BTS | Barcelona Technology School

Classical Data Analysis

Master in Big Data Solutions 2020-2021

Filipa Peleja

Víctor Pajuelo

filipa.peleja@bts.tech

victor.pajuelo@bts.tech

BTS | Barcelona Technology School

MBDS 2020-2021

Unsupervised Learning

Other clustering algorithms

Contents

- Other clustering algorithms:
  - Limitations of K-Means
  - Hierarchical clustering
  - Dendograms
  - Linkage functions
  - Limitations of hierarchical clustering
  - DBSCAN (Density-connected points)
  - Clustering algorithms recap

- *Gaussian Mixture Models (GMMs)*

Today's objective

- Get acquainted with more unsupervised techniques
- Learn the ins and outs of DBSCAN and HC

Let's git things done!

Let's see it again

Pull Session 12 notebooks

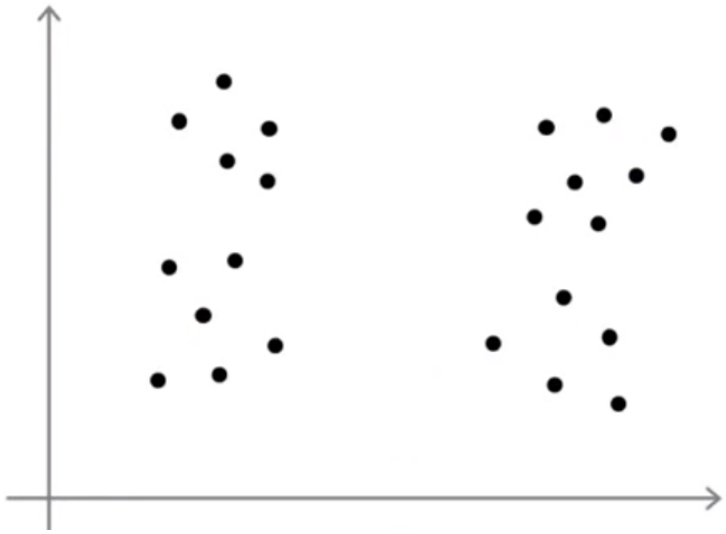
```
$ git clone https://github.com/vfp1/bts-cda-2020.git

# If you have done that already
$ git pull origin master
```

Limitations of K-Means

Limitations of K-Means

Choosing the number of clusters

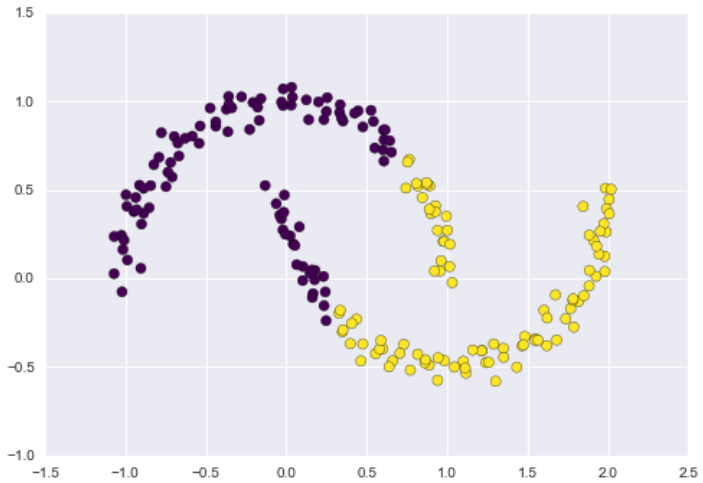
- An important limitation of K-means algorithm is that we need to specify the number of clusters beforehand. So how can we choose the best K for our data?
  - We saw that we could “kind of do it” through `GridSearchCV`, but is not optimal since we always depend on plugging a classifier after in the pipeline
    - That means, we cannot look at the optimal clustering *per se*



Limitations of K-Means

Choosing the number of clusters

- K-means is assuming the following:
  - Each cluster has a "cluster center" which is the average of all points in the cluster.
  - All points within a cluster are closer to that cluster center than to the center of any other cluster.
  - This basically means that the boundaries between clusters will always be linear.



Unsupervised Learning

Main tasks

- Without maybe being too aware of it, we have already performed some unsupervised learning tasks, i.e.:
  - Dimensionality Reduction
- However, there are **more tasks involved**

Task	Objective	Applications
Clustering	Group similar instances together into *clusters*	Data analysis, customer segmentation, recommender systems, search engines, image segmentation, semi-supervised learning, dim reduction, etc.
Anomaly detection	Learn what “normal” data looks like, then use that to detect abnormal instances	Detection of defective items in product lines, new trend in time series, etc.
Density estimation	Estimating the Probability Density Function (PDF) of the random process that generated the dataset.	Anomaly detection, data analysis and visualization

BTS | Barcelona Technology School

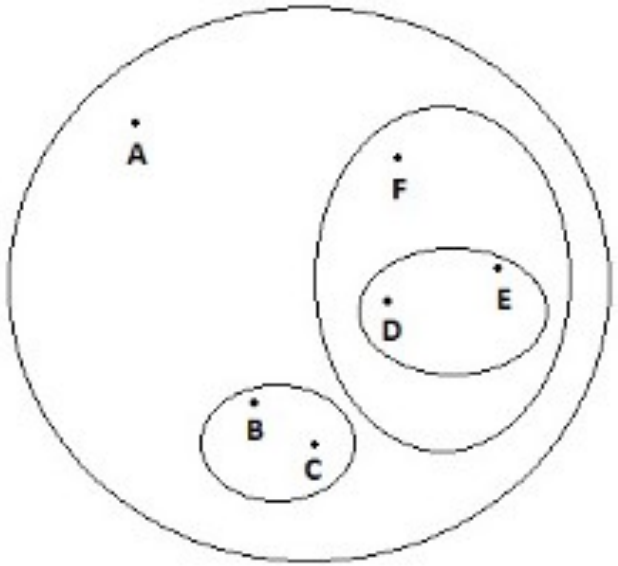
MBDS 2020-2021

Hierarchical Clustering

Hierarchical clustering

Introduction

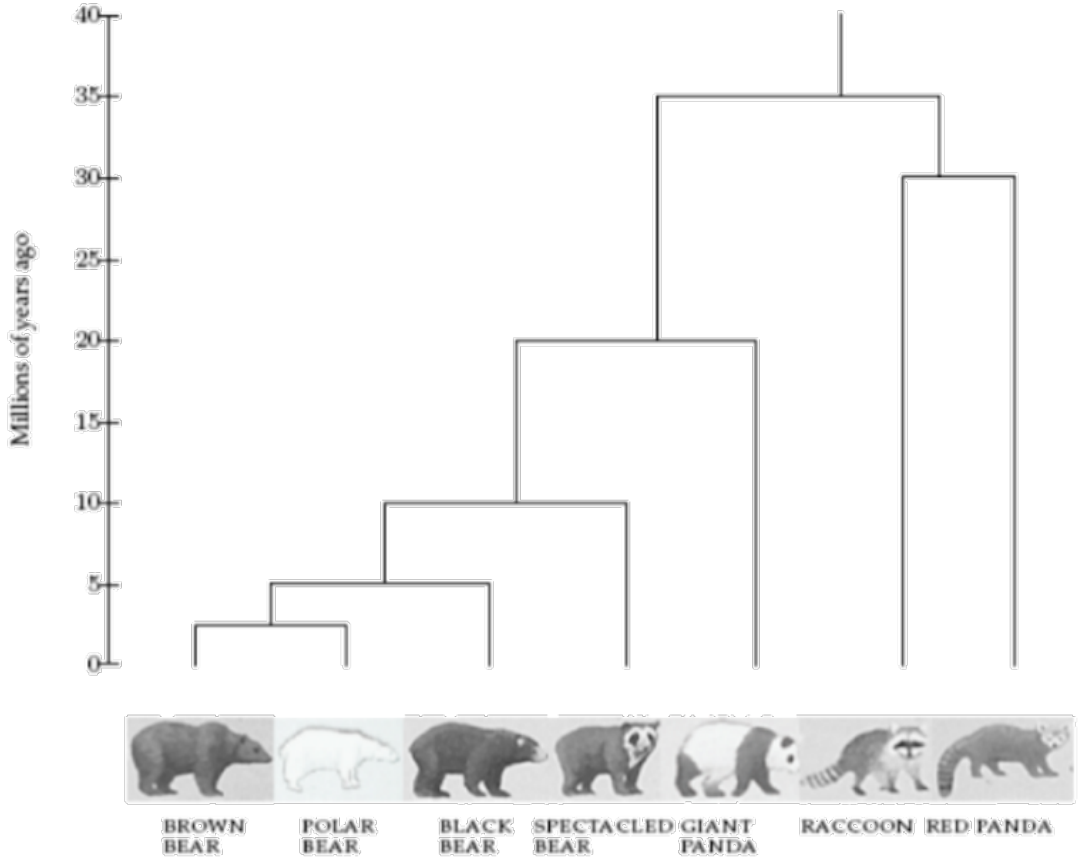
- There are two types of hierarchical clustering (aka agglomerative clustering):
  - **Divisive** (top-down): We start by one single cluster and we split this cluster in smaller clusters recursively. (Not used in practice)
  - **Agglomerative** (bottom-up): Initially each point is a cluster. We merge points into clusters recursively until there is only one cluster. (**Used in practice**)



Hierarchical clustering

Introduction

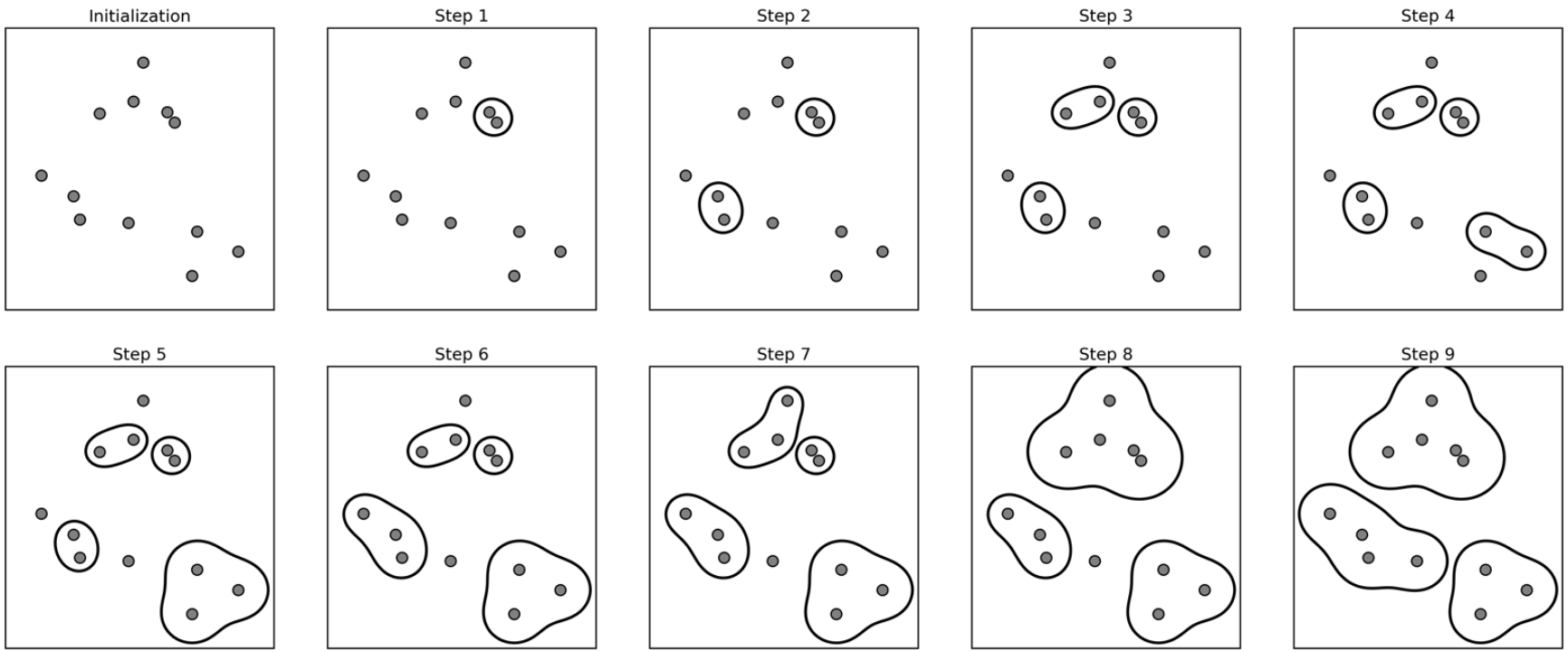
- Hierarchical clustering is particularly suitable to study clusters which follow some kind of evolution tree: such as clustering animal species.



Hierarchical clustering

Intuition

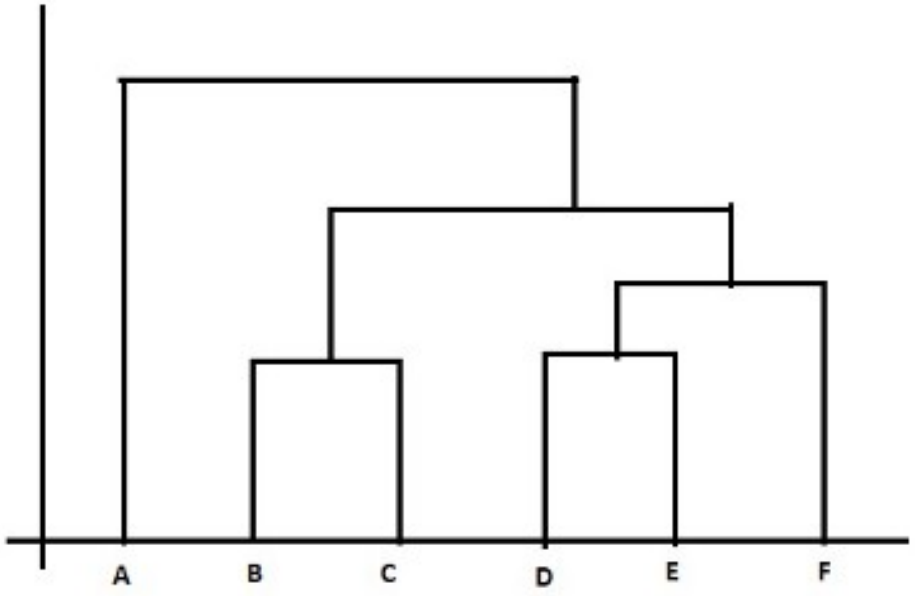
- We start with each cluster as a single point.
- We merge points into clusters recursively until there is only one cluster.

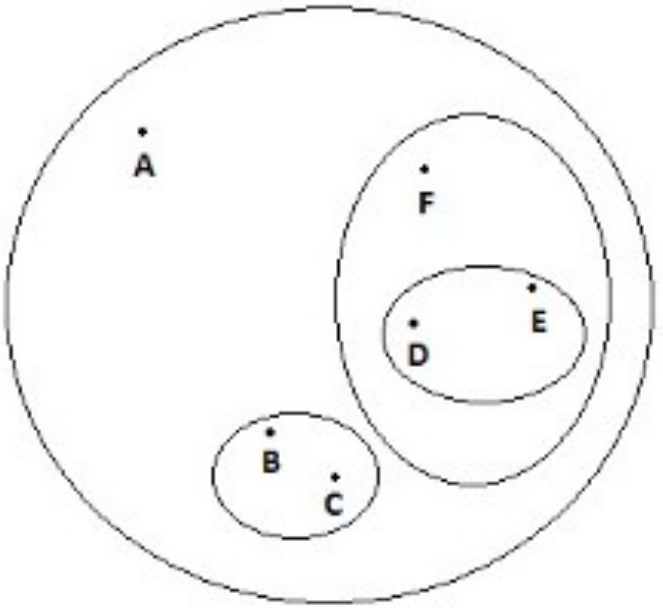


Hierarchical clustering

Intuition

- A dendogram is a representation of the clustering as a tree.

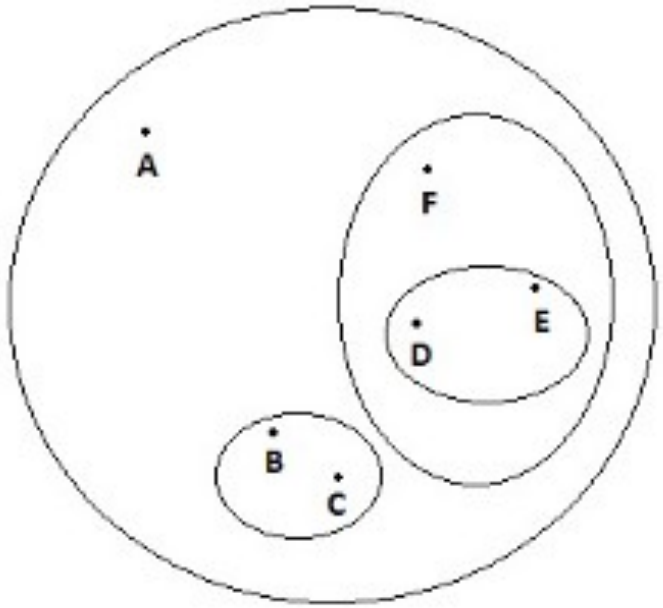




Hierarchical clustering

Distance computation

- How do we decide the “nearness” between clusters?

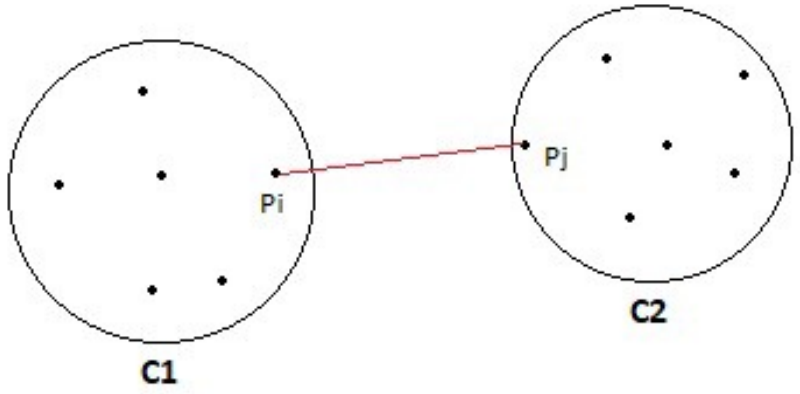


Hierarchical clustering

Linkage criterion → Min(single)

- The linkage criterion determines which distance to use between sets of observations.

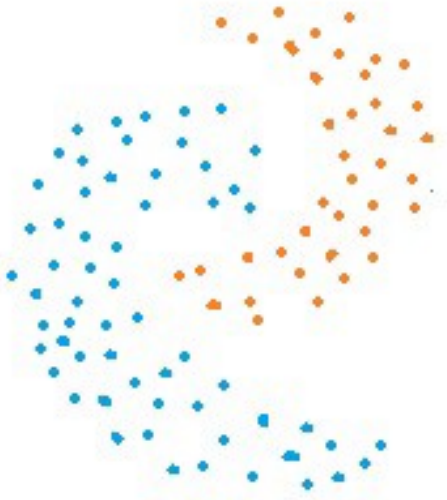
Min (single): minimum of the similarity between points P_i and P_j such that P_i belongs to C_1 and P_j belongs to C_2

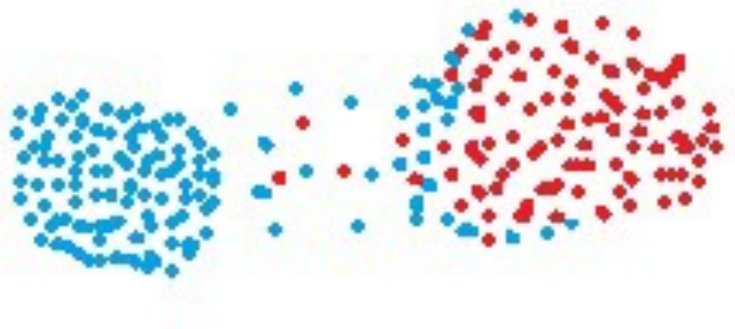


Hierarchical clustering

Linkage criterion → Min(single)

Min (single): minimum of the similarity between points P_i and P_j such that P_i belongs to C_1 and P_j belongs to C_2





Pros: It can separate non-elliptical shapes as long as the separation between them is large.

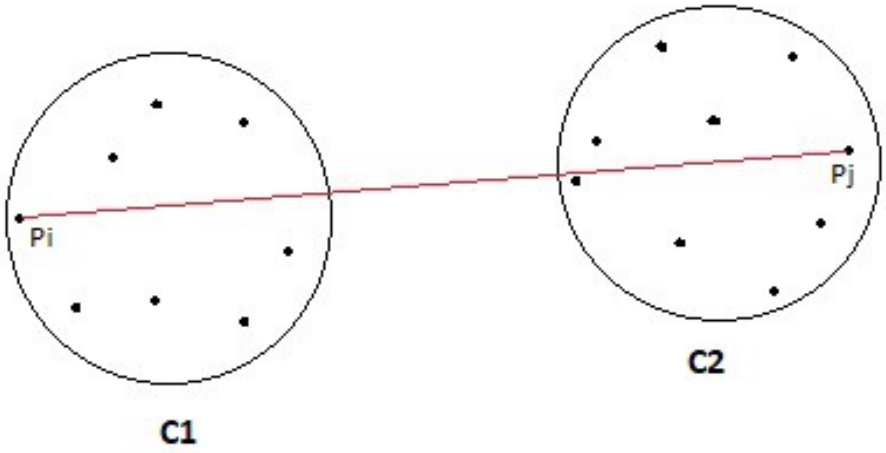
Cons: It is very sensitive to noise. It can not separate clusters properly if there is noise between them.

Hierarchical clustering

Linkage criterion → Max(complete)

- The linkage criterion determines which distance to use between sets of observations.

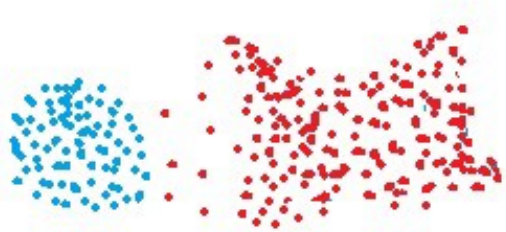
Max (complete): The similarity of two clusters C_1 and C_2 is equal to the maximum of the similarity between points P_i and P_j such that P_i belongs to C_1 and P_j belongs to C_2

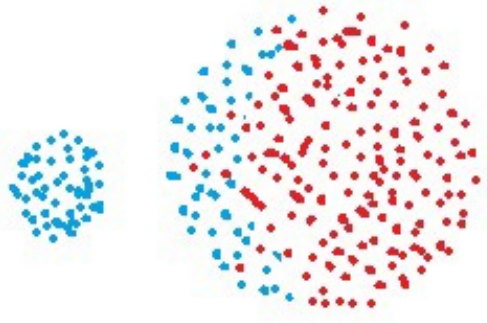


Hierarchical clustering

Linkage criterion → Max(complete)

- **Max (complete)**: The similarity of two clusters C_1 and C_2 is equal to the maximum of the similarity between points P_i and P_j such that P_i belongs to C_1 and P_j belongs to C_2





Pros: This approach does well in separating clusters if there is noise between clusters.

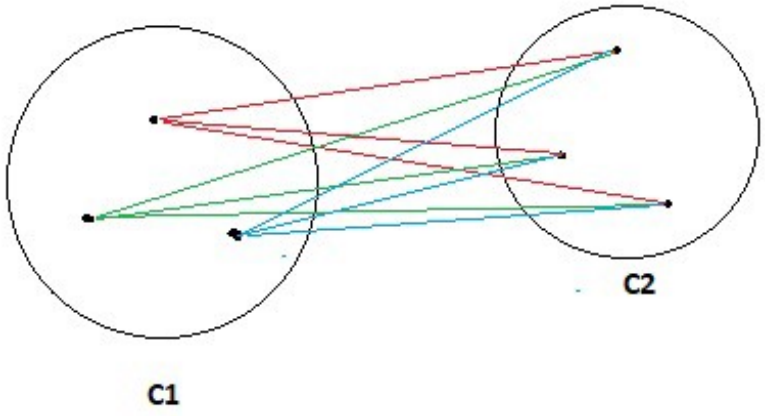
Cons: It is biased towards globular clusters. It tends to break large clusters.

Hierarchical clustering

Linkage criterion → Average & Ward

- The linkage criterion determines which distance to use between sets of observations.

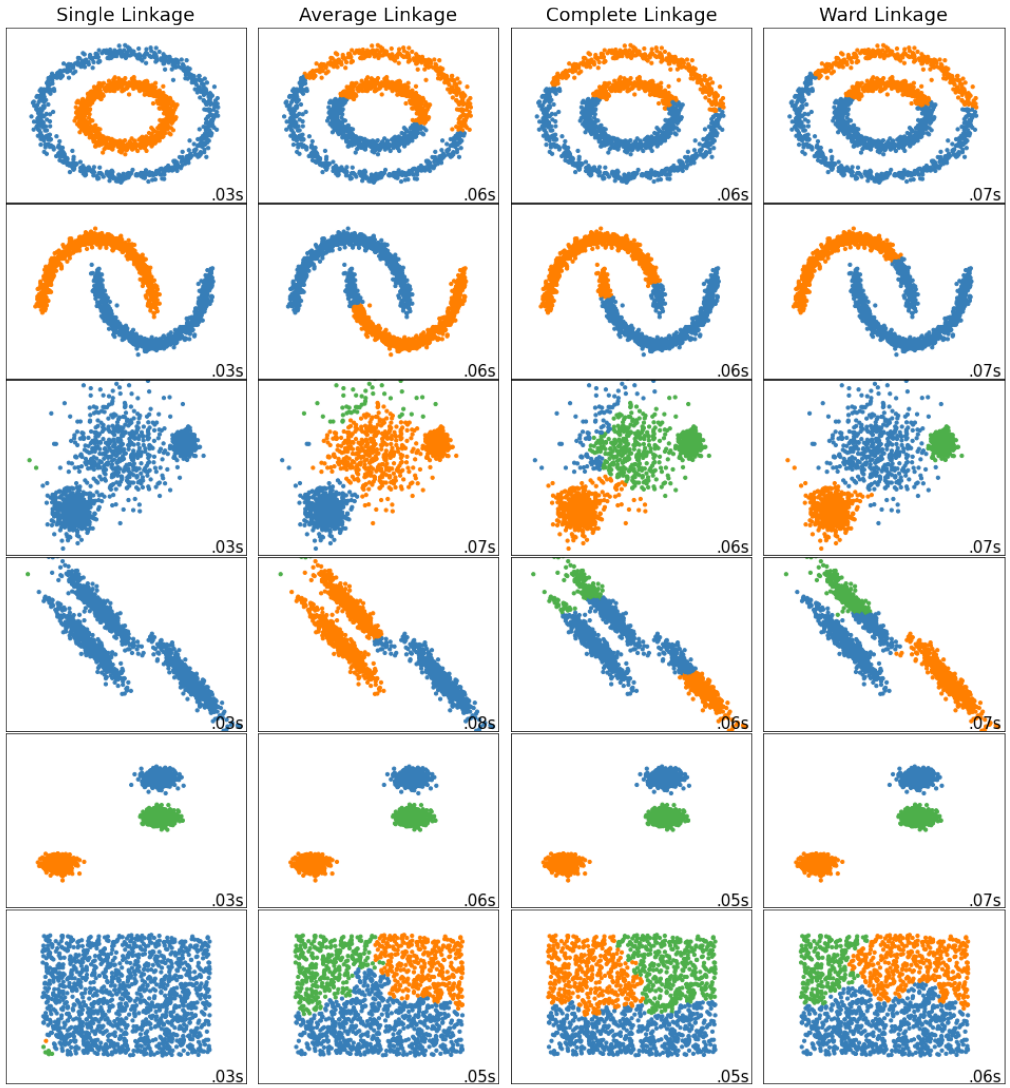
Average: Take all the pairs of points and compute their similarities and calculate the average of the similarities.



Ward: Exactly the same as average, but instead of looking at the similarty between points it computes the euclidean distance squared. (Default in sklearn, most used)

Hierarchical clustering

Linkage criterion comparison

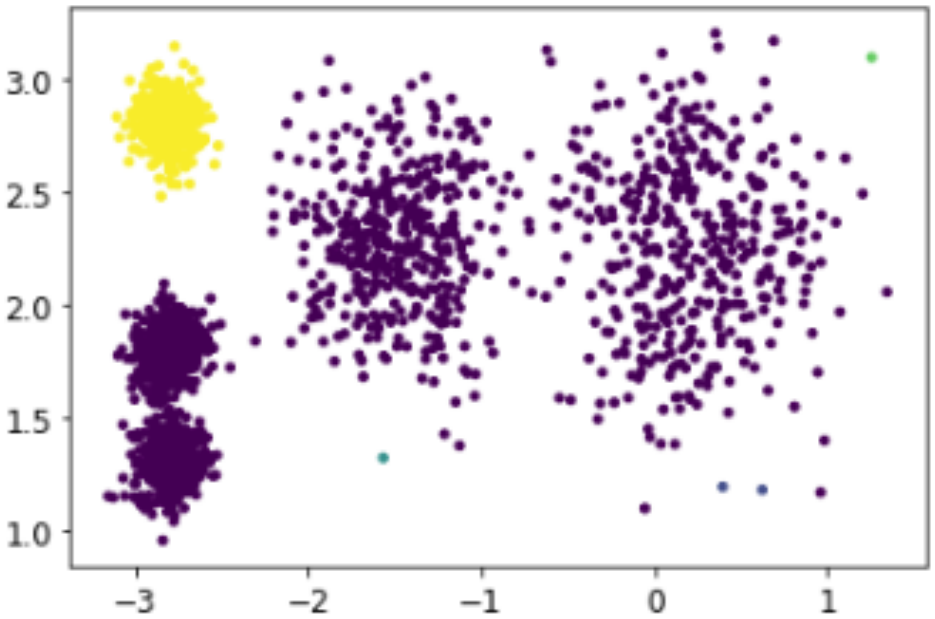


*Code to reproduce the graphs can be seen at **UUID - #S9BC1***

Hierarchical Clustering

Sklearn implementation

```
1 from sklearn.cluster import AgglomerativeClustering
2
3 clustering = AgglomerativeClustering(n_clusters=5, linkage='single').fit(X)
4
5 labels = clustering.labels_
6
7 plt.scatter(X[:, 0], X[:, 1], c=labels, s=10, cmap='viridis');
```

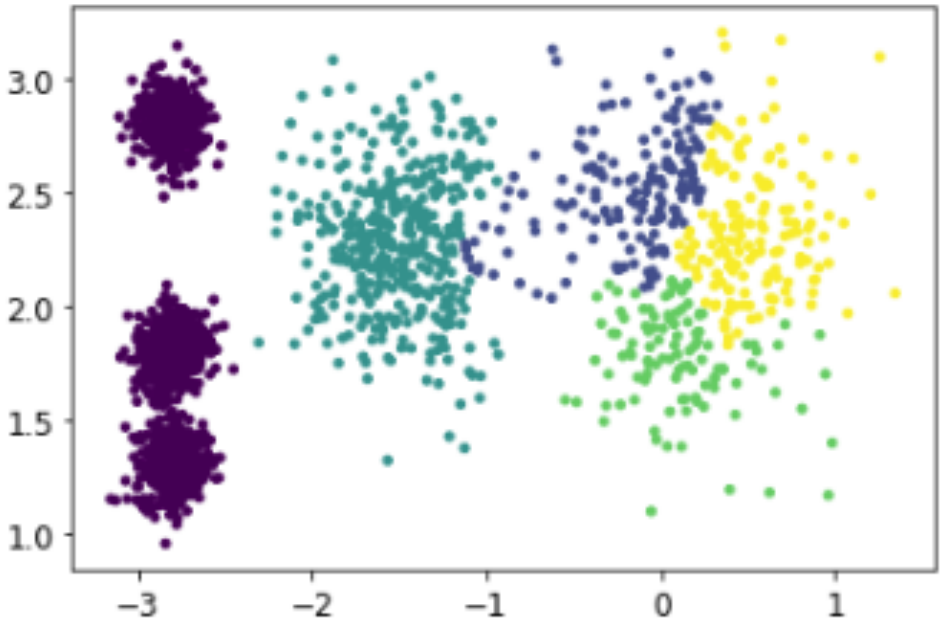


*Code to reproduce the graphs can be seen at **UUID - #S9BC2***

Hierarchical Clustering

Sklearn implementation

```
1 from sklearn.cluster import AgglomerativeClustering
2
3 clustering = AgglomerativeClustering(n_clusters=5, linkage='complete').fit(X)
4
5 labels = clustering.labels_
6
7 plt.scatter(X[:, 0], X[:, 1], c=labels, s=10, cmap='viridis');
```

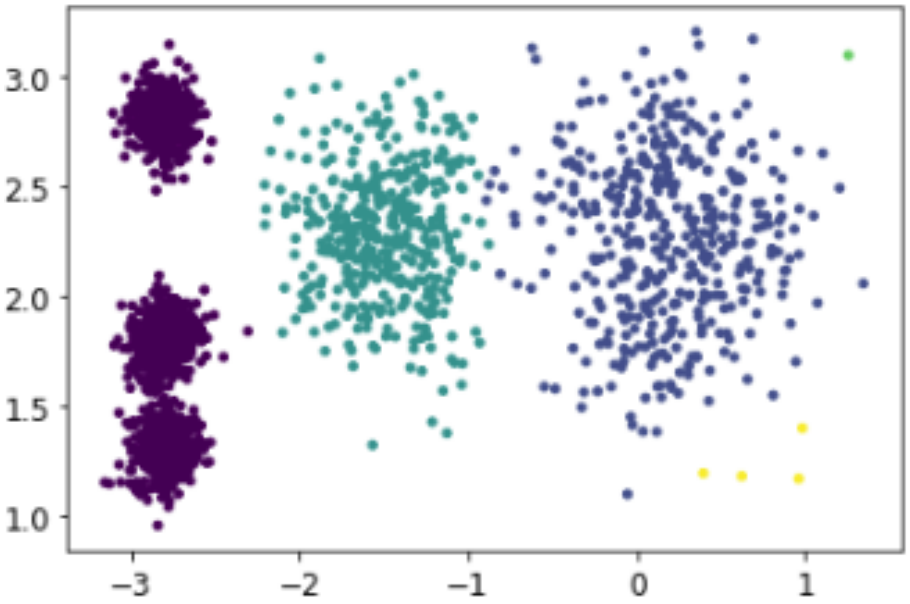


*Code to reproduce the graphs can be seen at **UUID - #S9BC2***

Hierarchical Clustering

Sklearn implementation

```
1 from sklearn.cluster import AgglomerativeClustering
2
3 clustering = AgglomerativeClustering(n_clusters=5, linkage='average').fit(X)
4
5 labels = clustering.labels_
6
7 plt.scatter(X[:, 0], X[:, 1], c=labels, s=10, cmap='viridis');
```

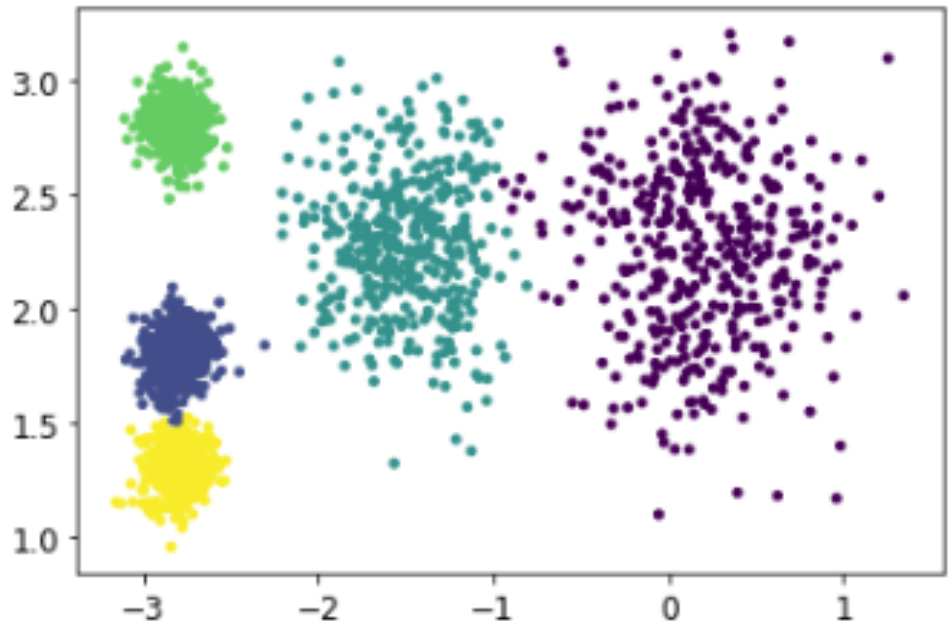


*Code to reproduce the graphs can be seen at **UUID - #S9BC2***

Hierarchical Clustering

Sklearn implementation

```
1 from sklearn.cluster import AgglomerativeClustering
2
3 clustering = AgglomerativeClustering(n_clusters=5, linkage='ward').fit(X)
4
5 labels = clustering.labels_
6
7 plt.scatter(X[:, 0], X[:, 1], c=labels, s=10, cmap='viridis');
```

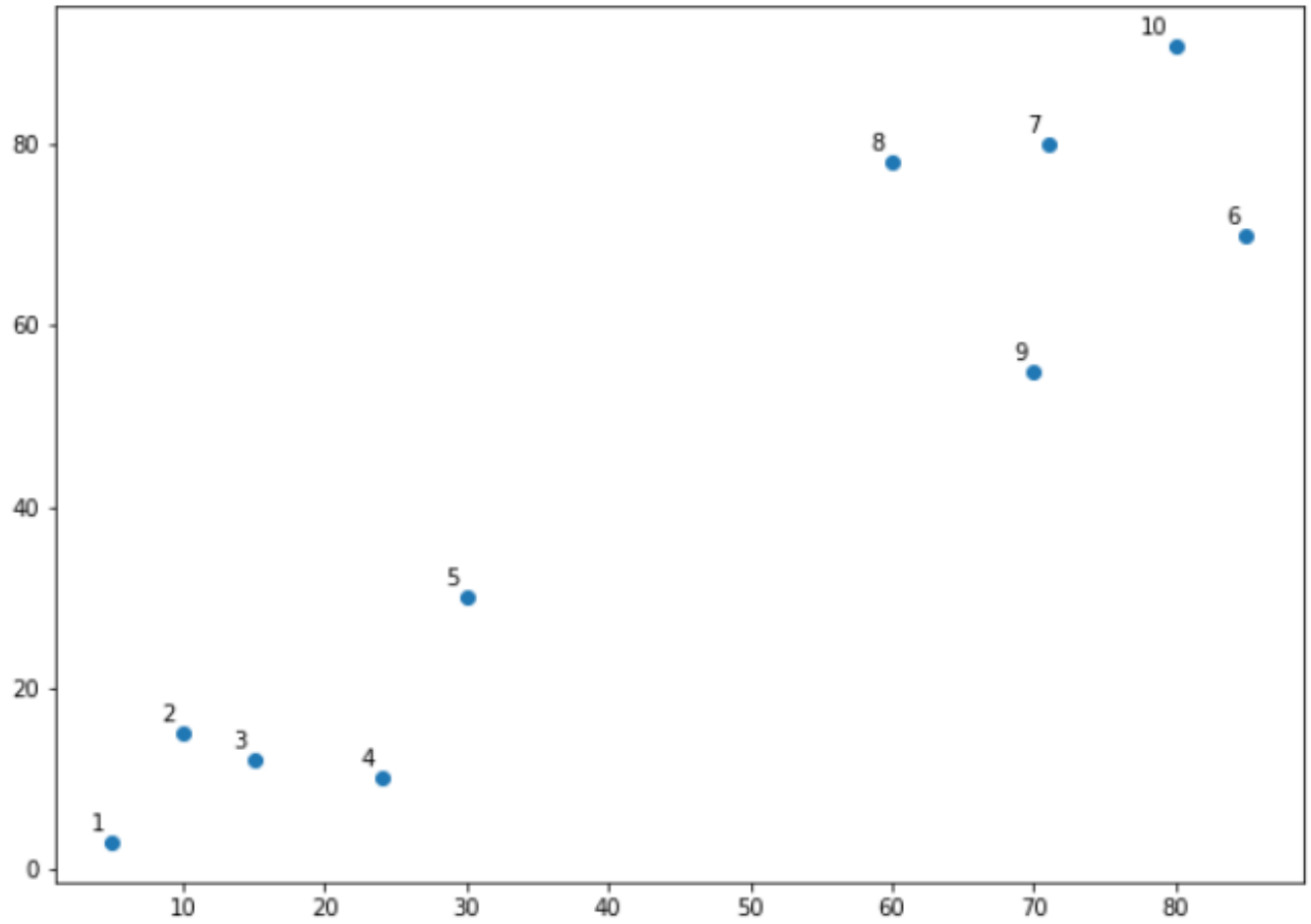


*Code to reproduce the graphs can be seen at **UUID - #S9BC2***

Hierarchical clustering

Choosing the number of clusters

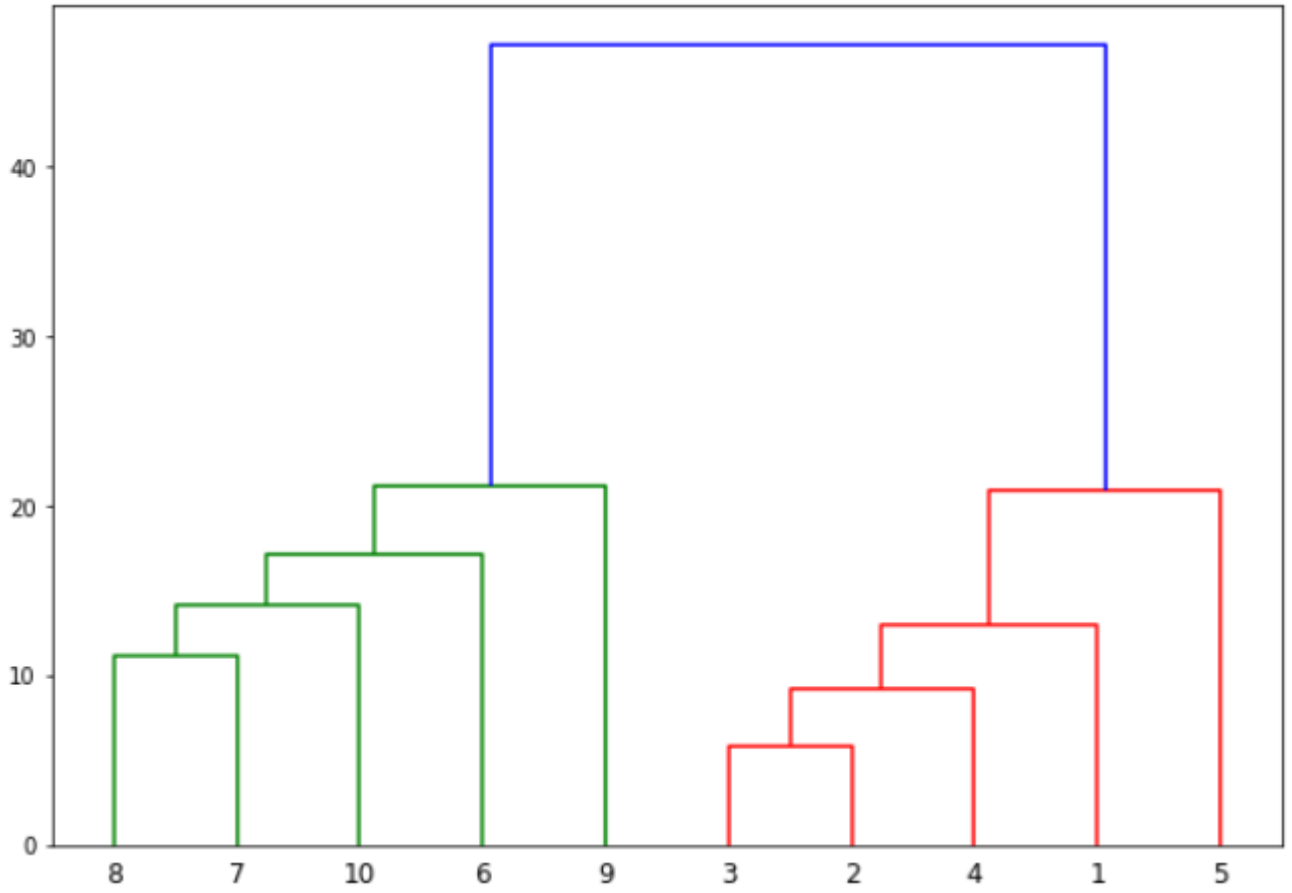
- With hierarchical clustering we still need to define a criterion to choose the number of clusters.



Hierarchical clustering

Choosing the number of clusters

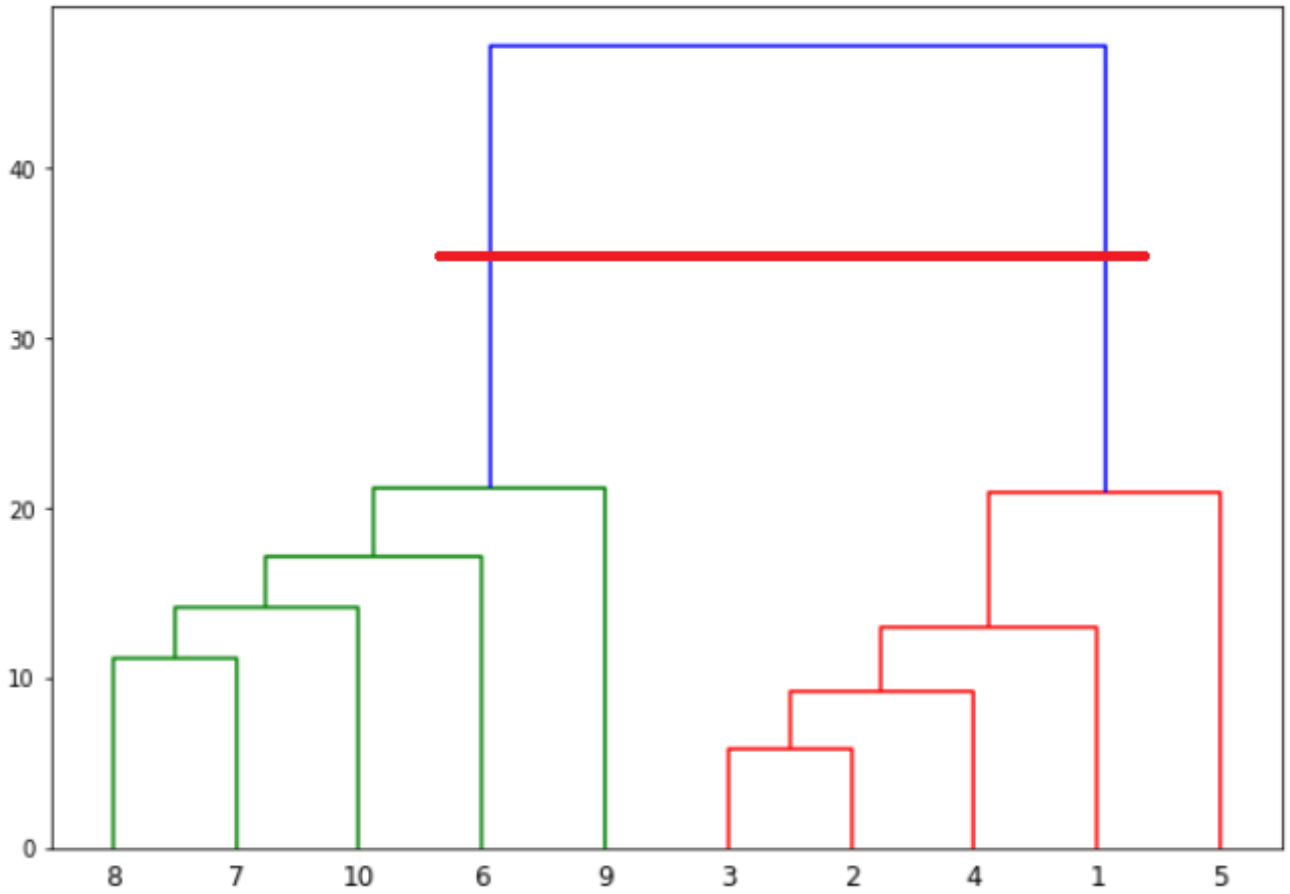
- In general, we decide how many clusters do we keep by looking at the dendogram.



Hierarchical clustering

Choosing the number of clusters

- The number of vertical lines that cross our horizonal line will determine the clusters.



Hierarchical clustering

Dendograms are…

- Useful when we speak about small datasets… but for big ones, it can get tricky to interpret

*See **UUID - #S12BC3***

Hierarchical clustering

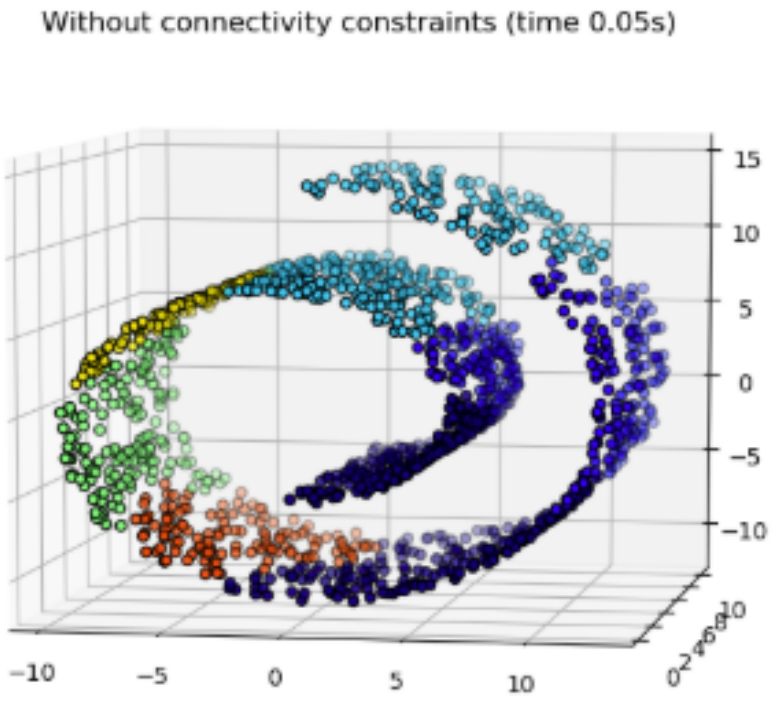
Connectivity constraints

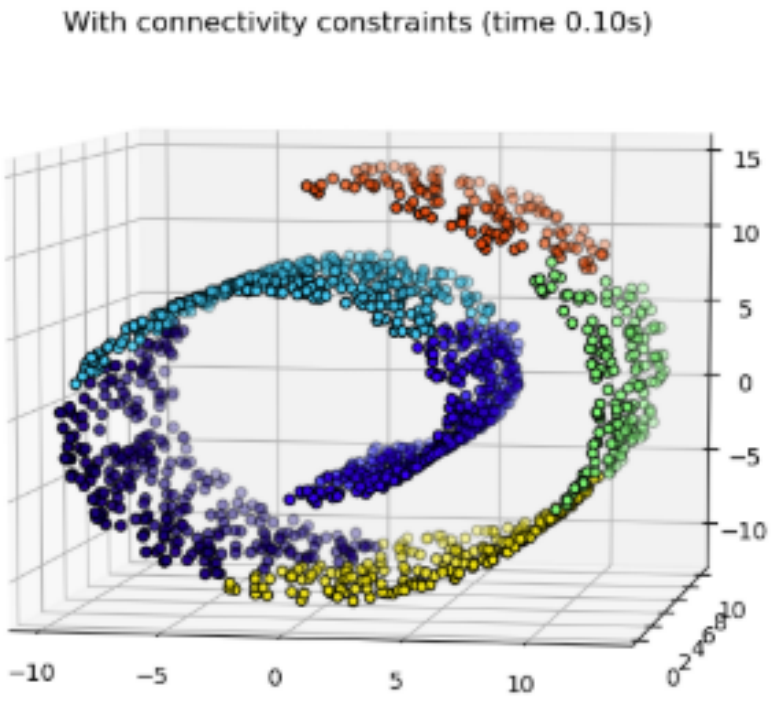
- **Connectivity constraints** can be added to `AgglomerativeClustering` (only adjacent clusters can be merged together), through a **connectivity matrix** that defines for each sample the neighboring samples following a given structure of the data.
- This makes sure that only adjacent clusters can be merged together
- They put constraints to impose certain local structure, but also **make the algorithm go faster**

- **But how do we define the connectivity matrix?**

Hierarchical clustering

Connectivity constraints





*Code to reproduce the graphs can be seen at **UUID - #S12BC4***

Hierarchical clustering

Connectivity constraints

- This connectivity matrix can be learned from the data, for instance using `sklearn.neighbors.kneighbors_graph` to restrict merging to nearest neighbors

- For images we can use `sklearn.feature_extraction.image.grid_to_graph` to enable only merging of neighboring pixels on an image

Hierarchical clustering

Connectivity constraints

Connectivity constraints and single, complete or average linkage can enhance the ‘rich getting richer’ aspect of agglomerative clustering, particularly so if they are built with `sklearn.neighbors.kneighbors_graph`.

In the limit of a small number of clusters, they tend to give a few macroscopically occupied clusters and almost empty ones. Single linkage is the most brittle linkage option with regard to this issue. Also having very low neighbor number.

Hierarchical clustering

Limitations of Hierarchical clustering

- All the approaches to calculate the similarity between clusters has its own disadvantages.
- High space and time complexity for Hierarchical clustering. Hence this clustering algorithm cannot be used when we have huge data.
- Too handcrafted cluster selection

Hierarchical clustering

Benefits of Hierarchical clustering

- “Easy” explainability to clients
- Good tool for when semantical segmentation is needed. For instance, quite useful to use for separating customers, land classes, etc.

BTS | Barcelona Technology School

MBDS 2020-2021

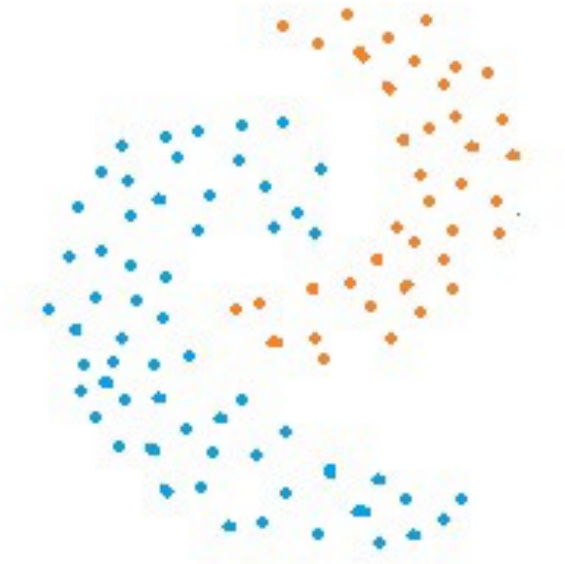
DBSCAN

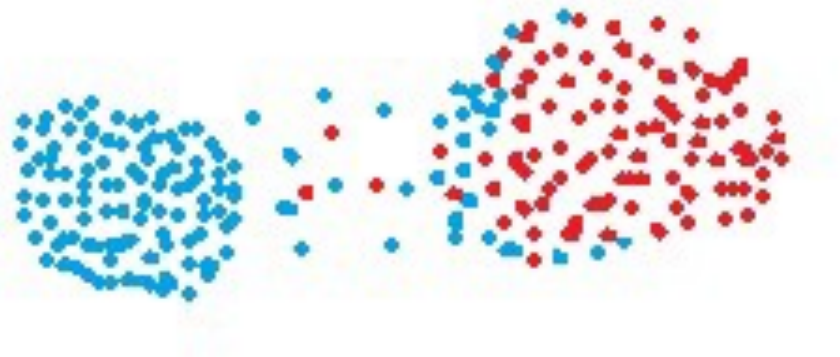
Density Based Spatial Clustering of Applications with Noise

DBSCAN

Density Based Spatial Clustering of Applications with Noise

- DBSCAN is a clustering algorithm that defines clusters as **density-connected** points





Ester, M., H. P. Kriegel, J. Sander, and X. Xu, “A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226-231. 1996

DBSCAN

Parameters

- DBSCAN has two main parameters that we need to take care
  - **Epsilon**: the radius of neighbourhood
  - **MinPts**: minimum number of points to consider a core point

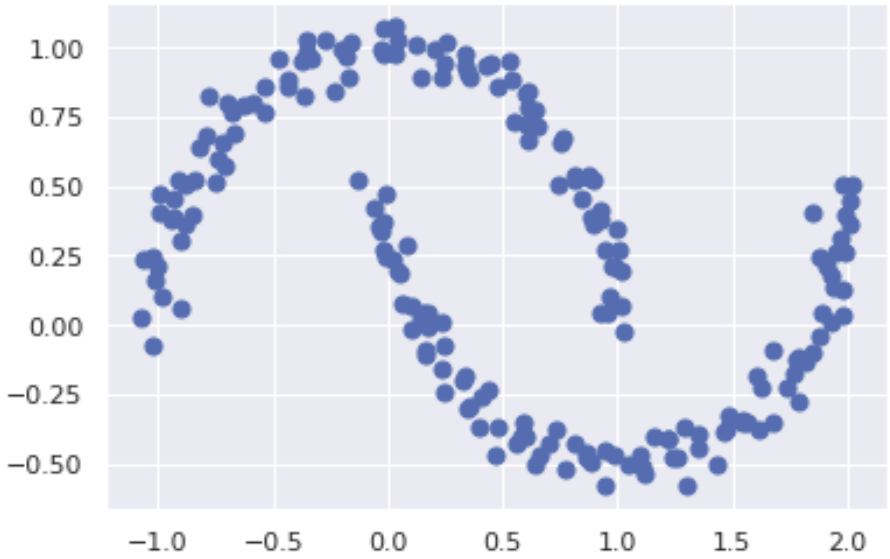
```
1 from sklearn.cluster import DBSCAN
2
3 clustering_db = DBSCAN(eps=0.2, min_samples=5).fit(X)
```

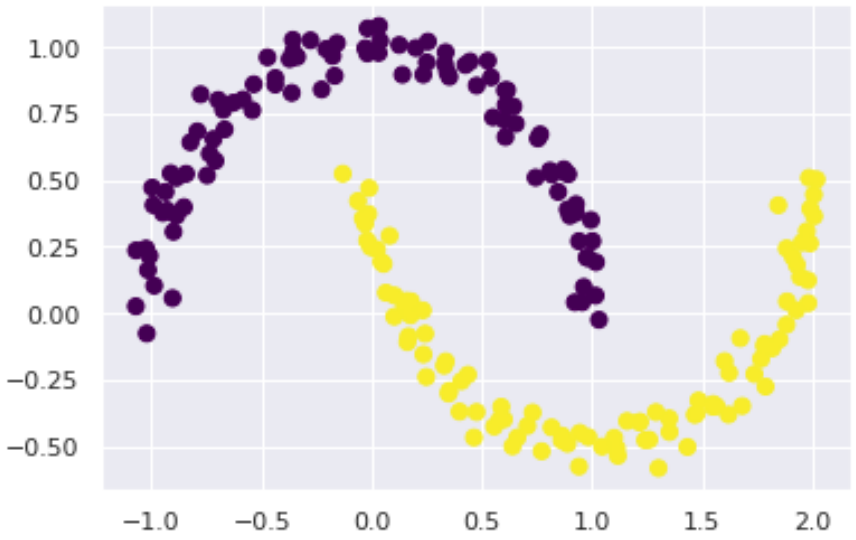
*Code to reproduce the graphs can be seen at **UUID - #S12BC5***

DBSCAN

Parameters

```
1 from sklearn.cluster import DBSCAN
2
3 clustering_db = DBSCAN(eps=0.2, min_samples=5).fit(X)
```



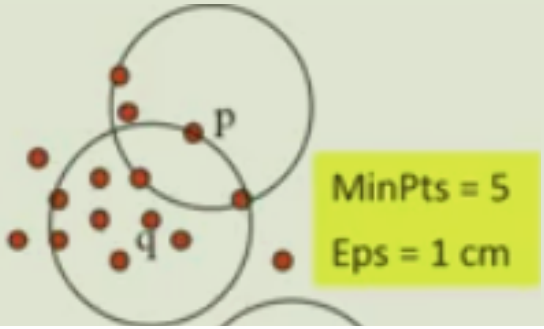



```
1 plt.scatter(X[:, 0], X[:, 1], c=clustering_db.labels_, s=50, cmap='viridis');
```

*Code to reproduce the graphs can be seen at **UUID - #S12BC5***

DBSCAN

- **Neighborhood**
  - `N(q) = {p in X | dist(p,q) < Eps}`
  - The instances belong to a neighborhood, when a point is in the dataset X so that the distance between the point and the instance is lower than the epsilon
- **Core point**: `|N(q)| > MinPts`
  - Any point with more than `MinPts` in its neighborhood



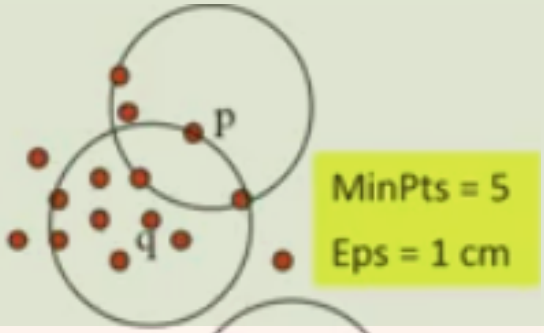
Directly density-reachable: A point `p` is DDR from `q` if:

`p` is in the `N(q)`

`q` is a **core point**.

DBSCAN

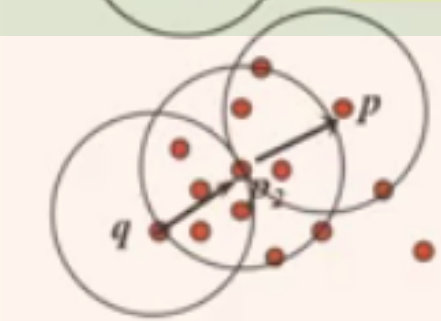
- **Neighborhood**
  - `N(q) = {p in X | dist(p,q) < Eps}`
  - The instances belong to a neighborhood, when a point is in the dataset X so that the distance between the point and the instance is lower than the epsilon
- **Core point**: `|N(q)| > MinPts`
  - Any point with more than `MinPts` in its neighborhood



Directly density-reachable: A point `p` is DDR from `q` if:

`p` is in the `N(q)`

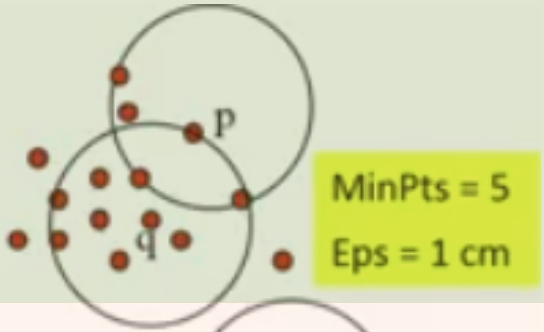
`q` is a **core point**.



Density-reachable: A point `p` is **DR** from q if there exists `p1, p2, …, pn (p1=p and pn=q)` such that p_i+1 is DDR from p_i.

DBSCAN

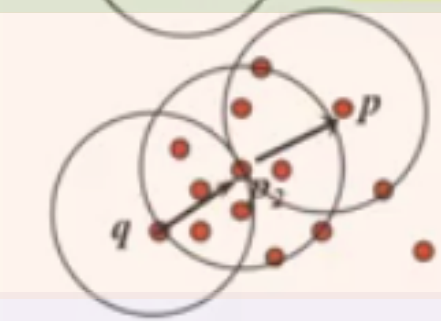
- **Neighborhood**
  - `N(q) = {p in X | dist(p,q) < Eps}`
  - The instances belong to a neighborhood, when a point is in the dataset X so that the distance between the point and the instance is lower than the epsilon
- **Core point**: `|N(q)| > MinPts`
  - Any point with more than `MinPts` in its neighborhood



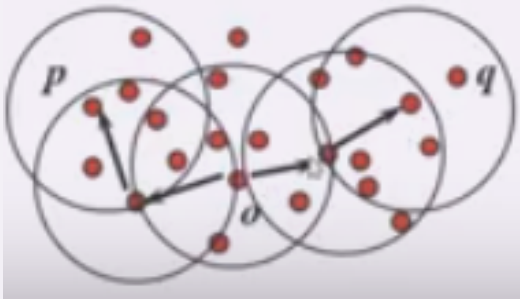
Directly density-reachable: A point `p` is DDR from `q` if:

`p` is in the `N(q)`

`q` is a **core point**.



Density-reachable: A point `p` is **DR** from q if there exists `p1, p2, …, pn (p1=p and pn=q)` such that p_i+1 is DDR from p_i.

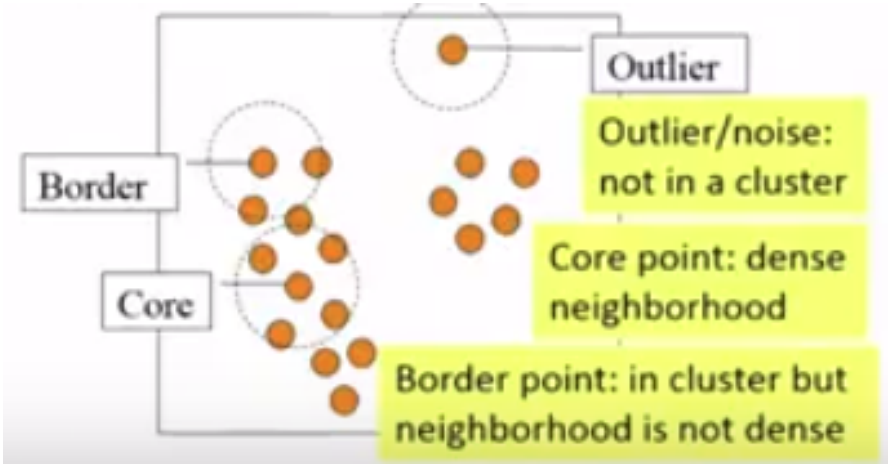


Density-connected: A point `p` is density-connected to `q` if there exists a point `o` such that both `p` and `q` are DR from `o`.

DBSCAN

Algorithm

1. Randomly select a point.
2. Retrieve all density-reachable points.
3. If p is core a cluster is defined.
    1. If p is not core then no points are DR and we take another point.
4. Repeat this until all points have been visited.



DBSCAN

Limitations

DBSCAN is extremely sensitive to parameter choice

DBSCAN: Sensitive to Parameters

Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.

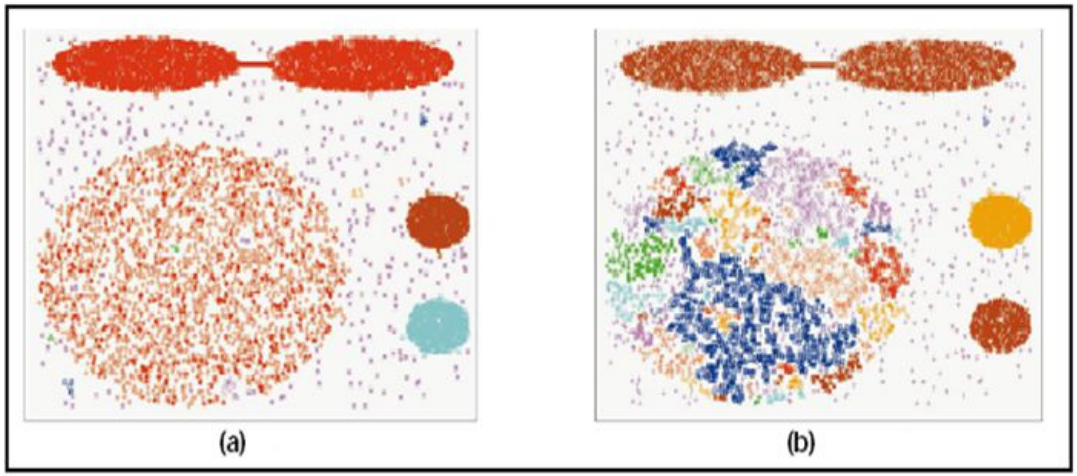
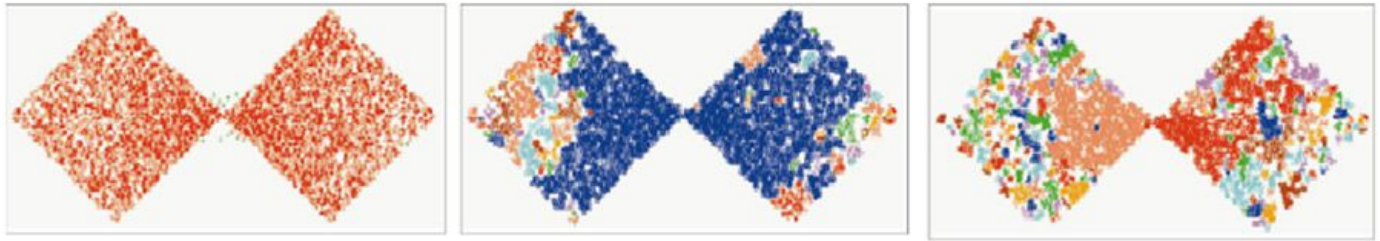


Figure 9. DBScan results for DS2 with MinPts at 4 and Eps at (a) 5.0, (b) 3.5, and (c) 3.0.



DBSCAN

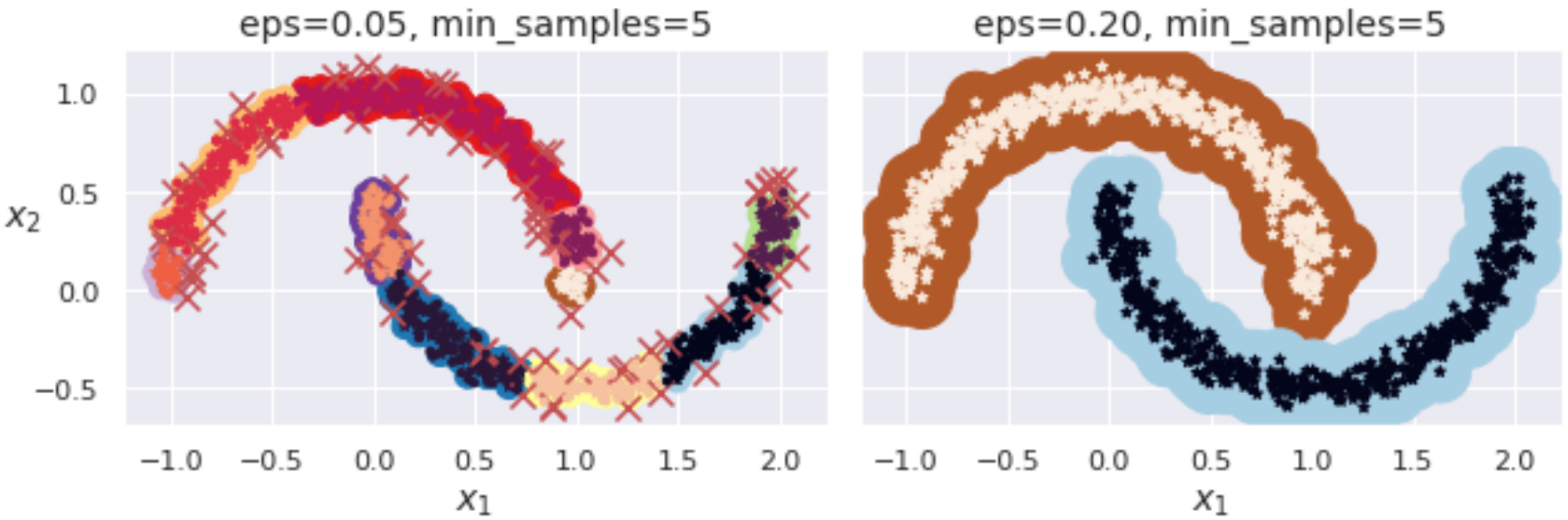
Limitations

DBSCAN is extremely sensitive to parameter choice

```
1 from sklearn.cluster import DBSCAN
2
3 clustering_db = DBSCAN(eps=0.05, min_samples=5).fit(X)
```

```
1 dbscan2 = DBSCAN(eps=0.2)
2 dbscan2.fit(X)
```

```
DBSCAN(algorithm='auto', eps=0.2, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=5, n_jobs=None, p=None)
```

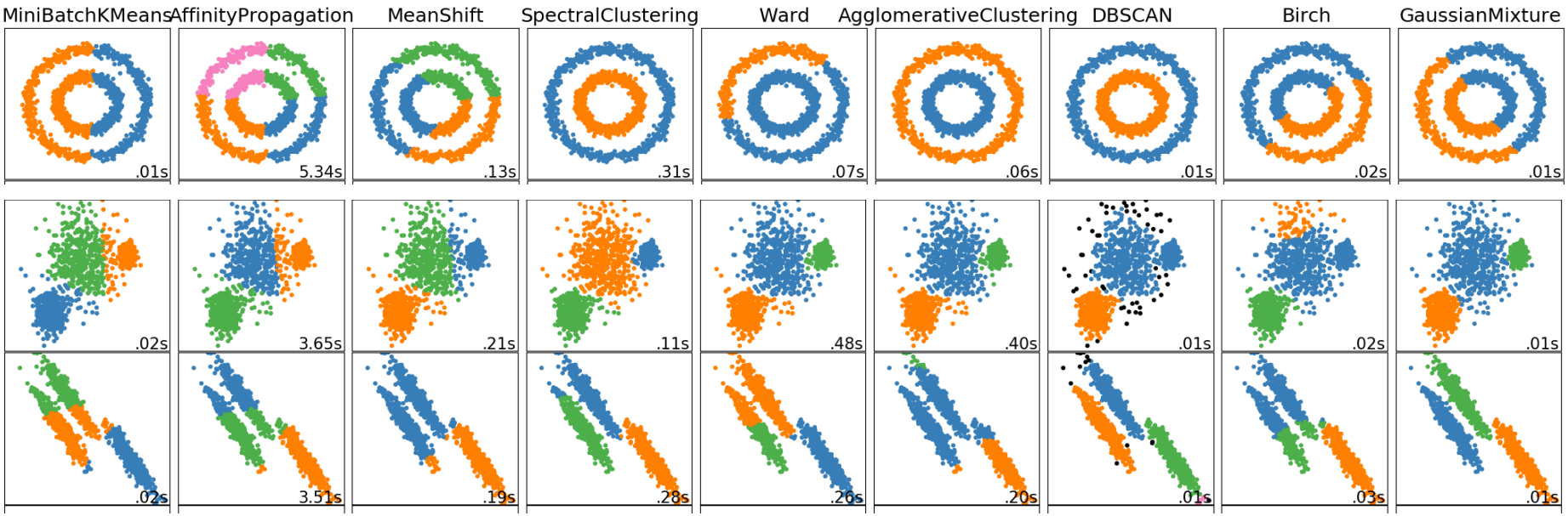


*Code to reproduce the graphs can be seen at **UUID - #S12BC6***

DBSCAN

Limitations

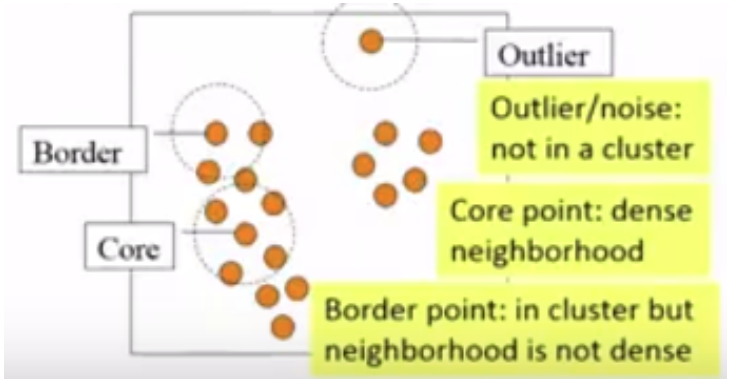
It does not work well with clusters having different densities



DBSCAN

Summary

- It considers outliers.
- It can separate clusters which have non-linear boundaries.
- It also has limitations (different density, sensitivity to parameters)
- There is no magic clustering algorithm!!



DBSCAN

Summary

- Still, DBSCAN is very powerful, capable of identifying clusters of any shape
  - It is robust to outliers and has just two hyperparameters

- If the density varies significantly across the clusters, however, it can be impossible for it to capture all the clusters properly.

- Its computational complexity is roughly O(m log m), making it pretty close to linear with regard to the number of instances, but Scikit-Learn's implementation can require up to O(m^2) memory if eps is large.

DBSCAN

Summary

For faster results when `eps` is large, you could use Sklearn's `OPTICS` implementation, which is similar to DBSCAN. We will see that in next session.

BTS | Barcelona Technology School

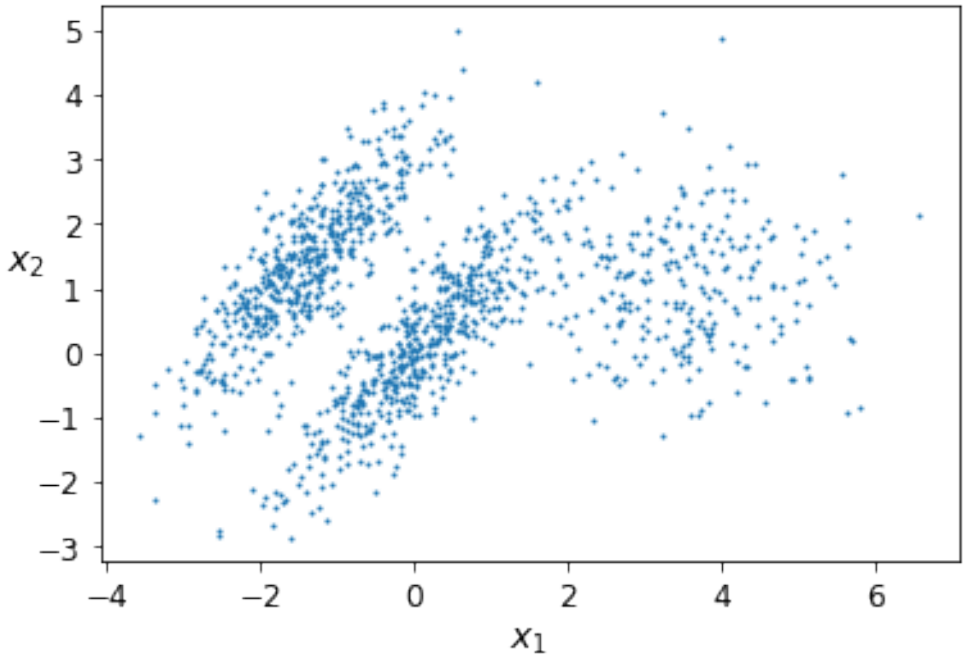
MBDS 2020-2021

GMMs

Gaussian Mixture Models

Generative models

- **GMMs** are a probabilistic model that assumes that the instances were generated from a mixture of several Gaussian distributions whose parameters are unknown.
  - All the instances generated from a single Gaussian distribution form a cluster that typically looks like an ellipsoid.

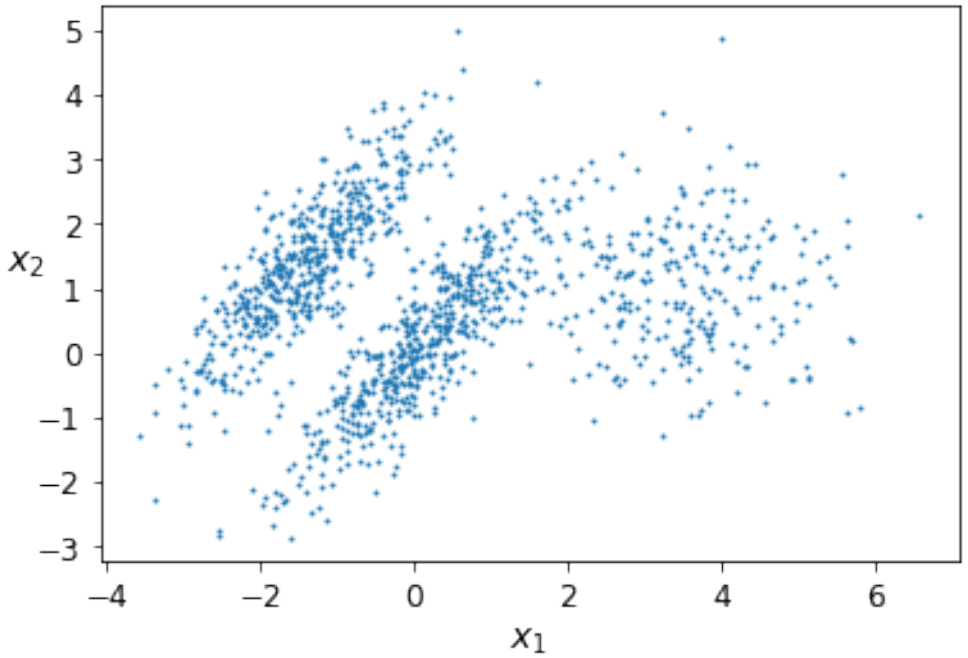


Code to reproduce the graphs can be seen at ***UUID - #S12CC1***

Gaussian Mixture Models

Generative models

- Each cluster can have a different ellipsoidal shape, size, density, and orientation.
  - We know it was generated from one of the Gaussian distributions, but you are not told which one, and you do not know what the parameters of these distributions are.



*Code to reproduce the graphs can be seen at **UUID - #S12CC1***

Gaussian Mixture Models

Sklearn implementation

- There are several GMM variants, but the one implemented in Sklearn is the `GaussianMixture`
  - In which you need to know in advance the number of *k* Gaussian distributions

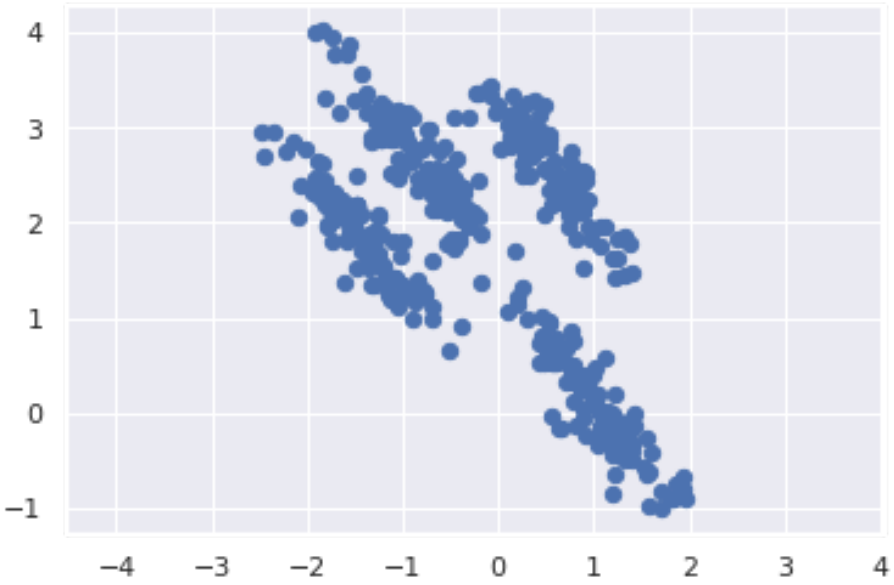
- Given a dataset **X** it is assumed to have been generated using a probabilistic process

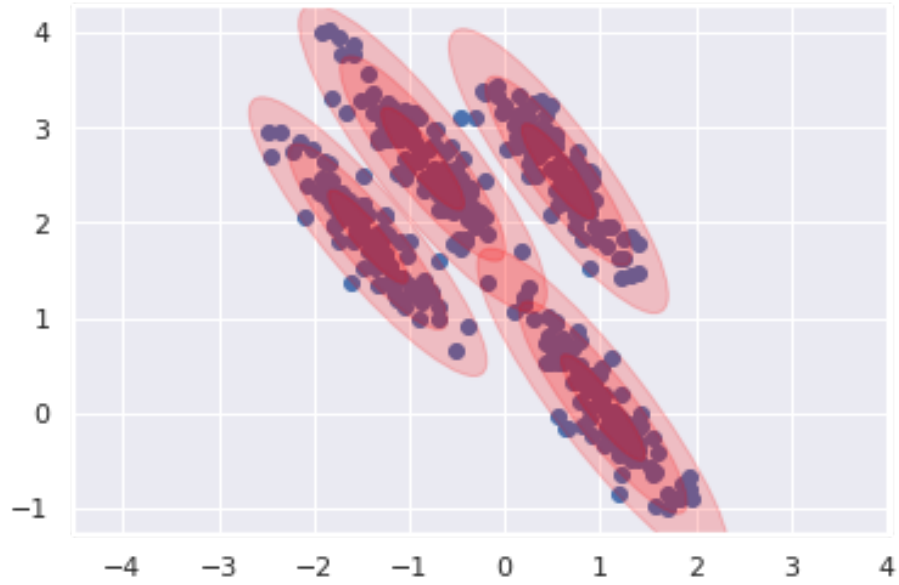
- The task of generative models is to “model” the distributions that they are given so that they can reproduce it

Gaussian Mixture Models

What is density estimation

- Estimate the underlying distribution of points in our dataset

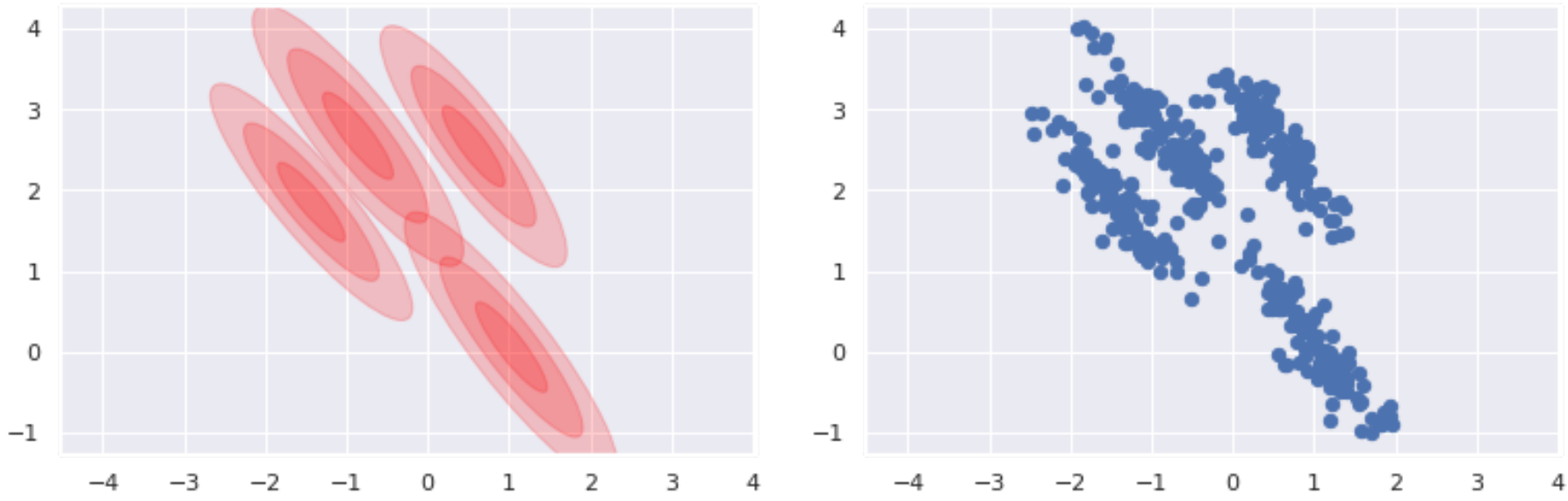




Gaussian Mixture Models

What can generative models do?

- Create new points that follow the same distribution as the points in our dataset



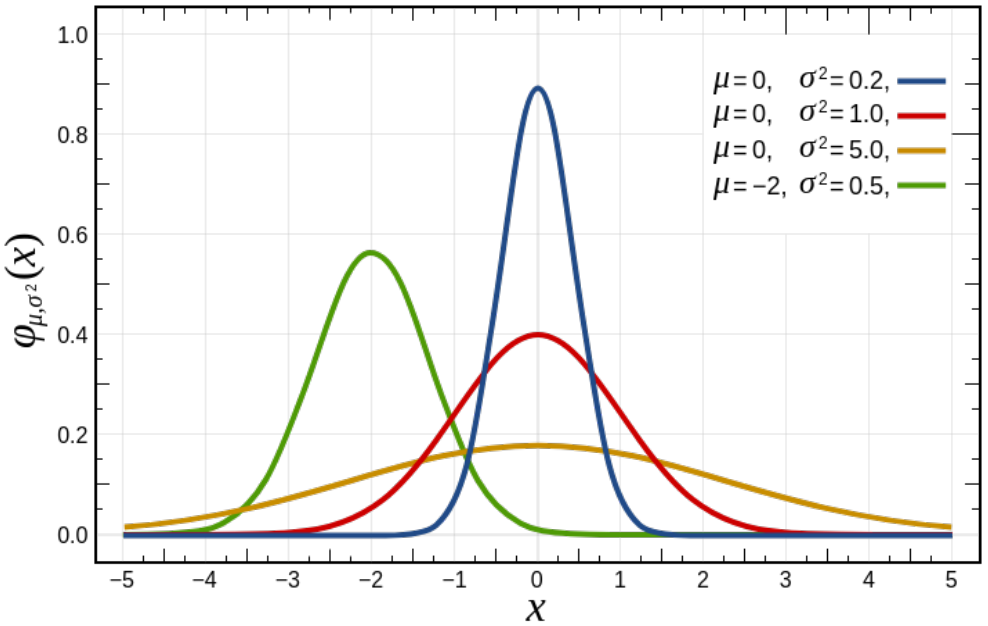
Gaussian Mixture Models

Normal Distribution

The normal distribution has two parameters

Mean = position of the center (mu)
Standard deviation = measure of dispersion of the points.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$



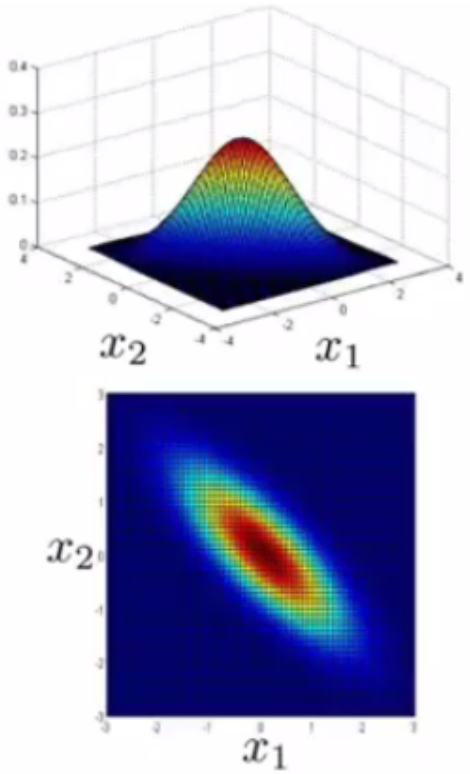
Gaussian Mixture Models

Multivariate Gaussian Distribution

Multivariate gaussian distribution models p(x) for all features at the same time

Mean = position of the center (mu) but it is a vector with the same dimensions as the data points.

Covariance matrix = measure of dispersion of the points but in a multi-dimensional space. (Usually denoted with letter Sigma)

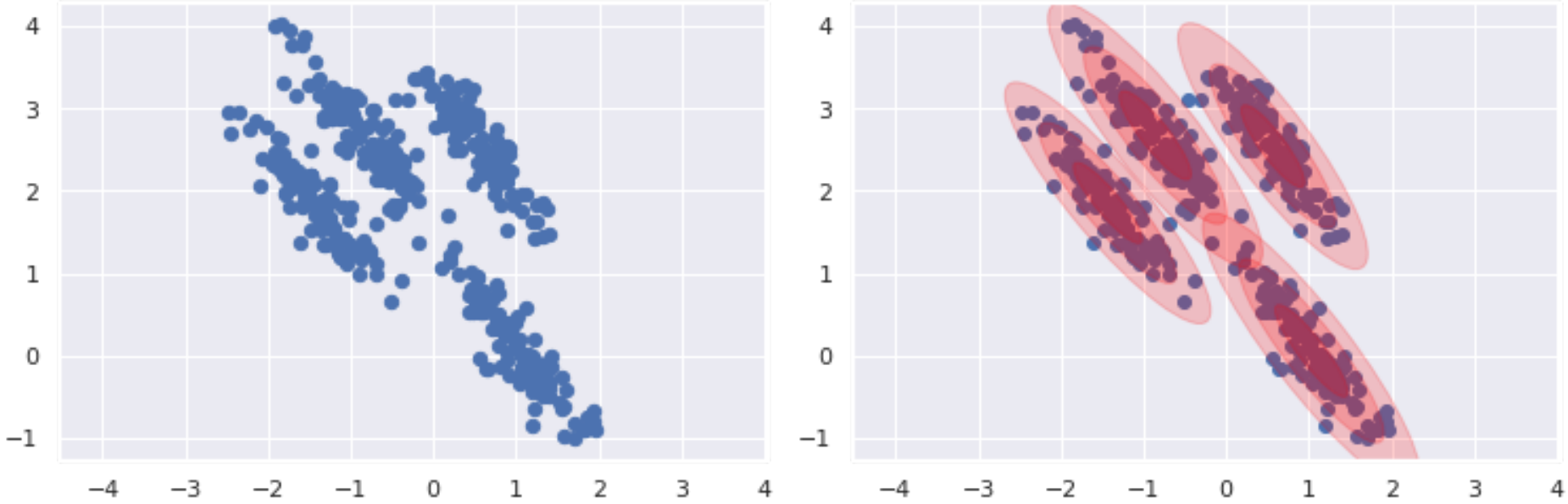


Gaussian Mixture Models

Assumptions

Gaussian mixture models are assuming that the underlying distribution of the data is actually a combination of multivariate gaussian distributions.

The parameters for the gaussian mixture models are obtained via the E-M algorithm. So, it can provide different results with different random seeds.



Gaussian Mixture Models

Probabilistic clustering

Once a GMM is fit to the data, we can extract for any point the probability of belonging to a specific gaussian. This can be used to cluster the points based on the probability.

Gaussian Mixture Models

Usage in Sklearn

- Given a dataset **X**, you typically want to start by estimating the weights **ϕ** and all the distribution parameters $\boldsymbol{\mu}^{(1)}$ to $\boldsymbol{\mu}^{(k)}$ and $\boldsymbol{\Sigma}^{(1)}$ to $\boldsymbol{\Sigma}^{(k)}$

```
1 from sklearn.mixture import GaussianMixture
```

```
1 gm = GaussianMixture(n_components=3, n_init=10, random_state=42)
2 gm.fit(X)
```

```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
                means_init=None, n_components=3, n_init=10,
                precisions_init=None, random_state=42, reg_covar=1e-06,
                tol=0.001, verbose=0, verbose_interval=10, warm_start=False,
                weights_init=None)
```

*Code to reproduce the graphs can be seen at **UUID - #S9CC2***

Gaussian Mixture Models

Estimation of weights ϕ

```
1 gm.weights_
```

```
array([0.39032584, 0.20961444, 0.40005972])
```

*Code to reproduce the graphs can be seen at **UUID - #S9CC2***

Gaussian Mixture Models

Expectation-Maximization (EM)

- Sklearn's `GaussianMixtures` class relies on the *Expectation-Maximization* (EM) algorithm, which has many similarities with the K-Means algorithm

- It also initializes the cluster parameters randomly, then it repeats two steps until convergence:
  - **EXPECTATION STEP:** first assigning instances to clusters
  - **MAXIMIZATION STEP:** updating the clusters

- Sounds a bit similar to what was going on in KMeans right?

Gaussian Mixture Models

As a generalization of KMeans

- GMMs (using the Expectation-Maximization algorithm) can be understood as a generalization of K-Means that not only finds the cluster centers ($\boldsymbol{\mu}^{(1)}$ to $\boldsymbol{\mu}^{(k)}$), but also their size, shape, and orientation ($\boldsymbol{\Sigma}^{(1)}$ to $\boldsymbol{\Sigma}^{(k)}$), as well as their relative weights ($\phi^{(1)}$ to $\phi^{(k)}$).

- Instead of using hard assignments, EM uses soft cluster assignments. Remember the linear decision boundary from KMeans?

Gaussian Mixture Models

EM's soft cluster assignment

- **EXPECTATION STEP**: for each instance, the algorithm estimates the probability that it belongs to each cluster (based on the current cluster parameters).

- **MAXIMIZATION STEP**: each cluster is updated using *all* the instances in the dataset, with each instance weighted by the estimated probability that it belongs to that cluster.
  - These probabilities are called the ***responsibilities*** of the clusters for the instances. During the maximization step, each cluster's update will mostly be impacted by the instances it is most responsible for.

Gaussian Mixture Models

`n_init` again

- Just like K-Means, EM can end up converging to poor solutions, so it needs to be run several times, keeping only the best solution. This is why we set `n_init` to 10. Be careful: by default `n_init` is set to 1.

Gaussian Mixture Models

Other info provided

- Sklearn also tells us if with the parameters given, we have actually converged the model, and how many iterations it took

```
1 gm.converged_
```

```
True
```

```
1 gm.n_iter_
```

```
4
```

*Code to reproduce the graphs can be seen at **UUID - #S9CC2***

Gaussian Mixture Models

Prediction and Generation

- When we fit our GMM we will get an estimate of the location, size, shape, orientation, and relative weight of each cluster

- So now we can easily assign each instance to the most likely cluster (**hard clustering**) using `predict()` method

- And we can also estimate the probability that it belongs to a particular cluster (**soft clustering**) using the `predict_proba()` method

Gaussian Mixture Models

Prediction and Generation

```
1 gm.predict(X)
```

```
array([0, 0, 2, ..., 1, 1, 1])
```

```
1 gm.predict_proba(X)
```

```
array([[9.76815996e-01, 2.31833274e-02, 6.76282339e-07],
       [9.82914418e-01, 1.64110061e-02, 6.74575575e-04],
       [7.52377580e-05, 1.99781831e-06, 9.99922764e-01],
       ...,
       [4.31902443e-07, 9.99999568e-01, 2.12540639e-26],
       [5.20915318e-16, 1.00000000e+00, 1.45002917e-41],
       [2.30971331e-15, 1.00000000e+00, 7.93266114e-41]])
```

*Code to reproduce the graphs can be seen at **UUID - #S12CC3***

Gaussian Mixture Models

Computational complexity of covariance_type

- The computational complexity of training a `GaussianMixture` model depends on the number of instances m, the number of dimensions n, the number of clusters k, and the constraints on the covariance matrices. If `covariance_type` is `"spherical"` or `"diag"`, it is $O(kmn)$, assuming the data has a clustering structure. If covariance_type is `"tied"` or `"full"`, it is $O(kmn^2 + kn^3)$, so it will not scale to large numbers of features

Gaussian Mixture Models

Anomaly Detection

- ***Anomaly detection*** (also called *outlier detection*) is the task of detecting instances that deviate strongly from the norm. These instances are called *anomalies*, or *outliers*, while the normal instances are called *inliers*

- Some useful cases contain
  - **fraud detection**
  - **detecting defective products** in manufacturing
  - **removing outliers** from a dataset before training another model (which can significantly improve the performance of the resulting model)

Gaussian Mixture Models

Anomaly Detection

- Using a Gaussian mixture model for anomaly detection is quite simple: any instance located in a low-density region can be considered an anomaly. You must define what density threshold you want to use.

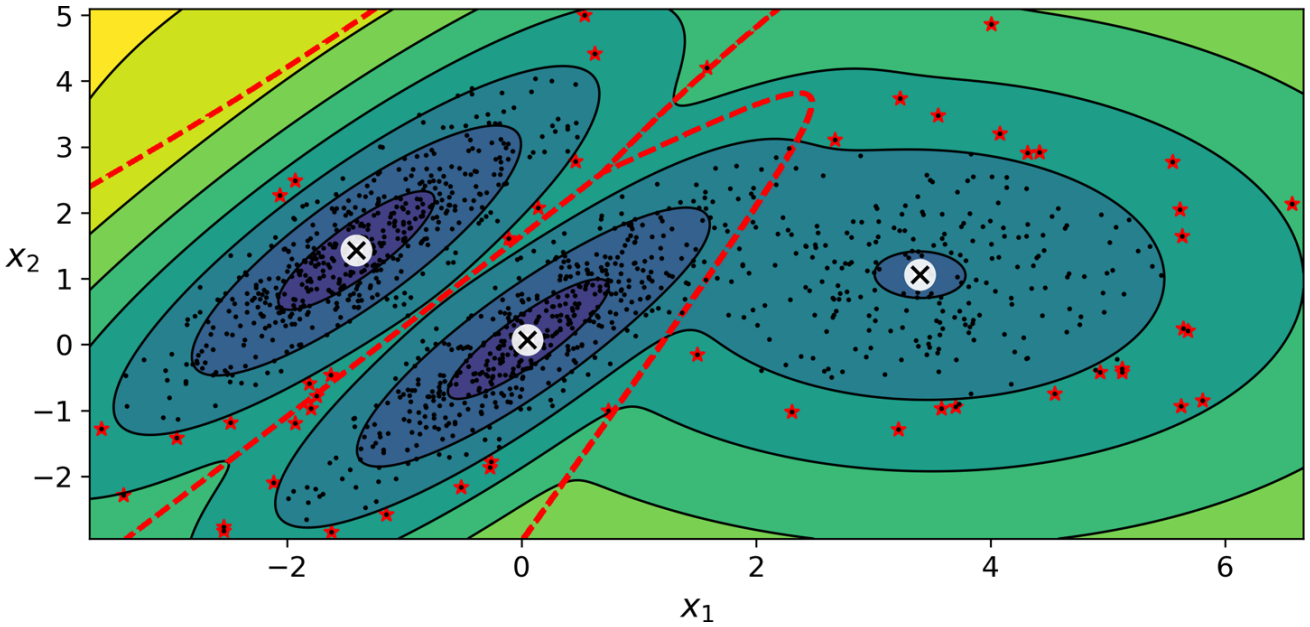
```
densities = gm.score_samples(X)
density_threshold = np.percentile(densities, 4)
anomalies = X[densities < density_threshold]
```

*Code to reproduce the graphs can be seen at **UUID - #S12CC7***

Gaussian Mixture Models

Anomaly Detection

- Using a Gaussian mixture model for anomaly detection is quite simple: any instance located in a low-density region can be considered an anomaly. You must define what density threshold you want to use.



*Code to reproduce the graphs can be seen at **UUID - #S12CC7***

Gaussian Mixture Models

Anomaly Detection

- Gaussian mixture models try to fit all the data, including the outliers, so if you have too many of them, this will bias the model's view of "normality," and some outliers may wrongly be considered as normal. If this happens, you can try to fit the model once, use it to detect and remove the most extreme outliers, then fit the model again on the cleaned-up dataset.

Gaussian Mixture Models

k number selection

- Remember that with K-Means we could use the inertia or the silhouette score to select the appropriate number of clusters.

- With Gaussian mixtures, it is not possible to use these metrics because they are not reliable when the clusters are not spherical or have different sizes.

- Instead, you can try to find the model that minimizes a *theoretical information criterion*, such as the *Bayesian information criterion* (BIC) or the *Akaike information criterion* (AIC),

Gaussian Mixture Models

k number selection

- Both the BIC and the AIC penalize models that have more parameters to learn (e.g., more clusters) and reward models that fit the data well.

- They often end up selecting the same model.

- When they differ, the **model selected by the BIC tends to be simpler** (fewer parameters) than the one selected by the AIC, but **tends to not fit the data quite as well** (this is especially true for larger datasets).

Gaussian Mixture Models

k number selection

```
1 gm.bic(X)
```

```
8189.733705221635
```

```
1 gm.aic(X)
```

```
8102.508425106597
```

*Code to reproduce the graphs can be seen at **UUID - #S12CC8***

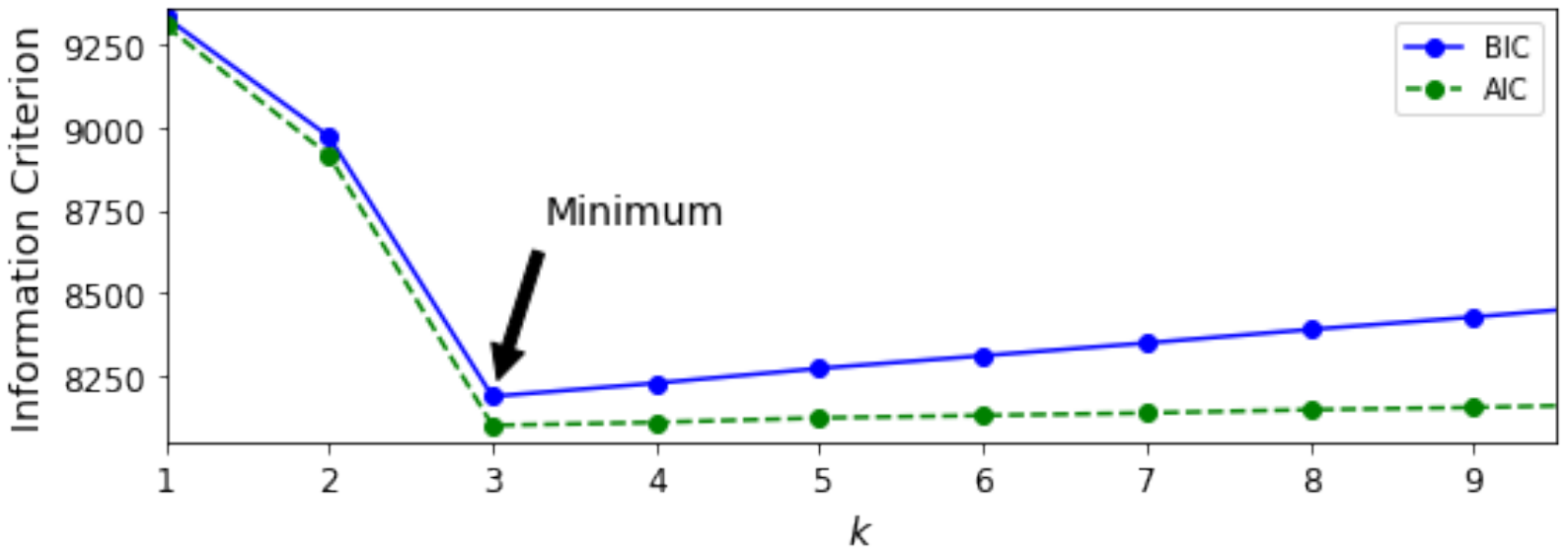
Gaussian Mixture Models

k number selection

- Let's plot different GMMs and their respective BICS and AICS

```
gms_per_k = [GaussianMixture(n_components=k, n_init=10, random_state=42).fit(X)
             for k in range(1, 11)]
```

```
1 bics = [model.bic(X) for model in gms_per_k]
2 aics = [model.aic(X) for model in gms_per_k]
```



*Code to reproduce the graphs can be seen at **UUID - #S12CC8***

BTS | Barcelona Technology School

MBDS 2020-2021

RECAP

Unsupervised learning

Recap

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

Unsupervised learning

Recap

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

Unsupervised learning

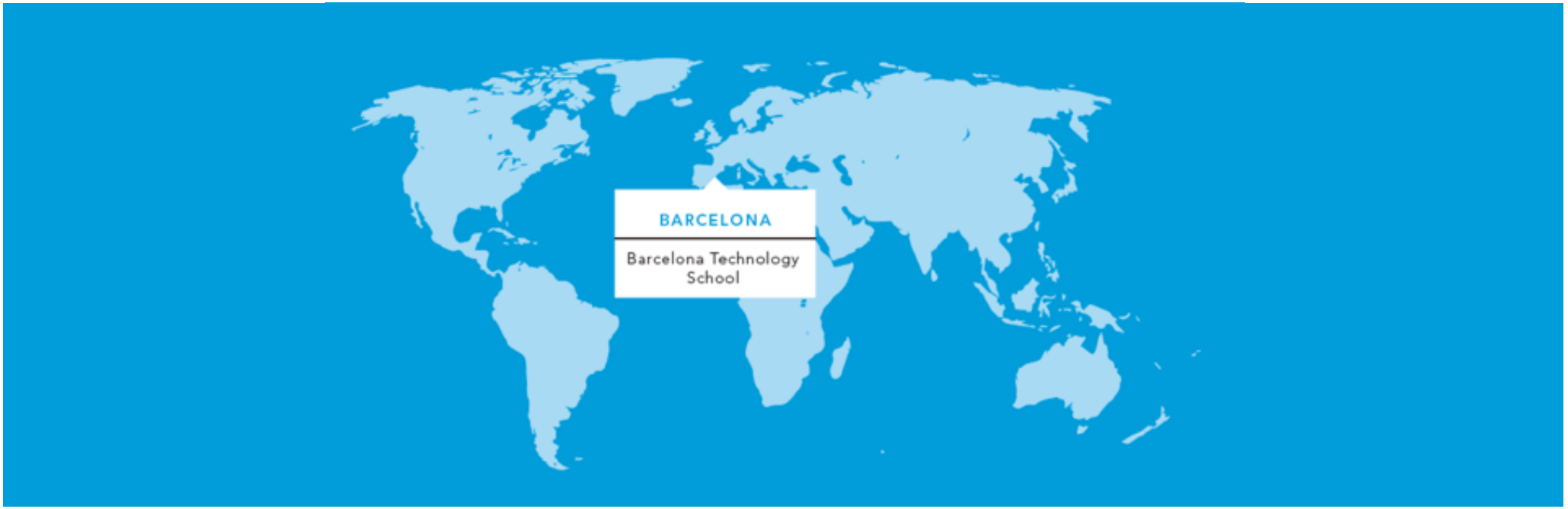
Recap

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large `n_samples`, medium `n_clusters` with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with `n_samples`	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium `n_samples`, small `n_clusters`	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large `n_samples` and `n_clusters`	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large `n_samples`, medium `n_clusters`	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large `n_samples`, large `n_clusters`	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large `n_clusters` and `n_samples`	Large dataset, outlier removal, data reduction.	Euclidean distance between points

Resources

Important resources

- Lex Friedman Series (MIT)
- Sklearn docs
- Papers linked through the algorithm explanation
- Tan, Pang-Ning. Introduction to data mining. Pearson Education India, 2006.
- Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning. Vol. 1. New York: Springer series in statistics, 2001.
- Introduction to Machine Learning with python. Andreas C. Müller & Sarah Guido.
- Jake VanderPlas, “Python Data Science Handbook”
- Aurelien Geron, “Hands-on machine learning with scikit-learn, Keras & Tensorflow”


BARCELONA
Barcelona Technology School