means as it computes just the eigenpairs of the Laplacian matrix as discussed in section 5. However, it is important to note that spectral clustering can be sensitive to the choice of similarity measure and the number of clusters.

Clearly, KMeans fails to cluster the data correctly, while spectral clustering does a good job of grouping the points based on their connectivity. Now let's understand the nitty-gritty of the algorithm.

*Thanks to Coding Club, IITG for this assignment.

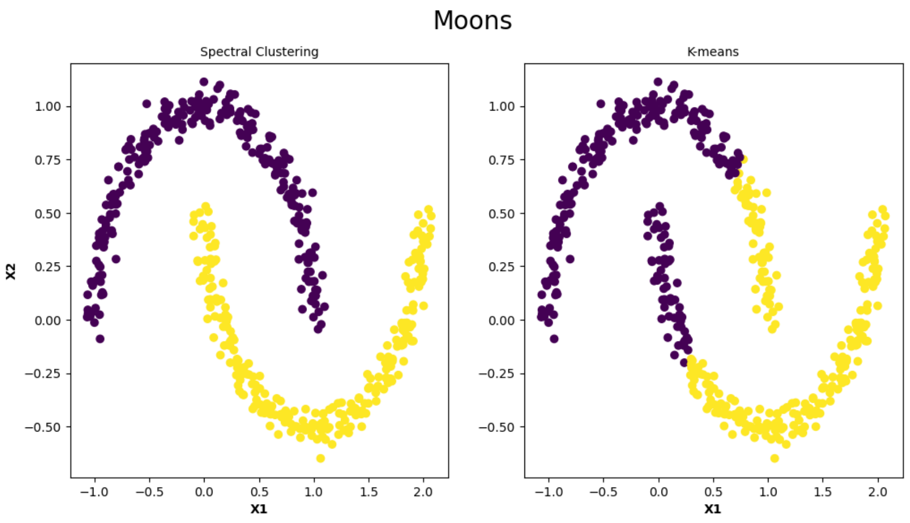

Figure 1: Comparison of Spectral Clustering and KMeans on the Moons dataset

3 Adjacency Matrix and Similarity

Given a set of data points $x_1, \ldots, x_n$ and some notion of similarity $s_{ij} \geq 0$ between all pairs of data points x_i and x_j, the intuitive goal of clustering is to divide the data points into several groups such that points in the same group are similar and points in different groups are dissimilar to each other. There are several popular constructions to transform a given set $x_1, \ldots, x_n$ of data points with pairwise similarities s_{ij} or pairwise distances d_{ij} into a graph. For this implementation, we will use the adjacency matrix A which is defined as follows:

$$s_{ij} = \begin{cases} e^{-\dfrac{\|x_i - x_j\|^2}{2\sigma^2}} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{1}$$

Here, σ is a hyperparameter that controls the connectivity of the datapoints. The greater the value of sigma, the more connected the points become,i.e, it increase the global connectivity of the graph.

The adjacency matrix A is then constructed from the similarity matrix $S = (s_{ij})$ as follows:

$$A_{ij} = \begin{cases} s_{ij} & \text{if } s_{ij} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

4 Degree Matrix

The degree matrix D is a diagonal matrix that contains the sum of the weights of the edges connected to each vertex in the graph. D represents the total connection strength of node i to all other nodes. The higher the value of D_{ii}, the higher the node i is connected to other nodes and the strongly it

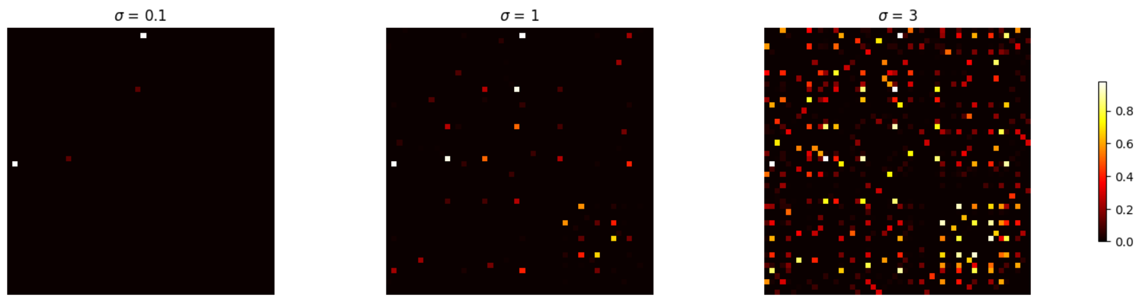

Figure 2: Comparison of σ values on the moons dataset

is connected. It is defined as follows:

$$D_{ii} = \sum_{j=1}^{n} A_{ij} \tag{3}$$

5 Laplacian Matrix

The Laplacian matrix L also known as Graph Laplacian is a matrix representation of a graph that captures the connectivity of the graph. We use the Laplacian matrix to transform the data into a lower-dimensional space (k dimensional) where the clusters can be easily separated. You select the first k eigenvectors corresponding to the smallest eigenvalues except the **trivial eigenvalue - 0**. The eigenvectors of L are sorted in ascending order of their eigenvalues, as the smallest eigenvalues give smooth, broad pattern and are not sensitive to noise. Here, we are going to use **Normalized Graph Laplacian**. It has several advantages :

1. Since it is symmetric, it has real, positive eigenvalues and orthogonal eigenvectors.
2. It is invariant to the scaling of the graph, i.e., avoids letting high degree nodes dominate the clustering.
3. Numerically stable due to normalization.

The normalized Laplacian matrix is defined as:

$$L = I - D^{-1/2} A D^{-1/2} \tag{4}$$

where I is the identity matrix, D is the degree matrix, and A is the adjacency matrix. We normalize the unnormalized Laplacian matrix with the matrix D.

$$L_{unnormalized} = D - A \tag{5}$$

The normalized Laplacian matrix is then given by:

$$\begin{aligned} L_{\text{sym}} &= D^{-1/2} L_{\text{unnormalized}} D^{-1/2} \\ &= D^{-1/2}(D - A) D^{-1/2} \\ &= I - D^{-1/2} A D^{-1/2} \end{aligned}$$

6 The Final Steps

Once we have the Laplacian matrix, we can compute the eigenvalues and eigenvectors of the Laplacian matrix. The eigenvectors corresponding to the smallest k eigenvalues (excluding the trivial eigenvalue 0) are used to form a new matrix X of size $n \times k$, where n is the number of data points and k is the number of clusters. The rows of this matrix represent the data points in the new space. We then apply a clustering algorithm, such as k-means, to the rows of this matrix to obtain the final clusters. The k-means algorithm is applied to the rows of the matrix X which are datapoints in the new space.

7 Implementation

The implementation of spectral clustering is done using Python classes. The name of the class is `SpectralClustering`, which takes the number of clusters `n_clusters` and scaling parameter `sigma` as inputs. You can call `fit_predict` method to fit the model and store the labels in a variable. Here is the implementation of the class:

```
class SpectralClustering :

  def __init__(self, n_cluster, sigma = 1.0) -> None:
    self.n_cluster = n_cluster
    self.sigma = sigma
    self.labels_ = None

  def compute_affinity_matrix(self, X) -> np.ndarray:
    affinity_matrix = np.exp(-squareform(pdist(X,
        'sqeuclidean'))/(2*self.sigma**2))
    np.fill_diagonal(affinity_matrix, 0)
    return affinity_matrix

  def compute_degree_matrix(self, affinity_matrix) -> np.ndarray:
    degree_matrix = np.diag(np.sum(affinity_matrix, axis=1))
    return degree_matrix

  def compute_laplacian_matrix(self, degree_matrix, affinity_matrix) ->
      np.ndarray:
    diag = np.diag(degree_matrix)
    diag = np.diag(1/np.sqrt(diag+1e-8))
    laplacian_matrix = np.eye(diag.shape[0]) - (diag @ affinity_matrix @
        diag)
    return laplacian_matrix

  def compute_eigenvectors(self, laplacian_matrix) -> np.ndarray:
    eigenvalues, eigenvectors = eigh(laplacian_matrix)
    return eigenvectors[:,1:self.n_cluster+1]

  def fit(self, X) -> None :
    affinity_matrix = self.compute_affinity_matrix(X)
    degree_matrix = self.compute_degree_matrix(affinity_matrix)
    laplacian_matrix = self.compute_laplacian_matrix(degree_matrix,
        affinity_matrix)
    eigenvectors = self.compute_eigenvectors(laplacian_matrix)
    eigenvectors = eigenvectors / (np.linalg.norm(eigenvectors,
        axis=1,keepdims=True)+1e-8)
    kmeans = KMeans(n_clusters=self.n_cluster,
        random_state=0).fit(eigenvectors)
    self.labels_ = kmeans.labels_

  def fit_predict(self, X) -> np.array:
    self.fit(X)
    return self.labels_
```

Listing 1: Spectral Clustering Implementation

8 Results

We have tested the implementation on two datasets - Moons and Circle. The results are shown in Figure 3 and Figure 4. The implementation is able to cluster the points correctly in both cases.

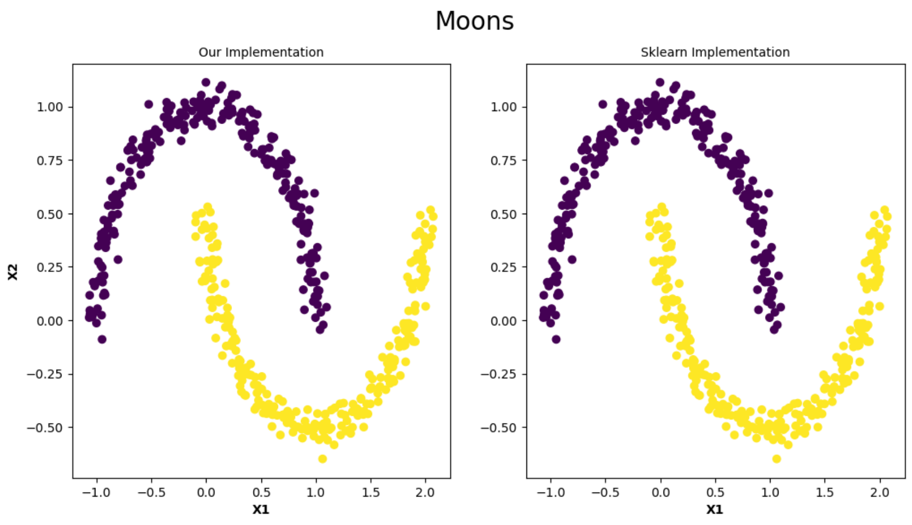

Figure 3: Spectral Clustering on Moons Dataset

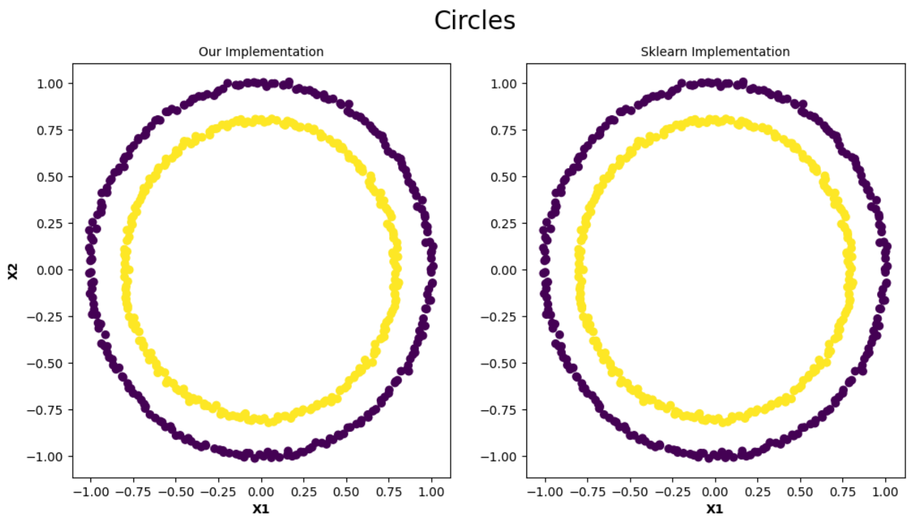

Figure 4: Spectral Clustering on Circle Dataset

References

[1] *Introduction to Spectral Clustering*, by Marina Chatterjee, July 25, 2024.

[2] A Tutorial on Spectral Clustering, by Ulrike von Luxburg, 2007.

[3] *Spectral Clustering : A comprehensive guide for begineers*, by Keerthana, Oct 14, 2024.

[4] *Spectral Clustering in Machine Learning*, GeeksforGeeks article, July 12, 2025.

[5] *Implementing Spectral Clustering from Scratch: A Step-by-Step Guide*, by Rahul Jain, May 23 2024.

[6] *3 easy steps to understand and implement spectral learning in python*, by Dr. Data Science, May 30, 2021.

[7] Spectral Graph Theory for Dummies, by Ron and Math, May 16, 2024.