CS-470: Machine Learning

Week 12 - Clustering Algorithms

Instructor: Dr. Sajjad Hussain

Department of Electrical and Computer Engineering
SEECS, NUST

December 4th, 2025

What is Clustering?

Definition

Clustering is an **unsupervised learning** technique that groups similar data points together while keeping dissimilar points in different groups.

- **Goal**: Discover natural groupings in data
- **Input**: Unlabeled data points
- **Output**: Groups (clusters) where intra-cluster similarity is high and inter-cluster similarity is low

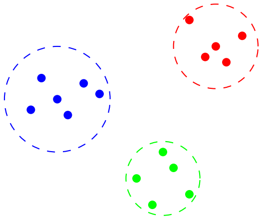


Figure: Example of clustered data with three clusters

Applications of Clustering

- **Customer Segmentation**
- **Image Segmentation**
- **Anomaly Detection**
- **Document Classification**
- **Social Network Analysis**
- **Market Research**
- **Gene Sequence Analysis**
- **Recommender Systems**

Real-World Example: Customer Segmentation

- Group customers by purchasing behavior
- Targeted marketing campaigns
- Personalized recommendations
- Customer retention strategies

K-Means: Basic Idea

Objective Function

Minimize the within-cluster sum of squares (WCSS):

$$J = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

where:

- k: number of clusters
- C_i: set of points in cluster i
- μ_i: centroid of cluster i

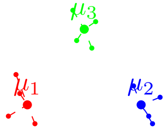


Figure: K-Means minimizes distances from points to their cluster centroids

K-Means Algorithm

Algorithm 1 K-Means Clustering

Require: Data points X, number of clusters k, maximum iterations T

```
Initialize k centroids μ_1, μ_2, ..., μ_k
for t = 1 to T do
    Assignment Step: Assign each point to nearest centroid
        C_i = {x : ||x − μ_i||^2 ≤ ||x − μ_j||^2 ∀j}
    Update Step: Recalculate centroids
        μ_i = (1/N_i) Σ_{x∈C_i} x
    if centroids don't change then
        break
    end if
end for
return Clusters C_1, C_2, ..., C_k and centroids μ_1, μ_2, ..., μ_k
```

$$C_i = \{x : \|x - \mu_i\|^2 \leq \|x - \mu_j\|^2 \ \forall j\}$$

$$\mu_i = \frac{1}{N_i} \sum_{x \in C_i} x$$

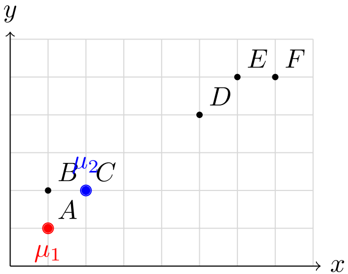
Toy Example: Step 1 - Initialization

Data Points

Points: $A(1,1), B(1,2), C(2,2), D(5,4), E(6,5), F(7,5)$
Choose $k = 2$ clusters

Random Initial Centroids

Let's initialize: $\mu_1 = (1,1)$ (point A), $\mu_2 = (2,2)$ (point C)



Step 2 - First Assignment

Calculate distances

Point	Distance to μ_1	Distance to μ_2	Cluster
A(1,1)	0	$\sqrt{(1-2)^2+(1-2)^2} = 1.41$	C_1
B(1,2)	1	$\sqrt{(1-2)^2+(2-2)^2} = 1$	C_2
C(2,2)	1.41	0	C_2
D(5,4)	5	$\sqrt{(5-2)^2+(4-2)^2} = 3.61$	C_2
E(6,5)	6.4	$\sqrt{(6-2)^2+(5-2)^2} = 5$	C_2
F(7,5)	7.2	$\sqrt{(7-2)^2+(5-2)^2} = 5.83$	C_2

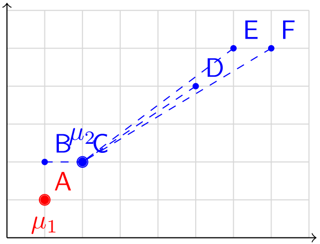


Figure: Assignment: Cluster 1: {A}, Cluster 2: {B,C,D,E,F}

Step 3 - Update Centroids

Recalculate Centroids

Cluster 1: $\mu_1 = \frac{1}{1}[(1,1)] = (1,1)$
Cluster 2: $\mu_2 = \frac{1}{5}[(1,2)+(2,2)+(5,4)+(6,5)+(7,5)] = (\frac{21}{5}, \frac{18}{5}) = (4.2, 3.6)$

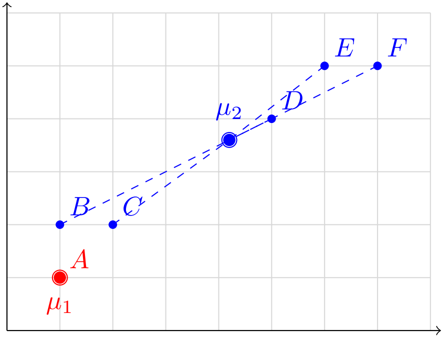


Figure: Updated centroids after first iteration

Step 4 - Reassignment

Recalculate distances to new centroids

Point	Dist to $\mu_1(1,1)$	Dist to $\mu_2(4.2,3.6)$	Cluster
A(1,1)	0	3.83	C_1
B(1,2)	1	3.30	C_1
C(2,2)	1.41	2.58	C_1
D(5,4)	5	1.08	C_2
E(6,5)	6.4	2.15	C_2
F(7,5)	7.2	3.08	C_2

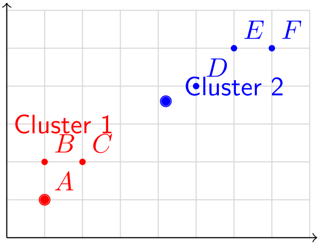


Figure: New assignment: Cluster 1: {A,B,C}, Cluster 2: {D,E,F}

Step 5 - Final Centroids

Update Centroids Again

Cluster 1: $\mu_1 = \frac{1}{3}[(1,1) + (1,2) + (2,2)] = (\frac{4}{3}, \frac{5}{3}) = (1.33, 1.67)$
Cluster 2: $\mu_2 = \frac{1}{3}[(5,4) + (6,5) + (7,5)] = (\frac{18}{3}, \frac{14}{3}) = (6, 4.67)$

Convergence Check

No points change clusters with these new centroids $\rightarrow$ Algorithm converges!

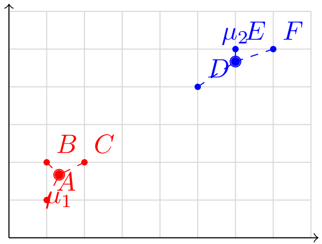


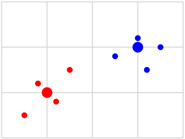
Figure: Final clustering after convergence

Centroid Initialization Methods

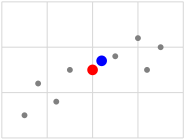
The Problem

K-Means is sensitive to initial centroid positions → Can converge to local minima

Good Initialization



Bad Initialization



Common Methods

1. **Random initialization**: Choose k random points from dataset
2. **K-Means++**: Smart initialization to spread out initial centroids
3. **Forgy method**: Choose k random data points as centroids
4. **MacQueen method**: Random partition then compute centroids

K-Means++ Initialization

Algorithm 2 K-Means++ Initialization

1: Choose first centroid μ_1 uniformly at random from data points
2: **for** $i = 2$ to k **do**
3: For each point x, compute $D(x) =$ distance to nearest centroid
4: Choose μ_i with probability proportional to $D(x)^2$
5: **end for**

In scikit-learn: `n_init` parameter

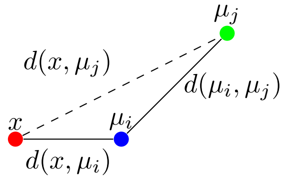
- `n_init`: Number of times algorithm is run with different centroid seeds
- Final results: Best output of `n_init` consecutive runs
- Default: `n_init=10`

Accelerated K-Means (Elkan's Algorithm)

Key Idea

Use triangle inequality to avoid unnecessary distance calculations

- Maintain lower bounds on distances
- If $d(\mu_i, \mu_j) \geq 2d(x, \mu_i)$, then x cannot be closer to μ_j
- Avoid computing $d(x, \mu_j)$
- **Speedup**: 2-10x faster for high-dimensional data



$$d(x, \mu_j) \geq |d(\mu_i, \mu_j) - d(x, \mu_i)|$$

Mini-Batch K-Means

Idea

Use random subsets (mini-batches) of data to update centroids

Algorithm 3 Mini-Batch K-Means

1: Initialize centroids
2: **for** each iteration **do**
3: Sample random mini-batch B from dataset
4: **for** each point x in B **do**
5: Assign x to nearest centroid
6: Update centroid using moving average:

$$\mu_i \leftarrow \frac{n_i \mu_i + x}{n_i + 1}$$

where n_i is number of points assigned to centroid i so far

7: **end for**
8: **end for**

Finding Optimal Number of Clusters: Elbow Method

Inertia (Within-Cluster Sum of Squares)

$$\text{Inertia} = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

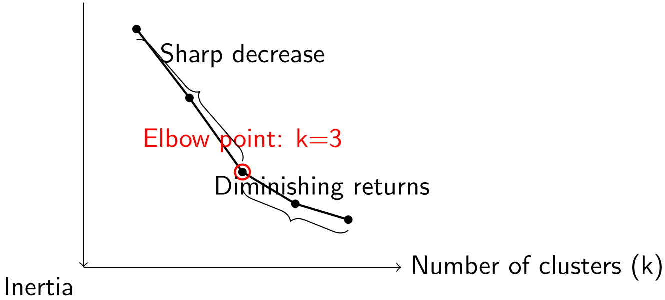


Figure: Elbow method: Choose k where inertia decrease slows down

Silhouette Score Method

Silhouette Score for a point

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

where:

- $a(i)$: Average distance to points in same cluster
- $b(i)$: Average distance to points in next nearest cluster

Interpretation

- $s(i) \in [-1, 1]$
- Close to 1: Well clustered
- Close to 0: Between clusters
- Close to -1: Wrong cluster

Limitations of K-Means

1. Assumes Spherical Clusters

Does not work well for non-spherical clusters

2. Sensitive to Outliers

Computes Centroids based on all points even if they are outliers

3. Requires Specifying K

Must know or guess number of clusters

4. Scale Dependent

Features must be scaled (normalized)

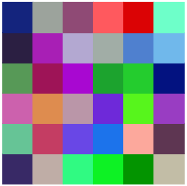
5. Equal Variance Assumption

Assumes clusters have similar sizes and densities

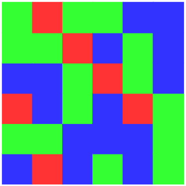
Application: Image Segmentation with K-Means

Idea

Treat each pixel as a data point in RGB space, cluster to find dominant colors



Original Image



Segmented (k=3)

Process

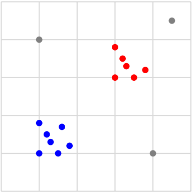
1. Reshape image to pixels $\times$ RGB values
2. Apply K-Means with desired k (number of colors)
3. Replace each pixel with its cluster centroid color
4. Reshape back to image dimensions

DBSCAN: Density-Based Spatial Clustering

Key Idea

Cluster = dense region of points separated by low-density regions

- **Density-based**: Finds arbitrarily shaped clusters
- **Robust**: Handles outliers well
- **Parameters**:
  - ϵ (eps): Neighborhood radius
  - minPts: Minimum points to form dense region



Core Concepts

- **Core point**: Has at least minPts points within ϵ radius
- **Border point**: Within ϵ of a core point but not core itself
- **Noise point**: Neither core nor border point

DBSCAN: Understanding the Key Concepts

1. ϵ-Neighborhood (The "Friend Circle")

In words: All points that are within a distance ϵ from point p. Think of it as drawing a circle of radius ϵ around p - any point inside this circle is in its ϵ-neighborhood.

Mathematically:

$$N_\epsilon(p) = \{q \in D \mid \text{distance}(p, q) \leq \epsilon\}$$

2. Core Point (The "Popular" Point)

In words: A point is a core point if it has at least `minPts` neighbors within its ϵ-neighborhood (including itself). These are the points that form the dense centers of clusters.

Mathematically:

$$\text{Point } p \text{ is a CORE POINT if: } |N_\epsilon(p)| \geq \text{minPts}$$

DBSCAN: How Points Connect to Form Clusters

3. Directly Density-Reachable (Immediate Connection)

In words: Point q is directly density-reachable from point p if:

- p is a core point (has enough neighbors)
- q is within p's ϵ-neighborhood (close to p)

Think: q is directly connected to the popular point p.

Mathematically: q is **directly density-reachable** from p if:

1. p is a core point
2. $q \in N_\epsilon(p)$

4. Density-Reachable (Connected Through Friends)

In words: Point q is density-reachable from point p if you can travel from p to q through a chain of points where each point is directly density-reachable from the previous one. Like a friendship chain: p knows p_2, p_2 knows p_3, ..., eventually reaching q.

DBSCAN: What Makes a Cluster?

5. Cluster Definition (The Complete Group)

In words: A cluster is a group of points where:

- **Completeness:** If a point p is in the cluster, then ALL points that are density-reachable from p must also be in the cluster.
- **Connectivity:** Any two points in the cluster are connected through density-reachability (directly or indirectly).

Mathematically: A cluster C satisfies:

1. If $p \in C$ and q is density-reachable from p, then $q \in C$
2. For any $p, q \in C$, p and q are density-connected

Key Insight

DBSCAN finds clusters by starting from a core point and "growing" the cluster by adding all density-reachable points. Points that aren't density-reachable from any core point are marked as noise/outliers.

Toy Example: DBSCAN Algorithm

Parameters

$\epsilon = 1.2$, minPts $= 3$

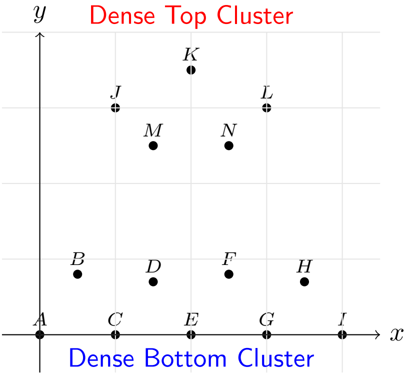
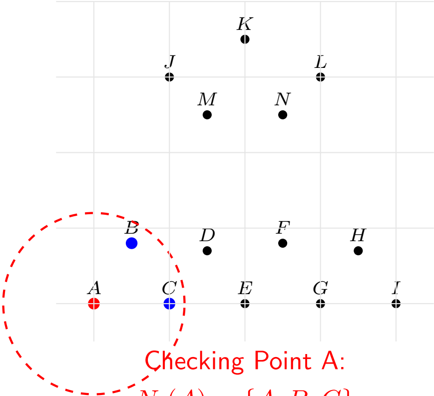


Figure: Two clearly separated clusters with $\epsilon = 1.2$

Step 1: Identify Core Points - Check Point A

Core Point Condition

A point is a **core point** if it has at least `minPts=3` points within $\epsilon = 1.2$ radius (including itself)

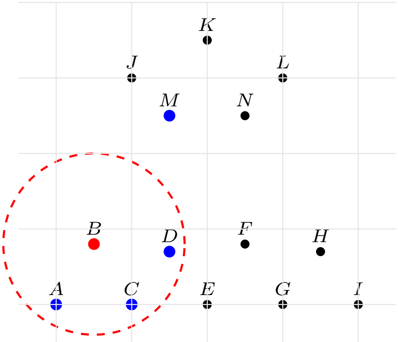


Checking Point A:

$$N_\epsilon(A) = \{A, B, C\}$$

$$|N_\epsilon(A)| = 3 = \text{minPts}$$

Step 2: Check Point B



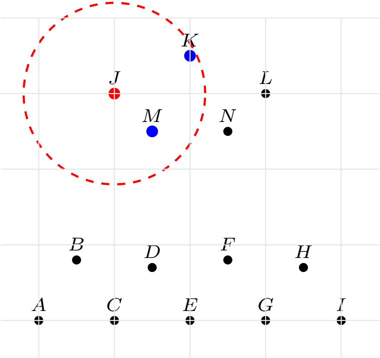
Checking Point B:

$$\text{Distance(B,M)} = \sqrt{(1.5 - 0.5)^2 + (2.5 - 0.8)^2} \approx 1.97 > 1.2$$

$$N_\epsilon(B) = \{A, B, C, D\}$$

$$|N_\epsilon(B)| = 4 \geq 3 \Rightarrow \text{CORE POINT}$$

Step 3: Check Point J (First point in top cluster)



Checking Point J (Top Cluster):

$$\text{Distance(J,K)} = \sqrt{(2-1)^2 + (3.5-3)^2} \approx 1.12 < 1.2$$

$$\text{Distance(J,M)} = \sqrt{(1.5-1)^2 + (2.5-3)^2} \approx 0.71 < 1.2$$

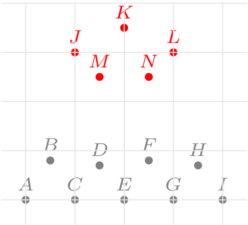
$$N_\epsilon(J) = \{J, M, K\} \Rightarrow |N_\epsilon(J)| = 3$$

$\Rightarrow$ J is a CORE POINT

Step 4: All Core Points Identified

Core Point Analysis Complete

After checking all points (showing selected results):



Core Points

Bottom Cluster:

- Core: A, B, C, D, E, F, G, H
- Non-core: I (edge point)

Top Cluster:

- Core: J, K, L, M, N (all points)

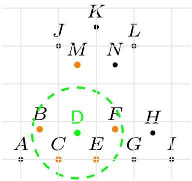
Why I is non-core

- $N_\epsilon(I) = \{H, I\}$ only
- $|N_\epsilon(I)| = 2 < 3$ (minPts)
- But I is within ϵ of core point H
- $\Rightarrow$ I is a BORDER point

Step 5: Start Clustering - Randomly Select Starting Point D

DBSCAN Algorithm Begins

- All points unvisited initially
- Randomly select unvisited core point D
- Create Cluster 1
- Add D to Cluster 1
- Begin expansion from D

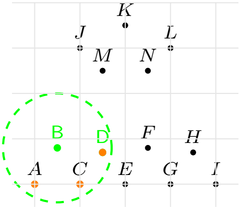


Starting with core point D
Create Cluster 1
Neighbors within $\epsilon = 1.2$: B, C, E, F
M is NOT within ϵ (distance $\approx 1.8 > 1.2$)

Step 6: Process B from Queue

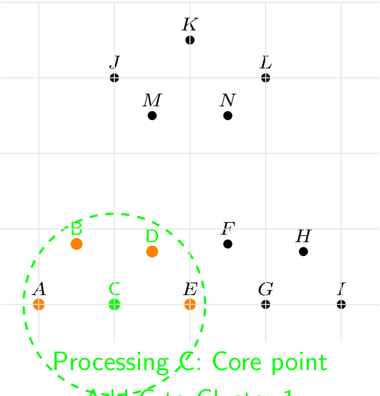
Expand Cluster 1

- Take B from queue
- B is core point (we identified earlier)
- Add B to Cluster 1
- Add B's unvisited neighbors to queue



Processing B: Core point
Add B to Cluster 1
B's new neighbors: A, C (D already visited)
Queue: C, E, F, A
Cluster 1: {B, D}

Step 7: Process C from Queue



Processing C: Core point

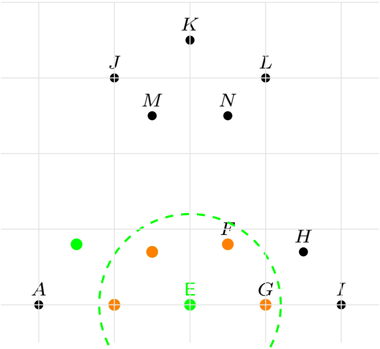
Add C to Cluster 1

C's new neighbor: E (A already in queue)

Queue: E, F, A, E (E appears twice - will be deduplicated)

Cluster 1: {B, C, D}

Step 8: Process E from Queue



Processing E: Core point

Add E to Cluster 1

E's new neighbors: F, G (C, D already in cluster)

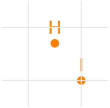
Queue: F, A, G

Cluster 1: {B, C, D, E}

Step 9: Continue Expansion - Process F, A, G

Processing Queue

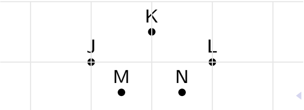
- **F**: Core point, adds to Cluster 1, adds H to queue
- **A**: Core point, adds to Cluster 1, no new neighbors
- **G**: Core point, adds to Cluster 1, adds H, I to queue



Next in queue

Current State

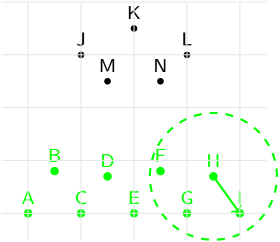
- Cluster 1: {A, B, C, D, E, F, G}
- Queue: H, I
- H is core point, I is border point



Step 10: Process H and I - Complete Cluster 1

Final Points in Bottom Cluster

- **H**: Core point, adds to Cluster 1
- **I**: Border point (not core), but within ϵ of H
- I is density-reachable from D through chain D-E-F-G-H-I

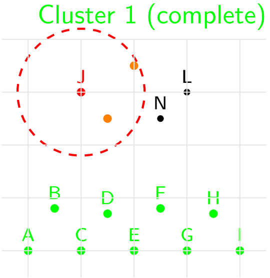


Processing H: Core point
Processing I: Border point (connected to H)
Cluster 1 COMPLETE! {A, B, C, D, E, F, G, H, I}
All bottom points clustered
Top cluster (J-N) still unvisited

Step 11: Start Cluster 2 - Randomly Select J

Find Next Unvisited Point

- All points A-I are visited and in Cluster 1
- Next unvisited point: J (in top cluster)
- J is core point (we identified earlier)
- Create Cluster 2



Start Cluster 2 with core point J

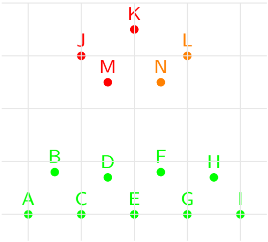
$$N_\epsilon(J) = \{J, M, K\}$$

Neighbors to expand: M, K

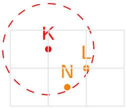
Step 12: Expand Cluster 2 - Process M and K

Process Queue

- **M**: Core point, adds to Cluster 2
- M's neighbors: J (visited), K (in queue), N, L
- Add N, L to queue
- **K**: Core point, adds to Cluster 2
- K's neighbors: J, M (both visited), L, N
- L, N already in queue



Cluster 2: {J, M, K}
Queue: L, N
All points in top cluster will connect

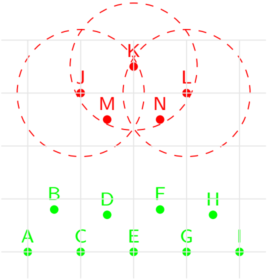


K connects to L and N

Step 13: Complete Cluster 2

Final Expansion

- **L**: Core point, adds to Cluster 2, connects to N
- **N**: Core point, adds to Cluster 2, connects to L, M, K
- All points in top cluster are density-connected



Cluster 1: 9 points Cluster 2: 5 points

All points clustered - NO NOISE!

DBSCAN successfully found 2 natural clusters

DBSCAN Algorithm Summary

What We Demonstrated

1. **Core Point Identification**: Check each point's ϵ-neighborhood
2. **Cluster Growth**: Start from unvisited core point, expand via density-reachability
3. **Multiple Clusters**: Find all density-connected components
4. **Noise Detection**: Points not density-reachable from any core (none in our example)

Key Insights from Our Example

- **Parameter Choice Matters**: $\epsilon = 1.2$ perfectly separates the two clusters
- **Density-Based**: Finds clusters of arbitrary shape (both clusters are non-linear)
- **Robust**: Handles border points (like I) correctly
- **Order Independence**: Could start with any core point, would find same clusters

DBSCAN: Advantages and Limitations

Advantages

- **Arbitrary shapes**: Finds non-convex clusters
- **Robust to outliers**: Identifies noise points
- **No need for k**: Determines number of clusters automatically
- **Hierarchical**: Can find nested clusters with right parameters

Limitations

- **Parameter sensitive**: ϵ and minPts choice critical
- **Border points**: Can belong to multiple clusters (ambiguity)
- **Varying density**: Struggles with clusters of different densities
- **High-dimensional data**: Distance measures become less meaningful

Choosing ϵ and minPts

- **minPts**: Rule of thumb: minPts $\geq$ dimensions $+ 1$
- ϵ: Use k-distance plot (sorted distances to k-th nearest neighbor)
- Domain knowledge often needed

K-Means vs DBSCAN: Comparison

Aspect	K-Means	DBSCAN
Cluster Shape	Spherical, convex	Arbitrary, non-convex
Outliers	Sensitive, affects centroids	Robust, identifies as noise
Parameters	Number of clusters (k)	ϵ (eps) and minPts
Number of Clusters	Must specify k	Determines automatically
Complexity	$O(n \cdot k \cdot d \cdot i)$	$O(n \log n)$ with spatial indexing
Scaling	Requires feature scaling	Distance-based, needs scaling
Use Case	Well-separated, spherical clusters	Density-based, arbitrary shapes

Table: Comparison of K-Means and DBSCAN

- **K-Means**: When you expect spherical clusters, know approximate k, need fast results
- **DBSCAN**: When clusters have arbitrary shapes, outliers exist, unknown k

Summary

Key Takeaways

1. **K-Means**: Centroid-based, minimizes within-cluster variance, spherical clusters
2. **DBSCAN**: Density-based, finds arbitrary shapes, robust to outliers
3. Both have strengths and weaknesses
4. Choice depends on data characteristics and problem requirements

Practical Considerations

- Always preprocess data (scale features)
- Use visualization to understand cluster shapes
- Consider hybrid approaches
- Validate with domain knowledge

References and Further Reading

- MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations.
- Arthur, D., Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding.
- Elkan, C. (2003). Using the triangle inequality to accelerate k-means.
- Ester, M., Kriegel, H., Sander, J., Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases.
- Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python.

Tools and Libraries

- `scikit-learn`: `KMeans`, `DBSCAN`, `MiniBatchKMeans`
- Visualization: Matplotlib, Seaborn, Plotly
- Large datasets: Spark MLlib, HDBSCAN