ECE 457B/657A: Data and Knowledge Modelling and Analysis

Unsupervised Learning and Clustering

Mark Crowley

Electrical and Computer Engineering

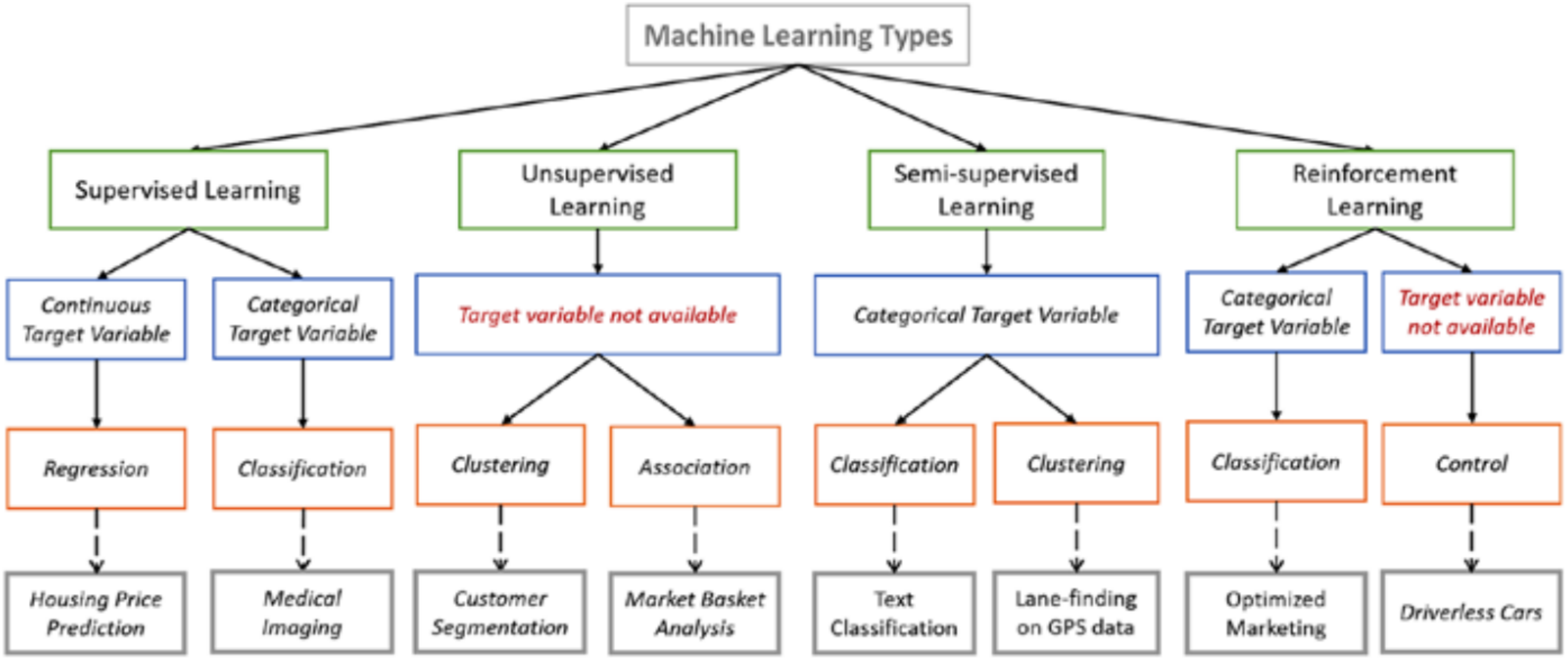
University of Waterloo

Where are we?

- **Supervised Learning:** $p(y|\boldsymbol{x}, \theta)$
- Learn from examples $\boldsymbol{x}$ to predict labels $\boldsymbol{y}$.
    - Prediction/Regression
    - Classification
- **Unsupervised Learning:** $p(\boldsymbol{x}, \theta)$
- Learn a possible distribution that generated data $\boldsymbol{x}$.
- Or find the "best" representation of the data automatically.
    - Density estimation
    - Synthesis
    - Denoising
    - Association Rule Mining
    - **Clustering**
- **People also talk about**
- *Semi-supervised Learning:* Only some datapoints have a label. learn distribution or classifier for remaining labels.
- *Multi-instance learning:* Set $\boldsymbol{X}$ contains label $Y = y$ but you don't know which datapoints.

None of these are rigid definitions, they are just useful categories for algorithms.

Overview of Machine Learning



Clustering Approaches

Hierarchical Approach

Single Link
Complete Link
Average Link
Ward's Algorithm

BIRCH
CURE
Chameleon

Partitioning Approach

K-means
PAM
Fuzzy C-means

CLARA
CLARANS

Density Based Approach

DBSCAN
DENCLUE

Others

NN
Evolutionary
Spectral
Kernel
Affinity Prop

Can handle larger data sets

Unsupervised Learning

- Find the "best" representation:
- A representation that preserves as much information about the data as possible
- Yet obeys given constraints
- Hopefully the representation is *simpler* than the data itself.

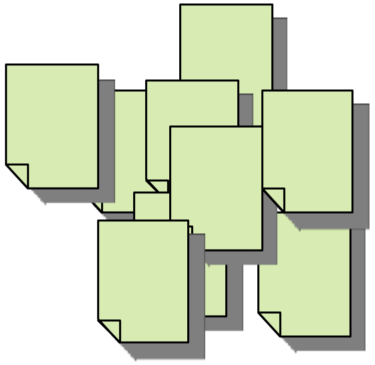
- Common ways to make a simpler representation:

  1. Dimensionality Reduction
  2. Sparse Representation
  3. Independent Representation

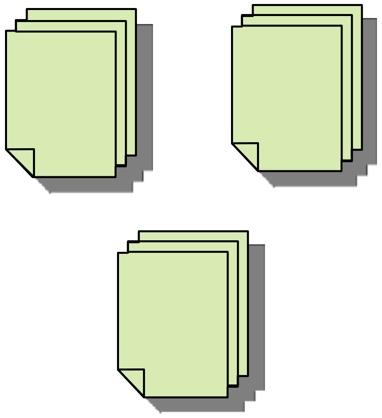
- These are not mutually exclusive.
- What methods have we seen so far to achieve some combination of these?

Data Clustering

- **Given**,
  - a set of data instances (e.g., documents, images)
  - a representation for data instances



- **Goal**: Divide data instances into groups
  - data instances into each group are similar
  - data instances into different groups are dissimilar



- **Constraint:** No prior knowledge of the labels or classes of the pattern (unsupervised)

Clustering vs. Classification

Classification

- Supervised
- Uses labeled data
- Requires training phase
- Domain sensitive
- Easy to evaluate
- Examples: Naive Bayes, KNN, SVM, Decision Trees

Clustering

- Unsupervised
- Uses unlabeled data
- Organize patterns
- w.r.t. an optimization criteria
- Notion of similarity
- Hard to evaluate
- Example: K- means, Fuzzy C- means, Hierarchical

Why Data Clustering?

- Discover natural groups in the data based on the similarity between data instances
  - Identify the groups of customers based on their purchase history
  - Identify the groups of users in a social network based on their connections
  - Identify the groups of proteins based on their structures
- Usually used for preliminary domain exploration / exploratory data analytics
- K-means is the most commonly used algorithm for data clustering

Other Applications of data clustering

- Data analysis
- Bioinformatics data
- Image segmentation
- GIS and spatial data
- Data and web mining
- Document clustering
- Medical data
- Economic and financial data

Issues

- How should we represent the patterns?
- What should we use as a similarity measure?
- How to decide on the number of clusters?
- How to actually group the patterns, i.e. what is the clustering criteria?
- How should we assess the results (performance measures)?
- How to interpret the resulting clusters

Notation

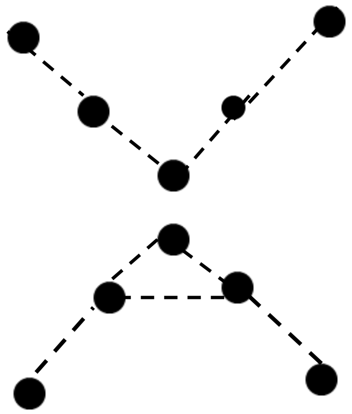
- Patterns can be represented as multidimensional vectors, each dimension is a single feature
- $x = \{f_1, f_2, \ldots, f_d\}$
- Clusters are sets: $C_1 = \{x_1, x_4, x_5\}, C_2 = \{x_2, x_3\}$
- Pair of points: $p, p' \in C_i$ in a cluster *I*
- Distance between points $d(p, p')$ or clusters $d(C_i, C_j)$ describes some measure of dissimilarity between them

Representation

- The features can be quantitative or qualitative (qualitative can be converted to quantitative by the appropriate measures)
- Other representations include structures (tree like) or symbolic objects (logical conjunction of events)

Similarity Measures

- Distance can be used as a measure of dissimilarity (review what we talked about back on distance measures).
- There are other applications where similarity could be defined based on other concepts than distance or proximity
- e.g. conceptual similarity or density



Points could be close in Euclidean distance but far in the model

Clustering Criteria

- Objective or cost function or rule to be evaluated during the process of clustering

Examples:

- Min sum of squared errors from reference or statistics (means, modes)
- Min intra-cluster distances
- Max inter-cluster distances
- Max number of common neighbors
- Distance Density

Clustering Approaches

Hierarchical Approach

Single Link
Complete Link
Average Link
Ward's Algorithm

Partitioning Approach

K-means
PAM
Fuzzy C-means

Density Based Approach

DBSCAN
DENCLUE

Others

NN
Evolutionary
Spectral
Kernel
Affinity Prop

BIRCH
CURE
Chameleon

CLARA
CLARANS

Can handle larger data sets

Hierarchical Approach

Use nested sets of clusters

- **Agglomerative:** start with each pattern in one cluster and merge close clusters until stopping criteria is satisfied
- **Divisive:** start with all patterns in one cluster and keep on dividing the dissimilar ones

- Set of points

Agglomerative Algorithms

- From current clusters, merge the two clusters that have the minimum distance between the clusters
  - But how do you define that?

Options 1: Single Link

- The distance between 2 clusters is the **minimum** of the distance between all pairs of patterns (one from each; similar to NN algorithm).

$$d(C_i, C_j) = \min d(p, p')$$

$$p \in C_i \, , p' \in C_j$$

Agglomerative Algorithms

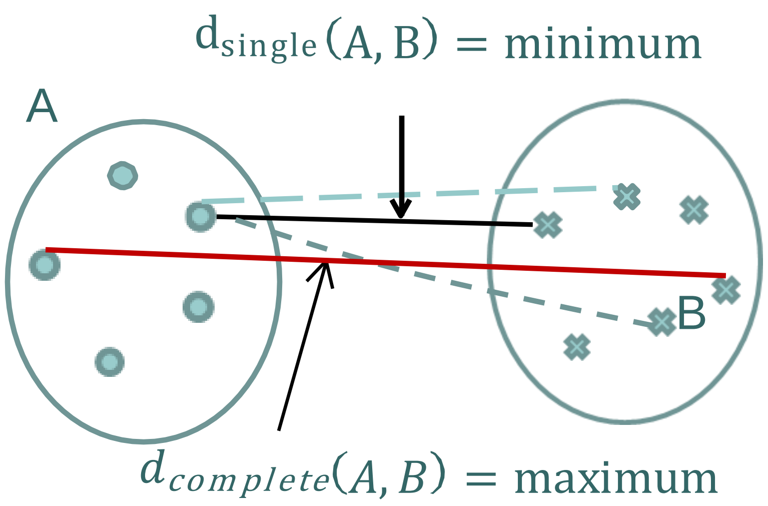
Option 2: Complete Link

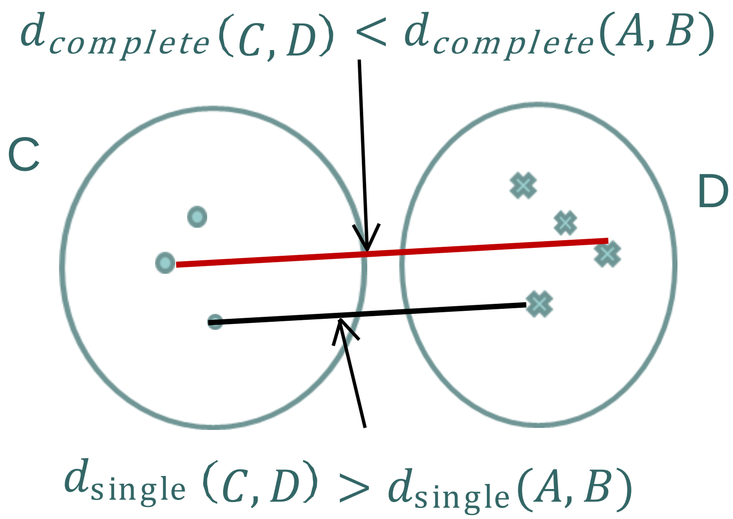
- The distance between 2 clusters is the maximum of all pairwise distance between the patterns

$$d(C_i, C_j) = \max d(p, p')$$

$$\text{For } p \in C_i, p' \in C_j$$

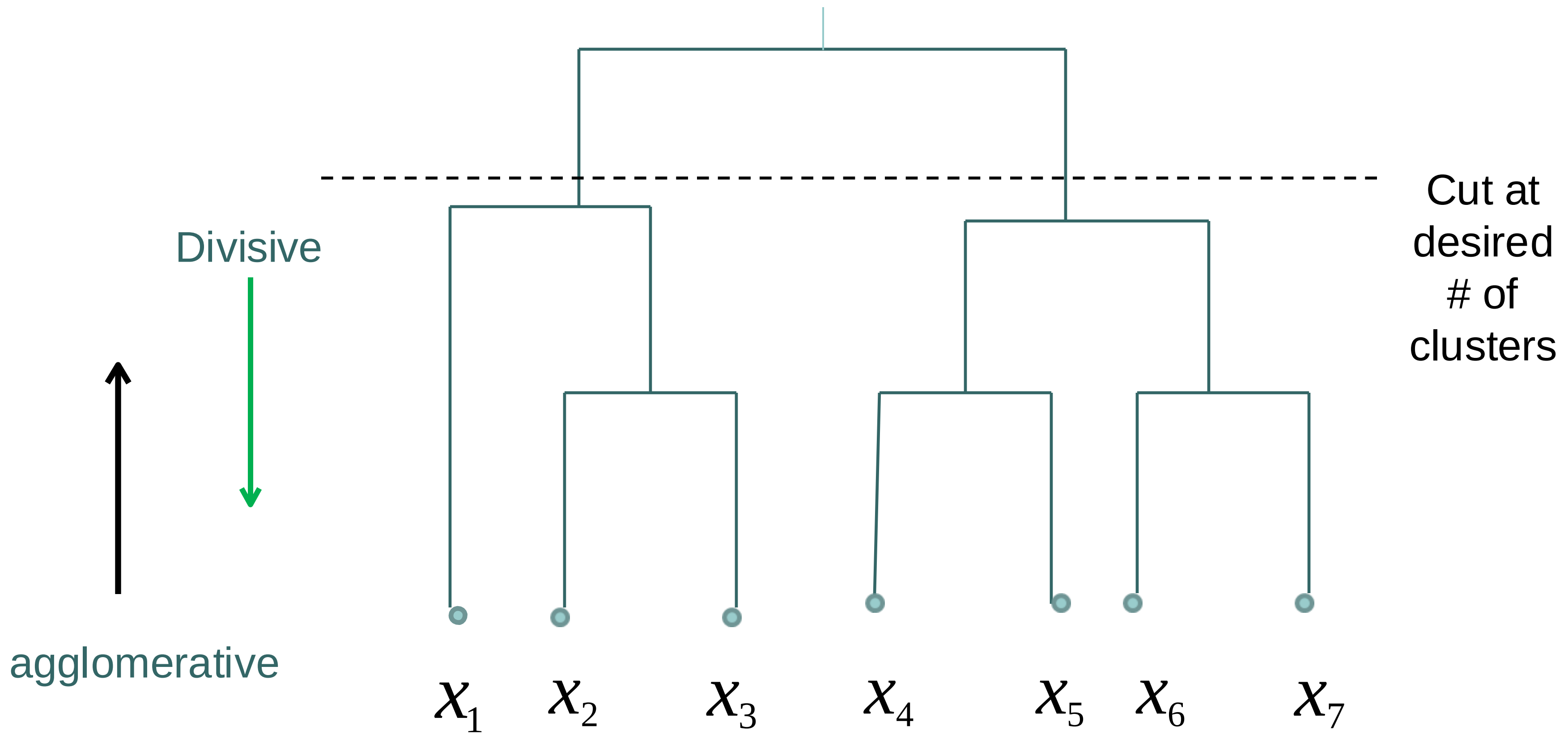
Single and complete link


$d_{single}(A, B) = \text{minimum}$
A
B
$d_{complete}(A, B) = \text{maximum}$


$d_{complete}(C, D) < d_{complete}(A, B)$
C
D
$d_{single}(C, D) > d_{single}(A, B)$

Single link algorithm will select A,B to merge
Complete Link will select C,D to merge

Hierarchal algorithms produce a Dendrogram



Merging or splitting Result in a Dendrogram
or Minimum Spanning Tree (MST)

Average Link

- Take average of distances.
- In both cases, the merge is done based on minimum distance between clusters.

$$d(C_i, C_j) = \frac{1}{n_i n_j} \sum_{p \in C_i} \sum_{p' \in C_j} d(p, p')$$

Space and Time Complexity $= O(n^2)$
Proximity matrix may be accessed each iteration

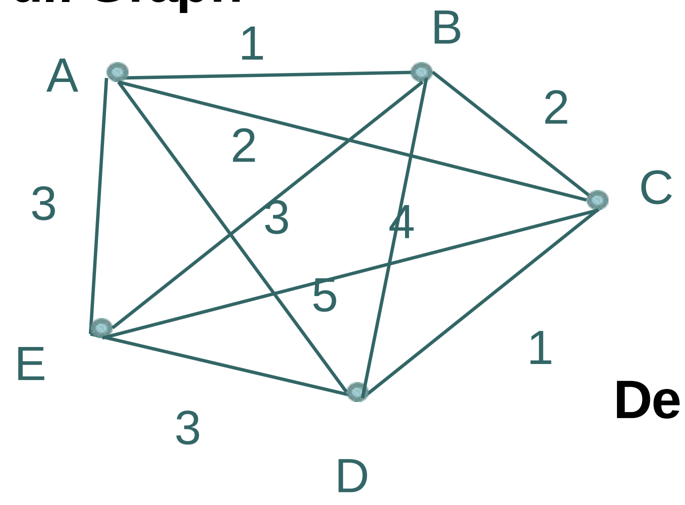
Divisive Algorithm

- **Reverse the process!**
- Splitt elements which are not sufficiently close.
- Use a threshold and minimum spanning tree (MST)
- One can look at it from a graph point of view (nodes are patterns, links are distances)
- Find the minimum spanning tree (corresponds to minimum distance), merge based on order of distances in MST

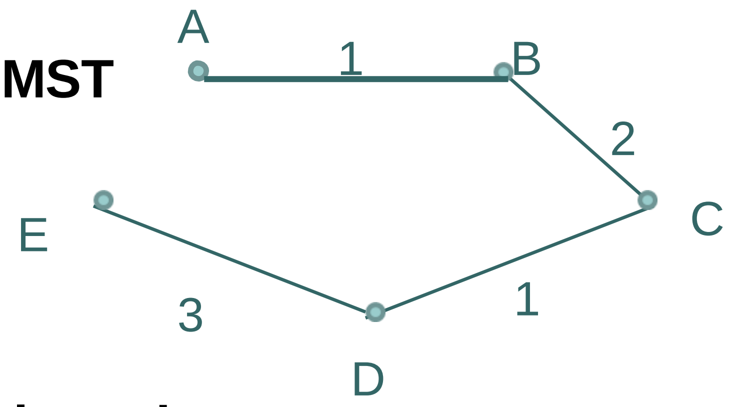
Single Link (agglomerative)

- If $\min d(p, p') < \epsilon$
  - the minimum distance between the points in the two clusters
  - is below the distance threshold
- Then merge the two clusters

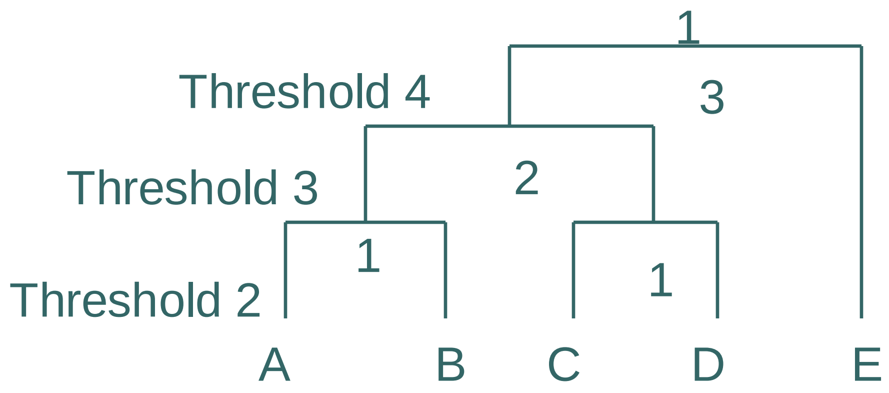
Full Graph



MST



Dendogram / Clustering



Single Link (divisive)

- We can use Minimum Spanning Tree (MST) to do a divisive version as well
- Start with one cluster {A,B,C, D,E}
- Cut the edge with the largest distance ED
- Now we have 2 clusters {E}, {A,B,C,D}
- Cut edge BC
- We have {A,B}, {C,D}, {E}
- Next {A}, {B}, {C}, {D}, {E}

Other agglomerative versions

- **Median linkage:** distance between clusters as distance between medians

$$d(C_i, C_j) = d(median_i, median_j)$$

- **Centroid linkage:** measure distance between clusters as distance between centroids (mean)

$$d(C_i, C_j) = d(m_i, m_j)$$

$$m_k = \frac{\sum_{l \in c_k} x_l}{n_k}$$

Ward’s Algorithm

- **Meta-algorithm** for agglomerative hierarchical clustering

- At each step, an objective function is used to determine if two clusters should be merged or not

Ward's Algorithm

More efficient: Instead of computing new distances between all new clusters just update the distance between the new cluster and other clusters

e.g. for single link merging of c_p, c_q into c_t

$$d(c_l, c_t) = \min(d(c_l, c_p), d(c_l, c_q))$$

c_l is the existing cluster

$\Rightarrow$

Update by deleting rows and columns corresponding to c_p, c_q and adding a row corresponding to the new cluster c_t

Ward's Minimum Variance Algorithm

- Min variance based – minimizes the total *within-cluster* variance (tight clusters)
- Computes the square of errors for each cluster (from the mean)

$$e_l^2 = \sum_{i=1}^{nl} \sum_{j=1}^{d} \left\| x_{ij}^l - m^l \right\|^2$$

- Total for all *k* clusters

$$E^2 = \sum_{l=1}^{k} e_l^2$$

- Merge pair of clusters p, q that minimize the change in total score: $\Delta E_{p,q}^2$

Ward's Minimum Variance Algorithm

- If merging *p, q* produce cluster *t*
- Then $\Delta E^2_{p,q} = e^2_t - e^2_p - e^2_q$
- With some algebra we get

$$\Delta E^2_{p,q} = \frac{n_p n_q}{n_p + n_q} \sum_{j=1}^{d} \left\| m_j^{(p)} - m_j^{(q)} \right\|^2$$

Problems with Hierarchical Clustering

Hierarchical Clustering Is sometimes called **connectivity based clustering**

Most common criticism of the previous HC algorithms is lack of robustness, and hence sensitivity to noise, outliers, and their complexity

$$O\left(n^2\right) \text{ in time and space}$$

With the need to handle large data some new algorithms have been developed

Clustering Approaches

Hierarchical Approach

Single Link
Complete Link
Average Link
Ward's Algorithm

Partitioning Approach

K-means
PAM
Fuzzy C-means

Density Based Approach

DBSCAN
DENCLUE

Others

NN
Evolutionary
Spectral
Kernel
Affinity Prop

BIRCH
CURE
Chameleon

CLARA
CLARANS

Can handle larger data sets

Partitional Approach

- Obtain a single partition of the patterns that optimizes a criterion function
- One common criterion is the sum of squared errors between patterns and cluster centroids

$$\text{minimize } J = \sum_{j=1}^{k} \sum_{x \in c_j} \left\| x - m_j \right\|^2 = \sum_{j=1}^{k} \sum_{x \in c_j} d^2$$

$$m_j = \frac{1}{n_j} \sum_{x \in c_j} x$$

$$C_j = \text{set of patterns in cluster } j$$

$$n_j = \text{number of patterns in } \mathrm{C_j}$$

Partitional Approach

- Tough optimization problem
  - Discrete objective function
  - non-convex
- Try all possible partitions, evaluate *J* and select the one that minimizes *J*
- Theoretically it is easy, but not practical

$$N(n,k) = \left(\frac{1}{k_1}\right) \sum_{m=1}^{k} (-1)^{k-m} \binom{k}{m} m^n$$

$$\begin{aligned} &For\ n = 5 && k = 2 && 15\ partitions \\ &\quad n = 10 && k = 4 && 34,105 \\ &For\ n = 50 && k = 4 && partition = 5.3 \times 10^{28} \\ &\quad \mathrm{n} = 100 && k = 4 && N = 6.7 \times 10^{58} \end{aligned}$$

Solution: approximate iterative solutions

K-means Clustering Algorithm

Most commonly used clustering algorithm in this type

Init: Choose K cluster centers $[m_1, \ldots, m_k]$ selecting K patterns randomly

1. Distribute patterns among clusters using similarity measure

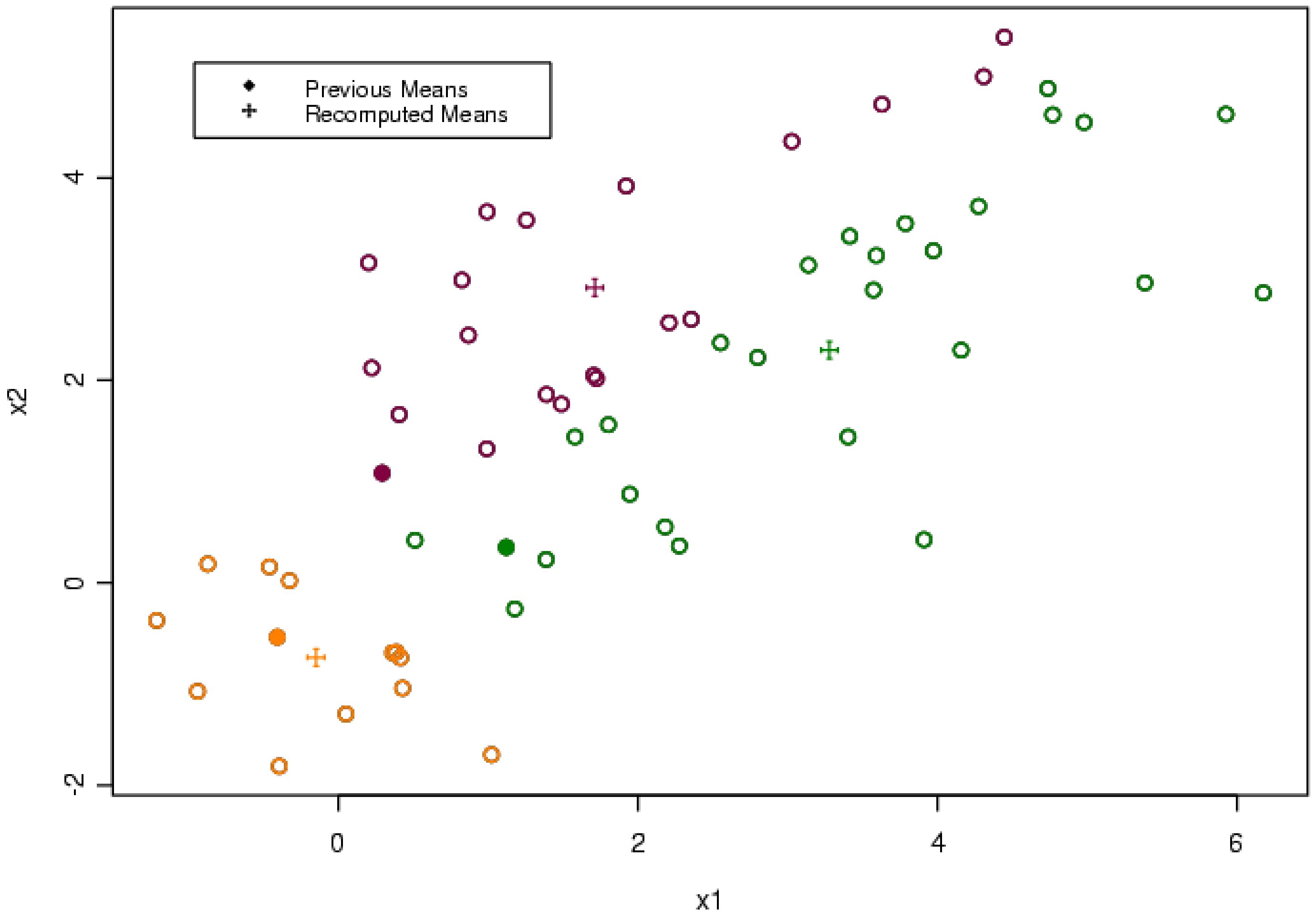
$$x_j \in c_w \text{ if } \|x_j - m_w\| < \|x_j - m_i\|$$

$$j = 1, \ldots, n, i \neq w, i = 1, \ldots, k$$

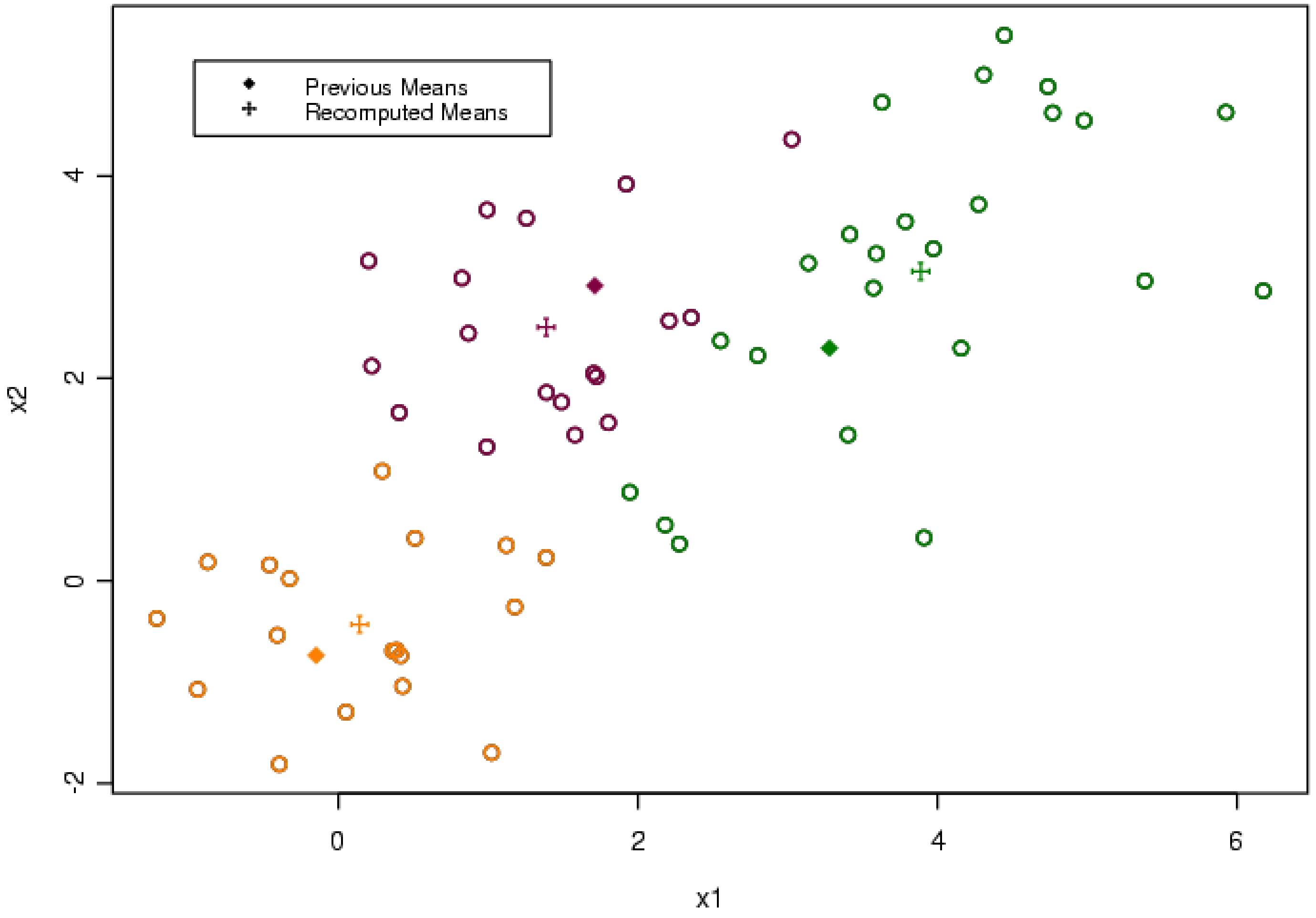
2. Compute new cluster centers using the current memberships

Repeat Until convergence

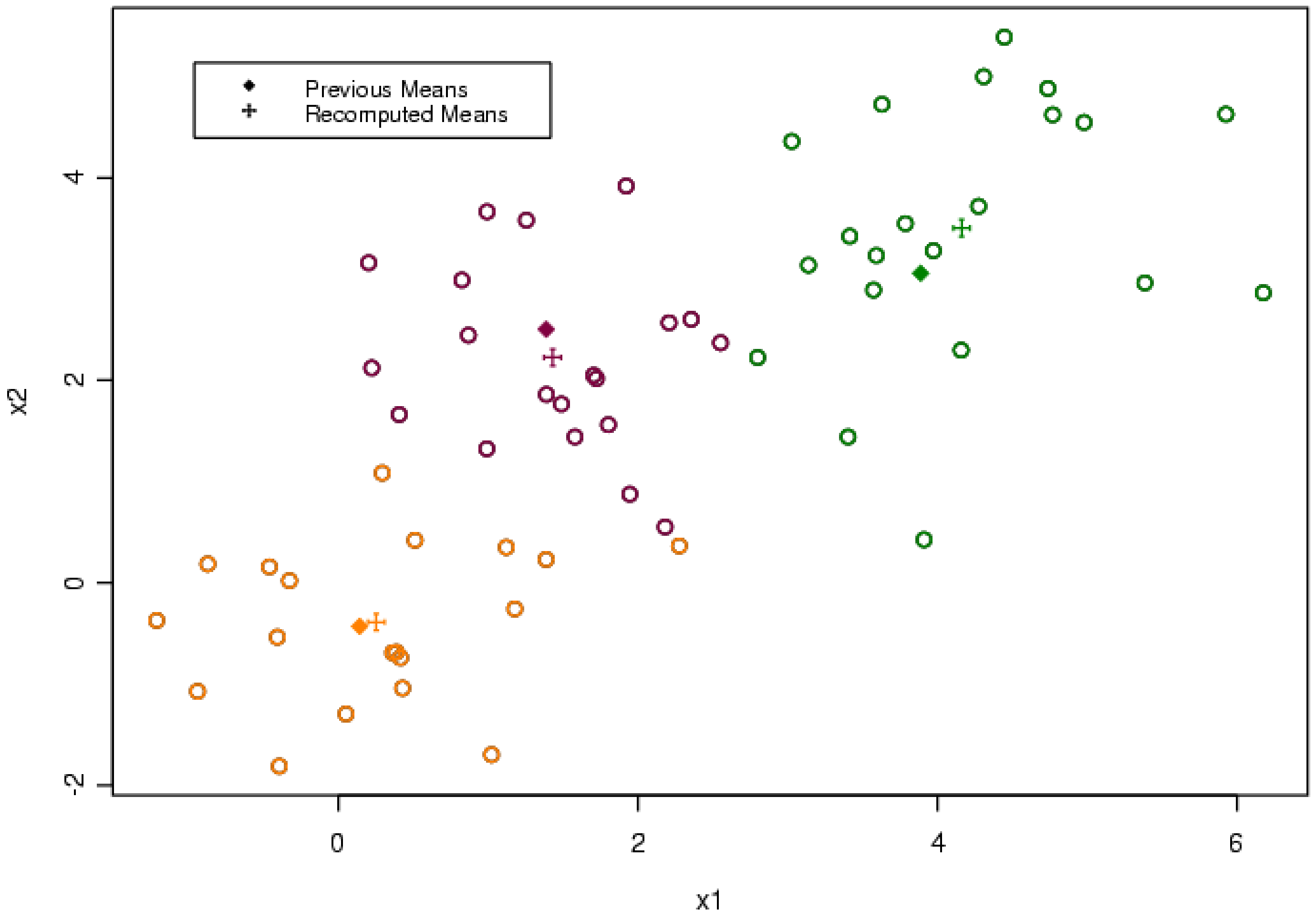
K-Means Example



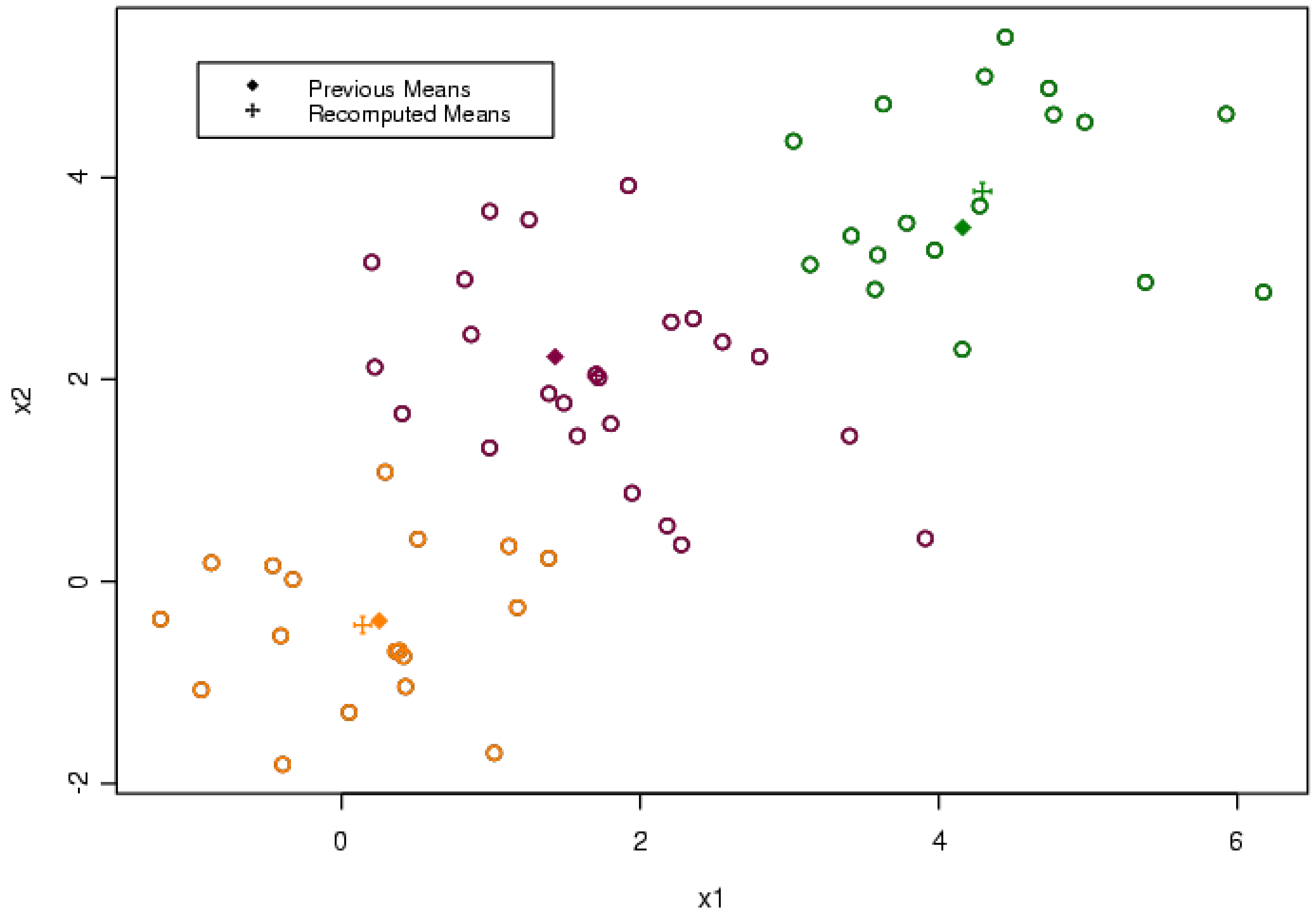
K-Means Example



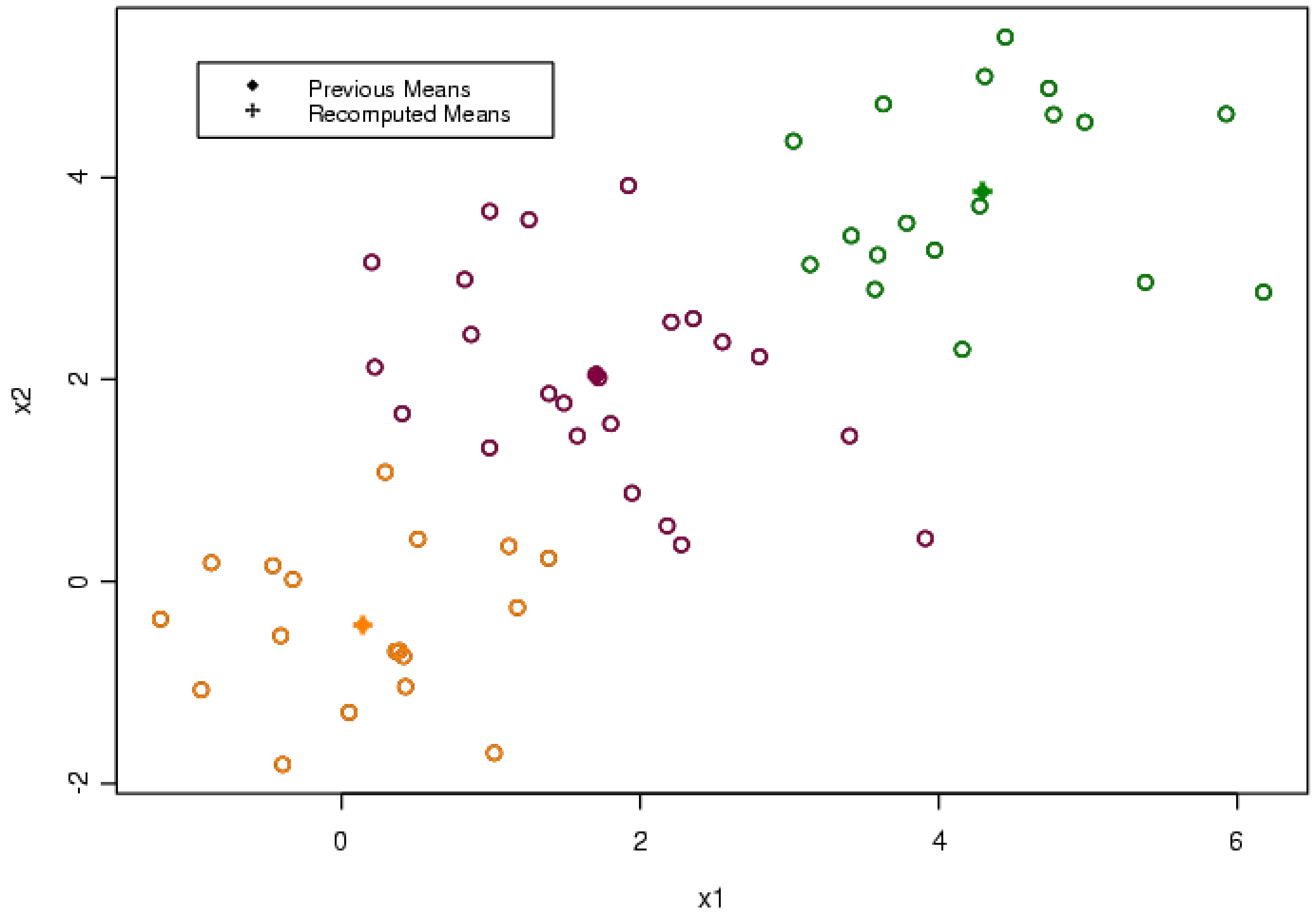
K-Means Example



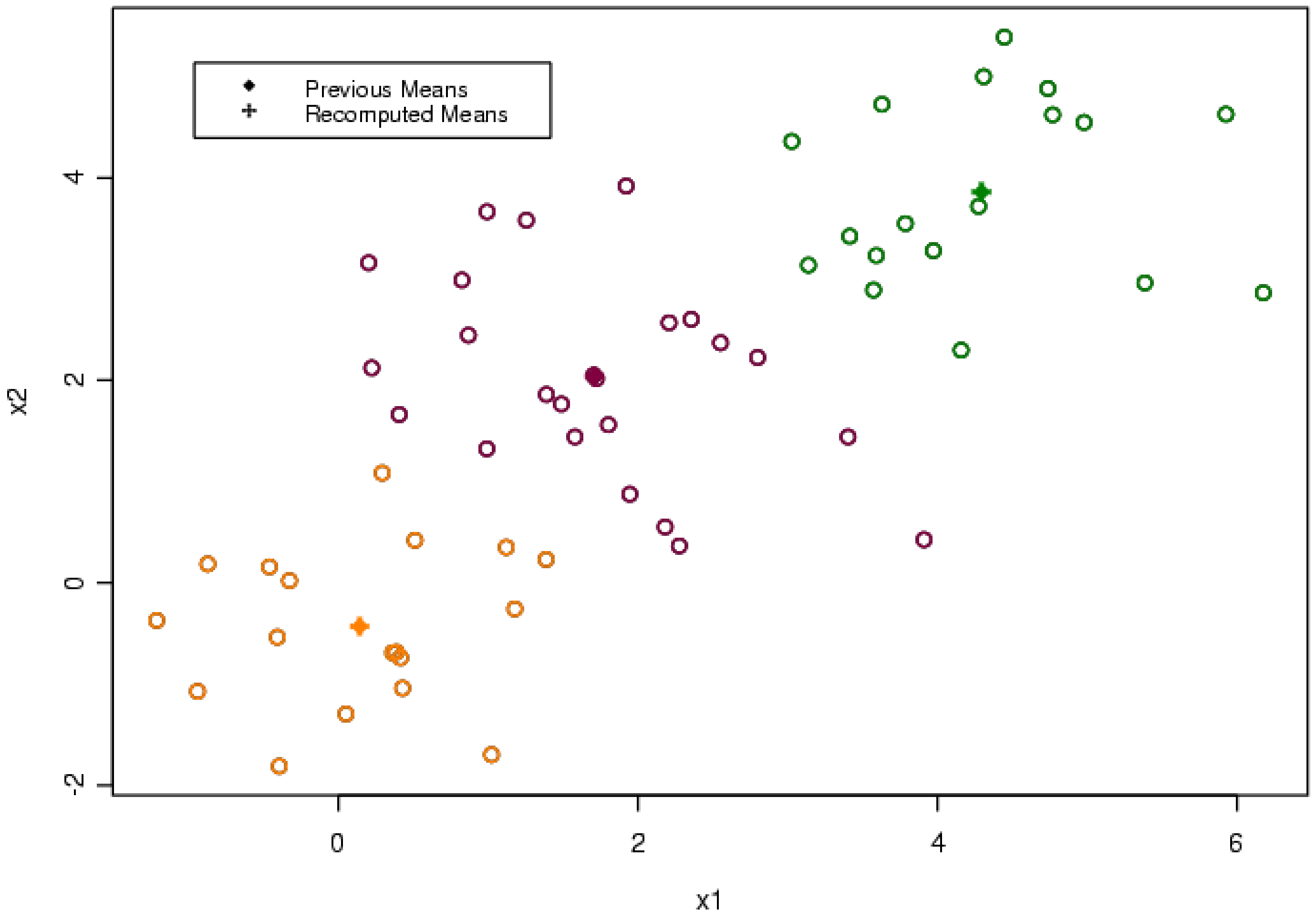
K-Means Example



K-Means Example



K-Means Example



What is Convergence?

Convergence can be:

1. No change in centers
2. Minimum decrease in objective function J
3. Minimum reassignment of patterns to new cluster centers

Other versions

- McQueen: Update centers after each pattern assignment.
- ISODATA: the resulting clusters are changed by splitting and merging.

K-means Clustering Algorithm

Sensitive to step 1, the choice of initial center

Complexity

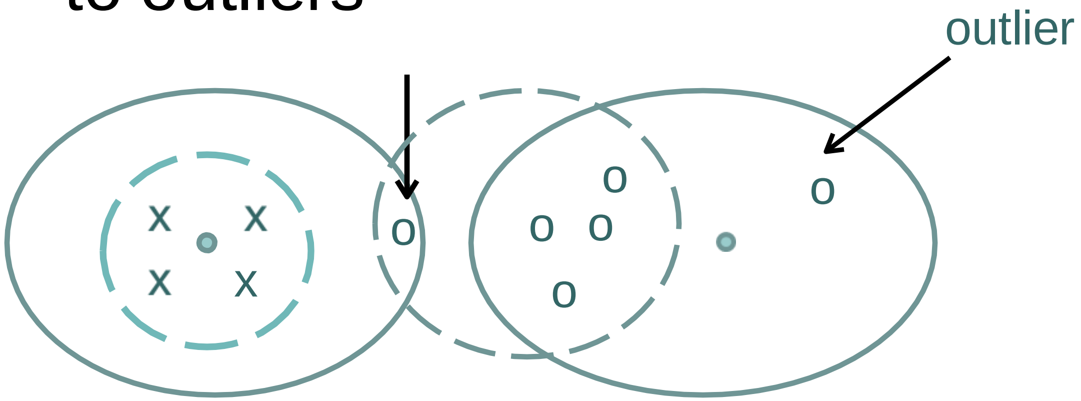
$$O\left(\underset{\substack{\uparrow \\ \text{iterations}}}{t}\ \underset{\substack{\uparrow \\ \text{clusters}}}{k}\ \underset{\substack{\uparrow \\ \text{patterns}}}{n}\right)$$

- It may find local optimum and may be stuck in a bad solution
- Does not work on categorical data due to use of mean
- Sensitive to outliers

K-means Clustering Algorithm

- There are variants of k-means, that permit merge and split of resulting clusters, or different use of objective function
- One variant that can handle categorical data is k-modes, it uses modes rather than mean

Sensitivity of k-means to outliers



K-mean

<u>Pros</u>

- Simple
- Fast for low dimensional data

<u>Cons</u>

- Can't handle non-globular clusters
- Will not identify outliers
    - Restricted to data which has notion of center
- May converge to a bad solution and produce empty clusters (only with centroids and no other members)

K-means Clustering as a Sparse Representation

- K-means divides data into k clusters of data points that are "near" each other.

- K-dimensional one-hot code :
- For $\boldsymbol{x_i} \in c$ then we have a vector $\boldsymbol{h_c} = 1$ and $\boldsymbol{h_d} = 0\ \forall d \neq c$
- So most entries of $\boldsymbol{h}$ are zero.

Clustering Approaches

Hierarchical Approach

Single Link
Complete Link
Average Link
Ward's Algorithm

BIRCH
CURE
Chameleon

Partitioning Approach

K-means
PAM
Fuzzy C-means

CLARA
CLARANS

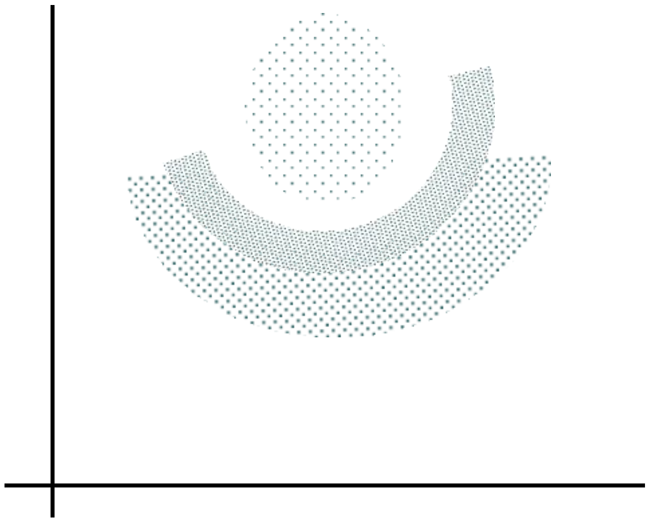
Density Based Approach

DBSCAN
DENCLUE

Others

NN
Evolutionary
Spectral
Kernel
Affinity Prop

Density Based Clustering



- ❑Group points based on their density
- ❑Handles outliers
- ❑Can detect clusters with arbitrary shapes and densities

`DBScan(X, ε, m)`: Density Based Spatial Clustering of Applications with Noise

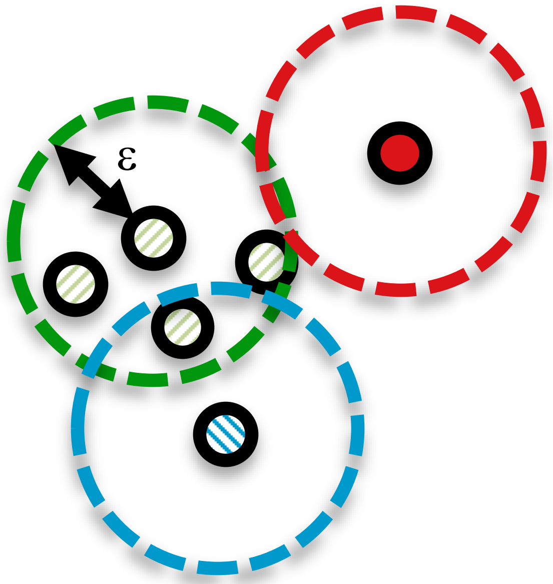
Dense Region = neighbourhood of distance ε contains at least ***m*** points

Classify the points according to their density:

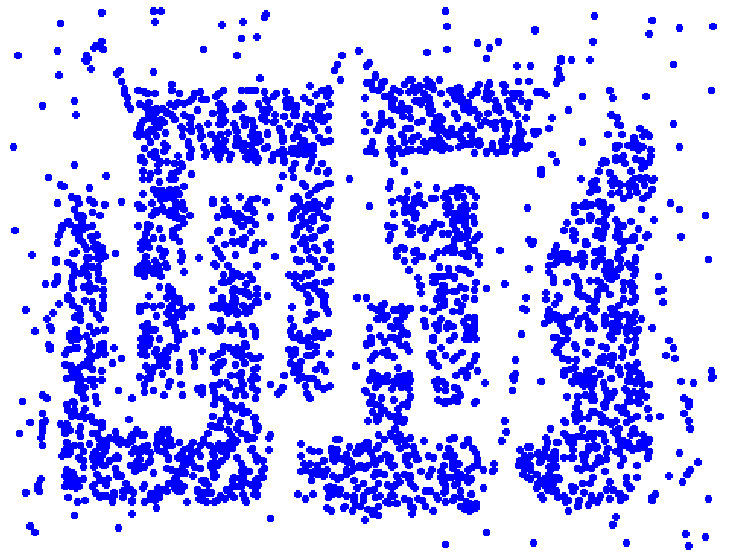
Core points : points in the interior of dense region.

Border points : points on the edge of dense region.

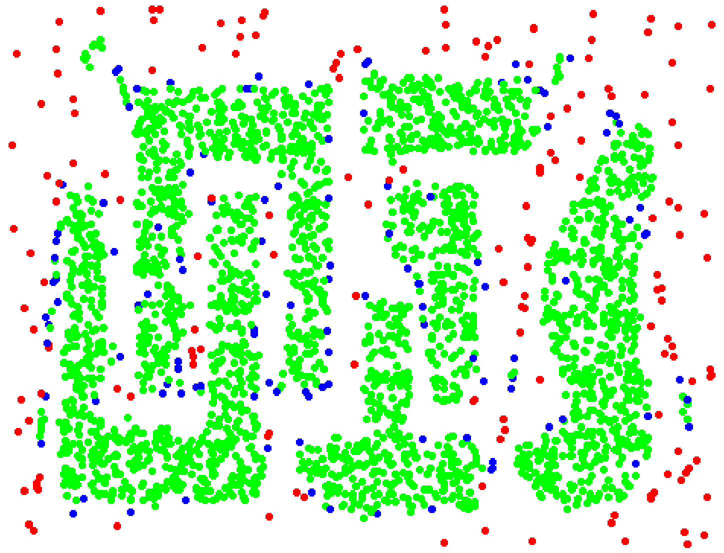
Noise points : points that are not border nor core.



DBSCAN: Core, Border and Noise Points



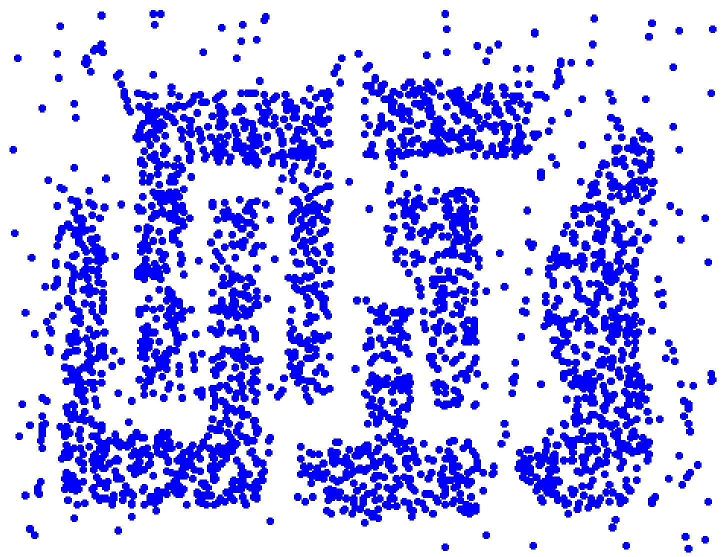
Original Points

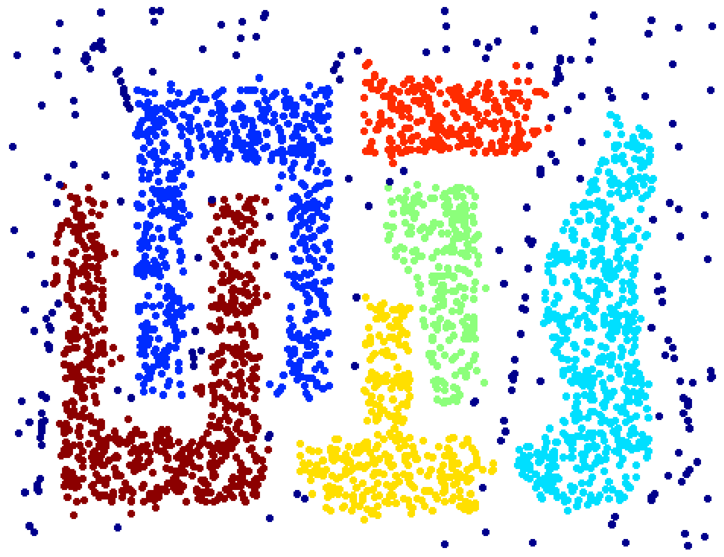


Point types: core, border and noise

Eps = 10, MinPts = 4

When DBSCAN Works Well


Original Points


Clusters

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

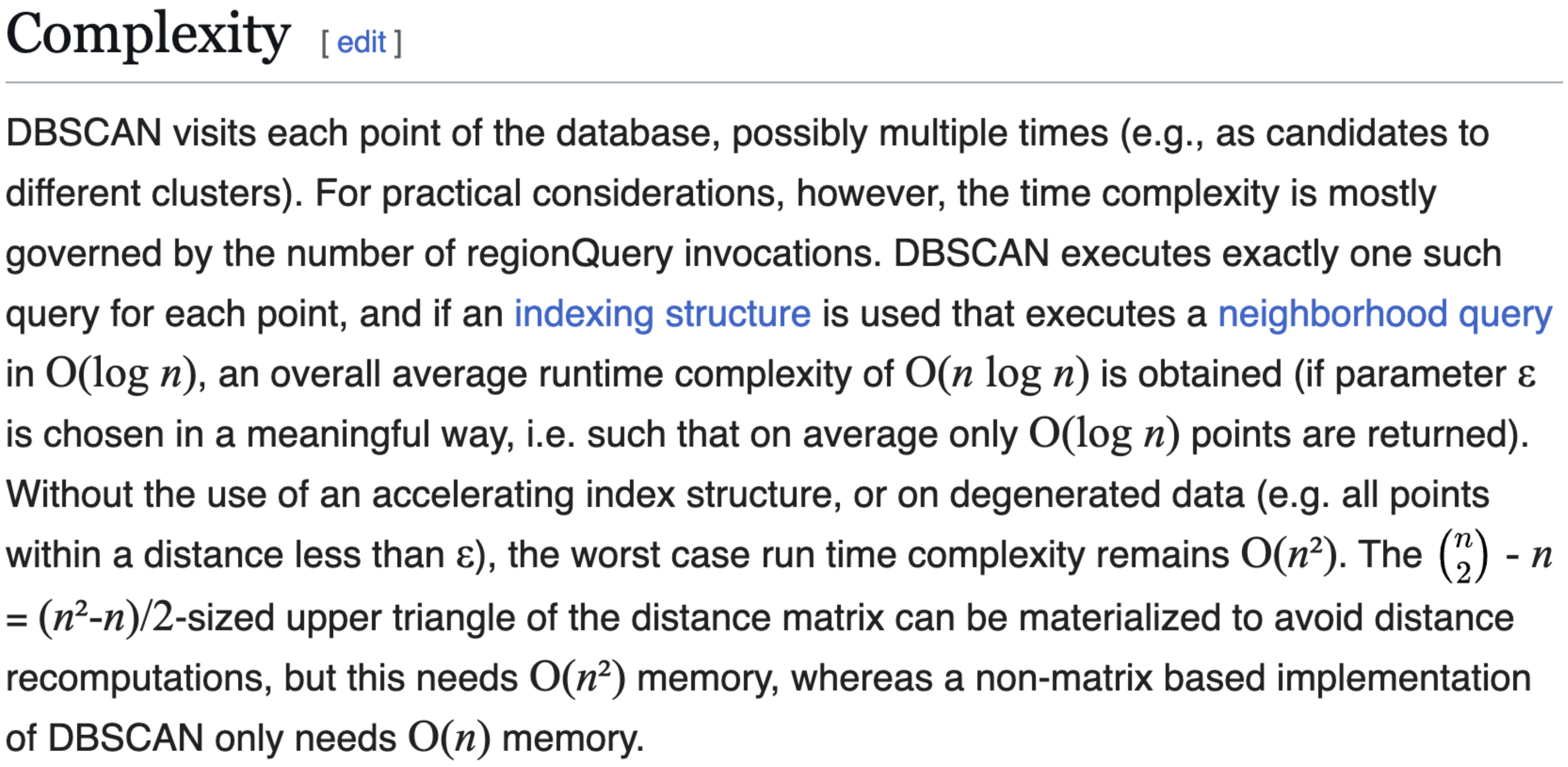
DB Scan General Algorithm – Iterative Form

1. m and ϵ eps are given
2. Select a random starting point
   - neighborhood area is determined using radius ϵ
   - If there are at least m number of points in the neighborhood, the point is marked as **core point** and *cluster formation starts (with cluster A for example)*
   - If not, the point is marked as **noise**.
     - *Noise points might later be added to a cluster or stay as noise.*
   - All the points within the neighborhood of initial point become a part of **cluster A**.
   - Then all those neighbour points can be considered and their neighbours found, and also added to cluster A
3. (eventually) Randomly select another point that has not yet been visited and repeat from step 2.
4. This process is finished when all points are visited.

DB Scan General Algorithm – Graph Approacg

1. *m* and ϵ eps are given
2. For all points:
   - neighborhood area is determined using radius ϵ
   - Find all neighbours for all points, if there are at least *m* number of points in the neighborhood, the point is marked as **core point** and *cluster formation starts (with cluster A for example)*
   - Build a neighbour graph of all points
3. Find the **connected components** of core points on the neighbour graph, ignoring all non-core points.
4. Assign each non-core point to a nearby cluster if the cluster is an ε (eps) neighbor, otherwise assign it to noise.

DBScan Complexity



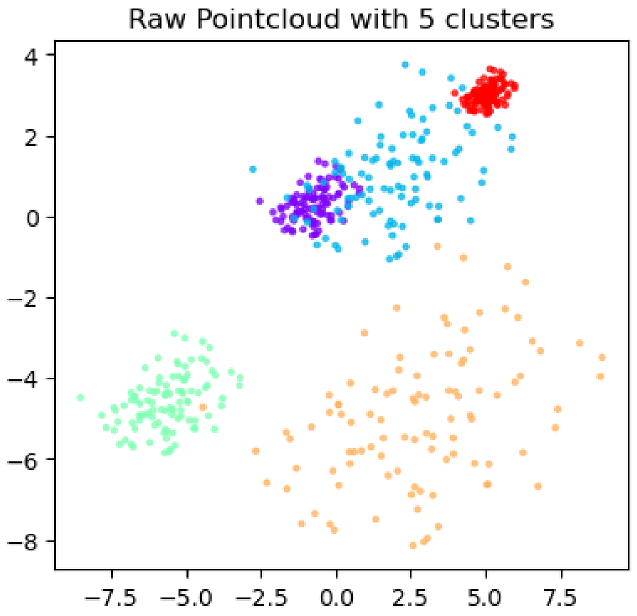
Complexity [edit]

DBSCAN visits each point of the database, possibly multiple times (e.g., as candidates to different clusters). For practical considerations, however, the time complexity is mostly governed by the number of regionQuery invocations. DBSCAN executes exactly one such query for each point, and if an indexing structure is used that executes a neighborhood query in $O(\log n)$, an overall average runtime complexity of $O(n \log n)$ is obtained (if parameter ε is chosen in a meaningful way, i.e. such that on average only $O(\log n)$ points are returned). Without the use of an accelerating index structure, or on degenerated data (e.g. all points within a distance less than ε), the worst case run time complexity remains $O(n^2)$. The $\binom{n}{2}$ - n = $(n^2-n)/2$-sized upper triangle of the distance matrix can be materialized to avoid distance recomputations, but this needs $O(n^2)$ memory, whereas a non-matrix based implementation of DBSCAN only needs $O(n)$ memory.

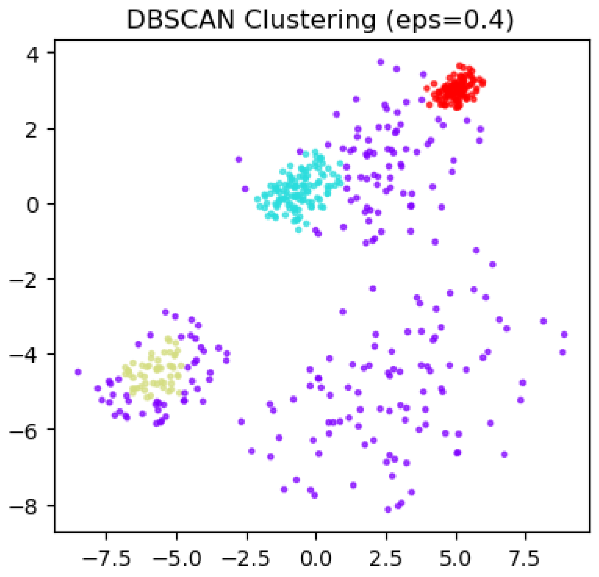
- From Wikipedia

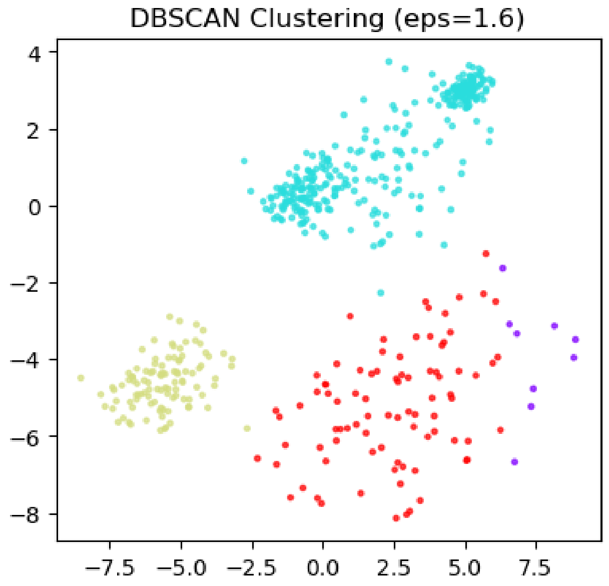
When DBSCAN Does NOT Work Well

- High-dimensional data
- **Overlapping clusters with varying densities**



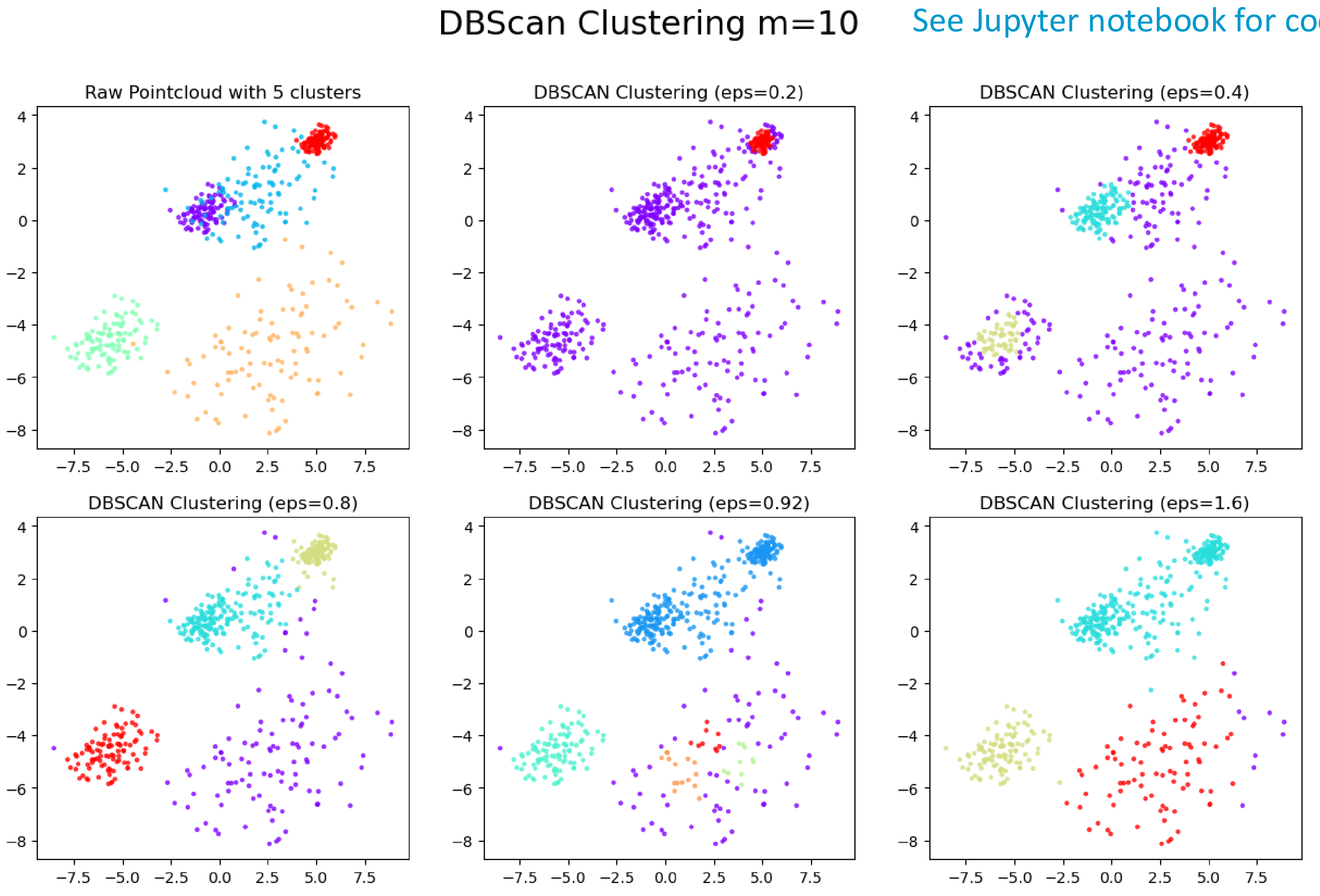
Run DBScan with m=10 for multiple epsilon values





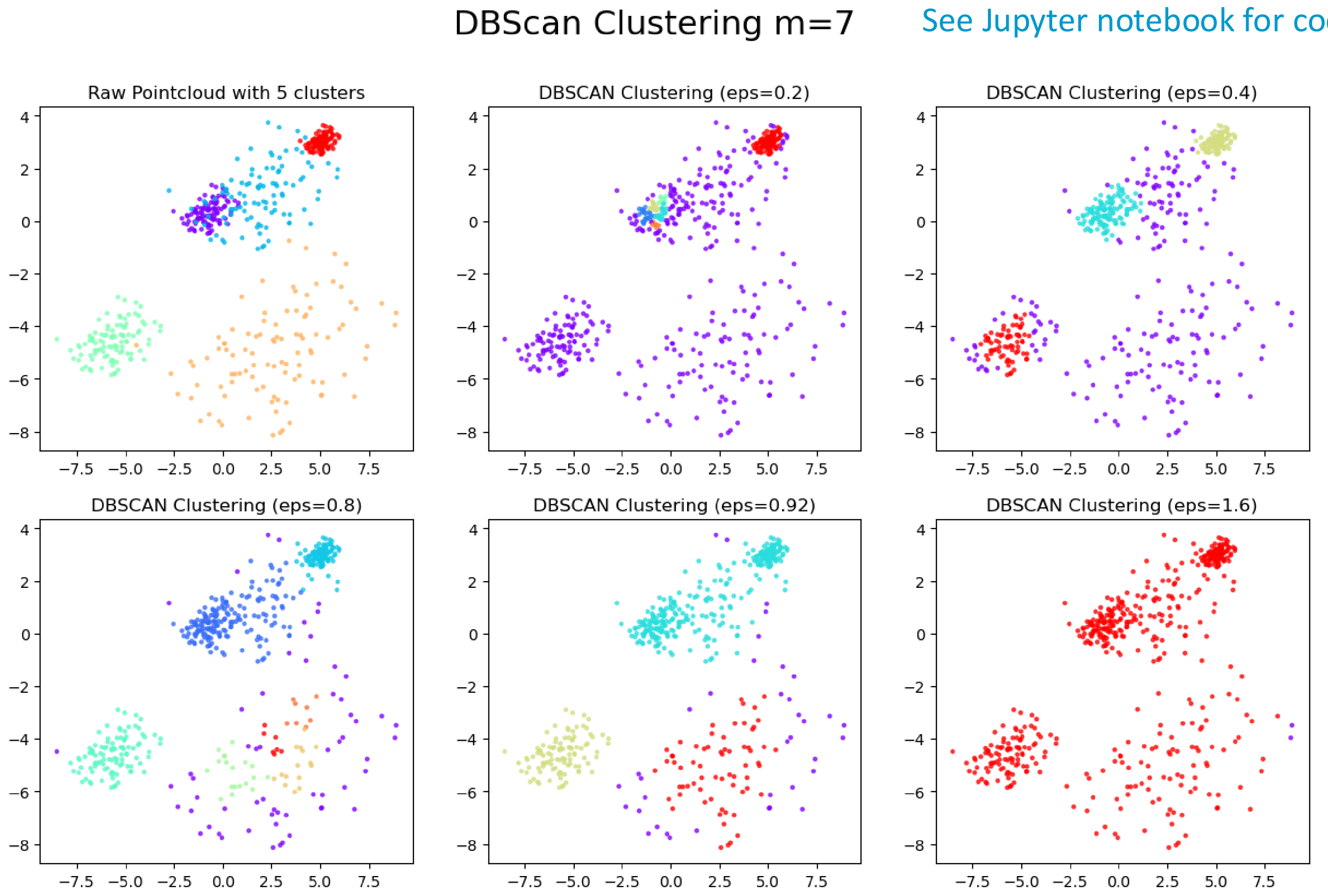
DBScan : Effect of varying epsilon

See Jupyter notebook for code



DBScan : Effect of varying m

See Jupyter notebook for code



Pros and Cons of DBScan

- DBSCAN can find clusters of different shapes and sizes.
- DBSCAN doesn’t handle clusters of varying densities very well.
- Can be expensive for large data due to neighborhood calculations (distances)