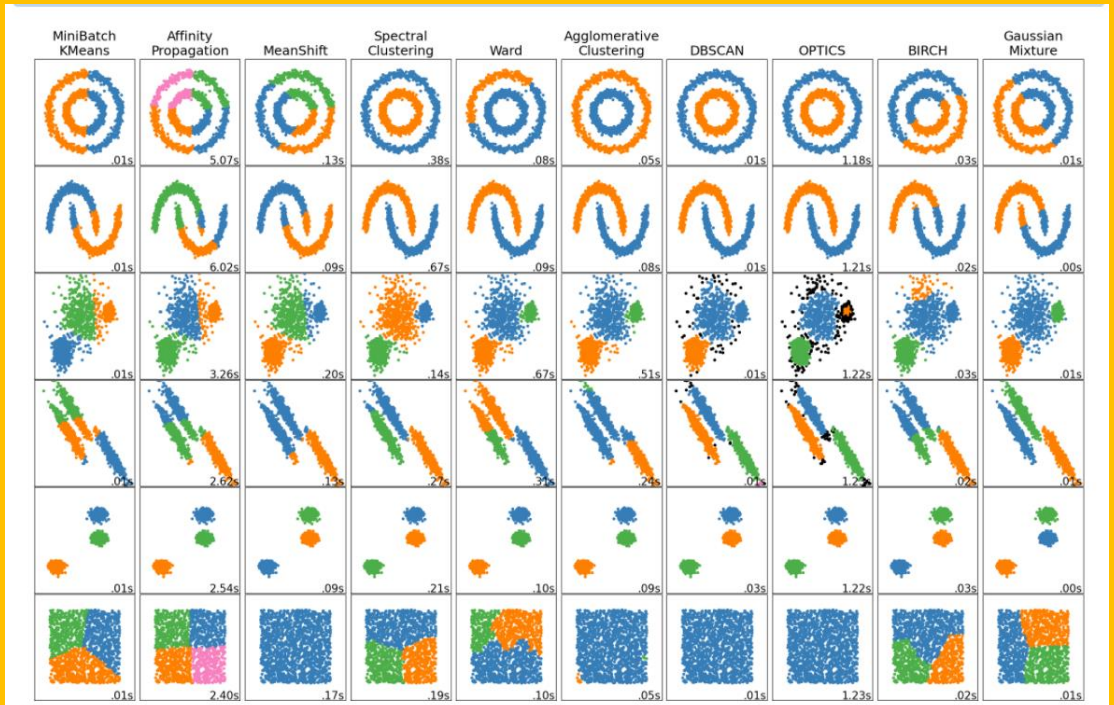


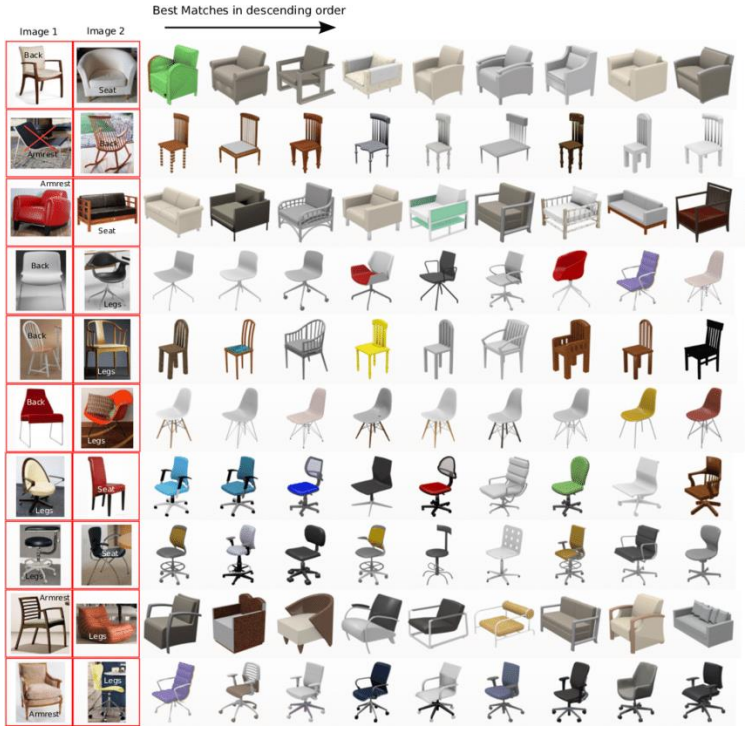
A comparison of the clustering algorithms in scikit-learn

Clustering Algorithms

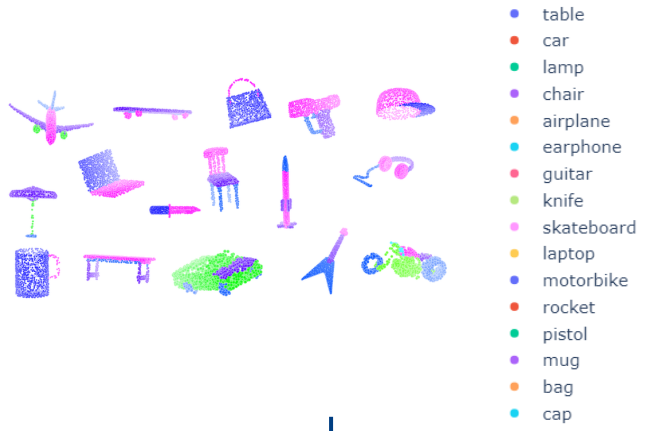
Part-1

Motivation

Problem: 3D Model Retrieval
Given a 3D Model find the most similar models



Shape-net dataset



Generate shape representations of these 3d models

Perform clustering on the shape representations

Ranking of the 3D models within a particular cluster

Given a 3D model, retrieve top-n ranks of 3D model

Agenda

01

Overview of Clustering

Types of clustering

02

K-Family Algorithms

K-Means

K-Medoids and more

03

Mean Shift Algorithm

Short Introduction

Pros and Cons

04

Affinity Propagation

Introduction

Pros and Cons

05

Spectral Clustering

Introduction

Pros and Cons

What is clustering?

Definition [edit]

The notion of a "cluster" cannot be precisely defined, which is one of the reasons why there are so many clustering algorithms.[5] There is a common denominator: a group of data objects. However, different researchers employ different cluster models, and for each of these cluster models again different algorithms can be given. The notion of a cluster, as found by different algorithms, varies significantly in its properties. Understanding these "cluster models" is key to understanding the differences between the various algorithms. Typical cluster models include:

- *Connectivity models*: for example, hierarchical clustering builds models based on distance connectivity.
- *Centroid models*: for example, the k-means algorithm represents each cluster by a single mean vector.
- *Distribution models*: clusters are modeled using statistical distributions, such as multivariate normal distributions used by the expectation-maximization algorithm.
- *Density models*: for example, DBSCAN and OPTICS defines clusters as connected dense regions in the data space.
- *Subspace models*: in biclustering (also known as co-clustering or two-mode-clustering), clusters are modeled with both cluster members and relevant attributes.
- *Group models*: some algorithms do not provide a refined model for their results and just provide the grouping information.
- *Graph-based models*: a clique, that is, a subset of nodes in a graph such that every two nodes in the subset are connected by an edge can be considered as a prototypical form of cluster. Relaxations of the complete connectivity requirement (a fraction of the edges can be missing) are known as quasi-cliques, as in the HCS clustering algorithm.
- *Signed graph models*: Every path in a signed graph has a sign from the product of the signs on the edges. Under the assumptions of balance theory, edges may change sign and result in a bifurcated graph. The weaker "clusterability axiom" (no cycle has exactly one negative edge) yields results with more than two clusters, or subgraphs with only positive edges.[6]
- *Neural models*: the most well known unsupervised neural network is the self-organizing map and these models can usually be characterized as similar to one or more of the above models, and including subspace models when neural networks implement a form of Principal Component Analysis or Independent Component Analysis.

A "clustering" is essentially a set of such clusters, usually containing all objects in the data set. Additionally, it may specify the relationship of the clusters to each other, for example, a hierarchy of clusters embedded in each other. Clusterings can be roughly distinguished as:

- *Hard clustering*: each object belongs to a cluster or not
- *Soft clustering* (also: *fuzzy clustering*): each object belongs to each cluster to a certain degree (for example, a likelihood of belonging to the cluster)

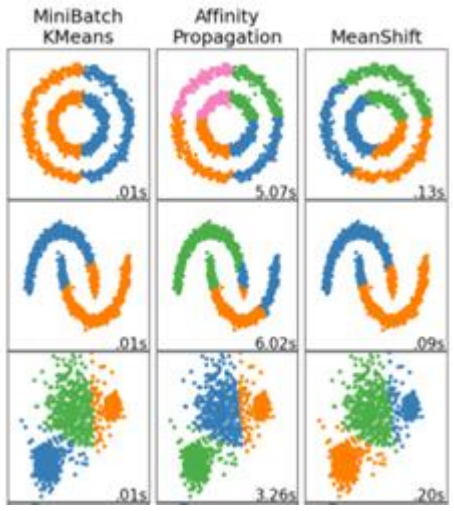
There are also finer distinctions possible, for example:

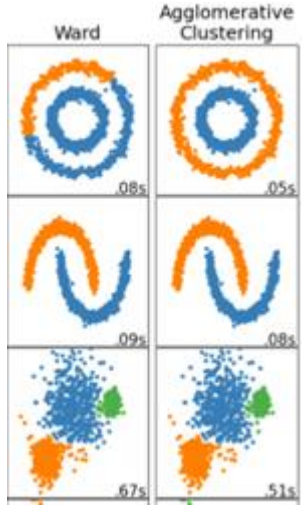
- *Strict partitioning clustering*: each object belongs to exactly one cluster
- *Strict partitioning clustering with outliers*: objects can also belong to no cluster, and are considered outliers
- *Overlapping clustering* (also: *alternative clustering*, *multi-view clustering*): objects may belong to more than one cluster; usually involving hard clusters
- *Hierarchical clustering*: objects that belong to a child cluster also belong to the parent cluster
- *Subspace clustering*: while an overlapping clustering, within a uniquely defined subspace, clusters are not expected to overlap

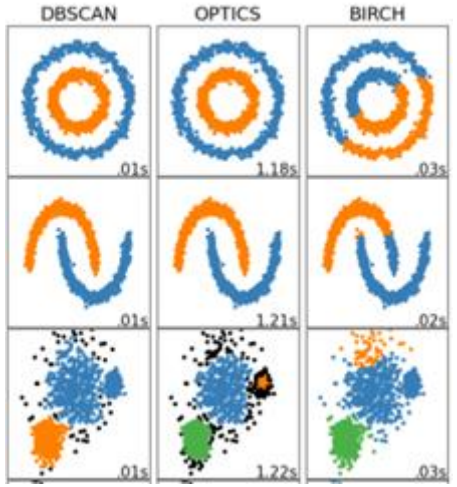
Types of Clustering

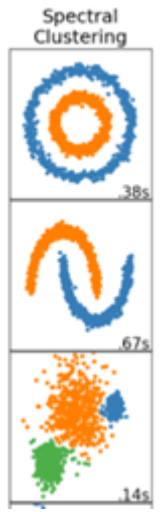
Clustering

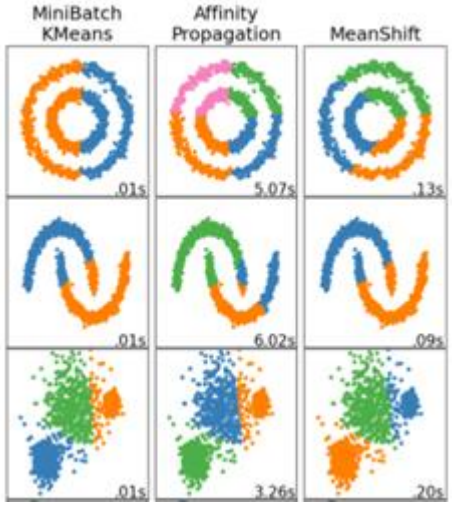
- Centroid Based Clustering
- Hierarchical Based Clustering
- Density Based Clustering
- Graph Based Clustering







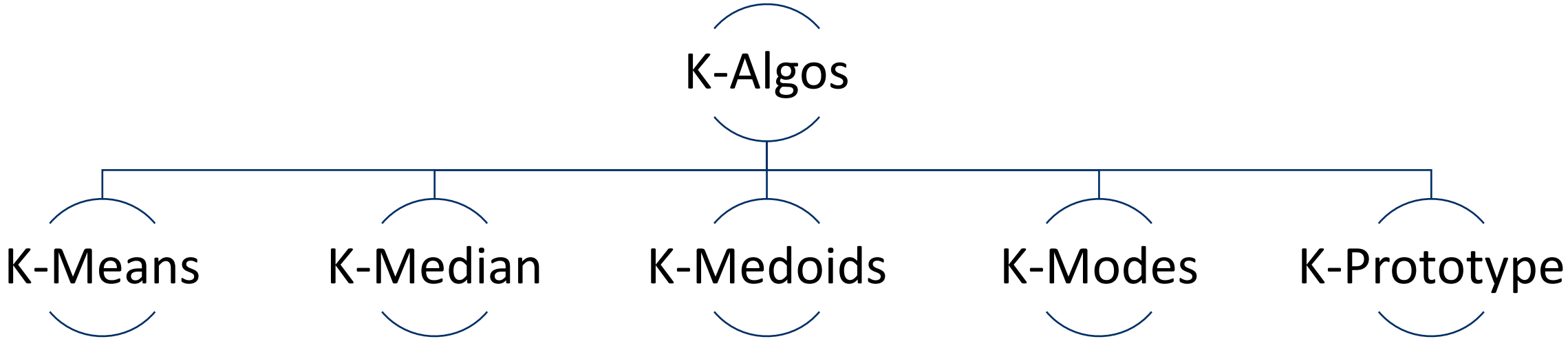




Centroid Based Clustering

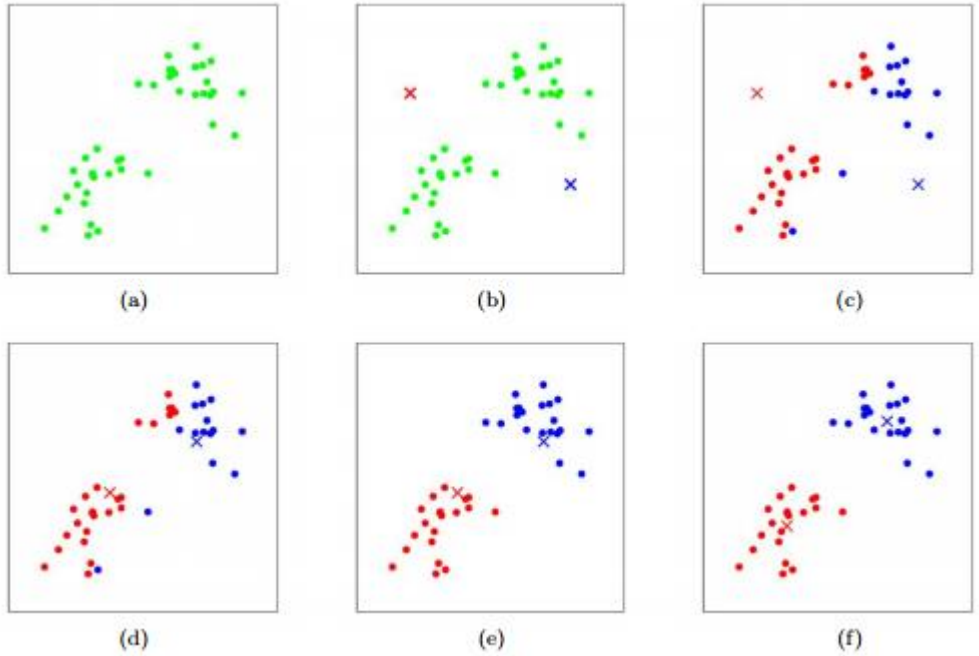
01

Family of K-Algos



K-Means (Lloyd's) Algorithm

1. Specify the number of clusters to assign (k)
2. Randomly initialize k centroids
3. Loop :
    1. Assign each point to the closest centroid
    2. Compute new centroid (mean) of each cluster
4. Until the centroid position do not change

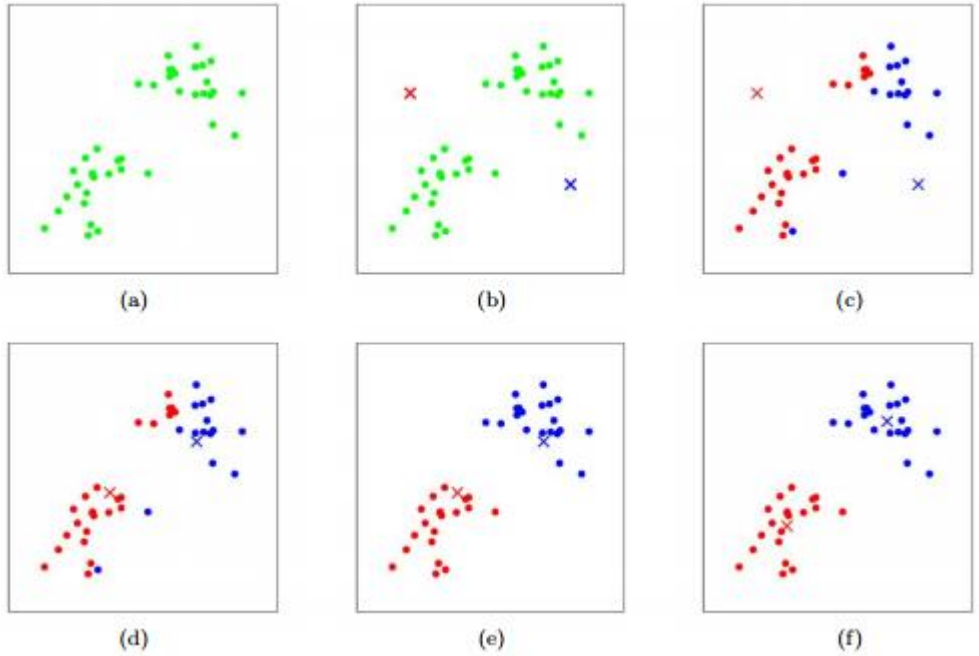


K-Median

1. Instead of using mean for each cluster to determine the centroid, one uses median.

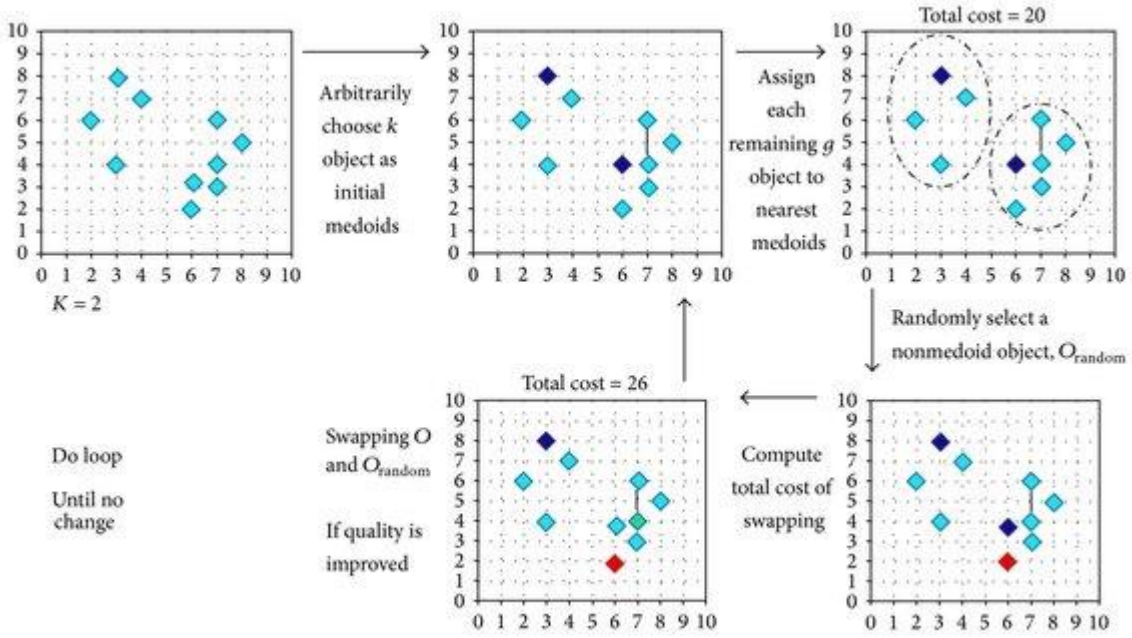
2. Has the effect of using 1-norm distance metric instead of using squared 2-norm distance metric (used by k-means)

3. This method is more robust to outliers since median is more robust than mean



K-Medoids Algorithm

1. Select k points as the initial representative medoids
2. Loop :
    1. Assign each point to the closest medoid
    2. Randomly select a non-representative object (medoid)
    3. Compute the cost of the cluster after swapping the object
    4. If cost decreases, then swap them to form the new set of medoids
3. Until the convergence criteria



- Partitioning Around Medoids (PAM)
- CLARANS

K-Medoids Properties

- Centroids in k-means, which represents the cluster are virtual points, whereas, in case of medoids they represents points from the actual dataset which represents the clusters. Hence providing greater interpretability of clusters
- K-Means is sensitive to outliers, since one outlier may pull the centroid in its direction. K-Medoids however, are robust to them
- K-medoids can be used with arbitrary dissimilarity measures, whereas K-Means requires Euclidian distances for efficient solutions
- K-Medoids has been show to perform better than K-Means for Big Data - [(PDF) Analysis of K-Means and K-Medoids Algorithm For Big Data (researchgate.net)]

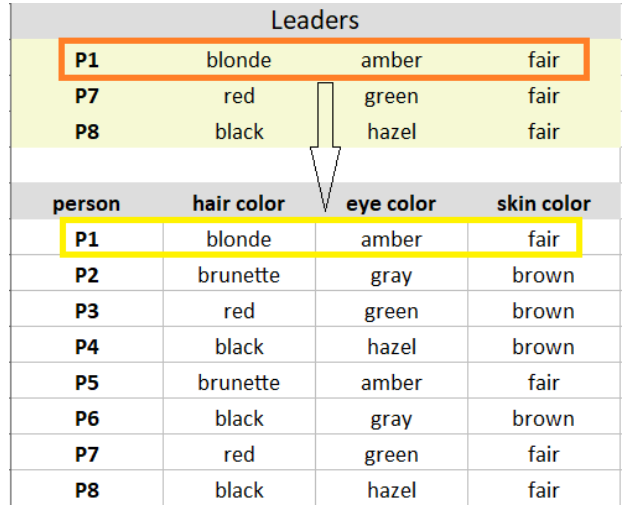
Algorithm is implemented in **sklearn-extra**

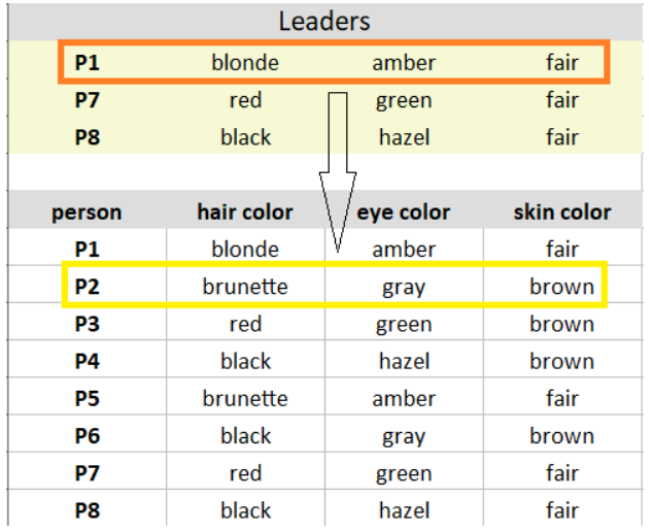
K-Modes

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

Leaders			
P1	blonde	amber	fair
P7	red	green	fair
P8	black	hazel	fair

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair





1. K-Modes clustering is used when you have categorical data

2. The notion of distance metric is the “mode” of the numbers

	Cluster 1 (P1)	Cluster 2 (P7)	Cluster 3 (P8)	Cluster
P1	0 ✓	2	2	Cluster 1
P2	3 ✓	3	3	Cluster 1
P3	3	1 ✓	3	Cluster 2
P4	3	3	1 ✓	Cluster 3
P5	1 ✓	2	2	Cluster 1
P6	3	3	2 ✓	Cluster 3
P7	2	0 ✓	2	Cluster 2
P8	2	2	0 ✓	Cluster 3

https://www.analyticsvidhya.com/blog/2021/06/kmodes-clustering-algorithm-for-categorical-data/

K-Modes Continued

Algorithm is implemented in **kmodes** package

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

New Leaders			
	hair color	eye color	skin color
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

person	hair color	eye color	skin color
P1	blonde	amber	fair
P2	brunette	gray	brown
P3	red	green	brown
P4	black	hazel	brown
P5	brunette	amber	fair
P6	black	gray	brown
P7	red	green	fair
P8	black	hazel	fair

New Leaders			
	hair color	eye color	skin color
Cluster 1	brunette	amber	fair
Cluster 2	red	green	fair
Cluster 3	black	hazel	brown

	Cluster 1	Cluster 2	Cluster 3	Cluster
P1	1 ✓	2	3	Cluster 1
P2	2 ✓	3	2	Cluster 1
P3	3	1 ✓	2	Cluster 2
P4	3	3	0 ✓	Cluster 3
P5	0 ✓	2	3	Cluster 1
P6	3	3	1 ✓	Cluster 3
P7	2	0 ✓	3	Cluster 2
P8	2	2	1 ✓	Cluster 3

K-Prototype (Heterogeneous Data)

Algorithm is implemented in **kmodes** package

	Region	Country	Item Type	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
0	Sub-Saharan Africa	Chad	Office Supplies	Online	L	1/27/2011	292494523	2/12/2011	4484	651.21	524.96	2920025.64	2353920.64	566105.00
1	Europe	Latvia	Beverages	Online	C	12/28/2015	361825549	1/23/2016	1075	47.45	31.79	51008.75	34174.25	16834.50
2	Middle East and North Africa	Pakistan	Vegetables	Offline	C	1/13/2011	141515767	2/1/2011	6515	154.06	90.93	1003700.90	592408.95	411291.95
3	Sub-Saharan Africa	Democratic Republic of the Congo	Household	Online	C	9/11/2012	500364005	10/6/2012	7683	668.27	502.54	5134318.41	3861014.82	1273303.59
4	Europe	Czech Republic	Beverages	Online	C	10/27/2015	127481591	12/5/2015	3491	47.45	31.79	165647.95	110978.89	54669.06

1. Used when we have mix of continuous and categorical data

2. K-Prototype solves this using K-Means for continuous data and K-Modes for categorical data

3. The dissimilarity measure is the weighted combination of Euclidian distance and the dissimilarity of modes

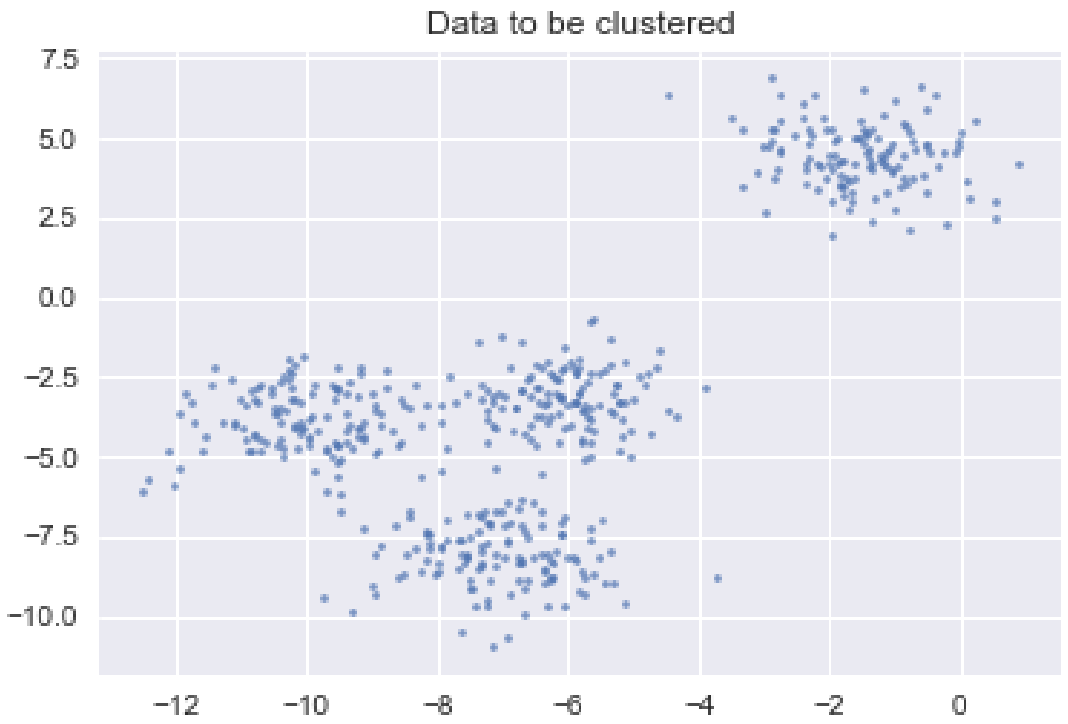
This problem can be solved using other types of clustering techniques like

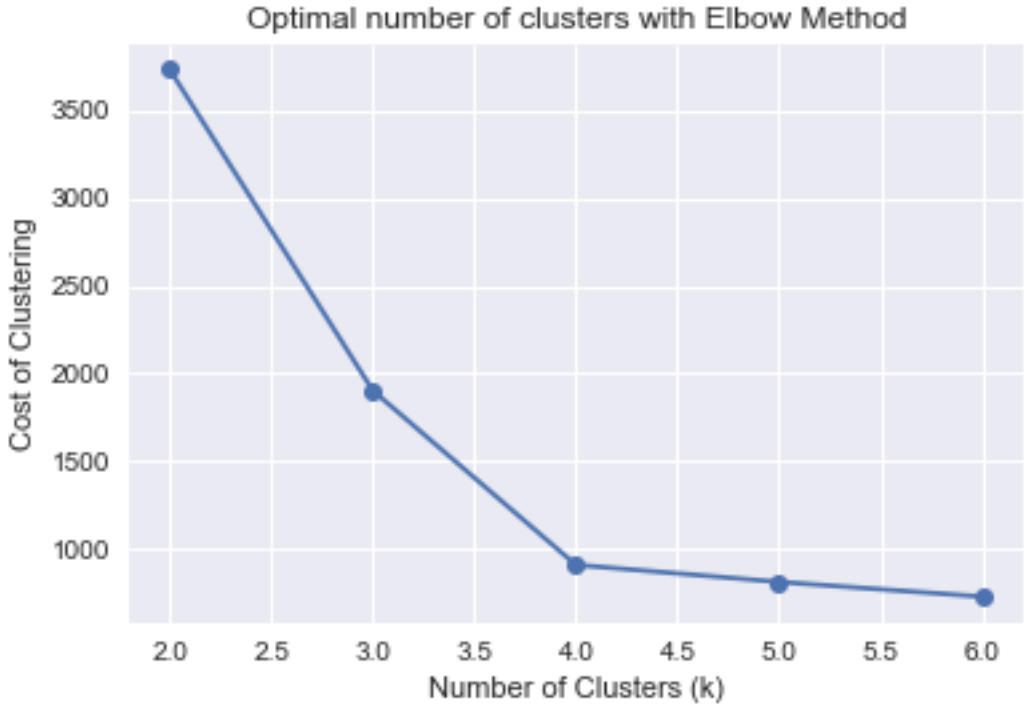
- Gower's distance + K-Medoids
- Kamila
- LCA
- LCM
- MixMode

https://www.nature.com/articles/s41598-021-83340-8

https://towardsdatascience.com/the-k-prototype-as-clustering-algorithm-for-mixed-data-type-categorical-and-numerical-fe7c50538ebb

How to find K? – Elbow Method





How good are the clusters? Silhouette Method

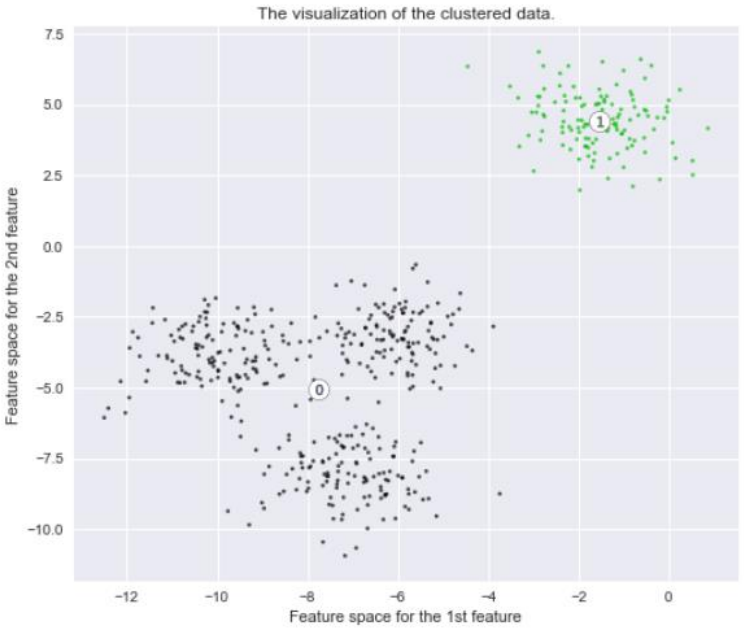
2.3.10.5. Silhouette Coefficient

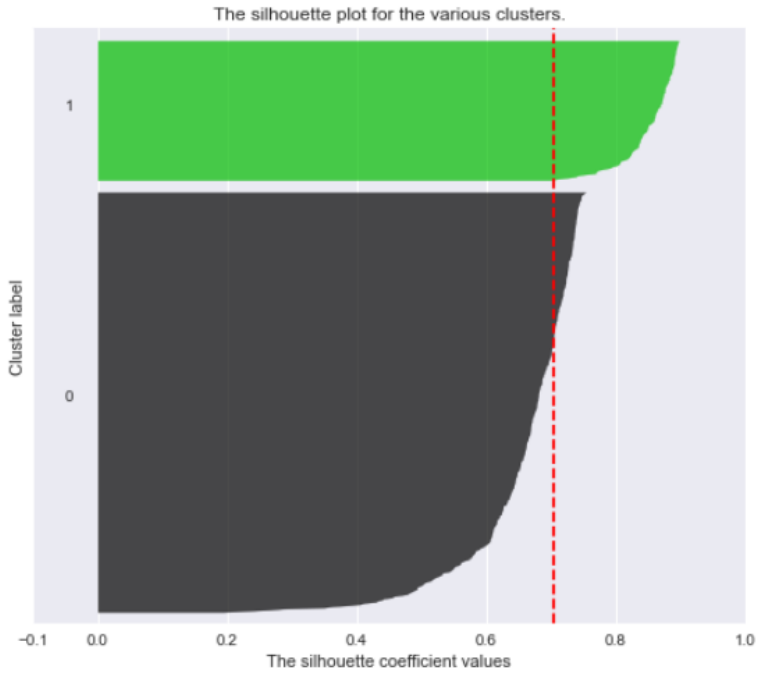
If the ground truth labels are not known, evaluation must be performed using the model itself. The Silhouette Coefficient (`sklearn.metrics.silhouette_score`) is an example of such an evaluation, where a higher Silhouette Coefficient score relates to a model with better defined clusters. The Silhouette Coefficient is defined for each sample and is composed of two scores:

- **a**: The mean distance between a sample and all other points in the same class.
- **b**: The mean distance between a sample and all other points in the *next nearest cluster*.

The Silhouette Coefficient *s* for a single sample is then given as:

$$s = \frac{b - a}{max(a, b)}$$





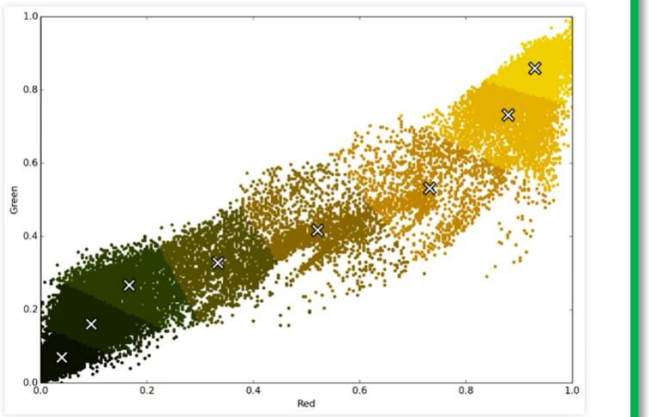
$$Silhouette\ Coefficient(SC)\ range\ from\ [-1, 1]$$

$$SC = +1 \rightarrow indicates, sample\ is\ far\ away\ from\ neighbouring\ clusters$$
$$SC = 0 \rightarrow sample\ is\ on\ or\ very\ close\ to\ the\ decision\ boundary$$
$$SC = -1 \rightarrow samples\ might\ have\ been\ assigned\ to\ wrong\ clusters$$

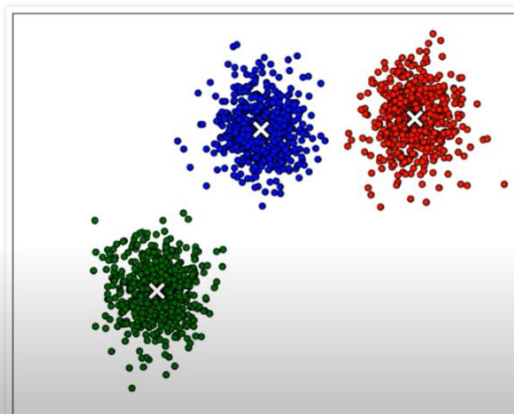
https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html

Pros and Cons of K-Family Algos

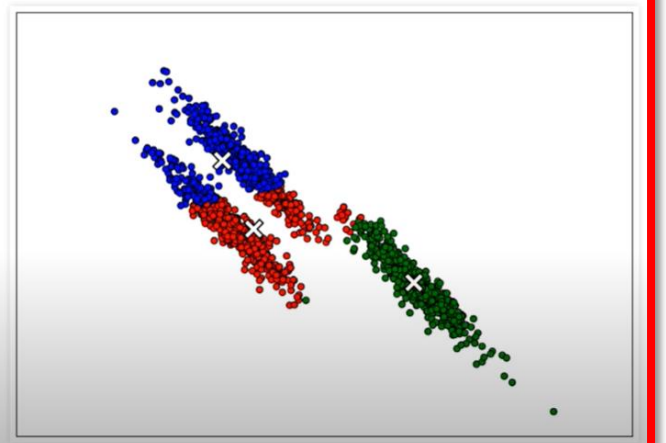
- Note: this may be useful for applications like vector quantization



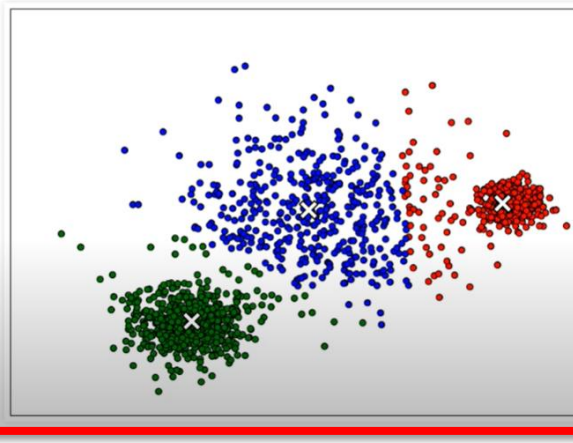
- Good for globular clusters with similar dispersion



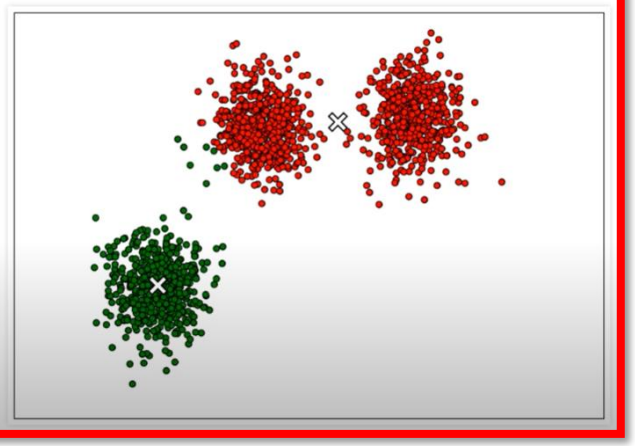
- Not good for non-globular clusters



- Or differing variances



- (assuming k is correct)

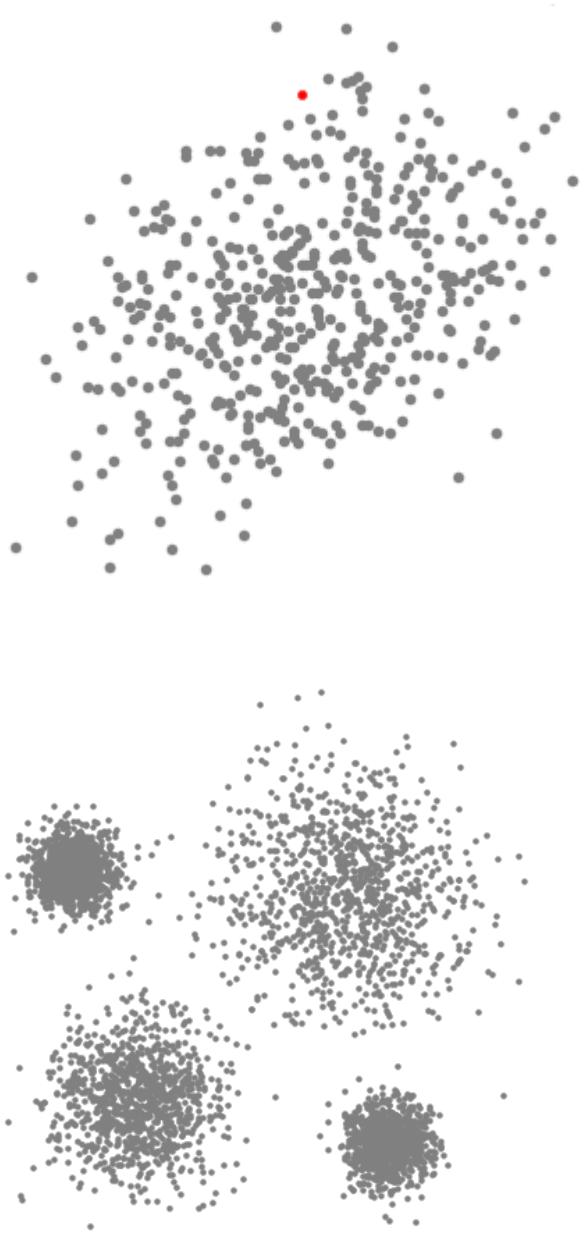


Mean Shift Algorithm

1. Initialize the centers randomly on the dataset

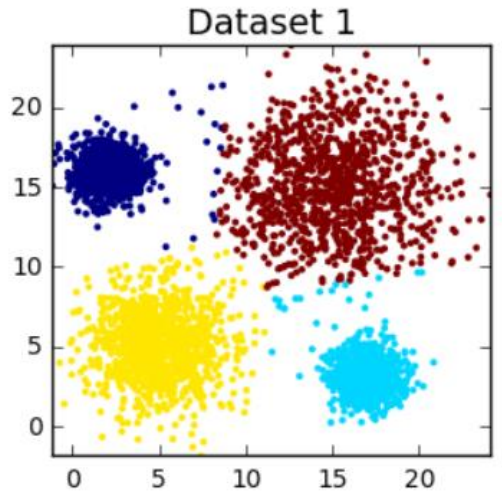
2. Loop:
    1. Around each centers is a ball (determined by the bandwidth parameter), calculate the density of the center.
    2. Update the center with the centroid calculated on the points inside the ball of the center

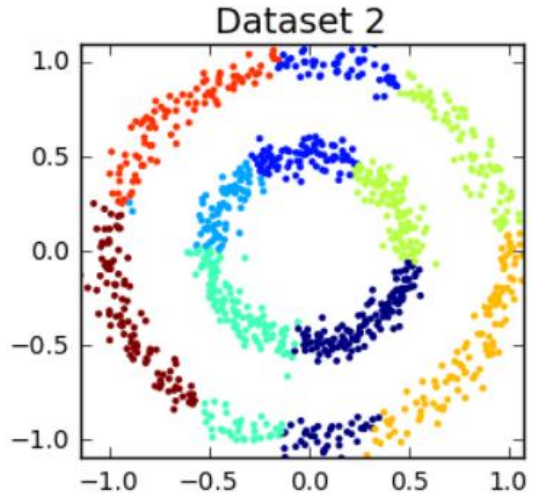
3. Until convergence



Pros and Cons of Mean Shift Clustering

- No need to pre-determine the Number of Clusters
- Can cluster data with varying density





- Still suffers from the problem of the data being globular in shape.
- Estimating bandwidth parameter is not easy.

Affinity Propagation Algorithm

Affinity Propagation is based on data points choosing the exemplar points

- Similarity Matrix (s)
- Responsibility Matrix (r)
- Availability Matrix (a)
- Criterion Matrix (c)

$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ such that } k' \neq k} \{a(i,k') + s(i,k')\}, \quad (1)$$

$$a(k,k) \leftarrow \sum_{i' \text{ such that } i' \neq k} \max\{0, r(i',k)\}, \quad (2)$$

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ such that } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\} \quad (3)$$

,

$$c(i,k) \leftarrow r(i,k) + a(i,k). \quad (4)$$

Working of Affinity Propogation

Table 1: Preferences of Five Participants

Participant	Tax Rate	Fee	Interest Rate	Quantity Limit	Price Limit
Alice	3	4	3	2	1
Bob	4	3	5	1	1
Cary	3	5	3	3	3
Doug	2	1	3	3	2
Edna	1	1	3	2	3

Table 2: Similarity Matrix

Participant	Alice	Bob	Cary	Doug	Edna
Alice	-22	-7	-6	-12	-17
Bob	-7	-22	-17	-17	-22
Cary	-6	-17	-22	-18	-21
Doug	-12	-17	-18	-22	-3
Edna	-17	-22	-21	-3	-22

Table 3: Responsibility Matrix

Participant	Alice	Bob	Cary	Doug	Edna
Alice	-16	-1	1	-6	-11
Bob	10	-15	-10	-10	-15
Cary	11	-11	-16	-12	-15
Doug	-9	-14	-15	-19	9
Edna	-14	-19	-18	14	-19

$$r(i,k) \leftarrow s(i,k) - \max_{k' \text{ such that } k' \neq k} \{a(i,k') + s(i,k')\},$$

Table 4: Availability Matrix

Participant	Alice	Bob	Cary	Doug	Edna
Alice	21	-15	-16	-5	-10
Bob	-5	0	-15	-5	-10
Cary	-6	-15	1	-5	-10
Doug	0	-15	-15	14	-19
Edna	0	-15	-15	-19	9

$$a(k,k) \leftarrow \sum_{i' \text{ such that } i' \neq k} \max\{0, r(i',k)\}, \qquad a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i' \text{ such that } i' \notin \{i,k\}} \max\{0, r(i',k)\}\right\}$$

Table 5: Criterion Matrix

Participant	Alice	Bob	Cary	Doug	Edna
Alice	**5**	-16	-15	-11	-21
Bob	**5**	-15	-25	-15	-25
Cary	**5**	-26	-15	-17	-25
Doug	-9	-29	-30	**-5**	-10
Edna	-14	-34	-33	**-5**	-10

$$c(i,k) \leftarrow r(i,k) + a(i,k).$$

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.490.7628&rep=rep1&type=pdf

Parameters of Affinity Propagation

class `sklearn.cluster.` **`AffinityPropagation`**(*, *damping=0.5*, *max_iter=200*, *convergence_iter=15*, *copy=True*, *preference=None*, *affinity='euclidean'*, *verbose=False*, *random_state='warn'*) [source]

Perform Affinity Propagation Clustering of data.

Read more in the User Guide.

Parameters:

damping : *float, default=0.5*
Damping factor (between 0.5 and 1) is the extent to which the current value is maintained relative to incoming values (weighted 1 - damping). This in order to avoid numerical oscillations when updating these values (messages).

max_iter : *int, default=200*
Maximum number of iterations.

convergence_iter : *int, default=15*
Number of iterations with no change in the number of estimated clusters that stops the convergence.

copy : *bool, default=True*
Make a copy of input data.

preference : *array-like of shape (n_samples,) or float, default=None*
Preferences for each point - points with larger values of preferences are more likely to be chosen as exemplars. The number of exemplars, ie of clusters, is influenced by the input preferences value. If the preferences are not passed as arguments, they will be set to the median of the input similarities.

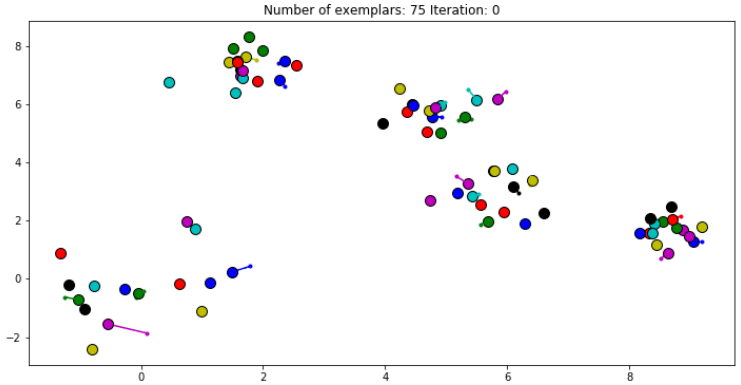
affinity : *{'euclidean', 'precomputed'}, default='euclidean'*
Which affinity to use. At the moment 'precomputed' and `euclidean` are supported. 'euclidean' uses the negative squared euclidean distance between points.

verbose : *bool, default=False*
Whether to be verbose.

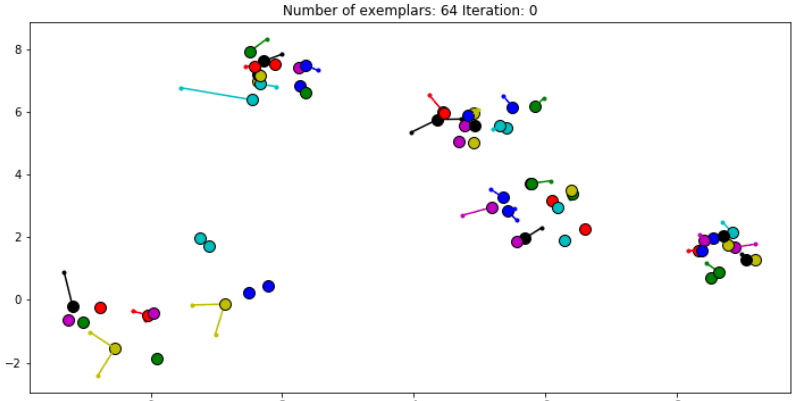
random_state : *int, RandomState instance or None, default=0*
Pseudo-random number generator to control the starting state. Use an int for reproducible results across function calls. See the Glossary.

New in version 0.23: this parameter was previously hardcoded as 0.

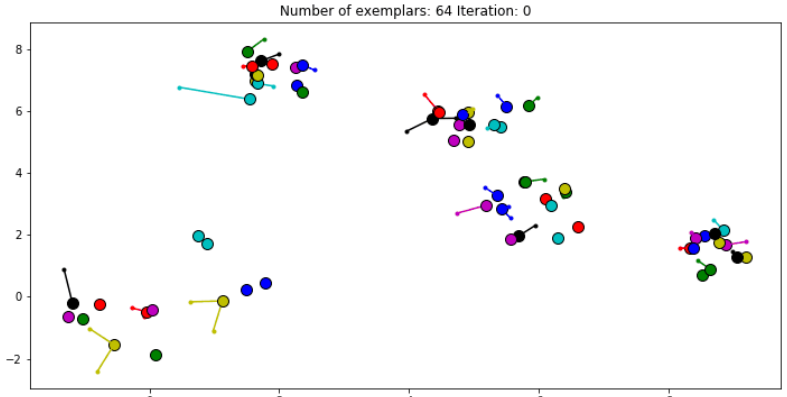
Choosing Preference is difficult



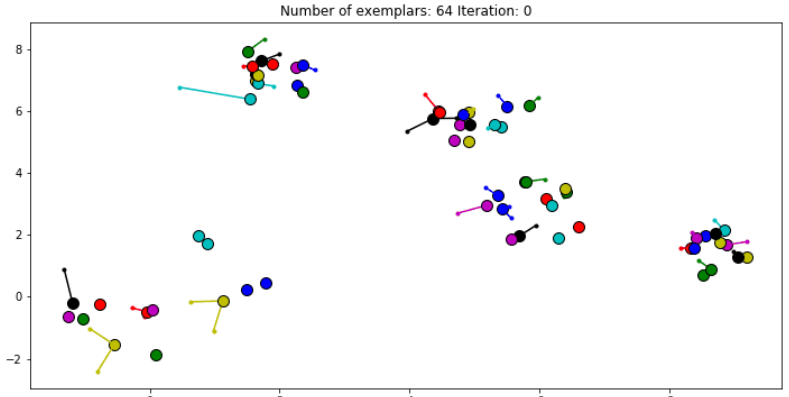
$preference = 0$



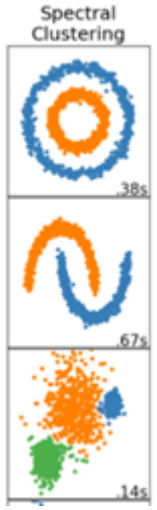
$preference = median(S)$



$preference = -70$



$preference = -1000$

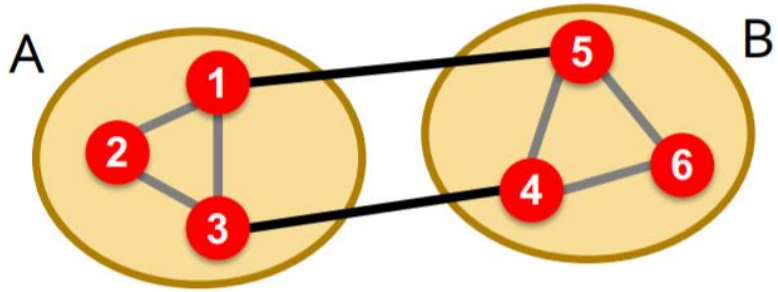


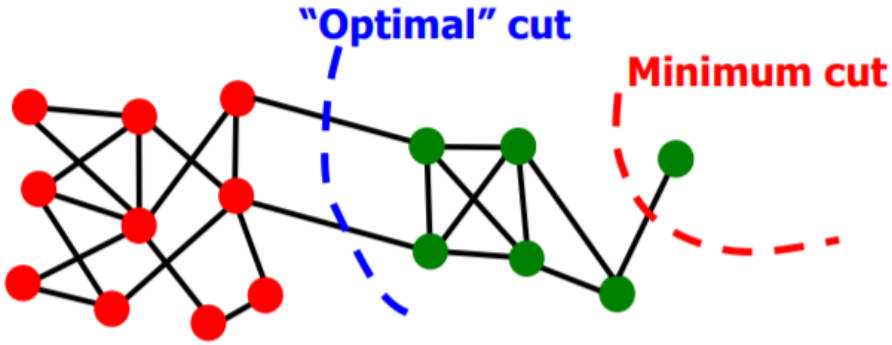
Graph Based Clustering

02

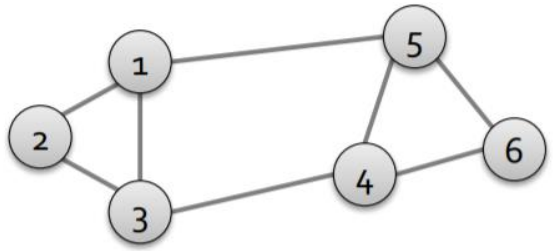
Spectral Clustering

- **Three basic stages:**
  - **1) Pre-processing**
    - Construct a matrix representation of the graph
  - **2) Decomposition**
    - Compute eigenvalues and eigenvectors of the matrix
    - Map each point to a lower-dimensional representation based on one or more eigenvectors
  - **3) Grouping**
    - Assign points to two or more clusters, based on the new representation





Working



A

	1	2	3	4	5	6
1	0	1	1	0	1	0
2	1	0	1	0	0	0
3	1	1	0	1	0	0
4	0	0	1	0	1	1
5	1	0	0	1	0	1
6	0	0	0	1	1	0

D

	1	2	3	4	5	6
1	3	0	0	0	0	0
2	0	2	0	0	0	0
3	0	0	3	0	0	0
4	0	0	0	3	0	0
5	0	0	0	0	3	0
6	0	0	0	0	0	2

$$L = D - A$$

- **1) Pre-processing:**
  - Build Laplacian matrix L of the graph

	1	2	3	4	5	6
1	3	-1	-1	0	-1	0
2	-1	2	-1	0	0	0
3	-1	-1	3	-1	0	0
4	0	0	-1	3	-1	-1
5	-1	0	0	-1	3	-1
6	0	0	0	-1	-1	2

- **2) Decomposition:**
  - Find eigenvalues λ and eigenvectors x of the matrix L
  - Map vertices to corresponding components of x_2

$\lambda =$

0.0
1.0
3.0
3.0
4.0
5.0

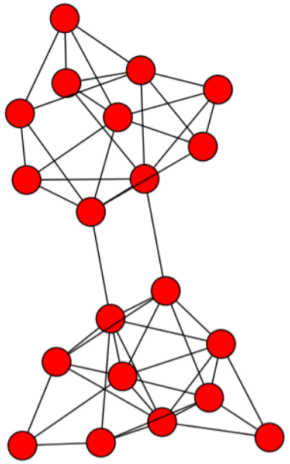
$X =$

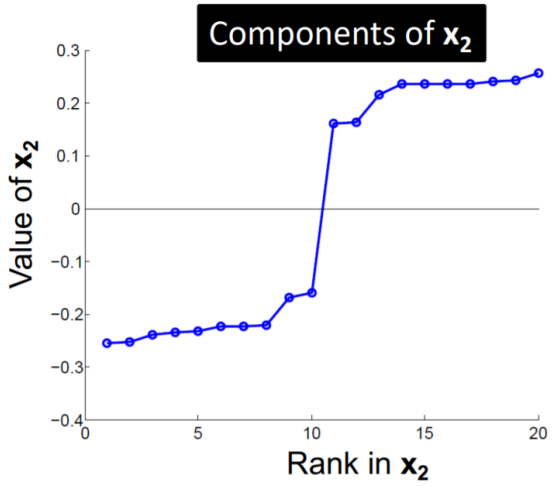
0.4	0.3	-0.5	-0.2	-0.4	-0.5
0.4	0.6	0.4	-0.4	0.4	0.0
0.4	0.3	0.1	0.6	-0.4	0.5
0.4	-0.3	0.1	0.6	0.4	-0.5
0.4	-0.3	-0.5	-0.2	0.4	0.5
0.4	-0.6	0.4	-0.4	-0.4	0.0

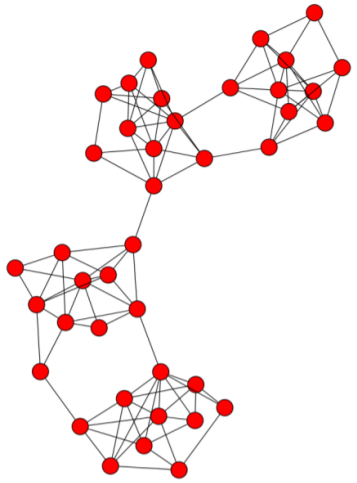
1	0.3
2	0.6
3	0.3
4	-0.3
5	-0.3
6	-0.6

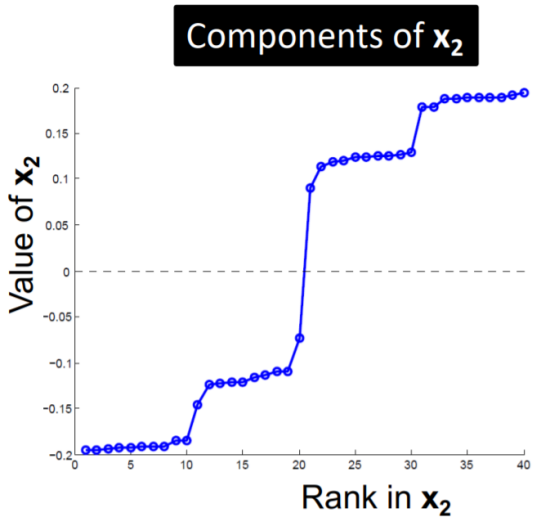
How do we now find the clusters?

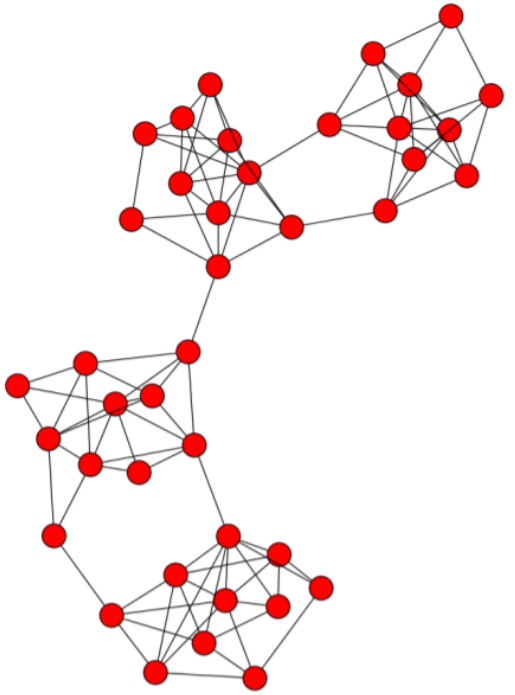
Examples

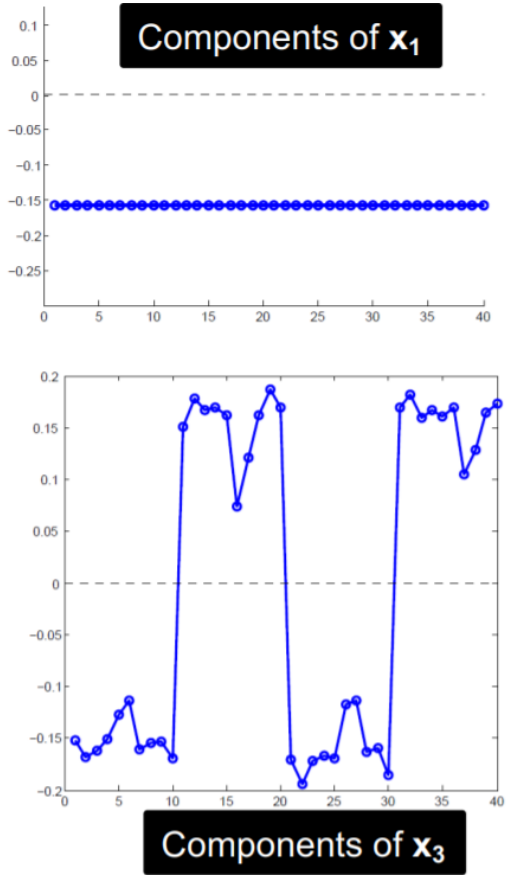



Components of x_2
Value of x_2
Rank in x_2




Components of x_2
Value of x_2
Rank in x_2




Components of x_1
Components of x_3

Spectral Clustering in sklearn

sklearn.cluster.SpectralClustering

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=0.0, assign_labels='kmeans', degree=3, coef0=1, kernel_params=None, n_jobs=None, verbose=False*) [source]

affinity : *str or callable, default='rbf'*

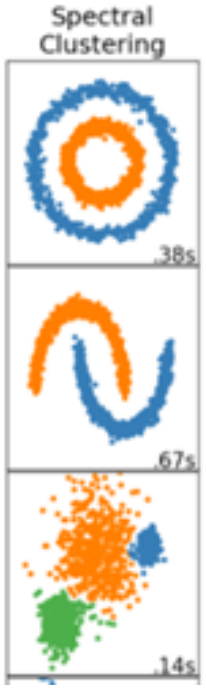
How to construct the affinity matrix.

- '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`.

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.

Pros and Cons

Pros



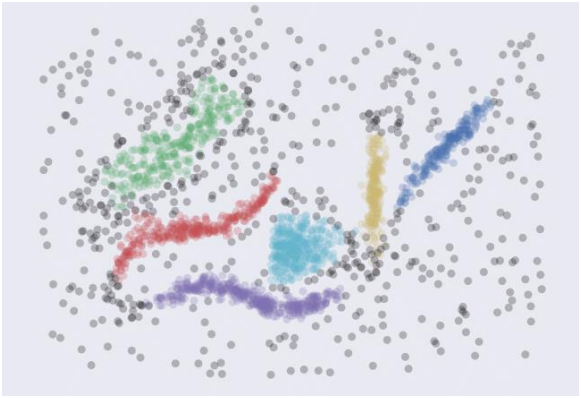
- Doesn’t assume the shape of the data.
- Doesn’t need the actual data, can work with similarity matrix or adjacency matrix

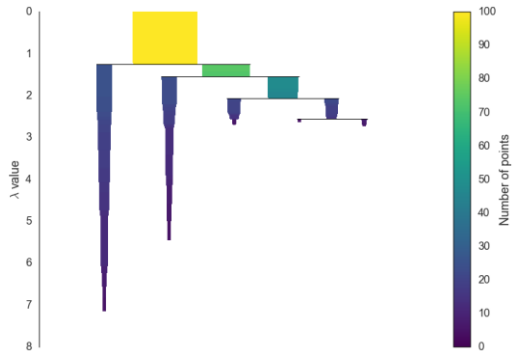
Cons

- Can be costly to compute
- Nosiy dataset causes problems
- Good for datasets containing inherent graph like structures

What will we see in Part-2

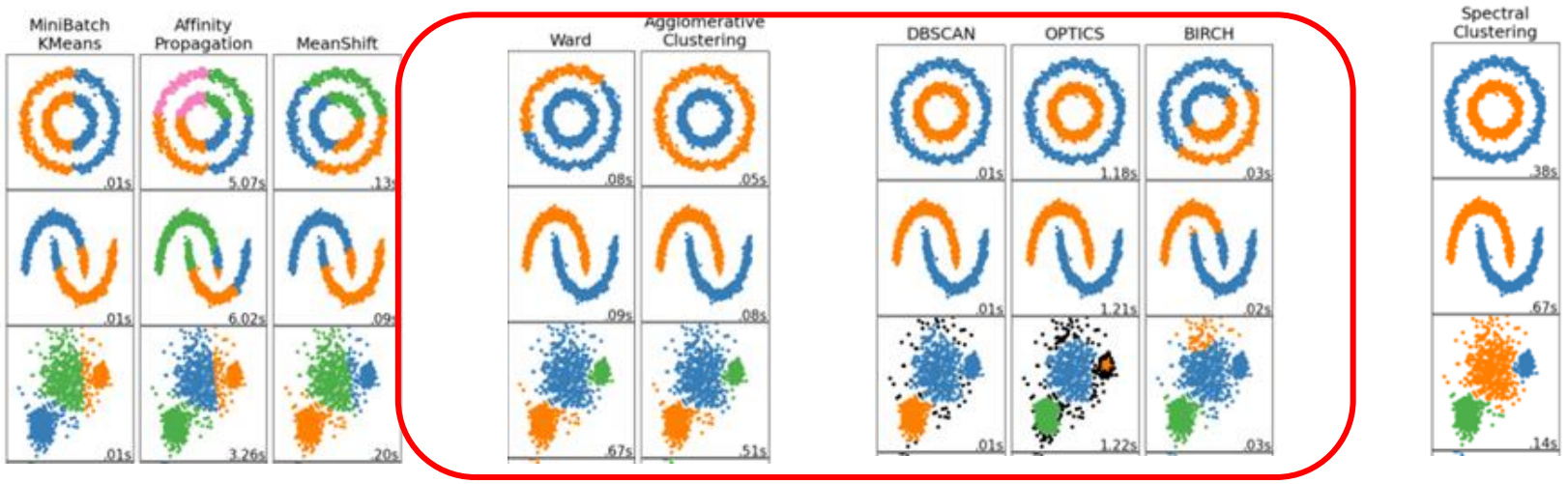
HDBSCAN



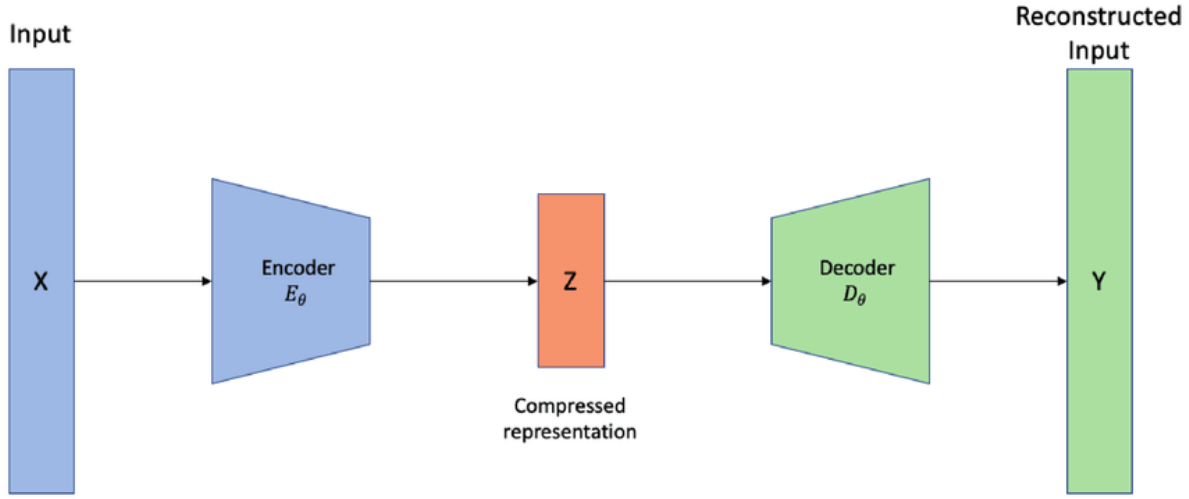


Clustering

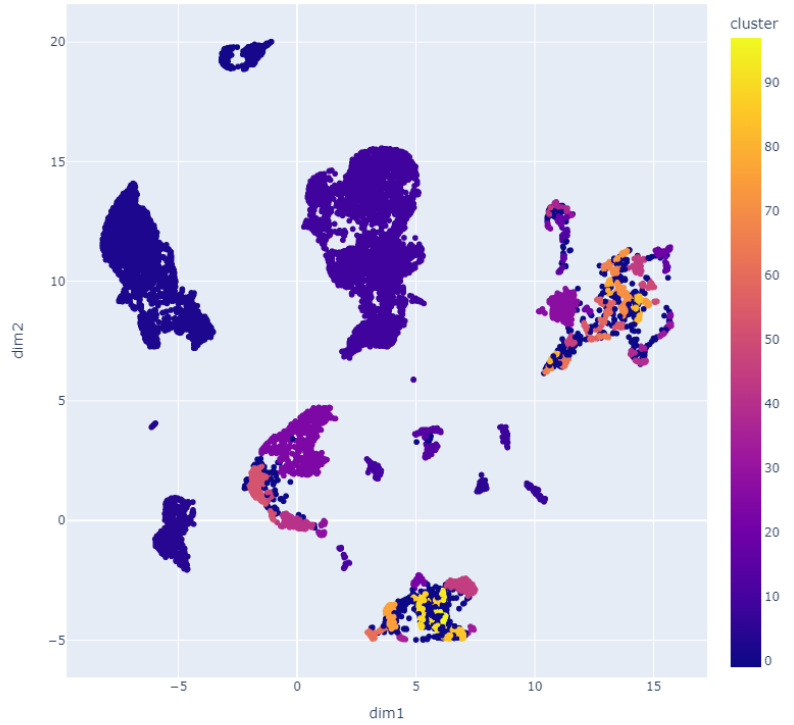
- Centroid Based Clustering
- Hierarchical Based Clustering
- Density Based Clustering
- Graph Based Clustering



Clustering of 3D-data

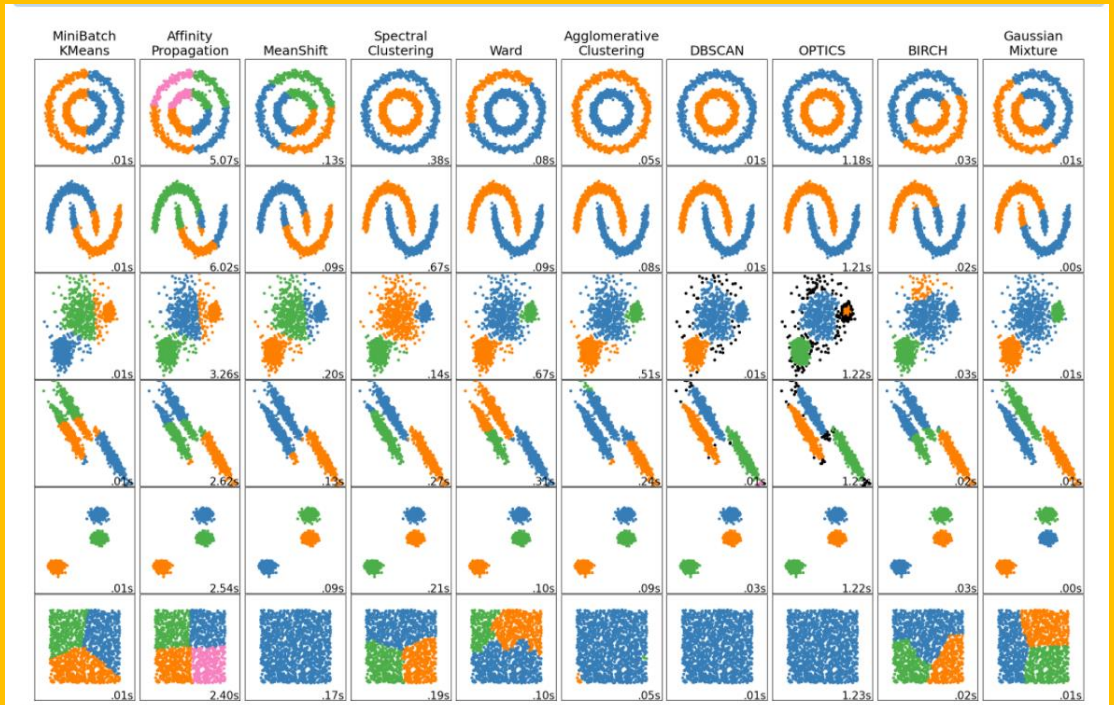


Auto-encoder on 3D data



Clustering on Compressed Representation

Thank You



A comparison of the clustering algorithms in scikit-learn

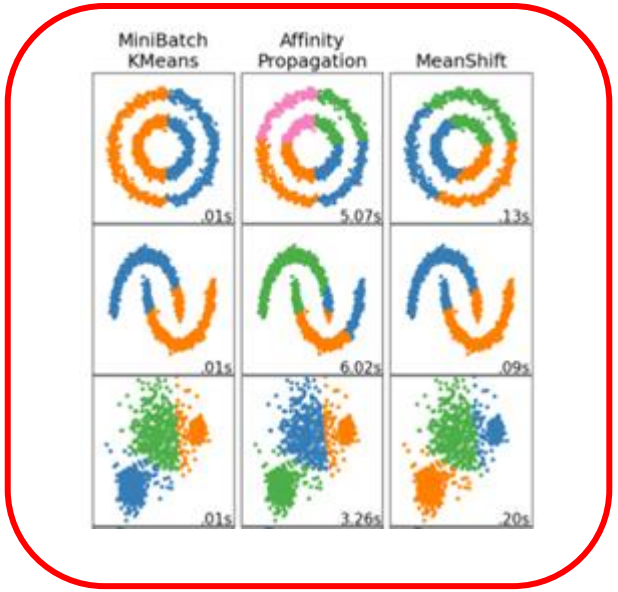
Clustering Algorithms

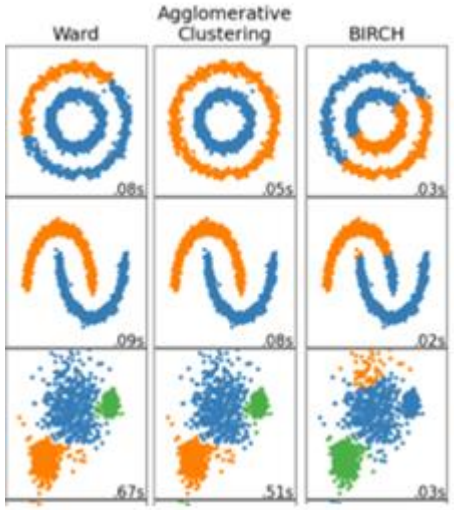
Part-2

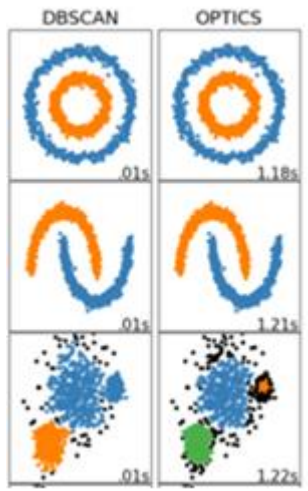
What did we see in Part-1…

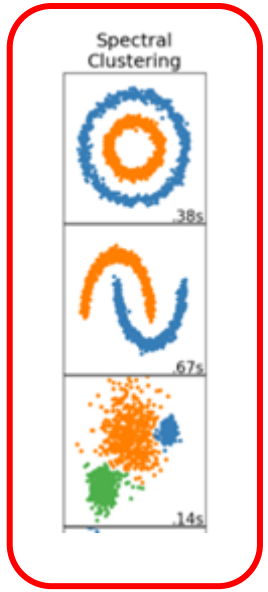
Clustering

- Centroid Based Clustering
- Hierarchical Based Clustering
- Density Based Clustering
- Graph Based Clustering









Agenda

01

Hierarchical Clustering Algos

Agglomerative Clustering

Ward Clustering

02

Density Based Clustering

DBSCAN

OPTICS

03

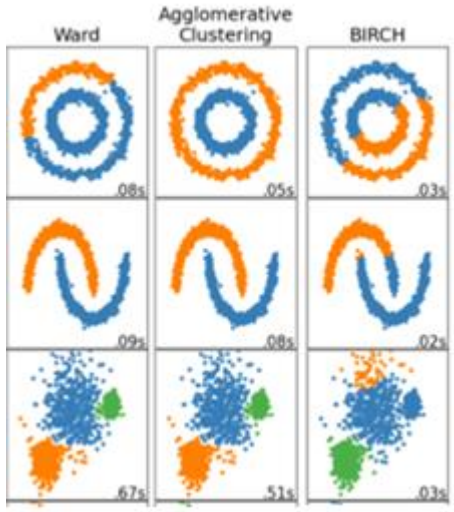
HDBSCAN

Workings

04

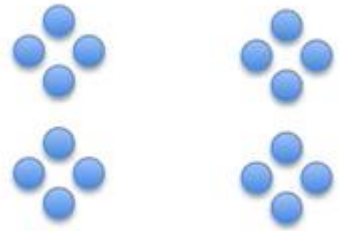
Clustering 3D Data

Autoencoder

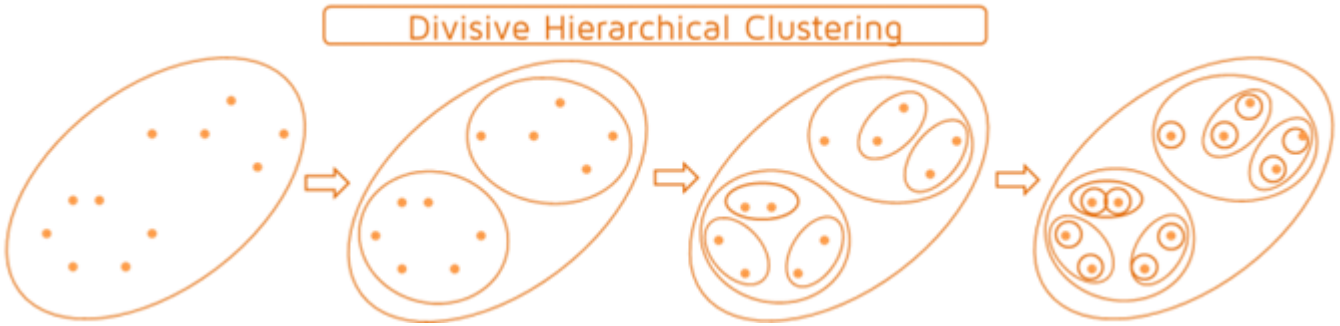


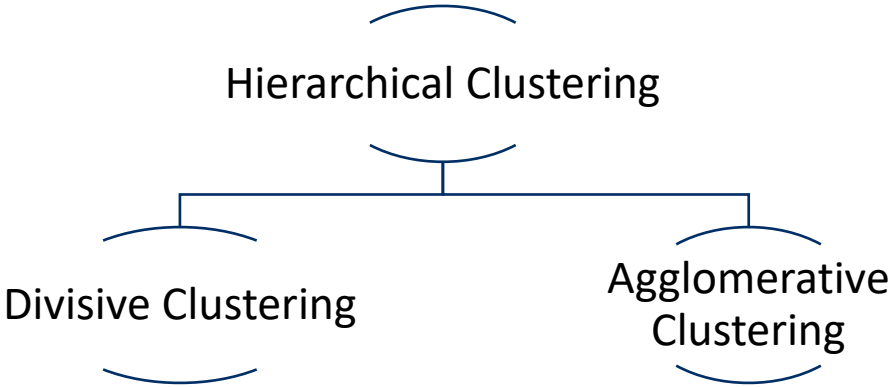
Hierarchical Clustering

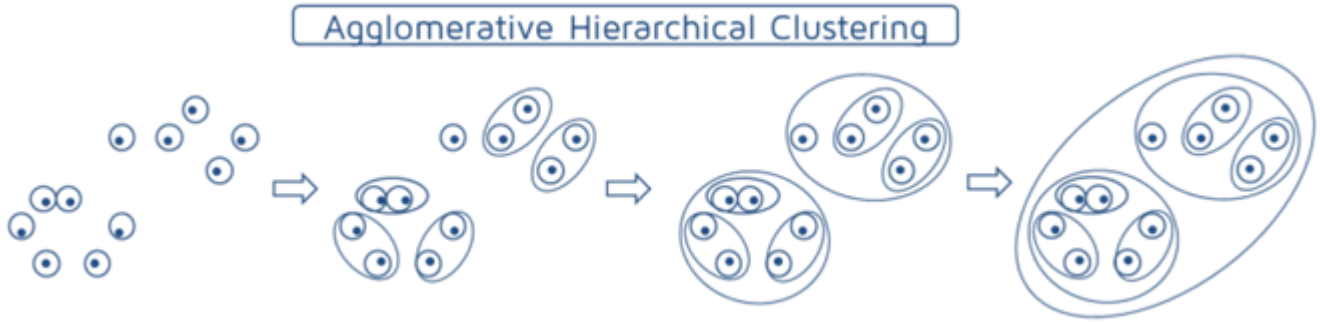
Agglomerative Clustering (Hierarchical Clustering)



Flat vs Hierarchical Clustering



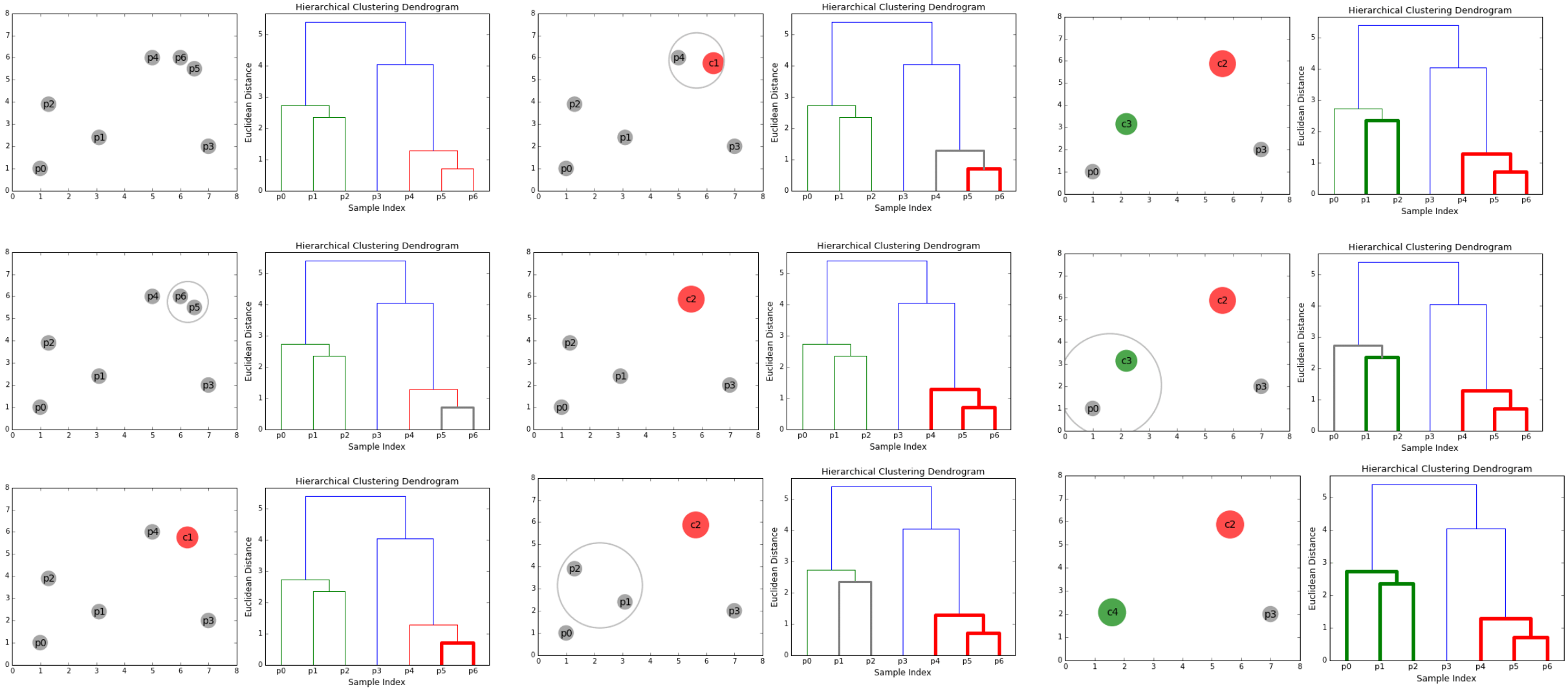




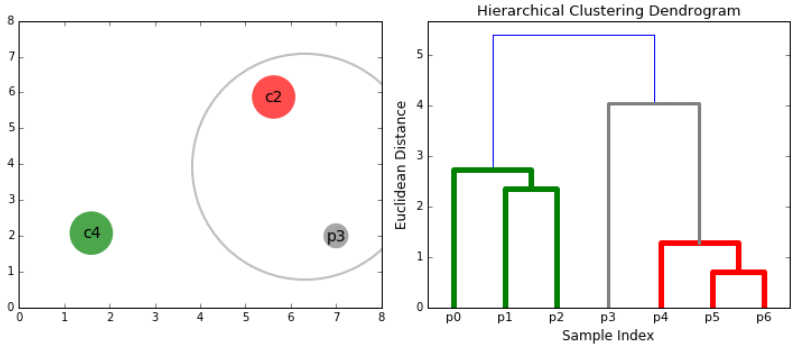
Top Down Approach

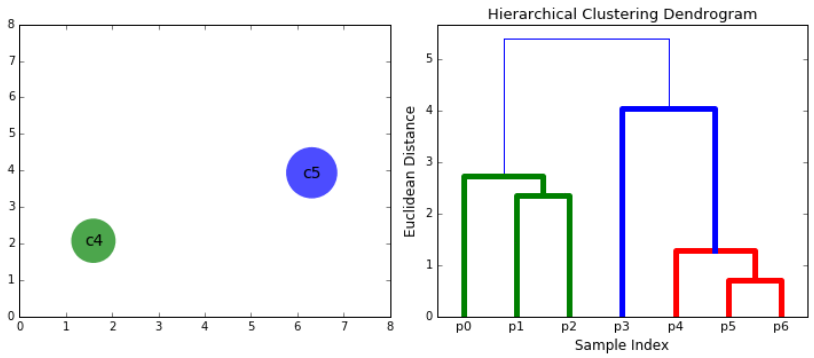
Bottom Up Approach

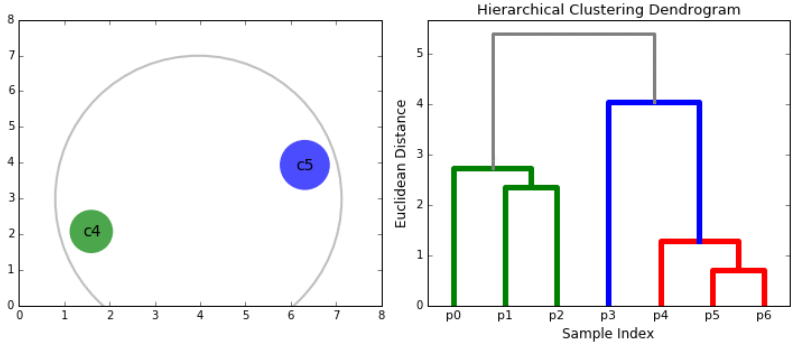
How Agglomerative Clustering Works

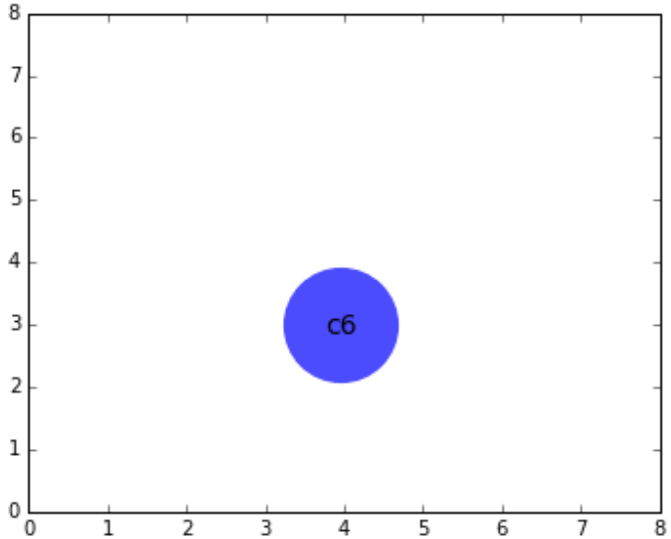


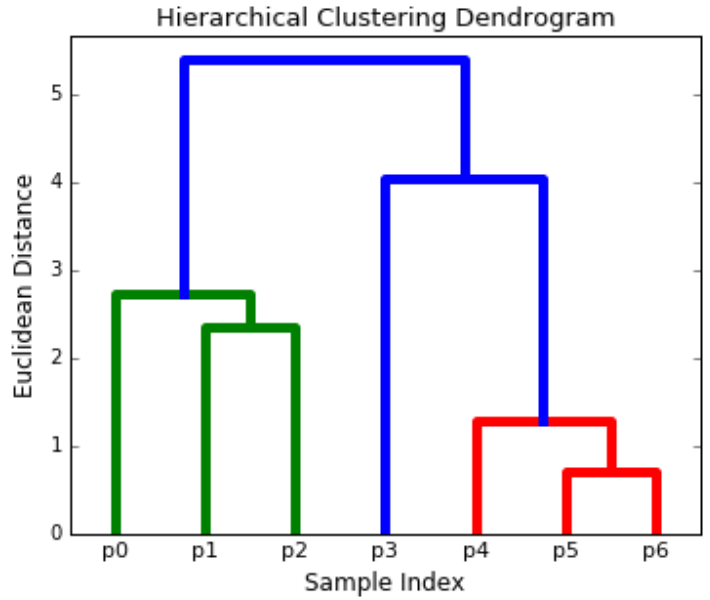
How it works….


c2
c4
p3
Hierarchical Clustering Dendrogram
Euclidean Distance
Sample Index
p0 p1 p2 p3 p4 p5 p6


c5
c4
Hierarchical Clustering Dendrogram
Euclidean Distance
Sample Index
p0 p1 p2 p3 p4 p5 p6

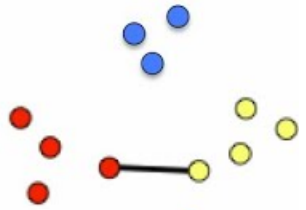

c5
c4
Hierarchical Clustering Dendrogram
Euclidean Distance
Sample Index
p0 p1 p2 p3 p4 p5 p6


c6

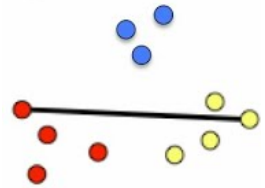

Hierarchical Clustering Dendrogram
Euclidean Distance
Sample Index
p0 p1 p2 p3 p4 p5 p6

Distance Measure for finding Closest Cluster

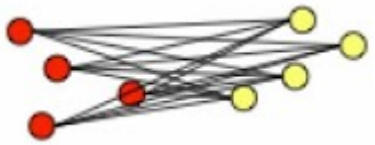
- Single link: $D(c_1,c_2) = \min_{x_1 \in c_1, x_2 \in c_2} D(x_1,x_2)$
  - distance between closest elements in clusters
  - produces long chains a→b→c→…→z



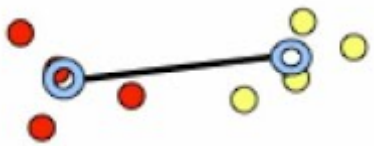
- Complete link: $D(c_1,c_2) = \max_{x_1 \in c_1, x_2 \in c_2} D(x_1,x_2)$
  - distance between farthest elements in clusters
  - forces "spherical" clusters with consistent "diameter"



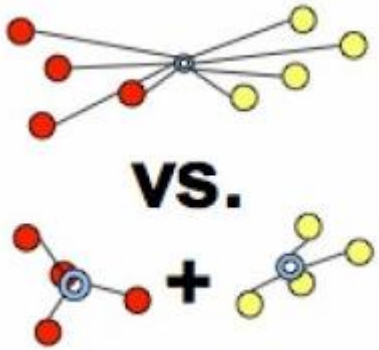
- Average link: $D(c_1,c_2) = \frac{1}{|c_1|}\frac{1}{|c_2|}\sum_{x_1 \in c_1}\sum_{x_2 \in c_2} D(x_1,x_2)$
  - average of all pairwise distances
  - less affected by outliers



- Centroids: $D(c_1,c_2) = D\left(\left(\frac{1}{|c_1|}\sum_{x \in c_1}\vec{x}\right),\left(\frac{1}{|c_2|}\sum_{x \in c_2}\vec{x}\right)\right)$
  - distance between centroids (means) of two clusters



- Ward's method: $TD_{c_1 \cup c_2} = \sum_{x \in c_1 \cup c_2} D(x,\mu_{c_1 \cup c_2})^2$
  - consider joining two clusters, how does it change the total distance (TD) from centroids?



[Hierarchical Clustering Playlist - Victor Lavrenko]

Agglomerative Clustering - sklearn

sklearn.cluster.AgglomerativeClustering

class sklearn.cluster.AgglomerativeClustering(*n_clusters=2, *, affinity='euclidean', memory=None, connectivity=None, compute_full_tree='auto', linkage='ward', distance_threshold=None, compute_distances=False*) [source]

linkage : *{'ward', 'complete', 'average', 'single'}, default='ward'*

Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

- 'ward' minimizes the variance of the clusters being merged.
- 'average' uses the average of the distances of each observation of the two sets.
- 'complete' or 'maximum' linkage uses the maximum distances between all observations of the two sets.
- 'single' uses the minimum of the distances between all observations of the two sets.

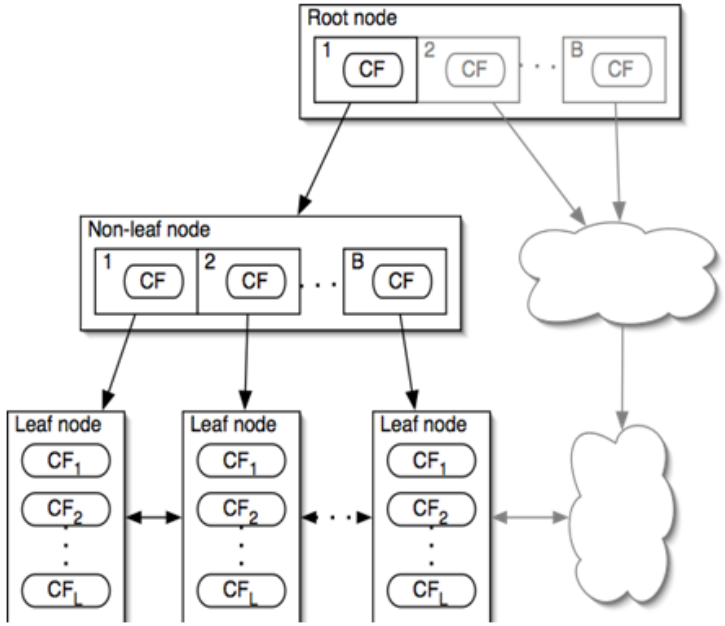
New in version 0.20: Added the 'single' option

- Lecture 59 — Hierarchical Clustering | Stanford University
- https://scikit-learn.org/stable/modules/clustering.html#hierarchical-clustering

Balanced Iterative Reducing and Clustering using Hierarchies - BIRCH

- Existing algorithm scales poorly with processing large datasets
- BIRCH algorithm is specifically used for handling large datasets by first generating compact representation of the data.

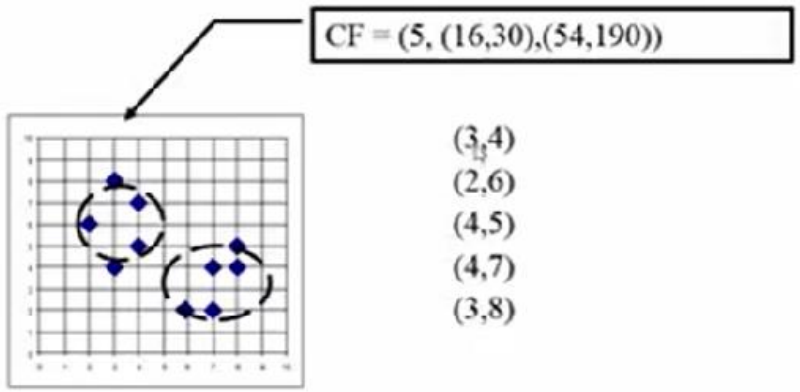
Step 1 – a rough clustering is performed – building a Clustering Feature (CF) tree
Step 2 – an arbitrary clustering algorithm is used to cluster the leaf node of CF tree



- https://www.youtube.com/watch?v=PMBkL9lkoq4
- https://www.youtube.com/watch?v=jALZ8dj8FXU

Clustering Features

- Clustering Feature (CF): *CF = (N, LS, SS)*
  - *N*: Number of data points
  - *LS: linear sum of N points:* $\sum_{i=1}^{N} X_i$
  - *SS: square sum of N points:* $\sum_{i=1}^{N} X_i^2$



- Centroid: $\vec{x}_0$
  - the "middle" of a cluster
  - n: number of points in a cluster
  - $\vec{x}_i$ is the *i*-th point in the cluster

$$\vec{x}_0 = \frac{\sum_{i}^{n} \vec{x}_i}{n}$$

- Radius: R
  - Average distance from member objects to the centroid
  - The square root of average distance from any point of the cluster to its centroid

$$R = \sqrt{\frac{\sum_{i}^{n} (\vec{x}_i - \vec{x}_0)^2}{n}}$$

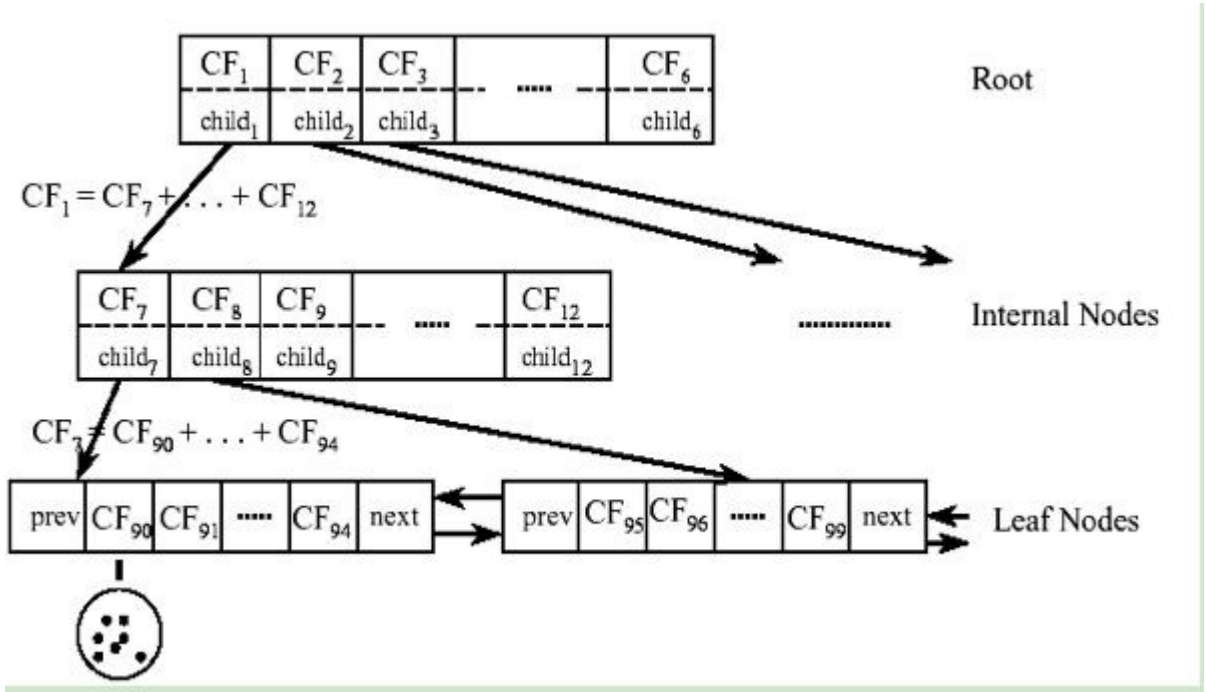
- Diameter: D
  - Average pairwise distance within a cluster
  - The square root of average mean squared distance between all pairs of points in the cluster

$$D = \sqrt{\frac{\sum_{i}^{n} \sum_{j}^{n} (\vec{x}_i - \vec{x}_j)^2}{n(n-1)}}$$

- DATA MINING 5 Cluster Analysis in Data Mining 4 5 BIRCH A Micro Clustering Based Approach – YouTube
- https://towardsdatascience.com/machine-learning-birch-clustering-algorithm-clearly-explained-fb9838cbeed9

Building of the Clustering Feature Tree

- **threshold:** The radius of the subcluster obtained by merging a new sample and the closest subcluster should be lesser than the threshold.
- **branching_factor:** Maximum number of CF subclusters in each node



BRICH - sklearn

sklearn.cluster.Birch

class sklearn.cluster.Birch(*, *threshold=0.5*, *branching_factor=50*, *n_clusters=3*, *compute_labels=True*, *copy=True*) ¶ [source]

Parameters:

threshold : *float, default=0.5*

The radius of the subcluster obtained by merging a new sample and the closest subcluster should be lesser than the threshold. Otherwise a new subcluster is started. Setting this value to be very low promotes splitting and vice-versa.

branching_factor : *int, default=50*

Maximum number of CF subclusters in each node. If a new samples enters such that the number of subclusters exceed the branching_factor then that node is split into two nodes with the subclusters redistributed in each. The parent subcluster of that node is removed and two new subclusters are added as parents of the 2 split nodes.

n_clusters : *int, instance of sklearn.cluster model, default=3*

Number of clusters after the final clustering step, which treats the subclusters from the leaves as new samples.

- `None` : the final clustering step is not performed and the subclusters are returned as they are.
- `sklearn.cluster` Estimator : If a model is provided, the model is fit treating the subclusters as new samples and the initial data is mapped to the label of the closest subcluster.
- `int` : the model fit is `AgglomerativeClustering` with `n_clusters` set to be equal to the int.

compute_labels : *bool, default=True*

Whether or not to compute labels for each fit.

copy : *bool, default=True*

Whether or not to make a copy of the given data. If set to False, the initial data will be overwritten.

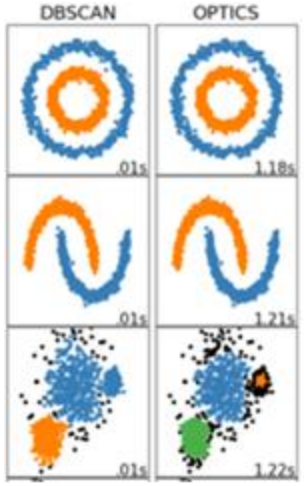
Pros and Cons

Pros

- Efficient for clustering large dataset that can’t fit into the memory
- Different clustering algorithms can be used on the leaf nodes

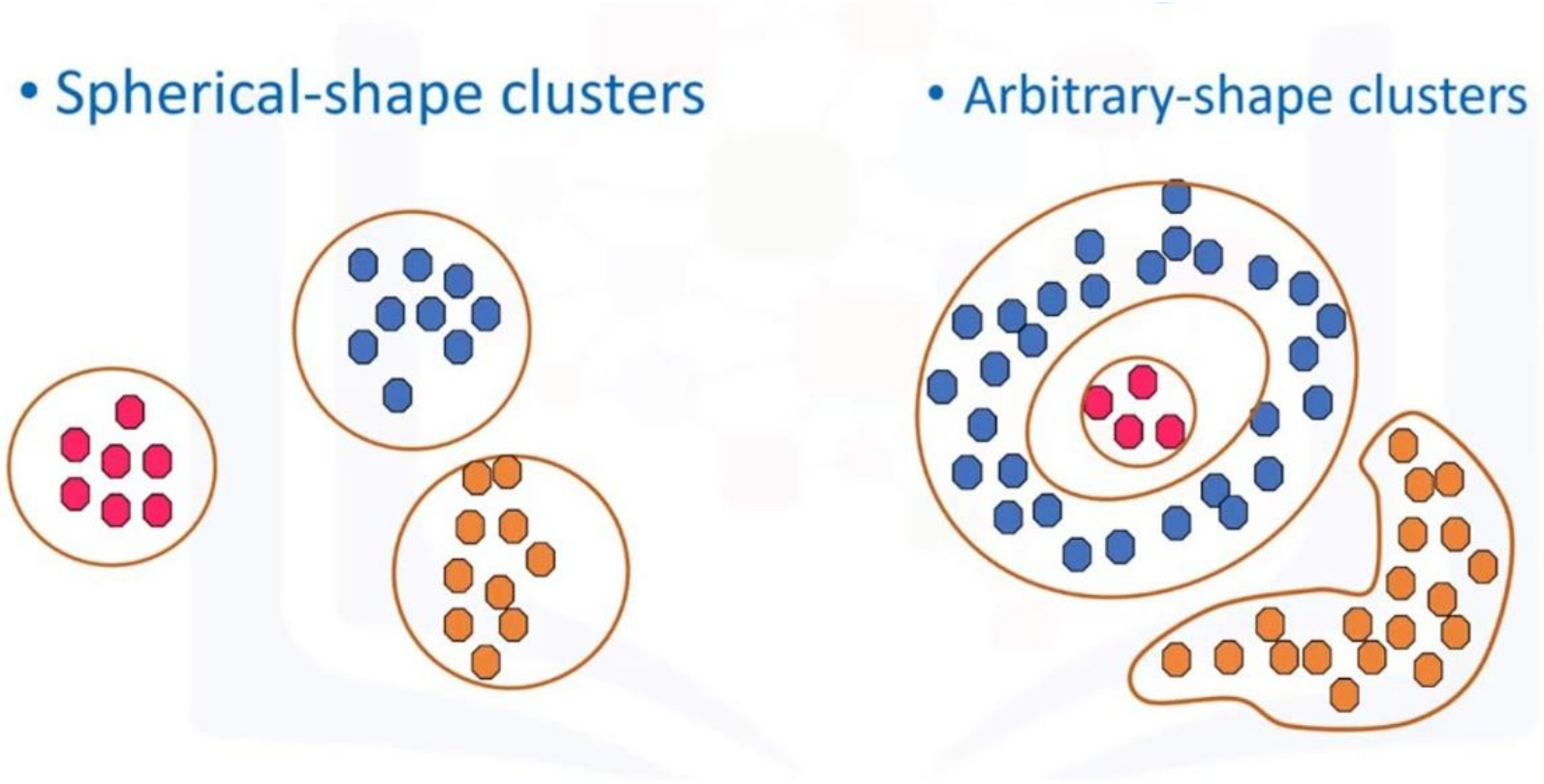
Cons

- Only works with numerical data
- Insertion order affects the CF tree
- Due to fixed size of the leaf nodes, final clusters may not be natural
- Clusters tend to be spherical due to the radius and diameter measures



Density Based Clustering

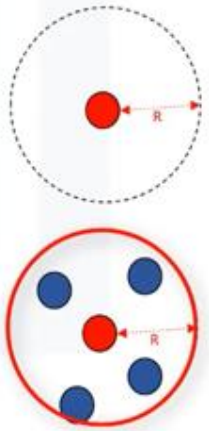
Density Based Spatial Clustering of Applications with Noise-DBSCAN

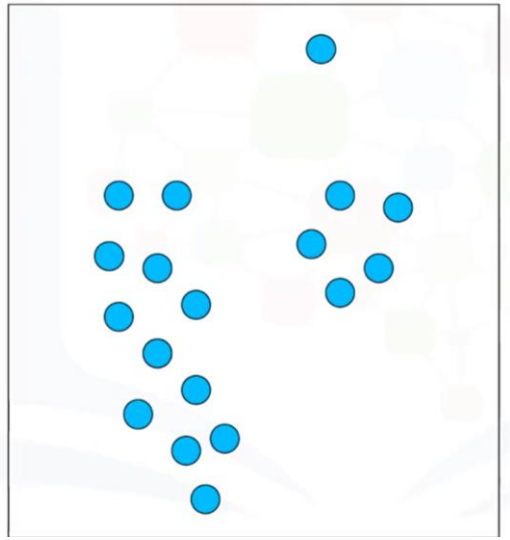


https://www.youtube.com/watch?v=6jl9KkmgDIw

How does it work?

- R (**R**adius of neighborhood)
  - Radius (R) that if includes enough number of points within, we call it a dense area
- M (**M**in number of neighbors)
  - The minimum number of data points we want in a neighborhood to define a cluster

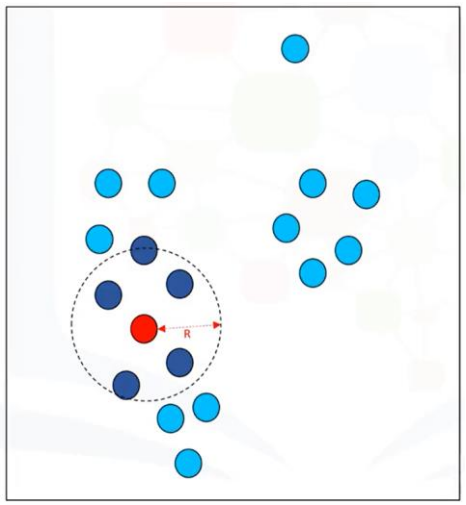


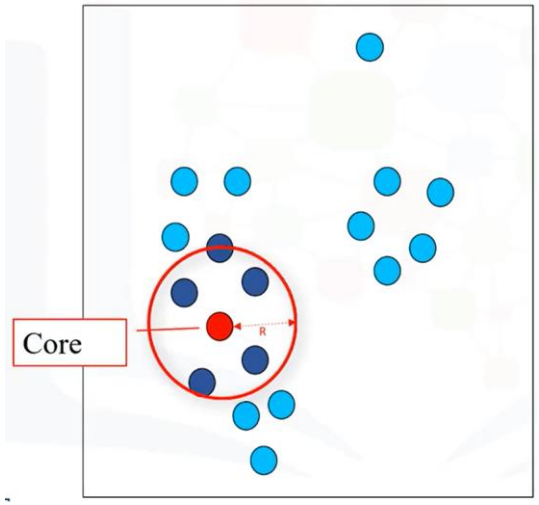


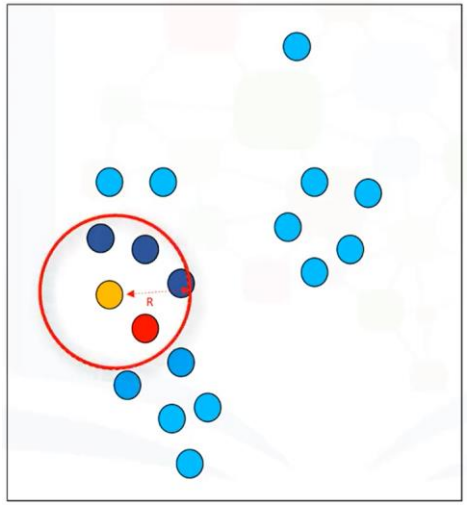
Each point is either:
- *core point*
- *border point*
- *outlier point*

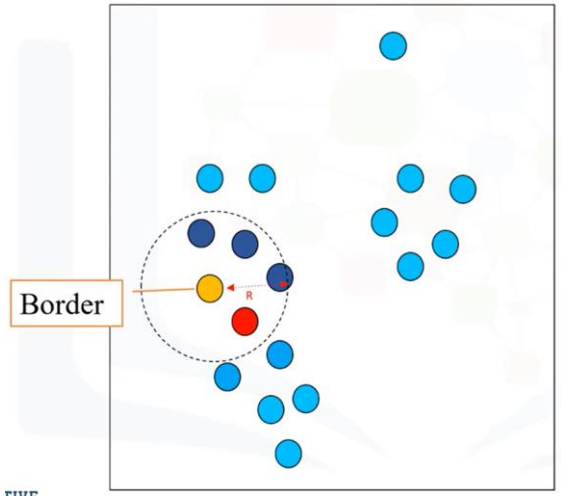
R = 2unit , M = 6

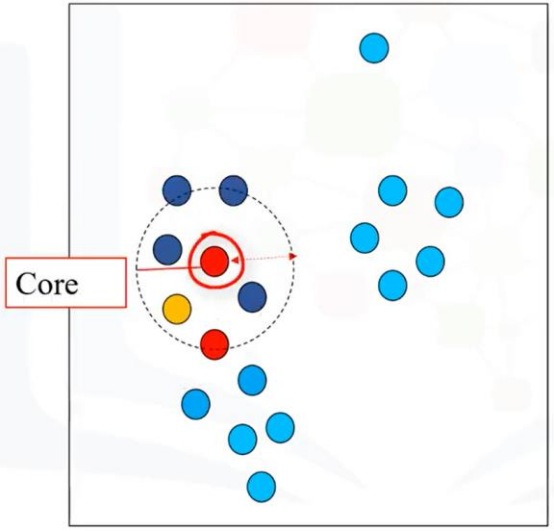
Workings of DBSCAN

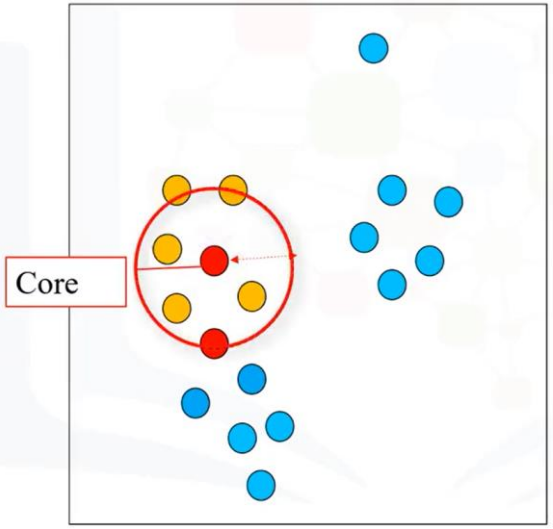


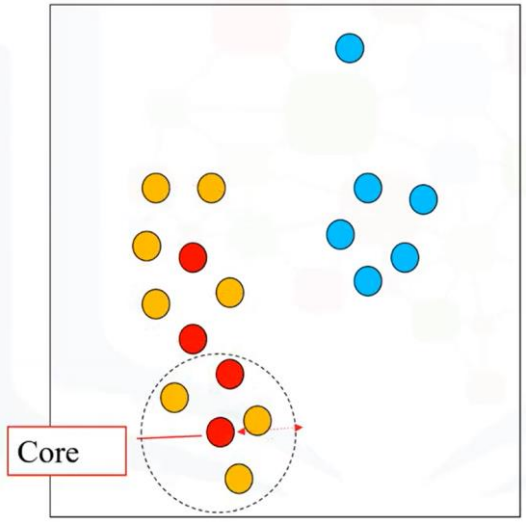


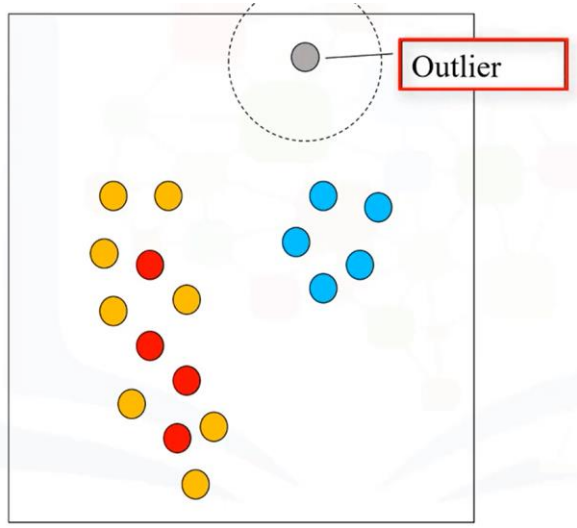




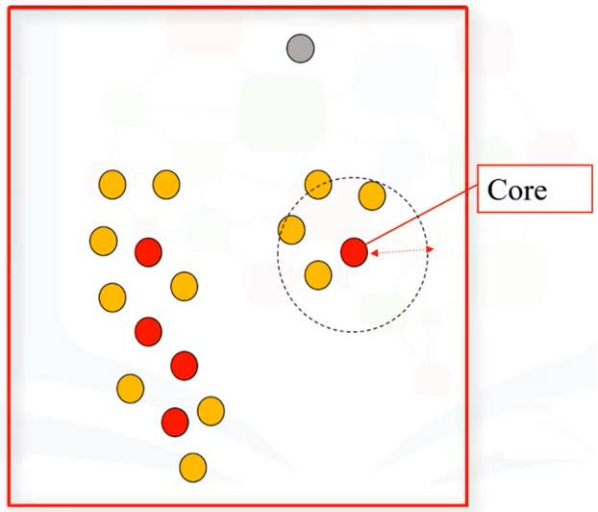


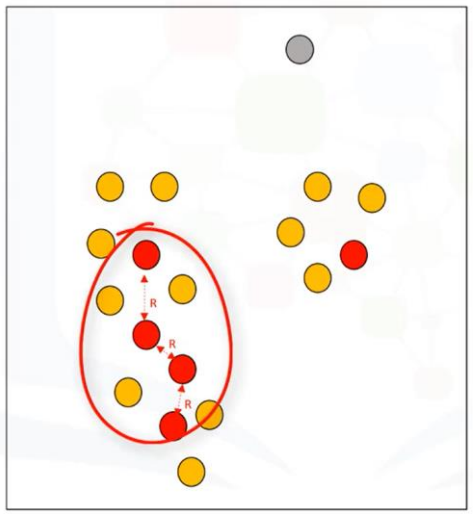


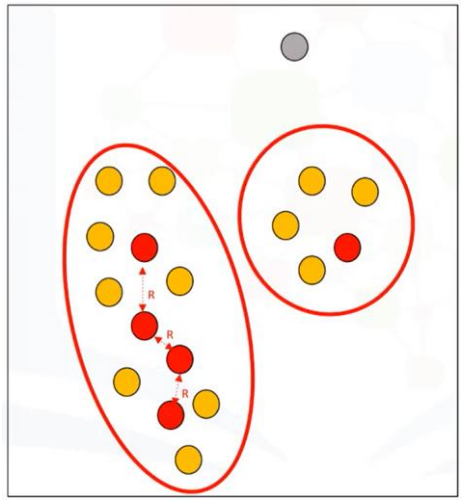


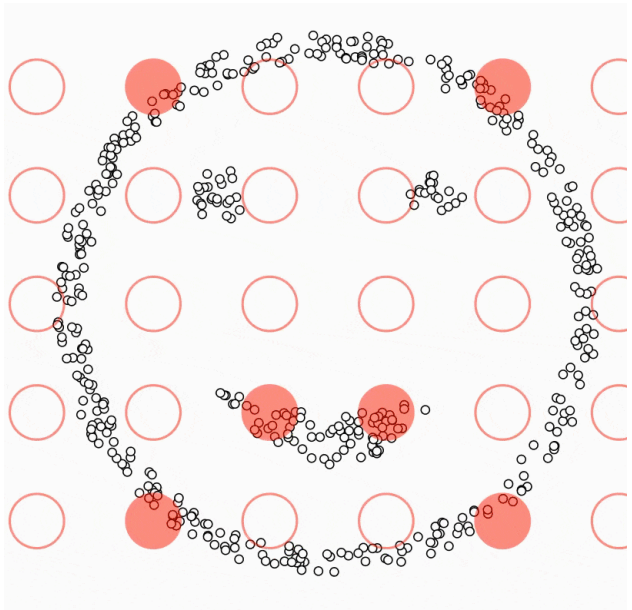


Workings contd…







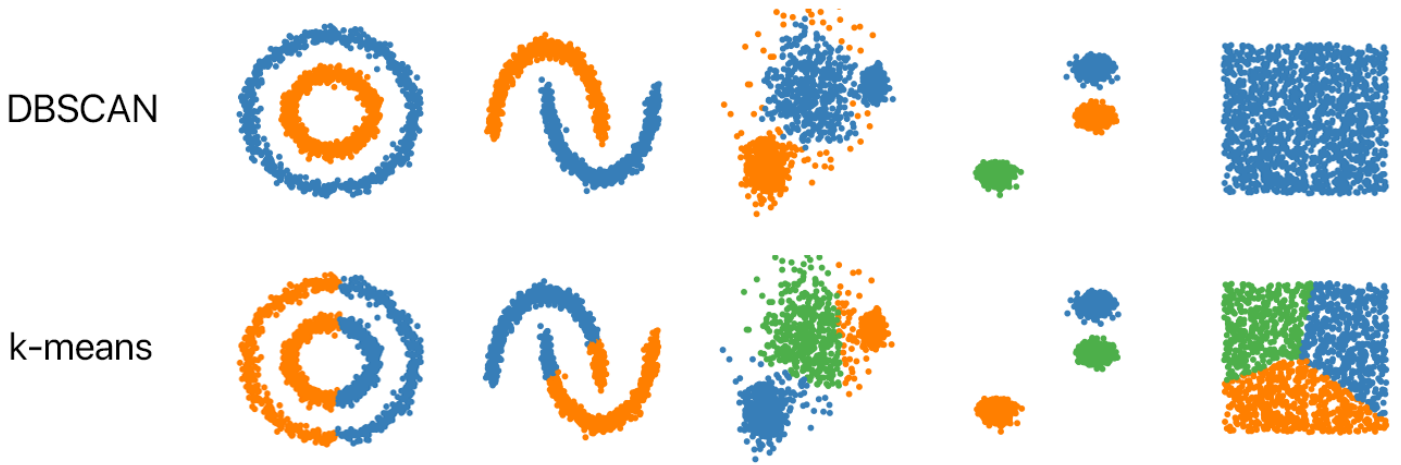


https://www.naftaliharris.com/blog/visualizing-dbscan-clustering/

Pros

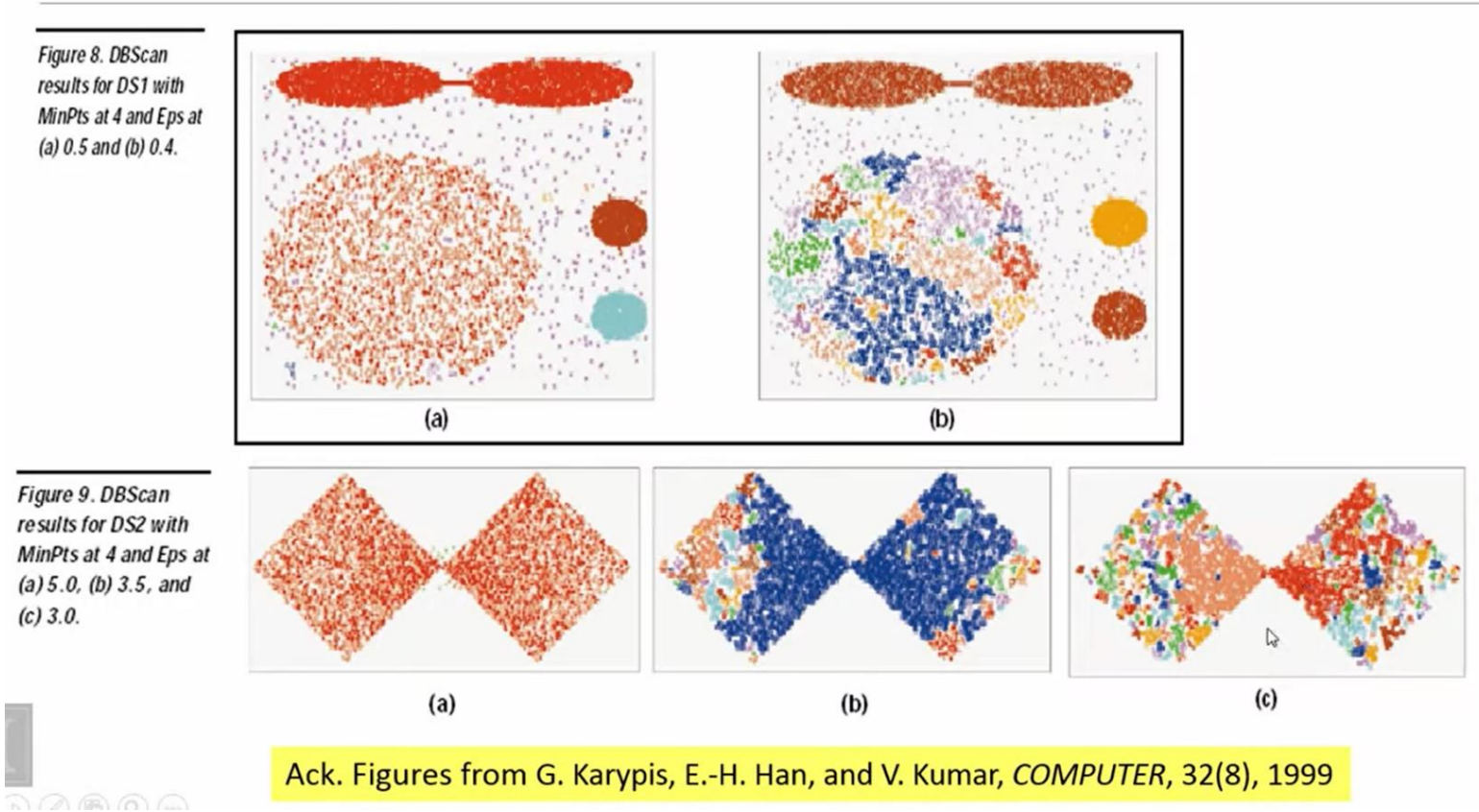
Advantages [edit]

1. DBSCAN does not require one to specify the number of clusters in the data a priori, as opposed to k-means.
2. DBSCAN can find arbitrarily-shaped clusters. It can even find a cluster completely surrounded by (but not connected to) a different cluster. Due to the MinPts parameter, the so-called single-link effect (different clusters being connected by a thin line of points) is reduced.
3. DBSCAN has a notion of noise, and is robust to outliers.
4. DBSCAN requires just two parameters and is mostly insensitive to the ordering of the points in the database. (However, points sitting on the edge of two different clusters might swap cluster membership if the ordering of the points is changed, and the cluster assignment is unique only up to isomorphism.)
5. DBSCAN is designed for use with databases that can accelerate region queries, e.g. using an R* tree.
6. The parameters minPts and ε can be set by a domain expert, if the data is well understood.



Cons

- DBSCAN is sensitive to the setting of parameters
- Doesn’t handle different density clusters very well
- Doesn’t work well in high dimensionality
- DBSCAN is not entirely deterministic. That's because the algorithm starts with a random point. Therefore, border points that are reachable from more than one cluster can be part of either cluster



Ack. Figures from G. Karypis, E.-H. Han, and V. Kumar, *COMPUTER*, 32(8), 1999

- https://www.youtube.com/watch?v=sKRUfsc8zp4
- https://www.youtube.com/watch?v=0Os5AydChR0

DBSCAN API – sklearn

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 - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

Parameters:

eps : *float, default=0.5*

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 : *int, default=5*

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.

metric : *string, or callable, default='euclidean'*

The metric to use when calculating distance between instances in a feature array. If metric is a string or callable, it must be one of the options allowed by `sklearn.metrics.pairwise_distances` for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square. X may be a Glossary, in which case only "nonzero" elements may be considered neighbors for DBSCAN.

New in version 0.17: metric *precomputed* to accept precomputed sparse matrix.

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

OPTICS - Ordering Points to Identify the Clustering Structure

OPTICS is an extension of the DBSCAN algorithm, developed by the same authors of DBSCAN

OPTICS algorithm

From Wikipedia, the free encyclopedia

Ordering points to identify the clustering structure (**OPTICS**) is an algorithm for finding density-based[1] clusters in spatial data. It was presented by Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel and Jörg Sander.[2] Its basic idea is similar to DBSCAN,[3] but it addresses one of DBSCAN's major weaknesses: the problem of detecting meaningful clusters in data of varying density. To do so, the points of the database are (linearly) ordered such that spatially closest points become neighbors in the ordering. Additionally, a special distance is stored for each point that represents the density that must be accepted for a cluster so that both points belong to the same cluster. This is represented as a dendrogram.

https://www.youtube.com/watch?v=GCnsmjwV3BE&list=RDQM5kYy1aM6fgE&start_radio=1

Hierarchical Density-based Spatial Clustering of Applications with Noise - HDBSCAN

HDBSCAN - Hierarchical Density-Based Spatial Clustering of Applications with Noise. Performs DBSCAN over varying epsilon values and integrates the result to find a clustering that gives the best stability over epsilon. This allows HDBSCAN to find clusters of varying densities (unlike DBSCAN), and be more robust to parameter selection.

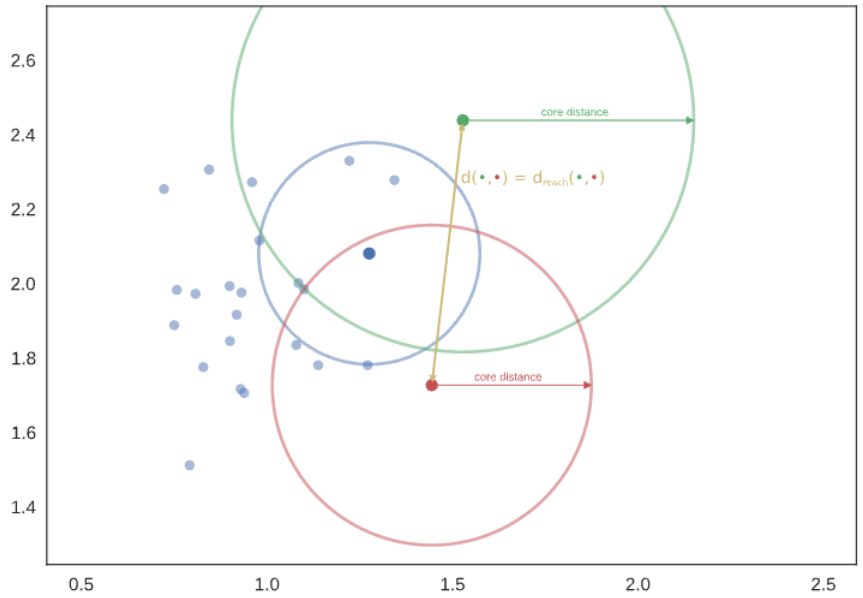
In practice this means that HDBSCAN returns a good clustering straight away with little or no parameter tuning -- and the primary parameter, minimum cluster size, is intuitive and easy to select.

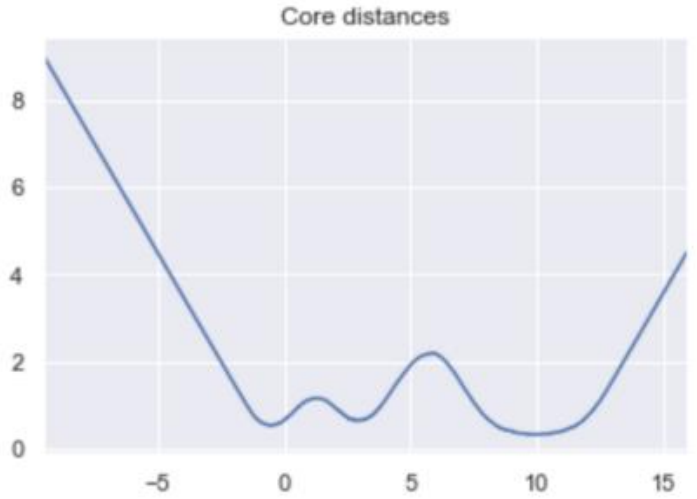
HDBSCAN is ideal for exploratory data analysis; it's a fast and robust algorithm that you can trust to return meaningful clusters (if there are any).

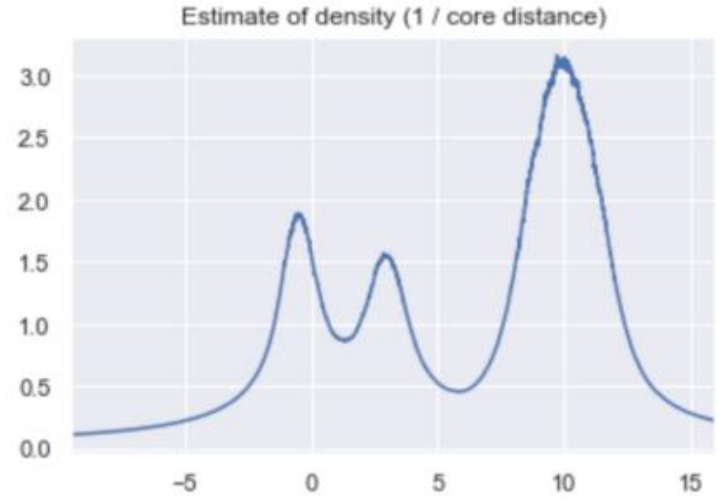
```
import hdbscan
from sklearn.datasets import make_blobs

data, _ = make_blobs(1000)

clusterer = hdbscan.HDBSCAN(min_cluster_size=10)
cluster_labels = clusterer.fit_predict(data)
```

How it works?

1. Estimate the densities
2. Generate Hierarchy
3. Extract Clusters

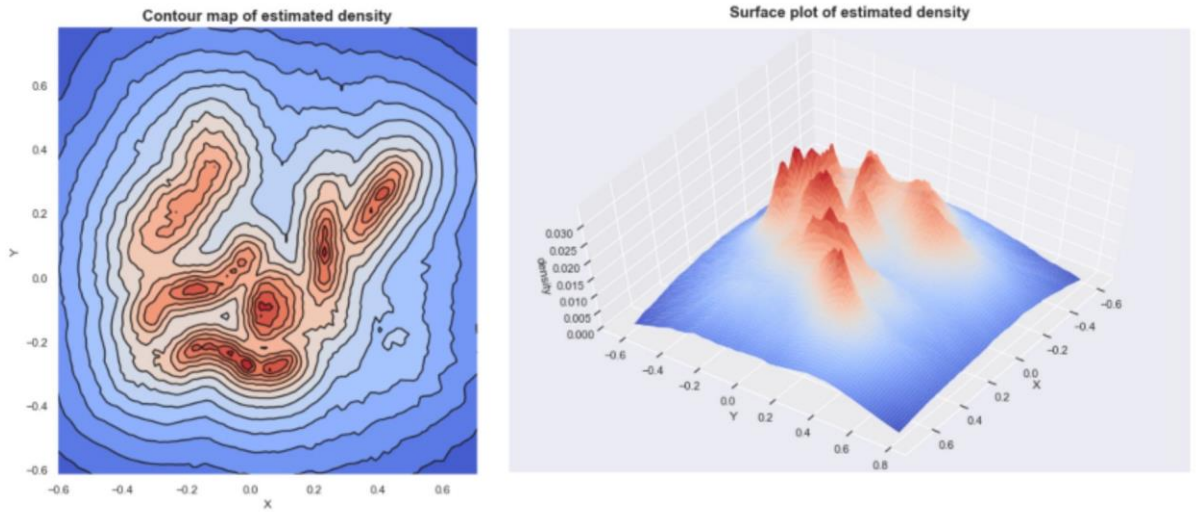


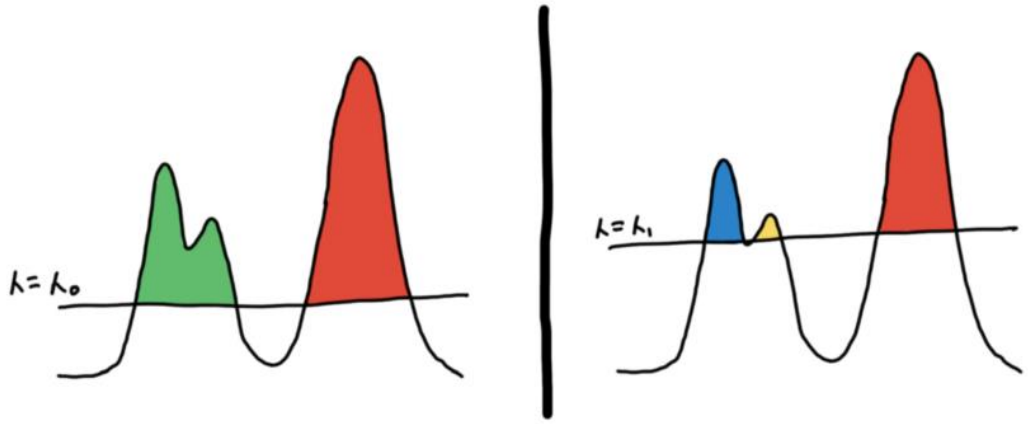


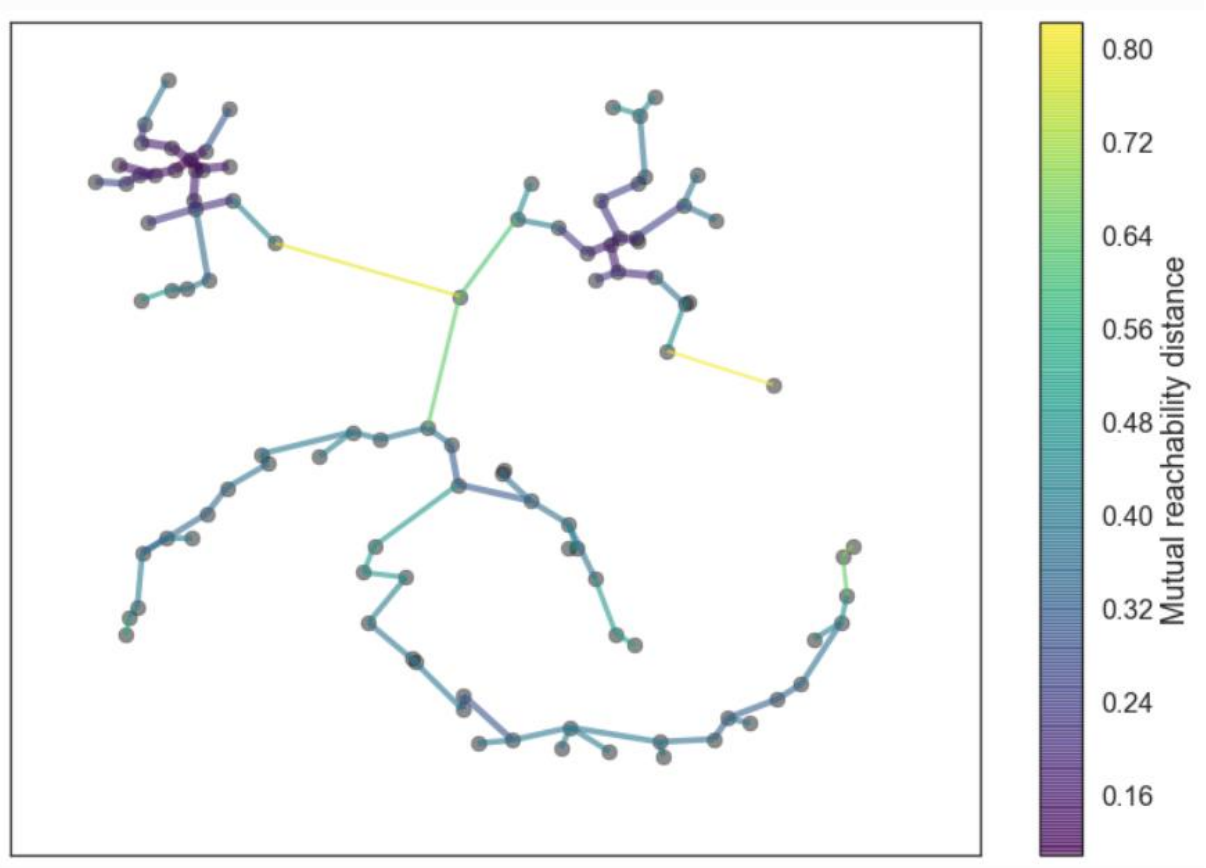


- https://towardsdatascience.com/a-gentle-introduction-to-hdbscan-and-density-based-clustering-5fd79329c1e8
- https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

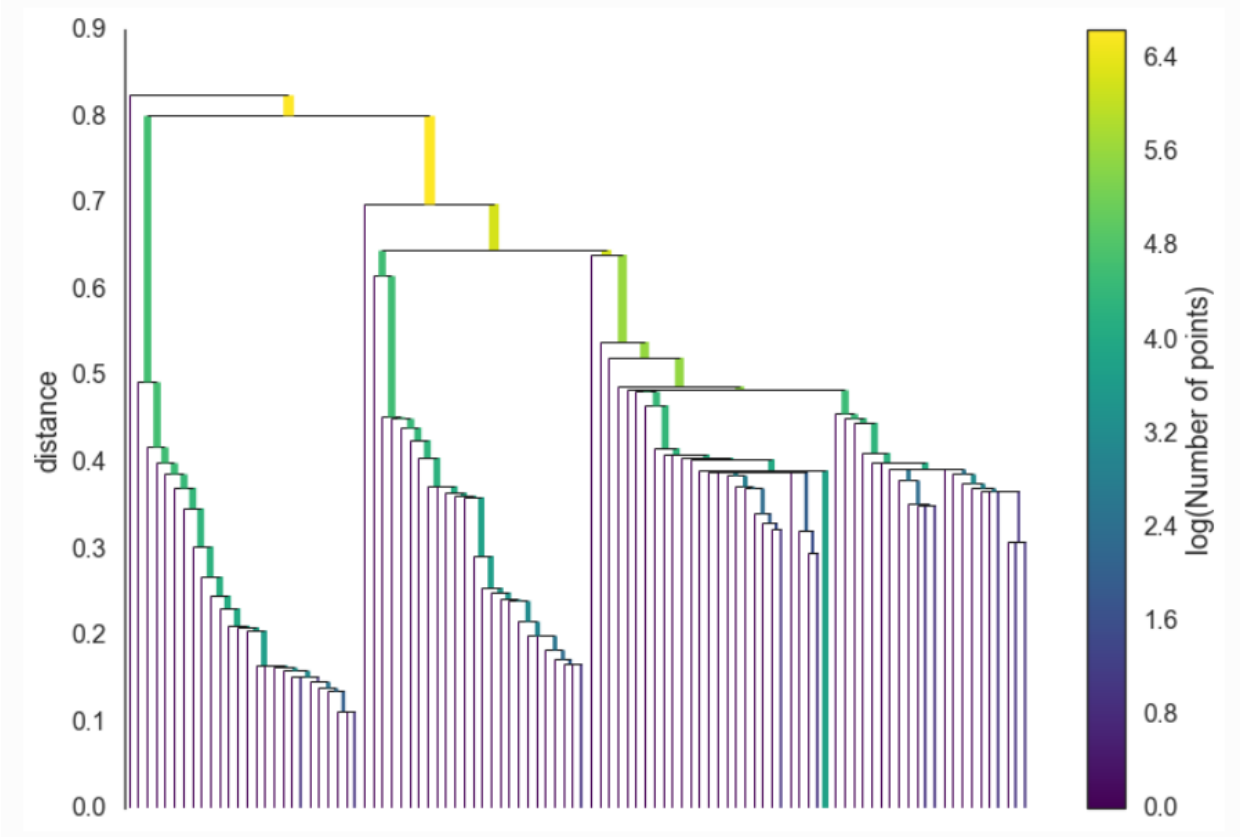
Sea vs Island

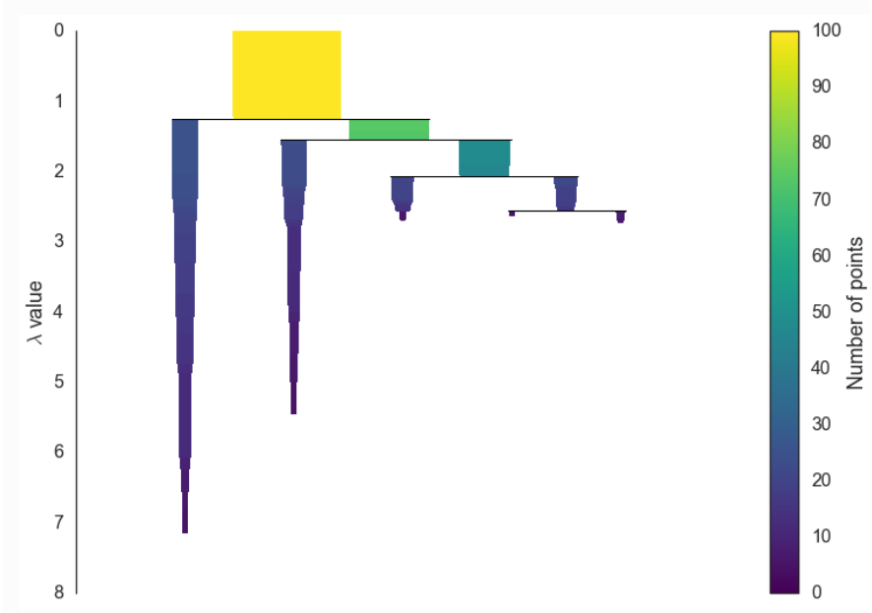


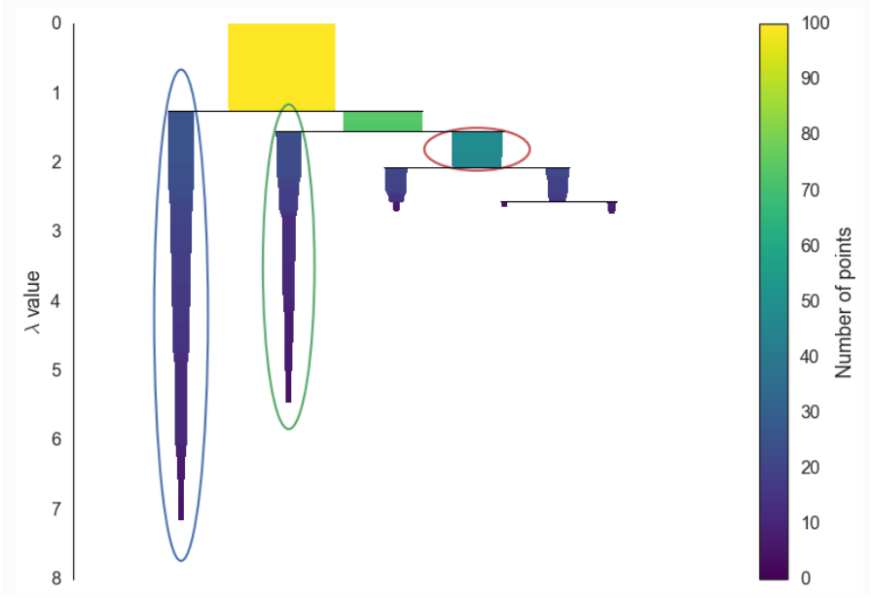




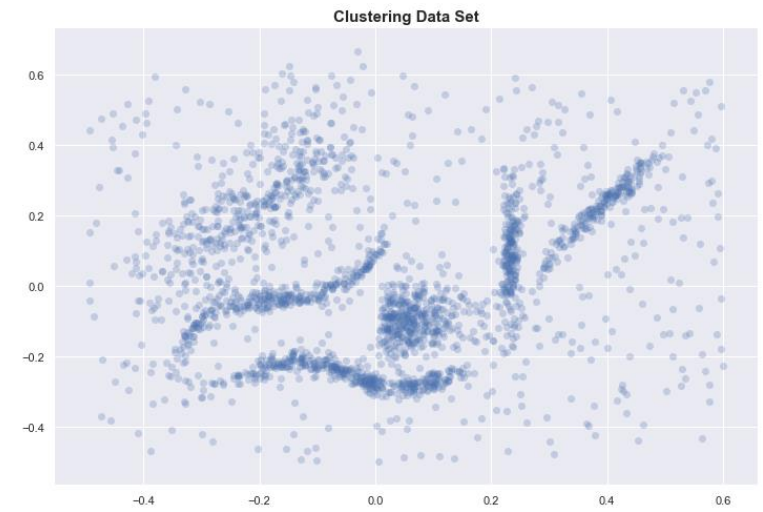
Generating Clusters

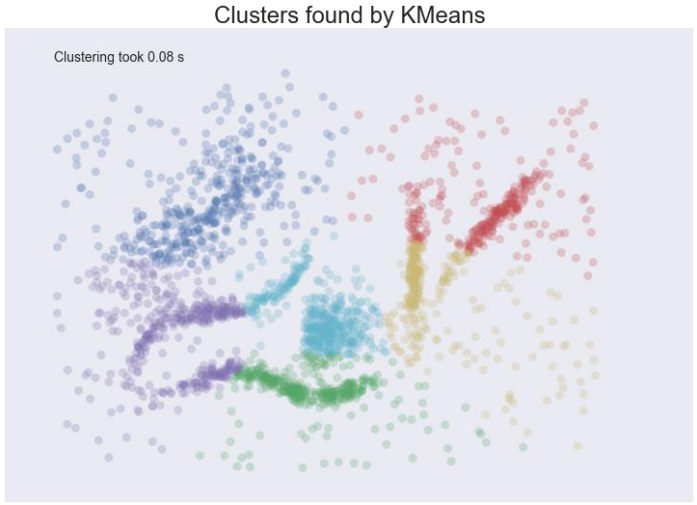


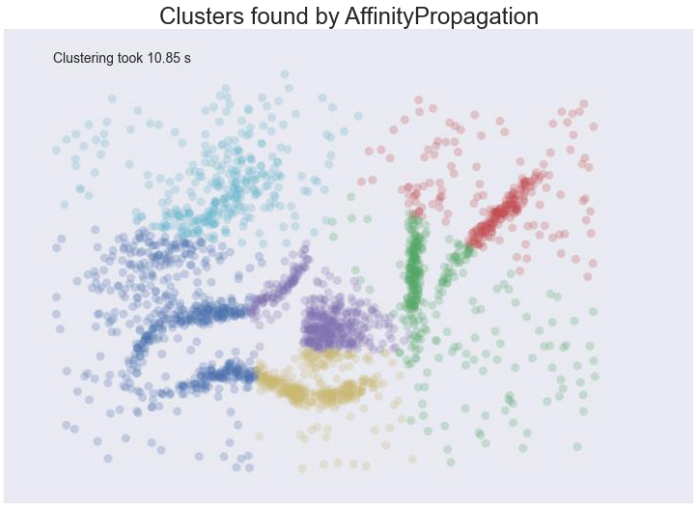


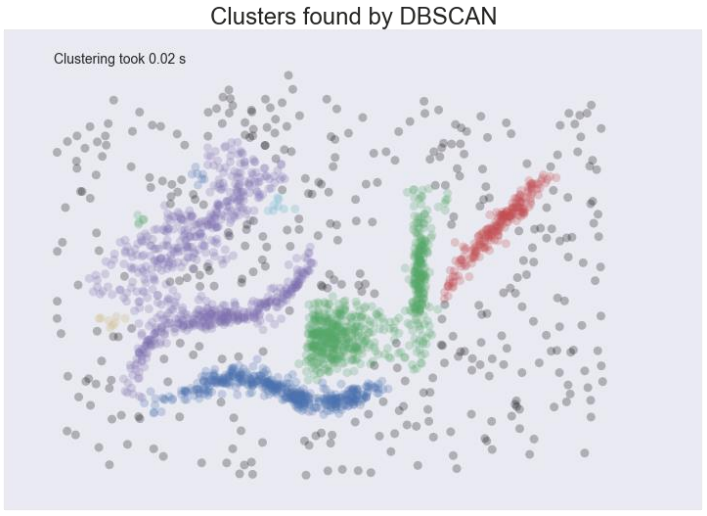


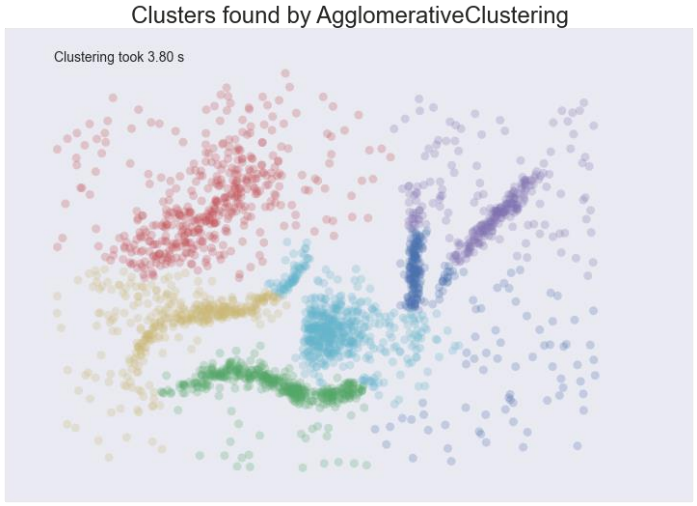
HDBSCAN - Comparison

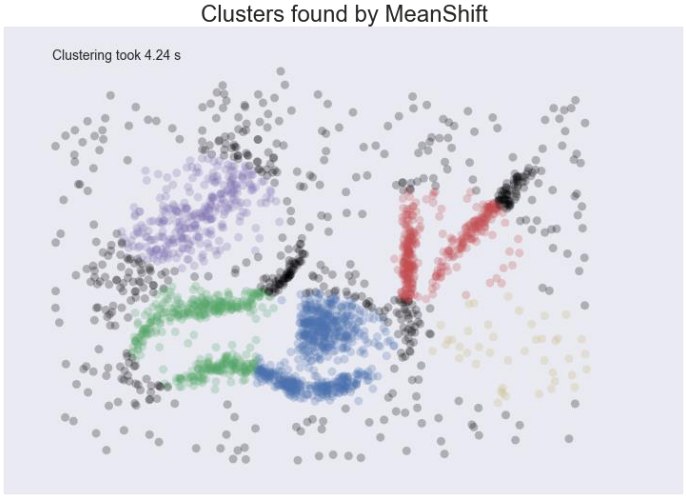




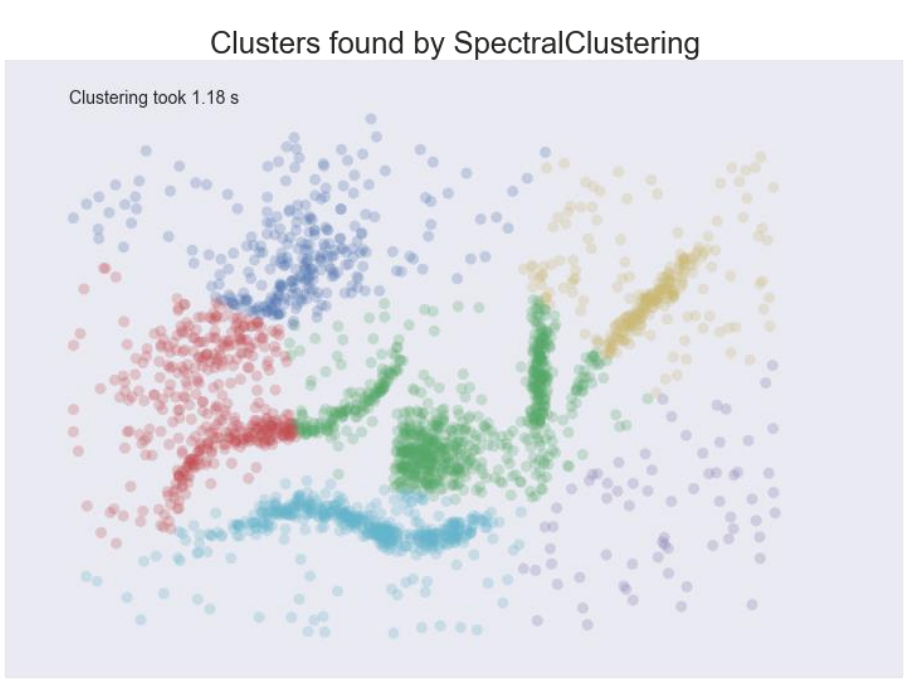


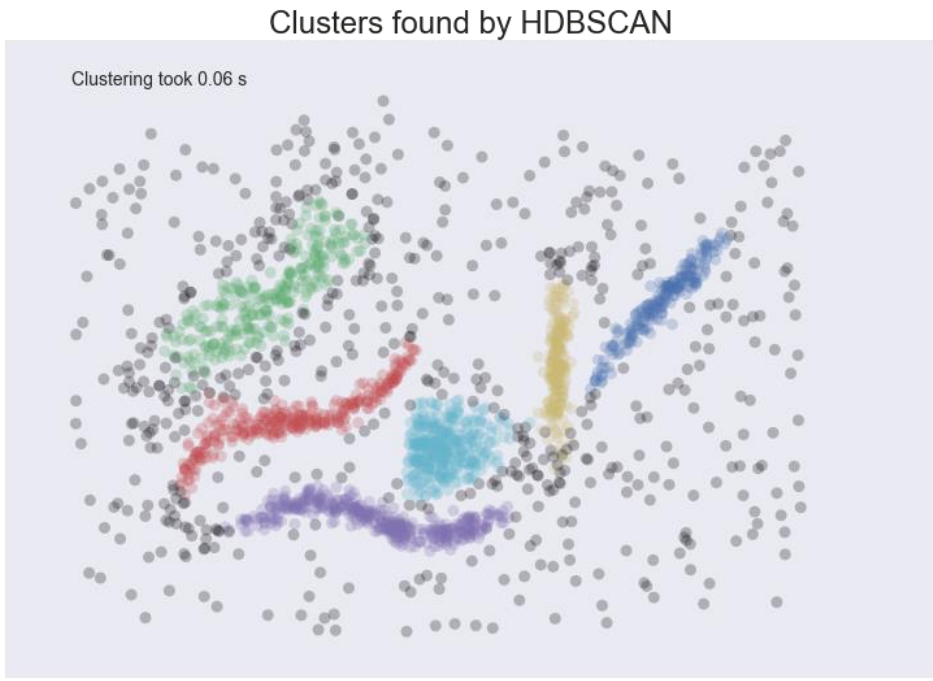






HDBSCAN - Comparision





Clustering using Deep Learning

Deep Clustering

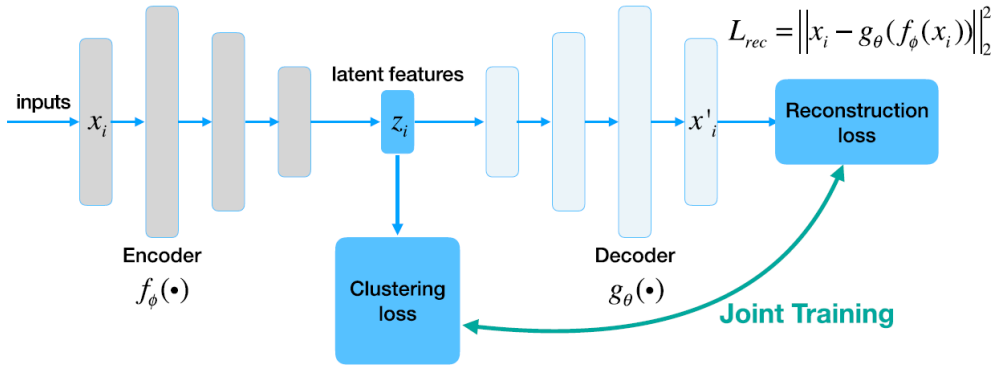


FIGURE 1. Architecture of clustering based on autoencoder. The network is trained by both clustering loss and reconstruction loss.

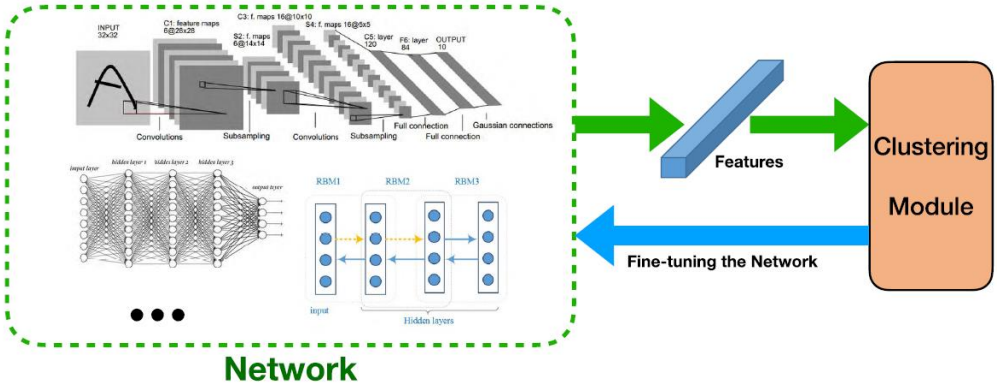


FIGURE 2. Architecture of CDNN-based deep clustering algorithms. The network is only adjusted by the clustering loss. The network architecture can be FCN, CNN, DBN and so on.

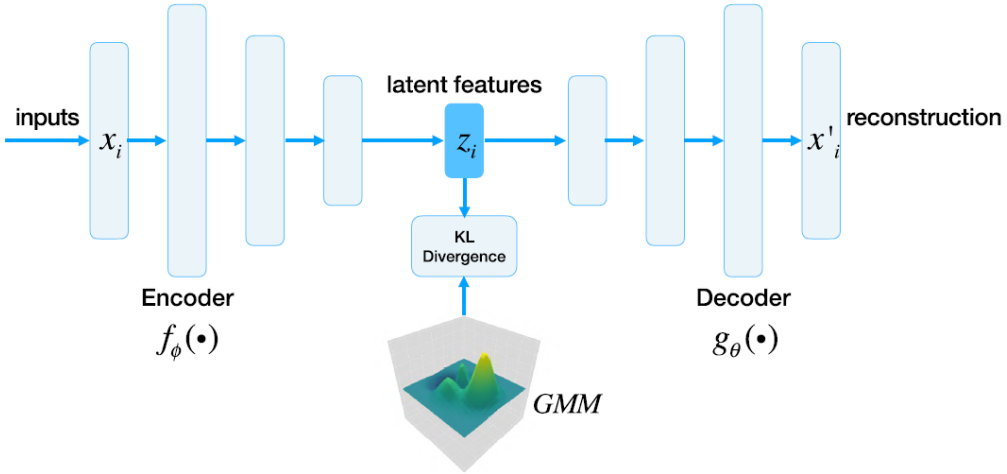


FIGURE 3. Architecture of VAE-based deep clustering algorithms. They impose a GMM priori over the latent code.

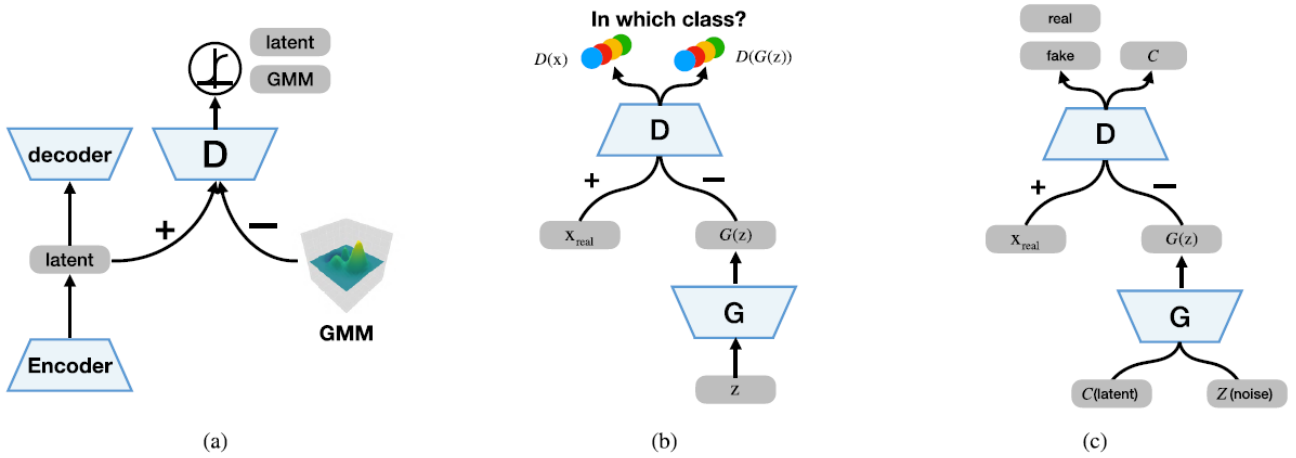


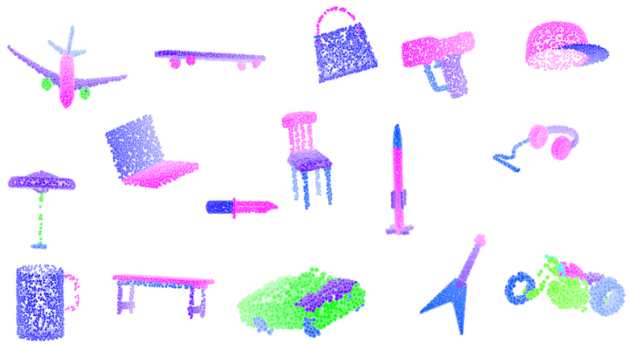
FIGURE 4. GAN-based deep clustering. (a) DAC. (b) CatGAN. (c) InfoGAN.

[A Survey of Clustering With Deep Learning]

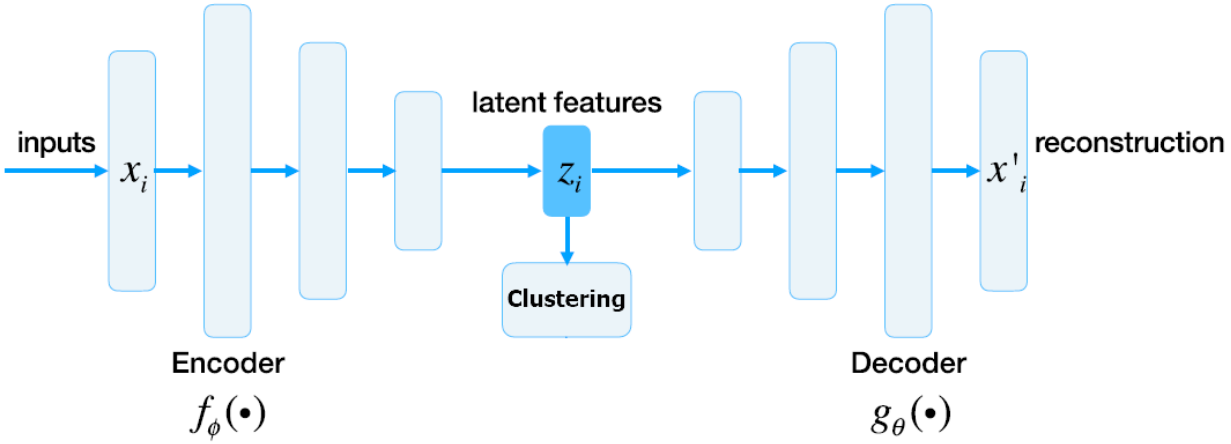
Clustering of 3D Models

Approach

Shape-net dataset



- table
- car
- lamp
- chair
- airplane
- earphone
- guitar
- knife
- skateboard
- laptop
- motorbike
- rocket
- pistol
- mug
- bag
- cap



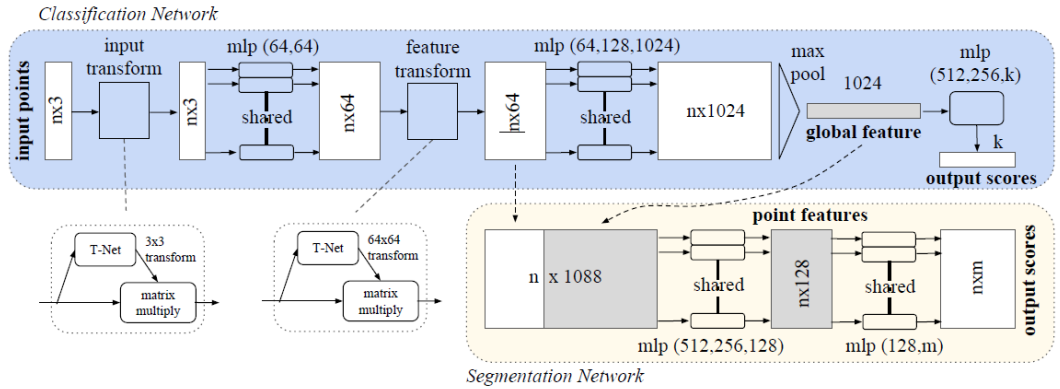
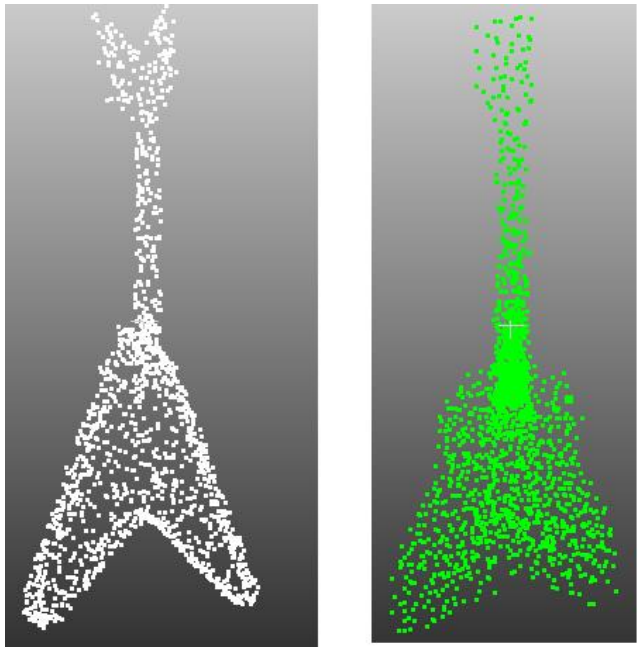
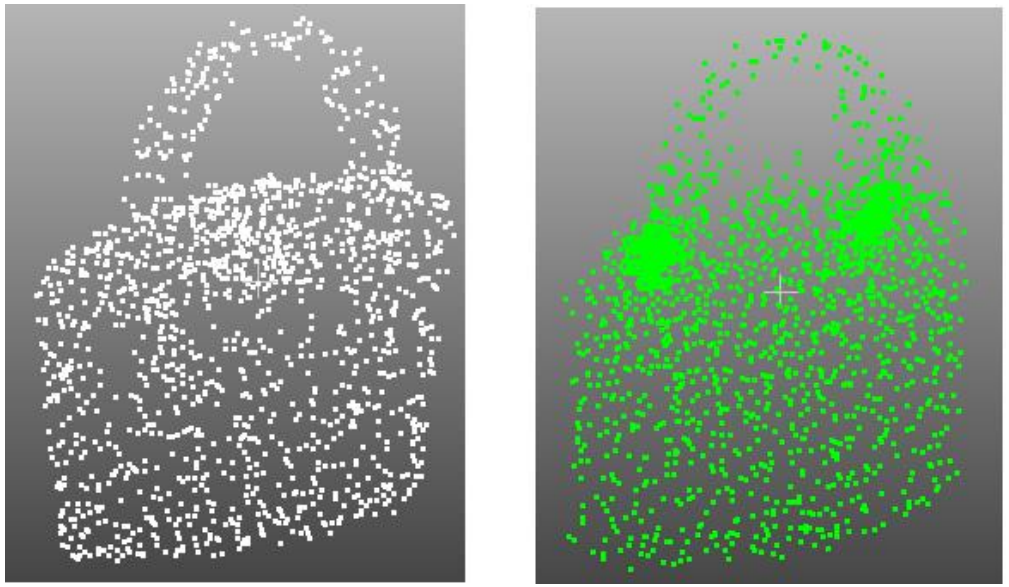
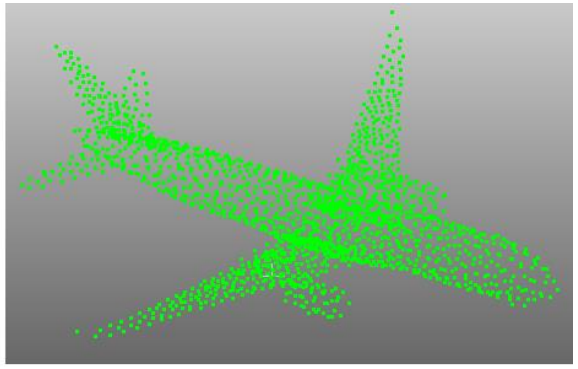


Figure 2. **PointNet Architecture.** The classification network takes n points as input, applies input and feature transformations, and then aggregates point features by max pooling. The output is classification scores for k classes. The segmentation network is an extension to the classification net. It concatenates global and local features and outputs per point scores. "mlp" stands for multi-layer perceptron, numbers in bracket are layer sizes. Batchnorm is used for all layers with ReLU. Dropout layers are used for the last mlp in classification net.

Results of Autoencoder







Applying clustering

Let’s go to notebook…