DBSCAN: Density-Based Spatial Clustering of Applications with Noise

Riva Davide, ID 4632421

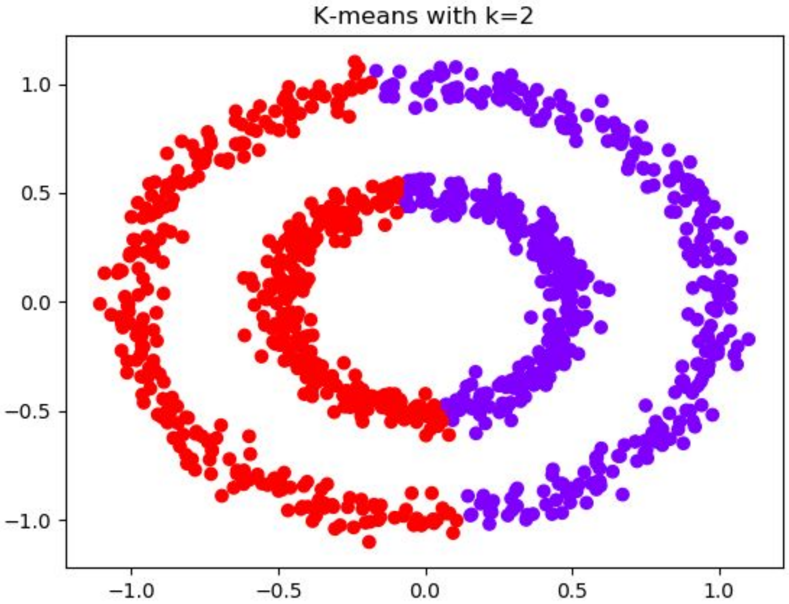
DBSCAN

DBSCAN is an unsupervised learning algorithm for clustering problems.

Basic ideas:

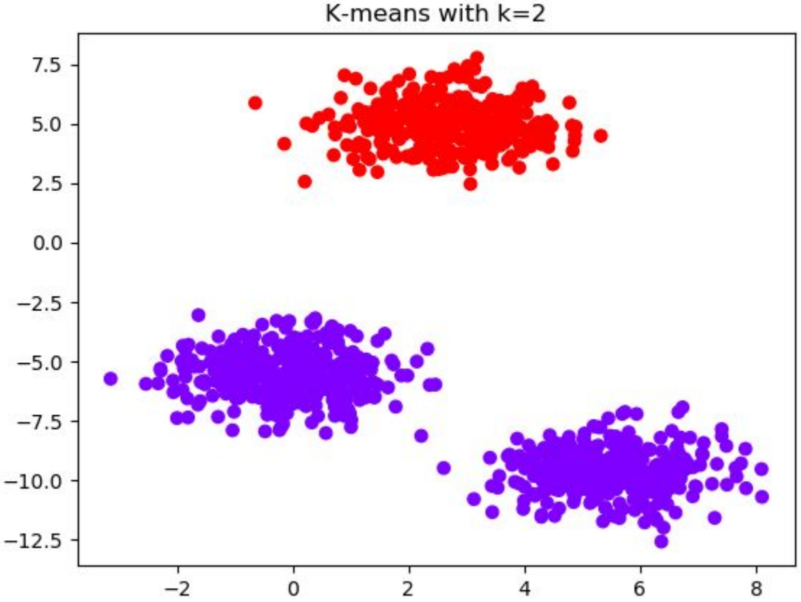
- group together points in high-density regions, marking them as a cluster;
- mark as noise the points which lie in low-density regions.

Is K-means not enough?



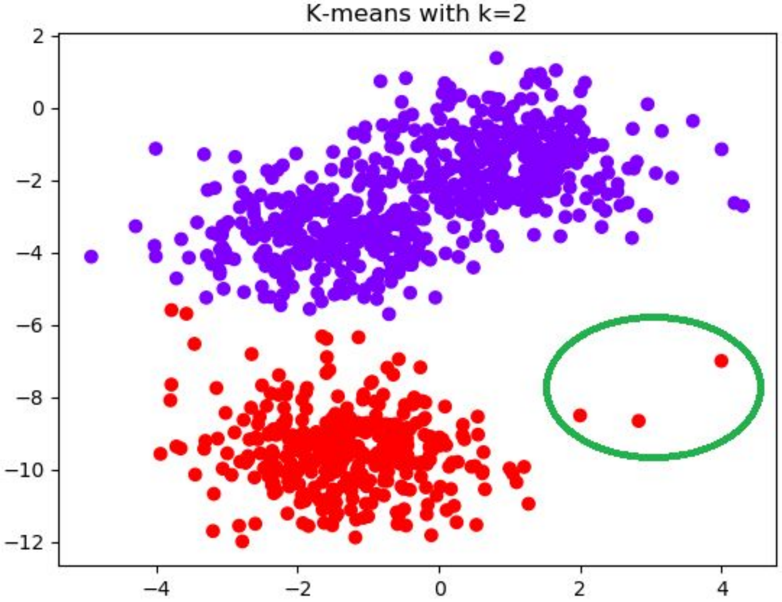
It doesn't work well when there are cluster centroids with similar values or cluster which are not circular

Is K-means not enough?



You have to know in advance the number of clusters (the k parameter)

Is K-means not enough?



It doesn’t have a notion of noise/outliers

Parameters of the algorithm

Ɛ (Epsilon): given a generic point *P* in a dataset *D*, it's the radius of the neighbourhood of the point *P*.

$$\mathcal{E}\text{-neighbourhood}(P) = \{Q \in D \setminus \{P\} \mid d(P, Q) \leq \mathcal{E}\}$$

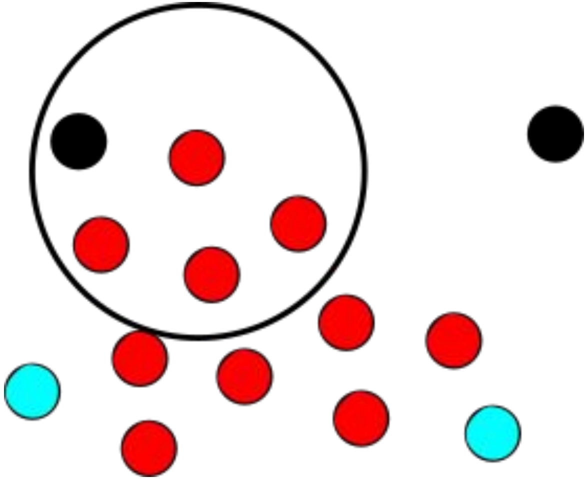
MinPts: given the Ɛ-neighbourhood of a point P, it's the minimum number of points in the neighbourhood to mark P as a core point.

Core Point Condition: $N_\varepsilon(P) \geq \text{MinPts}$, where $N_\varepsilon(P) = |\mathcal{E}\text{-neighbourhood}(P)|$

Some definitions

A generic point in the dataset can be a:

- core point: it satisfies the CPC;
- border point: it doesn’t satisfy the CPC, but it’s in the neighbourhood of at least one core point;
- noise point otherwise;

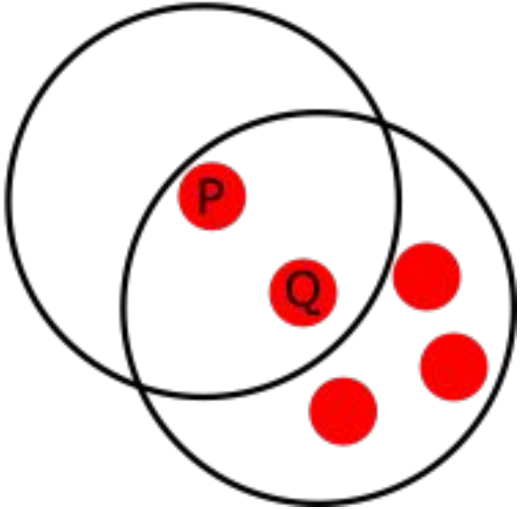


Direct Density Reachability

P is directly density-reachable from Q if and only if:

- Q satisfies the CPC;
- P ∈ Ɛ-neighbourhood(Q).

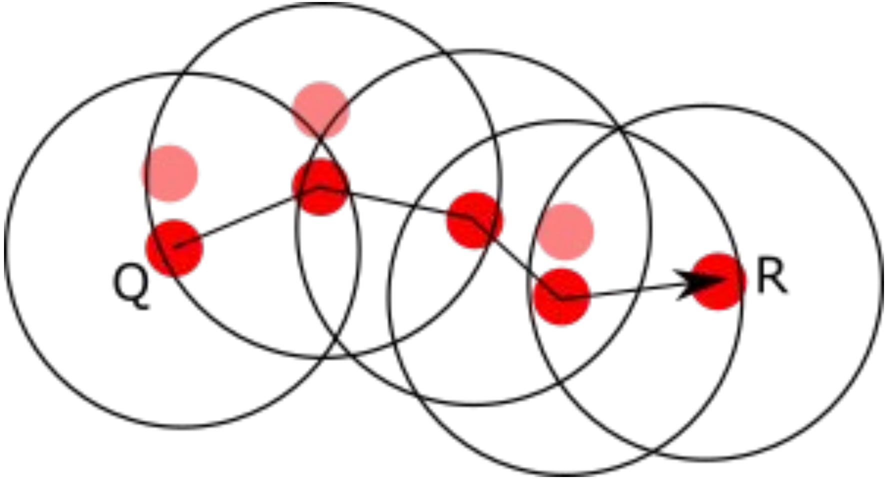
DDR is asymmetric: if P is DDR from Q, it doesn't imply that Q is DDR from P.



Density Reachability

R is density-reachable from Q if and only if it exists a chain of points (P_1, P_2, …, P_N) with P_1= Q, P_N= R | ∀i ∈ {0, …, N-1} P_{i+1} is DDR from P_i.

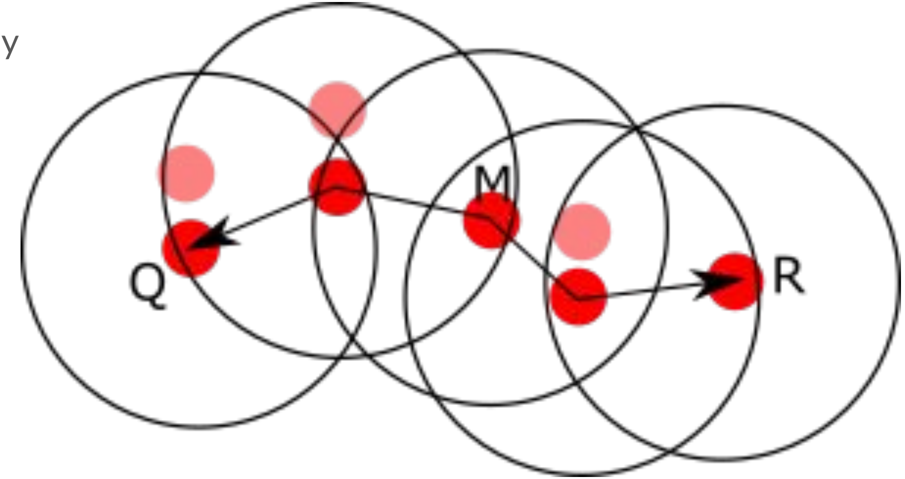
Since the DDR is asymmetric, the DR is also asymmetric.



Density Connectivity

Two points Q and R are density connected if they are both density-reachable from a point M.

Unlike the DR, the DC is symmetric.



Clusters

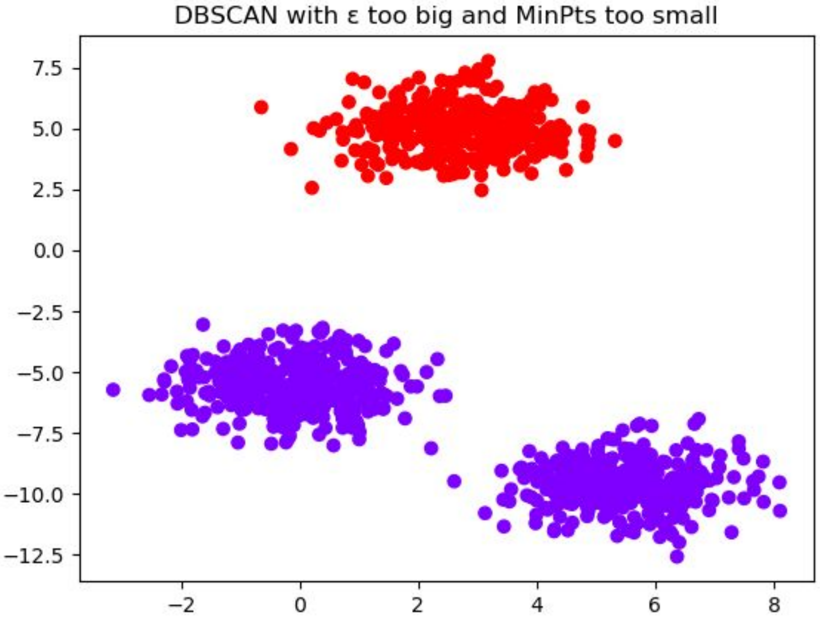
A cluster is a set of points which are density connected to each other.

For each point P which is not yet classified as a cluster point or an outlier:

- if P doesn’t satisfy the CPC, it’s labelled as noise;
- if P satisfies the CPC, the point will be added to a cluster C_i. All points in the **Ɛ**-neighbourhood(P) are also in the cluster C_i. Points that satisfy CPC are also searched from the **Ɛ**-neighbourhood(P). This process is repeated recursively for neighbours of neighbours, until no point satisfies the CPC.

Choosing DBSCAN parameters

Choosing the right value for ε and MinPts is a non-trivial problem



Choosing DBSCAN parameters: MinPts

If MinPts = 1, every point in the dataset are a cluster themselves.

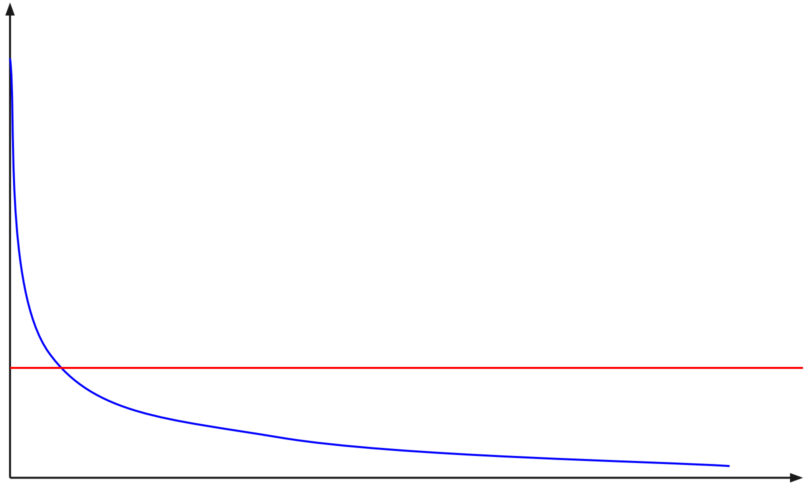
If MinPts = 2, DBSCAN will produce the same result as hierarchical clustering with Single Linkage, cutting the dendrogram when all cluster distances are greater than Ɛ.

The choice of MinPts is strongly related to the number of dimensions D of the dataset (see Rosasco’s notes, chapter 2.4). As a rule of thumb, MinPts ≥ D+1.

Choosing DBSCAN parameters: ε

For each point in the dataset, we calculate the average distance of the K-nearest neighbours, where K = MinPts - 1. Then, we sort the points by this value in descending order and we plot the result. As a rule of thumb, the value for Ɛ is chosen where the plot shows a “knee”.

If Ɛ is too small, lots of points will be marked as noise. If Ɛ is too big, clusters that were separated will merge.



Choosing DBSCAN parameters: distance function

The quality of the result depends on the chosen distance function.

The most commonly used distance function is the Euclidean metric d(P, Q) = sqrt((P_1 - Q_1)^2 + … + (P_N - Q_N)^2). This is no longer true for high dimensional data.

Different distance functions can also be chosen to fit a particular problem. For instance, the Hamming metric is used to measure the distance between strings.

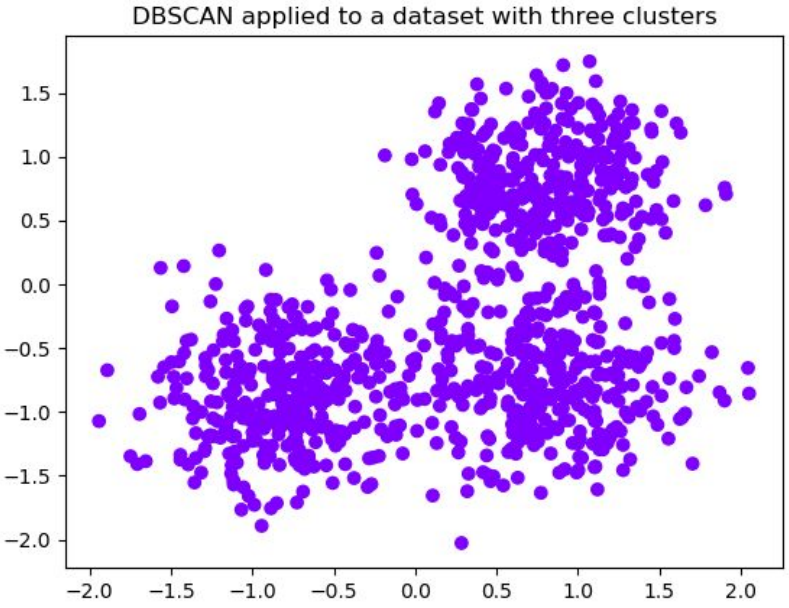
Complexity

The complexity of the algorithm is strongly related to the function that finds neighbours.

Assuming the complexity of that function is O(log n), we obtain that the overall complexity is O (n * log n) since we are iterating through the entire dataset of n points.

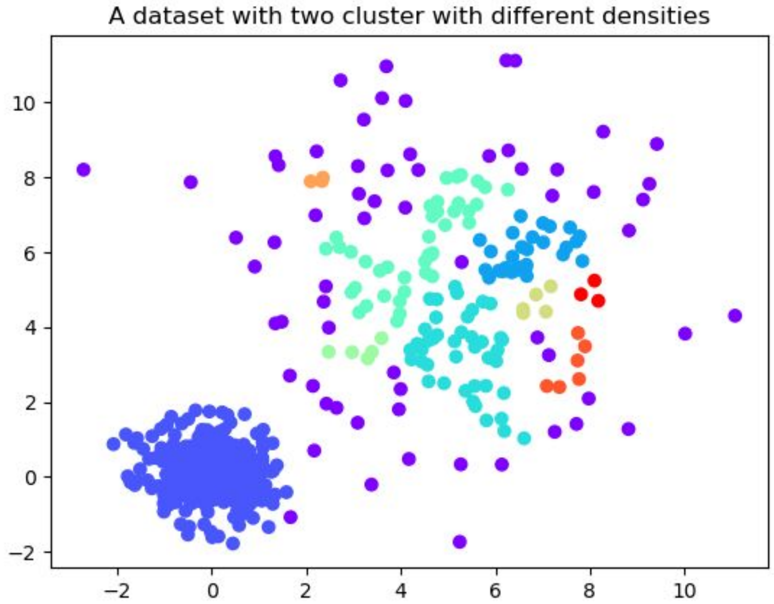
The implementation of DBSCAN that I'm going to propose in the next slides has instead a complexity function of O (n * n) because it doesn't use an index structure.

Cons of DBSCAN



DBSCAN doesn’t work well if clusters have wide overlapping regions

Cons of DBSCAN



DBSCAN doesn’t work well when we’re dealing with clusters of different densities

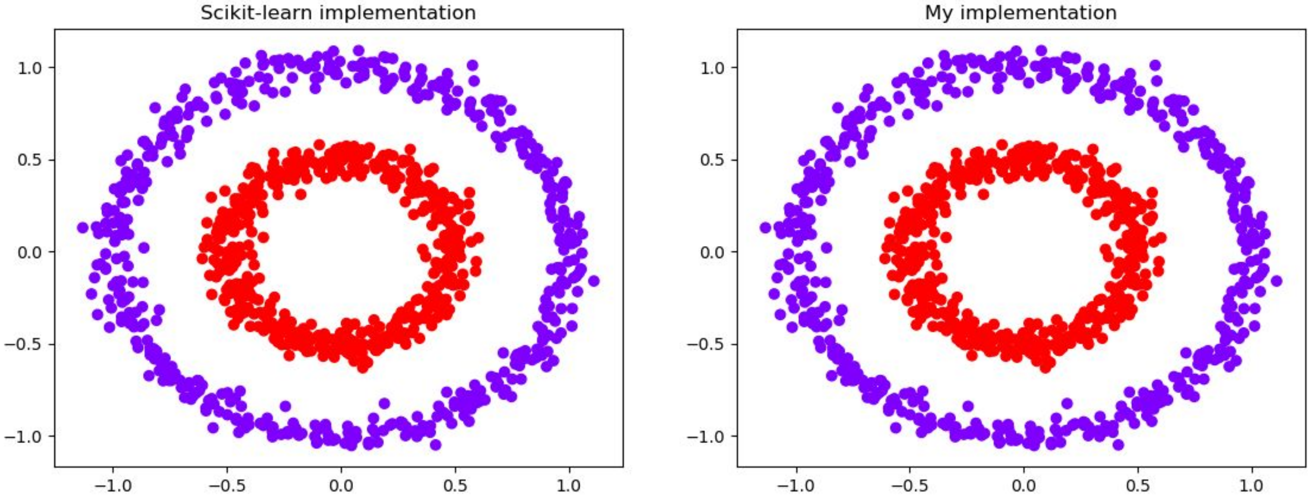
Hands on

A well-documented implementation of this algorithm is proposed as a Python script.

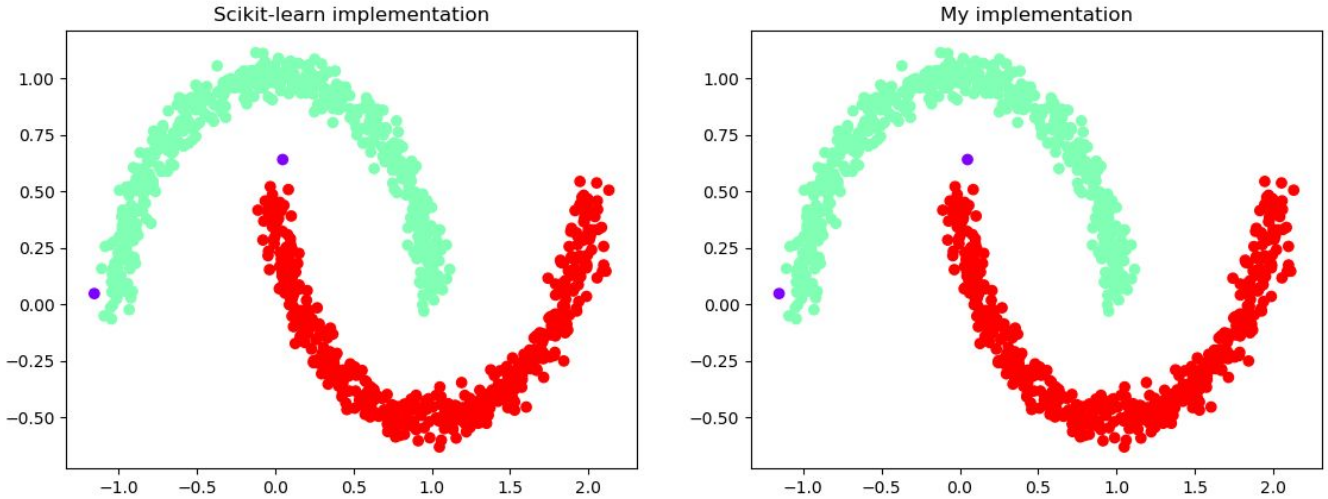
This implementation uses the Euclidean metric as a distance function and it’s independent from the cardinality and the dimensionality of the dataset.

For testing purposes, it’s also graphically compared with the Scikit-learn implementation.

Comparisons



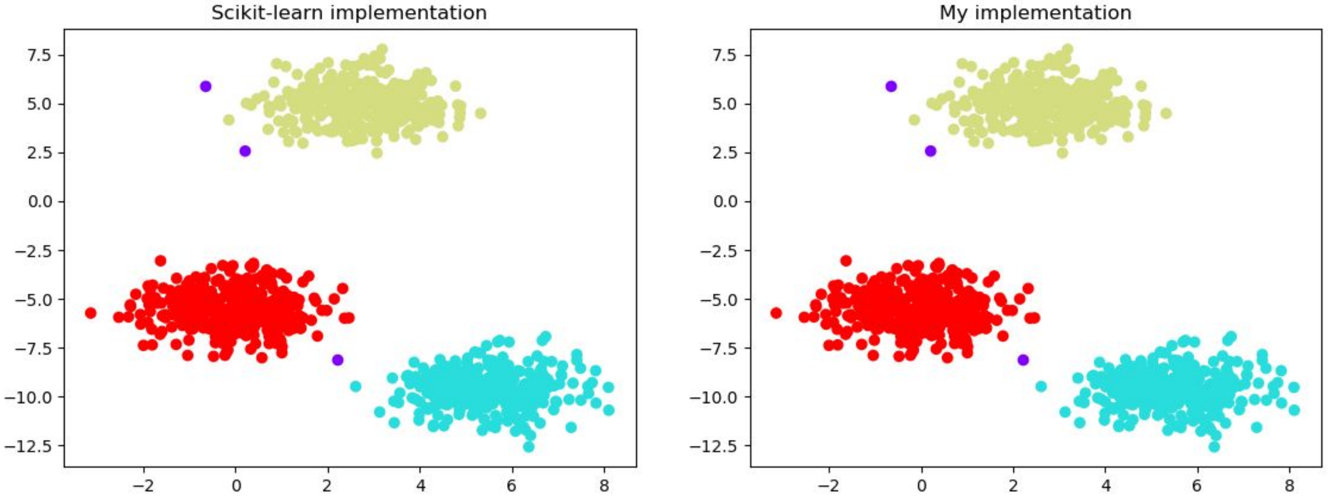
Comparisons



Scikit-learn implementation

My implementation

Comparisons



Scikit-learn implementation

My implementation

Recap

Pros	Cons
It works well even if clusters have different shapes	It doesn’t work well with clusters of different densities
It has a notion of noise/outliers	It doesn’t work well if clusters have overlapping regions
Easy to implement	It’s not suited for high dimensional datasets
Lots of literature on this algorithm (used since 1996)	Not entirely deterministic

Additional information

All images in which clustering techniques are applied were generated from a Python script made by myself.

The main sources of information from which I prepared these slides are the original paper introducing DBSCAN (“A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”) and the Wikipedia page of DBSCAN (https://en.wikipedia.org/wiki/DBSCAN).