CLUSTERING

TYPES OF CLUSTERING

- **Centroid-Based Clustering**
  - K-means clustering
- **Density-Based Clustering**
  - DBSCANS
  - OPTICS
- **Hierarchical Clustering**
  - Divisive Clustering
  - Agglomerative Clustering
- **Distribution-Based Clustering**
- **Fuzzy Clustering**

Affinity Propagation

Affinity Propagation creates clusters by sending messages between pairs of samples until convergence.

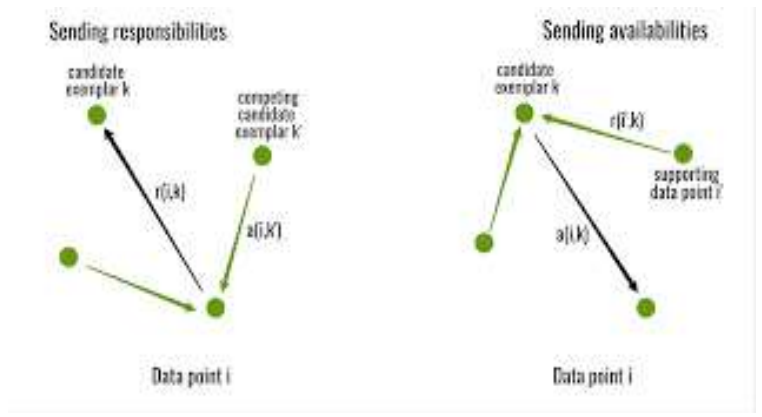
A dataset is then described using a small number of exemplars, which are identified as those most representative of other samples.

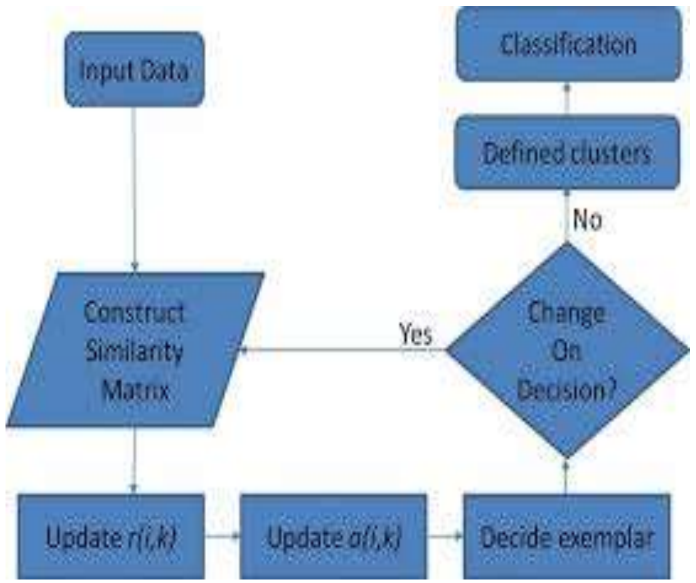
The messages sent between pairs represent the suitability for one sample to be the exemplar of the other, which is updated in response to the values from other pairs.

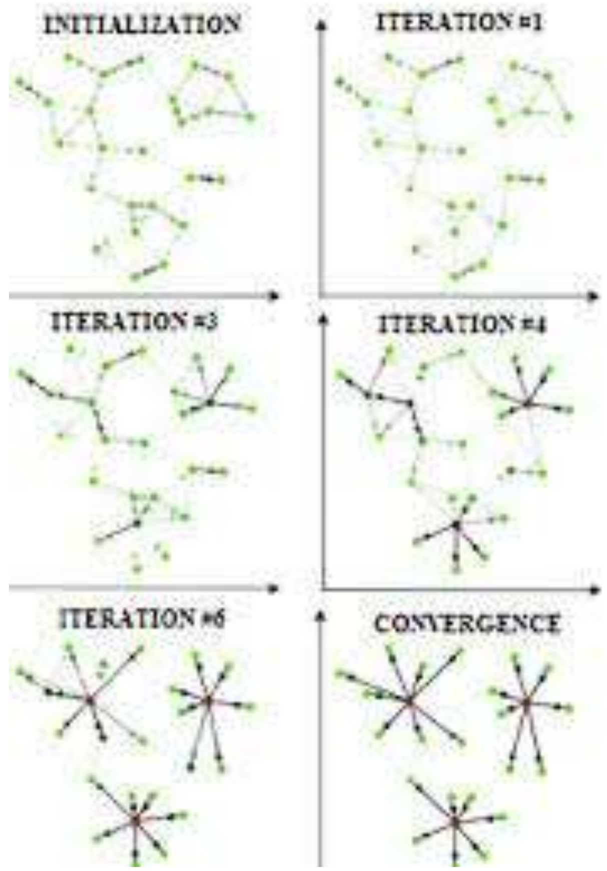
This updating happens iteratively until convergence, at which point the final exemplars are chosen, and hence the final clustering is given.

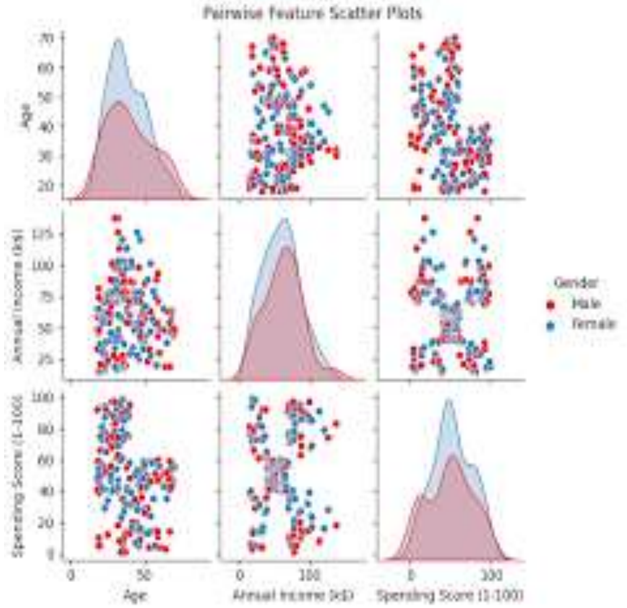
The two important parameters are the ***preference***, which controls how many exemplars are used,

The ***damping factor*** which damps the responsibility and availability messages to avoid numerical oscillations when updating these messages.


Sending responsibilities
candidate exemplar k
competing candidate exemplar k'
r(i,k)
a(i,k')
Data point i
Sending availabilities
candidate exemplar k
r(i',k)
supporting data point i'
a(i,k)
Data point i


Input Data
Construct Similarity Matrix
Update r(i,k)
Update a(i,k)
Decide exemplar
Change On Decision?
Yes
No
Defined clusters
Classification


INITIALIZATION
ITERATION #1
ITERATION #3
ITERATION #4
ITERATION #6
CONVERGENCE


Pairwise Feature Scatter Plots
Age
Annual Income (k$)
Spending Score (1-100)
Gender
Male
Female

Affinity Propagation Clustering and performance evaluation

The required parameters are discussed below:

1. **Preference**: The preference parameter plays an important role and represents the input that indicates how likely a data point is to be chosen as an exemplar. It influences the **number of clusters that will be formed** and the data points that will be assigned as exemplars

2. **max_iter:** This parameter determines the maximum number of iterations (passes over the data).

3. **convergence_iter:** This parameter controls the number of iterations with no change in the cluster assignments which must occur for the algorithm to consider the solution as converged.

4. **random_state:** This parameter handles the randomness which ensures the initialization of cluster centers and the order of processing data points remains same between different runs.

5. **damping:** It controls the convergence speed of the algorithm which it set to 0.9(close to 1) means that algorithm converges more slowly.

Mean Shift

Mean Shift aims to discover *blobs* in a smooth density of samples.

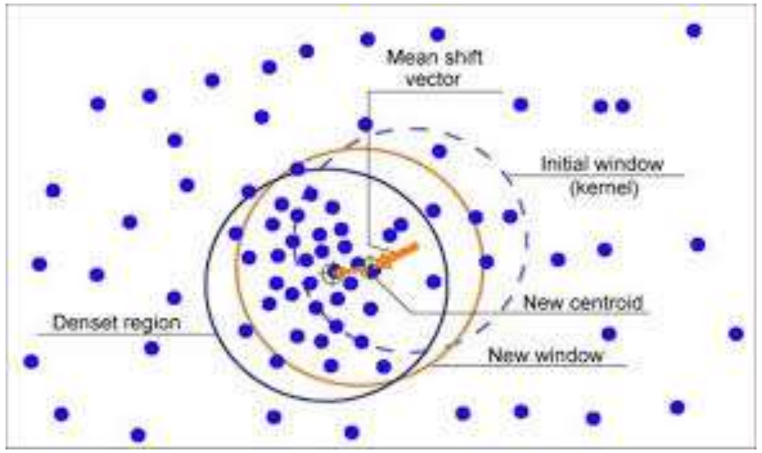
It is a centroid based algorithm, which works by updating candidates for centroids to be the mean of the points within a given region.

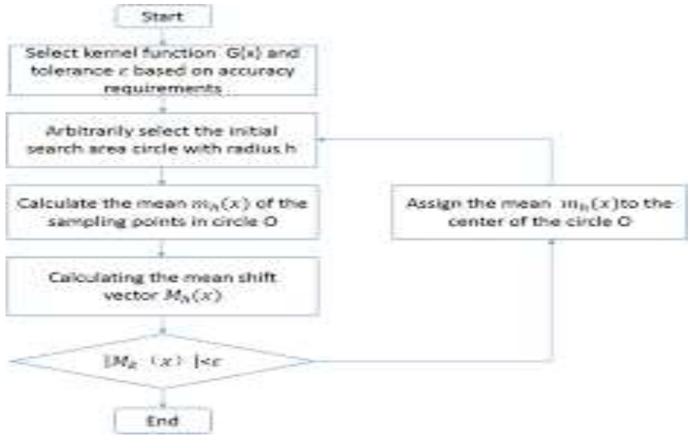
These candidates are then filtered in a post-processing stage to eliminate near-duplicates to form the final set of centroids.

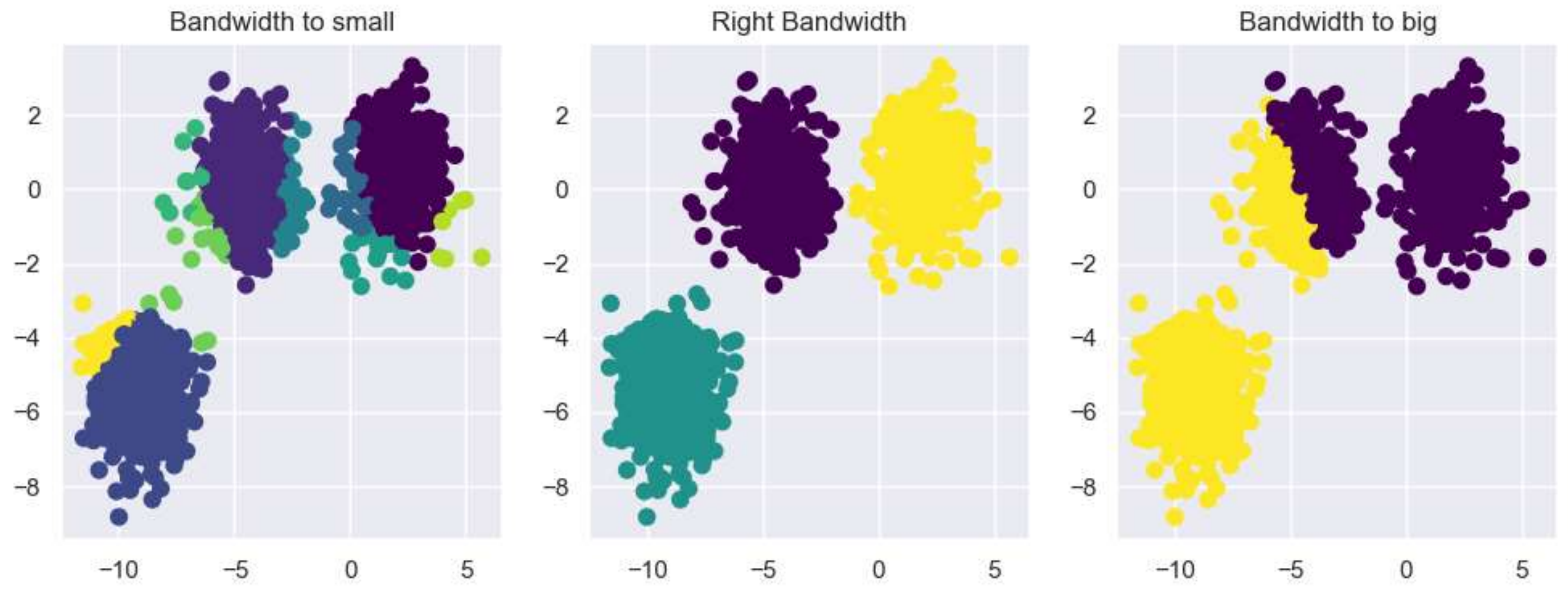
The algorithm automatically sets the number of clusters, instead of relying on a parameter bandwidth, which dictates the size of the region to search through. This parameter can be set manually, but can be estimated using the provided estimate_bandwidth function, which is called if the bandwidth is not set.

The algorithm is not highly scalable, as it requires multiple nearest neighbor searches during the execution of the algorithm

The algorithm is guaranteed to converge, however the algorithm will stop iterating when the change in centroids is small.
Labelling a new sample is performed by finding the nearest centroid for a given sample.


Mean shift vector
Initial window (kernel)
New centroid
Denset region
New window


Start
Select kernel function G(x) and tolerance ε based on accuracy requirements
Arbitrarily select the initial search area circle with radius h
Calculate the mean $m_h(x)$ of the sampling points in circle O
Calculating the mean shift vector $M_h(x)$
$|M_h(x)|<\varepsilon$
Assign the mean $m_h(x)$ to the center of the circle O
End

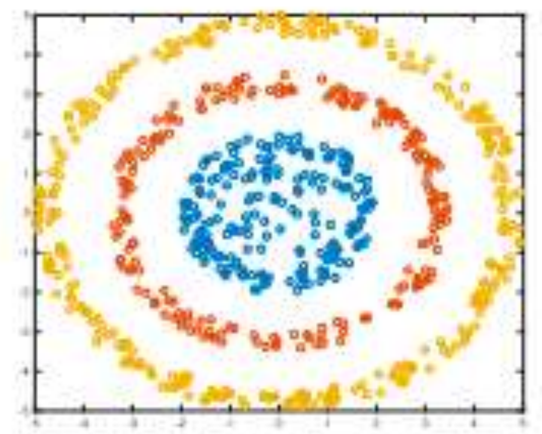

Bandwidth to small
Right Bandwidth
Bandwidth to big
2
0
−2
−4
−6
−8
−10
−5
0
5

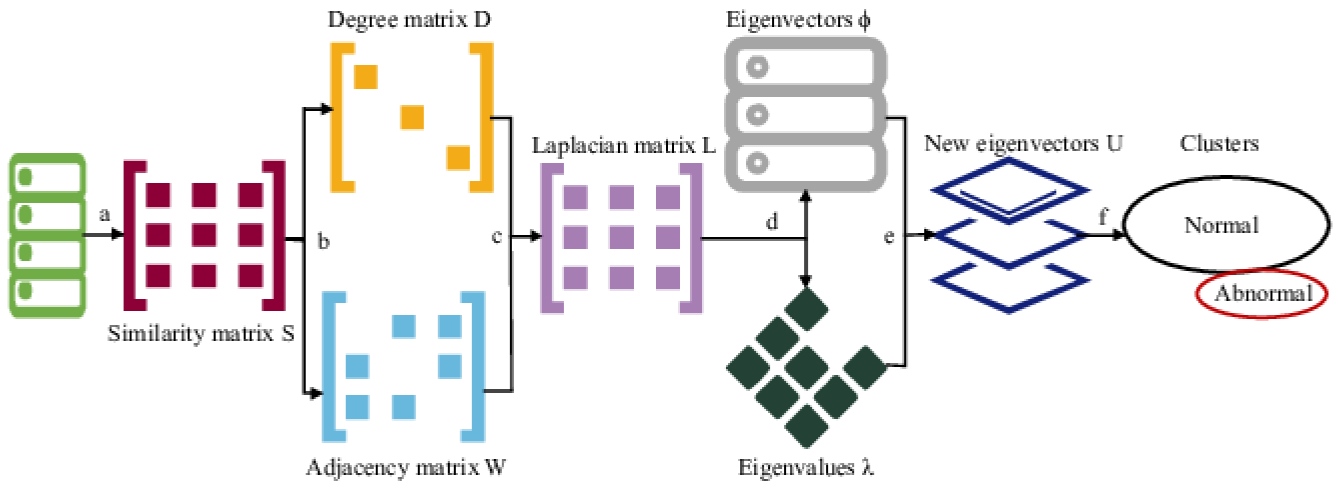
The process of mean-shift clustering algorithm can be summarized as follows:

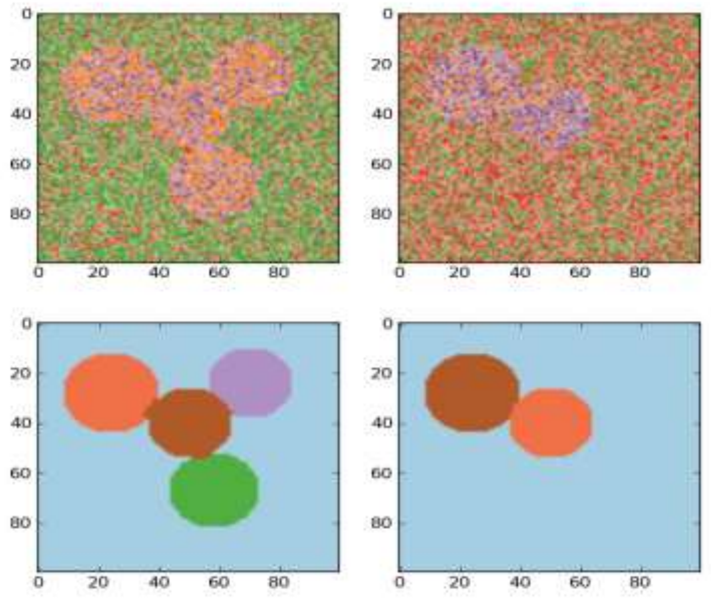
- Initialize the data points as cluster centroids.
- Repeat the following steps until convergence or a maximum number of iterations is reached:
- For each data point, calculate the mean of all points within a certain radius (i.e., the “kernel”) centered at the data point.
- Shift the data point to the mean.
- Identify the cluster centroids as the points that have not moved after convergence.
- Return the final cluster centroids and the assignments of data points to clusters.
- One of the main advantages of mean-shift clustering is that it does not require the number of clusters to be specified beforehand. Mean-Shift clustering can be applied to various types of data, including image and video processing, object tracking and bioinformatics.

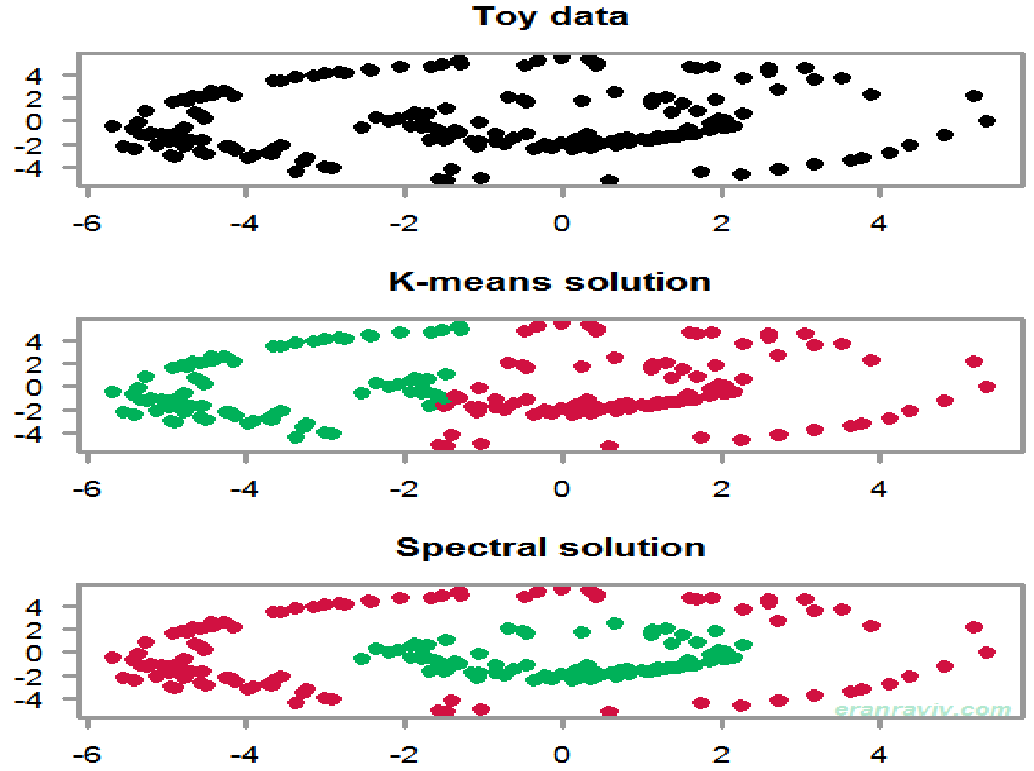
Spectral clustering

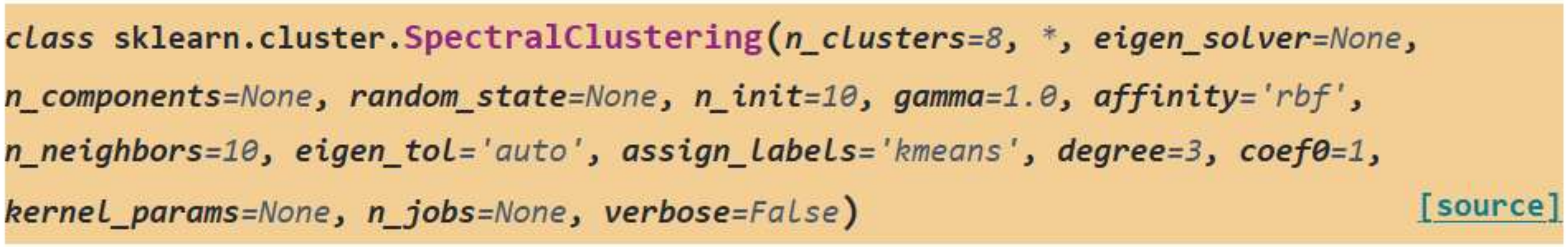
- Spectral clustering performs a low-dimension embedding of the affinity matrix between samples, followed by clustering, e.g., by KMeans, of the components of the eigenvectors in the low dimensional space.
- It is especially computationally efficient if the affinity matrix is sparse and the amg solver is used for the eigenvalue problem
- The present version of SpectralClustering requires the number of clusters to be specified in advance. It works well for a small number of clusters, but is not advised for many clusters.




Degree matrix D
Eigenvectors ϕ
Laplacian matrix L
New eigenvectors U
Clusters
a
b
c
d
e
f
Normal
Abnormal
Similarity matrix S
Adjacency matrix W
Eigenvalues λ


0
20
40
60
80


Toy data
K-means solution
Spectral solution
-6
-4
-2
0
2
4
eranraviv.com



```
class sklearn.cluster.SpectralClustering(n_clusters=8, *, eigen_solver=None,
n_components=None, random_state=None, n_init=10, gamma=1.0, affinity='rbf',
n_neighbors=10, eigen_tol='auto', assign_labels='kmeans', degree=3, coef0=1,
kernel_params=None, n_jobs=None, verbose=False)
```
[source]

Important Parameters used here:

Affinity : The affinity matrix describing the relationship of the samples to embed

n_clusters*:* Number of clusters to extract.

eigen_solver: The eigenvalue decomposition method. [*None ‘arpack’, ‘lobpcg’, or ‘amg’**]***

eigen_tol *:* Stopping criterion for eigendecomposition of the Laplacian matrix *['auto’]*

assign_labels The strategy to use to assign labels in the embedding space. [k-means, Discretization, Cluster_qr]

Advantages of Spectral clustering:

- Handling Non-Convex Clusters
- Hidden Structure Discovery
- Robustness to Noise
- Versatility

Disadvantages of Spectral Clustering:

1. Sensitivity to Hyperparameters
2. Computational Complexity
3. Scalability

Hierarchical clustering

Hierarchical clustering is a algorithms that build nested clusters by merging or splitting them successively.

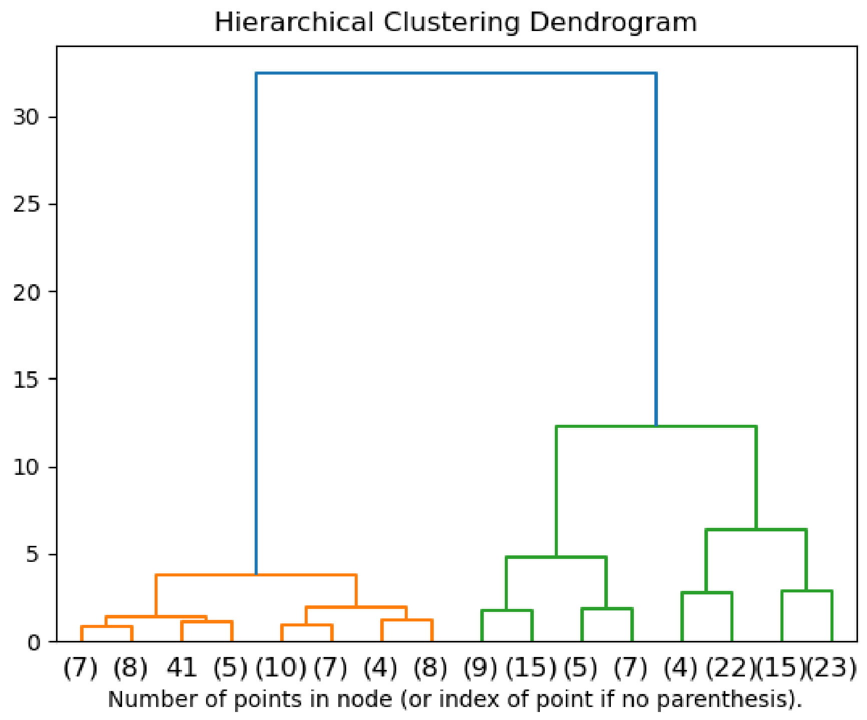
This hierarchy of clusters is represented as a tree (or dendrogram).

The root of the tree is the unique cluster that gathers all the samples, the leaves being the clusters with only one sample

Hierarchical clustering encompasses two main types:

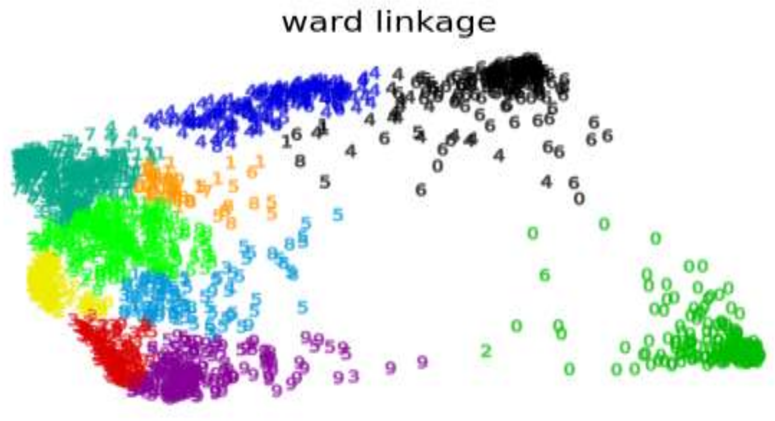
- Agglomerative (bottom-up)
- Divisive (top-down),

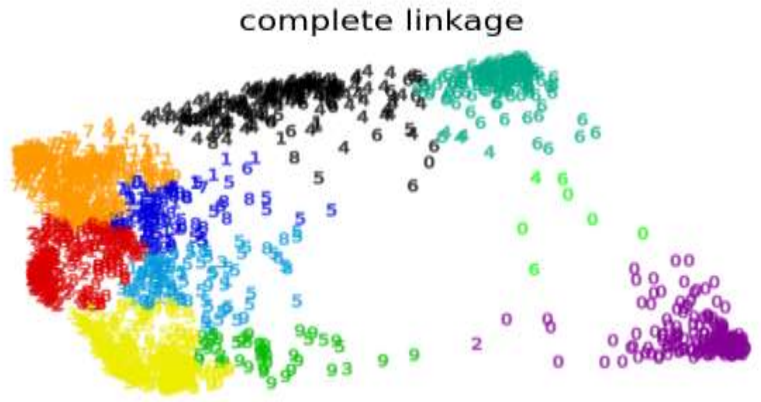
Each organizing data points into a hierarchy of clusters.



The **Agglomerative Clustering** object performs a hierarchical clustering using a bottom up approach, each observation starts in its own cluster, and clusters are successively merged together. The linkage criteria determines the metric used for the merge strategy

- **Ward** minimizes the sum of squared differences within all clusters. It is a variance-minimizing approach and in this sense is similar to the k-means objective function but tackled with an agglomerative hierarchical approach.
- **Maximum or complete linkage** minimizes the maximum distance between observations of pairs of clusters.
- **Average linkage** minimizes the average of the distances between all observations of pairs of clusters.
- **Single linkage** minimizes the distance between the closest observations of pairs of clusters.





Let's say we have six data points **A, B, C, D, E, and F**.

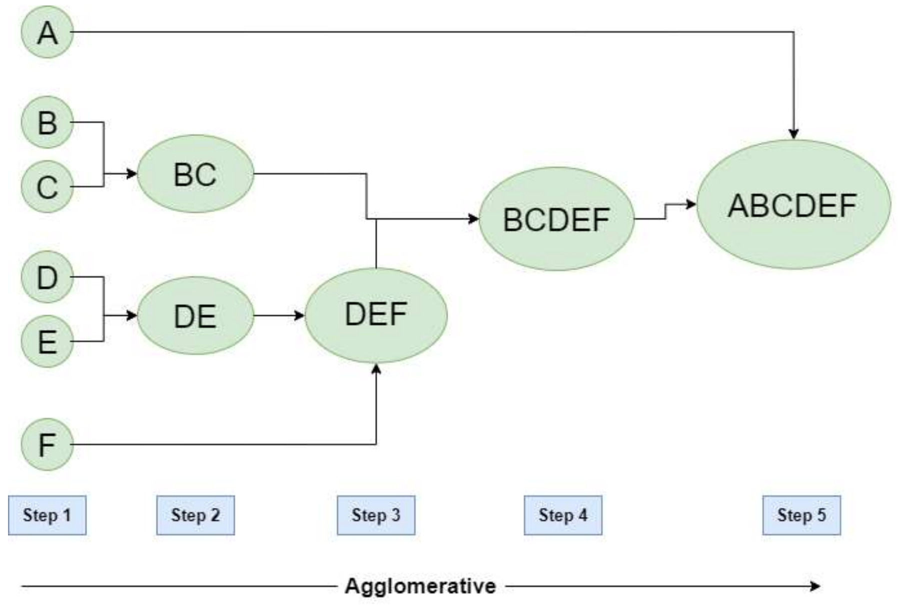
Step-1: Consider each alphabet as a single cluster and calculate the distance of one cluster from all the other clusters.

•**Step-2:** In the second step comparable clusters are merged together to form a single cluster. Let's say cluster (B) and cluster (C) are very similar to each other therefore we merge them in the second step similarly to cluster (D) and (E) and at last, we get the clusters [(A), (BC), (DE), (F)]

•**Step-3:** We recalculate the proximity according to the algorithm and merge the two nearest clusters([(DE), (F)]) together to form new clusters as [(A), (BC), (DEF)]

•**Step-4:** Repeating the same process; The clusters DEF and BC are comparable and merged together to form a new cluster. We're now left with clusters [(A), (BCDEF)].

•**Step-5:** At last, the two remaining clusters are merged together to form a single cluster [(ABCDEF)].



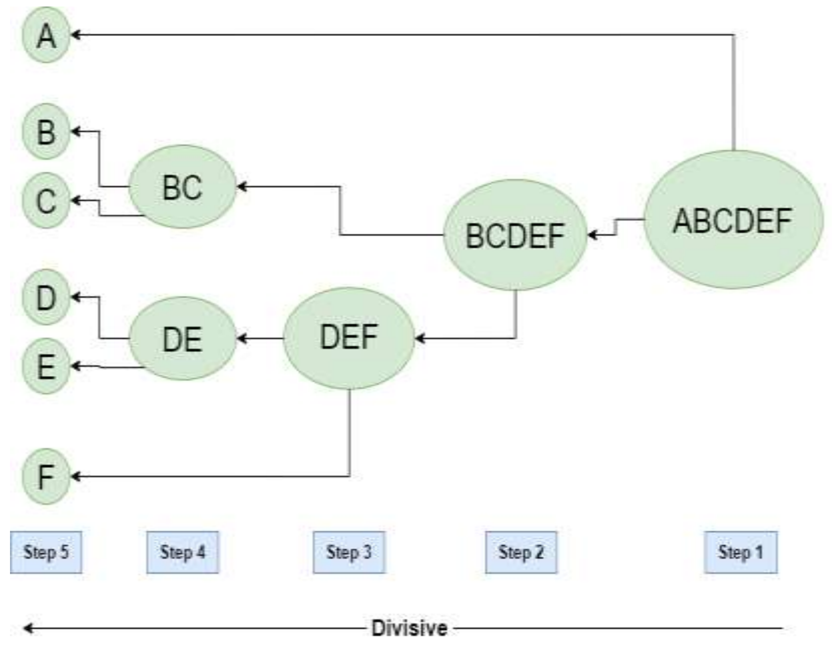
Divisive Clustering

We can say that Divisive Hierarchical clustering is precisely the **opposite** of Agglomerative Hierarchical clustering.

In Divisive Hierarchical clustering, we take into account all of the data points as a single cluster

In every iteration, we separate the data points from the clusters which aren’t comparable.

In the end, we are left with N clusters.



DBSCAN

DBSCAN - Density-Based Spatial Clustering of Applications with Noise

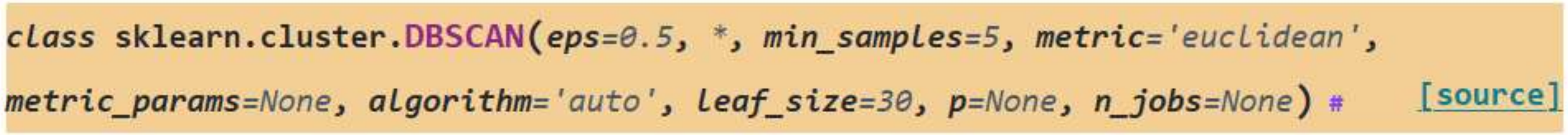
The DBSCAN algorithm views clusters as areas of high density separated by areas of low density. Due to this view, clusters can be any shape and be convex shaped.

The central component to the DBSCAN is the concept of core samples, which are samples that are in areas of high density.

A cluster is a set of core samples, each close to each other (measured by some distance measure) and a set of non-core samples that are close to a core sample (but are not themselves core samples).

There are two parameters to the algorithm**, min_samples and eps**, which define formally what we mean when we say dense.

Higher min_samples or lower eps indicate higher density necessary to form a cluster.



```
class sklearn.cluster.DBSCAN(eps=0.5, *, min_samples=5, metric='euclidean',
metric_params=None, algorithm='auto', leaf_size=30, p=None, n_jobs=None) #    [source]
```

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN, finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

This implementation which can occur when the eps param is large and min_samples is low, while the original DBSCAN only uses linear memory.

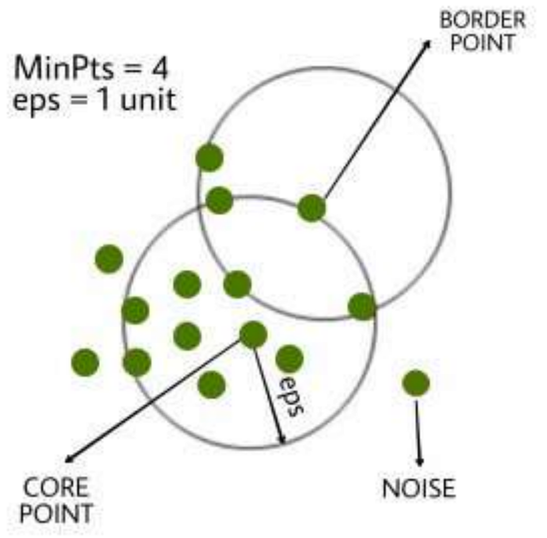
Parameters:

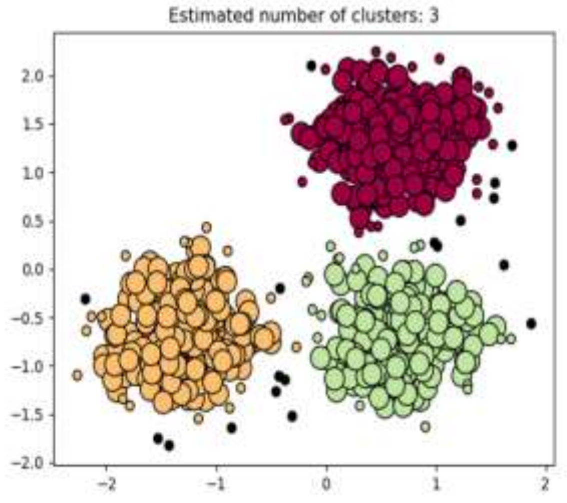
eps -The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

min_samples -The number of samples (or total weight) in a neighborhood for a point to be considered as a core point. This includes the point itself. If min_samples is set to a higher value, DBSCAN will find denser clusters, whereas if it is set to a lower value, the found clusters will be more sparse**.**

Steps in the DBSCAN Algorithm

1. **Identify Core Points**: For each point in the dataset, count the number of points within its eps neighborhood. If the count meets or exceeds MinPts, mark the point as a core point.
2. **Form Clusters**: For each core point that is not already assigned to a cluster, create a new cluster. Recursively find all density-connected points and add to the cluster.
3. **Density Connectivity**: Two points, a and b, are density-connected if there exists a chain of points where each point is within the eps radius of the next, and at least one point in the chain is a core point. This chaining process ensures that all points in a cluster are connected through a series of dense regions.
4. **Label Noise Points**: After processing all points, any point that does not belong to a cluster is labeled as noise.





OPTICS (Ordering Points To Identify the Clustering Structure)

The OPTICS algorithm shares many similarities with the DBSCAN algorithm, and that relaxes the eps requirement from a single value to a value range.

The key difference between DBSCAN and OPTICS is that the OPTICS algorithm builds a **reachability graph**, which assigns each sample both a reachability_ distance, and a spot within the cluster ordering_ attribute

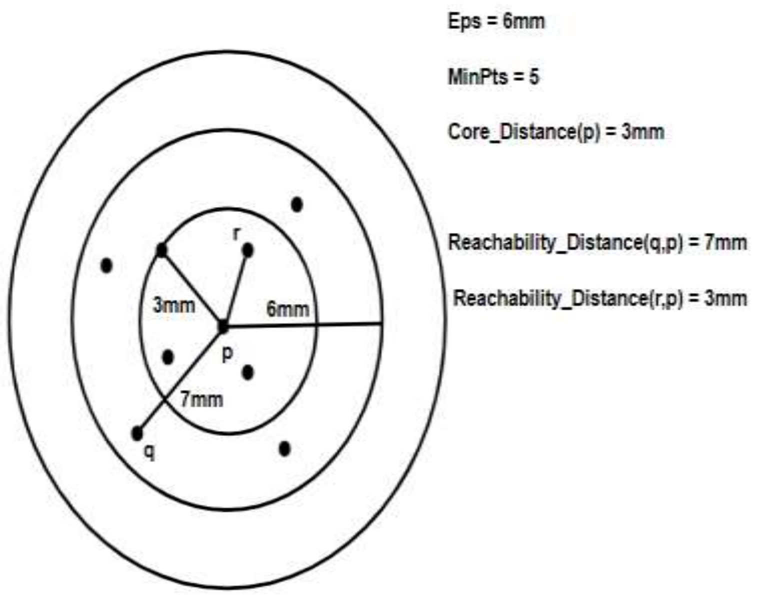
These two attributes are assigned when the model is fitted, and are used to determine cluster membership. If OPTICS is run with the default value of inf set for max_eps, then DBSCAN style cluster extraction can be performed repeatedly in linear time for any given eps value using the cluster_optics_dbscan method.

Setting max_eps to a lower value will result in shorter run times, and can be thought of as the maximum neighborhood radius from each point to find other potential reachable points.

Illustration of Core and Reachability Distances

- Epsilon (Eps) = 6mm and MinPts = 5.
- The core distance of point p is 3mm meaning it needs at least 5 points within a 3mm radius to be considered as a core point.
- The reachability distance from q to p is 7mm (since q is farther than p‘s core distance).
- The reachability distance from r to p is 3mm (since r is within p‘s core distance).

It is more informative than DBSCAN as the reachability plot provides better understanding of clustering structure. Now we will learn about its working.


Eps = 6mm
MinPts = 5
Core_Distance(p) = 3mm
Reachability_Distance(q,p) = 7mm
Reachability_Distance(r,p) = 3mm
r
3mm
6mm
p
7mm
q

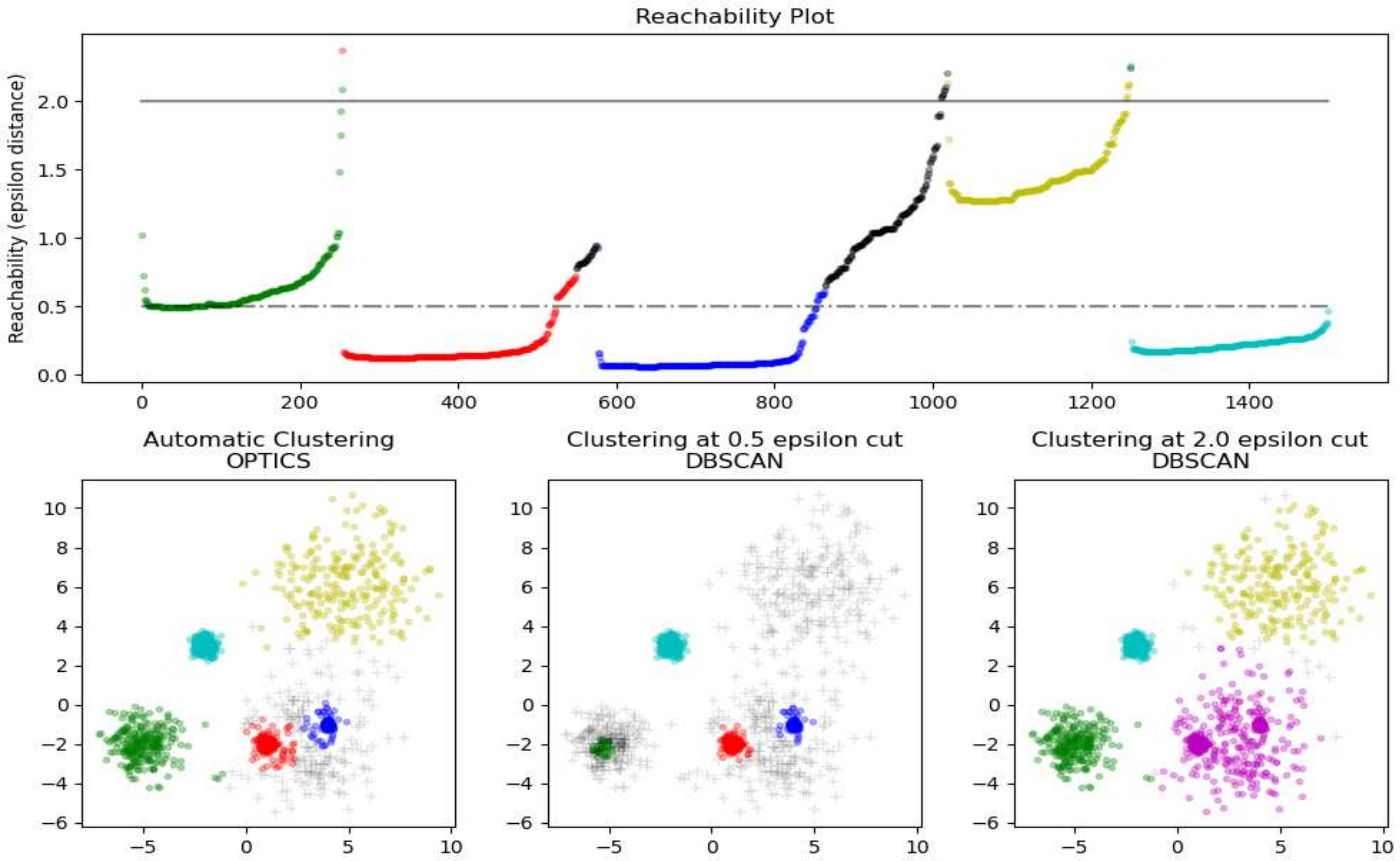
Step-by-Step Working of OPTICS from the Image

- The algorithm selects a starting point and checks if it has at least MinPts neighbors within Eps.
- If the point meets the density requirement it is marked as a core point and nearby points are then analyzed.
- Reachability distance is computed for each neighboring point.
- Points are then processed in order of their reachability distance hence forming a reachability plot.
- Clusters appear as valleys (low reachability distances) and noise appears as peaks (high reachability distances).

Demo of OPTICS clustering algorithm

Finds core samples of high density and expands clusters from them. This example uses data that is generated so that the clusters have different densities.

The OPTICS is first used with its Xi cluster detection method, and then setting specific thresholds on the reachability, which corresponds to DBSCAN. We can see that the different clusters of OPTICS's Xi method can be recovered with different choices of thresholds in DBSCAN.


Reachability Plot
Reachability (epsilon distance)
Automatic Clustering
OPTICS
Clustering at 0.5 epsilon cut
DBSCAN
Clustering at 2.0 epsilon cut
DBSCAN

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)

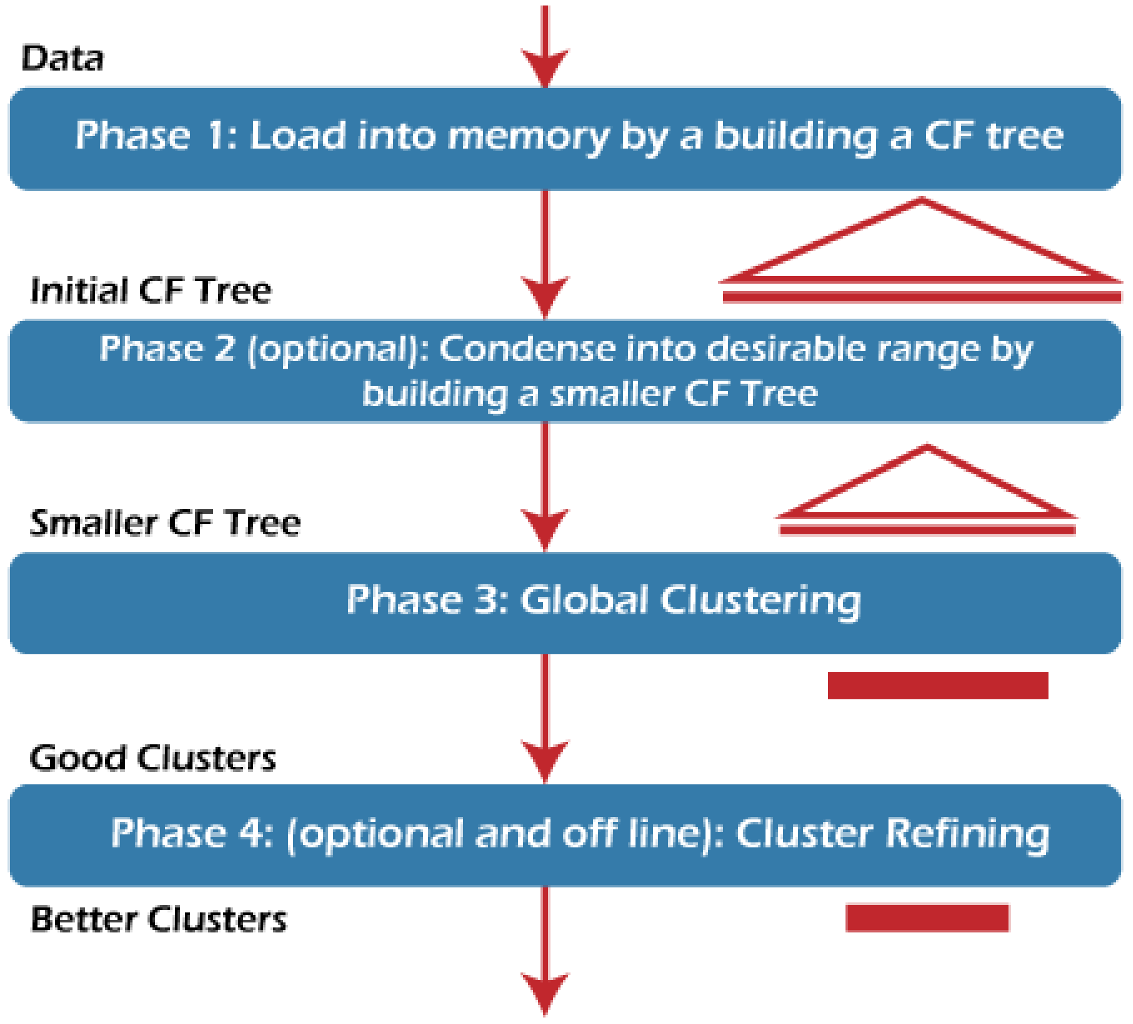
The Birch builds a tree called the Clustering Feature Tree (CFT) for the given data. The data is essentially lossy compressed to a set of Clustering Feature nodes (CF Nodes).

The CF Nodes have a number of subclusters called Clustering Feature subclusters (CF Subclusters) and these CF Subclusters located in the non-terminal CF Nodes can have CF Nodes as children.

The CF Subclusters hold the necessary information for clustering which prevents the need to hold the entire input data in memory. This information includes:

- Number of samples in a subcluster.
- Linear Sum - An n-dimensional vector holding the sum of all samples
- Squared Sum - Sum of the squared L2 norm of all samples.
- Centroids - To avoid recalculation linear sum / n_samples.
- Squared norm of the centroids.

The BIRCH Clustering Algorithm


Data
Phase 1: Load into memory by a building a CF tree
Initial CF Tree
Phase 2 (optional): Condense into desirable range by building a smaller CF Tree
Smaller CF Tree
Phase 3: Global Clustering
Good Clusters
Phase 4: (optional and off line): Cluster Refining
Better Clusters

Algorithm description

The BIRCH algorithm has two parameters, the **threshold** and the **branching factor**. The branching factor limits the number of subclusters in a node and the threshold limits the distance between the entering sample and the existing subclusters.

This algorithm can be viewed as an instance or data reduction method, since it reduces the input data to a set of subclusters which are obtained directly from the leaves of the CFT. This reduced data can be further processed by feeding it into a global clusterer.

This global clusterer can be set by n_clusters. If n_clusters is set to None, the subclusters from the leaves are directly read off, otherwise a global clustering step labels these subclusters into global clusters (labels) and the samples are mapped to the global label of the nearest subcluster.

A new sample is inserted into the root of the CF Tree which is a CF Node. It is then merged with the subcluster of the root, that has the smallest radius after merging, constrained by the threshold and branching factor conditions. If the subcluster has any child node, then this is done repeatedly till it reaches a leaf. After finding the nearest subcluster in the leaf, the properties of this subcluster and the parent subclusters are recursively updated.

If the radius of the subcluster obtained by merging the new sample and the nearest subcluster is greater than the square of the threshold and if the number of subclusters is greater than the branching factor, then a space is temporarily allocated to this new sample. The two farthest subclusters are taken and the subclusters are divided into two groups on the basis of the distance between these subclusters.

If this split node has a parent subcluster and there is room for a new subcluster, then the parent is split into two. If there is no room, then this node is again split into two and the process is continued recursively, till it reaches the root.

THANK YOU