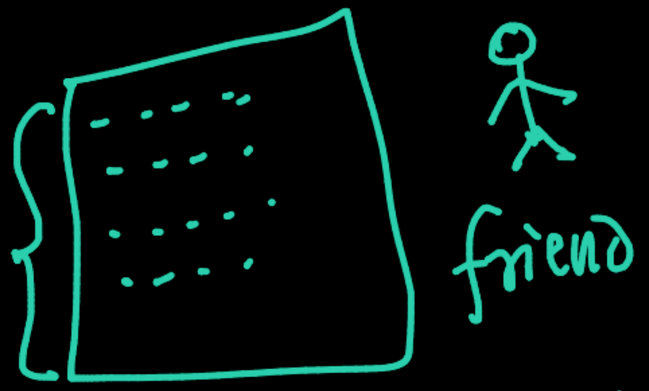
Topics:

- Agglomerative clustering
- DBSCAN

Q

1. Spaced revision :→ 3-4 hrs Week; Month; Quarter (3M)

2. Feynman's recipe :-



→ first principles "why"? "how"?

→ scribbled → video

Agglomerative

↓

n → each point is a cluster

⋮

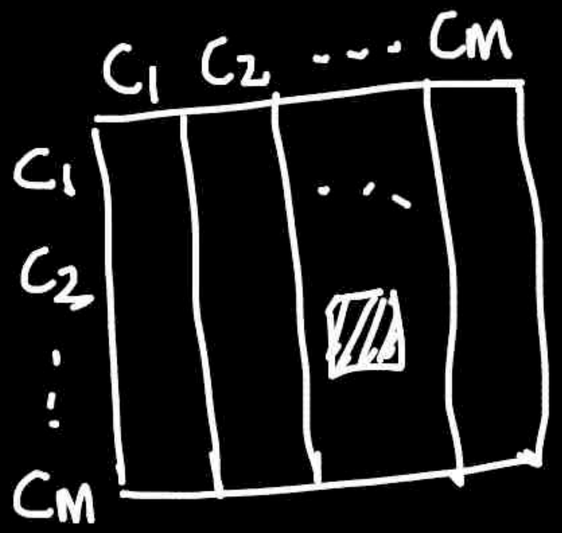
1 → one large cluster

Basic - structure:

1. each point is a cluster : n data points
2. proximity - matrix $P_{n \times n}$ (3) → how to compute distance b/w Ci & Cj
3. Repeat:
   - 3a. Merge the closest clusters
   - 3b. Update my proximity Matrix

   until a single cluster remains

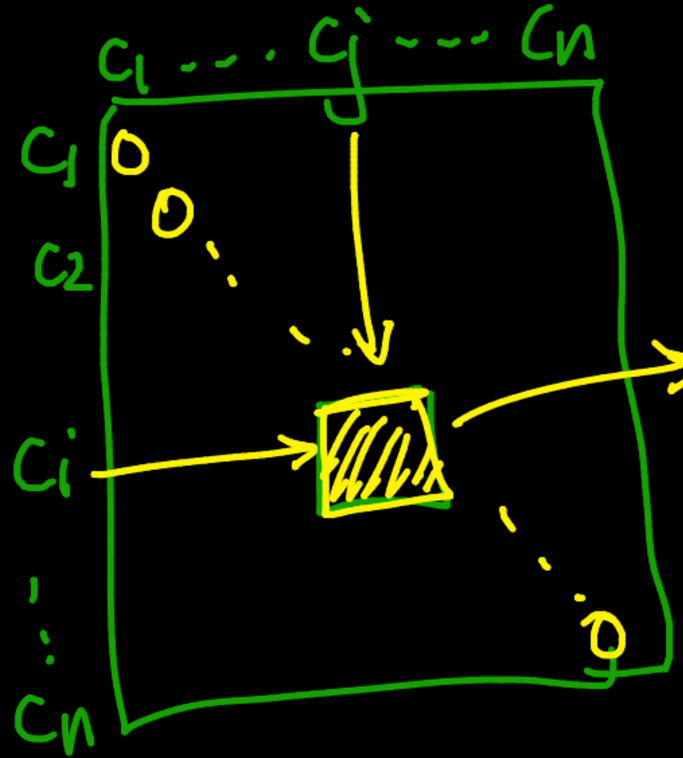
$P =$



$P_{ij} = Sim(C_1, C_2)$

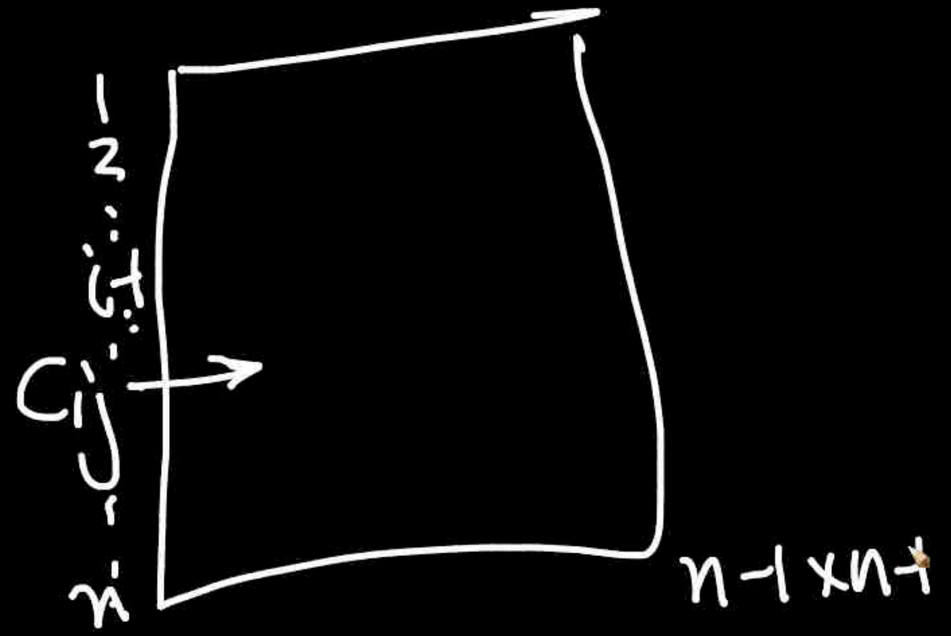
↑ inv of distance

Initially: P = n-clusters of 1 point each



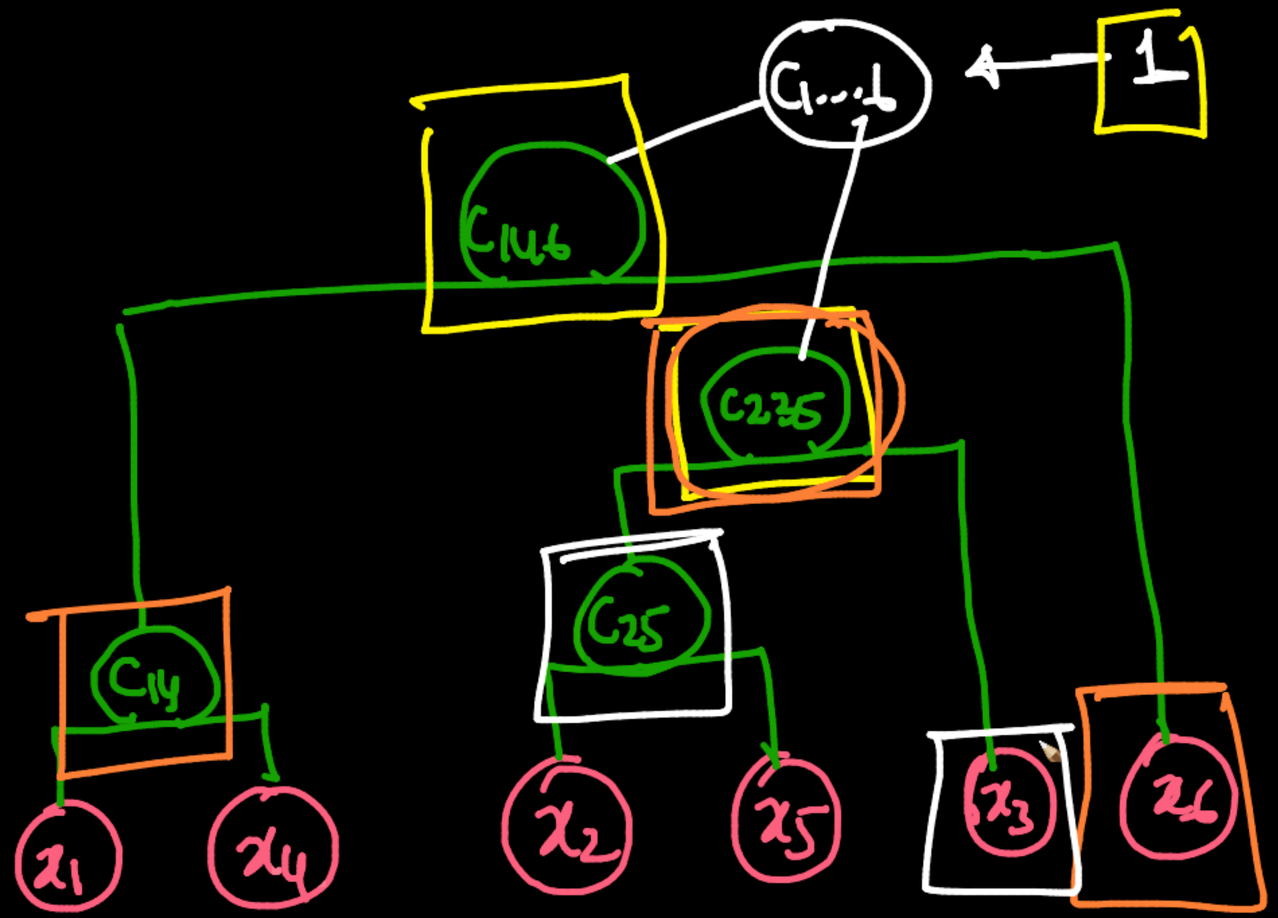
smallest $i \neq j$ ⇒ merge Ci & Cj

$P_{n-1 \times n-1}$



$C_{ij} = \{x_i, x_j\}$

dendrogram:

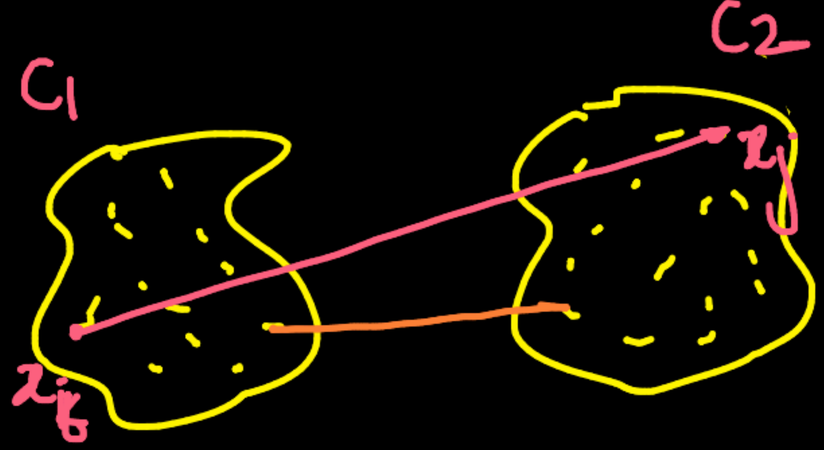


n →

K= 2-clusters

K=3

(Q) How to compute dist(Ci, Cj)?



1. euc. dist b/w centroids → d b/w centroids

2. MAX: $\max_{x_i \in C_1, x_j \in C_2} dist(x_i, x_j)$

3. MIN: $\min dist(x_i, x_j)$ s.t $x_i \in C_1, x_j \in C_2$

4. group average dist $= \frac{\sum_{x_j \in C_2} \sum_{x_i \in C_1} dist(x_i, x_j)}{|C_1| \, |C_2|}$

default (*) 2

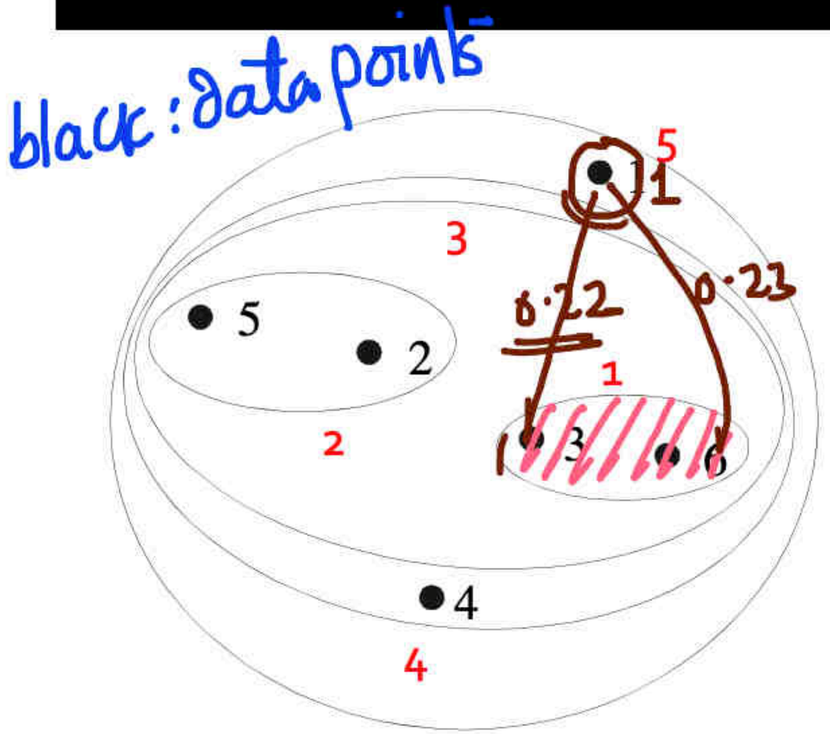
5: Wards dist $= \sum_{x_j \in C_2} \sum_{x_i \in C_1} \frac{[\text{euc. } dist(x_i, x_j)]^{(2)}}{|C_1| \, |C_2|}$

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Simple & low-malli

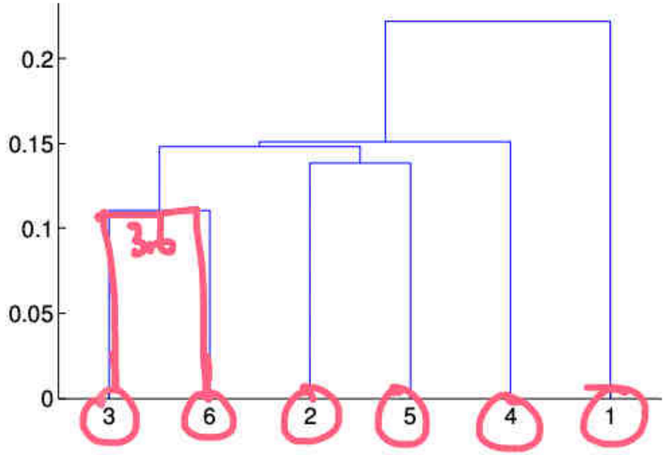
(logical & intuitive)

Hierarchical Clustering: MIN



Nested Clusters

	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0



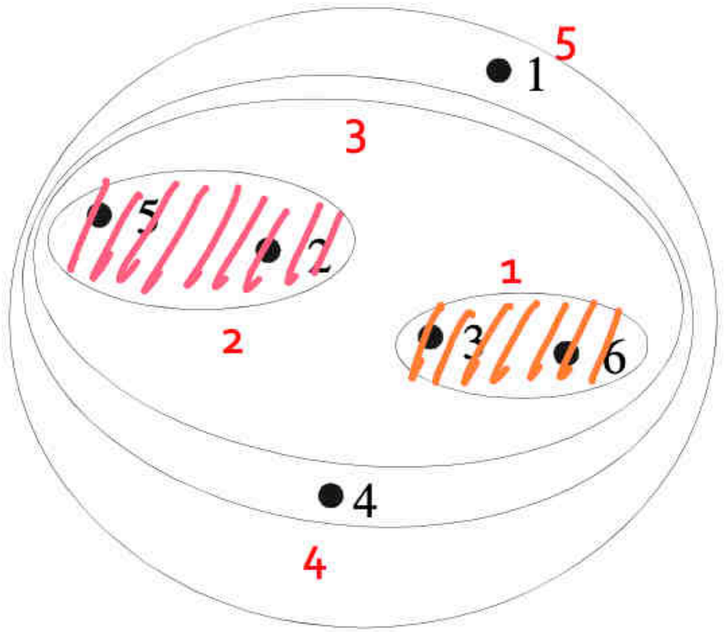
Dendrogram

Hierarchical Clustering: MIN

let

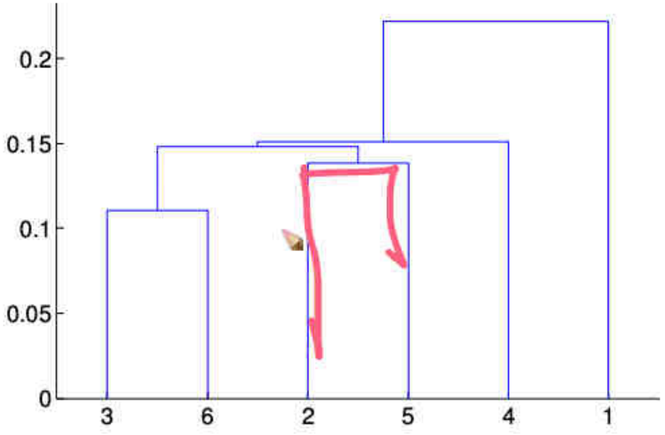
$C_1 - C_2$ ↑ 0.15

$C_2 - C_3$ 0.15



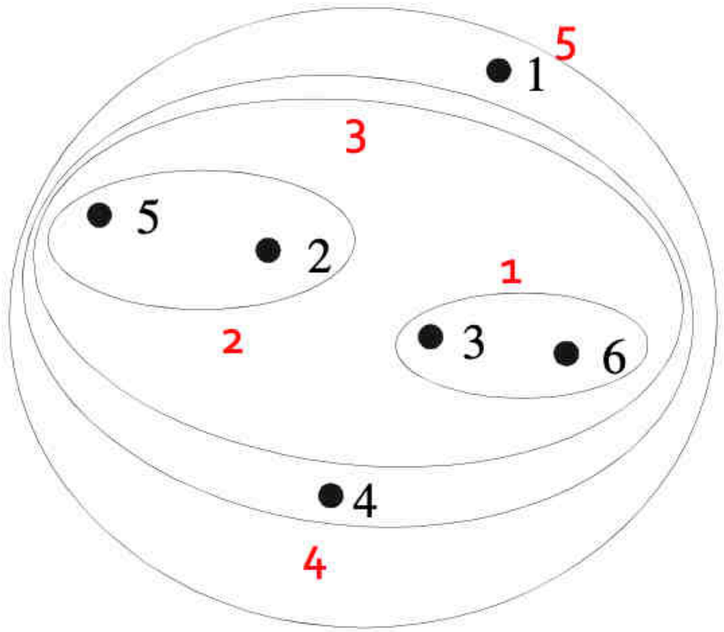
Nested Clusters

	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0



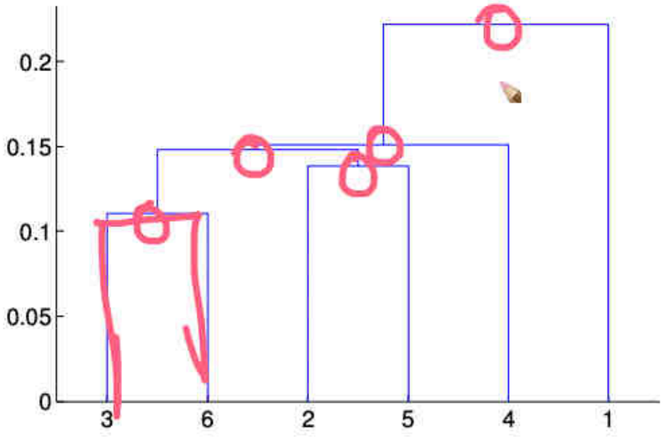
Dendrogram

Hierarchical Clustering: MIN



Nested Clusters

	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0

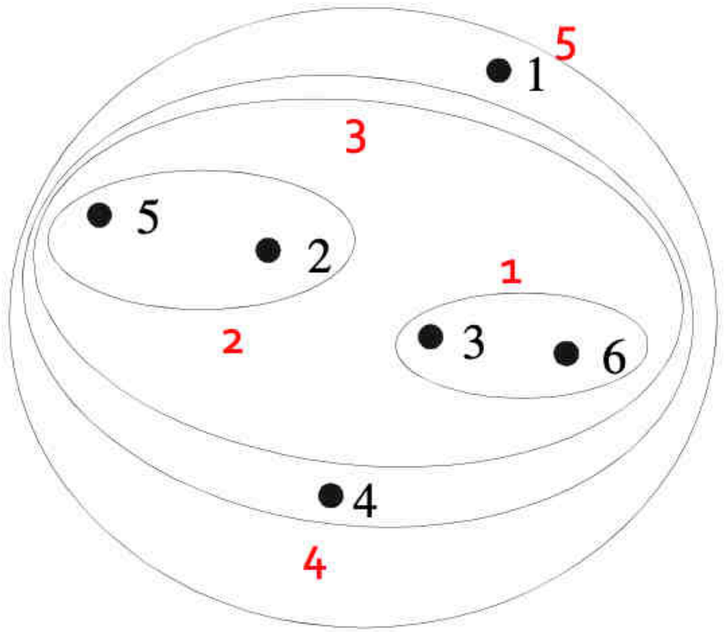


Dendrogram

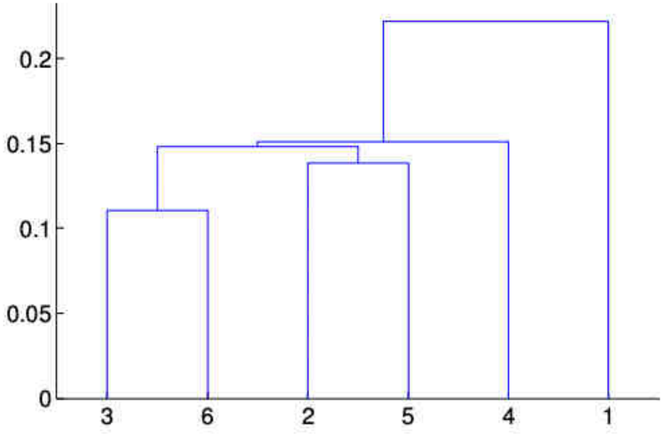
Hierarchical Clustering: MIN

Manually

	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0



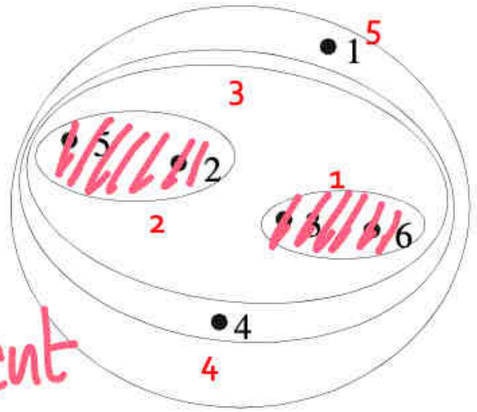
Nested Clusters



Dendrogram

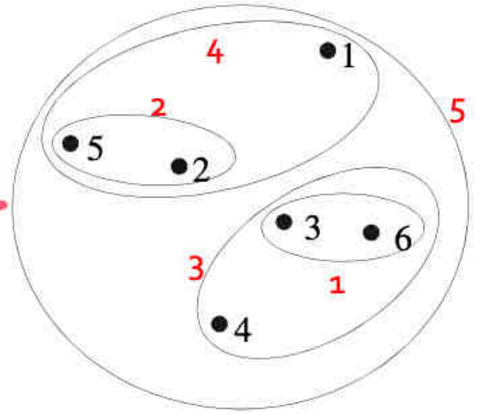
Hierarchical Clustering: Comparison

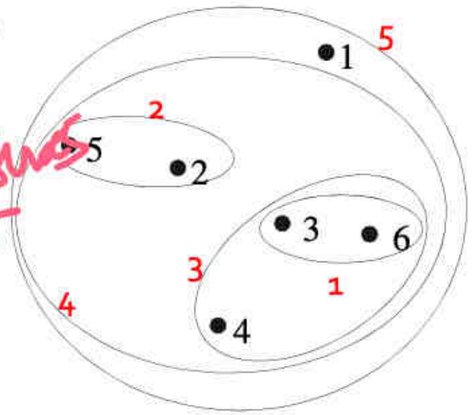
clustering results can be different for different dist-measures



MIN

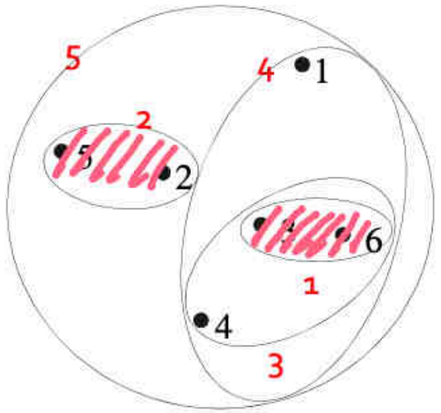
MAX



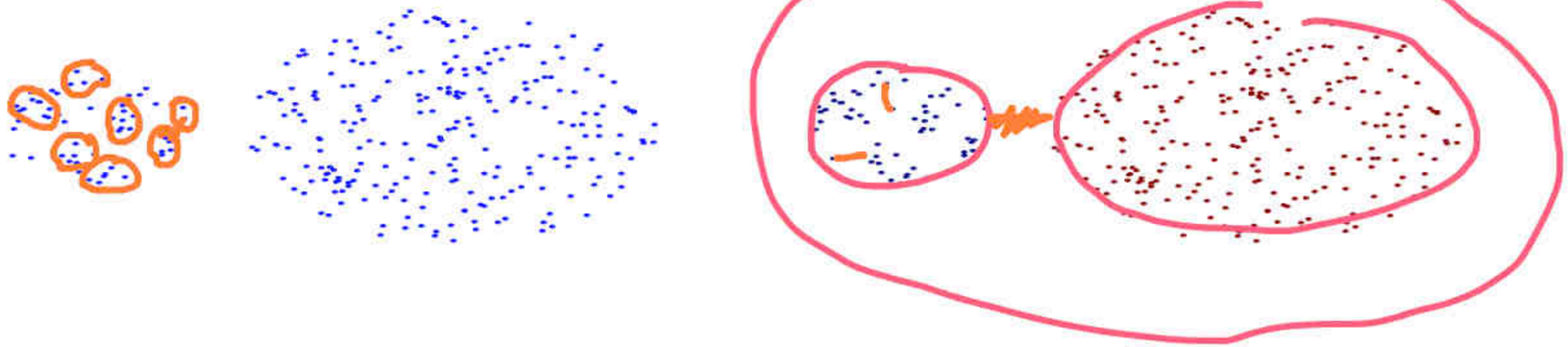


Group Average

Ward's Method



Strength of MIN

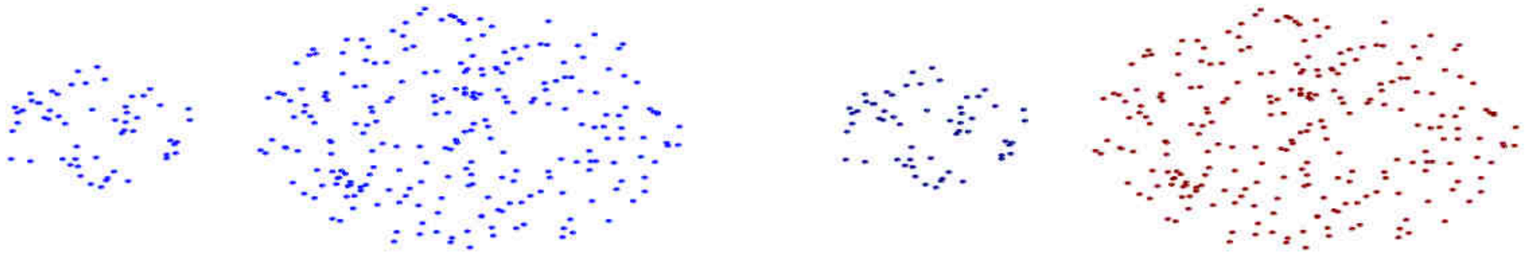


Original Points

Two Clusters

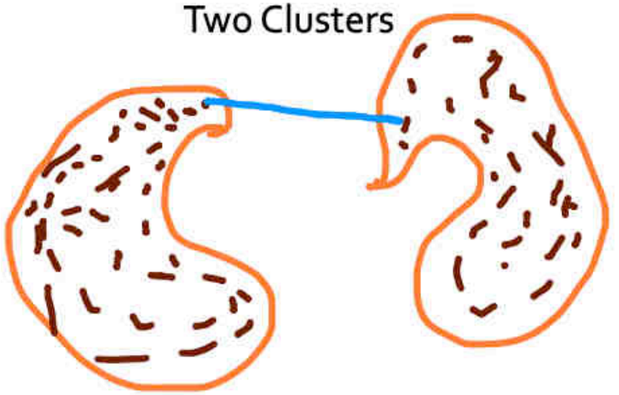
- Can handle non-elliptical shapes

Strength of MIN



Original Points

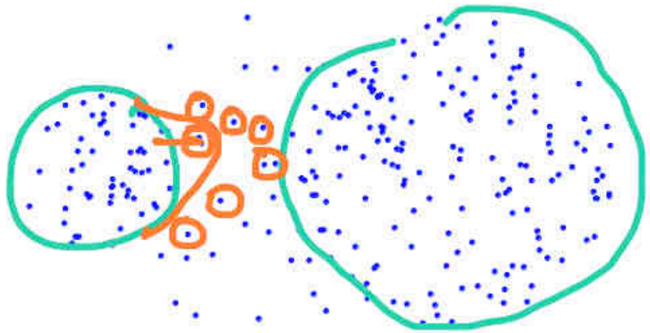
Two Clusters



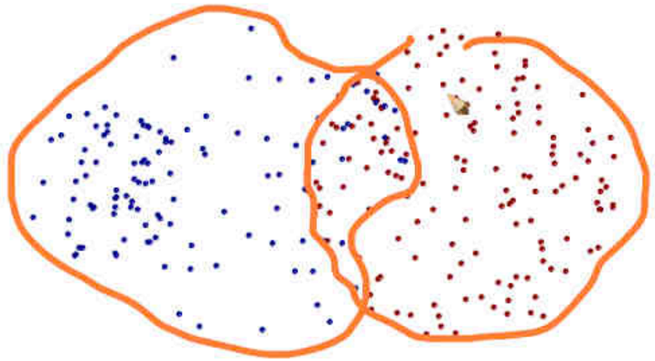
- Can handle non-elliptical shapes

non-globular

Limitations of MIN



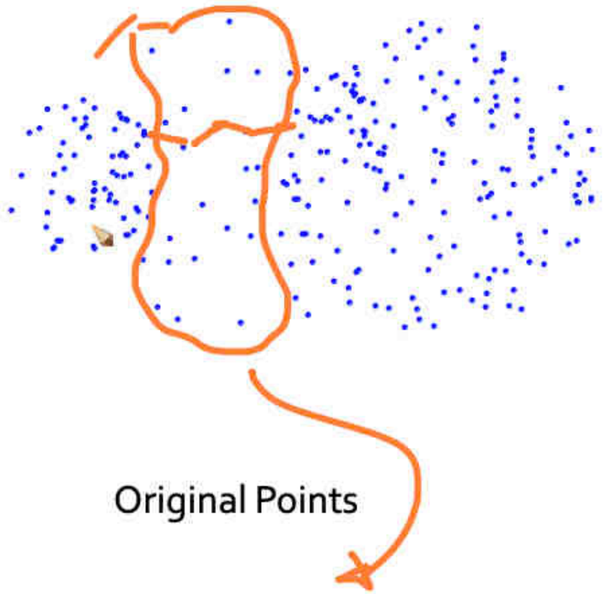
Original Points



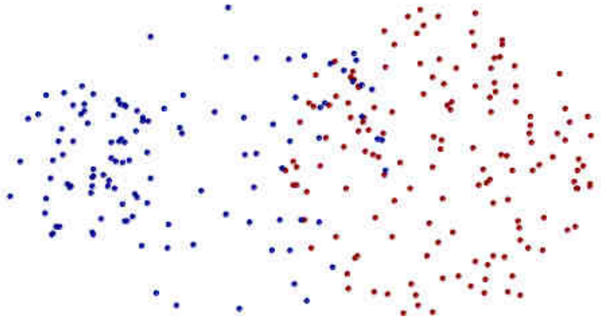
Two Clusters

- Sensitive to noise and outliers

Limitations of MIN → Single-linkage



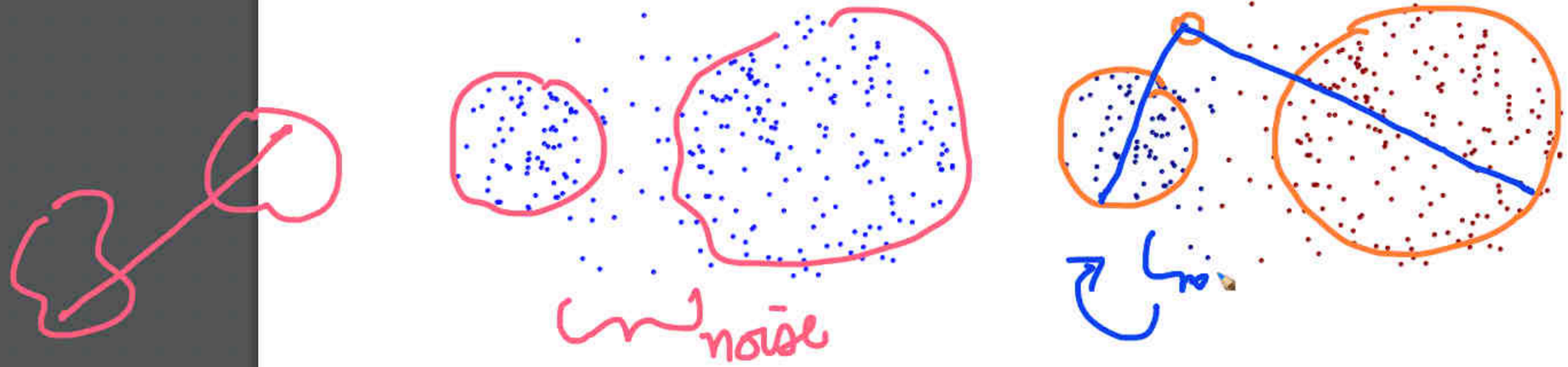
Original Points



Two Clusters

- Sensitive to noise and outliers

Strength of MAX : Complete linkage

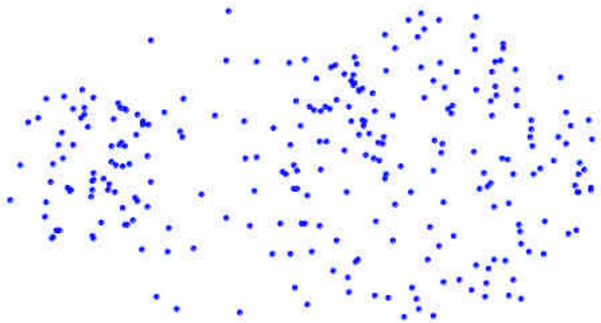


Original Points

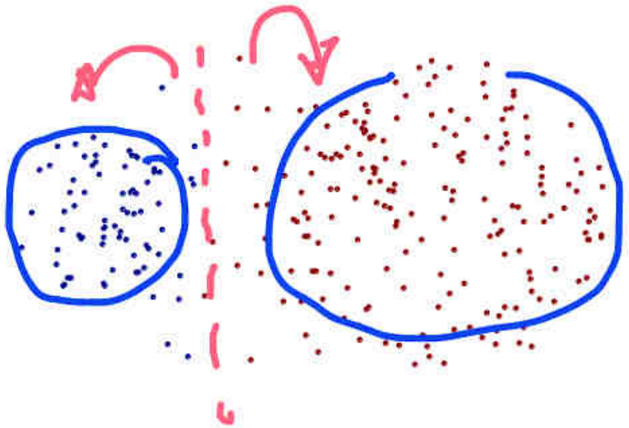
Two Clusters

- Less susceptible to noise and outliers

Strength of MAX



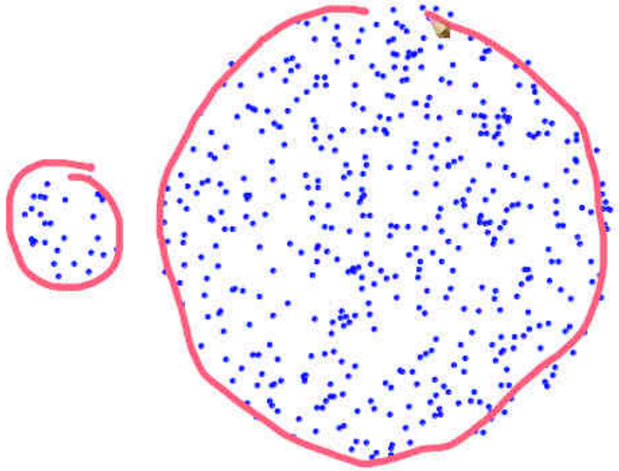
Original Points



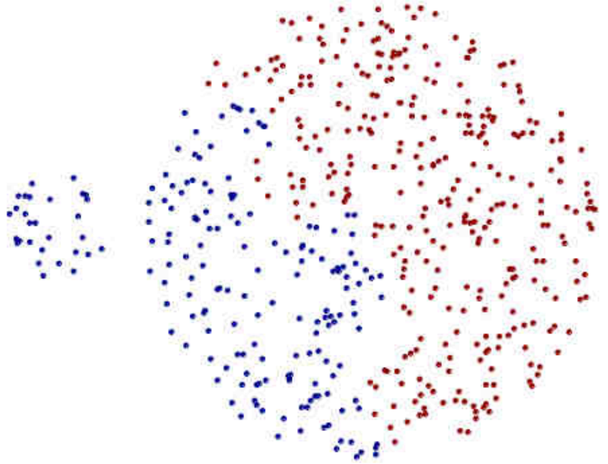
Two Clusters

- Less susceptible to noise and outliers

Limitations of MAX



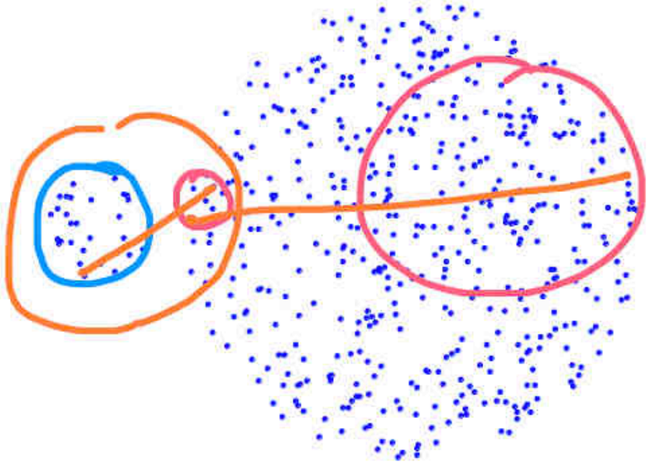
Original Points



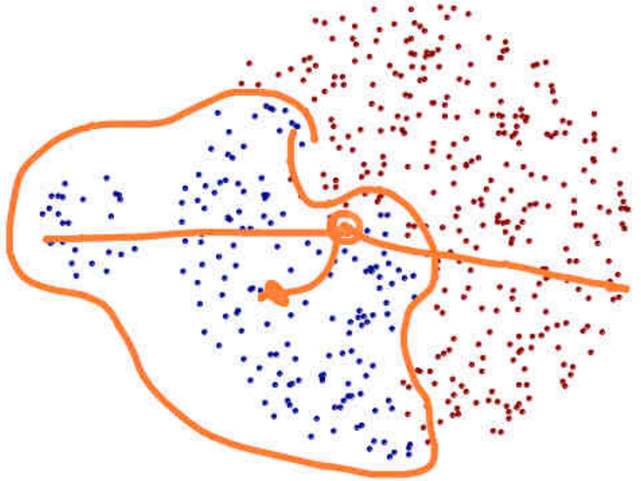
Two Clusters

- Tends to break large clusters
- Biased towards globular clusters

Limitations of MAX



Original Points



Two Clusters

- Tends to break large clusters
- Biased towards globular clusters

} → K-Means

Cluster Similarity: Group Average

- Proximity of two clusters is the average of pairwise proximity between points in the two clusters.

$$\text{proximity}(\text{Cluster}_i, \text{Cluster}_j) = \frac{\sum_{\substack{p_i \in \text{Cluster}_i \\ p_j \in \text{Cluster}_j}} \text{proximity}(p_i, p_j)}{|\text{Cluster}_i| * |\text{Cluster}_j|}$$

- Need to use average connectivity for scalability since total proximity favors large clusters

	1	2	3	4	5	6
1	0	.24	.22	.37	.34	.23
2	.24	0	.15	.20	.14	.25
3	.22	.15	0	.15	.28	.11
4	.37	.20	.15	0	.29	.22
5	.34	.14	.28	.29	0	.39
6	.23	.25	.11	.22	.39	0

Group-avg or ward algo

Hierarchical Clustering: Group Average

- Compromise between Single and Complete Link

Min Max

- Strengths
  - Less susceptible to noise and outliers

- Limitations
  - Biased towards globular clusters

Average

- Compromise between Single and Complete Link
- Strengths
  - Less susceptible to noise and outliers
- Limitations
  - Biased towards globular clusters

sklearn.cluster.AgglomerativeClustering

class sklearn.cluster.**AgglomerativeClustering**(*n_clusters=2*, ***, *affinity='euclidean'*, *memory=None*, *connectivity=None*, *compute_full_tree='auto'*, *linkage='ward'*, *distance_threshold=None*, *compute_distances=False*) [source]

Agglomerative Clustering.

Recursively merges pair of clusters of sample data; uses linkage distance.

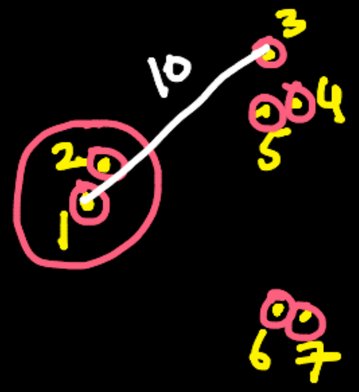
Read more in the User Guide.

Parameters:

n_clusters : *int or None, default=2*
The number of clusters to find. It must be `None` if `distance_threshold` is not `None`.

affinity : *str or callable, default='euclidean'*
Metric used to compute the linkage. Can be “euclidean”, “l1”, “l2”, “manhattan”, “cosine”, or “precomputed”. If linkage is “ward”, only “euclidean” is accepted. If “precomputed”, a distance matrix (instead of a similarity matrix) is needed as input for the fit method.

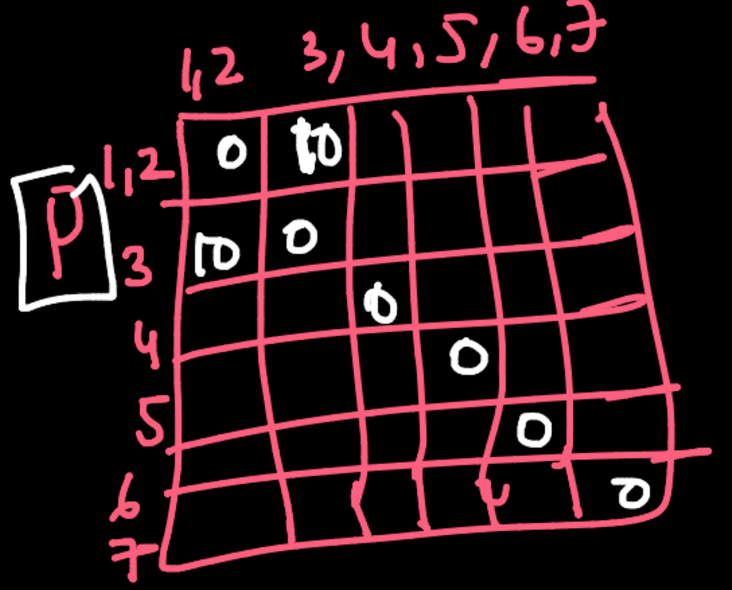
Common & dangerous mis conception



MAx / Complete linkage

P: Min / Max / wards

✓ merge 2 closest clusters



n is large →

Agglomerative Clustering

Expensive

Space: $O(n \times n)$: Proximity-matrix

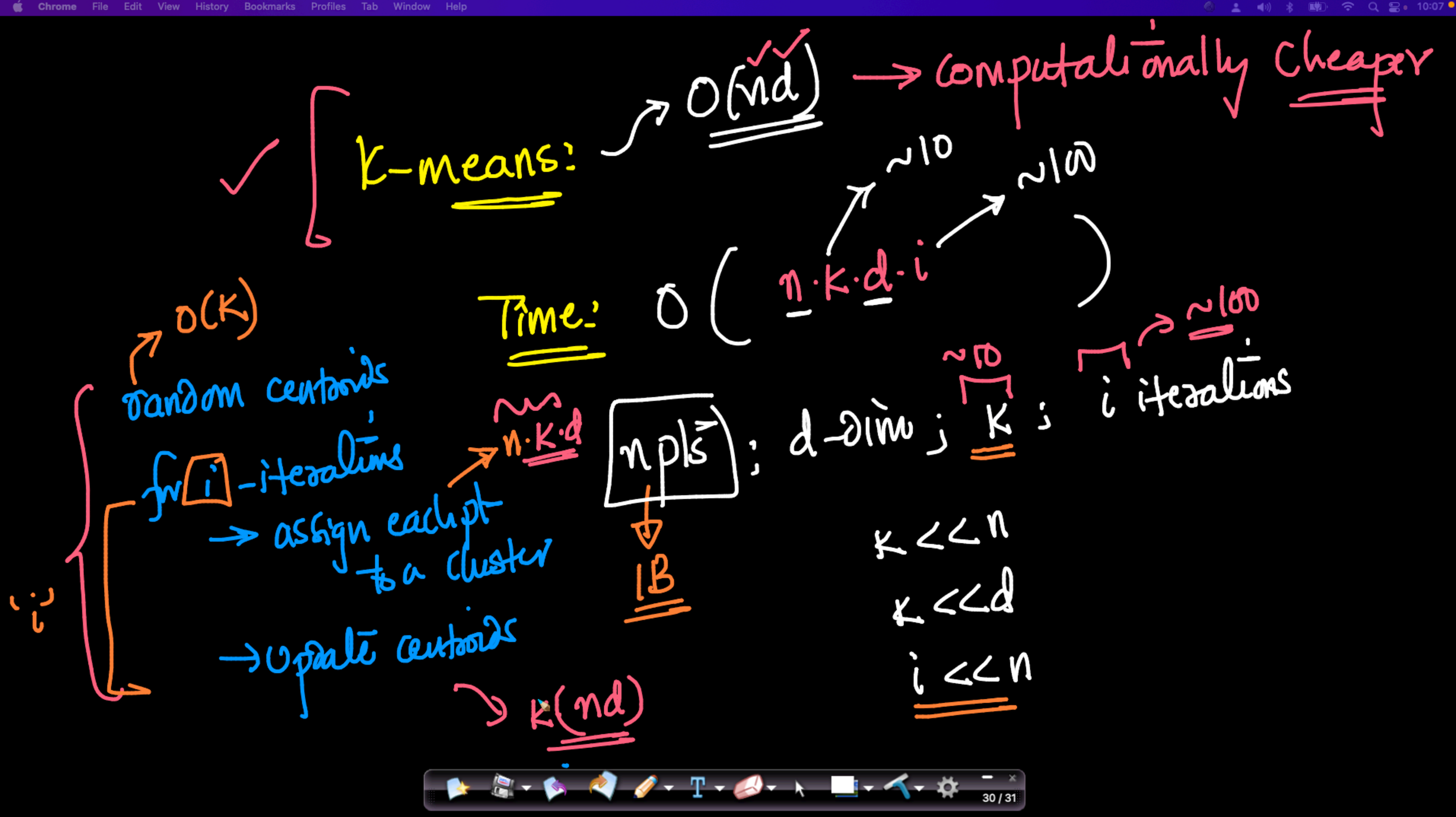
Time: $O(n^3)$ worst-case

optimized :- $O(n^2 \log n)$

'n'

- compute P
- repeat { merge; Update P } n^2
- Until one cluster

$n \to n-1 \to n-2 \to \cdots 1$


K-means:
O(nd)
→ computationally Cheaper
Time:
O(n·k·d·i)
~10
~100
O(K)
random centroids
for i - iterations
→ assign each pt to a cluster
→ update centroids
k(nd)
n·k·d
n pts
; d-dim ; K ; i iterations
~10
~100
LB
k << n
k << d
i << n
30 / 31

No Math obj that is being minimized in Agg-clustering

K-means: min within cluster ss-dist

MIN & MAX → resolved using Ward's method

sklearn.cluster.AgglomerativeClustering

class sklearn.cluster.**AgglomerativeClustering**(*n_clusters=2*, ***, *affinity='euclidean'*, *memory=None*, *connectivity=None*, *compute_full_tree='auto'*, *linkage='ward'*, *distance_threshold=None*, *compute_distances=False*) [source]

Agglomerative Clustering.

Recursively merges pair of clusters of sample data; uses linkage distance.

Read more in the User Guide.

Parameters:

n_clusters : *int or None, default=2*

The number of clusters to find. It must be `None` if `distance_threshold` is not `None`.

affinity : *str or callable, default='euclidean'*

Metric used to compute the linkage. Can be “euclidean”, “l1”, “l2”, “manhattan”, “cosine”, or “precomputed”. If linkage is “ward”, only “euclidean” is accepted. If “precomputed”, a distance matrix (instead of a similarity matrix) is needed as input for the fit method.

memory : *str or object with the joblib.Memory interface, default=None*

Read more in the User Guide.

Parameters:

n_clusters : *int or None, default=2*

The number of clusters to find. It must be `None` if `distance_threshold` is not `None`.

affinity : *str or callable, default='euclidean'*

Metric used to compute the linkage. Can be "euclidean", "l1", "l2", "manhattan", "cosine", or "precomputed". If linkage is "ward", only "euclidean" is accepted. If "precomputed", a distance matrix (instead of a similarity matrix) is needed as input for the fit method.

memory : *str or object with the joblib.Memory interface, default=None*

Used to cache the output of the computation of the tree. By default, no caching is done. If a string is given, it is the path to the caching directory.

connectivity : *array-like or callable, default=None*

Connectivity matrix. Defines for each sample the neighboring samples following a given structure of the data. This can be a connectivity matrix itself or a callable that transforms the data into a connectivity matrix, such as derived from `kneighbors_graph`. Default is `None`, i.e, the hierarchical clustering algorithm is unstructured.

compute_full_tree : *'auto' or bool, default='auto'*

`sters`. This is useful to decrease

scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
sklearn.cluster.AgglomerativeClustering
Examples using sklearn.cluster.AgglomerativeCl

dendogram

Connectivity matrix. Defines for each sample the neighboring samples following a given structure of the data. This can be a connectivity matrix itself or a callable that transforms the data into a connectivity matrix, such as derived from `kneighbors_graph`. Default is `None`, i.e, the hierarchical clustering algorithm is unstructured.

compute_full_tree : *'auto' or bool, default='auto'* =10

Stop early the construction of the tree at `n_clusters`. This is useful to decrease computation time if the number of clusters is not small compared to the number of samples. This option is useful only when specifying a connectivity matrix. Note also that when varying the number of clusters and using caching, it may be advantageous to compute the full tree. It must be `True` if `distance_threshold` is not `None`. By default `compute_full_tree` is "auto", which is equivalent to `True` when `distance_threshold` is not `None` or that `n_clusters` is inferior to the maximum between 100 or `0.02 * n_samples`. Otherwise, "auto" is equivalent to `False`.

linkage : *{'ward', 'complete', 'average', 'single'}, default='ward'*

Which linkage criterion to use. The linkage criterion determines which distance to use between sets of observation. The algorithm will merge the pairs of cluster that minimize this criterion.

- 'ward' minimizes the variance of the clusters being merged.

of each observation of the two sets.

maximum distances between all obser-

K-means: centroid-based

Agg-clustering: hierarchical system

DB-SCAN → density-based

databases & data mining

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

DBSCAN

From Wikipedia, the free encyclopedia

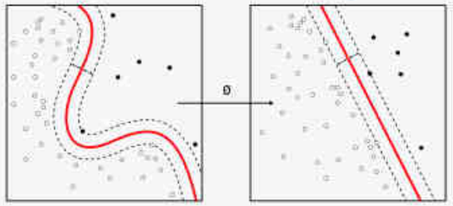
Density-based spatial clustering of applications with noise (**DBSCAN**) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.[1] It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.[2]

In 2014, the algorithm was awarded the test of time award (an award given to algorithms which have received substantial attention in theory and practice) at the leading data mining conference, ACM SIGKDD.[3] As of July 2020, the follow-up paper "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN"[4] appears in the list of the 8 most downloaded articles of the prestigious ACM Transactions on Database Systems (TODS) journal.[5]

Contents [hide]

1 History
2 Preliminary
3 Algorithm
  3.1 Original query-based algorithm
  3.2 Abstract algorithm
4 Complexity
5 Advantages
6 Disadvantages
7 Parameter estimation
8 Relationship to spectral clustering

Part of a series on
Machine learning and data mining



Problems [show]

Supervised learning [show]
(**classification** · **regression**)

Clustering [hide]
BIRCH · CURE · Hierarchical · *k*-means · Expectation–maximization (EM) · **DBSCAN** · OPTICS · Mean shift

Dimensionality reduction [show]

Structured prediction [show]

Anomaly detection [show]

Artificial neural network [show]

Reinforcement learning [show]

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page
Wikidata item

DBSCAN

From Wikipedia, the free encyclopedia

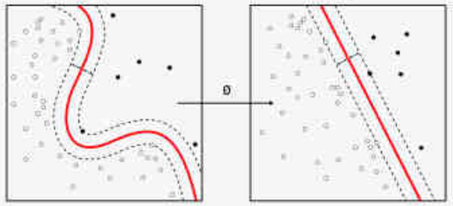
Density-based spatial clustering of applications with noise (**DBSCAN**) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.[1] It is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.[2]

In 2014, the algorithm was awarded the test of time award (an award given to algorithms which have received substantial attention in theory and practice) at the leading data mining conference, ACM SIGKDD.[3] As of July 2020, the follow-up paper "DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN"[4] appears in the list of the 8 most downloaded articles of the prestigious ACM Transactions on Database Systems (TODS) journal.[5]

Contents [hide]

1 History
2 Preliminary
3 Algorithm
3.1 Original query-based algorithm
3.2 Abstract algorithm
4 Complexity
5 Advantages
6 Disadvantages
7 Parameter estimation
8 Relationship to spectral clustering

Part of a series on
Machine learning and data mining



Problems [show]
Supervised learning [show]
(**classification** · **regression**)
Clustering [hide]
BIRCH · CURE · Hierarchical · *k*-means · Expectation–maximization (EM) · **DBSCAN** · OPTICS · Mean shift
Dimensionality reduction [show]
Structured prediction [show]
Anomaly detection [show]
Artificial neural network [show]
Reinforcement learning [show]

Terms / concepts

(key-ideas)

- Min pts } → hyper-params
- Eps }
- Core-point
- border-point
- noise-point

density @ $\vec{p}$ = # points within the hypersphere centered @ $\vec{p}$ with radius = ϵps

ϵps → 1.0

"dense" region

→ eps

→ Minpts

to

≥ Minpts → th

e.g: eps = 10 → radius

✓ {Core-point:

↳ { if p has ≥ Minpts in an Eps radius around it then P is a core-pt

⇓

p has a 'dense' region around it → Eps, Minpts

border-pt:

→ p is not a core-pt

→ p ∈ Neighborhood(q) s.t q is a core pt

⇓

dist(p, q) ≤ eps

pt some dense region

not-core

density @ P = 6 ≱ Minpts

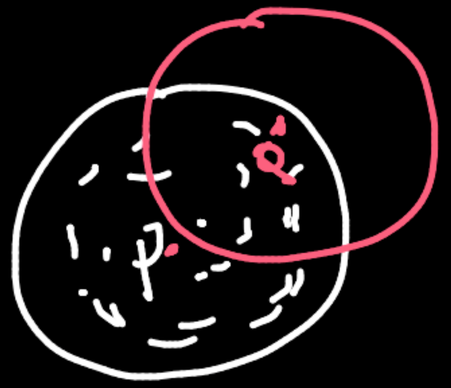
density @ Q = 12 ≥ Minpts

Q → core

Minpts = 10

Noise pt!

neither core nor border pt



p: core

q: border

not border

R not in a dense region

Too few pts in eps-radius

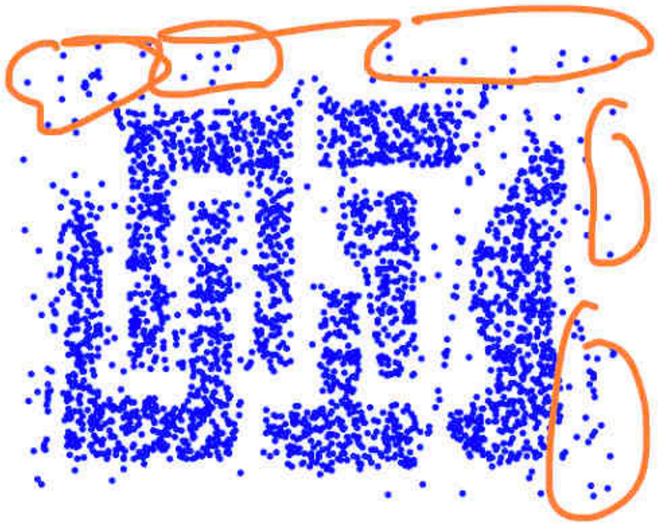
non-core

DB SCAN → Minpts ; Eps → hyper-params

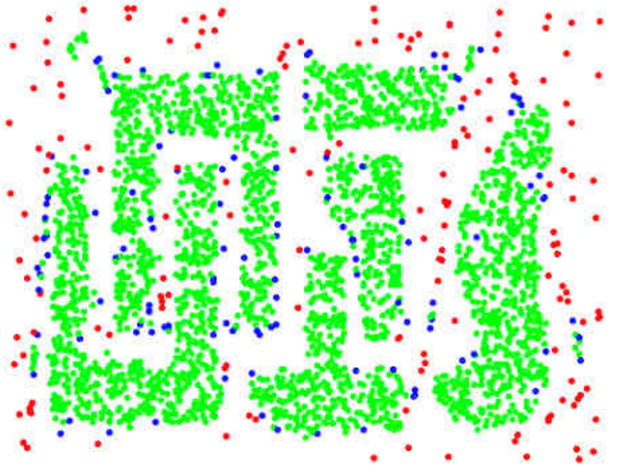
↓

Core pts ; border pts ; noise pts

DBSCAN: Core, Border and Noise Points



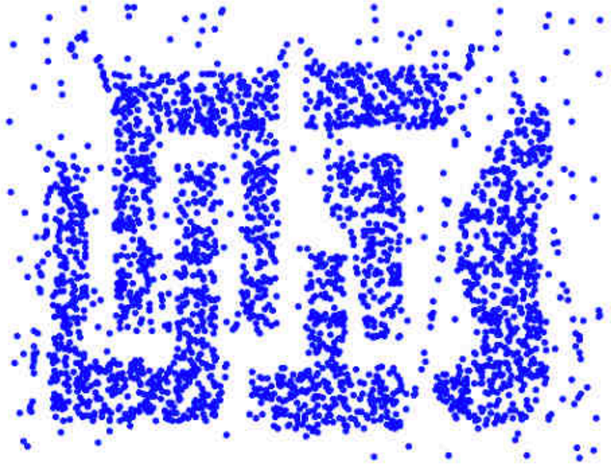
Original Points



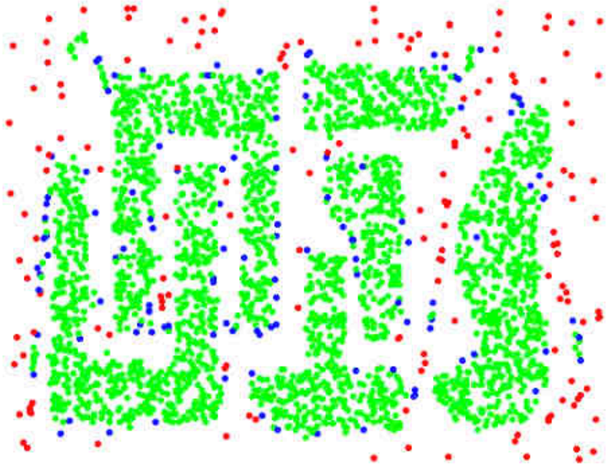
Point types: core, border and noise

Eps = 10, MinPts = 4

DBSCAN: Core, Border and Noise Points



Original Points



Point types: core, border and noise

Eps = 10, MinPts = 4

Chrome File Edit View History Bookmarks Profiles Tab Window Help

density - edge/connection

Vertex / node

Graphs

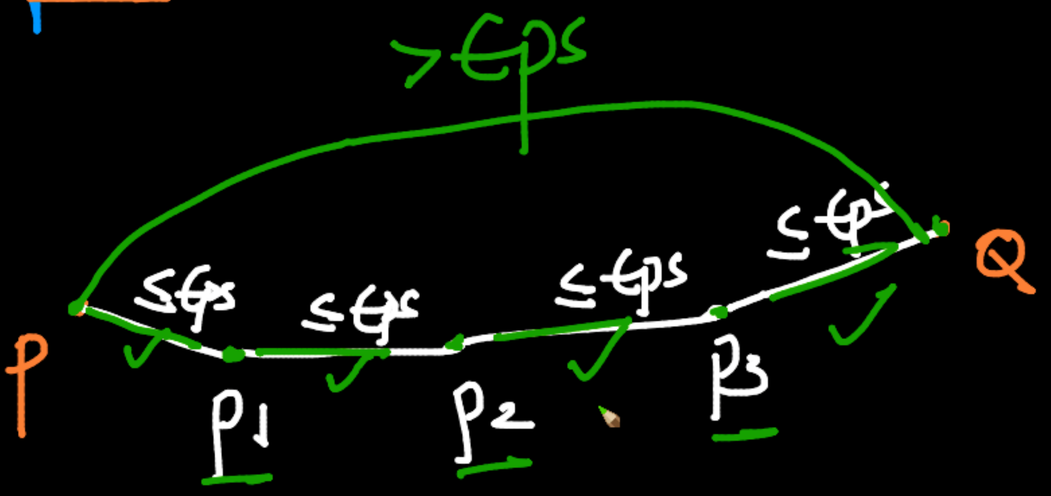
✓ {each-pt is a vertex
line $p_1 - p_2$: edge

edge pq is a density edge

if P & Q are core-pts (and) dist(P,Q) $\leq$ eps

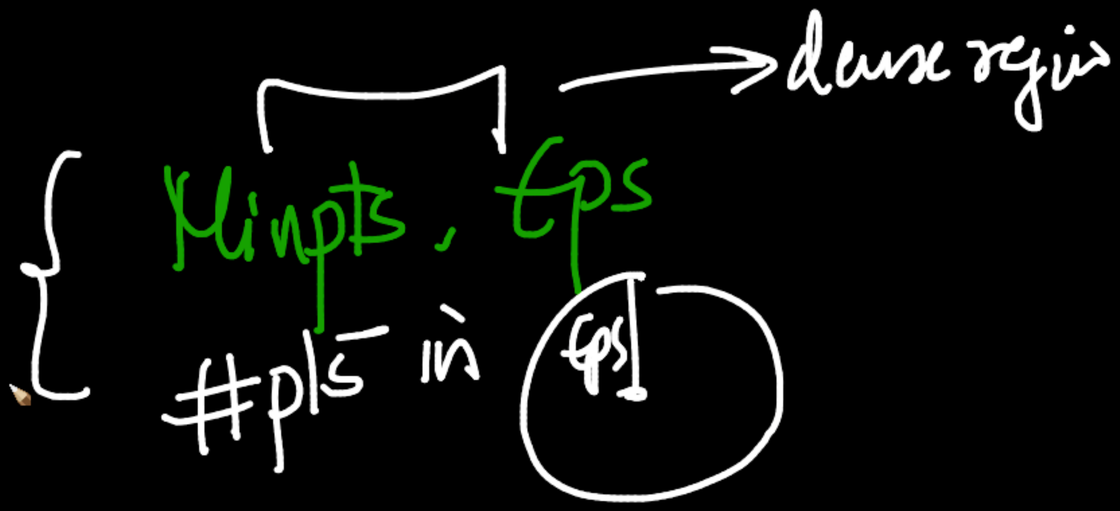
↳ dense regions

density connected pts



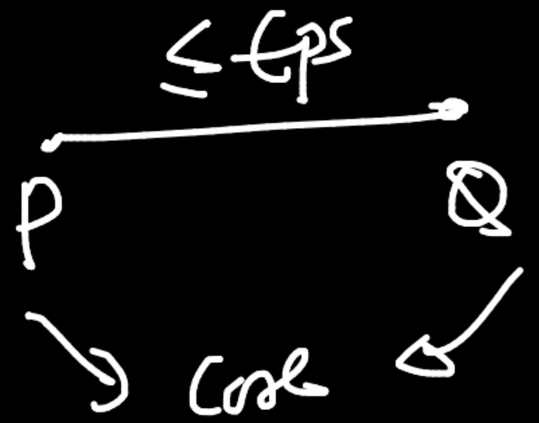
{ P & Q : core pts
P1, P2, P3 : core pts

P & Q are density connected if there exists density edges connecting P & Q

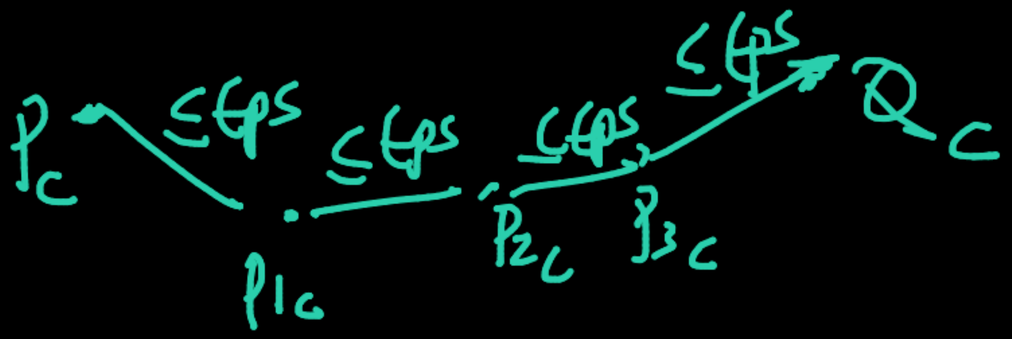


- Core pt: dense region
- border pt: non core but lies in a dense
- noise pt: non-core + non-border

density edge



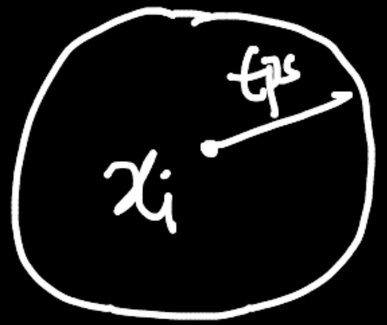
density connected



(Minpts, Eps) DB-SCAN:

1. $\forall x_i \in D$ label: core pt; border pt; noise pt

$n \lg n$ →



DS: R^+ trees, kd-trees ...

RangeQuery(D, x_i, Eps) → fast $\lg(n)$

↓

#pts / density?

(2) noise-removal (density)

remove all noise pts from your datasets

Sparse regions

→ O(n)

(3) For each core pt p (not assigned to any cluster) ← Plc

- create a new cluster with p
- add all points that are density connected to p in p's cluster

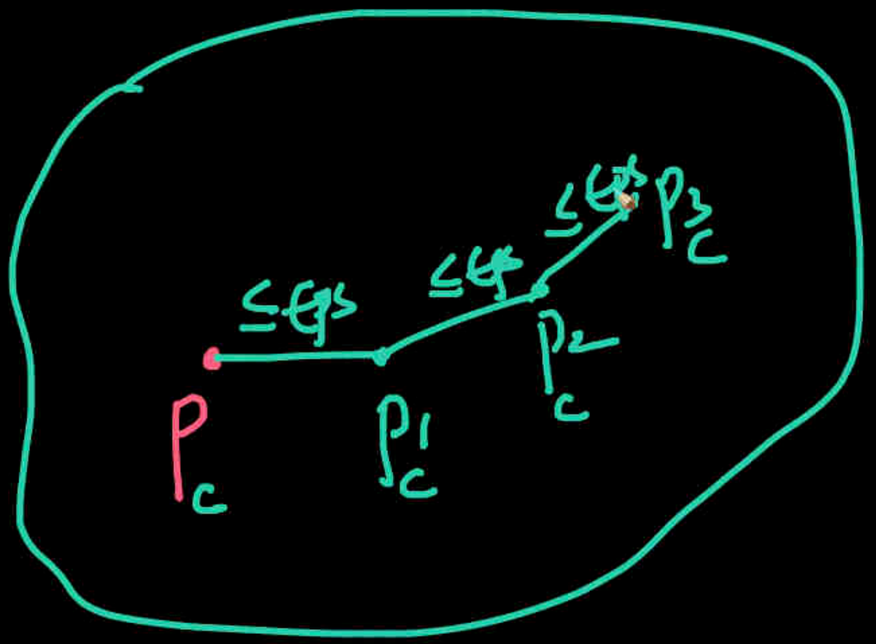
core-pts

density connected

↓

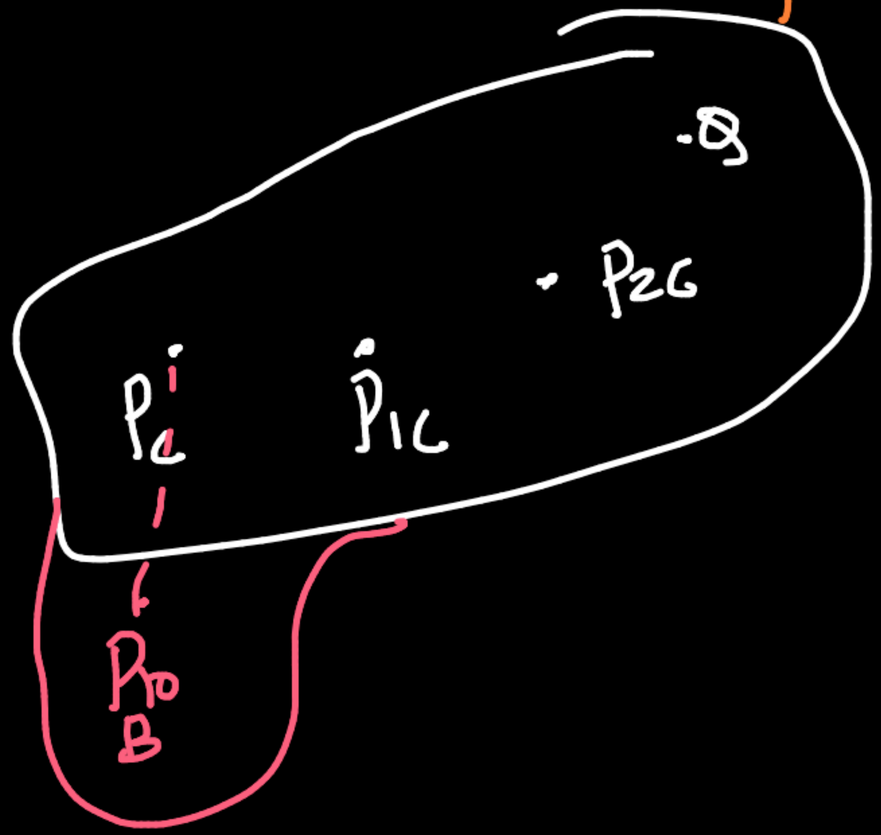
range queries

n · log n



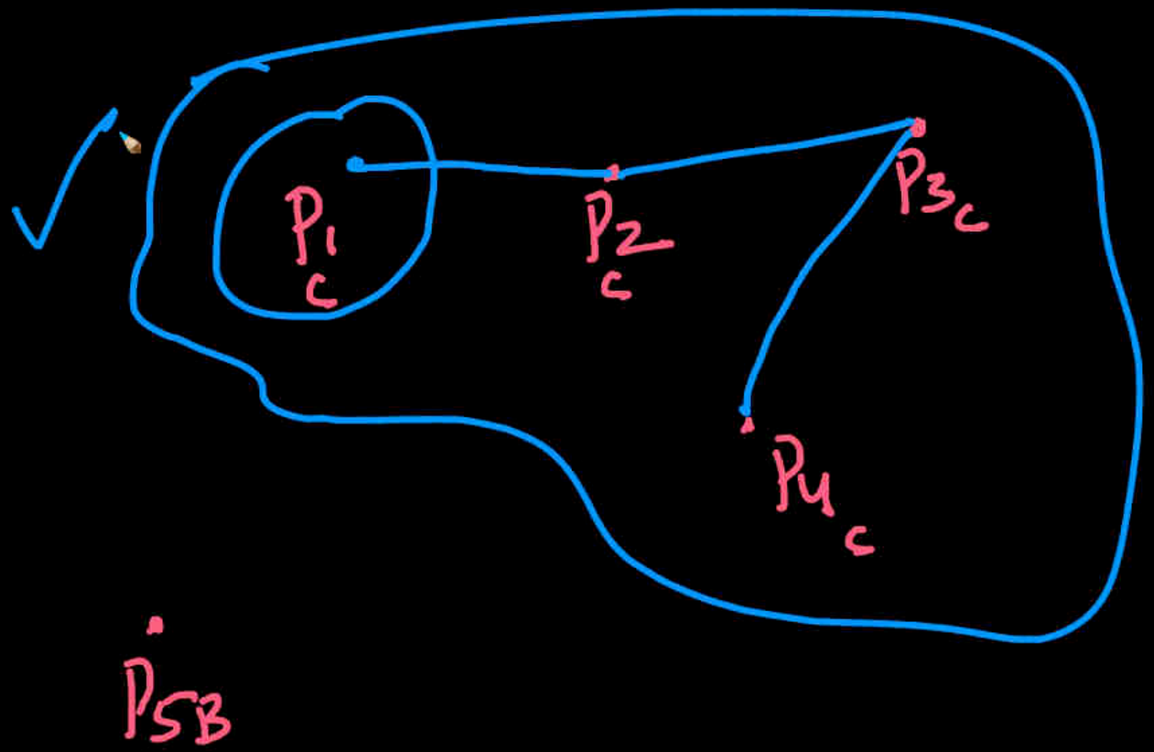
$n \lg n$ (4) ∀ border pt ; assign it to the nearest core-pts Cluster

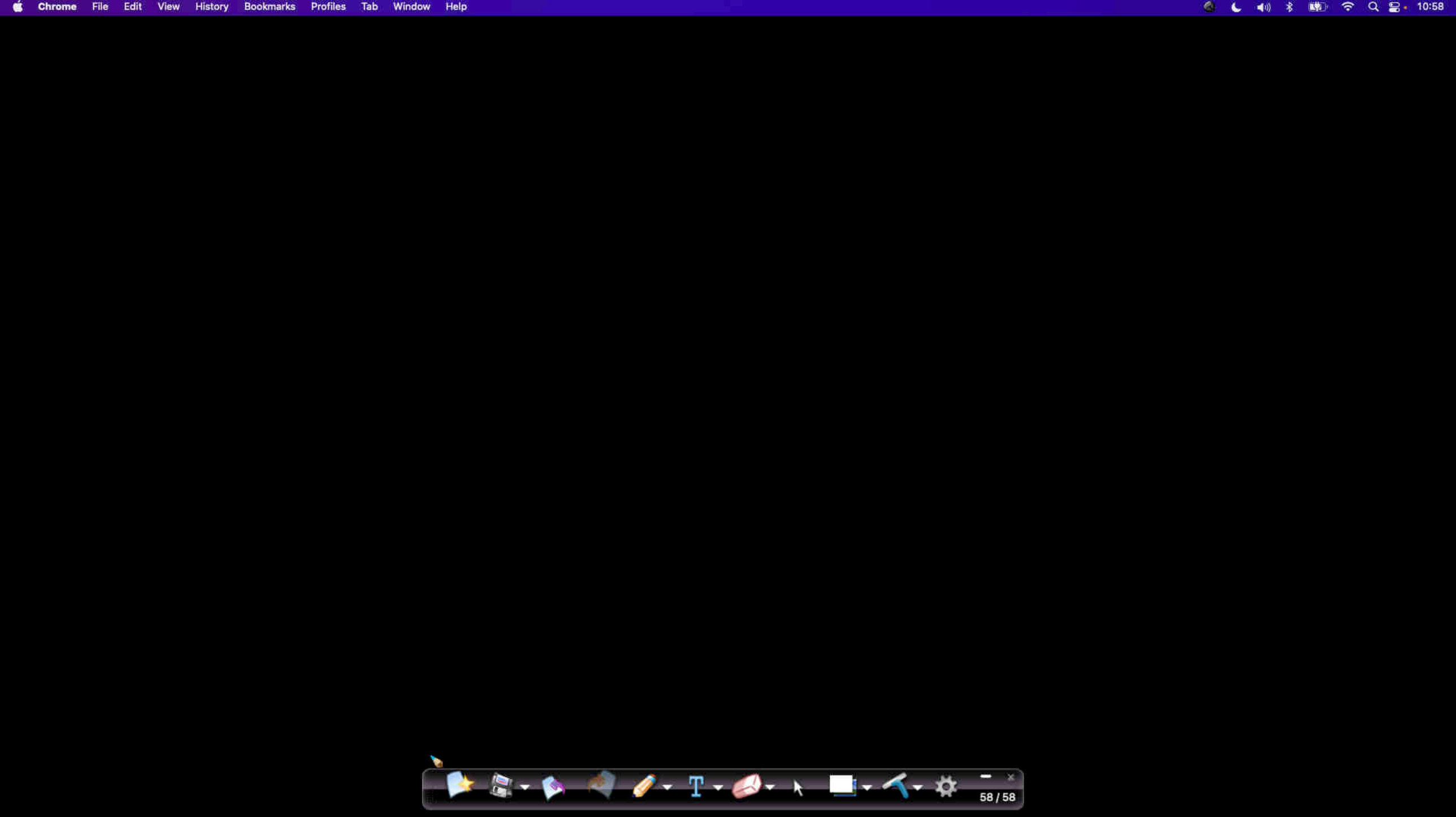
rangeQueries



Time-complx: $O(n \lg n)$ worst-case

- not computationally cheaper than Agg-clustering
- noise Sensitive



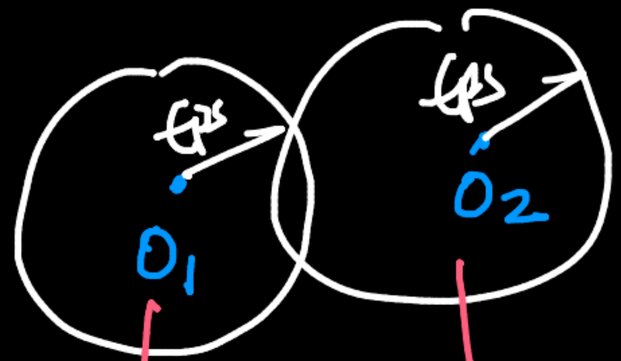

Chrome File Edit View History Bookmarks Profiles Tab Window Help
10:58
58 / 58

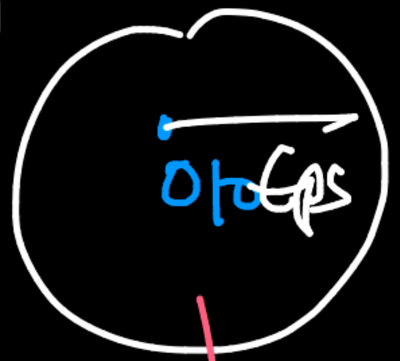
Minpts

1. rules of thumb:
   - Minpts $\geq d+1$ (d → dimensionality)
   - Typically:- Minpts $\approx 2 \times d$
2. dataset is noisy; fixed eps
   - larger Minpts

(c) Outliers using other methods : KNN

(hack)





3) + 1

Max (6 , 8 , . . .

↳ 8 + 1 = 9

Minpts

d

Various values of Minpts

→ different clusters

↓

domain expert

↓

[validation]

Eps:

MinPts = 4 (let) → k

step 1:

$x_1 \rightarrow d_1$

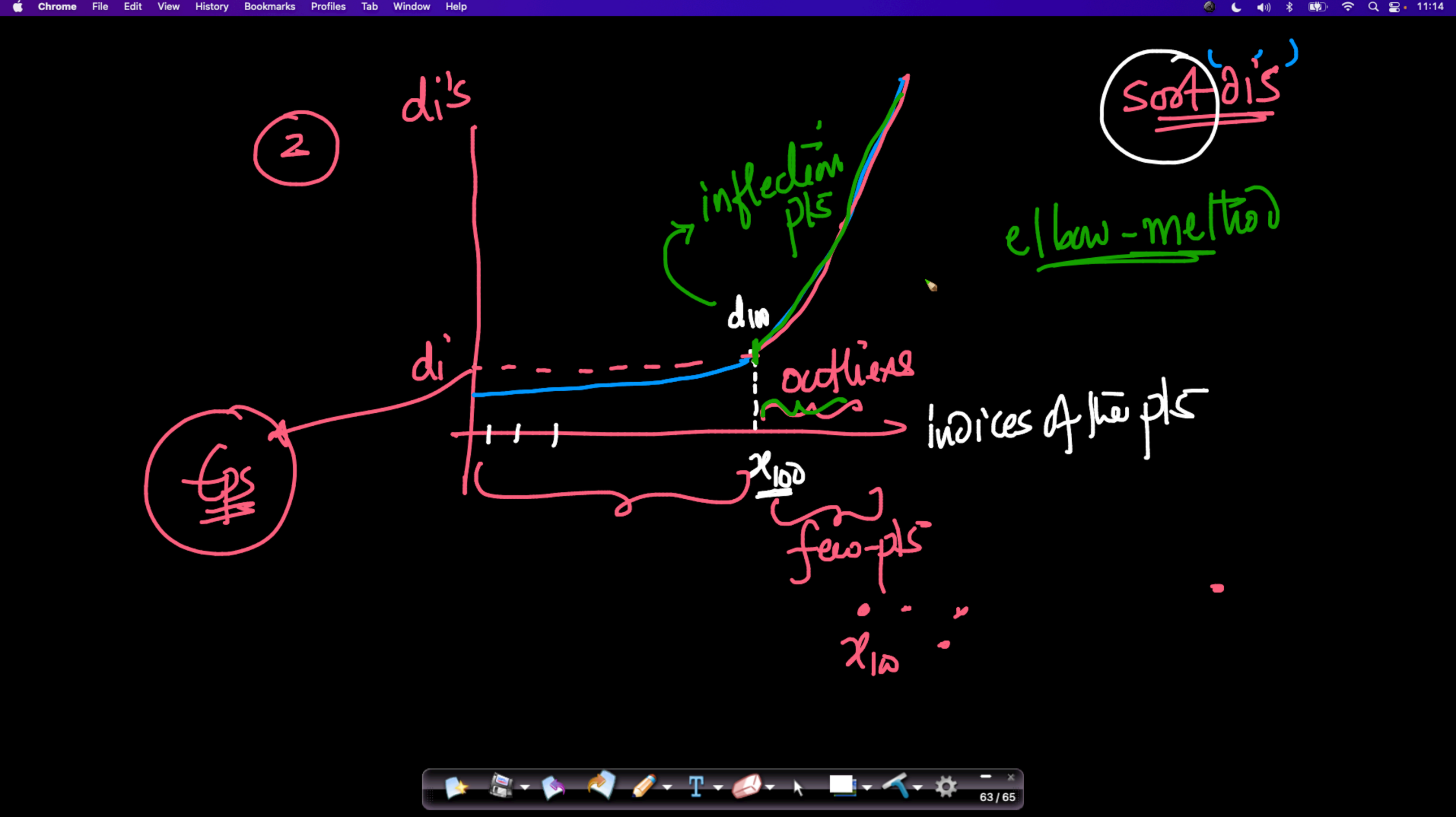
$x_2 \rightarrow d_2$

$\vdots$

$x_i \rightarrow d_i$ → distance from x_i to x_i's 4th NN

$\vdots$

$x_n \rightarrow d_n$


Chrome File Edit View History Bookmarks Profiles Tab Window Help
11:14
2
di's
inflection pts
dm
di
outliers
indices of the pts
x_{100}
feat-pts
x_{10}
Sort (di's)
elbow-method
63 / 65

C.q:

$x_1 \rightarrow 10$

$x_2 \rightarrow 6$

$x_3 \rightarrow 12$

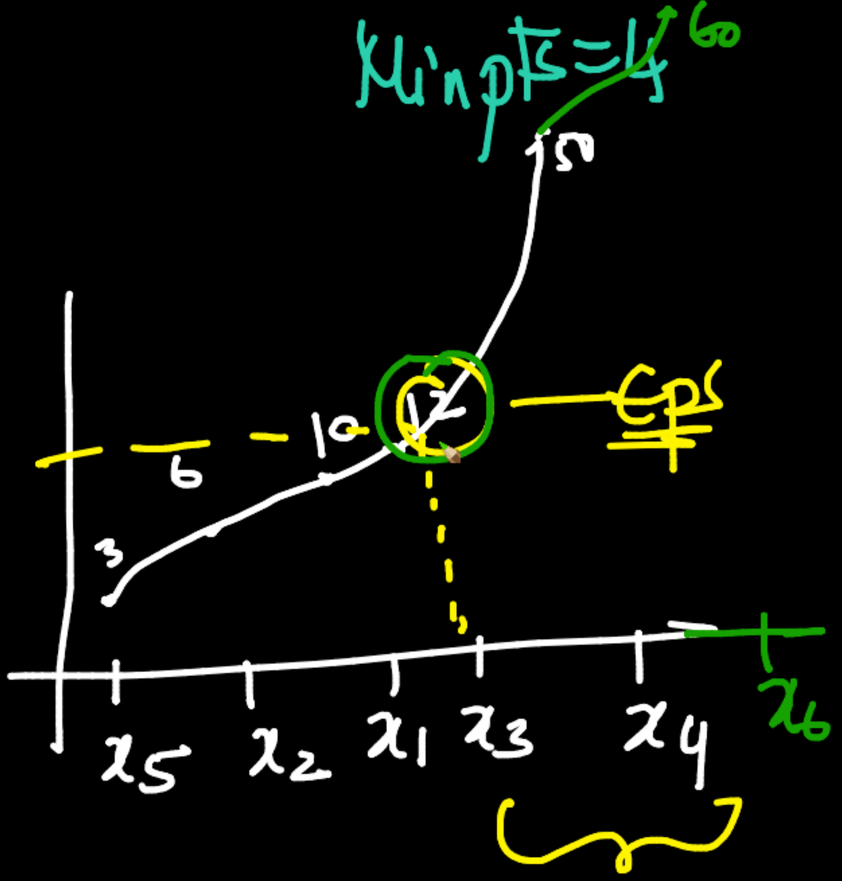
$x_4 \rightarrow 50$

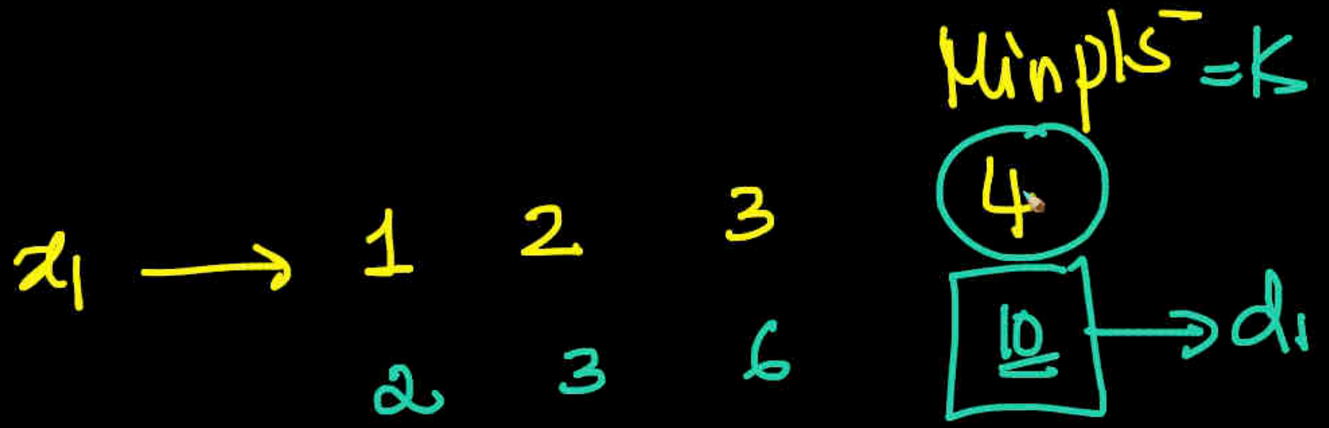
$x_5 \rightarrow 3$

$x_6 \rightarrow 60$

outlier

3, 6, 10, 12, 50



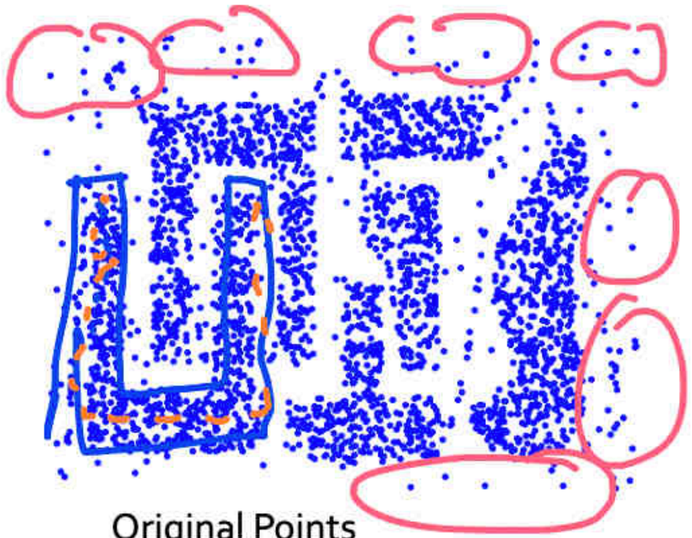


MinPts & Eps: hyper-param → Clusters → Domain expert

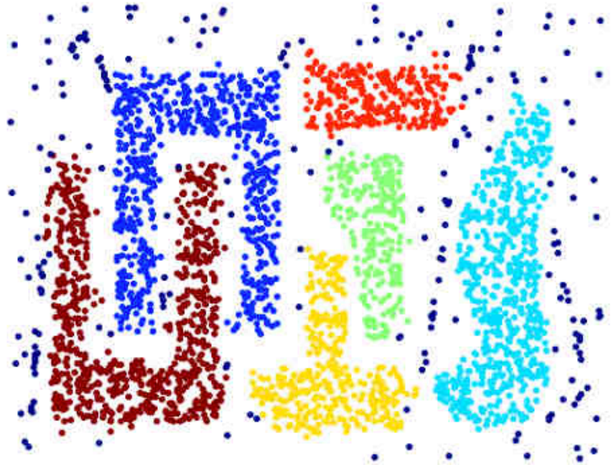
↓

logical hacks to arrive @ reasonable values (heuristics)

When DBSCAN Works Well



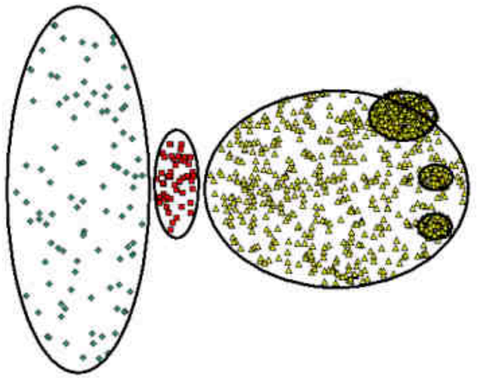
Original Points



Clusters

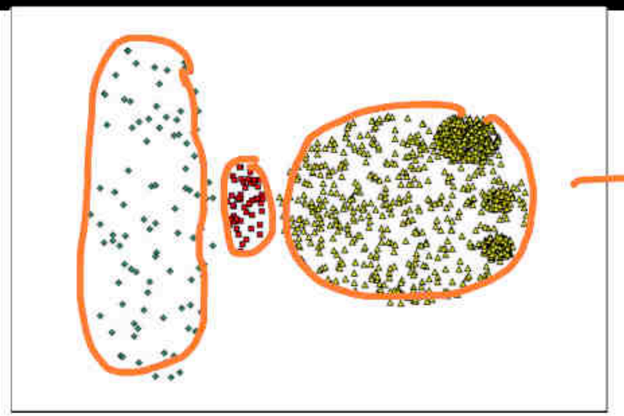
- Resistant to Noise
- Can handle clusters of different shapes and sizes

Well

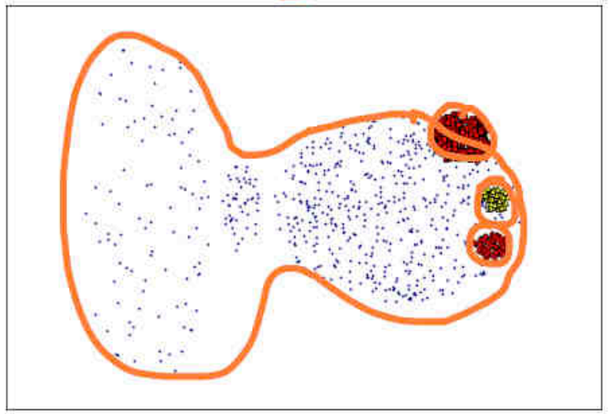


Original Points

- Varying densities
- High-dimensional data



(MinPts=4, Eps=9.75).



(MinPts=4, Eps=9.92)

decent

sensitive to params: Minpts & Eps

DBSCAN: Sensitive to Parameters

0.5 ← Eps → 0.4

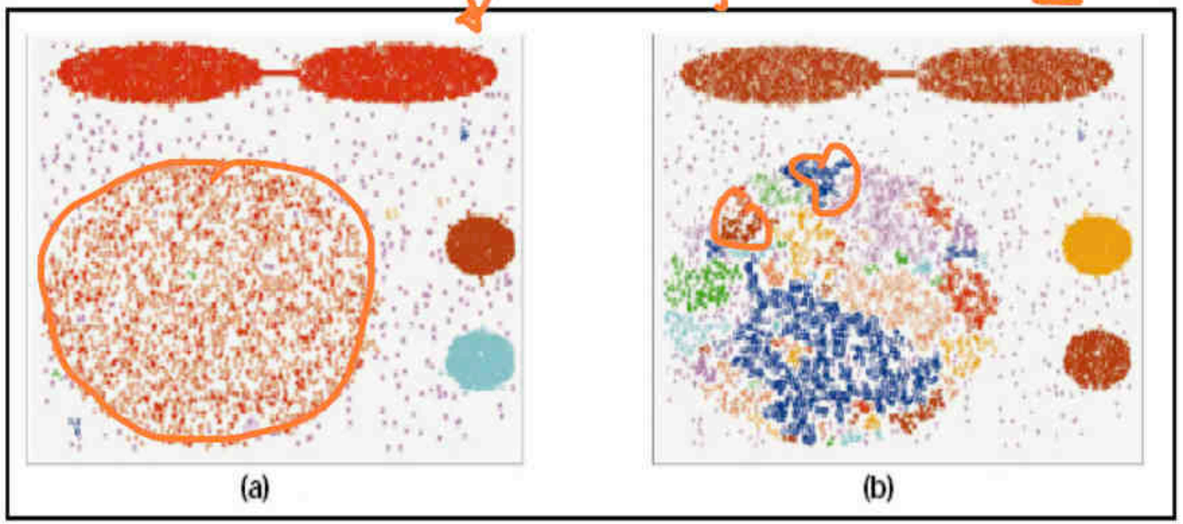
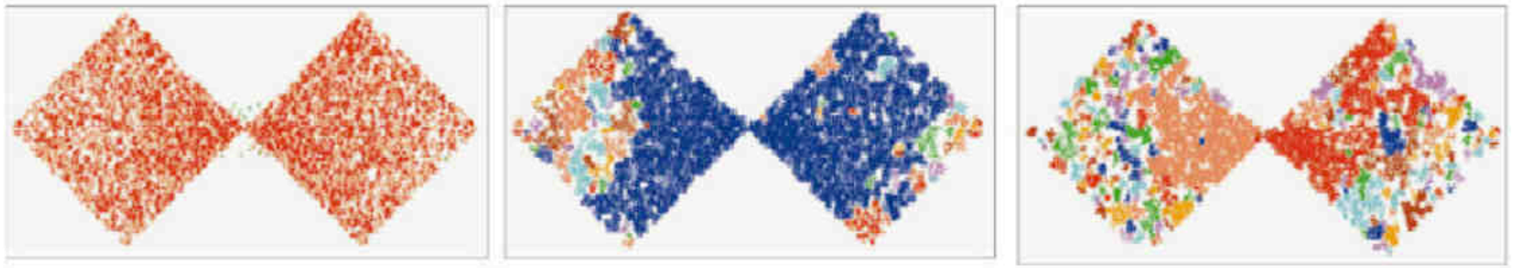


Figure 8. DBScan results for DS1 with MinPts at 4 and Eps at (a) 0.5 and (b) 0.4.

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



DBSCAN: Sensitive to Parameters

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

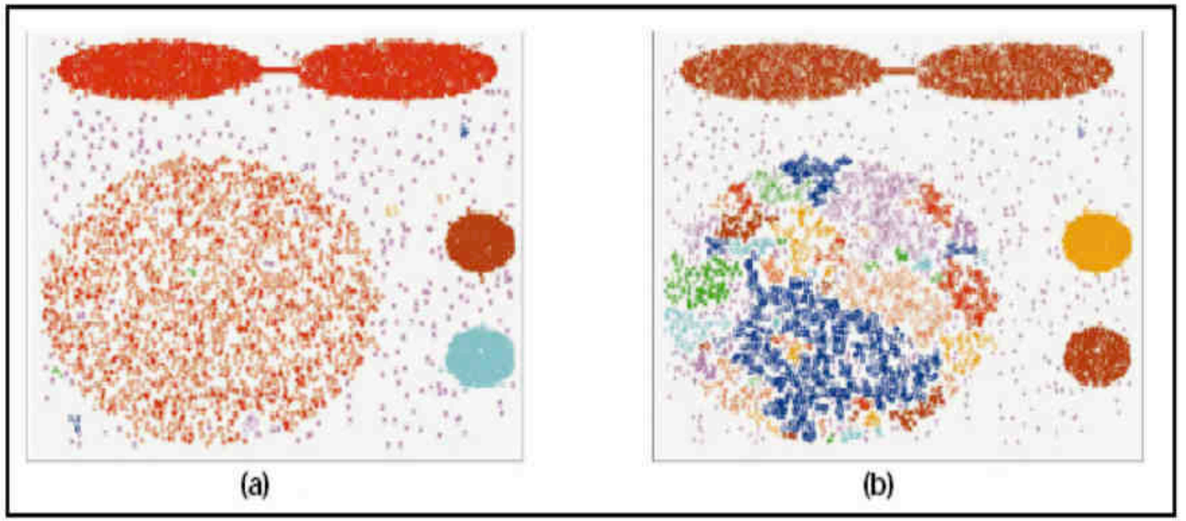
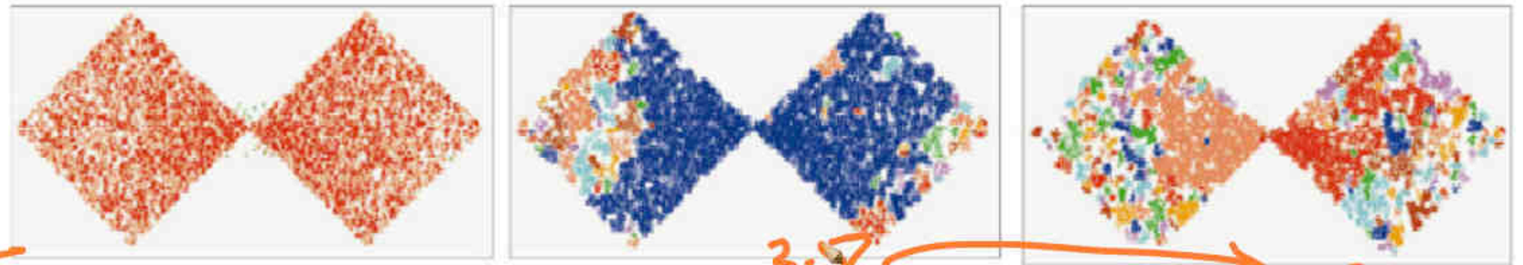


Figure 9. DBScan results for DS2 with MinPts at 4 and Eps at (a) 5.0, (b) 3.5, and (c) 3.0.



DBSCan

→ # clusters not needed like K-means

Adv

→ non-globular

→ robust to outliers

→ databases (large data) because of efficient range queries

→ Very sensitive to params: Minpts & Eps

disadv:

sklearn.cluster.DBSCAN

class `sklearn.cluster.`**`DBSCAN`**(*eps=0.5*, ***, *min_samples=5*, *metric='euclidean'*, *metric_params=None*, *algorithm='auto'*, *leaf_size=30*, *p=None*, *n_jobs=None*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

Parameters:

eps : *float, default=0.5*

The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

min_samples : *int, default=5*

The number of samples (or total weight) in a neighborhood for a point to be

scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
sklearn.cluster.DBSCAN
Examples using
sklearn.cluster.DBSCAN

metric is a string or callable, it must be one of the options allowed by `sklearn.metrics.pairwise_distances` for its metric parameter. If metric is "precomputed", X is assumed to be a distance matrix and must be square. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors for DBSCAN.

New in version 0.17: metric *precomputed* to accept precomputed sparse matrix.

metric_params : *dict, default=None*

Additional keyword arguments for the metric function.

New in version 0.19.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*

The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors. See NearestNeighbors module documentation for details.

leaf_size : *int, default=30*

Leaf size passed to BallTree or cKDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *float, default=None*

to calculate distance between points. If None, then p=2 (equivalent to the Euclidean distance)

scikit learn
Prev Up Next
scikit-learn 1.1.1
Other versions
Please cite us if you use the software.
sklearn.cluster.DBSCAN
Examples using
sklearn.cluster.DBSCAN

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*

The algorithm to be used by the NearestNeighbors module to compute pointwise distances and find nearest neighbors. See NearestNeighbors module documentation for details.

leaf_size : *int, default=30*

Leaf size passed to BallTree or cKDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *float, default=None*

The power of the Minkowski metric to be used to calculate distance between points. If None, then `p=2` (equivalent to the Euclidean distance).

n_jobs : *int, default=None*

The number of parallel jobs to run. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details.

Attributes:

core_sample_indices_ : *ndarray of shape (n_core_samples,)*

Indices of core samples.

components_ : *ndarray of shape (n_core_samples, n_features)*

Simple to understand → Agg-clustering & DBSCAN

↓

lots of terms / concepts

↳ revise (Feynman's tech & Spaced repitition)

sklearn.cluster.DBSCAN

class `sklearn.cluster.`**`DBSCAN`**(*eps=0.5*, ***, *min_samples=5*, *metric='euclidean'*, *metric_params=None*, *algorithm='auto'*, *leaf_size=30*, *p=None*, *n_jobs=None*) [source]

Perform DBSCAN clustering from vector array or distance matrix.

DBSCAN - Density-Based Spatial Clustering of Applications with Noise. Finds core samples of high density and expands clusters from them. Good for data which contains clusters of similar density.

Read more in the User Guide.

Parameters:

eps : *float, default=0.5*
The maximum distance between two samples for one to be considered as in the neighborhood of the other. This is not a maximum bound on the distances of points within a cluster. This is the most important DBSCAN parameter to choose appropriately for your data set and distance function.

min_samples : *int, default=5*


ghborhood for a point to be considered as a core point. This includes the point itself.