

DENSITY BASED CLUSTERING: DBSCAN

Michele Rossi

michele.rossi@unipd.it

Dept. of Information Engineering
University of Padova, IT



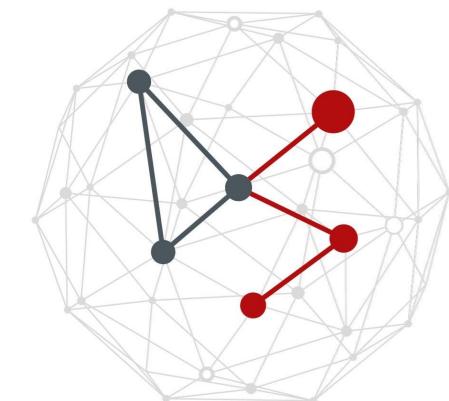
DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE



DIPARTIMENTO
MATEMATICA



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



Overview

- Motivation
 - Limitations of partition-based clustering
- DBSCAN
 - Reference papers
 - Rationale, main features
 - Density reachability
 - Density connected points
- The DBSCAN algorithm
- Examples, discussion
- Setting DBSCAN pars?

The clustering problem

- **Dataset:** $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ in an m-dimensional space $\mathbf{x}_i \in \mathbb{R}^m$
- **Goal (informal)**
 - partition the points into K disjoint clusters (K is given)
 - Use Euclidean distance (i.e., norm-2)
- **For each cluster i =1,2,..., K**
 - We define a **prototype** (or **centroid**) $\boldsymbol{\mu}_i \in \mathbb{R}^m$
 - Can be thought of as the “center” of the cluster
- **Goal (restated)**
 - Assign the N data points to the K clusters (*each point assigned to a single cluster*) and find a set of centroids such that:

the sum of the squares of the distance of each data point to its closest centroid is a minimum

Cost function

- For each data point, we introduce a binary indicator variable $r_{nk} \in \{0, 1\}$

$$r_{nk} = \begin{cases} 1 & \text{if } \mathbf{x}_n \rightarrow \mathcal{C}_k \quad (\text{cluster k}) \\ 0 & \text{otherwise} \end{cases}$$

- This is known as the 1-of-K coding scheme
- We define a cost function (aka “distortion measure”):

$$J = \sum_{n=1}^N \sum_{k=1}^K r_{nk} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

- Goal: minimize J

Partition vs density-based clustering

- Partitioning algorithms (e.g., K-means and the like)
 - N objects, to cluster into K classes
 - K may be known or not
- Usual two-step procedure
 - Determine K by minimizing an objective (cost) function
 - Assign each object to the closest cluster: this means that a partition is equivalent to a Voronoi diagram → each cluster is contained in a Voronoi cell → clusters tend to be convex
- Problem
 - Convex clusters are not always appropriate

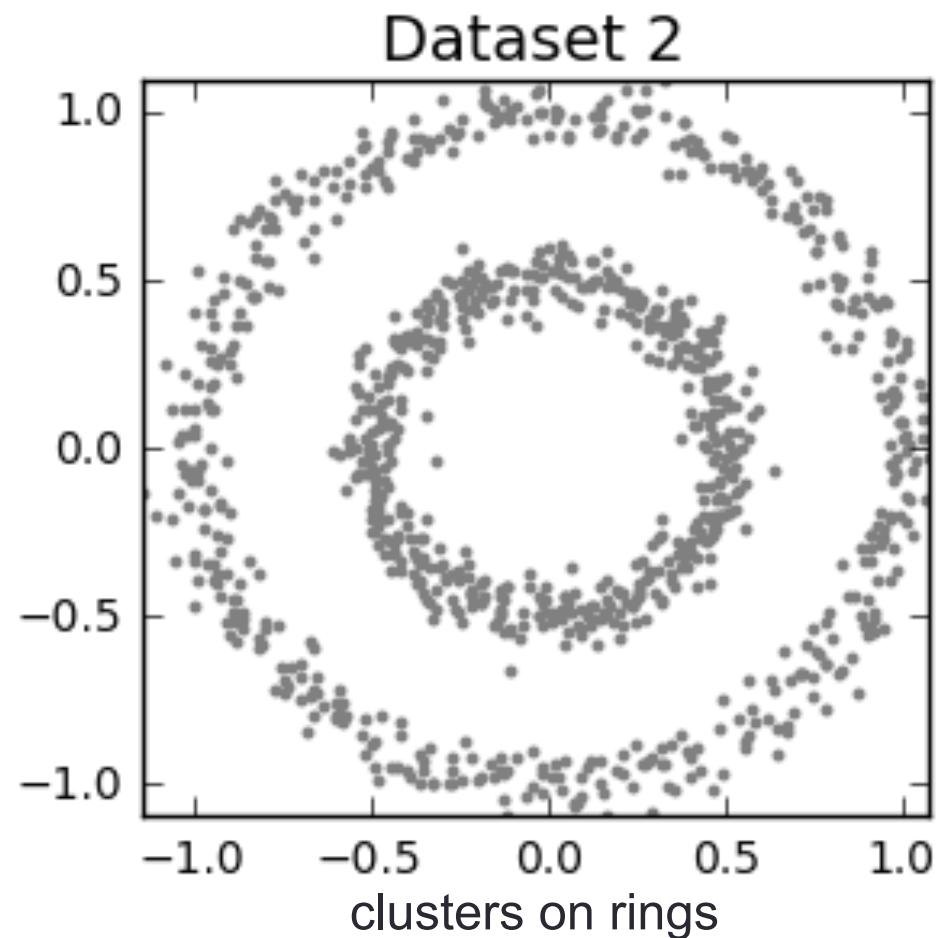
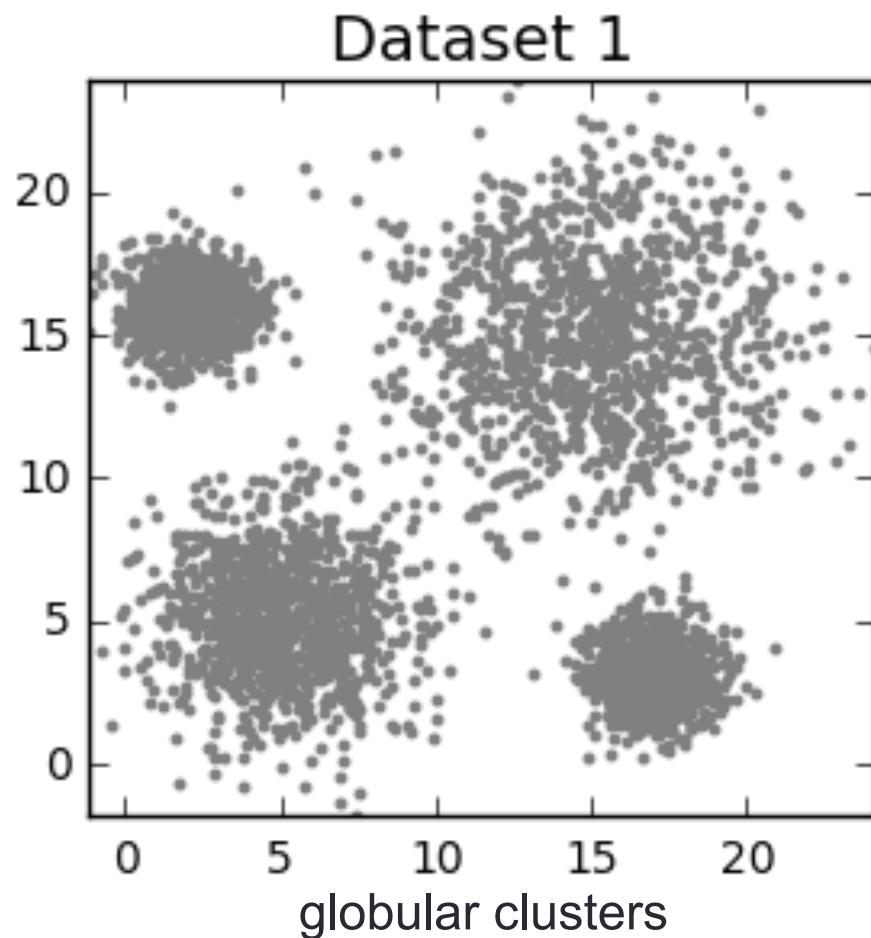
Popular solutions

- K-means
 - K centroids, K fixed and given
 - Hard association of points to clusters
- Soft-K means
 - K centroids, K fixed and given
 - Soft association of points to clusters
 - Probabilistic version of K-means
- X-means [Peleg-00]
 - Unsupervised
 - K is found automatically

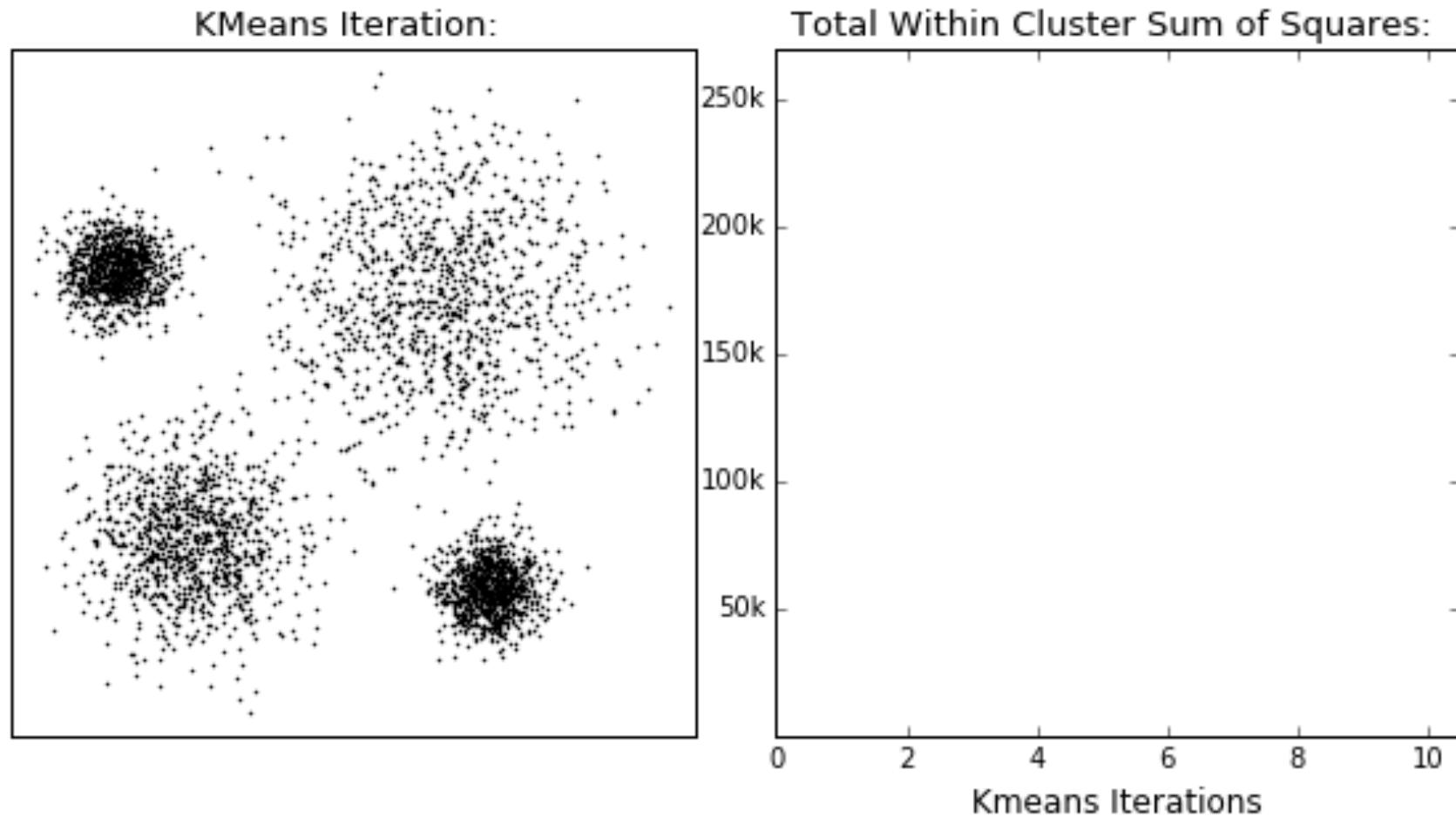
[Peleg-00] Dan Peleg, Andrew Moore, “X-means: Extending K-means with Efficient Estimation of the Number of Clusters,” *International Conference on Machine Learning (ICML)*, Stanford, CA, USA, 2000.

[4010 citations, Oct 2025]

Two datasets



Dataset 1



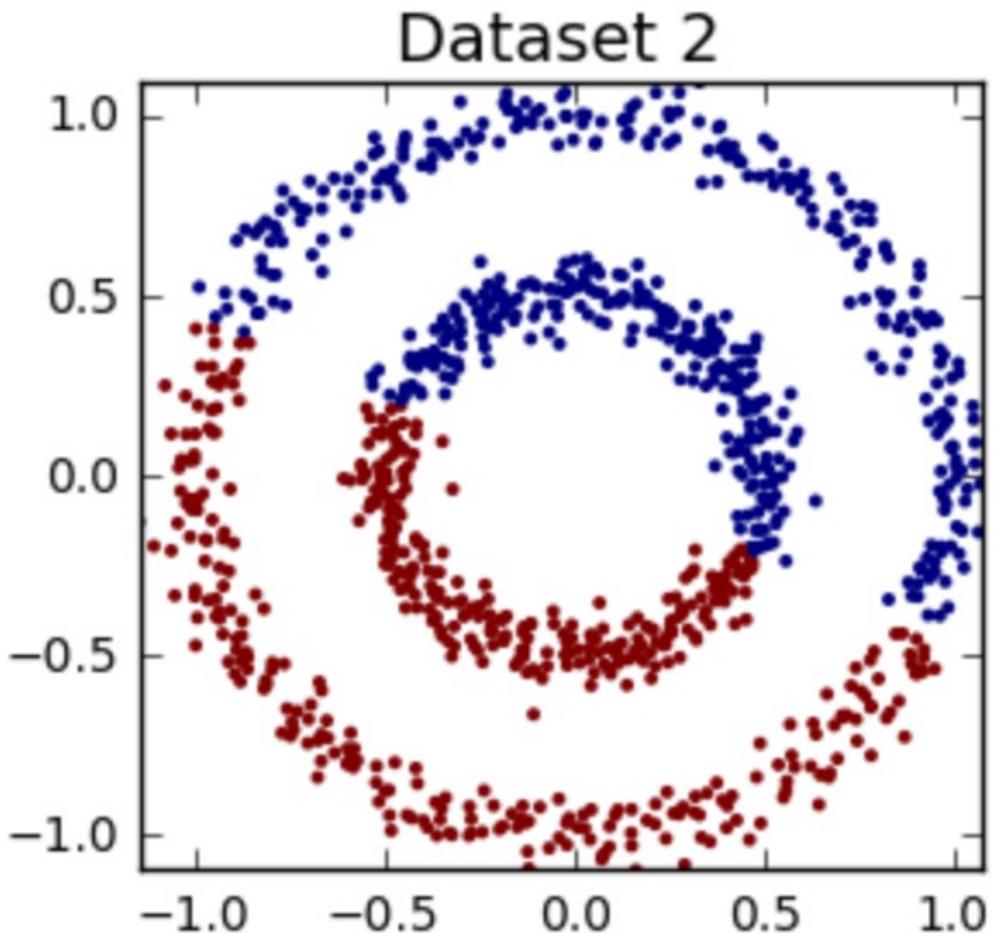
- All OK, clusters are satisfactory!

Dataset 2

K-means identifies globular clusters
(essentially spherical clusters)

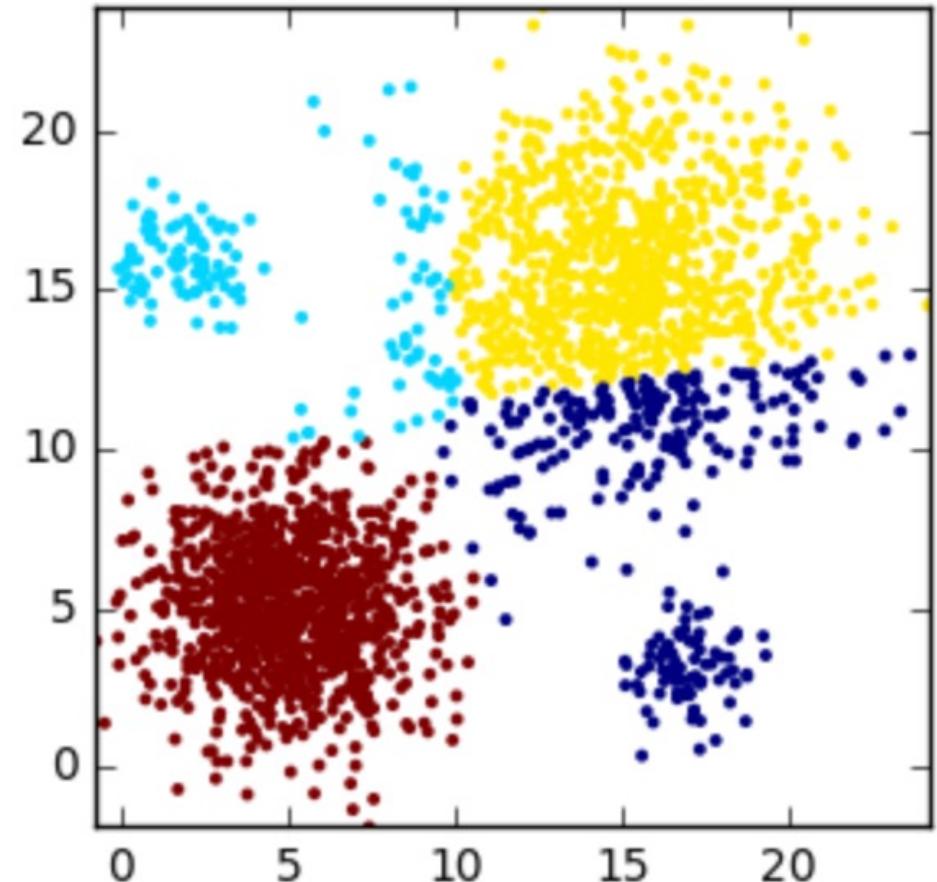
If this assumption does not hold
K-means **may perform poorly**

Note that: if we change the coordinate system, K-means may still perform OK



- With this dataset **K-means fails (miserably)**
- The problem is the distance: Euclidean assumes clusters are balls, with mean in the center, and points around it

Yet another dataset



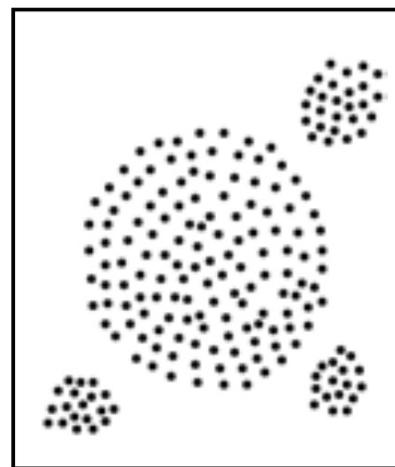
- K-means may also underperform
 - With clusters with different size and density

Reference papers

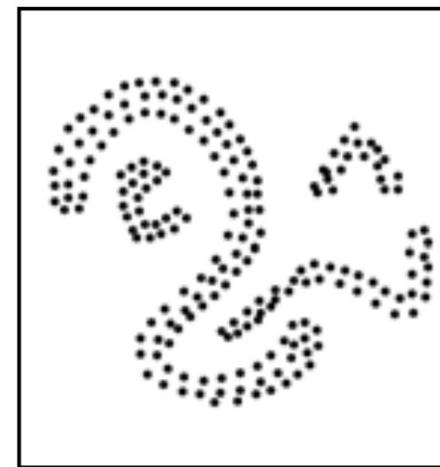
[Ester96] Martin Ester, Hans-Peter Kriegel, Joerg Sander, Xiaowei Xu, [A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise](#), 2nd International Conference on Knowledge Discovery and Data Mining (KDD), 1996. [38477 citations, Oct 2025]

[Shubert] Eric Shubert, Joerg Sander, Martin Ester, Hans Peter Kriegel, Xiaowei Xu, [DBSCAN Revisited, Revisited: Why and How You Should \(Still\) Use DBSCAN](#), ACM Transactions on Database Systems, No. 19, July 2017. [3420 citations, Oct 2025]

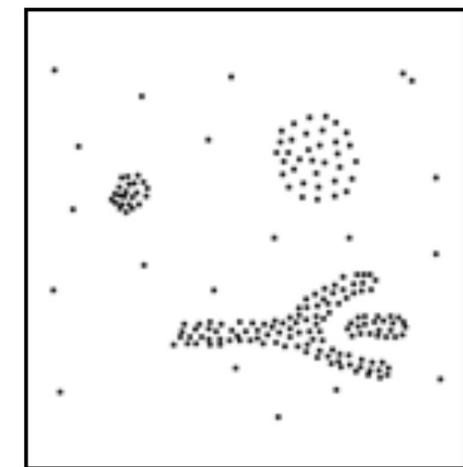
Density based notion of clusters



database 1



database 2



database 3

- We, as humans, easily recognize the clusters
- Main reasons
 - Density of points is considerably *higher* than outside the clusters
 - Density within the areas of noise is *lower* than that of any cluster

DBSCAN: pars & Eps-neighborhood

- Uses 2 parameters: (**Eps**, **MinPts**)
- **Eps**: radius of the **neighborhood**
- **MinPts**: min. no. of points in an Eps-neighborhood
- **Eps-Neighborhood** of point p

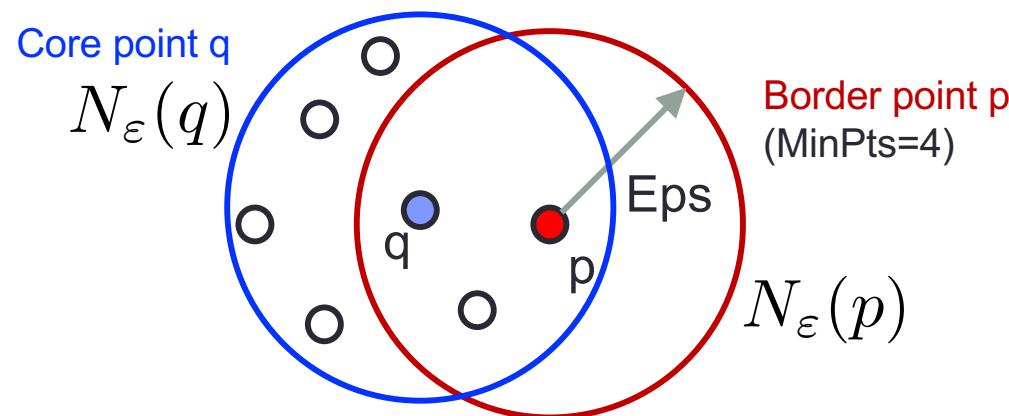
$$N_{\varepsilon}(p) = \{q \in \text{dataset} \mid \text{dist}(p, q) \leq \epsilon\}$$

- **Remarks**
 - Any distance function can be used
 - Points p,q can be vectors in a space of any number of dimensions

DBSCAN: review of features

- Single scan of data points
- Clustering based on density (local criterion)
 - Region around a point: within a distance **Eps** (any distance metric)
 - Density of a region: more than **MinPts** within a region
 - Core point
 - whose “Eps-neighborhood” contains at least **MinPts** points
 - Border point
 - fewer than **MinPts** within Eps-neighborhood
 - but **density reachable** from a core point
 - Two points **belong to the same cluster**
 - if connected by a dense region of points (see later)
- **DBSCAN handles noise**
 - Points that are not enrolled in any valid cluster → treated as noise

Def. directly density reachable

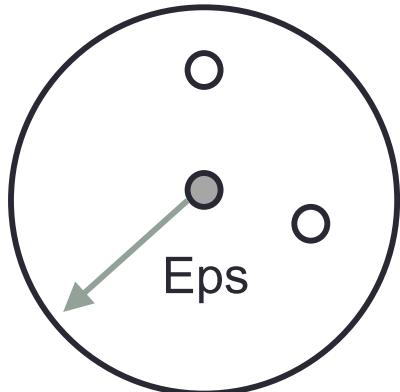


- p is directly density reachable from q wrt $(\text{Eps}, \text{MinPts})$ IF
 1. p is within Eps-neighborhood of q , $N_\varepsilon(q)$
 2. q is a core point, i.e., $|N_\varepsilon(q)| \geq \text{MinPts}$

NOTE: this relationship is **asymmetric**

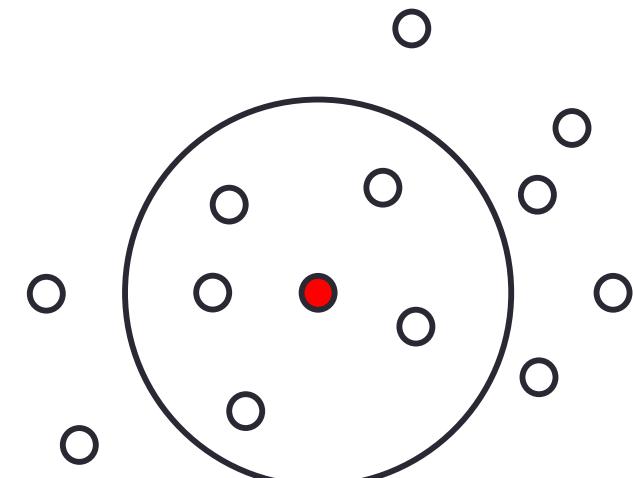
- in the above figure, q is not density reachable from p , as p is not a core point (does not meet the local density criterion)

Summary of point types



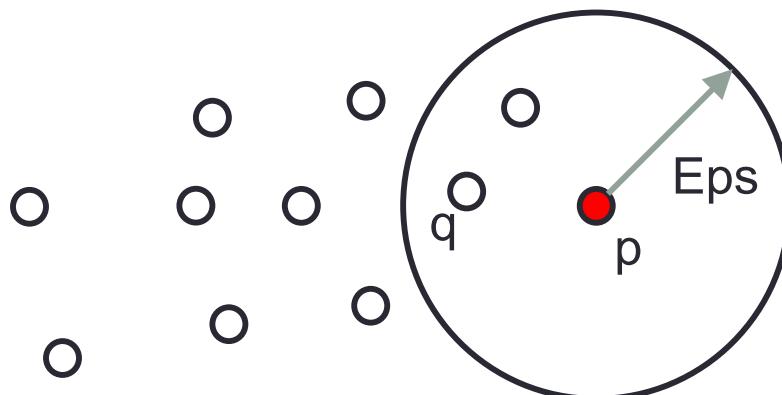
Noise point (MinPts=4)

1. has fewer than MinPts within Eps
2. no neighbor is a core point



Core point (MinPts=4)

1. has more than MinPts within Eps



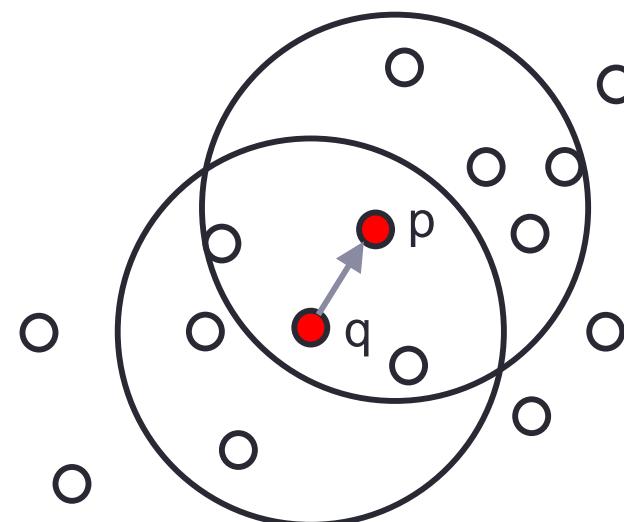
Border point p (MinPts=4)

1. has fewer than MinPts within Eps
2. at least **one core point** (e.g., q) within Eps

Density reachability example (direct)

MinPts=4

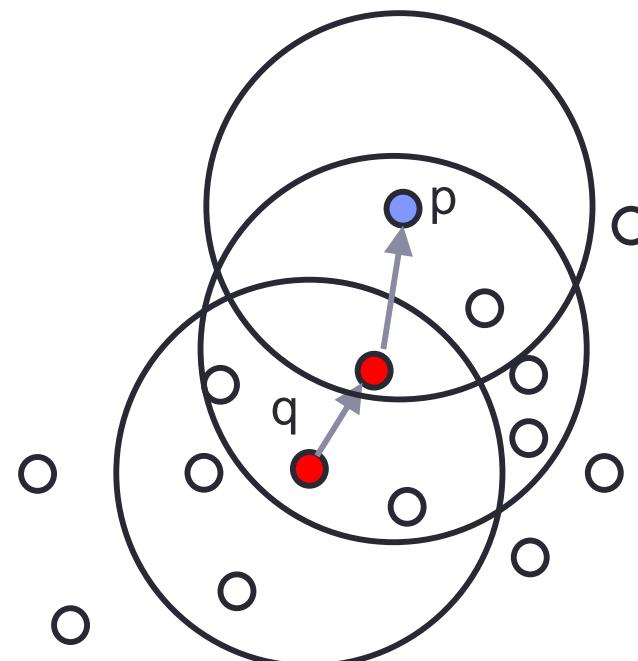
- The two **red points** are within reach and density connected
- They are both **core points** as
 - they have more than MinPts within their Eps-beighborhood
- **p** is density reachable from **q** and vice versa
- In this case the relation holds in both ways ($p \rightarrow q, q \rightarrow p$)
→ **not true in general**



Density reachability example (path)

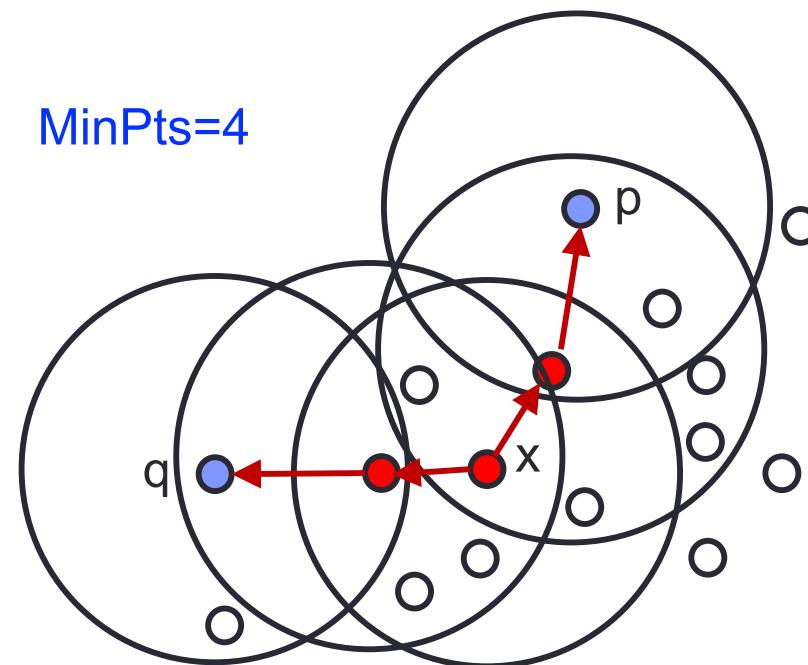
MinPts=4

- point **p** is density reachable from point **q** (via a multi-hop relationship)
- **q** is not density reachable from **p**, as **p** in this case is not a core point
- density reachability is in general **asymmetric**



Density connected points

- Any two points are **density connected**: if there exists another point x such that they are both density reachable from x
- Example in the figure: p and q are border points
 - are density connected to each other by point x
 - the red points are all core points



DBSCAN: main loop

- **SetOfPoints**: all the points in the dataset
- **Eps, MinPts**: DBSCAN parameters

```
DBSCAN (SetOfPoints, Eps, MinPts)  
  
// SetOfPoints is UNCLASSIFIED  
ClusterId := nextId(NOISE);  
FOR i FROM 1 TO SetOfPoints.size DO  
    Point := SetOfPoints.get(i);  
    IF Point.ClId = UNCLASSIFIED THEN  
        IF ExpandCluster(SetOfPoints, Point,  
                          ClusterId, Eps, MinPts) THEN  
            ClusterId := nextId(ClusterId)  
        END IF  
    END IF  
END FOR  
END; // DBSCAN
```

Try to **expand a cluster** of density connected points from each point in the dataset

Once the cluster has been computed for a point i, switch to the next point, advancing the Cluster ID counter

If a point was already assigned a ClusterID (included in a cluster) in a previous expand action → skip it

DBSCAN: ExpandCluster

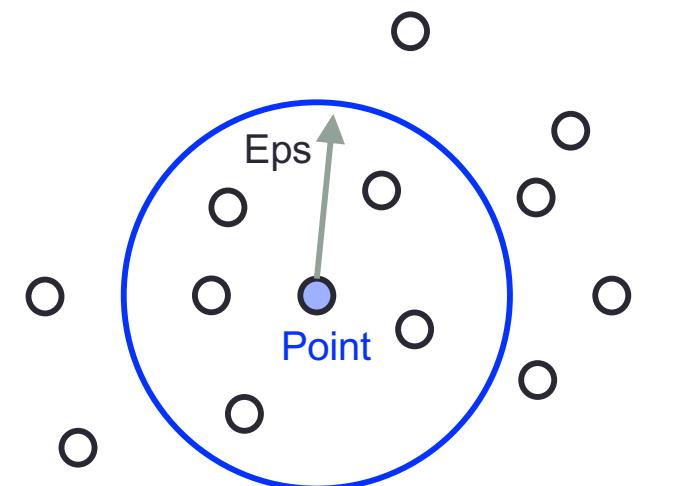
```
ExpandCluster(SetOfPoints, Point, C1Id, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeC1Id(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
        // reachable from Point
    SetOfPoints.changeC1Ids(seeds,C1Id);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                         Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.C1Id
            IN {UNCLASSIFIED, NOISE} THEN
              IF resultP.C1Id = UNCLASSIFIED THEN
                seeds.append(resultP);
              END IF;
              SetOfPoints.changeC1Id(resultP,C1Id);
            END IF; // UNCLASSIFIED or NOISE
          END IF;
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

It starts with:

- point **Point**, which is **UNCLASSIFIED**

seeds = **SetOfPoints.regionQuery()**

- **seeds**: a set containing all the points that are within distance **Eps** (the so called “**Eps**-neighborhood” of **Point**)



seeds → **Eps**-neighborhood
MinPts=4

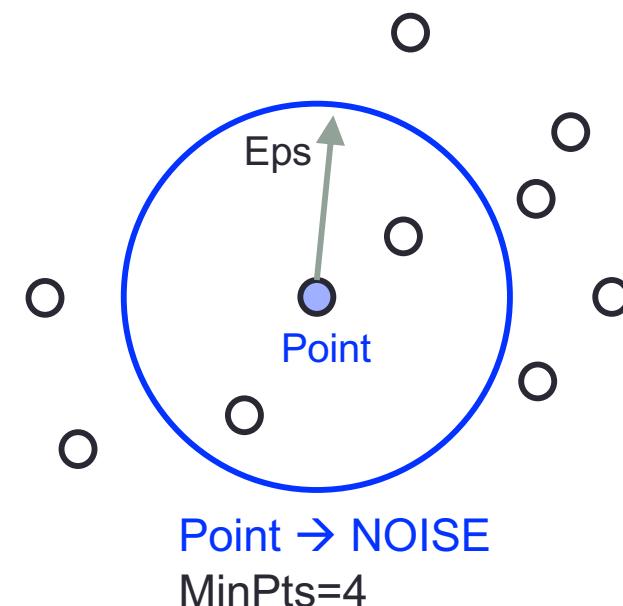
DBSCAN: ExpandCluster

```
ExpandCluster(SetOfPoints, Point, CId, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeCId(Point,NOISE);
    RETURN False;
  ELSE // all points in seeds are density-
    // reachable from Point
    SetOfPoints.changeCIds(seeds,CId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                         Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.CId
            IN {UNCLASSIFIED, NOISE} THEN
              IF resultP.CId = UNCLASSIFIED THEN
                seeds.append(resultP);
              END IF;
              SetOfPoints.changeCId(resultP,CId);
            END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

If **seeds** contains fewer than **MinPts** →
Point is marked as **NOISE**

Point cannot be a core point

NOTE: the **NOISE** label can be changed
at a later stage, if **Point** is density
reachable from some other (core) point in
the dataset



DBSCAN: ExpandCluster

```
ExpandCluster(SetOfPoints, Point, CllId, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeCllId(Point,NOISE);
    RETURN False;
  ELSE    // all points in seeds are density-
         // reachable from Point
    SetOfPoints.changeCllIds(seeds,CllId);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                         Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.CllId
            IN {UNCLASSIFIED, NOISE} THEN
              IF resultP.CllId = UNCLASSIFIED THEN
                seeds.append(resultP);
              END IF;
              SetOfPoints.changeCllId(resultP,CllId);
            END IF; // UNCLASSIFIED or NOISE
          END FOR;
        END IF; // result.size >= MinPts
        seeds.delete(currentP);
      END WHILE; // seeds <> Empty
      RETURN True;
    END IF
END; // ExpandCluster
```

all the points in **seeds** are density-reachable from **Point** (by construction)

hence, all these points take the same cluster ID (**CllId**) of **Point**

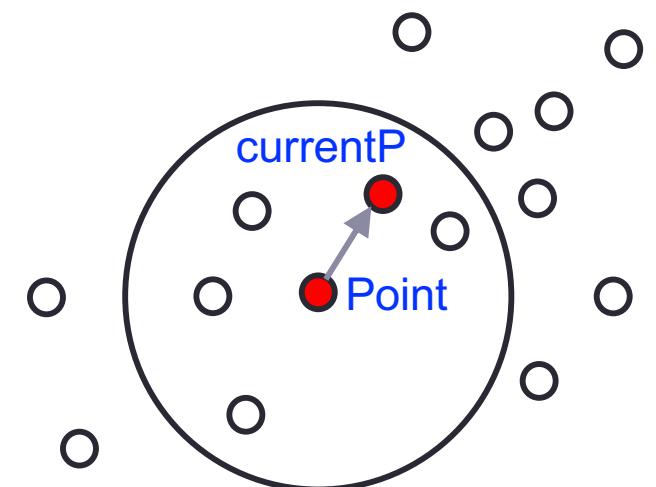
→ **SetOfPoints.changeCllIds(seeds,CllId)**

DBSCAN: ExpandCluster

```
ExpandCluster(SetOfPoints, Point, C1Id, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeC1Id(Point,NOISE);
    RETURN False;
  ELSE // all points in seeds are density-
    // reachable from Point
    SetOfPoints.changeC1Ids(seeds,C1Id);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                         Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.C1Id
            IN {UNCLASSIFIED, NOISE} THEN
              IF resultP.C1Id = UNCLASSIFIED THEN
                seeds.append(resultP);
              END IF;
              SetOfPoints.changeC1Id(resultP,C1Id);
            END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

remove **Point** from the **seeds** set

scan all the remaining points in the **seeds** set → **currentP** is the first of such points



DBSCAN: ExpandCluster

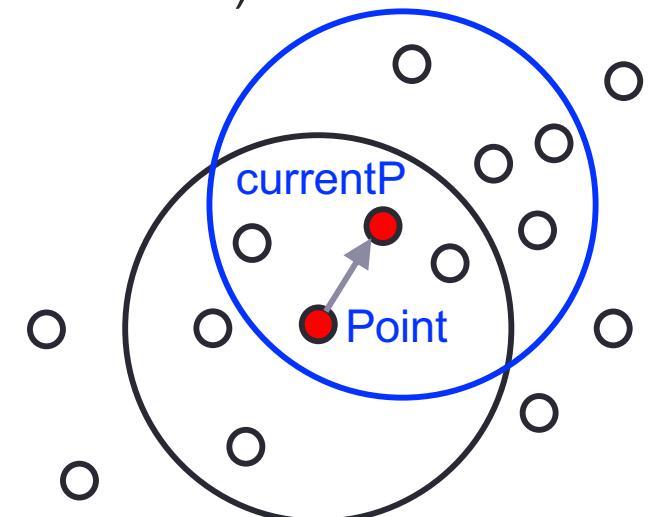
```
ExpandCluster(SetOfPoints, Point, C1Id, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeC1Id(Point,NOISE);
    RETURN False;
  ELSE // all points in seeds are density-
    // reachable from Point
    SetOfPoints.changeC1Ids(seeds,C1Id);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                         Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.C1Id
            IN {UNCLASSIFIED, NOISE} THEN
              IF resultP.C1Id = UNCLASSIFIED THEN
                seeds.append(resultP);
              END IF;
              SetOfPoints.changeC1Id(resultP,C1Id);
            END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

`result = SetOfPoints.regionQuery()`

`result` contains all the points in the Eps-neighborhood of `currentP`

we are scanning the second order Eps-neighborhood of `Point`

if `result` contains more than `MinPts` →
continue, this set contains points that
are all density reachable from `currentP`
(and, in turn, from `Point`)



DBSCAN: ExpandCluster

```
ExpandCluster(SetOfPoints, Point, C1Id, Eps,
              MinPts) : Boolean;
  seeds:=SetOfPoints.regionQuery(Point,Eps);
  IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeC1Id(Point,NOISE);
    RETURN False;
  ELSE   // all points in seeds are density-
        // reachable from Point
    SetOfPoints.changeC1Ids(seeds,C1Id);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
      currentP := seeds.first();
      result := SetOfPoints.regionQuery(currentP,
                                         Eps);
      IF result.size >= MinPts THEN
        FOR i FROM 1 TO result.size DO
          resultP := result.get(i);
          IF resultP.C1Id
            IN {UNCLASSIFIED, NOISE} THEN
              IF resultP.C1Id = UNCLASSIFIED THEN
                seeds.append(resultP);
              END IF;
              SetOfPoints.changeC1Id(resultP,C1Id);
            END IF; // UNCLASSIFIED or NOISE
        END FOR;
      END IF; // result.size >= MinPts
      seeds.delete(currentP);
    END WHILE; // seeds <> Empty
    RETURN True;
  END IF
END; // ExpandCluster
```

FOR each point in set **result**

If **UNCLASSIFIED**

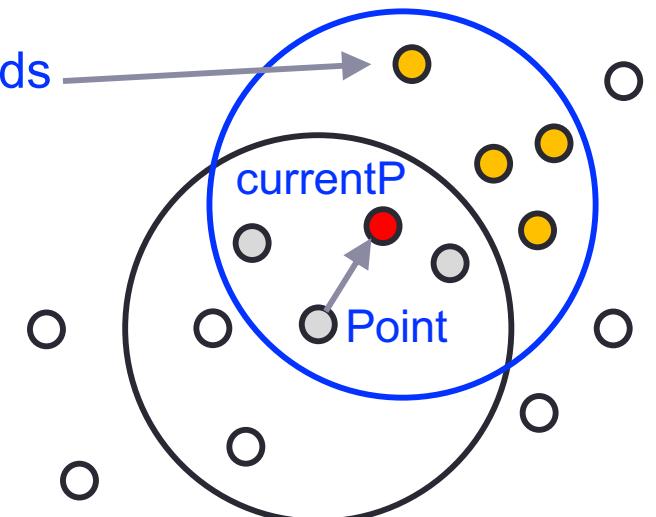
→ add to set **seeds**

If **UNCLASSIFIED** or **NOISE**

→ change cluster ID with current **C1Id**

NOTE: if the point is marked as **NOISE**
we do not re-add it to seeds, as it was
already processed, we only change its
cluster ID

added to **seeds**



DBSCAN: ExpandCluster

```
ExpandCluster(SetOfPoints, Point, C1Id, Eps,
              MinPts) : Boolean;
seeds:=SetOfPoints.regionQuery(Point,Eps);
IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeC1Id(Point,NOISE);
    RETURN False;
ELSE    // all points in seeds are density-
        // reachable from Point
    SetOfPoints.changeC1Ids(seeds,C1Id);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
        currentP := seeds.first();
        result := SetOfPoints.regionQuery(currentP,
                                            Eps);
        IF result.size >= MinPts THEN
            FOR i FROM 1 TO result.size DO
                resultP := result.get(i);
                IF resultP.C1Id
                    IN {UNCLASSIFIED, NOISE} THEN
                        IF resultP.C1Id = UNCLASSIFIED THEN
                            seeds.append(resultP);
                        END IF;
                        SetOfPoints.changeC1Id(resultP,C1Id);
                    END IF; // UNCLASSIFIED or NOISE
                END FOR;
            END IF; // result.size >= MinPts
            seeds.delete(currentP);
        END WHILE; // seeds <> Empty
        RETURN True;
    END IF
END; // ExpandCluster
```

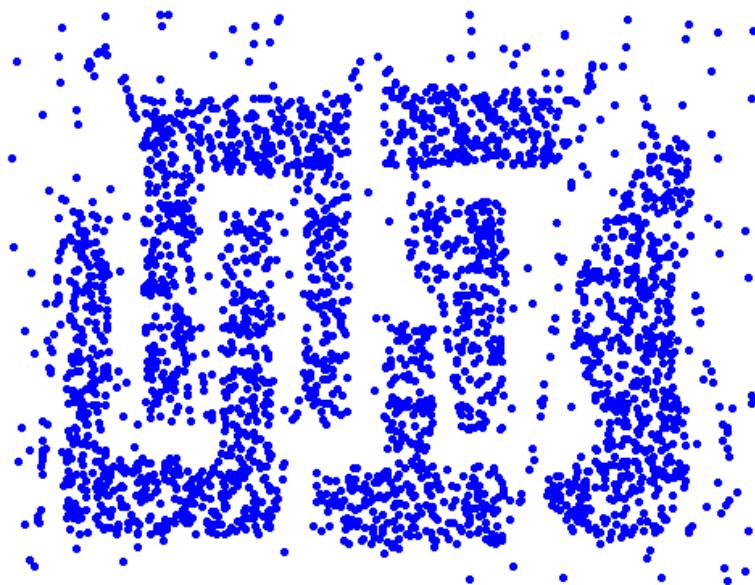
NOTE: adding points to the **seeds** set reiterates the search by reaching all the points that are **density connected through a path** → assigning them the same cluster ID of the starting **Point**

DBSCAN: ExpandCluster

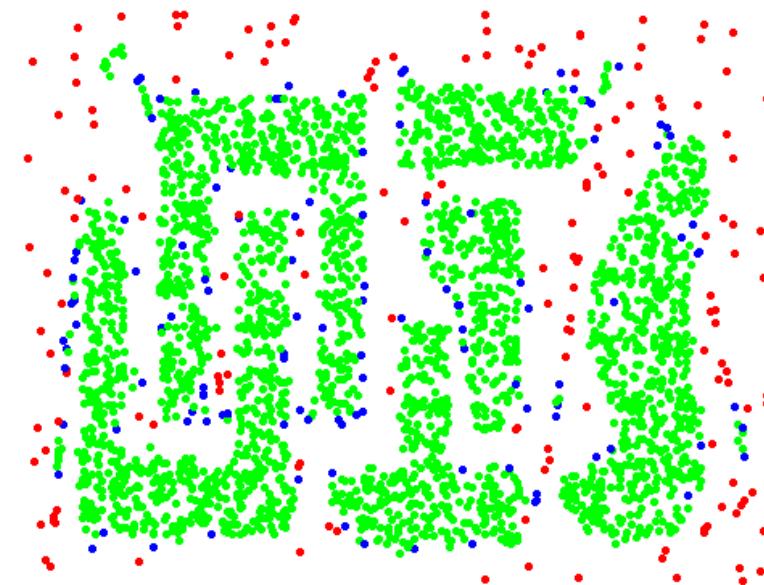
```
ExpandCluster(SetOfPoints, Point, C1Id, Eps,
              MinPts) : Boolean;
seeds:=SetOfPoints.regionQuery(Point,Eps);
IF seeds.size<MinPts THEN // no core point
    SetOfPoint.changeC1Id(Point,NOISE);
    RETURN False;
ELSE    // all points in seeds are density-
    // reachable from Point
    SetOfPoints.changeC1Ids(seeds,C1Id);
    seeds.delete(Point);
    WHILE seeds <> Empty DO
        currentP := seeds.first();
        result := SetOfPoints.regionQuery(currentP,
                                           Eps);
        IF result.size >= MinPts THEN
            FOR i FROM 1 TO result.size DO
                resultP := result.get(i);
                IF resultP.C1Id
                    IN {UNCLASSIFIED, NOISE} THEN
                        IF resultP.C1Id = UNCLASSIFIED THEN
                            seeds.append(resultP);
                        END IF;
                        SetOfPoints.changeC1Id(resultP,C1Id);
                    END IF; // UNCLASSIFIED or NOISE
                END FOR;
            END IF; // result.size >= MinPts
            seeds.delete(currentP);
        END WHILE; // seeds <> Empty
        RETURN True;
    END IF
END; // ExpandCluster
```

- delete **currentP** from **seeds**
- move to next point in **seeds**
- repeat until **seeds** is empty

2D example



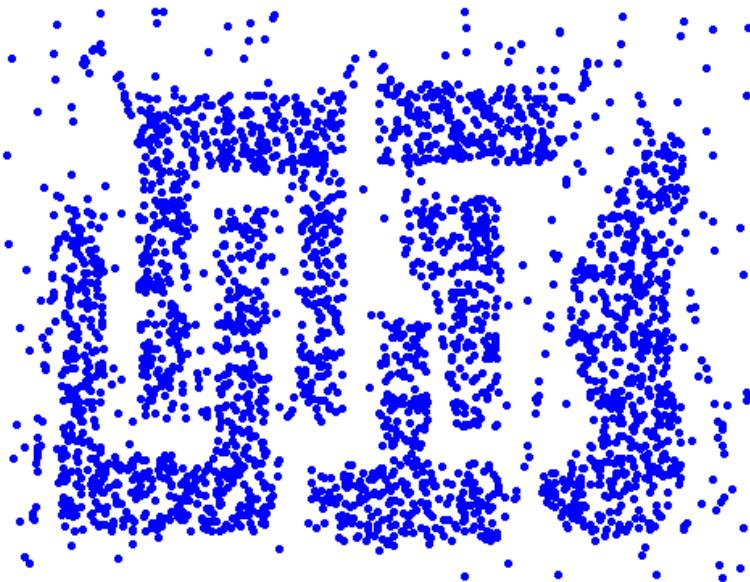
original points



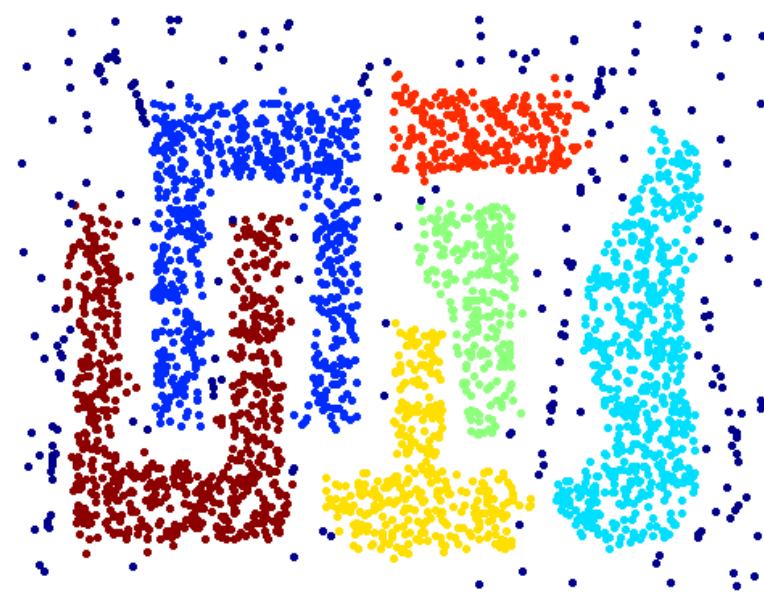
DBSCAN: **core points**, border & noise

Eps=10, MinPts=4

2D clustering result



original points



DBSCAN: clusters

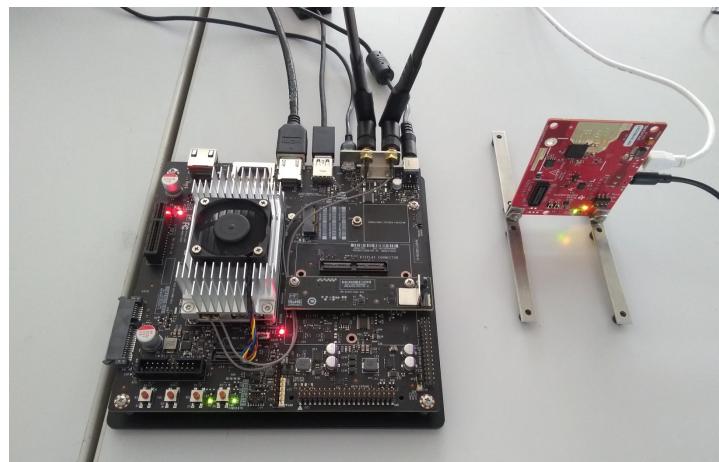
- DBSCAN works well
 - Resistant to noise
 - Handles clusters of different shape and size

DBSCAN complexity

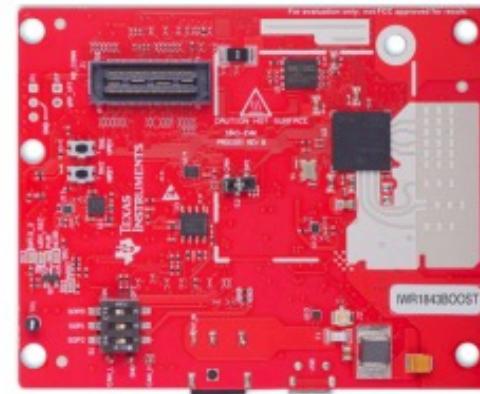
- n : number of points to be clustered
- Time complexity: $O(n^2)$ – for each point it has to be determined if it is a core point. It can be reduced to $O(n * \log(n))$ in low dimensional spaces by using efficient data structures
- Space complexity: $O(n)$

Practical example: mm-wave sensing

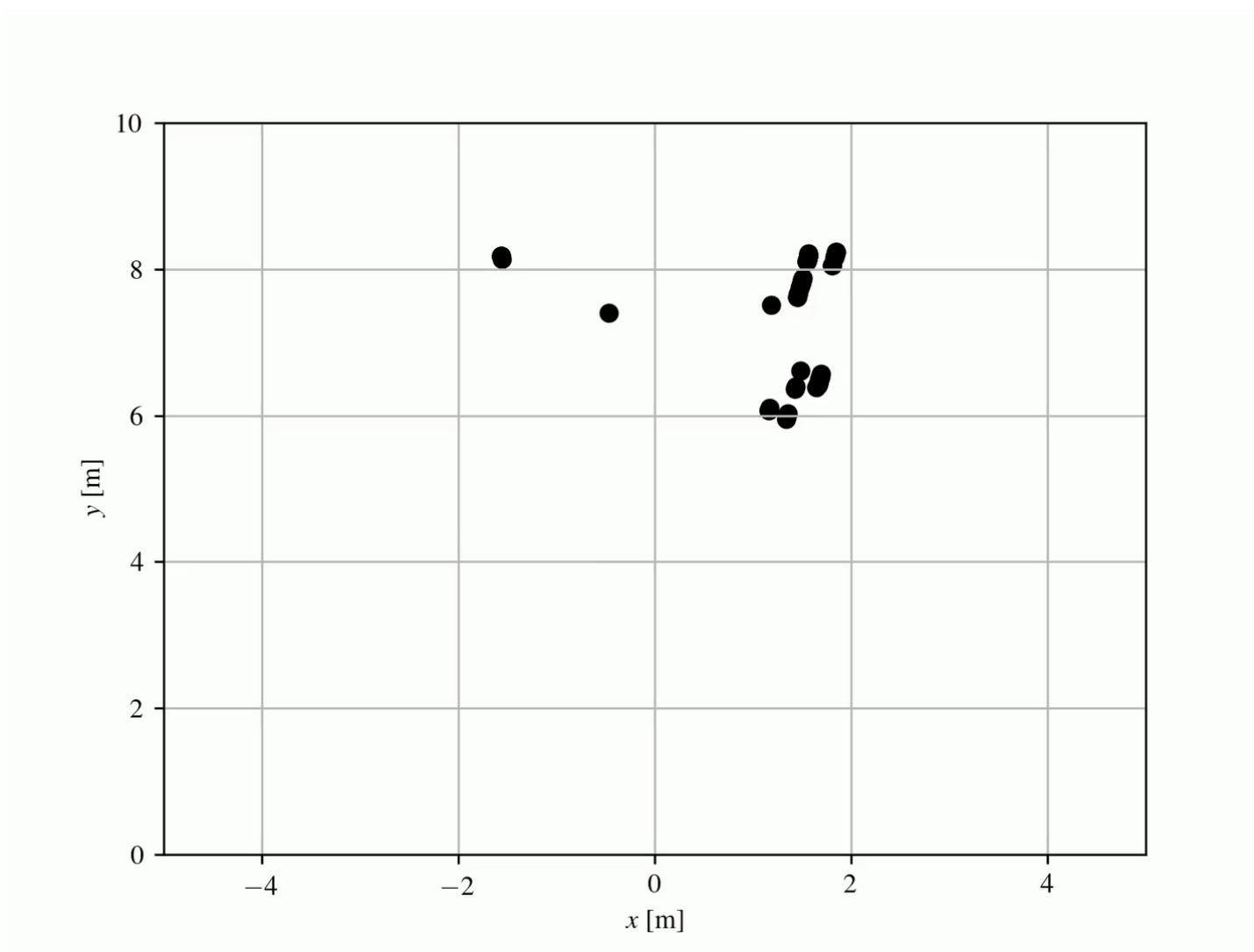
- **IWR1843** single-chip 76-GHz to 81-GHz industrial radar
 - FMCW radar, 76 to 81 GHz (4GHz bandwidth)
 - Evaluation board from Texas Instruments
 - Cortex R4F micro-controller for object tracking
 - Antennas: 3(TX), 4(RX)



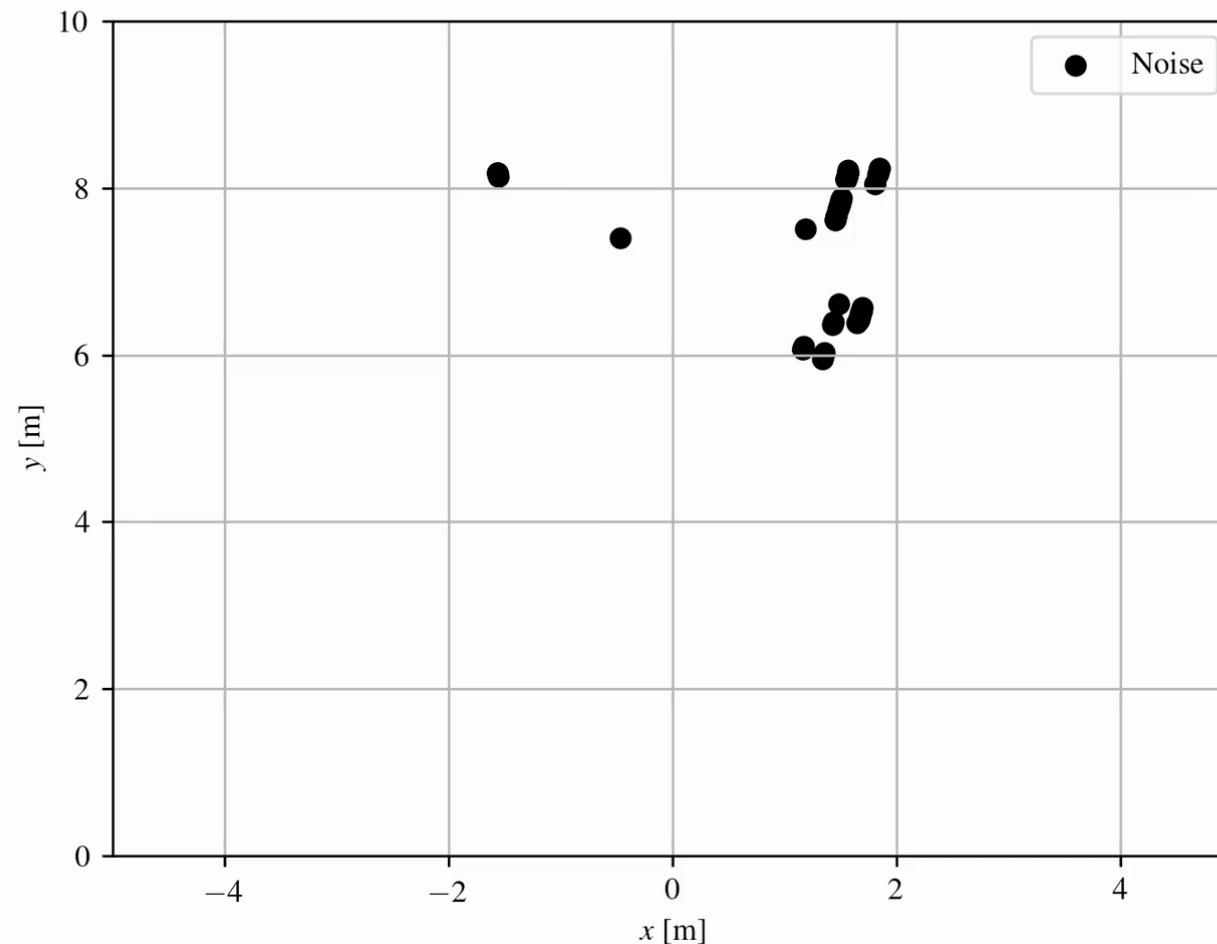
NVIDIA Edge Computer (Jetson TX2)



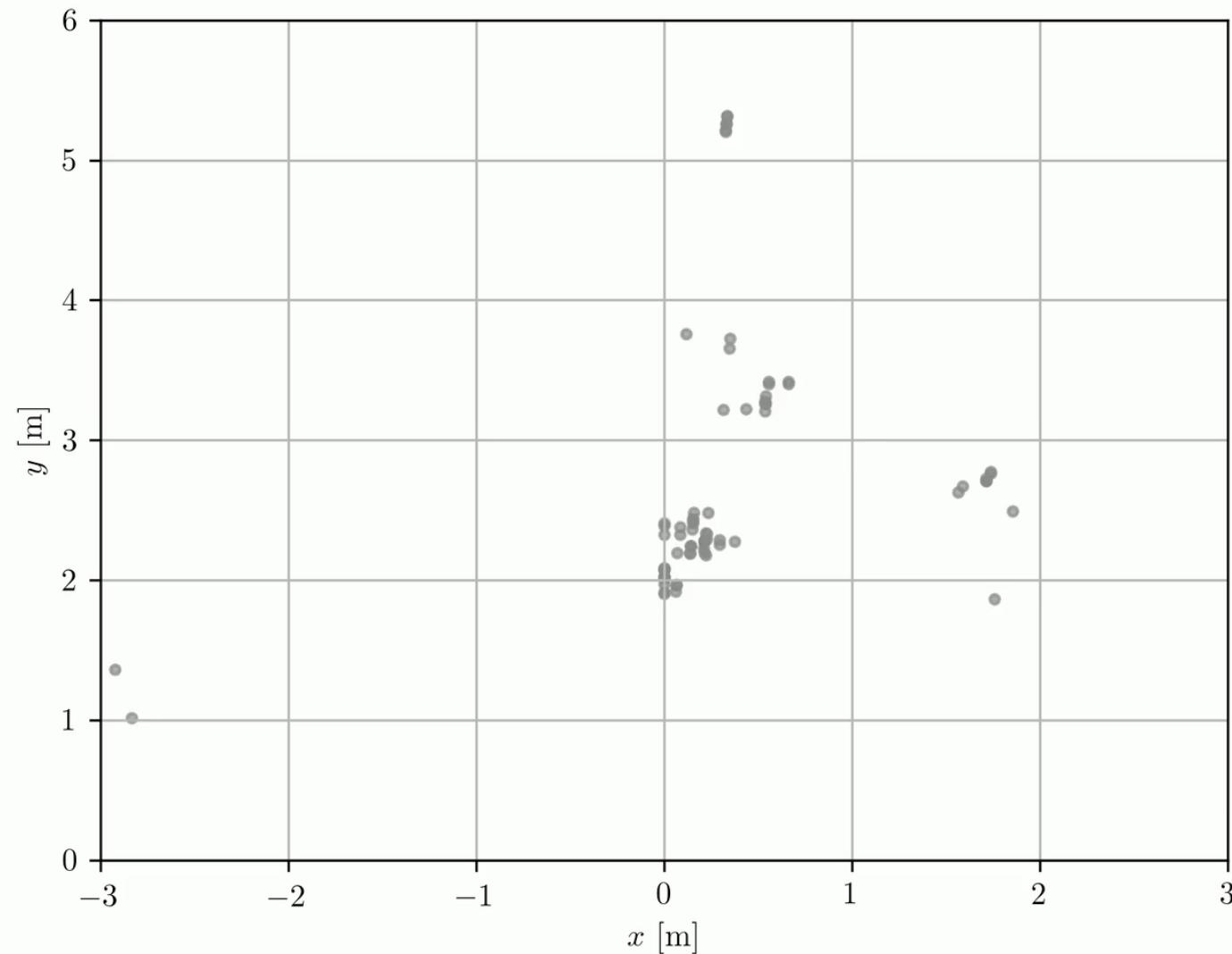
Raw data: mm-wave point cloud



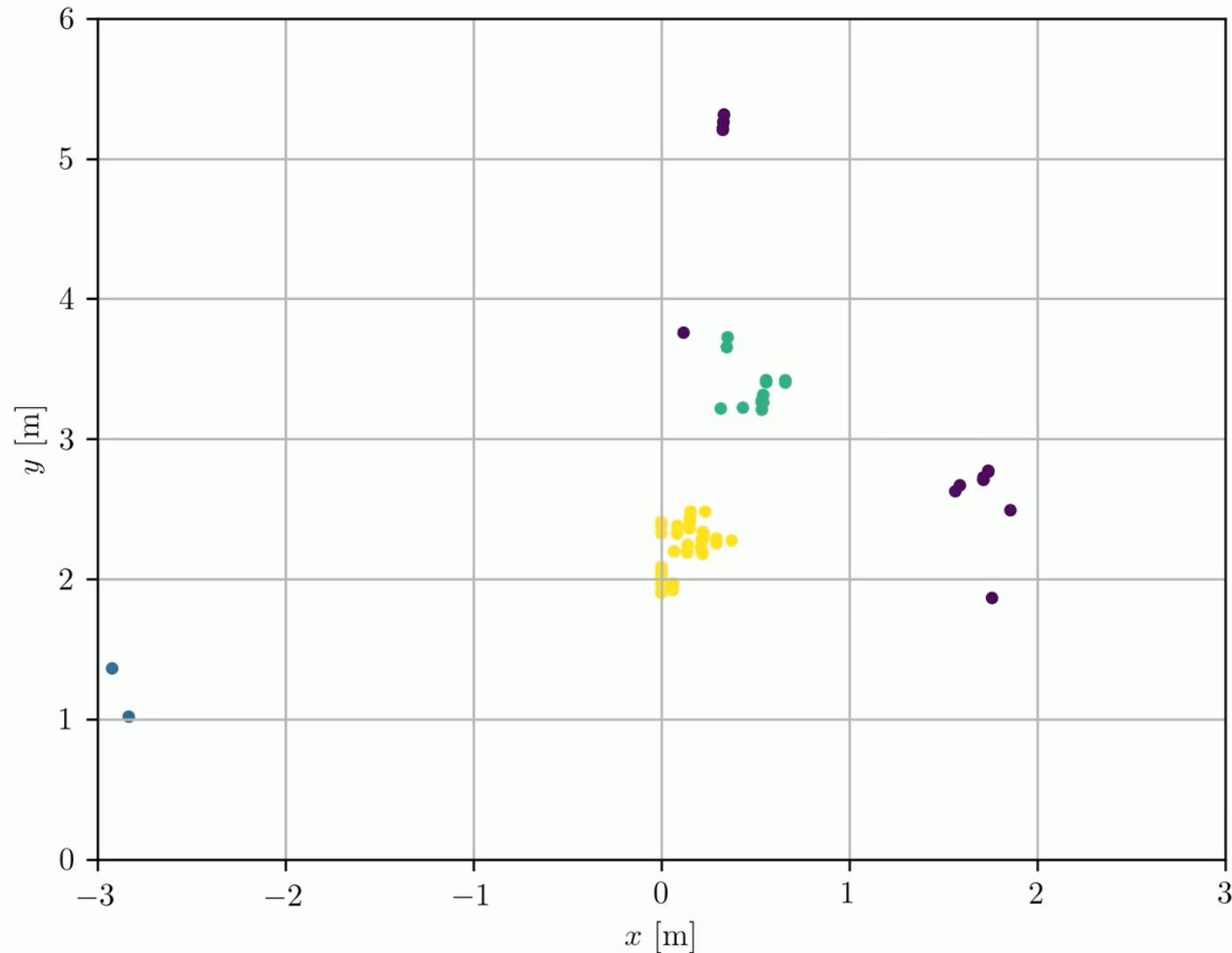
DBSCAN applied to point clouds



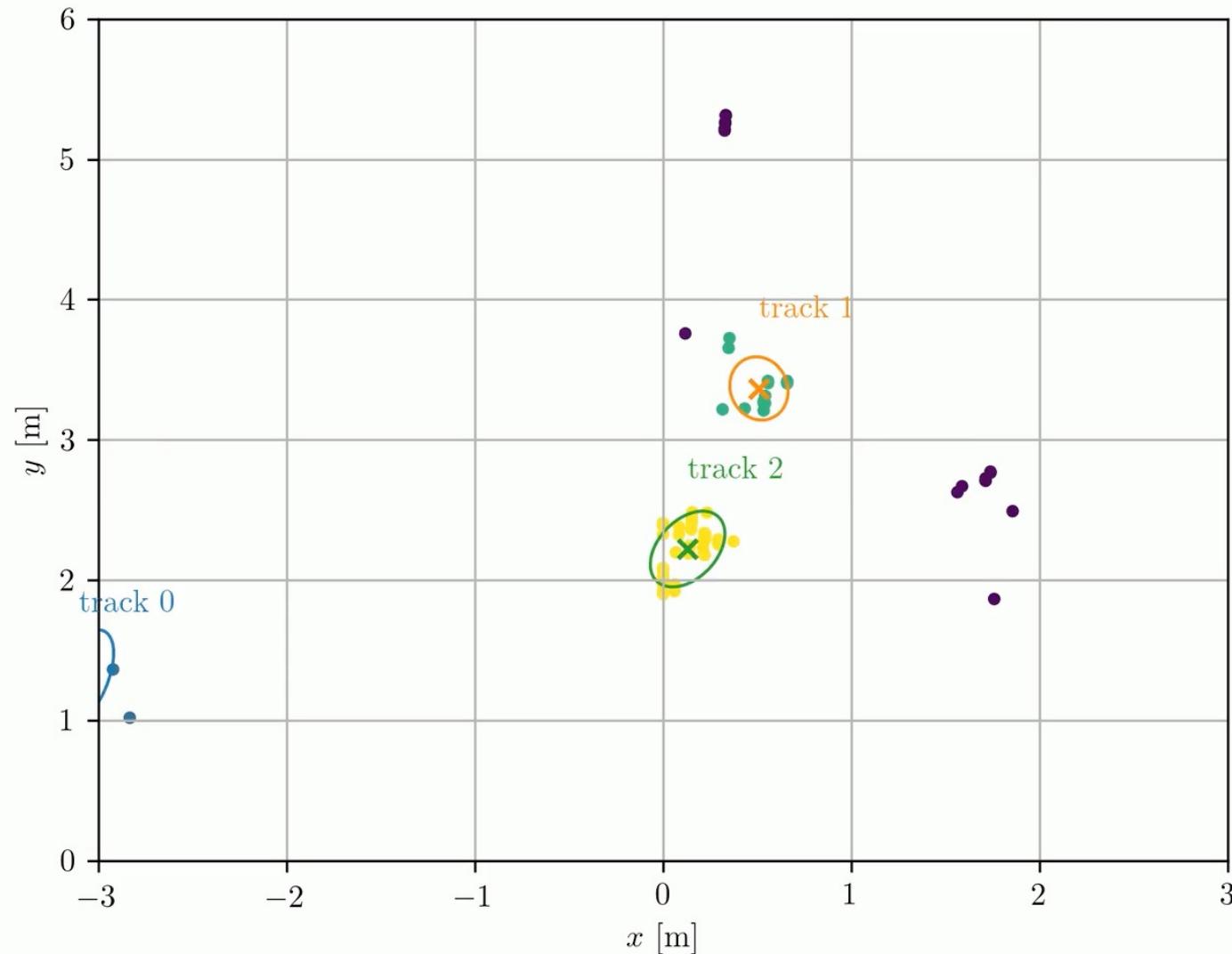
Raw point clouds from the radar



DBSCAN clusters



After some more magic...



Setting MinPts

- Is there an exact method? No, it is heuristic, and empirical...
- The larger the data set, the larger the MinPts variable should be
- If the data set is noisier, choose a larger value of MinPts
- Guidelines from experiments
 - Generally, MinPts should be greater than or equal to the dimensionality of the data set
 - For 2-dimensional data, default value of MinPts = 4 [Ester1996]
 - If your data has more than 2 dimensions, choose MinPts = 2*dim, where dim= the dimensions of your data [Sander1998]

[Sander1998] X. Xu, M. Ester; H.-P. Kriegel, J. Sander, A distribution-based clustering algorithm for mining in large spatial databases, Int. Conference on Data Engineering, February 1998.

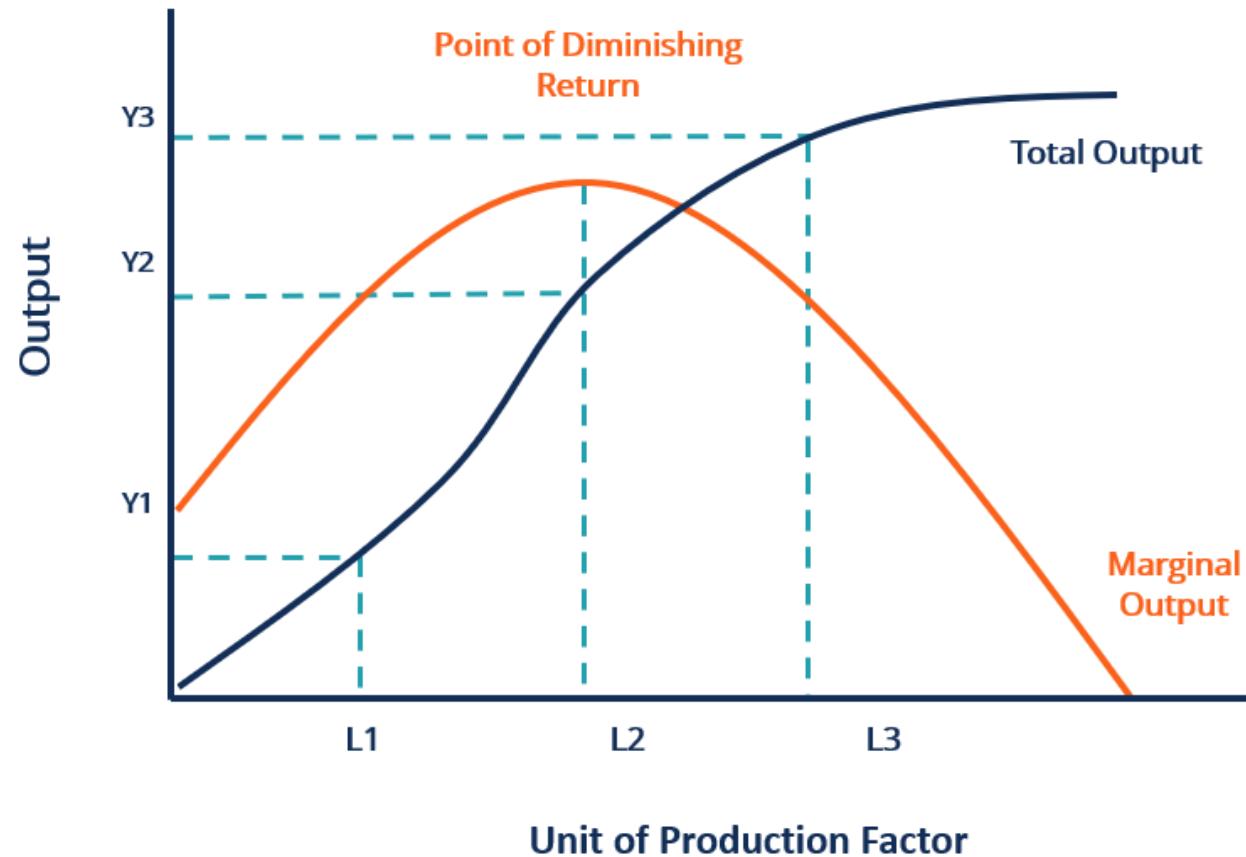
Dimishing returns (microeconomics)

Production process

- As production factor increases (e.g., workforce), the total output also increases
- At some point, however, it will reach an optimal output level, after which the output flattens or decreases
- **Production factors** = labor, machine hours, raw material

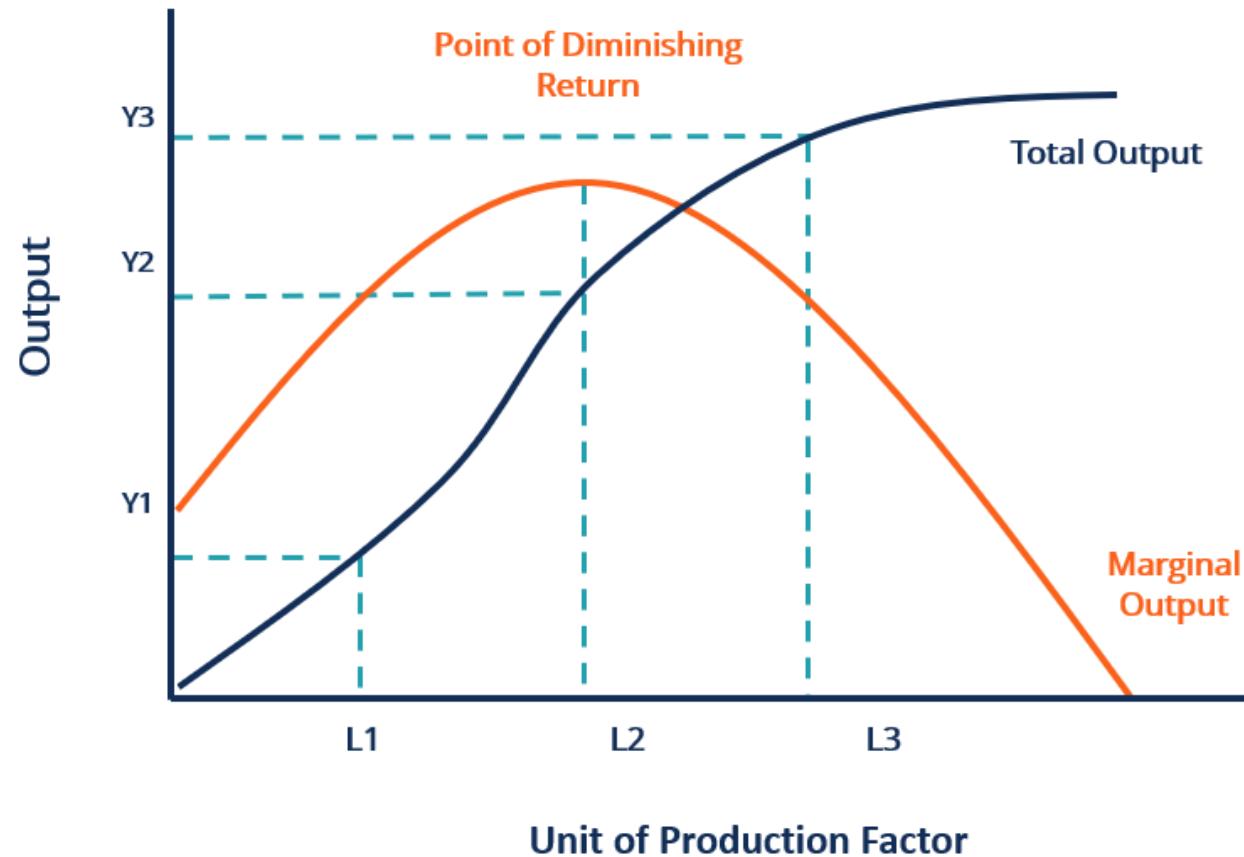
“Assuming a constant level of other production factors, every additional unit of a production factor leads to a greater increase in total output (marginal output) initially. After reaching a certain optimal production level, every additional unit of the production factor will result in a smaller increase in total output with a diminishing marginal output, as the efficiency is limited by the other production factors...”

Dimishing returns



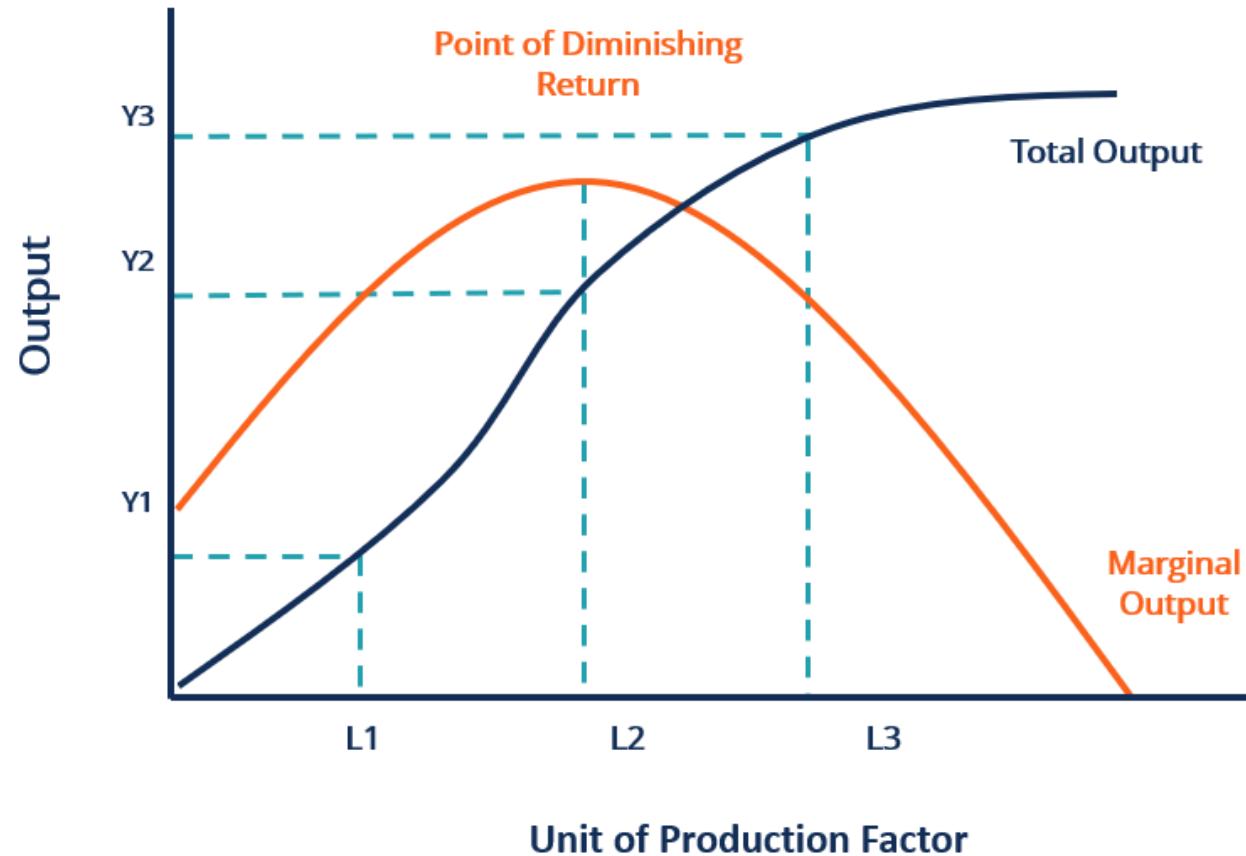
If production factor increases from $L_1 \rightarrow L_2$, the increase in the output is Y_2-Y_1 , this is called **marginal return**

Dimishing returns



If production factor increases from L₂ → L₃, the marginal return is Y₃-Y₂, **which is smaller than Y₂-Y₁**

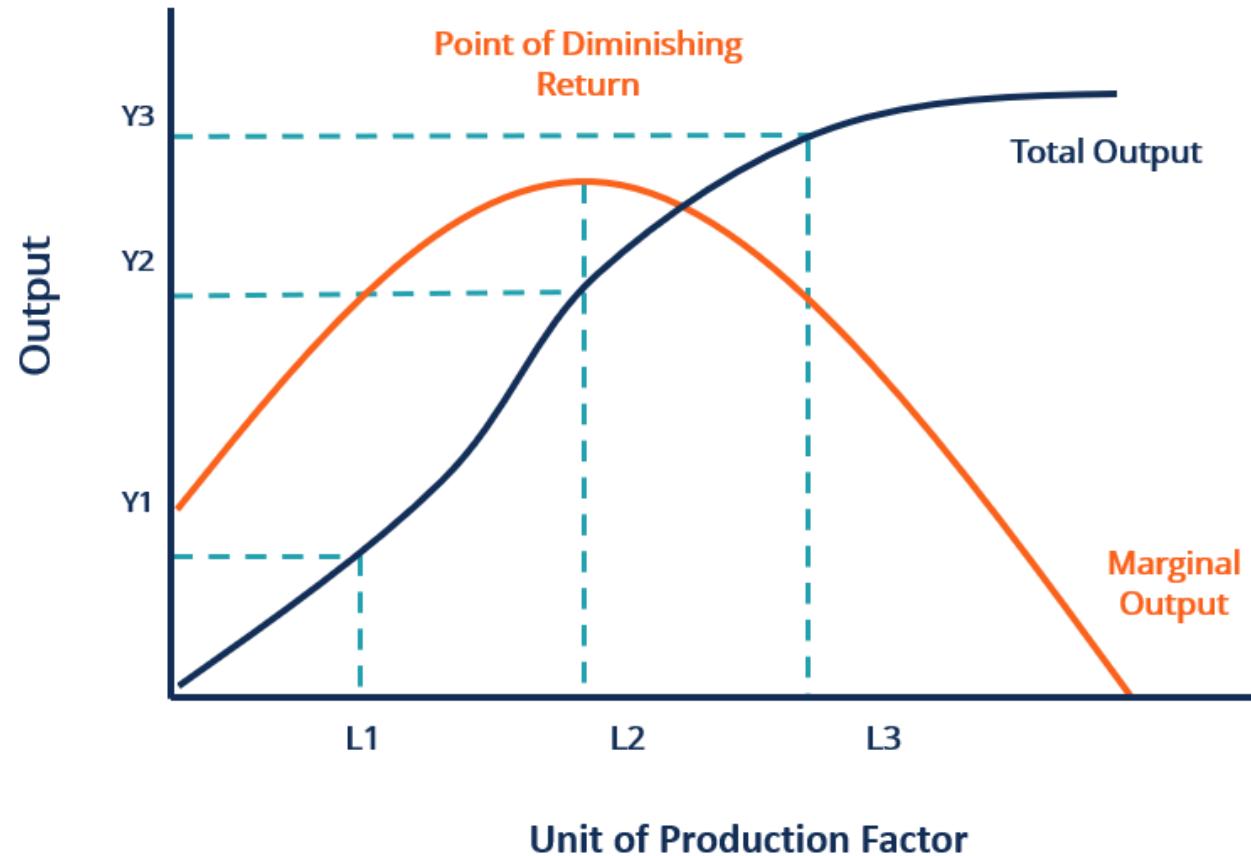
Dimishing returns



If production factor keeps increasing the marginal return eventually goes to zero

Dimishing returns

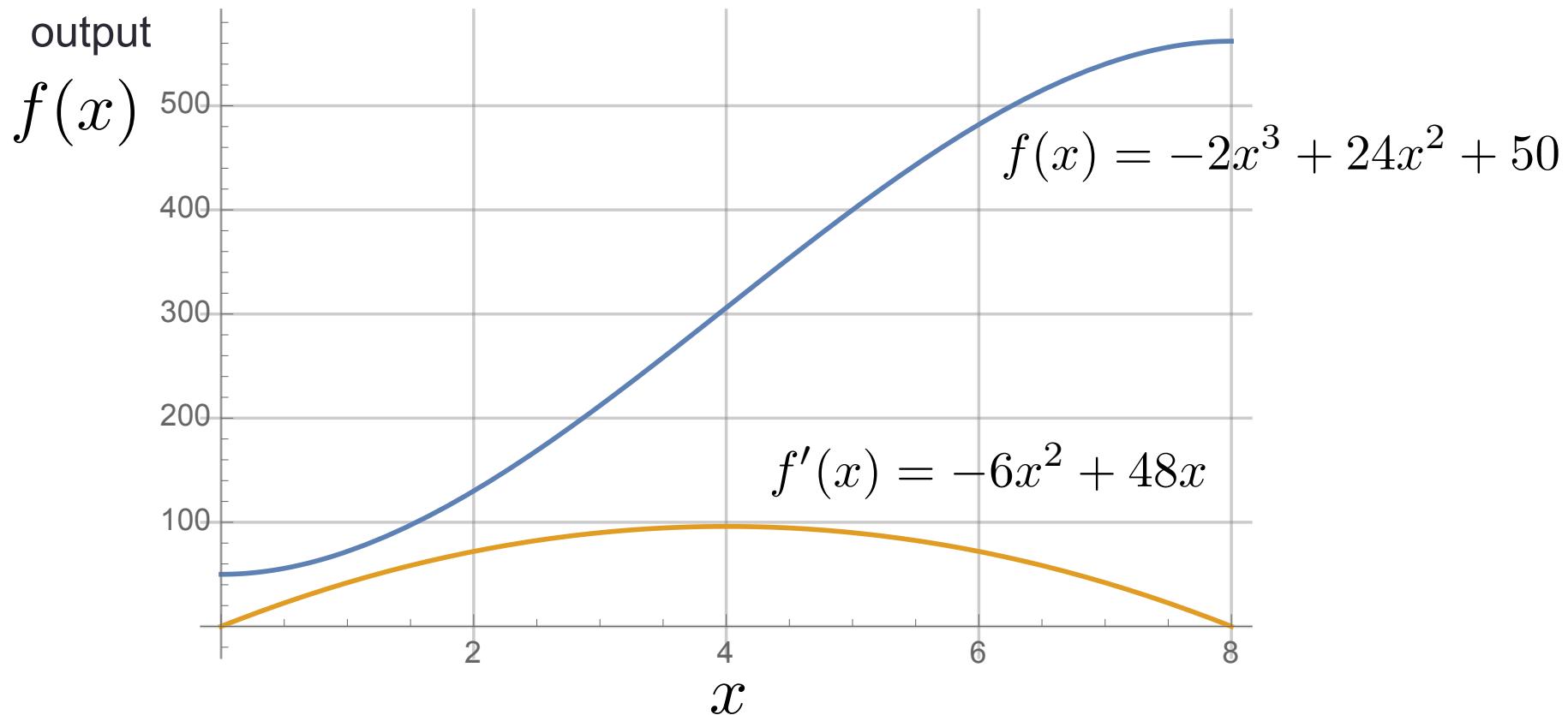
$$\text{maximize } \frac{dY}{dL}$$



The point where the marginal return is maximum is L2

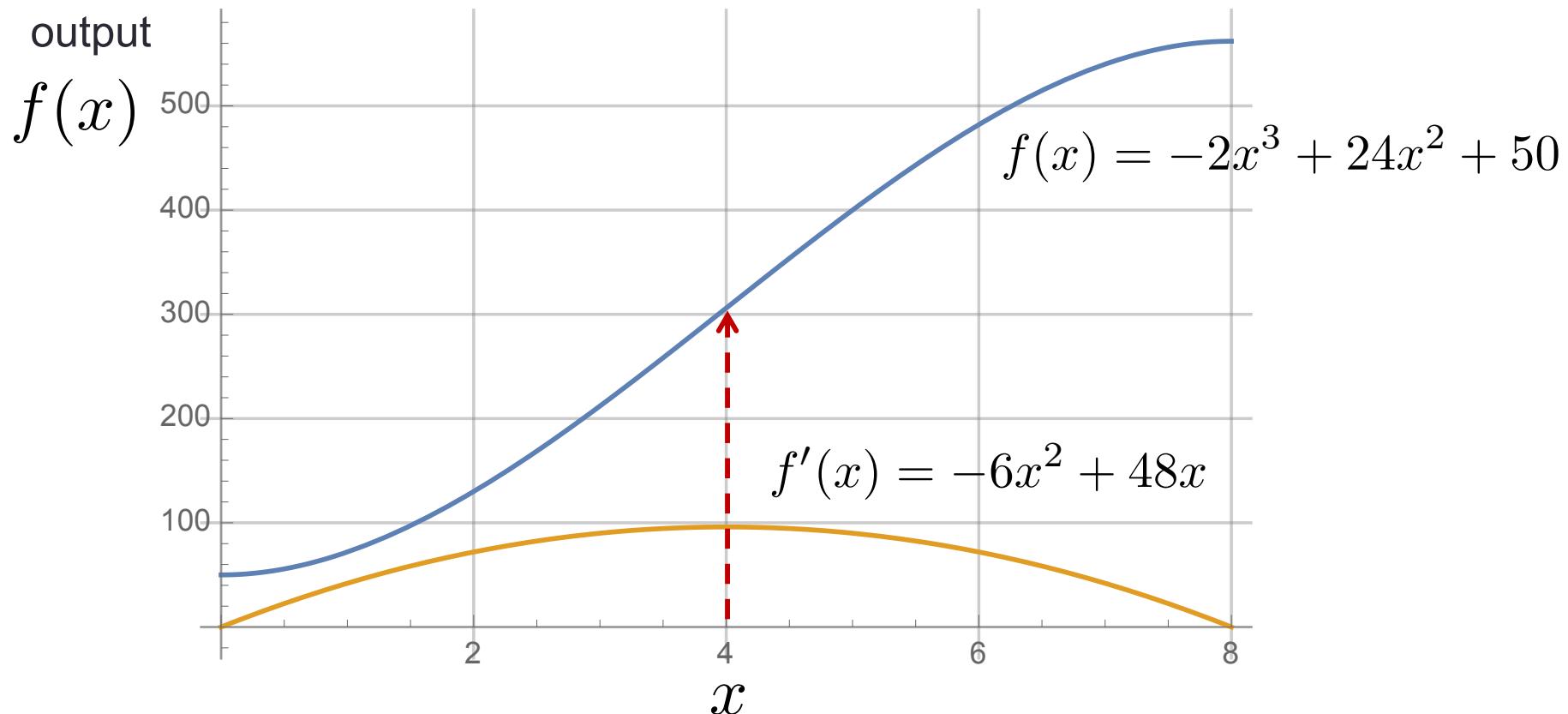
How do we find it?

Marginal returns example



We look at the first order derivative of the output as a function of the production factor

Marginal returns example

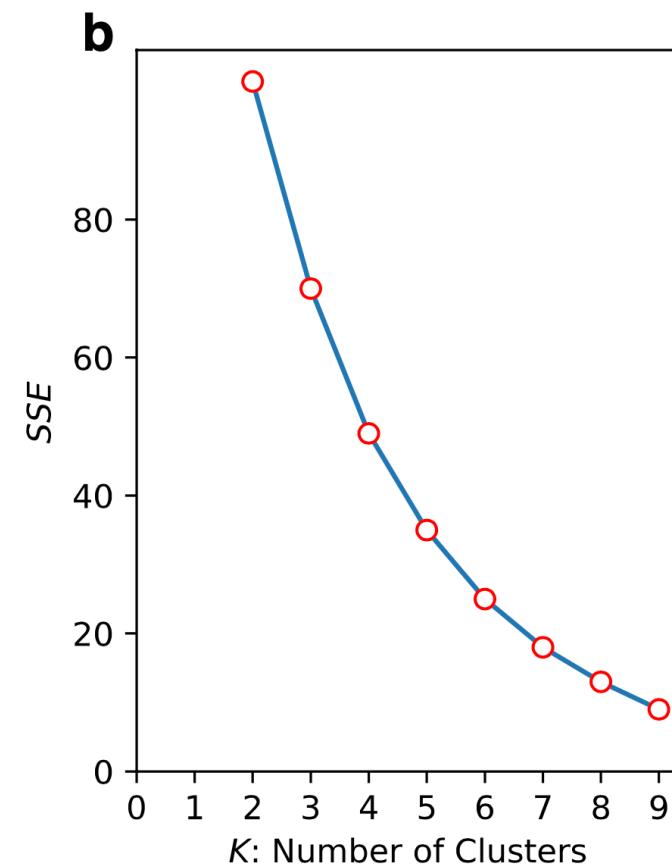
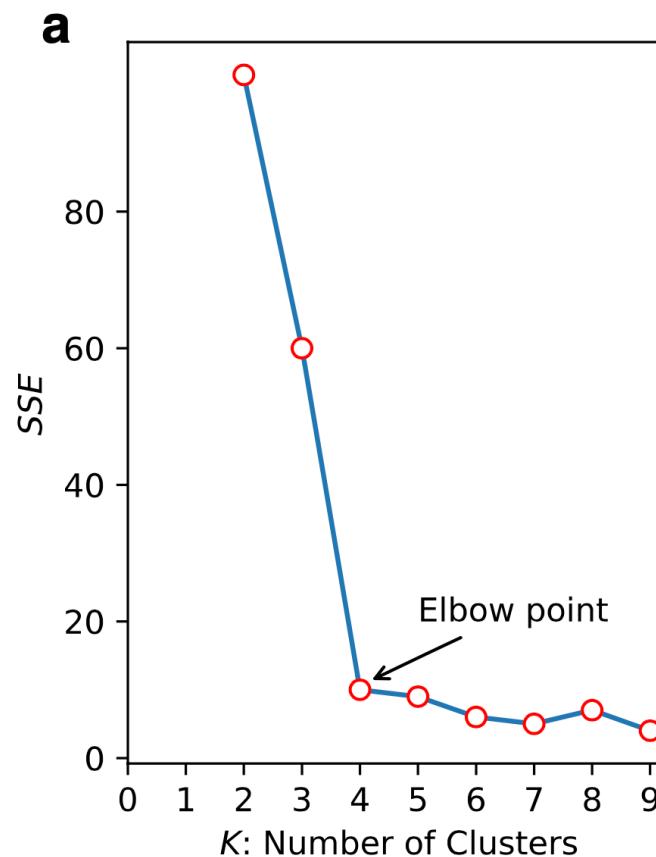


The maximum marginal return is achieved **when the first order derivative is maximum**, or when the second order derivative is zero (inflection point of $f(x)$)

Clustering – choice of K? (e.g.. K-means)

SSE = Sum of Squared Distances $SSE = \sum_{k=1}^K \sum_{\mathbf{x}_i \in \mathcal{C}_k} \|\mathbf{x}_i - \mu_i\|^2$

SSE for two datasets a & b



Knee detection

- **Problem:** “finding the inflection point can be done analytically for continuous analytical functions, for which the first (and second) order derivatives can be analytically computed. It is much harder to compute it for empirical curves obtained from discrete data points (domain of x is discrete and points are not equally spaced) – this is often the case in practice”
- **Possible solution:** “Kneedle” [Satopää2011], a heuristic algorithm that robustly works with experimental datasets

[Satopaa2011] V. Satopää, J. Albrecht, D. Irwin, B. Raghavan, A "Kneedle" in a Haystack: Detecting Knee Points in System Behavior, Int. Conference on Distributed Computing Systems, 2011. [1648 citations. October 2025]

A simple but effective algorithm

- In [Satopaa2011], the “elbow” points is detected as a point of maximum curvature of $f(x)$
- Curvature of $f(x)$, continuos with 1st & 2nd order derivatives

$$K(x) = \frac{|f''(x)|}{[1 + f'(x)^2]^{3/2}}$$

- The point of max curvature is

$$x^* = \operatorname{argmax}_x K(x)$$

[Satopaa2011] V. Satopää, J. Albrecht, D. Irwin, B. Raghavan, A "Kneedle" in a Haystack: Detecting Knee Points in System Behavior, Int. Conference on Distributed Computing Systems, 2011.

A simple but effective algorithm

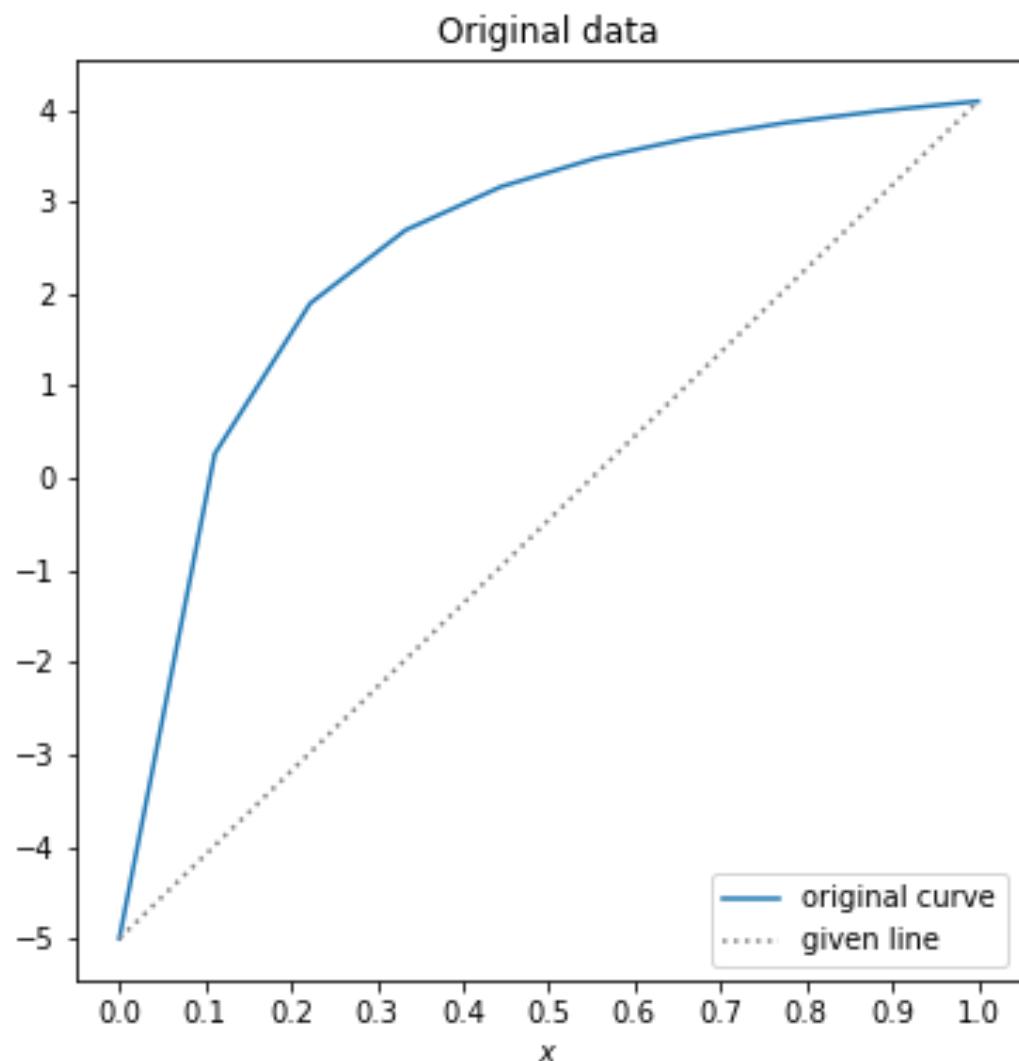
- Problem
 - Curvature **is not well-defined** for discrete (and noisy) functions
 - And it is difficult to approximate considering the analytical equation

[Satopaa2011] V. Satopää, J. Albrecht, D. Irwin, B. Raghavan, A "Kneedle" in a Haystack: Detecting Knee Points in System Behavior, Int. Conference on Distributed Computing Systems, 2011.

1. Original data

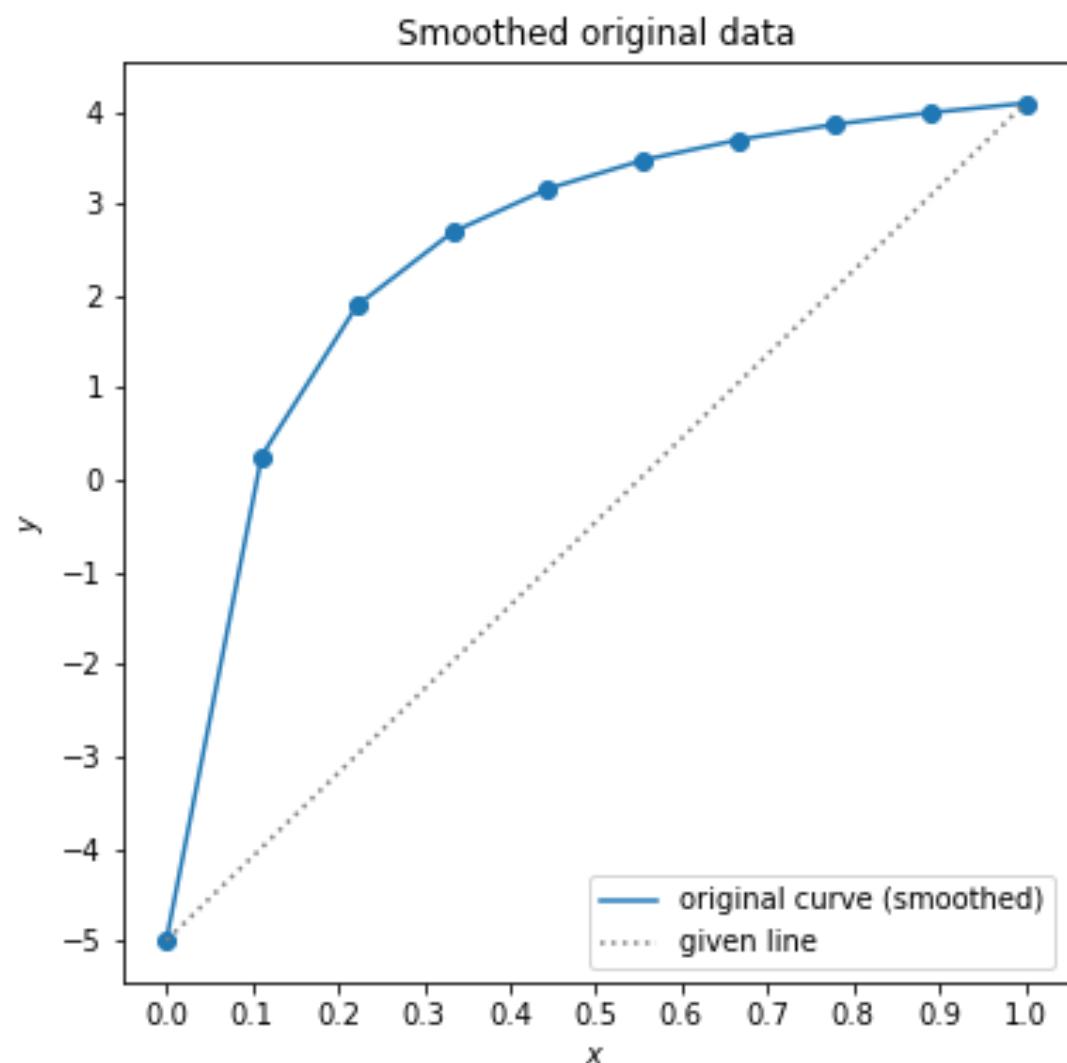
Kneedle by example

- Pragmatic approach
- Plot quality measure
 - Possibly rotated
- Can be
 - K-means: distortion J vs K
 - DBSCAN: average distance in neighborhood vs points
- ...



2. Smooth data

- Smooth data
 - e.g. using splines
(among many options)



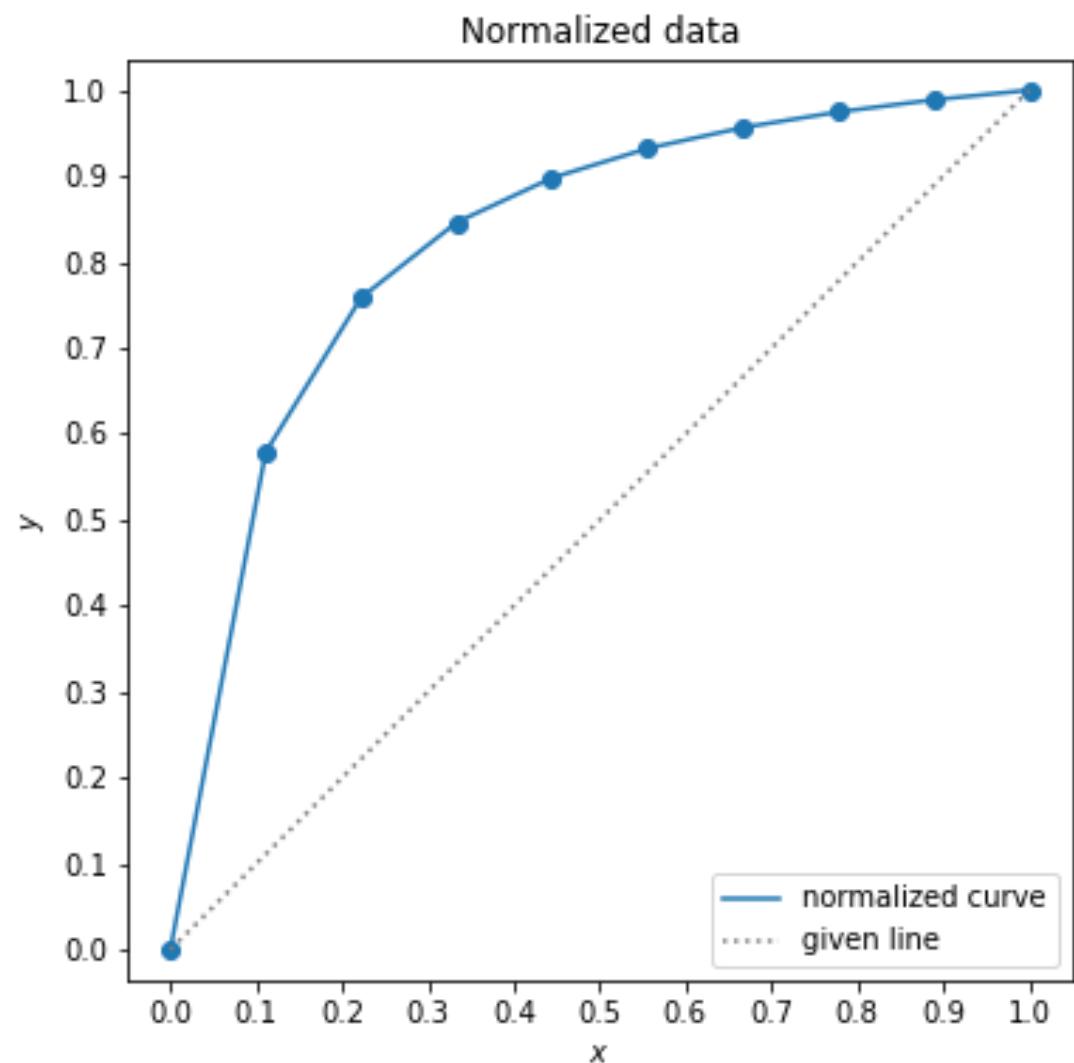
3. Normalize data

- Both axes in $[0, 1]$

Transformation

$$x_i \leftarrow \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$$

$$y_i \leftarrow \frac{y_i - y_{\min}}{y_{\max} - y_{\min}}$$

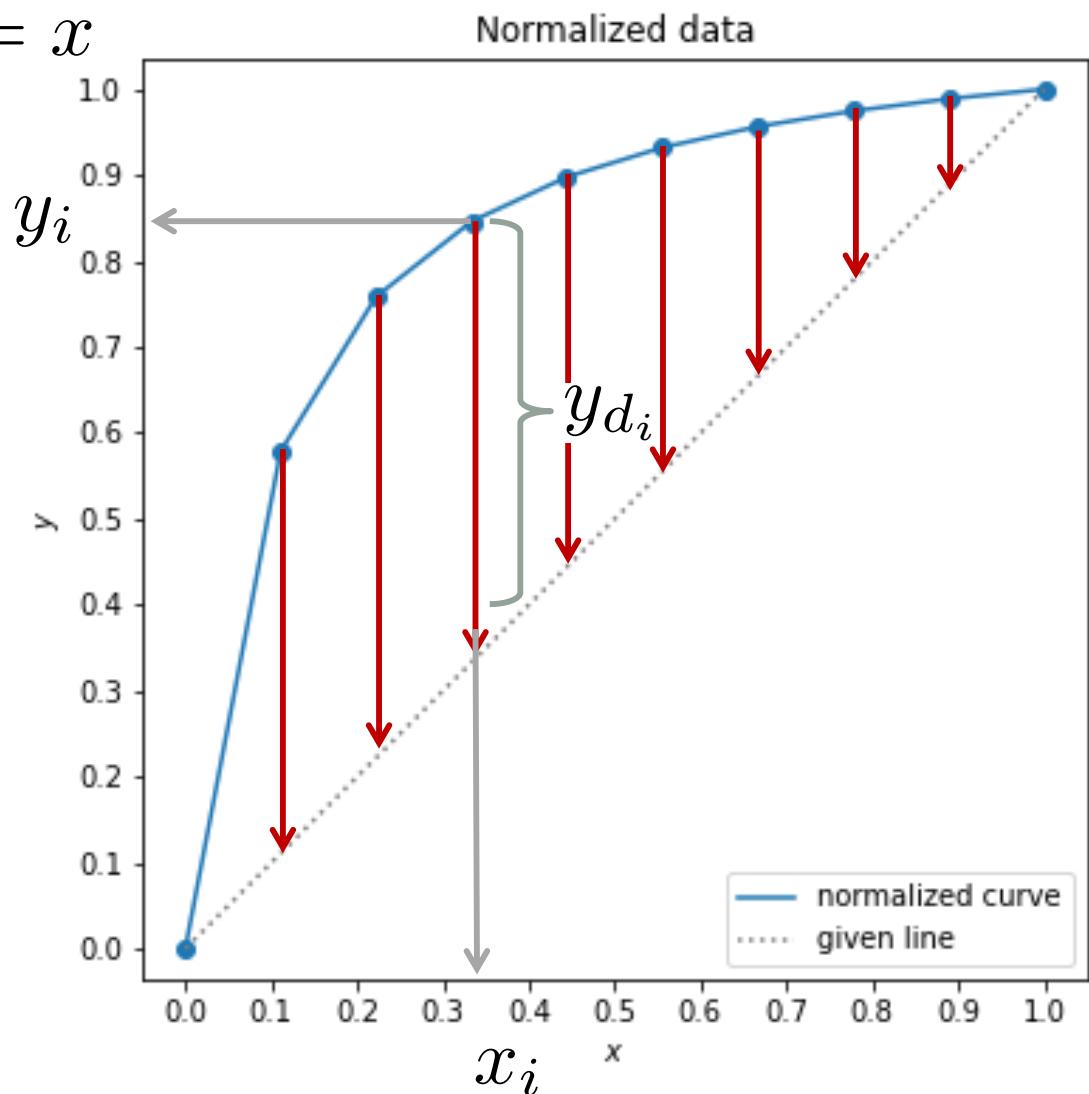


4. Calculate distances from $y=x$ line

- With respect to line $y = x$ such distances are related to the local value of the curvature of $f(x)$

Note:

- the curvature measures how much a curve deviates from a straight line
- hence, y_{d_i} is a good proxy to the curvature metric



5. Plotting such distances

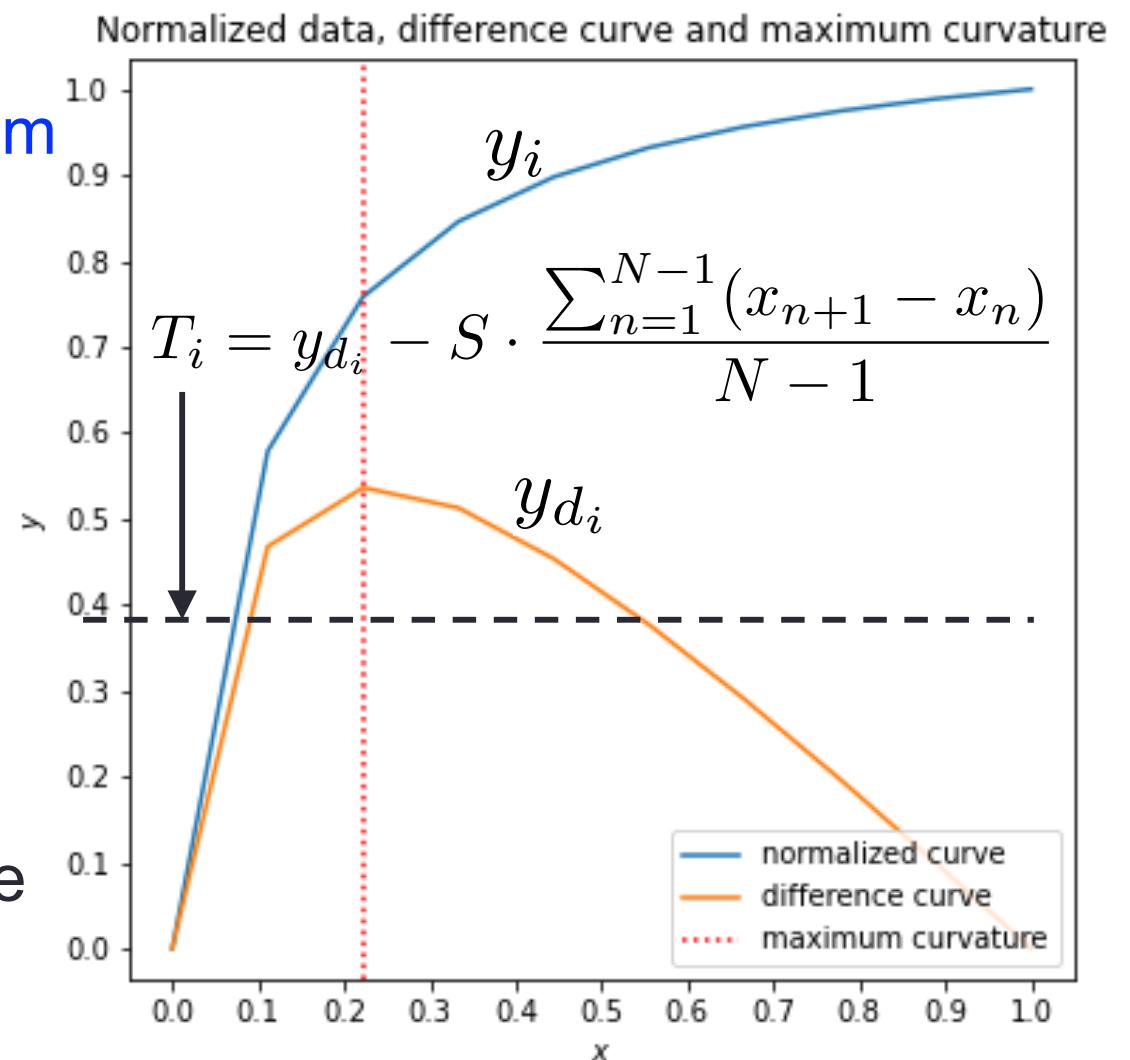
Assume:

(x_i, y_{d_i}) is a **local maximum**

y_{d_i} difference value for x_i

A threshold T_i is defined
for each local maximum
candidate

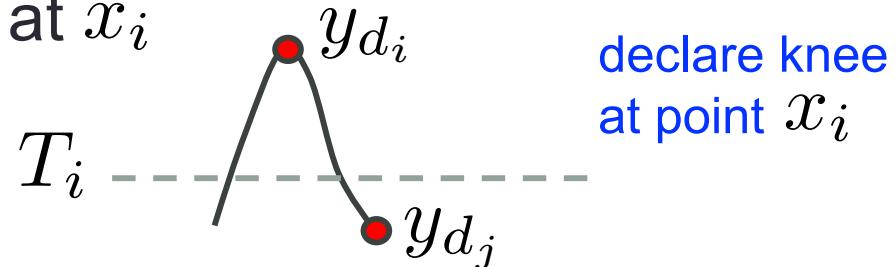
S is a (tunable) *sensitivity*
par (e.g., $S=1$) – to declare
the maximum



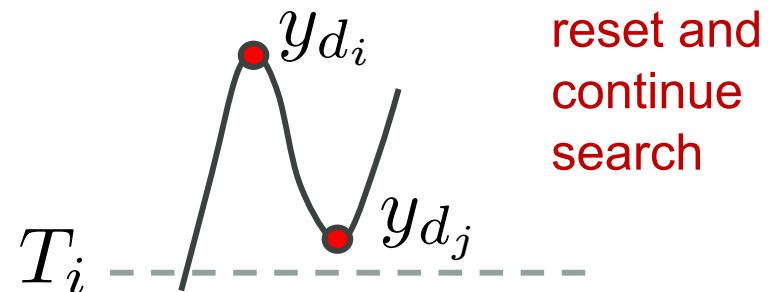
Locating maxima

If (x_i, y_{d_i}) is a local maximum

- If any subsequent value (x_j, y_{d_j}) with $j > i$ drops below T_i before the next local maximum in the difference curve is reached, a Knee is declared at x_i



- If (x_j, y_{d_j}) reach a local minimum and start to increase before reaching T_i we reset the threshold to 0 and wait for another local maximum to be reached



Threshold meaning

$$T_i = y_{d_i} - S \cdot$$

$$\frac{\sum_{n=1}^{N-1} (x_{n+1} - x_n)}{N - 1}$$

average distance of
two subsequent
points in the dataset

S: is a measure of how many (subsequent) flat points in the original curve we expect to see **before declaring a knee**

Note: if the original curve is flat as we move from $x_i \rightarrow x_{i+1}$
it holds

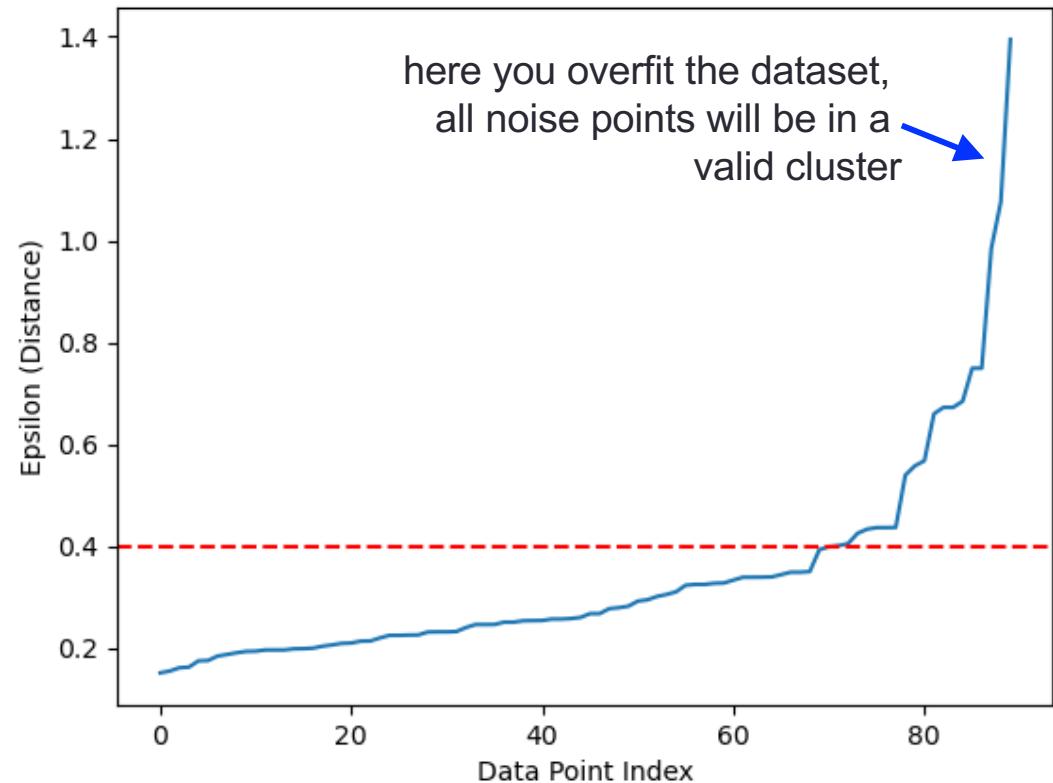
$$\Delta y_{d_i} = y_{d_{i+1}} - y_{d_i} = x_{i+1} - x_i$$

The **threshold** is

$$T_i = y_{d_i} - S \cdot \overline{(x_{n+1} - x_n)}$$

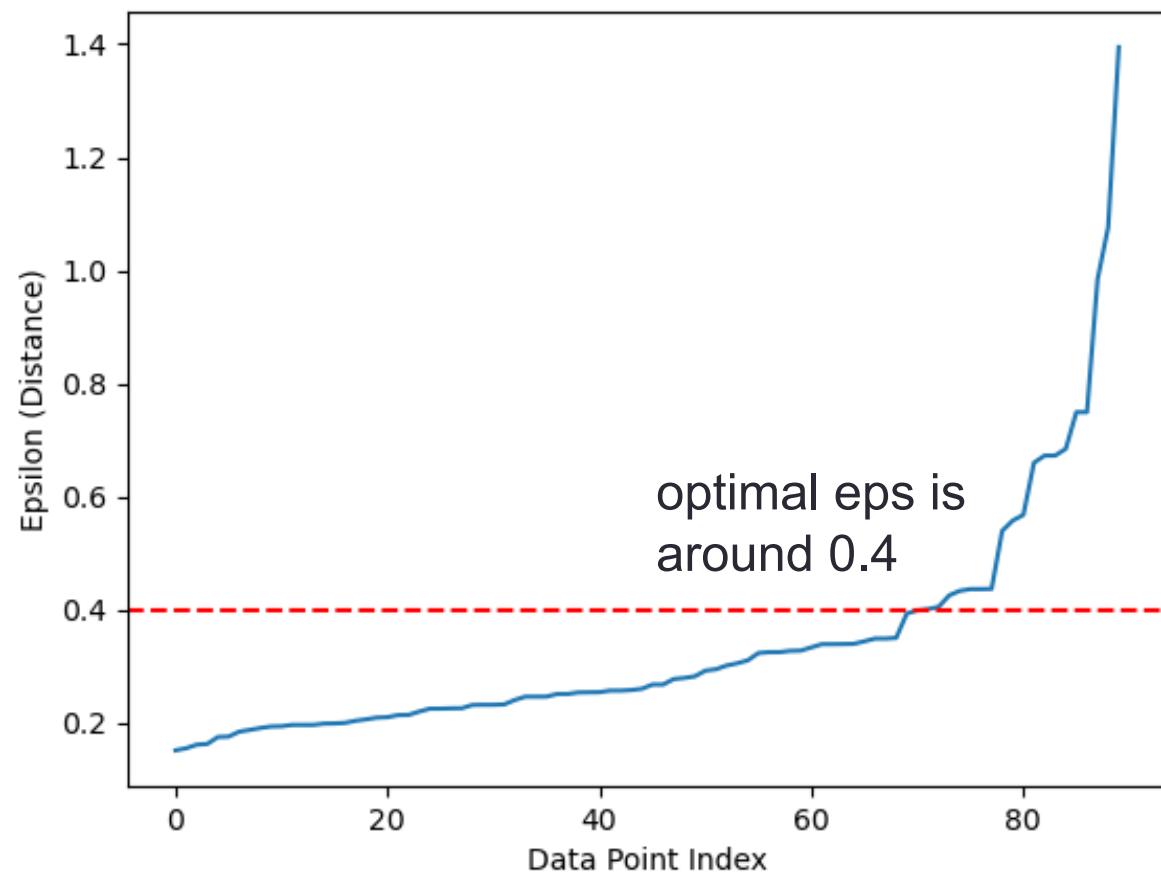
DBSCAN

- **For each point:** measure distance from its **MinPts** neighbors, compute & plot its average
- Order points based on such distance (ascending order)
- Plot distances for all points (ascending order)



DBSCAN

- Flip, smooth and normalize the curve
- Locate the optimal ε using Kneedle

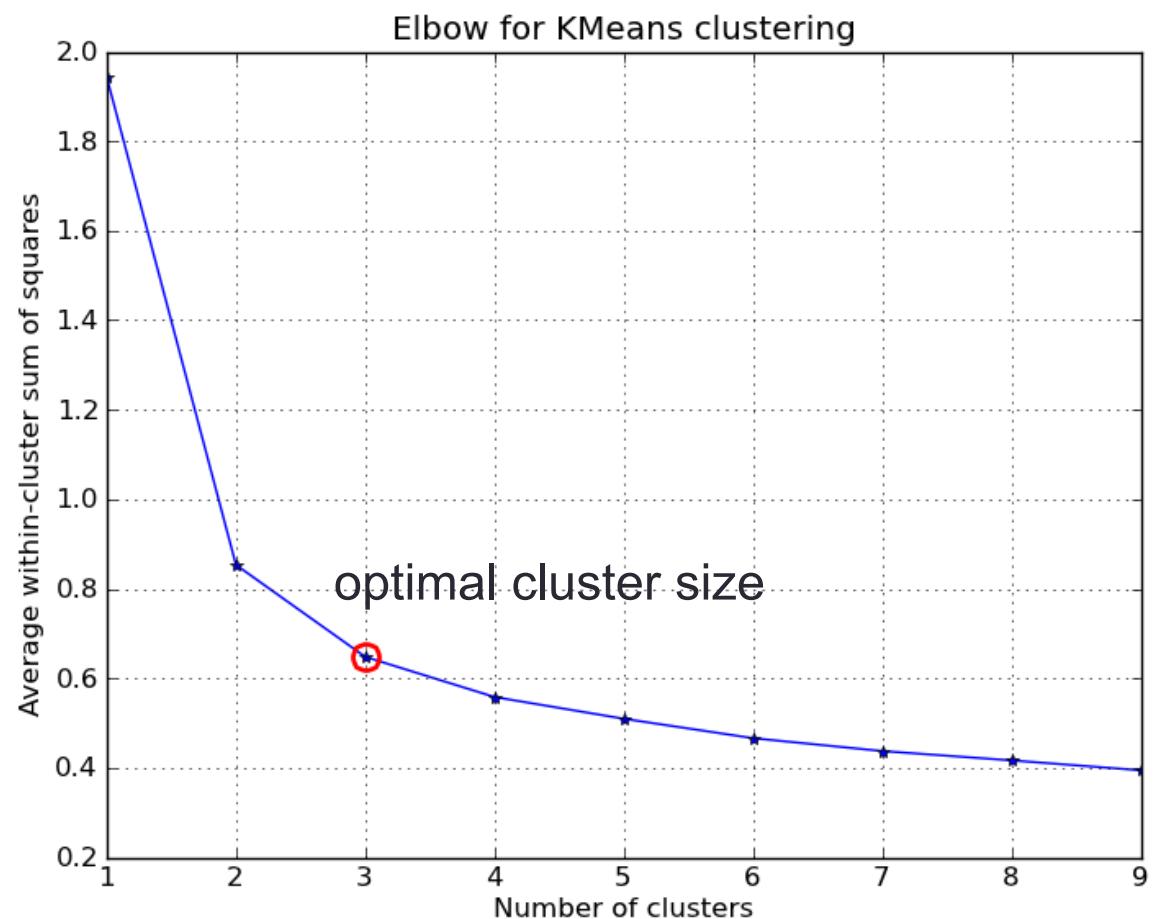


Clustering – selection of K

- Plot distortion measure (e.g., sum of distances from centroids)
- Look for inflection point
- Kneedle still applies
- Just replot using

$$y_i \leftarrow y_{\max} - y_i$$

- Smooth
- Normalize in $[0, 1]$



Python libraries

- Kneed
 - <https://github.com/arvkevi/ikneed>
- Kneebow
 - <https://pypi.org/project/kneebow/>

DENSITY BASED CLUSTERING: DBSCAN

Michele Rossi

michele.rossi@unipd.it

Dept. of Information Engineering
University of Padova, IT



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE



DIPARTIMENTO
MATEMATICA



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

