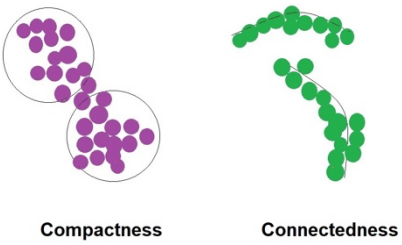
What is clustering?

Clustering is a widely used unsupervised learning method. The grouping is such that points in a cluster are similar to each other, and less similar to points in other clusters. Thus, it is up to the algorithm to find patterns in the data and group it for us and, depending on the algorithm used, we may end up with different clusters.

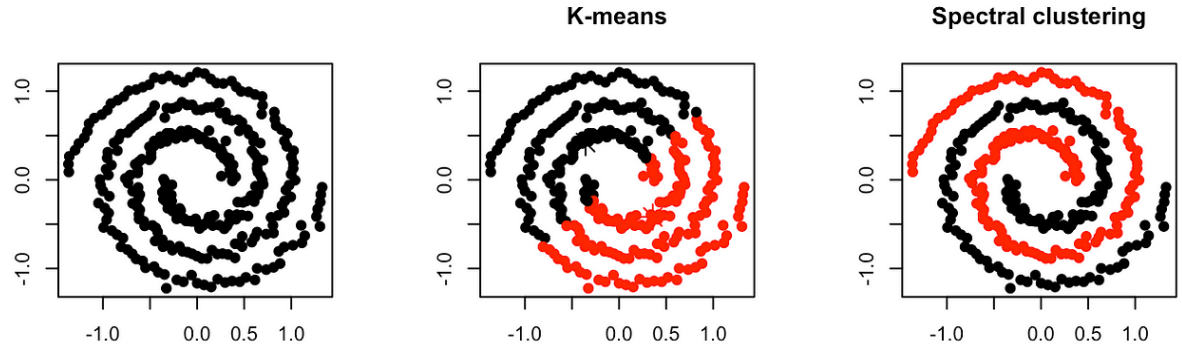
There are 2 main broad approaches for clustering:

1. Compactness — Points that lie close to each other fall in the same cluster and are compact around the cluster center. The closeness can be measured by the distance between the observations. E.g.: K-Means Clustering.

2. Connectivity — Points that are connected or immediately next to each other are put in the same cluster. Even if the distance between 2 points is less, if they are not connected, they are not clustered together. Spectral clustering is a technique that follows this approach.



The difference between the 2 can easily be shown by this illustration:



Spectral Clustering

Spectral clustering is a technique used in machine learning and data analysis for clustering or partitioning a dataset into meaningful groups or clusters. In spectral clustering, the data points are treated as nodes of a graph. Thus, clustering is treated as a graph partitioning problem. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters. An important point to note is that no assumption is made about the shape/form of the clusters.

A) In which cases might it be more useful to apply?

Can be particularly useful in various scenarios and types of data where other traditional clustering methods like K-means may struggle, such as

1. Non-convex Clusters:
   Spectral clustering can effectively handle non-convex or oddly shaped clusters, which K-means may struggle with due to its assumption of convex clusters.

2. Graph-based Data:
   Spectral clustering is well-suited for graph-based data or data that can be represented as a graph, such as social networks, citation networks, or any data with an underlying graph structure.

3. Manifold Data:
   Data often lie on a low-dimensional manifold embedded in a high-dimensional space. Spectral clustering can identify and exploit this underlying manifold structure to cluster the data accurately.

4. Disconnected Clusters:
   Spectral clustering can successfully detect disconnected or sparsely connected clusters in the data, making it suitable for applications where clusters are not tightly packed.

5. Spectral Analysis Applications:
   It is useful in various applications related to spectral analysis, such as image segmentation, where pixels can be represented as nodes in a graph and clustered based on similarity.

6. Data with Noise or Outliers:
   Spectral clustering is more robust to noise or outliers in the data compared to K-means, making it a good choice for datasets that might have some level of noise.

7. High-dimensional Data:
   In high-dimensional spaces, traditional distance-based clustering methods like K-means can be ineffective. Spectral clustering, through dimensionality reduction using eigenvectors, can be more effective in high-dimensional data analysis.

8. Unevenly Sized Clusters:
   Spectral clustering doesn't assume equal-sized clusters, making it suitable for datasets where clusters might have varying sizes.

9. Imbalanced Data:

Spectral clustering can handle imbalanced datasets where the number of data points in different classes or clusters is significantly different.

10. Applications Requiring Preprocessing or Embedding:
    If preprocessing steps like affinity computation, similarity computation, or embedding the data into a different space are necessary, spectral clustering provides a flexible framework to incorporate these steps.

B) What are the mathematical fundamentals of it?

The mathematical fundamentals of spectral clustering involve linear algebra, graph theory, and optimization. The key mathematical components and steps involved in spectral clustering are

1. Project your data into R^n
2. Define an A*ffinity* matrix *A*, using a Gaussian Kernel *K* or say just an Adjacency matrix (i.e., $A_{i,j} = \delta_{i,j}$)
3. Construct the Graph Laplacian from *A* (i.e., decide on a normalization)
4. Solve an Eigenvalue problem, such as $Lv = \lambda v$ (or a Generalized Eigenvalue problem $v = \lambda Dv$)
5. Select k eigenvectors $\{v_i i = 1, k\}$ corresponding to the *k* lowest (or highest) eigenvalues $\{\lambda_i i = 1, k\}$, to define a k-dimensional subspace $P^t LP$
6. Form clusters in this subspace using, say, k-means

Spectral clustering leverages the spectral properties of the Laplacian matrix to embed the data into a lower-dimensional space where clustering is more apparent. The eigenvectors corresponding to the smallest eigenvalues capture the underlying structure of the data in this reduced space, making subsequent clustering more effective.

The choice of Laplacian matrix (unnormalized, normalized, or random walk) and the number of clusters (*k*) often involve domain knowledge or heuristics, such as the eigengap heuristic for determining k.

C) What is the algorithm to compute it?

Spectral clustering algorithm in Python can be used with scikit-learn class “SpectralClustering”. Here's an overview of the key parameters and attributes associated with the class:

Parameters:

- n_clusters: (int, default=8) - The number of clusters to form. This needs to be specified by the user.
- affinity: (string or callable, default='rbf') - The affinity metric used to compute the affinity matrix. It can be one of the following:
    - 'nearest_neighbors': Uses the n_neighbors nearest neighbors of each sample to compute the affinity matrix.
    - 'rbf': Uses the Radial Basis Function (RBF) kernel to compute pairwise similarities.
    - 'precomputed': Assumes that affinity is precomputed and passed as an input.
    - A callable that accepts a 2D array and returns a pairwise affinity matrix.
- gamma: (float, default=1.0) - Parameter for RBF kernel. Higher values of gamma will result in a more complex decision boundary.
- n_neighbors: (int, default=10) - Number of nearest neighbors for nearest neighbor affinity.
- eigen_solver: ({'arpack', 'lobpcg', 'amg'}, default=None) - Eigenvalue decomposition strategy. 'arpack' and 'lobpcg' are efficient for large datasets, while 'amg' is faster for small datasets.
- random_state: (int, RandomState instance, default=None) - Seed or random number generator for random initialization.

Attributes:

- labels_: (array, shape [n_samples]) - Cluster labels for each point.
- affinity_matrix_: (array-like, shape [n_samples, n_samples]) - Affinity matrix used for clustering. This is available only if the affinity parameter is set to 'precomputed'.

Methods:

- fit(X): Fit the spectral clustering model to the input data X.
- fit_predict(X): Fit the model and predict the cluster assignments for the samples in X.
- fit_transform(X): Fit the model and transform X, returning the cluster assignments.
- fit_predict_proba(X): Not implemented for spectral clustering.

D) Does it hold any relation to some of the concepts previously mentioned in class? Which, and how?

Spectral clustering and dimensionality reduction techniques are related in terms of the mathematical principles involved, particularly in terms of eigenvectors and the embedding of data into a lower-dimensional space.

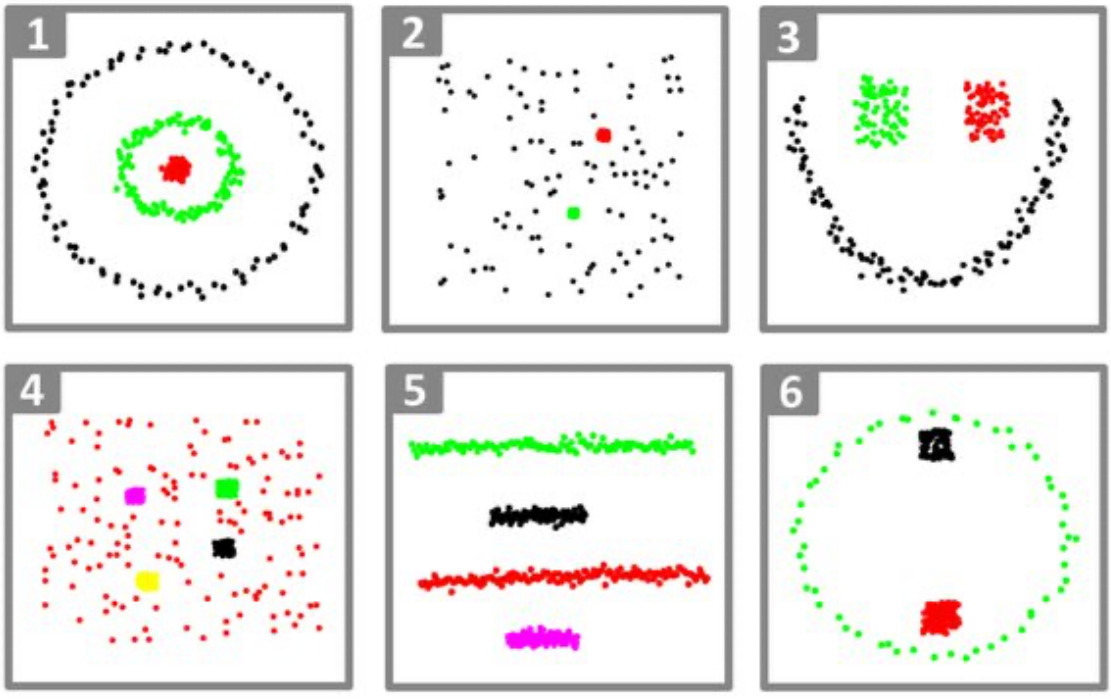
In spectral clustering, the process often involves constructing a similarity (affinity) matrix based on pairwise distances or affinities between data points. This matrix is then used to create a graph representation, where the data points are nodes and edges represent the relationships between the points (usually based on the affinity matrix).

The eigenvectors (eigenfunctions) associated with the graph's Laplacian matrix are computed, and these eigenvectors correspond to lower-dimensional representations of the data. The idea is that these eigenvectors capture the underlying structure or clustering of the data in a more compact and lower-dimensional space.

This process of obtaining eigenvectors and using them to embed the data into a lower-dimensional space is analogous to dimensionality reduction. The eigenvectors represent a reduced set of features or dimensions that capture essential information about the data's structure.

E) K-means or spectral clustering?

K-means will fail to effectively cluster these, even when the true number of clusters K is known to the algorithm.



This is because K-means, as a *data-clustering* algorithm, is ideal for discovering globular clusters like the ones shown below, where all members of each cluster are near each other (in the Euclidean sense).

In contrast to *data-clustering*, we have *graph-clustering* techniques such as spectral clustering, where you don't cluster data points directly in their native data space but instead form a similarity matrix where the (i, j)-th entry is some similarity distance you define between the i-th and j-th data points in your dataset.

So, in a sense, spectral clustering is more general (and powerful) because whenever K-means is appropriate for use then so too is spectral clustering (just use a simple Euclidean distance as the similarity measure). Understanding the nature of the data and the specific clustering requirements of the problem at hand will guide the choice of whether to apply spectral clustering.

DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a density-based clustering non-parametric algorithm given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away).

A) In which cases might it be more useful to apply?

DBSCAN is especially useful for working with outliers or anomalies and correctly detecting these outliers and points that stand out. DBSCAN clustering does not need the number of clusters to be specified prior to plotting and only needs two parameters, minPoints (radiuses of the circles that are built around every data point) and epsilons (determine the number of data points that are required inside the plotted circle).

1. Arbitrarily Shaped Clusters:
   DBSCAN can effectively identify clusters of various shapes (e.g., circular, elliptical, irregular) because it doesn't assume any particular cluster shape. This is beneficial when dealing with data where clusters might have non-convex or complex shapes.

2. Unknown or Variable Number of Clusters:
   When you don't have prior knowledge about the number of clusters in your data, DBSCAN is a great choice. It automatically determines the number of clusters based on the data's density distribution, unlike algorithms like k-means that require specifying the number of clusters beforehand.

3. Data with Noise and Outliers:
   DBSCAN is robust to noise and outliers because it classifies points that are not part of any cluster as noise. It's also capable of handling regions of lower point density as potential noise.

4. Non-Uniform Density Data:
   DBSCAN can identify clusters in data where the point density varies. It's effective at identifying both dense and sparse regions in the dataset.

5. Large Databases and High-Dimensional Data:
   DBSCAN can handle large datasets efficiently, making it suitable for applications involving big data. It's also applicable to high-dimensional data, making it versatile for various domains.

6. Spatial Data and Geographical Clustering:
   DBSCAN is widely used in geographical information systems (GIS) and location-based services. It can identify spatial clusters, such as finding hotspots of crime or clustering GPS data.

7. Anomaly Detection:
   By considering noise points as anomalies or outliers, DBSCAN can be used for anomaly detection. Points that are not part of any cluster or fall in sparse areas can be flagged as potential anomalies.

8. Customer Segmentation and Marketing:
   DBSCAN can be used to segment customers based on their behaviors and preferences. It's useful for clustering customers into groups for targeted marketing or personalized recommendations.

Understanding the nature of your data and the underlying problem you're trying to solve is crucial in determining if DBSCAN is a suitable clustering algorithm for your specific application. It's often a good idea to experiment with DBSCAN and potentially compare its performance with other clustering algorithms based on the characteristics of your dataset and your analysis objectives.

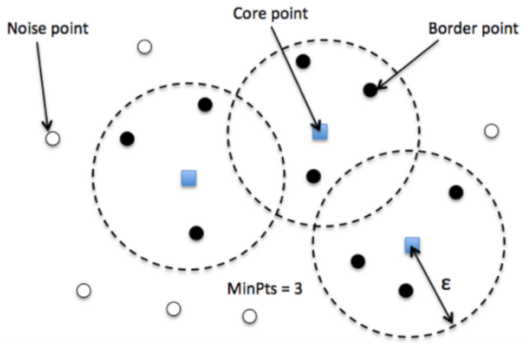
B) What are the mathematical fundamentals of it?

The DBSCAN algorithm is based on the concept of density reachability and density connectivity.

1. Density Reachability
   A point "p" is said to be density reachable from a point "q" if point "p" is within ε distance from point "q" and "q" has enough points in its neighbors which are within distance ε.

2. Density Connectivity
   A point "p" and "q" are said to be density connected if there exists a point "r" which has enough points in its neighbors and both the points "p" and "q" is within the ε distance. This is a chaining process. So, if "q" is neighbor of "r", "r" is neighbor of "s", "s" is neighbor of "t" which in turn is neighbor of "p" implies that "q" is neighbor of "p".

3. Density Conectivity
   Core point: a point *p* is a core point if there are at least a minimum point (including p) in its ε-neighborhood.
   Border point: A point *q* is a border point if it is within ε-distance of a core point but does not have enough points in its ε-neighborhood to be a core point.
   Noise point: A point that is neither a core point nor a border point.



C) What is the algorithm to compute it?

Let X = {x1, x2, x3, …, xn} be the set of data points. DBSCAN requires two parameters: ε (eps) and the minimum number of points required to form a cluster (minPts).

The steps followed by the algorithm are:

1. Start with an arbitrary starting point that has not been visited.
2. Extract the neighborhood of this point using ε (All points which are within the ε distance are neighborhood).

3. If there are sufficient neighborhoods around this point, then the clustering process starts and the point is marked as visited else this point is labeled as noise (Later this point can become the part of the cluster).
4. If a point is found to be a part of the cluster, then its ε neighborhood is also the part of the cluster and the above procedure from step 2 is repeated for all ε neighborhood points. This is repeated until all points in the cluster is determined.
5. A new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise.
6. This process continues until all points are marked as visited.

The algorithm marks each point as belonging to a specific cluster or as noise based on the density connectivity and reachability properties. The resulting clusters are sets of density-connected points, and any remaining unvisited points are treated as noise.

D) Is there any relation between DBSCAN and Spectral Clustering?

They both are popular clustering algorithms, but they operate on different principles and are suited for different types of data and clustering tasks.

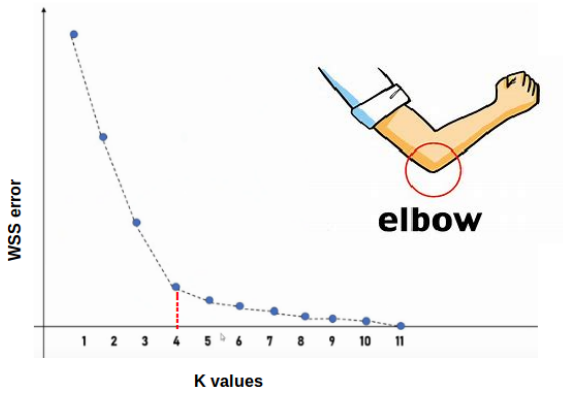
DBSCAN is a density-based clustering algorithm while Spectral Clustering is a graph-based clustering. DBSCAN identifies clusters by connecting data points based on their proximity and density while Spectral Clustering treats data points as nodes in a graph and uses graph theory to identify clusters.

Elbow Method

What is the elbow method in clustering?
The elbow method is a graphical representation of finding the optimal 'K' clusters. It works by finding WCSS (Within-Cluster Sum of Square) i.e. the sum of the square distance between points in a cluster and the cluster centroid.

The elbow graph shows WCSS values (on the y-axis) corresponding to the different values of K (on the x-axis). When we see an elbow shape in the graph, we pick the K-value where the elbow gets created. We can call this point the Elbow point.

Elbow method



The intuition is that increasing the number of clusters will naturally improve the fit (explain more of the variation), since there are more parameters (more clusters) to use, but that at some point this is over-fitting, and the elbow reflects this.

Which flaws does it pose to assess quality?
It's important to recognize its limitations and potential flaws when assessing the quality of clustering

- Determining the "elbow" point is somewhat subjective and can vary depending on the person analyzing the plot.
- In some cases, the plot may not exhibit a clear "elbow" or a point of abrupt curvature change
- The elbow method assumes a linear relationship between the number of clusters and the within-cluster sum of squares. However, the relationship may not always be linear, especially in complex or non-convex data distributions.
- The shape of the clusters and the density of the data points can affect the appearance of the elbow.