Section 5: DBSCAN

Section 5.1: DBSCAN Algorithm

DBSCAN : What is it?

- DBSCAN is an acronym for Density Based Spatial Clustering of Applications with Noise
- Density based clustering approach grouping points closely clustered together and classifying points in low density regions as noise
- User specifies density (a radius and minimum number of points) for a cluster to exist

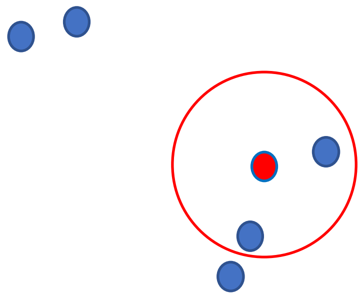
DBSCAN: Core Points and Noise Points

Specify minimum number of points (minpts=3 in example) and radius ε

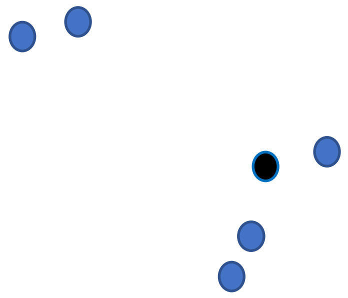
(A) Find neighbours of a data point (all points within distance of ε)

(B) If number of neighbours is at least minpts (including data point), then trial point is CORE

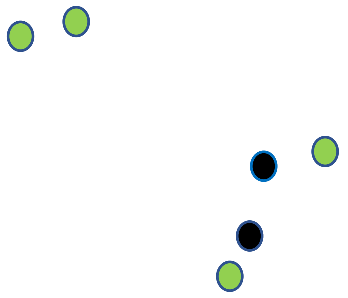
(C) If data point doesn't have minpts neighbours, then it is a NOISE point



A: Focus on red point and count neighbours in ε ball



B: Since number of neighbours is 3 it is CORE – label as black



C: Initial analysis shows 2 CORE and 4 NOISE points

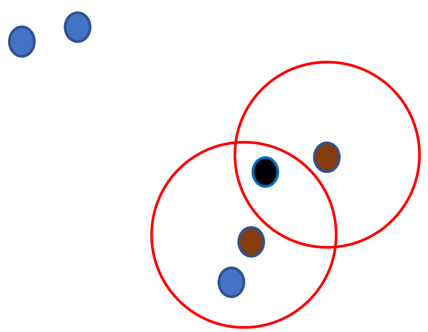
DBSCAN: Building a Cluster from Core Point

(D) Determine if neighbours of original CORE point are also CORE points

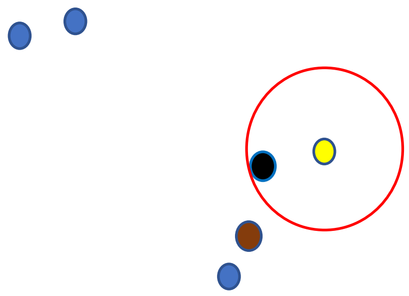
(E) If neighbour is not a core point, then it is a BORDER point

(F) If neighbour is CORE point, then repeat steps (D) and (E) until one runs out of core points

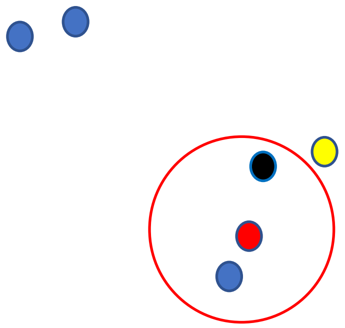
Start a new cluster by checking if unvisited point is CORE



D: Look ε balls around neighbours of original CORE point



E: yellow is BORDER point since it doesn't have 3 neighbour points

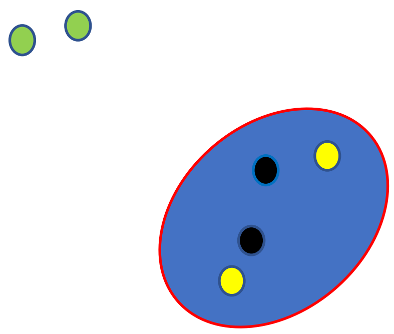


F: red is a CORE point, so repeat steps D and E for red point

DBSCAN: Summary

In this example

- Green are NOISE
- Black are CORE
- Yellow are BORDER
- There is single connected cluster (shown in blue) and the 2 NOISE points



DBSCAN Algorithm

- Assume M data points $\{X_i\}$
- Specify minpts and radius ε

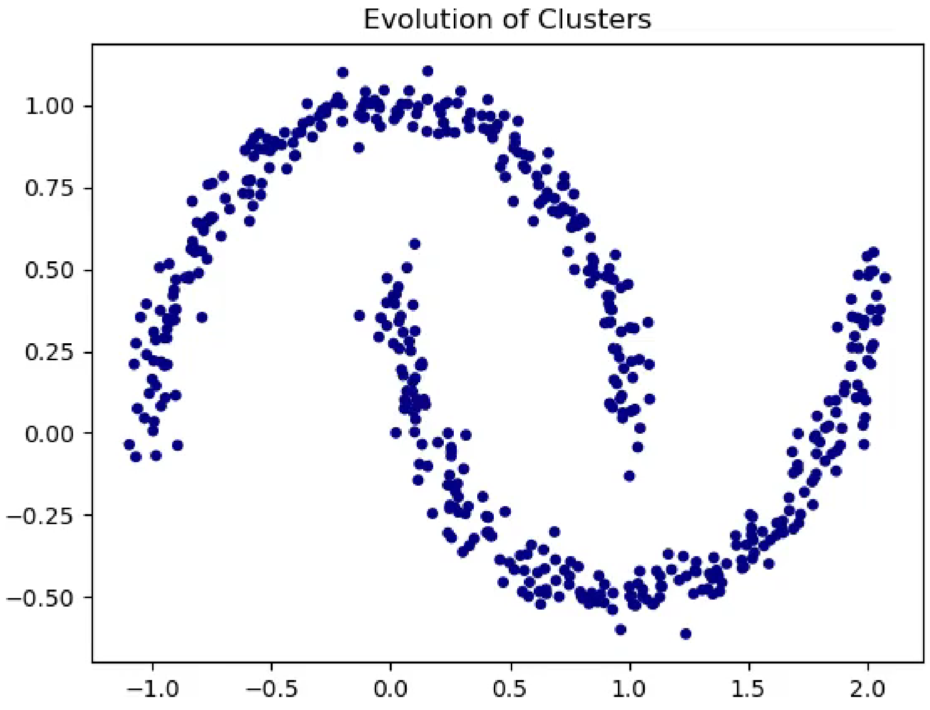
(1) Loop over all data points $\{X_i\}$

  - If X_i is unvisited, then find neighbours else go to next point
  - If number of neighbours less than minpts, label as NOISE and go to next point
  - Label X_i as CORE point and start new cluster
  - S is set of neighbours of X_i
  - Loop over points Y in S
    - If Y is previously defined as NOISE, then relabel as BORDER and go to next Y
    - If Y was visited before, then go to next Y
    - If Y is not Core, then label as BORDER and go to next Y
    - If Y is CORE point, then label as CORE and add its neighbours to S

DBSCAN: Example

Example:

- sklearn noisy_moons dataset with 500 points
- Use minpts = 3 and ε = 0.2



DBSCAN: Choosing minpts and ε

Choosing minpts:

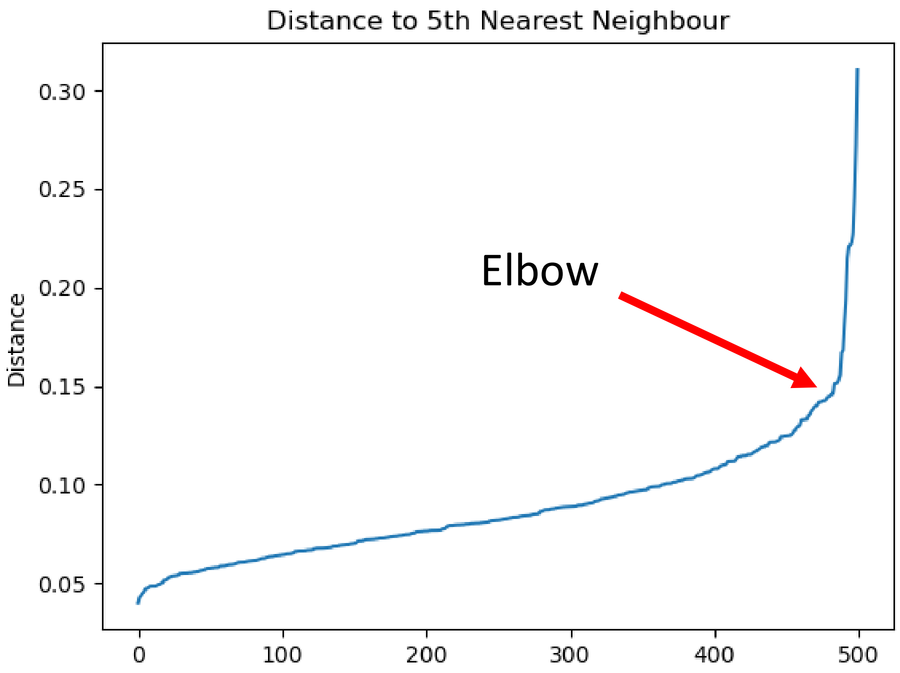
- Rule of thumb is minpts related to number of dimensions d
- Suggested values minpts $\geq$ d+1 or minpts $\geq$ 2d

Choosing ε:

- If ε is too large, then there will be large clusters
- If ε is too small, then there will many small clusters
- Suggested approach using elbow method:
  - For each data point, compute distance to k = minpts – 1 closest point
  - Plot all these distances on a graph and choose ε to be at elbow

Nearest Neighbour Approach for Choosing ε

- Choose minpts
- For each data point compute distance to k = minpts – 1 nearest neighbour
- Sort and plot on graph and choose ε to be at elbow

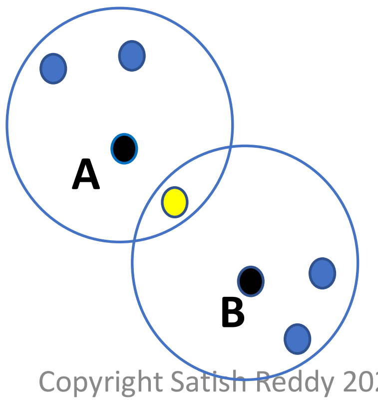


DBSCAN: Complexity

- Assume: M data points in d dimensions
- Worst case scenario requires $O(M^2)$ operations as $M \to \infty$ (each point is a NOISE point, for example)
- Memory requirement is $O(M)$ operations as $M \to \infty$
- Number of operations and memory depend linearly on dimension d

DBSCAN: Notes

- DBSCAN can identify clusters that are arbitrarily shaped
- Must specify minpts and radius ε so tuning required
- Single minpts and ε so method does not do well if clusters have varying densities
  - OPTICS is a variant of DBSCAN that can handle varying densities
- Division into clusters is not unique as a BORDER point may belong to more than one cluster group
  - Consider example of minpts = 4 where yellow BORDER point can belong to cluster based on CORE A or CORE B - BORDER point assigned to first cluster that is created



Section 5.2: DBSCAN Code Design

DBSCAN Code Design

- This section presents design of the Hierarchical Clustering code
- Design is based on algorithm described in Section 6.1
- Stop video here, if you would like to do code design yourself

DBSCAN Code Design: To Do

Component	Description
class dbscan	class kmeans derived from clustering_base
driver_dbscan	driver for dbscan

dbscan class: Principal Variables

Variable	Type	Description
self.X	2d numpy array	Column j is the j’th data point
self.clustersave	list of 1d numpy arrays	self.clustersave[i][j] is the cluster assignment for data point j at level i of algorithm
self.list_label	list of strings	Label for each data point. Label is “unvisited”, “core”, “border” or “noise”

dbscan class – Key Methods

Method	Input	Description
__init__	minpts (integer) epsilon (float)	Constructor for class
initialize_algorithm		Initialize variables for the algorithm: self.clustersave, self.list_cluster Return: nothing
fit	X (2d numpy array)	Performs dbscan clustering Return: nothing
neighbours	Xidx (2d numpy array)	Finds all points within epsilon neighbourhood of point Xidx Return: list of indices of points
extend_cluster	cluster_number (integer) idx (integer) list_neighbours (list)	This function starts to build cluster with label cluster_number starting with "core" point at index idx and the neighbours in list_neighbours Return: nothing

dbscan class – Key Methods

Method	Input	Description
add_points	list_seed (list) list_idx (list)	This function adds points that are potentially part of the current cluster. list_seed is a list of indices of points currently being reviewed. list_idx are indices of points that are to be added to list_seed Return: updated list_seed
update_cluster_ assignment	cluster_number (integer) idx (integer	Update the cluster assignment (cluster is cluster_number) for data point with index idx

Section 5.3: DBSCAN Code Walkthrough

dbscan Clustering: Code Walkthrough

- Code is located in:

  Folder: UnsupervisedML/Code/Clustering

  Files: dbscan.py, driver_dbscan.py

- Stop video here, if you would like to do coding yourself before seeing my implementation