Unsupervised Machine Learning with Python

Section 5.1: DBSCAN Algorithm

DBSCAN Clustering

- DBSCAN is an acronym for Density Based Spatial Clustering of Applications with Noise
- DBSCAN is density based clustering approach grouping close points into clusters and classifying points in low density regions as noise
- User specifies density (a radius and minimum number of points) for a cluster to exist
- See UnsupervisedML_Resources.pdf for links to additional resources

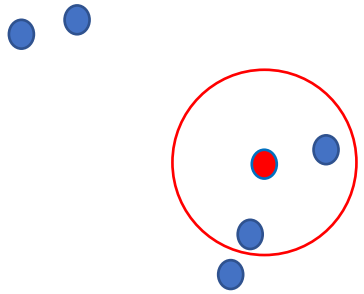
DBSCAN: Core Points and Noise Points

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

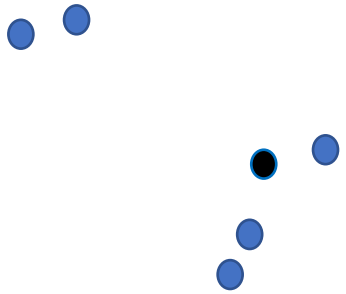
(A) Find neighbours of a data point (all points within distance of ε including the data point itself)

(B) If number of neighbours is at least minpts, then trial point is CORE and we will build cluster from this point

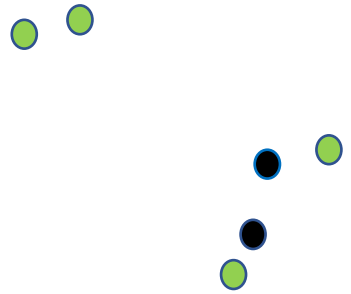
(C) If data point doesn't have minpts neighbours, then it is a NOISE point



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



B: Number of neighbours is at least minpts=3 so point is CORE – label as black



C: Initial analysis shows 2 CORE and 4 NOISE points

DBSCAN: Building a Cluster from Core Point

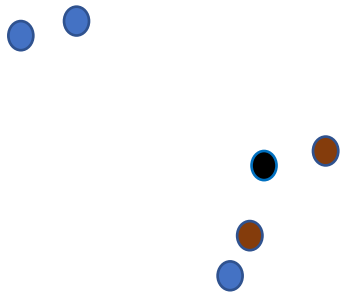
Build cluster starting from CORE point

(D) Investigate neighbours of CORE point

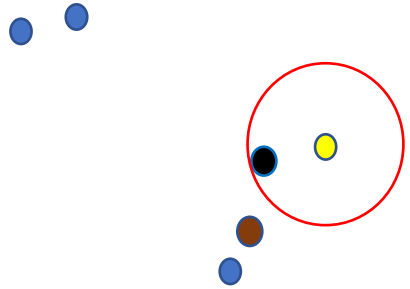
(E) If neighbour is not a core point, then it is a BORDER point (ADD TO CLUSTER)

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

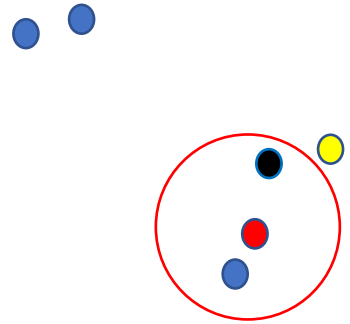
Start new cluster from CORE point that is not part of existing cluster



D: Look ε balls around neighbours of original CORE point



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

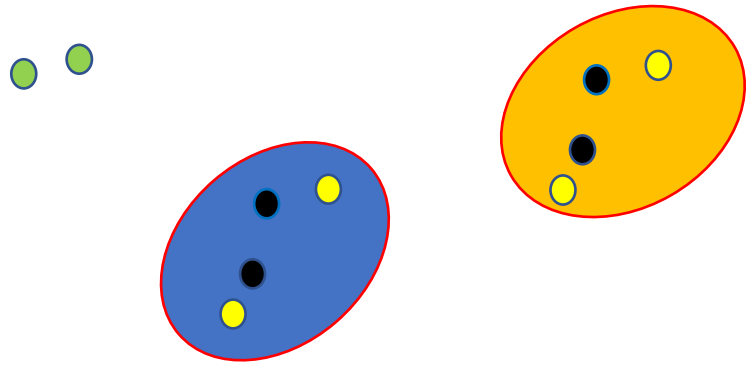


F: red is a CORE point, so repeat steps D, E, F

DBSCAN: Summary

Example with 2 clusters

- Green are NOISE
- Black are CORE
- Yellow are BORDER



DBSCAN Algorithm

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

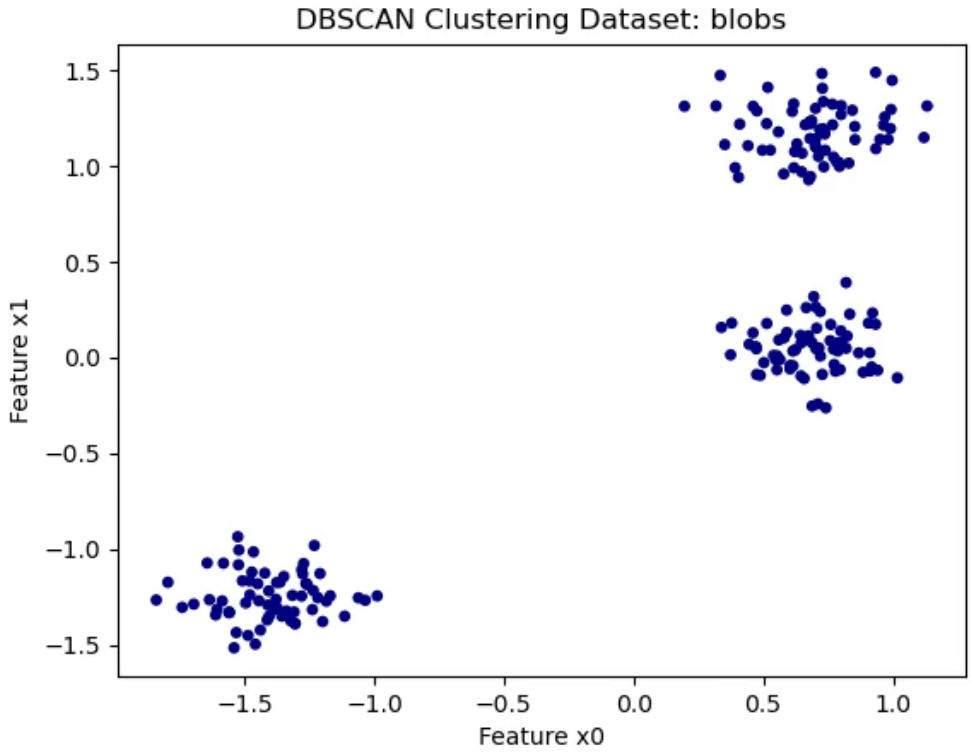
(1) Loop over all data points $\{X_i\}$

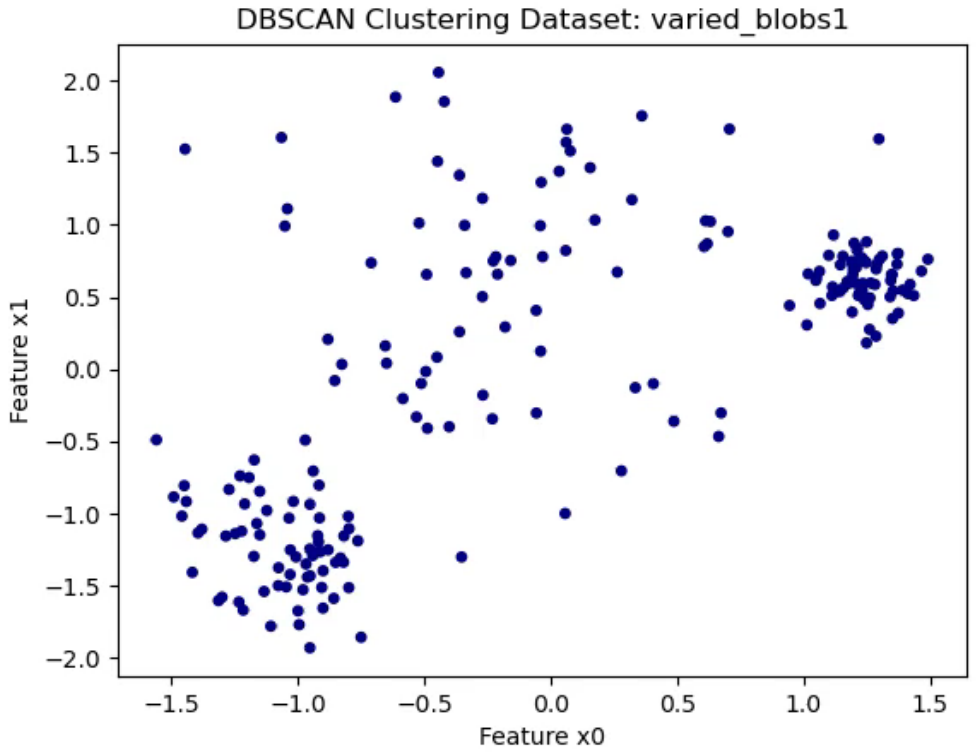
- If X_i has been visited/processed before, then go to next point
- Determine the neighbours of X_i
- If # of neighbours less than minpts, label as NOISE and go to next point
- Label X_i as CORE point and start new cluster
- Loop over the set of points Y in set S (neighbours of X_i)
  - If Y is previously labeled as NOISE, then relabel BORDER, add to cluster and go to next Y
  - If Y was visited before, then go to next Y
  - Add Y to cluster and determine neighbours for Y
  - If # of neighbours less than minpts, label Y as BORDER and go to next Y
  - Else label Y as CORE point and add its neighbours to S

DBSCAN: Example

Example:

- sklearn blobs and varied_blobs1 datasets with 200 points
- Use minpts = 4 and ε = 0.3





DBSCAN: Choice of minpts and ε

Choosing minpts:

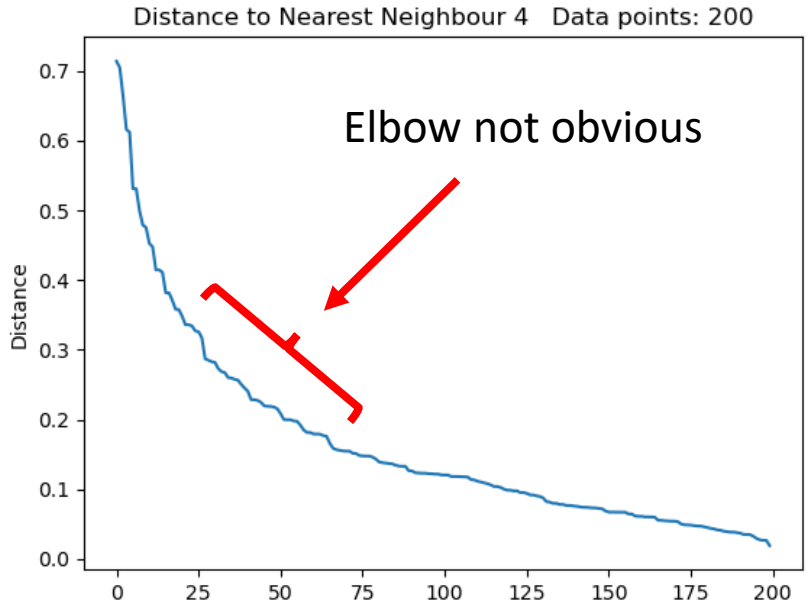
- Rule of thumb is minpts related to number of dimensions d
- Suggested values minpts $\geq$ d+1 or minpts $\geq$ 2d

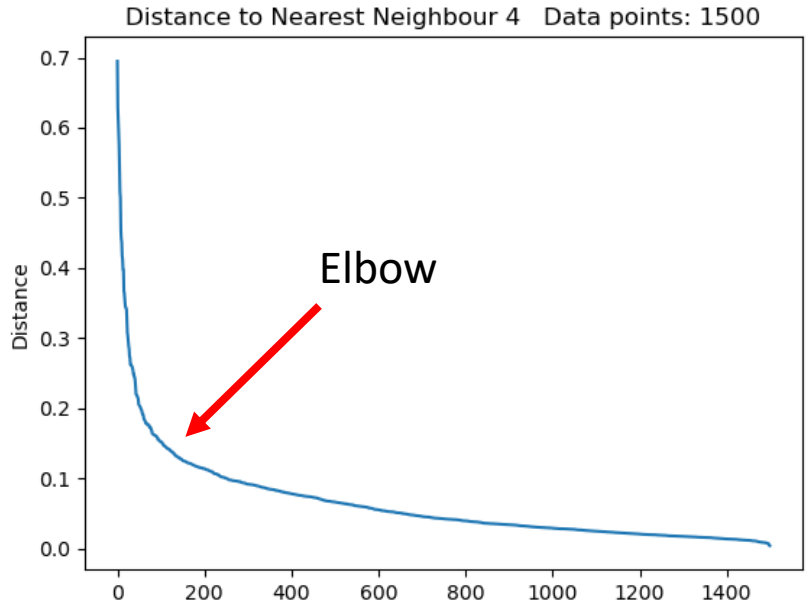
Choosing ε:

- If ε is large (clusters can have low density) -> cluster with many points
- If ε is small (clusters must have high density) -> clusters with few points and many noise points

Example: Choosing ε using Elbow Method

- Choose minpts=4 and “varied_blobs1” dataset
- For each data point compute distance to k = 4 nearest neighbour
- Sort distances in descending order and plot
- Elbow is point where curve starts to level off
- For more details, see Wikipedia page for Elbow Method (link in Resources file)



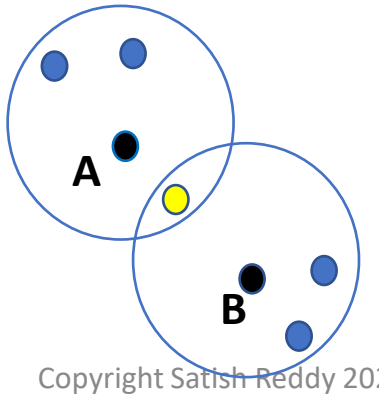


DBSCAN: Complexity

- Assume: M data points in d dimensions
- Worst case scenario requires $O(M^2)$ operations as $M \to \infty$
- Memory requirement is $O(M)$ storage as $M \to \infty$
- Number of operations and storage depend linearly on dimension d
- Can reduce number of operations by using more sophisticated approaches for identifying points in neighbourhood - see Wikipedia page for DBSCAN for more details

DBSCAN: Notes

- Approach identifies clusters that are arbitrarily shaped
- Number of clusters is not pre-defined
- Must specify minpts and radius ε so tuning required
- Single minpts & ε, so method performs poorly 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



Unsupervised Machine Learning with Python

Section 5.2: DBSCAN Code Design

DBSCAN Code Design

- This section presents a design of the DBSCAN Clustering code
- Design is based on algorithm described in Section 5.1
- Stop video here, if you would like to do code design yourself

DBSCAN Code Design

(1) Derive dbscan class from clustering_base class

(2) Introduce self.list_label, which is label for each data point

- “unvisited” if point has not yet been processed/assigned a label
- “core”
- “noise”
- “border”

(3) Assign cluster label = -1 to all data points initially

- Points in first cluster assigned label 0, points in second cluster assigned label 1, …
- Noise points continue to have label = -1

(4) Code design includes additional functionality to create animation

- Included for teaching purposes

dbscan class: Principal Variables

Variable	Type	Description
self.time_fit	float	Time for clustering
self.X	2d numpy array	Contains the dataset Number of rows = number of dimensions for data Number of cols = number of data points Example: 2 dimensions and 5 data points $\begin{bmatrix} 1 & 1.1 & 0.8 & 0.6 & 0.6 \\ 0.9 & 1.0 & 0.7 & 0.5 & 0.5 \end{bmatrix}$
self.clustersave	list of 1d numpy arrays	self.clustersave[i][j] is cluster assignment for iteration i, data point j Example: for 3 iterations: $[[-1 \quad -1 \quad -1 \quad -1 \quad -1], [-1 \quad -1 \quad 0 \quad -1 \quad -1],$ $[-1 \quad -1 \quad 0 \quad 0 \quad -1]]$
self.list_label	list of strings	Label for each data point: one of “unvisited”, “core”, “border” or “noise”
self.animation	boolean	If True, then each time data point assigned to a cluster, add new entry for self.clustersave for animation purposes. If false, self.clustersave has 2 entries: initial assignment and final assignment.
self.minpts self.epsilon2	integer float	Minimum number of points for a core point and epsilon squared

dbscan class – Key Methods

Method	Input	Description
__init__	minpts (integer) epsilon (float) animation (boolean)	Constructor for class
initialize_algorithm		Initialize variables for the algorithm: self.clustersave, self.list_label Return: nothing
fit	X (2d numpy array)	Performs dbscan clustering Return: nothing
neighbours	idx (integer)	Finds all points within distance epsilon of point with index idx Return: list of indices
extend_cluster	cluster_number (integer) list_neighbour (list)	This function builds cluster with label cluster_number starting with data points in list_neighbour Return: nothing
add_points_to_s	list_s (list) already_in_s (list) list_neighbour (list)	Add points to list_s, the points in set S (neighbours of core points). list_neighbour is set of points to be added, already_in_s is boolean indicating if point has already been considered Return: list_s, already_in_s
update_cluster_ assignment	cluster_number (integer) idx (integer)	Update self.clustersave: point idx is assigned label cluster_number Return: nothing

Unsupervised Machine Learning with Python

Section 5.3: DBSCAN Code Walkthrough

DBSCAN Clustering: Code Walkthrough

Code located at:

- UnsupervisedML/Code/Programs

Files to Review	Description
dbscan.py	Class for DBSCAN clustering
driver_dbscan.py	Driver for DBSCAN clustering

Course Resources at:

- https://github.com/satishchandrareddy/UnsupervisedML/

- Stop video if you would like to implement code yourself first