**Spectral Clustering.**

In K-Means algorithm we use compactness(distance) between the data points as a characteristic to cluster our data points. However, we can also use connectivity between the data point as a feature to cluster our data points. Using connectivity, we can cluster two data points into the same clusters even if the distance between the two data points is larger.

**Spectral Clustering**

Spectral Clustering is a variant of the clustering algorithm that uses the connectivity between the data points to form the clustering. It uses eigenvalues and eigenvectors of the data matrix to forecast the data into lower dimensions space to cluster the data points. It is based on the idea of a graph representation of data where the data points are represented as nodes and the similarity between the data points are represented by an edge.

### **Steps performed for spectral Clustering:**

**1) Building the Similarity Graph of The Data:** This step builds the Similarity Graph in the form of an adjacency matrix which is represented by A. The adjacency matrix can be built in the following manners:

* **Epsilon-neighborhood Graph:** A parameter epsilon is fixed beforehand. Then, each point is connected to all the points which lie in its epsilon-radius. If all the distances between any two points are similar in scale then typically the weights of the edges ie the distance between the two points are not stored since they do not provide any additional information. Thus, in this case, the graph built is an undirected and unweighted graph.
* **K-Nearest Neighbors** A parameter k is fixed beforehand. Then, for two vertices u and v, an edge is directed from u to v only if v is among the k-nearest neighbors of u. Note that this leads to the formation of a weighted and directed graph because it is not always the case that for each u having v as one of the k-nearest neighbors, it will be the same case for v having u among its k-nearest neighbors. To make this graph undirected, one of the following approaches is followed:
  1. Direct an edge from u to v and from v to u if either v is among the k-nearest neighbors of u **OR** u is among the k-nearest neighbors of v.
  2. Direct an edge from u to v and from v to u if v is among the k-nearest neighbors of u **AND** u is among the k-nearest neighbors of v.
  3. **Fully-Connected Graph:** To build this graph, each point is connected with an undirected edge-weighted by the distance between the two points to every other point. Since this approach is used to model the local neighborhood relationships thus typically the Gaussian similarity metric is used to calculate the distance.

**2) Projecting the data onto a lower Dimensional Space:** This step is done to account for the possibility that members of the same cluster may be far away in the given dimensional space. Thus, the dimensional space is reduced so that those points are closer in the reduced dimensional space and thus can be clustered together by a traditional clustering algorithm. It is done by computing the **Graph Laplacian Matrix**.

This Matrix is then normalized for mathematical efficiency. To reduce the dimensions, first, the eigenvalues and the respective eigenvectors are calculated. If the number of clusters is k then the first eigenvalues and their eigenvectors are taken and stacked into a matrix such that the eigenvectors are the columns.

**3)** **Clustering the Data:** This process mainly involves clustering the reduced data by using any traditional clustering technique – typically K-Means Clustering. First, each node is assigned a row of the normalized of the Graph Laplacian Matrix. Then this data is clustered using any traditional technique. To transform the clustering result, the node identifier is retained.

**Properties:**

1. **Assumption-Less:** This clustering technique, unlike other traditional techniques do not assume the data to follow some property. Thus this makes this technique to answer a more-generic class of clustering problems.
2. **Ease of implementation and Speed:** This algorithm is easier to implement than other clustering algorithms and is also very fast as it mainly consists of mathematical computations.
3. **Not-Scalable:** Since it involves the building of matrices and computation of eigenvalues and eigenvectors it is time-consuming for dense datasets.
4. **Dimensionality Reduction**: The algorithm uses eigenvalue decomposition to reduce the dimensionality of the data, making it easier to visualize and analyze.
5. **Cluster Shape**: This technique can handle non-linear cluster shapes, making it suitable for a wide range of applications.
6. **Noise Sensitivity**: It is sensitive to noise and outliers, which may affect the quality of the resulting clusters.
7. **Number of Clusters:**The algorithm requires the user to specify the number of clusters beforehand, which can be challenging in some cases.
8. **Memory Requirements:** The algorithm requires significant memory to store the similarity matrix, which can be a limitation for large datasets.

**Parameters:**

**https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html**

In practice Spectral Clustering is very useful when the structure of the individual clusters is highly non-convex, or more generally when a measure of the center and spread of the cluster is not a suitable description of the complete cluster, such as when clusters are nested circles on the 2D plane.

**n\_clusters *int, default=8***

The dimension of the projection subspace.

**eigen\_solver*{‘arpack’, ‘lobpcg’, ‘amg’}, default=None***

The eigenvalue decomposition strategy to use. AMG requires pyamg to be installed. It can be faster on very large, sparse problems, but may also lead to instabilities. If None, then 'arpack' is used. See [[4]](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#r5f6cbeb1558e-4) for more details regarding 'lobpcg'.

**n\_components *int, default=None***

Number of eigenvectors to use for the spectral embedding. If None, defaults to n\_clusters.

**random\_state *int, RandomState instance, default=None***

A pseudo random number generator used for the initialization of the lobpcg eigenvectors decomposition when eigen\_solver == 'amg', and for the K-Means initialization. Use an int to make the results deterministic across calls (See [Glossary](https://scikit-learn.org/stable/glossary.html#term-random_state)).

**Note**

When using eigen\_solver == 'amg', it is necessary to also fix the global numpy seed with np.random.seed(int) to get deterministic results. See <https://github.com/pyamg/pyamg/issues/139> for further information.

**n\_init *int, default=10***

Number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of n\_init consecutive runs in terms of inertia. Only used if assign\_labels='kmeans'.

**Gamma *float, default=1.0***

Kernel coefficient for rbf, poly, sigmoid, laplacian and chi2 kernels. Ignored for affinity='nearest\_neighbors'.

**Affinity *str or callable, default=’rbf’***

**How to construct the affinity matrix.**

* ‘nearest\_neighbors’: construct the affinity matrix by computing a graph of nearest neighbors.
* ‘rbf’: construct the affinity matrix using a radial basis function (RBF) kernel.
* ‘precomputed’: interpret X as a precomputed affinity matrix, where larger values indicate greater similarity between instances.
* ‘precomputed\_nearest\_neighbors’: interpret X as a sparse graph of precomputed distances, and construct a binary affinity matrix from the n\_neighbors nearest neighbors of each instance.
* one of the kernels supported by **[pairwise\_kernels](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.pairwise_kernels.html" \l "sklearn.metrics.pairwise.pairwise_kernels" \o "sklearn.metrics.pairwise.pairwise_kernels)**.

Only kernels that produce similarity scores (non-negative values that increase with similarity) should be used. This property is not checked by the clustering algorithm.

**n\_neighbors *int, default=10***

Number of neighbors to use when constructing the affinity matrix using the nearest neighbors method. Ignored for affinity='rbf'.

**eigen\_tol *float, default=”auto”***

Stopping criterion for eigen decomposition of the Laplacian matrix. If eigen\_tol="auto" then the passed tolerance will depend on the eigen\_solver:

* If eigen\_solver="arpack", then eigen\_tol=0.0;
* If eigen\_solver="lobpcg" or eigen\_solver="amg", then eigen\_tol=None which configures the underlying lobpcg solver to automatically resolve the value according to their heuristics. See, **[scipy.sparse.linalg.lobpcg](https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.lobpcg.html" \l "scipy.sparse.linalg.lobpcg" \o "(in SciPy v1.11.3))** for details.

Note that when using eigen\_solver="lobpcg" or eigen\_solver="amg" values of tol<1e-5 may lead to convergence issues and should be avoided.

*New in version 1.2:*Added ‘auto’ option.

**assign\_labels *{‘kmeans’, ‘discretize’, ‘cluster\_qr’}, default=’kmeans’***

The strategy for assigning labels in the embedding space. There are two ways to assign labels after the Laplacian embedding. k-means is a popular choice, but it can be sensitive to initialization. Discretization is another approach which is less sensitive to random initialization [[3]](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#r5f6cbeb1558e-3). The cluster\_qr method [[5]](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html#r5f6cbeb1558e-5) directly extract clusters from eigenvectors in spectral clustering. In contrast to k-means and discretization, cluster\_qr has no tuning parameters and runs no iterations, yet may outperform k-means and discretization in terms of both quality and speed.

*Changed in version 1.1:*Added new labeling method ‘cluster\_qr’.

**Degree *float, default=3***

Degree of the polynomial kernel. Ignored by other kernels.

**coef0 *float, default=1***

Zero coefficient for polynomial and sigmoid kernels. Ignored by other kernels.

**kernel\_params *dict of str to any, default=None***

Parameters (keyword arguments) and values for kernel passed as callable object. Ignored by other kernels.

**n\_jobs *int, default=None***

The number of parallel jobs to run when affinity='nearest\_neighbors' or affinity='precomputed\_nearest\_neighbors'. The neighbors search will be done in parallel. None means 1 unless in a **[joblib.parallel\_backend](https://joblib.readthedocs.io/en/latest/generated/joblib.parallel_backend.html" \l "joblib.parallel_backend" \o "(in joblib v1.4.dev0))** context. -1 means using all processors.

**Verbose *bool, default=False***

Spectral Clustering is a type of clustering algorithm in machine learning that uses eigenvectors of a similarity matrix to divide a set of data points into clusters. The basic idea behind spectral clustering is to use the eigenvectors of the Laplacian matrix of a graph to represent the data points and find clusters by applying k-means or another clustering algorithm to the eigenvectors.

### **Advantages of Spectral Clustering:**

1. Scalability: Spectral clustering can handle large datasets and high-dimensional data, as it reduces the dimensionality of the data before clustering.
2. Flexibility: Spectral clustering can be applied to non-linearly separable data, as it does not rely on traditional distance-based clustering methods.
3. Robustness: Spectral clustering can be more robust to noise and outliers in the data, as it considers the global structure of the data, rather than just local distances between data points.

### **Disadvantages of Spectral Clustering:**

1. Complexity: Spectral clustering can be computationally expensive, especially for large datasets, as it requires the calculation of eigenvectors and eigenvalues.
2. Model selection: Choosing the right number of clusters and the right similarity matrix can be challenging and may require expert knowledge or trial and error.