Unsupervised Learning: Clustering

Finding Structure in Unlabeled Data

Nipun Batra

IIT Gandhinagar

October 29, 2025

What We'll Learn Today

1. **Why** unsupervised learning? (Motivation)
2. **What** is clustering? (Intuition first!)
3. **How** does K-Means work? (Step-by-step)
4. **When** does it fail? (Limitations)
5. **What** are alternatives? (Hierarchical, DBSCAN)

What We'll Learn Today

1. **Why** unsupervised learning? (Motivation)
2. **What** is clustering? (Intuition first!)
3. **How** does K-Means work? (Step-by-step)
4. **When** does it fail? (Limitations)
5. **What** are alternatives? (Hierarchical, DBSCAN)

Key Points: Key Points

Key Philosophy: We'll build intuition with examples, then add rigor!

Motivation

Supervised vs Unsupervised Learning

Supervised Learning

- Have: Features X + Labels Y
- Goal: Learn $f : X \to Y$
- Example: Spam detection
  - Email text $\to$ Spam/Not Spam

Supervised vs Unsupervised Learning

Supervised Learning

- Have: Features X + Labels Y
- Goal: Learn $f : X \rightarrow Y$
- Example: Spam detection
  - Email text → Spam/Not Spam

Unsupervised Learning

- Have: Features X only (no labels!)
- Goal: Find structure/patterns
- Example: Customer segmentation
  - Customer data → Find groups

Supervised vs Unsupervised Learning

Supervised Learning

- Have: Features X + Labels Y
- Goal: Learn $f : X \to Y$
- Example: Spam detection
  - Email text → Spam/Not Spam

Unsupervised Learning

- Have: Features X only (no labels!)
- Goal: Find structure/patterns
- Example: Customer segmentation
  - Customer data → Find groups

Important:

Key Difference: We don't know the "right answer" beforehand!

Why Unsupervised Learning?

Three Main Reasons:

Why Unsupervised Learning?

Three Main Reasons:

1. Labels are expensive/impossible to get

- Medical images: Need expert radiologists (costly!)
- Customer behavior: No "true" groupings exist
- Exploratory analysis: Don't know what to look for yet

Why Unsupervised Learning?

Three Main Reasons:

1. **Labels are expensive/impossible to get**
   - Medical images: Need expert radiologists (costly!)
   - Customer behavior: No "true" groupings exist
   - Exploratory analysis: Don't know what to look for yet
2. **Discover hidden patterns**
   - Find new disease subtypes
   - Identify market segments you didn't know existed
   - Detect anomalies (fraud, network intrusion)

Why Unsupervised Learning?

Three Main Reasons:

1. Labels are expensive/impossible to get

- Medical images: Need expert radiologists (costly!)
- Customer behavior: No "true" groupings exist
- Exploratory analysis: Don't know what to look for yet

2. Discover hidden patterns

- Find new disease subtypes
- Identify market segments you didn't know existed
- Detect anomalies (fraud, network intrusion)

3. Data preprocessing

- Dimensionality reduction before supervised learning
- Feature extraction
- Data compression

Real-World Applications

Business

- **E-commerce**: Group customers by purchase behavior
- **Marketing**: Segment markets for targeted campaigns
- **Recommendation**: "Customers who bought X also bought Y"

Real-World Applications

Business

- **E-commerce**: Group customers by purchase behavior
- **Marketing**: Segment markets for targeted campaigns
- **Recommendation**: "Customers who bought X also bought Y"

Healthcare

- **Disease subtypes**: Find variants of cancer
- **Patient stratification**: Personalized treatment

Real-World Applications

Business

- **E-commerce**: Group customers by purchase behavior
- **Marketing**: Segment markets for targeted campaigns
- **Recommendation**: "Customers who bought X also bought Y"

Healthcare

- **Disease subtypes**: Find variants of cancer
- **Patient stratification**: Personalized treatment

Technology

- **Image segmentation**: Divide image into regions
- **Document clustering**: Organize large text collections
- **Anomaly detection**: Find unusual patterns

Real-World Applications

Business

- **E-commerce**: Group customers by purchase behavior
- **Marketing**: Segment markets for targeted campaigns
- **Recommendation**: "Customers who bought X also bought Y"

Healthcare

- **Disease subtypes**: Find variants of cancer
- **Patient stratification**: Personalized treatment

Technology

- **Image segmentation**: Divide image into regions
- **Document clustering**: Organize large text collections
- **Anomaly detection**: Find unusual patterns

Science

- **Astronomy**: Classify galaxies
- **Climate**: Identify weather patterns
- **Biology**: Group similar

What is Clustering?

Clustering: The Intuition

Informal Definition:
Group similar objects together, separate dissimilar objects

Clustering: The Intuition

Informal Definition:
Group similar objects together, separate dissimilar objects

Key Questions:

1. What does "similar" mean? (Need a distance/similarity measure)

Clustering: The Intuition

Informal Definition:
Group similar objects together, separate dissimilar objects

Key Questions:

1. What does "similar" mean? (Need a distance/similarity measure)
2. How many groups? (May or may not know beforehand)

Clustering: The Intuition

Informal Definition:
Group similar objects together, separate dissimilar objects

Key Questions:

1. What does "similar" mean? (Need a distance/similarity measure)
2. How many groups? (May or may not know beforehand)
3. What shape are clusters? (Spherical? Elongated? Arbitrary?)

Clustering: The Intuition

Informal Definition:
Group similar objects together, separate dissimilar objects

Key Questions:

1. What does "similar" mean? (Need a distance/similarity measure)
2. How many groups? (May or may not know beforehand)
3. What shape are clusters? (Spherical? Elongated? Arbitrary?)

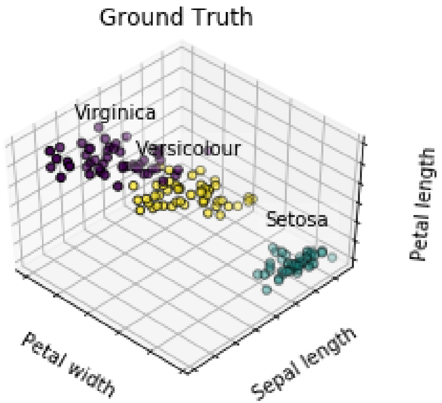
Example: Everyday Example

Organizing your wardrobe:

- **By color**: All red clothes together
- **By type**: All shirts together
- **By season**: All winter clothes together

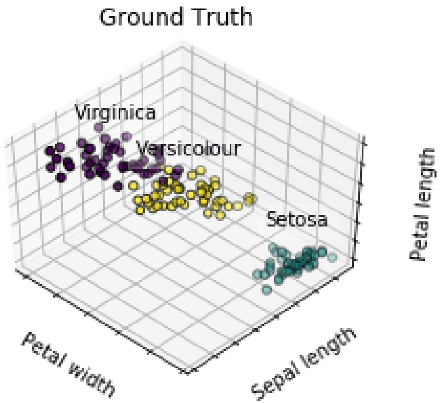
Same clothes, different clusterings! Choice depends on your

Clustering: Visual Intuition



Iris Dataset: Can you spot the groups?

Clustering: Visual Intuition



Iris Dataset: Can you spot the groups?

- **Visual**: Easy to see 2-3 distinct groups
- **Question**: How do we make a computer see this?

Formal Problem Statement

Given:

- n data points: $\{x_1, x_2, \ldots, x_n\}$ where $x_i \in \mathbb{R}^d$
- Number of clusters K (sometimes)
- Distance/similarity measure (usually Euclidean)

Formal Problem Statement

Given:

- n data points: $\{x_1, x_2, \ldots, x_n\}$ where $x_i \in \mathbb{R}^d$
- Number of clusters K (sometimes)
- Distance/similarity measure (usually Euclidean)

Find:

- Partition data into K clusters: $C_1, C_2, \ldots, C_K$
- Such that:
  - Points in same cluster are similar (small distances)
  - Points in different clusters are dissimilar (large distances)

Formal Problem Statement

Given:

- n data points: $\{x_1, x_2, \ldots, x_n\}$ where $x_i \in \mathbb{R}^d$
- Number of clusters K (sometimes)
- Distance/similarity measure (usually Euclidean)

Find:

- Partition data into K clusters: $C_1, C_2, \ldots, C_K$
- Such that:
  - Points in same cluster are similar (small distances)
  - Points in different clusters are dissimilar (large distances)

Mathematically:

- $C_1 \cup C_2 \cup \ldots \cup C_K = \{1, 2, \ldots, n\}$ (every point in some cluster)
- $C_i \cap C_j = \emptyset$ for $i \neq j$ (no overlaps)

Types of Clustering Algorithms

Definition (Clustering Algorithm Families)

Different approaches to finding clusters:

1. **Partitioning**: Divide data into K groups
   - K-Means, K-Medoids
   - Need to specify K

Types of Clustering Algorithms

Definition (Clustering Algorithm Families)

Different approaches to finding clusters:

1. **Partitioning**: Divide data into K groups
   - K-Means, K-Medoids
   - Need to specify K
2. **Hierarchical**: Build tree of clusters
   - Agglomerative (bottom-up), Divisive (top-down)
   - Don't need to specify K

Types of Clustering Algorithms

Definition (Clustering Algorithm Families)

Different approaches to finding clusters:

1. **Partitioning**: Divide data into K groups
   - K-Means, K-Medoids
   - Need to specify K
2. **Hierarchical**: Build tree of clusters
   - Agglomerative (bottom-up), Divisive (top-down)
   - Don't need to specify K
3. **Density-Based**: Find dense regions
   - DBSCAN, OPTICS
   - Can find arbitrary shapes

Types of Clustering Algorithms

Definition (Clustering Algorithm Families)

Different approaches to finding clusters:

1. **Partitioning**: Divide data into K groups
   - K-Means, K-Medoids
   - Need to specify K
2. **Hierarchical**: Build tree of clusters
   - Agglomerative (bottom-up), Divisive (top-down)
   - Don't need to specify K
3. **Density-Based**: Find dense regions
   - DBSCAN, OPTICS
   - Can find arbitrary shapes
4. **Model-Based**: Assume statistical model
   - Gaussian Mixture Models (GMM)
   - Probabilistic approach

Types of Clustering Algorithms

Definition (Clustering Algorithm Families)

Different approaches to finding clusters:

1. **Partitioning**: Divide data into K groups
   - K-Means, K-Medoids
   - Need to specify K
2. **Hierarchical**: Build tree of clusters
   - Agglomerative (bottom-up), Divisive (top-down)
   - Don't need to specify K
3. **Density-Based**: Find dense regions
   - DBSCAN, OPTICS
   - Can find arbitrary shapes
4. **Model-Based**: Assume statistical model
   - Gaussian Mixture Models (GMM)
   - Probabilistic approach

Today's Focus: K-Means (most popular), then Hierarchical and DBSCAN

K-Means: Intuition First!

K-Means: The Big Idea

Core Intuition:

Each cluster has a center point (centroid).
Assign each point to the nearest centroid.
Update centroids based on assigned points.
Repeat until stable!

K-Means: The Big Idea

Core Intuition:

Each cluster has a center point (centroid).
Assign each point to the nearest centroid.
Update centroids based on assigned points.
Repeat until stable!

Why "K-Means"?

- **K**: Number of clusters
- **Means**: Centroids are computed as means (averages)

K-Means: The Big Idea

Core Intuition:

Each cluster has a center point (centroid).
Assign each point to the nearest centroid.
Update centroids based on assigned points.
Repeat until stable!

Why "K-Means"?

- **K**: Number of clusters
- **Means**: Centroids are computed as means (averages)

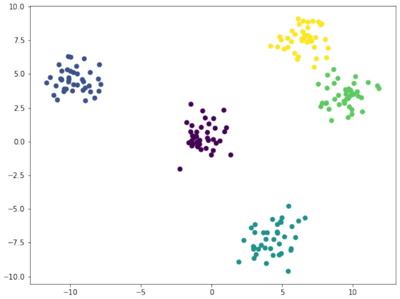
Key Points: Key Points

Two-Step Dance:

1. **Assignment**: Points $\rightarrow$ Nearest centroid
2. **Update**: Centroids $\rightarrow$ Mean of assigned points

Repeat until nothing changes!

K-Means: Visual Walkthrough (Step 0)



Raw data: Can you guess $K = 5$?

Start: Pick $K = 5$ centroids randomly (colored points)

K-Means: What Are We Optimizing?

Intuitive Goal:

Make clusters tight - minimize distances within each cluster

K-Means: What Are We Optimizing?

Intuitive Goal:
Make clusters tight - minimize distances within each cluster

Formal Objective: Minimize Within-Cluster Sum of Squares (WCSS)

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

where:

- $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$ is the centroid of cluster k
- $\|x_i - \mu_k\|^2$ is squared distance from point to its centroid

K-Means: What Are We Optimizing?

Intuitive Goal:
Make clusters tight - minimize distances within each cluster

Formal Objective: Minimize Within-Cluster Sum of Squares (WCSS)

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

where:

- $\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$ is the centroid of cluster k
- $\|x_i - \mu_k\|^2$ is squared distance from point to its centroid

Important:

In words: Sum of squared distances from each point to its cluster center

Why This Objective Makes Sense

Minimizing WCSS means:

- Points close to their centroid $\rightarrow$ Tight clusters ✓

Why This Objective Makes Sense

Minimizing WCSS means:

- Points close to their centroid $\rightarrow$ Tight clusters $\checkmark$
- If WCSS is small $\rightarrow$ Clusters are compact $\checkmark$

Why This Objective Makes Sense

Minimizing WCSS means:

- Points close to their centroid $\rightarrow$ Tight clusters $\checkmark$
- If WCSS is small $\rightarrow$ Clusters are compact $\checkmark$
- Lower WCSS $\rightarrow$ Better clustering (usually) $\checkmark$

Why This Objective Makes Sense

Minimizing WCSS means:

- Points close to their centroid $\rightarrow$ Tight clusters $\checkmark$
- If WCSS is small $\rightarrow$ Clusters are compact $\checkmark$
- Lower WCSS $\rightarrow$ Better clustering (usually) $\checkmark$

Example: Concrete Example

Suppose we have 3 points in cluster C_1: $(0,0),(1,0),(0,1)$

- Centroid: $\mu_1 = (\frac{1}{3}, \frac{1}{3})$
- WCSS for C_1:

$$\|(0,0)-\mu_1\|^2 + \|(1,0)-\mu_1\|^2 + \|(0,1)-\mu_1\|^2 = \frac{2}{3}$$

K-Means Algorithm

K-Means Algorithm: The Recipe

Input:

- Data points $\{x_1, \ldots, x_n\}$
- Number of clusters K

K-Means Algorithm: The Recipe

Input:

- Data points $\{x_1, \ldots, x_n\}$
- Number of clusters K

Algorithm:

1. **Initialize**: Randomly choose K points as initial centroids $\{\mu_1, \ldots, \mu_K\}$

K-Means Algorithm: The Recipe

Input:

- Data points $\{x_1, \ldots, x_n\}$
- Number of clusters K

Algorithm:

1. **Initialize**: Randomly choose K points as initial centroids $\{\mu_1, \ldots, \mu_K\}$
2. **Repeat** until convergence:

K-Means Algorithm: The Recipe

Input:

- Data points $\{x_1, \ldots, x_n\}$
- Number of clusters K

Algorithm:

1. **Initialize**: Randomly choose K points as initial centroids $\{\mu_1, \ldots, \mu_K\}$
2. **Repeat** until convergence:
    1) **Assignment Step**:
        - For each point x_i, assign to nearest centroid:

$$C_k^{(t)} = \{i : \|x_i - \mu_k^{(t)}\| \leq \|x_i - \mu_j^{(t)}\| \text{ for all } j\}$$

K-Means Algorithm: The Recipe

Input:

- Data points $\{x_1, \ldots, x_n\}$
- Number of clusters K

Algorithm:

1. **Initialize**: Randomly choose K points as initial centroids $\{\mu_1, \ldots, \mu_K\}$
2. **Repeat** until convergence:
   1) **Assignment Step**:
      - For each point x_i, assign to nearest centroid:
        $$C_k^{(t)} = \{i : \|x_i - \mu_k^{(t)}\| \leq \|x_i - \mu_j^{(t)}\| \text{ for all } j\}$$
   2) **Update Step**:
      - Recompute each centroid as mean of assigned points:
        $$\mu_k^{(t+1)} = \frac{1}{|C_k^{(t)}|} \sum_{i \in C_k^{(t)}} x_i$$

K-Means Algorithm: The Recipe

Input:

- Data points $\{x_1, \ldots, x_n\}$
- Number of clusters K

Algorithm:

1. **Initialize**: Randomly choose K points as initial centroids $\{\mu_1, \ldots, \mu_K\}$
2. **Repeat** until convergence:
    1) **Assignment Step**:
        - For each point x_i, assign to nearest centroid:

$$C_k^{(t)} = \{i : \|x_i - \mu_k^{(t)}\| \leq \|x_i - \mu_j^{(t)}\| \text{ for all } j\}$$

    2) **Update Step**:
        - Recompute each centroid as mean of assigned points:

$$\mu_k^{(t+1)} = \frac{1}{|C_k^{(t)}|} \sum_{i \in C_k^{(t)}} x_i$$

3. **Convergence**: Stop when assignments don't change (or change is tiny)

K-Means: Assignment Step in Detail

For each point x_i:

1. Compute distance to ALL K centroids:

$$d_k = \|x_i - \mu_k\|^2 \quad \text{for } k = 1, \ldots, K$$

K-Means: Assignment Step in Detail

For each point x_i:

1. Compute distance to ALL K centroids:

$$d_k = \|x_i - \mu_k\|^2 \quad \text{for } k = 1, \ldots, K$$

2. Find minimum distance:

$$k^* = \arg\min_k d_k$$

K-Means: Assignment Step in Detail

For each point x_i:

1. Compute distance to ALL K centroids:
$$d_k = \|x_i - \mu_k\|^2 \quad \text{for } k = 1, \ldots, K$$
2. Find minimum distance:
$$k^* = \arg\min_k d_k$$
3. Assign x_i to cluster k^*:
$$\text{label}[i] = k^*$$

K-Means: Assignment Step in Detail

For each point x_i:

1. Compute distance to ALL K centroids:

$$d_k = \|x_i - \mu_k\|^2 \quad \text{for } k = 1, \ldots, K$$

2. Find minimum distance:

$$k^* = \arg\min_k d_k$$

3. Assign x_i to cluster k^*:

$$\text{label}[i] = k^*$$

Example: Example

Point $x = (2, 3)$, centroids $\mu_1 = (1, 1)$, $\mu_2 = (4, 4)$

- $d_1 = (2 - 1)^2 + (3 - 1)^2 = 5$

K-Means: Update Step in Detail

For each cluster C_k:

1. Collect all points assigned to cluster k:

$$C_k = \{x_i : \text{label}[i] = k\}$$

K-Means: Update Step in Detail

For each cluster C_k:

1. Collect all points assigned to cluster k:

$$C_k = \{x_i : \text{label}[i] = k\}$$

2. Compute mean across all dimensions:

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$

K-Means: Update Step in Detail

For each cluster C_k:

1. Collect all points assigned to cluster k:
$$C_k = \{x_i : \text{label}[i] = k\}$$
2. Compute mean across all dimensions:
$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$
3. This new μ_k becomes the centroid for next iteration

K-Means: Update Step in Detail

For each cluster C_k:

1. Collect all points assigned to cluster k:

$$C_k = \{x_i : \text{label}[i] = k\}$$

2. Compute mean across all dimensions:

$$\mu_k = \frac{1}{|C_k|} \sum_{x_i \in C_k} x_i$$

3. This new μ_k becomes the centroid for next iteration

Example: Example

Cluster C_1 has points: $(0,0), (2,0), (0,2)$

$$\mu_1 = \frac{1}{3}[(0,0) + (2,0) + (0,2)] = \left(\frac{2}{3}, \frac{2}{3}\right)$$

K-Means: Worked Example (1/6)

Dataset: 6 points in 2D

Point	**Coordinates**
x_1	$(1, 1)$
x_2	$(2, 1)$
x_3	$(4, 3)$
x_4	$(5, 4)$
x_5	$(1, 3)$
x_6	$(2, 2)$

Goal: Cluster into $K = 2$ groups
Step 0: Initialize centroids randomly

- $\mu_1 = (1, 1)$ (pick x_1)
- $\mu_2 = (5, 4)$ (pick x_4)

K-Means: Worked Example (2/6) - Iteration 1: Assignment

Current centroids: $\mu_1 = (1,1)$, $\mu_2 = (5,4)$

Assign each point to nearest centroid:

Point	$d(\mu_1)$	$d(\mu_2)$	**Nearest**	**Cluster**
$x_1 = (1,1)$	0	$\sqrt{25} = 5$	μ_1	C_1
$x_2 = (2,1)$	1	$\sqrt{18} \approx 4.2$	μ_1	C_1
$x_3 = (4,3)$	$\sqrt{13} \approx 3.6$	$\sqrt{2} \approx 1.4$	μ_2	C_2
$x_4 = (5,4)$	5	0	μ_2	C_2
$x_5 = (1,3)$	2	$\sqrt{17} \approx 4.1$	μ_1	C_1
$x_6 = (2,2)$	$\sqrt{2} \approx 1.4$	$\sqrt{13} \approx 3.6$	μ_1	C_1

Result: $C_1 = \{x_1, x_2, x_5, x_6\}$, $C_2 = \{x_3, x_4\}$

K-Means: Worked Example (3/6) - Iteration 1: Update

Update centroids as means:
Cluster 1: $C_1 = \{(1,1),(2,1),(1,3),(2,2)\}$

$$\mu_1^{\text{new}} = \frac{1}{4}[(1,1)+(2,1)+(1,3)+(2,2)] = \frac{1}{4}(6,7) = (1.5, 1.75)$$

K-Means: Worked Example (3/6) - Iteration 1: Update

Update centroids as means:
Cluster 1: $C_1 = \{(1,1),(2,1),(1,3),(2,2)\}$

$$\mu_1^{\text{new}} = \frac{1}{4}[(1,1)+(2,1)+(1,3)+(2,2)] = \frac{1}{4}(6,7) = (1.5, 1.75)$$

Cluster 2: $C_2 = \{(4,3),(5,4)\}$

$$\mu_2^{\text{new}} = \frac{1}{2}[(4,3)+(5,4)] = \frac{1}{2}(9,7) = (4.5, 3.5)$$

K-Means: Worked Example (3/6) - Iteration 1: Update

Update centroids as means:

Cluster 1: $C_1 = \{(1,1),(2,1),(1,3),(2,2)\}$

$$\mu_1^{\text{new}} = \frac{1}{4}[(1,1)+(2,1)+(1,3)+(2,2)] = \frac{1}{4}(6,7) = (1.5, 1.75)$$

Cluster 2: $C_2 = \{(4,3),(5,4)\}$

$$\mu_2^{\text{new}} = \frac{1}{2}[(4,3)+(5,4)] = \frac{1}{2}(9,7) = (4.5, 3.5)$$

Important:

Notice: Centroids moved from $(1,1),(5,4)$ to $(1.5,1.75),(4.5,3.5)$
They shifted toward the "center of mass" of their clusters!

K-Means: Worked Example (4/6) - Iteration 2: Assignment

New centroids: $\mu_1 = (1.5, 1.75)$, $\mu_2 = (4.5, 3.5)$
Reassign points:

Point	$d(\mu_1)$	$d(\mu_2)$	Nearest	Cluster
$x_1 = (1, 1)$	0.9	6.8	μ_1	C_1
$x_2 = (2, 1)$	0.8	7.1	μ_1	C_1
$x_3 = (4, 3)$	3.8	0.4	μ_2	C_2
$x_4 = (5, 4)$	6.8	0.4	μ_2	C_2
$x_5 = (1, 3)$	1.6	3.9	μ_1	C_1
$x_6 = (2, 2)$	0.5	6.6	μ_1	C_1

Result: Same as before! $C_1 = \{x_1, x_2, x_5, x_6\}$, $C_2 = \{x_3, x_4\}$
Convergence: Assignments didn't change $\rightarrow$ STOP!

K-Means: Worked Example (5/6) - Final Clustering

Final clusters:

- **Cluster 1**: $(1, 1), (2, 1), (1, 3), (2, 2)$ with centroid $(1.5, 1.75)$
- **Cluster 2**: $(4, 3), (5, 4)$ with centroid $(4.5, 3.5)$

K-Means: Worked Example (5/6) - Final Clustering

Final clusters:

- **Cluster 1**: $(1,1),(2,1),(1,3),(2,2)$ with centroid $(1.5,1.75)$
- **Cluster 2**: $(4,3),(5,4)$ with centroid $(4.5,3.5)$

WCSS calculation:

$$\begin{aligned}\text{WCSS} &= \sum_{i\in C_1} \|x_i-\mu_1\|^2 + \sum_{i\in C_2} \|x_i-\mu_2\|^2 \\ &= [0.9+0.8+1.6+0.5]+[0.4+0.4] \\ &= 3.8+0.8=4.6\end{aligned}$$

K-Means: Worked Example (5/6) - Final Clustering

Final clusters:

- **Cluster 1**: $(1,1),(2,1),(1,3),(2,2)$ with centroid $(1.5, 1.75)$
- **Cluster 2**: $(4,3),(5,4)$ with centroid $(4.5, 3.5)$

WCSS calculation:

$$\begin{aligned}\text{WCSS} &= \sum_{i\in C_1} \|x_i - \mu_1\|^2 + \sum_{i\in C_2} \|x_i - \mu_2\|^2 \\ &= [0.9 + 0.8 + 1.6 + 0.5] + [0.4 + 0.4] \\ &= 3.8 + 0.8 = 4.6\end{aligned}$$

Key Points: Key Points

Converged in 2 iterations!
Typically converges quickly (5-10 iterations), but worst-case can be slow.

K-Means: Worked Example (6/6) - Visualization

Visual summary:

- **Initial**: Random centroids far from optimal
- **Iteration 1**: Centroids move toward cluster centers
- **Iteration 2**: Centroids stabilize, assignments don't change
- **Final**: Two clear, compact clusters

K-Means: Worked Example (6/6) - Visualization

Visual summary:

- **Initial**: Random centroids far from optimal
- **Iteration 1**: Centroids move toward cluster centers
- **Iteration 2**: Centroids stabilize, assignments don't change
- **Final**: Two clear, compact clusters

Example: Key Insight

K-Means is iteratively refining the clustering:

1. Assignment makes WCSS smaller (each point goes to nearest centroid)
2. Update makes WCSS smaller (centroid is optimal for its assigned points)
3. Repeat until can't improve further (local optimum)

K-Means: Mathematical Rigor

Why K-Means Converges: Intuitive Proof

Claim: K-Means always converges

Why K-Means Converges: Intuitive Proof

Claim: K-Means always converges
Intuition: Each step decreases (or keeps same) the objective function WCSS

Why K-Means Converges: Intuitive Proof

Claim: K-Means always converges
Intuition: Each step decreases (or keeps same) the objective function WCSS
Proof sketch:

1. **Assignment step**:
   - Each point assigned to nearest centroid
   - Cannot increase WCSS (optimal choice given fixed centroids)

Why K-Means Converges: Intuitive Proof

Claim: K-Means always converges
Intuition: Each step decreases (or keeps same) the objective function WCSS
Proof sketch:

1. **Assignment step**:
   - Each point assigned to nearest centroid
   - Cannot increase WCSS (optimal choice given fixed centroids)
2. **Update step**:
   - Centroid = mean of assigned points
   - Mean minimizes sum of squared distances (calculus!)
   - Cannot increase WCSS (optimal centroid given fixed assignments)

Why K-Means Converges: Intuitive Proof

Claim: K-Means always converges

Intuition: Each step decreases (or keeps same) the objective function WCSS

Proof sketch:

1. **Assignment step**:
   - Each point assigned to nearest centroid
   - Cannot increase WCSS (optimal choice given fixed centroids)
2. **Update step**:
   - Centroid = mean of assigned points
   - Mean minimizes sum of squared distances (calculus!)
   - Cannot increase WCSS (optimal centroid given fixed assignments)
3. **Conclusion**:
   - WCSS decreases or stays same at each step
   - WCSS ≥ 0 (bounded below)
   - Finite number of possible assignments (K^n)
   - Therefore, must converge! ✓

Why Mean Minimizes Sum of Squared Distances

Claim: For points $\{x_1, \ldots, x_m\}$, the mean $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$ minimizes $\sum_{i=1}^{m} \|x_i - c\|^2$

Why Mean Minimizes Sum of Squared Distances

Claim: For points $\{x_1, \ldots, x_m\}$, the mean $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$ minimizes $\sum_{i=1}^{m} \|x_i - c\|^2$

Proof: Let $f(c) = \sum_{i=1}^{m} \|x_i - c\|^2$. Taking derivative and setting to zero:

$$\frac{\partial f}{\partial c} = \sum_{i=1}^{m} 2(x_i - c)(-1) = 0$$

$$\sum_{i=1}^{m} (c - x_i) = 0$$

$$mc = \sum_{i=1}^{m} x_i$$

$$c = \frac{1}{m}\sum_{i=1}^{m} x_i = \bar{x} \quad \checkmark$$

Why Mean Minimizes Sum of Squared Distances

Claim: For points $\{x_1, \ldots, x_m\}$, the mean $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i$ minimizes $\sum_{i=1}^{m} \|x_i - c\|^2$

Proof: Let $f(c) = \sum_{i=1}^{m} \|x_i - c\|^2$. Taking derivative and setting to zero:

$$\frac{\partial f}{\partial c} = \sum_{i=1}^{m} 2(x_i - c)(-1) = 0$$

$$\sum_{i=1}^{m} (c - x_i) = 0$$

$$mc = \sum_{i=1}^{m} x_i$$

$$c = \frac{1}{m}\sum_{i=1}^{m} x_i = \bar{x} \quad \checkmark$$

Key Points: Key Points

This is why we use **means** in K-Means - they're optimal!

K-Means Objective: Equivalent Forms

Form 1: Within-Cluster Sum of Squares

$$\min \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

K-Means Objective: Equivalent Forms

Form 1: Within-Cluster Sum of Squares

$$\min \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Form 2: Pairwise distances within clusters

$$\min \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{x_i \in C_k} \sum_{x_j \in C_k} \|x_i - x_j\|^2$$

K-Means Objective: Equivalent Forms

Form 1: Within-Cluster Sum of Squares

$$\min \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Form 2: Pairwise distances within clusters

$$\min \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{x_i \in C_k} \sum_{x_j \in C_k} \|x_i - x_j\|^2$$

Equivalence:

$$\sum_{x_i \in C_k} \sum_{x_j \in C_k} \|x_i - x_j\|^2 = 2|C_k| \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

K-Means Objective: Equivalent Forms

Form 1: Within-Cluster Sum of Squares

$$\min \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Form 2: Pairwise distances within clusters

$$\min \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{x_i \in C_k} \sum_{x_j \in C_k} \|x_i - x_j\|^2$$

Equivalence:

$$\sum_{x_i \in C_k} \sum_{x_j \in C_k} \|x_i - x_j\|^2 = 2|C_k| \sum_{x_i \in C_k} \|x_i - \mu_k\|^2$$

Definition (Within-Cluster Variation)

Both forms measure how "tight" clusters are - smaller is better!

K-Means: Complexity Analysis

Time Complexity per iteration:

- **Assignment**: $O(nKd)$
  - For each of n points
  - Compute distance to K centroids
  - Each distance is $O(d)$ (dimensionality)
- **Update**: $O(nd)$
  - Sum up n points across d dimensions
  - Divide by cluster sizes

K-Means: Complexity Analysis

Time Complexity per iteration:

- **Assignment**: $O(nKd)$
  - For each of n points
  - Compute distance to K centroids
  - Each distance is $O(d)$ (dimensionality)
- **Update**: $O(nd)$
  - Sum up n points across d dimensions
  - Divide by cluster sizes

Total: $O(nKdt)$ where t = number of iterations

K-Means: Complexity Analysis

Time Complexity per iteration:

- **Assignment**: $O(nKd)$
    - For each of n points
    - Compute distance to K centroids
    - Each distance is $O(d)$ (dimensionality)
- **Update**: $O(nd)$
    - Sum up n points across d dimensions
    - Divide by cluster sizes

Total: $O(nKdt)$ where t = number of iterations
Space Complexity: $O((n + K)d)$

- Store n data points and K centroids

K-Means: Complexity Analysis

Time Complexity per iteration:

- **Assignment**: $O(nKd)$
  - For each of n points
  - Compute distance to K centroids
  - Each distance is $O(d)$ (dimensionality)
- **Update**: $O(nd)$
  - Sum up n points across d dimensions
  - Divide by cluster sizes

Total: $O(nKdt)$ where t = number of iterations
Space Complexity: $O((n + K)d)$

- Store n data points and K centroids

Key Points: Key Points

Practical: Usually converges in < 10 iterations $\rightarrow$ Very fast!

Choosing K: The Elbow Method

How Many Clusters?

Problem: K-Means requires specifying K beforehand

How Many Clusters?

Problem: K-Means requires specifying K beforehand
Question: How do we choose K?

How Many Clusters?

Problem: K-Means requires specifying K beforehand
Question: How do we choose K?
Bad idea: Just minimize WCSS

- $K = 1$: WCSS is large (one big cluster)
- $K = n$: WCSS is zero (each point is its own cluster)
- WCSS always decreases as K increases!

How Many Clusters?

Problem: K-Means requires specifying K beforehand
Question: How do we choose K?
Bad idea: Just minimize WCSS

- $K = 1$: WCSS is large (one big cluster)
- $K = n$: WCSS is zero (each point is its own cluster)
- WCSS always decreases as K increases!

Important:

Trade-off: We want low WCSS but also not too many clusters!

The Elbow Method: Intuition

Idea: Plot WCSS vs K and look for an "elbow"

The Elbow Method: Intuition

Idea: Plot WCSS vs K and look for an "elbow"
Elbow = point where:

- Adding more clusters doesn't help much
- Diminishing returns set in
- Curve flattens out

The Elbow Method: Intuition

Idea: Plot WCSS vs K and look for an "elbow"
Elbow = point where:

- Adding more clusters doesn't help much
- Diminishing returns set in
- Curve flattens out

Analogy: Hiring employees

- 1$\rightarrow$2 employees: Huge productivity gain
- 2$\rightarrow$3 employees: Still helpful
- 10$\rightarrow$11 employees: Marginal benefit
- 100$\rightarrow$101 employees: Almost no difference

The Elbow Method: Intuition

Idea: Plot WCSS vs K and look for an "elbow"
Elbow = point where:

- Adding more clusters doesn't help much
- Diminishing returns set in
- Curve flattens out

Analogy: Hiring employees

- 1→2 employees: Huge productivity gain
- 2→3 employees: Still helpful
- 10→11 employees: Marginal benefit
- 100→101 employees: Almost no difference

Key Points: Key Points

Elbow = sweet spot where benefit of adding clusters drops sharply

Elbow Method: Mathematical Formulation

Algorithm:

1. Run K-Means for $K = 1, 2, 3, \ldots, K_{\text{max}}$
2. For each K, compute WCSS(K)
3. Plot WCSS(K) vs K
4. Find "elbow" $=$ maximum curvature point

Elbow Method: Mathematical Formulation

Algorithm:

1. Run K-Means for $K = 1, 2, 3, \ldots, K_{\text{max}}$
2. For each K, compute $\text{WCSS}(K)$
3. Plot $\text{WCSS}(K)$ vs K
4. Find "elbow" = maximum curvature point

Finding elbow programmatically:

- Compute second derivative: $\frac{d^2(\text{WCSS})}{dK^2}$
- Elbow = maximum of second derivative

Elbow Method: Mathematical Formulation

Algorithm:

1. Run K-Means for $K = 1, 2, 3, \ldots, K_{\text{max}}$
2. For each K, compute $\text{WCSS}(K)$
3. Plot $\text{WCSS}(K)$ vs K
4. Find "elbow" = maximum curvature point

Finding elbow programmatically:

- Compute second derivative: $\frac{d^2(\text{WCSS})}{dK^2}$
- Elbow = maximum of second derivative

Or use heuristic:

$$\text{Elbow} = \arg\max_{K} [\text{WCSS}(K-1) - \text{WCSS}(K)]$$

(biggest drop in WCSS)

Elbow Method: Example

Example WCSS vs K:

K	WCSS
1	1000
2	600 (drop: 400)
3	400 (drop: 200)
4	300 (drop: 100) $\leftarrow$ **Elbow!**
5	250 (drop: 50)
6	220 (drop: 30)

Elbow Method: Example

Example WCSS vs K:

K	WCSS
1	1000
2	600 (drop: 400)
3	400 (drop: 200)
4	300 (drop: 100) $\leftarrow$ **Elbow!**
5	250 (drop: 50)
6	220 (drop: 30)

- Drops slow down after $K = 4$
- Suggests $K = 4$ is optimal

Elbow Method: Example

Example WCSS vs K:

K	WCSS
1	1000
2	600 (drop: 400)
3	400 (drop: 200)
4	300 (drop: 100) $\leftarrow$ **Elbow!**
5	250 (drop: 50)
6	220 (drop: 30)

- Drops slow down after $K = 4$
- Suggests $K = 4$ is optimal

Important:

Note: Elbow method is a heuristic, not a theorem!
Use domain knowledge + elbow method together.

Other Methods to Choose K

Silhouette Score:

- Measures how similar a point is to its own cluster vs other clusters
- Range: $[-1, 1]$, higher is better
- Choose K with highest average silhouette score

Other Methods to Choose K

Silhouette Score:

- Measures how similar a point is to its own cluster vs other clusters
- Range: $[-1, 1]$, higher is better
- Choose K with highest average silhouette score

Gap Statistic:

- Compare WCSS to expected WCSS under null (random) data
- Choose K where gap is largest

Other Methods to Choose K

Silhouette Score:

- Measures how similar a point is to its own cluster vs other clusters
- Range: $[-1, 1]$, higher is better
- Choose K with highest average silhouette score

Gap Statistic:

- Compare WCSS to expected WCSS under null (random) data
- Choose K where gap is largest

Domain Knowledge:

- Sometimes you know K from context
- Example: Customer segments (budget/mid/premium)
- Example: Image compression (fixed color palette)

When K-Means Fails

K-Means Assumptions

K-Means works well when:

1. Clusters are spherical (same in all directions)

K-Means Assumptions

K-Means works well when:

1. Clusters are spherical (same in all directions)
2. Clusters have similar sizes

K-Means Assumptions

K-Means works well when:

1. Clusters are spherical (same in all directions)
2. Clusters have similar sizes
3. Clusters have similar densities

K-Means Assumptions

K-Means works well when:

1. Clusters are spherical (same in all directions)
2. Clusters have similar sizes
3. Clusters have similar densities
4. Data is isotropic (no strong directional patterns)

K-Means Assumptions

K-Means works well when:

1. Clusters are spherical (same in all directions)
2. Clusters have similar sizes
3. Clusters have similar densities
4. Data is isotropic (no strong directional patterns)

Important:

Reality: Many real datasets violate these assumptions!
Let's see what happens...

Failure Mode 1: Non-Spherical Clusters

Problem: K-Means assumes clusters are "ball-shaped"

Failure Mode 1: Non-Spherical Clusters

Problem: K-Means assumes clusters are "ball-shaped"
Example: Two elongated, parallel clusters

Failure Mode 1: Non-Spherical Clusters

Problem: K-Means assumes clusters are "ball-shaped"
Example: Two elongated, parallel clusters
What happens:

- K-Means uses Euclidean distance
- Creates spherical decision boundaries
- Incorrectly splits elongated clusters

Failure Mode 1: Non-Spherical Clusters

Problem: K-Means assumes clusters are "ball-shaped"
Example: Two elongated, parallel clusters
What happens:

- K-Means uses Euclidean distance
- Creates spherical decision boundaries
- Incorrectly splits elongated clusters

Solution:

- Transform data first (e.g., PCA rotation)
- Use Gaussian Mixture Models (handles ellipsoids)
- Use spectral clustering

Failure Mode 2: Non-Convex Shapes

Problem: K-Means creates convex decision boundaries

Failure Mode 2: Non-Convex Shapes

Problem: K-Means creates convex decision boundaries
Example: Two interleaving crescents (moons)

Failure Mode 2: Non-Convex Shapes

Problem: K-Means creates convex decision boundaries
Example: Two interleaving crescents (moons)
What happens:

- Cannot separate non-convex shapes
- Centroids end up in wrong regions
- Boundaries cut through clusters

Failure Mode 2: Non-Convex Shapes

Problem: K-Means creates convex decision boundaries
Example: Two interleaving crescents (moons)
What happens:

- Cannot separate non-convex shapes
- Centroids end up in wrong regions
- Boundaries cut through clusters

Solution:

- DBSCAN (handles arbitrary shapes)
- Spectral clustering
- Kernel K-Means (kernel trick!)

Failure Mode 3: Different Cluster Sizes

Problem: K-Means prefers equal-sized clusters

Failure Mode 3: Different Cluster Sizes

Problem: K-Means prefers equal-sized clusters
Example: One large cluster + one tiny cluster

Failure Mode 3: Different Cluster Sizes

Problem: K-Means prefers equal-sized clusters
Example: One large cluster + one tiny cluster
What happens:

- Large cluster gets split into multiple parts
- Small cluster gets merged with nearby large cluster
- Objective function is minimized, but result is wrong!

Failure Mode 3: Different Cluster Sizes

Problem: K-Means prefers equal-sized clusters
Example: One large cluster + one tiny cluster
What happens:

- Large cluster gets split into multiple parts
- Small cluster gets merged with nearby large cluster
- Objective function is minimized, but result is wrong!

Why?

- Splitting large cluster reduces more WCSS
- Algorithm is greedy - doesn't know true structure

Failure Mode 3: Different Cluster Sizes

Problem: K-Means prefers equal-sized clusters
Example: One large cluster + one tiny cluster
What happens:

- Large cluster gets split into multiple parts
- Small cluster gets merged with nearby large cluster
- Objective function is minimized, but result is wrong!

Why?

- Splitting large cluster reduces more WCSS
- Algorithm is greedy - doesn't know true structure

Solution:

- Weighted K-Means
- DBSCAN (density-based)
- Hierarchical clustering

Failure Mode 4: Different Densities

Problem: K-Means assumes similar density across clusters

Failure Mode 4: Different Densities

Problem: K-Means assumes similar density across clusters
Example: Dense cluster next to sparse cluster

Failure Mode 4: Different Densities

Problem: K-Means assumes similar density across clusters
Example: Dense cluster next to sparse cluster
What happens:

- Dense cluster may be over-segmented
- Sparse cluster absorbs nearby points from dense cluster
- Boundary is placed incorrectly

Failure Mode 4: Different Densities

Problem: K-Means assumes similar density across clusters
Example: Dense cluster next to sparse cluster
What happens:

- Dense cluster may be over-segmented
- Sparse cluster absorbs nearby points from dense cluster
- Boundary is placed incorrectly

Solution:

- DBSCAN (explicitly models density)
- HDBSCAN (hierarchical density-based)
- GMM with different covariances

Failure Mode 5: Outliers

Problem: K-Means uses squared distances - very sensitive to outliers!

Failure Mode 5: Outliers

Problem: K-Means uses squared distances - very sensitive to outliers!
Example: Clean cluster + a few outliers

Failure Mode 5: Outliers

Problem: K-Means uses squared distances - very sensitive to outliers!
Example: Clean cluster + a few outliers
What happens:

- Outliers "pull" centroids away from true cluster centers
- Can create spurious clusters for noise points
- Squared distance amplifies effect ($100^2 = 10000$!)

Failure Mode 5: Outliers

Problem: K-Means uses squared distances - very sensitive to outliers!
Example: Clean cluster + a few outliers
What happens:

- Outliers "pull" centroids away from true cluster centers
- Can create spurious clusters for noise points
- Squared distance amplifies effect ($100^2 = 10000!$)

Solutions:

- Pre-process: Remove outliers first
- K-Medoids (uses medians, more robust)
- Trimmed K-Means (ignores worst $\alpha\%$ of points)
- DBSCAN (marks outliers as noise)

Failure Mode 6: Bad Initialization

Problem: Random initialization can lead to poor local minima

Failure Mode 6: Bad Initialization

Problem: Random initialization can lead to poor local minima
Example: All initial centroids in same region

Failure Mode 6: Bad Initialization

Problem: Random initialization can lead to poor local minima
Example: All initial centroids in same region
What happens:

- Converges to nearby local optimum
- Misses true clusters far away
- Different runs give different results!

Failure Mode 6: Bad Initialization

Problem: Random initialization can lead to poor local minima
Example: All initial centroids in same region
What happens:

- Converges to nearby local optimum
- Misses true clusters far away
- Different runs give different results!

Solution: K-Means++ (discussed next!)

- Smart initialization - spread out initial centroids
- Much more likely to find good solution
- Now default in most libraries

K-Means++

K-Means++: Smarter Initialization

Problem with random init: Can start with all centroids clustered together

K-Means++: Smarter Initialization

Problem with random init: Can start with all centroids clustered together
K-Means++ Idea: Choose initial centroids that are far apart

K-Means++: Smarter Initialization

Problem with random init: Can start with all centroids clustered together
K-Means++ Idea: Choose initial centroids that are far apart
Key Insight:

- If true clusters are far apart
- Initial centroids should also be far apart
- More likely to have one centroid per true cluster

K-Means++: Smarter Initialization

Problem with random init: Can start with all centroids clustered together
K-Means++ Idea: Choose initial centroids that are far apart
Key Insight:

- If true clusters are far apart
- Initial centroids should also be far apart
- More likely to have one centroid per true cluster

Key Points: Key Points

How to choose "far apart" centroids?
Use weighted random sampling - points far from existing centroids have higher probability!

K-Means++ Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$, number of clusters K

K-Means++ Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$, number of clusters K
Initialization:

1. Choose first centroid μ_1 uniformly at random from data

K-Means++ Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$, number of clusters K
Initialization:

1. Choose first centroid μ_1 uniformly at random from data
2. For $k = 2, 3, \ldots, K$:

K-Means++ Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$, number of clusters K
Initialization:

1. Choose first centroid μ_1 uniformly at random from data
2. For $k = 2, 3, \ldots, K$:
    1) For each point x_i, compute:

$$D(x_i) = \min_{j<k} \|x_i - \mu_j\|^2$$

(squared distance to nearest already-chosen centroid)

K-Means++ Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$, number of clusters K
Initialization:

1. Choose first centroid μ_1 uniformly at random from data
2. For $k = 2, 3, \ldots, K$:
   1) For each point x_i, compute:

      $$D(x_i) = \min_{j<k} \|x_i - \mu_j\|^2$$

      (squared distance to nearest already-chosen centroid)
   2) Choose next centroid μ_k with probability:

      $$P(x_i) = \frac{D(x_i)}{\sum_{j=1}^{n} D(x_j)}$$

K-Means++ Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$, number of clusters K
Initialization:

1. Choose first centroid μ_1 uniformly at random from data
2. For $k = 2, 3, \ldots, K$:
    1) For each point x_i, compute:

    $$D(x_i) = \min_{j<k} \|x_i - \mu_j\|^2$$

    (squared distance to nearest already-chosen centroid)
    2) Choose next centroid μ_k with probability:

    $$P(x_i) = \frac{D(x_i)}{\sum_{j=1}^{n} D(x_j)}$$

Then: Run standard K-Means with these initial centroids

K-Means++: Why It Works

Intuition:

- Points far from existing centroids have high $D(x_i)^2$

K-Means++: Why It Works

Intuition:

- Points far from existing centroids have high $D(x_i)^2$
- Points near existing centroids have low $D(x_i)^2$

K-Means++: Why It Works

Intuition:

- Points far from existing centroids have high $D(x_i)^2$
- Points near existing centroids have low $D(x_i)^2$
- Weighted sampling favors distant points

K-Means++: Why It Works

Intuition:

- Points far from existing centroids have high $D(x_i)^2$
- Points near existing centroids have low $D(x_i)^2$
- Weighted sampling favors distant points
- Result: Centroids spread out across data

K-Means++: Why It Works

Intuition:

- Points far from existing centroids have high $D(x_i)^2$
- Points near existing centroids have low $D(x_i)^2$
- Weighted sampling favors distant points
- Result: Centroids spread out across data

Example: Concrete Example

Suppose we've chosen $\mu_1 = (0, 0)$. Consider two points:

- $x_a = (1, 0)$: $D(x_a) = 1 \rightarrow P(x_a) \propto 1$
- $x_b = (10, 0)$: $D(x_b) = 100 \rightarrow P(x_b) \propto 100$

Point x_b is $100\times$ more likely to be chosen as μ_2!

K-Means++: Theoretical Guarantee

Theorem (Arthur & Vassilvitskii, 2007):
K-Means++ initialization gives an expected approximation ratio of $O(\log K)$ to the optimal clustering.

K-Means++: Theoretical Guarantee

Theorem (Arthur & Vassilvitskii, 2007):
K-Means++ initialization gives an expected approximation ratio of $O(\log K)$ to the optimal clustering.
In English:

- Let $\text{WCSS}_{\text{optimal}} =$ best possible WCSS
- Let $\text{WCSS}_{\text{KM++}} =$ WCSS from K-Means++ (after convergence)
- Then:

$$\mathbb{E}[\text{WCSS}_{\text{KM++}}] \leq O(\log K) \cdot \text{WCSS}_{\text{optimal}}$$

K-Means++: Theoretical Guarantee

Theorem (Arthur & Vassilvitskii, 2007):
K-Means++ initialization gives an expected approximation ratio of $O(\log K)$ to the optimal clustering.
In English:

- Let $\text{WCSS}_{\text{optimal}}$ = best possible WCSS
- Let $\text{WCSS}_{\text{KM++}}$ = WCSS from K-Means++ (after convergence)
- Then:

$$\mathbb{E}[\text{WCSS}_{\text{KM++}}] \leq O(\log K) \cdot \text{WCSS}_{\text{optimal}}$$

Random initialization: No such guarantee!

K-Means++: Theoretical Guarantee

Theorem (Arthur & Vassilvitskii, 2007):
K-Means++ initialization gives an expected approximation ratio of $O(\log K)$ to the optimal clustering.
In English:

- Let $\text{WCSS}_{\text{optimal}}$ = best possible WCSS
- Let $\text{WCSS}_{\text{KM++}}$ = WCSS from K-Means++ (after convergence)
- Then:

$$\mathbb{E}[\text{WCSS}_{\text{KM++}}] \leq O(\log K) \cdot \text{WCSS}_{\text{optimal}}$$

Random initialization: No such guarantee!

Key Points: Key Points

K-Means++ is provably better than random initialization
This is why it's now the default in scikit-learn, etc.

K-Means++ vs Standard K-Means

Property	**Random Init**	**K-Means++**
Convergence guarantee	Yes	Yes
Quality guarantee	No	$O(\log K)$-approx
Iterations needed	More	Fewer
Consistency across runs	Poor	Better
Initialization time	$O(1)$	$O(nKd)$
Total time	$O(nKdt)$	$O(nKd(t+K))$

K-Means++ vs Standard K-Means

Property	**Random Init**	**K-Means++**
Convergence guarantee	Yes	Yes
Quality guarantee	No	$O(\log K)$-approx
Iterations needed	More	Fewer
Consistency across runs	Poor	Better
Initialization time	$O(1)$	$O(nKd)$
Total time	$O(nKdt)$	$O(nKd(t+K))$

Practical Advice:

- Always use K-Means++ (default in scikit-learn)
- Initialization cost is negligible compared to iterations
- Better results with fewer iterations

Hierarchical Clustering

Limitations of K-Means

K-Means requires:

- Specifying K beforehand
- Running multiple times with different K (elbow method)
- Assumes spherical clusters

Limitations of K-Means

K-Means requires:

- Specifying K beforehand
- Running multiple times with different K (elbow method)
- Assumes spherical clusters

Can we do better?

- Get clustering at all levels of granularity
- From $K = n$ (each point alone) to $K = 1$ (everything together)
- Visualize as a tree (dendrogram)
- Cut tree at any height to get desired K

Limitations of K-Means

K-Means requires:

- Specifying K beforehand
- Running multiple times with different K (elbow method)
- Assumes spherical clusters

Can we do better?

- Get clustering at all levels of granularity
- From $K = n$ (each point alone) to $K = 1$ (everything together)
- Visualize as a tree (dendrogram)
- Cut tree at any height to get desired K

Key Points: Key Points

Hierarchical Clustering: Build a tree of nested clusters!

Two Approaches to Hierarchical Clustering

1. Agglomerative (Bottom-Up)

- Start: *n* clusters (each point alone)
- Repeat: Merge two closest clusters
- End: 1 cluster (everything together)
- Most common approach

Two Approaches to Hierarchical Clustering

1. Agglomerative (Bottom-Up)

- Start: *n* clusters (each point alone)
- Repeat: Merge two closest clusters
- End: 1 cluster (everything together)
- Most common approach

Intuition: Like building from atoms to molecules to structures

Two Approaches to Hierarchical Clustering

1. Agglomerative (Bottom-Up)

- Start: n clusters (each point alone)
- Repeat: Merge two closest clusters
- End: 1 cluster (everything together)
- Most common approach

Intuition: Like building from atoms to molecules to structures

2. Divisive (Top-Down)

- Start: 1 cluster (everything together)
- Repeat: Split cluster into two
- End: n clusters (each point alone)
- Less common (computationally harder)

Two Approaches to Hierarchical Clustering

1. Agglomerative (Bottom-Up)

- Start: n clusters (each point alone)
- Repeat: Merge two closest clusters
- End: 1 cluster (everything together)
- Most common approach

Intuition: Like building from atoms to molecules to structures

2. Divisive (Top-Down)

- Start: 1 cluster (everything together)
- Repeat: Split cluster into two
- End: n clusters (each point alone)
- Less common (computationally harder)

Intuition: Like breaking a rock into smaller and smaller pieces

Two Approaches to Hierarchical Clustering

1. Agglomerative (Bottom-Up)

- Start: n clusters (each point alone)
- Repeat: Merge two closest clusters
- End: 1 cluster (everything together)
- Most common approach

Intuition: Like building from atoms to molecules to structures

2. Divisive (Top-Down)

- Start: 1 cluster (everything together)
- Repeat: Split cluster into two
- End: n clusters (each point alone)
- Less common (computationally harder)

Intuition: Like breaking a rock into smaller and smaller pieces

Important:

Agglomerative Clustering: Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$
Algorithm:

1. **Initialize**: Start with n clusters, one per point

$$C_i = \{x_i\} \quad \text{for } i = 1, \ldots, n$$

Agglomerative Clustering: Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$
Algorithm:

1. **Initialize**: Start with n clusters, one per point
$$C_i = \{x_i\} \quad \text{for } i = 1, \ldots, n$$
2. **Repeat** until only 1 cluster remains:

Agglomerative Clustering: Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$
Algorithm:

1. **Initialize**: Start with n clusters, one per point

$$C_i = \{x_i\} \quad \text{for } i = 1, \ldots, n$$

2. **Repeat** until only 1 cluster remains:
   1) Find the two "closest" clusters C_i and C_j

Agglomerative Clustering: Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$
Algorithm:

1. **Initialize**: Start with n clusters, one per point
$$C_i = \{x_i\} \quad \text{for } i = 1, \ldots, n$$
2. **Repeat** until only 1 cluster remains:
    1) Find the two "closest" clusters C_i and C_j
    2) Merge them: $C_{\text{new}} = C_i \cup C_j$

Agglomerative Clustering: Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$
Algorithm:

1. **Initialize**: Start with n clusters, one per point

$$C_i = \{x_i\} \quad \text{for } i = 1, \ldots, n$$

2. **Repeat** until only 1 cluster remains:
    1) Find the two "closest" clusters C_i and C_j
    2) Merge them: $C_{\text{new}} = C_i \cup C_j$
    3) Record merge in tree structure (dendrogram)

Agglomerative Clustering: Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$
Algorithm:

1. **Initialize**: Start with n clusters, one per point
$$C_i = \{x_i\} \quad \text{for } i = 1, \ldots, n$$
2. **Repeat** until only 1 cluster remains:
    1) Find the two "closest" clusters C_i and C_j
    2) Merge them: $C_{\text{new}} = C_i \cup C_j$
    3) Record merge in tree structure (dendrogram)
3. **Output**: Dendrogram (tree showing all merges)

Agglomerative Clustering: Algorithm

Input: Data points $\{x_1, \ldots, x_n\}$
Algorithm:

1. **Initialize**: Start with n clusters, one per point
$$C_i = \{x_i\} \quad \text{for } i = 1, \ldots, n$$
2. **Repeat** until only 1 cluster remains:
    1) Find the two "closest" clusters C_i and C_j
    2) Merge them: $C_{\text{new}} = C_i \cup C_j$
    3) Record merge in tree structure (dendrogram)
3. **Output**: Dendrogram (tree showing all merges)

Key Points: Key Points

Key Question: How do we measure distance between clusters (not just points)?

Linkage Criteria: Measuring Cluster Distance

Given two clusters C_i and C_j, what is their distance?

Linkage Criteria: Measuring Cluster Distance

Given two clusters C_i and C_j, what is their distance?

1. Single Linkage (Minimum):

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|$$

- Distance between closest points
- Can create "chains" (long, thin clusters)

Linkage Criteria: Measuring Cluster Distance

Given two clusters C_i and C_j, what is their distance?

1. **Single Linkage** (Minimum):

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|$$

- Distance between closest points
- Can create "chains" (long, thin clusters)

2. **Complete Linkage** (Maximum):

$$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|$$

- Distance between farthest points
- Creates compact, spherical clusters

Linkage Criteria: Measuring Cluster Distance

Given two clusters C_i and C_j, what is their distance?

1. **Single Linkage** (Minimum):

$$d(C_i, C_j) = \min_{x \in C_i, y \in C_j} \|x - y\|$$

- Distance between closest points
- Can create "chains" (long, thin clusters)

2. **Complete Linkage** (Maximum):

$$d(C_i, C_j) = \max_{x \in C_i, y \in C_j} \|x - y\|$$

- Distance between farthest points
- Creates compact, spherical clusters

3. **Average Linkage**:

$$d(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum \sum \|x - y\|$$

More Linkage Criteria

4. Centroid Linkage:

$$d(C_i, C_j) = \|\mu_i - \mu_j\|$$

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

- Distance between centroids
- Can have inversions (not monotonic!)

More Linkage Criteria

4. Centroid Linkage:

$$d(C_i, C_j) = \|\mu_i - \mu_j\|$$

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

- Distance between centroids
- Can have inversions (not monotonic!)

5. Ward's Method (Minimum Variance):

$$d(C_i, C_j) = \frac{|C_i||C_j|}{|C_i| + |C_j|} \|\mu_i - \mu_j\|^2$$

- Minimizes increase in total within-cluster variance
- Tends to create equal-sized clusters
- Most popular choice in practice

More Linkage Criteria

4. Centroid Linkage:

$$d(C_i, C_j) = \|\mu_i - \mu_j\|$$

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

- Distance between centroids
- Can have inversions (not monotonic!)

5. Ward's Method (Minimum Variance):

$$d(C_i, C_j) = \frac{|C_i||C_j|}{|C_i| + |C_j|} \|\mu_i - \mu_j\|^2$$

- Minimizes increase in total within-cluster variance
- Tends to create equal-sized clusters
- Most popular choice in practice

Key Points: Key Points

Dendrogram: Visualizing the Hierarchy

Dendrogram = Tree diagram showing merge history

Dendrogram: Visualizing the Hierarchy

Dendrogram = Tree diagram showing merge history
Reading a dendrogram:

- **Bottom**: Individual points
- **Height**: Distance at which clusters merge
- **Horizontal cut**: Determines number of clusters

Dendrogram: Visualizing the Hierarchy

Dendrogram = Tree diagram showing merge history
Reading a dendrogram:

- **Bottom**: Individual points
- **Height**: Distance at which clusters merge
- **Horizontal cut**: Determines number of clusters

Example:

- Cut at height $h = 2$: Might give 5 clusters
- Cut at height $h = 5$: Might give 2 clusters
- Cut at height $h = 10$: Everything in 1 cluster

Dendrogram: Visualizing the Hierarchy

Dendrogram = Tree diagram showing merge history
Reading a dendrogram:

- **Bottom**: Individual points
- **Height**: Distance at which clusters merge
- **Horizontal cut**: Determines number of clusters

Example:

- Cut at height $h = 2$: Might give 5 clusters
- Cut at height $h = 5$: Might give 2 clusters
- Cut at height $h = 10$: Everything in 1 cluster

Example: Flexibility

Advantage: Don't need to specify K beforehand!
Run once, then choose K by cutting dendrogram at desired height.

Hierarchical Clustering: Complexity

Naive Algorithm:

- Need to merge $n-1$ times
- At each step, compute distances between all cluster pairs
- Worst case: $O(n^3)$ time

Hierarchical Clustering: Complexity

Naive Algorithm:

- Need to merge $n - 1$ times
- At each step, compute distances between all cluster pairs
- Worst case: $O(n^3)$ time

Optimized (using priority queue):

- Time: $O(n^2 \log n)$
- Space: $O(n^2)$ (store pairwise distances)

Hierarchical Clustering: Complexity

Naive Algorithm:

- Need to merge $n-1$ times
- At each step, compute distances between all cluster pairs
- Worst case: $O(n^3)$ time

Optimized (using priority queue):

- Time: $O(n^2 \log n)$
- Space: $O(n^2)$ (store pairwise distances)

Comparison to K-Means:

	K-Means	**Hierarchical**
Time	$O(nKdt)$	$O(n^2 \log n)$
Space	$O((n+K)d)$	$O(n^2)$

Hierarchical Clustering: Complexity

Naive Algorithm:

- Need to merge $n-1$ times
- At each step, compute distances between all cluster pairs
- Worst case: $O(n^3)$ time

Optimized (using priority queue):

- Time: $O(n^2 \log n)$
- Space: $O(n^2)$ (store pairwise distances)

Comparison to K-Means:

	K-Means	**Hierarchical**
Time	$O(nKdt)$	$O(n^2 \log n)$
Space	$O((n+K)d)$	$O(n^2)$

Important: H

ierarchical is much slower for large n (thousands+)

K-Means scales better to large datasets

Hierarchical vs K-Means: When to Use Each?

Property	**K-Means**	**Hierarchical**
Specify K?	Yes	No
Speed (large n)	Fast	Slow
Deterministic?	No	Yes
Cluster shape	Spherical	Flexible
Dendrogram?	No	Yes
Scalability	Excellent	Poor

Hierarchical vs K-Means: When to Use Each?

Property	**K-Means**	**Hierarchical**
Specify K?	Yes	No
Speed (large n)	Fast	Slow
Deterministic?	No	Yes
Cluster shape	Spherical	Flexible
Dendrogram?	No	Yes
Scalability	Excellent	Poor

Use K-Means when:

- Large dataset ($n > 10000$)
- Roughly spherical clusters
- Know K (or can use elbow method)

Hierarchical vs K-Means: When to Use Each?

Property	**K-Means**	**Hierarchical**
Specify K?	Yes	No
Speed (large n)	Fast	Slow
Deterministic?	No	Yes
Cluster shape	Spherical	Flexible
Dendrogram?	No	Yes
Scalability	Excellent	Poor

Use K-Means when:

- Large dataset ($n > 10000$)
- Roughly spherical clusters
- Know K (or can use elbow method)

Use Hierarchical when:

- Small-medium dataset ($n < 5000$)
- Want full hierarchy (dendrogram)
- Don't know K beforehand

DBSCAN

Limitations of K-Means and Hierarchical

Both algorithms struggle with:

- Non-convex shapes (crescents, rings)

Limitations of K-Means and Hierarchical

Both algorithms struggle with:

- Non-convex shapes (crescents, rings)
- Outliers and noise

Limitations of K-Means and Hierarchical

Both algorithms struggle with:

- Non-convex shapes (crescents, rings)
- Outliers and noise
- Varying density clusters

Limitations of K-Means and Hierarchical

Both algorithms struggle with:

- Non-convex shapes (crescents, rings)
- Outliers and noise
- Varying density clusters

Example: Motivating Example

Consider two clusters:

- Dense core of points (100 points in small area)
- Scattered points around core (10 points far away)

Question: Should scattered points be their own clusters or noise?

Limitations of K-Means and Hierarchical

Both algorithms struggle with:

- Non-convex shapes (crescents, rings)
- Outliers and noise
- Varying density clusters

Example: Motivating Example

Consider two clusters:

- Dense core of points (100 points in small area)
- Scattered points around core (10 points far away)

Question: Should scattered points be their own clusters or noise?

DBSCAN: Density-Based Spatial Clustering

Core Idea:
A cluster is a region where points are densely packed together

DBSCAN: Density-Based Spatial Clustering

Core Idea:

A cluster is a region where points are densely packed together

Advantages:

- Can find arbitrarily shaped clusters
- Robust to outliers (marks them as noise)
- No need to specify number of clusters K
- Deterministic (same result every time)

DBSCAN: Density-Based Spatial Clustering

Core Idea:
A cluster is a region where points are densely packed together

Advantages:

- Can find arbitrarily shaped clusters
- Robust to outliers (marks them as noise)
- No need to specify number of clusters K
- Deterministic (same result every time)

Parameters:

- ε (epsilon): Radius of neighborhood
- MinPts: Minimum number of points in neighborhood to be "dense"

DBSCAN: Key Definitions

1. ε-**neighborhood** of point p:

$$N_\varepsilon(p) = \{q : \|p - q\| \leq \varepsilon\}$$

All points within radius ε of p

DBSCAN: Key Definitions

1. ε-**neighborhood** of point p:

$$N_\varepsilon(p) = \{q : \|p - q\| \leq \varepsilon\}$$

All points within radius ε of p

2. Core point:

- Point p is a core point if $|N_\varepsilon(p)| \geq \text{MinPts}$
- Has at least MinPts neighbors (including itself)
- These form the "dense regions"

DBSCAN: Key Definitions

1. ε**-neighborhood** of point p:

$$N_\varepsilon(p) = \{q : \|p - q\| \leq \varepsilon\}$$

All points within radius ε of p

2. Core point:

- Point p is a core point if $|N_\varepsilon(p)| \geq \text{MinPts}$
- Has at least MinPts neighbors (including itself)
- These form the "dense regions"

3. Border point:

- Not a core point
- But is in $N_\varepsilon(p)$ of some core point p
- On the "edge" of a cluster

DBSCAN: Key Definitions

1. **ε-neighborhood** of point p:

$$N_\varepsilon(p) = \{q : \|p - q\| \leq \varepsilon\}$$

All points within radius ε of p

2. **Core point**:

- Point p is a core point if $|N_\varepsilon(p)| \geq \text{MinPts}$
- Has at least MinPts neighbors (including itself)
- These form the "dense regions"

3. **Border point**:

- Not a core point
- But is in $N_\varepsilon(p)$ of some core point p
- On the "edge" of a cluster

4. **Noise point**:

- Neither core nor border
- Isolated outliers

DBSCAN: Visual Example of Point Types

Setup: $\varepsilon = 1$, MinPts $= 4$

[Visual would show: points with circles of radius ε]

DBSCAN: Visual Example of Point Types

Setup: $\varepsilon = 1$, MinPts $= 4$

[Visual would show: points with circles of radius ε]

Point Classification:

- **Point A**: Has 5 neighbors $\rightarrow$ Core point
- **Point B**: Has 6 neighbors $\rightarrow$ Core point
- **Point C**: Has 2 neighbors, but in neighborhood of A $\rightarrow$ Border point
- **Point D**: Has 1 neighbor, not near any core $\rightarrow$ Noise

DBSCAN: Visual Example of Point Types

Setup: $\varepsilon = 1$, MinPts $= 4$

[Visual would show: points with circles of radius ε]

Point Classification:

- **Point A**: Has 5 neighbors $\rightarrow$ Core point
- **Point B**: Has 6 neighbors $\rightarrow$ Core point
- **Point C**: Has 2 neighbors, but in neighborhood of A $\rightarrow$ Border point
- **Point D**: Has 1 neighbor, not near any core $\rightarrow$ Noise

Key Points: Key Points

Cluster = All core points connected + their border points

DBSCAN: Density Connectivity

Direct density-reachable:

- Point q is directly density-reachable from p if:
  1. p is a core point
  2. $q \in N_{\varepsilon}(p)$

DBSCAN: Density Connectivity

Direct density-reachable:

- Point q is directly density-reachable from p if:
  1. p is a core point
  2. $q \in N_\varepsilon(p)$

Density-reachable:

- q is density-reachable from p if there's a chain:

$$p = p_1, p_2, \ldots, p_n = q$$

where each p_{i+1} is directly density-reachable from p_i

DBSCAN: Density Connectivity

Direct density-reachable:

- Point q is directly density-reachable from p if:
  1. p is a core point
  2. $q \in N_\varepsilon(p)$

Density-reachable:

- q is density-reachable from p if there's a chain:

$$p = p_1, p_2, \ldots, p_n = q$$

  where each p_{i+1} is directly density-reachable from p_i

Density-connected:

- Points p and q are density-connected if both are density-reachable from some point o

DBSCAN: Density Connectivity

Direct density-reachable:

- Point q is directly density-reachable from p if:
  1. p is a core point
  2. $q \in N_\varepsilon(p)$

Density-reachable:

- q is density-reachable from p if there's a chain:

$$p = p_1, p_2, \ldots, p_n = q$$

  where each p_{i+1} is directly density-reachable from p_i

Density-connected:

- Points p and q are density-connected if both are density-reachable from some point o

Definition (DBSCAN Cluster)

A **cluster** is the maximal set of density-connected points.

DBSCAN Algorithm

Input: Data points, ε, MinPts
Algorithm:

1. Mark all points as `UNVISITED`

DBSCAN Algorithm

Input: Data points, ε, MinPts
Algorithm:

1. Mark all points as `UNVISITED`
2. **For each** unvisited point *p*:

DBSCAN Algorithm

Input: Data points, ε, MinPts
Algorithm:

1. Mark all points as `UNVISITED`
2. **For each** unvisited point *p*:
    1) Mark *p* as `VISITED`

DBSCAN Algorithm

Input: Data points, ε, MinPts
Algorithm:

1. Mark all points as `UNVISITED`
2. **For each** unvisited point p:
    1) Mark p as `VISITED`
    2) Find $N_\varepsilon(p)$ (all neighbors within ε)

DBSCAN Algorithm

Input: Data points, ε, MinPts
Algorithm:

1. Mark all points as `UNVISITED`
2. **For each** unvisited point p:
    1) Mark p as `VISITED`
    2) Find $N_\varepsilon(p)$ (all neighbors within ε)
    3) **If** $|N_\varepsilon(p)| <$ MinPts:
        - Mark p as `NOISE` (may change later!)

DBSCAN Algorithm

Input: Data points, ε, MinPts
Algorithm:

```
1. Mark all points as UNVISITED
2. For each unvisited point $p$:
    1) Mark $p$ as VISITED
    2) Find $N_\varepsilon(p)$ (all neighbors within $\varepsilon$)
    3) If $|N_\varepsilon(p)| < \text{MinPts}$:
        – Mark $p$ as NOISE (may change later!)
    4) Else ($p$ is core point):
        – Create new cluster $C$
        – Add $p$ to $C$
        – Expand cluster: recursively add all density-reachable points
```

DBSCAN Algorithm

Input: Data points, ε, MinPts
Algorithm:

1. Mark all points as `UNVISITED`
2. **For each** unvisited point p:
   1) Mark p as `VISITED`
   2) Find $N_\varepsilon(p)$ (all neighbors within ε)
   3) **If** $|N_\varepsilon(p)| <$ MinPts:
      - Mark p as `NOISE` (may change later!)
   4) **Else** (p is core point):
      - Create new cluster C
      - Add p to C
      - Expand cluster: recursively add all density-reachable points
3. **Output**: Set of clusters + noise points

DBSCAN: Expand Cluster Procedure

ExpandCluster$(p, N_\varepsilon(p), C)$:

1. Add p to cluster C
2. **For each** point $q \in N_\varepsilon(p)$:
   1) **If** q is `UNVISITED`:
      - Mark q as `VISITED`
      - Find $N_\varepsilon(q)$
      - **If** $|N_\varepsilon(q)| \geq \text{MinPts}$:
      - q is a core point!
      - Add $N_\varepsilon(q)$ to queue (will expand recursively)
   2) **If** q not yet in any cluster:
      - Add q to C

DBSCAN: Expand Cluster Procedure

ExpandCluster$(p, N_\varepsilon(p), C)$:

1. Add p to cluster C
2. **For each** point $q \in N_\varepsilon(p)$:
   1) **If** q is `UNVISITED`:
      - Mark q as `VISITED`
      - Find $N_\varepsilon(q)$
      - **If** $|N_\varepsilon(q)| \geq$ MinPts:
      - q is a core point!
      - Add $N_\varepsilon(q)$ to queue (will expand recursively)
   2) **If** q not yet in any cluster:
      - Add q to C

Key Points: Key Points

Result: All density-reachable points from p are in cluster C

DBSCAN: Worked Example (1/4)

Setup: $\varepsilon = 1.5$, MinPts $= 3$
Data: 10 points in 2D

Point	Coords	Point	Coords
A	$(1, 1)$	F	$(2, 2)$
B	$(1, 2)$	G	$(10, 10)$
C	$(2, 1)$	H	$(10, 11)$
D	$(1.5, 1.5)$	I	$(11, 10)$
E	$(2, 3)$	J	$(5, 5)$

Visualize: Three regions - dense cluster left, dense cluster right, one isolated point

DBSCAN: Worked Example (2/4) - Finding Core Points

Check each point's neighborhood:

- **Point A** $(1, 1)$: Neighbors = $\{$A, B, C, D$\}$ (4 points) $\rightarrow$ Core
- **Point B** $(1, 2)$: Neighbors = $\{$A, B, D, E, F$\}$ (5 points) $\rightarrow$ Core
- **Point C** $(2, 1)$: Neighbors = $\{$A, C, D, F$\}$ (4 points) $\rightarrow$ Core
- **Point D** $(1.5, 1.5)$: Neighbors = $\{$A, B, C, D, F$\}$ (5 points) $\rightarrow$ Core
- **Point E** $(2, 3)$: Neighbors = $\{$B, E, F$\}$ (3 points) $\rightarrow$ Core
- **Point F** $(2, 2)$: Neighbors = $\{$B, C, D, E, F$\}$ (5 points) $\rightarrow$ Core
- **Point G** $(10, 10)$: Neighbors = $\{$G, H, I$\}$ (3 points) $\rightarrow$ Core
- **Point H** $(10, 11)$: Neighbors = $\{$G, H, I$\}$ (3 points) $\rightarrow$ Core
- **Point I** $(11, 10)$: Neighbors = $\{$G, H, I$\}$ (3 points) $\rightarrow$ Core
- **Point J** $(5, 5)$: Neighbors = $\{$J$\}$ (1 point) $\rightarrow$ Noise

DBSCAN: Worked Example (3/4) - Forming Clusters

Start with point A:

- A is core $\rightarrow$ Create **Cluster 1**
- Expand: A can reach B, C, D (all core)
- B can reach E, F (both core)
- Continue expanding: All of {A, B, C, D, E, F} are density-connected
- **Cluster 1** = {A, B, C, D, E, F}

DBSCAN: Worked Example (3/4) - Forming Clusters

Start with point A:

- A is core $\rightarrow$ Create **Cluster 1**
- Expand: A can reach B, C, D (all core)
- B can reach E, F (both core)
- Continue expanding: All of {A, B, C, D, E, F} are density-connected
- **Cluster 1** = {A, B, C, D, E, F}

Continue with unvisited point G:

- G is core $\rightarrow$ Create **Cluster 2**
- Expand: G can reach H, I (both core)
- All of {G, H, I} are density-connected
- **Cluster 2** = {G, H, I}

DBSCAN: Worked Example (3/4) - Forming Clusters

Start with point A:

- A is core $\rightarrow$ Create **Cluster 1**
- Expand: A can reach B, C, D (all core)
- B can reach E, F (both core)
- Continue expanding: All of {A, B, C, D, E, F} are density-connected
- **Cluster 1** = {A, B, C, D, E, F}

Continue with unvisited point G:

- G is core $\rightarrow$ Create **Cluster 2**
- Expand: G can reach H, I (both core)
- All of {G, H, I} are density-connected
- **Cluster 2** = {G, H, I}

DBSCAN: Worked Example (4/4) - Final Result

Final Clustering:

- **Cluster 1**: $\{$A, B, C, D, E, F$\}$ (6 points)
- **Cluster 2**: $\{$G, H, I$\}$ (3 points)
- **Noise**: $\{$J$\}$ (1 point)

DBSCAN: Worked Example (4/4) - Final Result

Final Clustering:

- **Cluster 1**: {A, B, C, D, E, F} (6 points)
- **Cluster 2**: {G, H, I} (3 points)
- **Noise**: {J} (1 point)

Key Points: Key Points

Key Observations:

- Found 2 clusters automatically (no K specified!)
- Identified outlier J as noise
- Clusters can have different sizes (6 vs 3)
- Clusters based on density, not distance to centroid

DBSCAN: Worked Example (4/4) - Final Result

Final Clustering:

- **Cluster 1**: {A, B, C, D, E, F} (6 points)
- **Cluster 2**: {G, H, I} (3 points)
- **Noise**: {J} (1 point)

Key Points: Key Points

Key Observations:

- Found 2 clusters automatically (no K specified!)
- Identified outlier J as noise
- Clusters can have different sizes (6 vs 3)
- Clusters based on density, not distance to centroid

DBSCAN: Advantages

1. **Arbitrary shapes**: Can find non-convex, elongated clusters
2. **No K needed**: Number of clusters determined automatically
3. **Noise detection**: Explicitly identifies outliers
4. **Deterministic**: Same result every run
5. **Only 2 parameters**: ε and MinPts (vs K for K-Means)

DBSCAN: Advantages

1. **Arbitrary shapes**: Can find non-convex, elongated clusters
2. **No K needed**: Number of clusters determined automatically
3. **Noise detection**: Explicitly identifies outliers
4. **Deterministic**: Same result every run
5. **Only 2 parameters**: ε and MinPts (vs K for K-Means)

Example Applications:

- Geographic data (cities, disease outbreaks)
- Anomaly detection (fraud, network intrusion)
- Image segmentation (arbitrary region shapes)
- Point cloud processing (3D scanning)

DBSCAN: Disadvantages

1. **Parameter sensitivity**: Results depend heavily on ε and MinPts
2. **Varying density**: Struggles when clusters have different densities
3. **High dimensions**: "Curse of dimensionality" - distances become meaningless
4. **Complexity**: $O(n^2)$ naive, $O(n \log n)$ with spatial index

DBSCAN: Disadvantages

1. **Parameter sensitivity**: Results depend heavily on ε and MinPts
2. **Varying density**: Struggles when clusters have different densities
3. **High dimensions**: "Curse of dimensionality" - distances become meaningless
4. **Complexity**: $O(n^2)$ naive, $O(n \log n)$ with spatial index

   Example of varying density problem:

- Dense cluster: 100 points in radius 1
- Sparse cluster: 10 points in radius 5
- No single ε works well for both!

DBSCAN: Disadvantages

1. **Parameter sensitivity**: Results depend heavily on ε and MinPts
2. **Varying density**: Struggles when clusters have different densities
3. **High dimensions**: "Curse of dimensionality" - distances become meaningless
4. **Complexity**: $O(n^2)$ naive, $O(n \log n)$ with spatial index

Example of varying density problem:

- Dense cluster: 100 points in radius 1
- Sparse cluster: 10 points in radius 5
- No single ε works well for both!

Solutions:

- HDBSCAN (hierarchical DBSCAN) - handles varying densities

Choosing DBSCAN Parameters

How to choose ε?
K-distance graph method:

1. For each point, compute distance to k-th nearest neighbor (use $k = \text{MinPts}$)
2. Sort these distances
3. Plot sorted k-distances
4. Look for "elbow" - sharp increase indicates noise threshold
5. Set $\varepsilon =$ distance at elbow

Choosing DBSCAN Parameters

How to choose ε?
K-distance graph method:

1. For each point, compute distance to k-th nearest neighbor (use $k = \text{MinPts}$)
2. Sort these distances
3. Plot sorted k-distances
4. Look for "elbow" - sharp increase indicates noise threshold
5. Set $\varepsilon = $ distance at elbow

How to choose MinPts?

- Rule of thumb: $\text{MinPts} \geq d + 1$ where d is dimensionality
- For 2D data: $\text{MinPts} = 3$ or 4
- Larger MinPts: More robust to noise, but may merge clusters

Choosing DBSCAN Parameters

How to choose ε?
K-distance graph method:

1. For each point, compute distance to k-th nearest neighbor (use $k = \text{MinPts}$)
2. Sort these distances
3. Plot sorted k-distances
4. Look for "elbow" - sharp increase indicates noise threshold
5. Set $\varepsilon =$ distance at elbow

How to choose MinPts?

- Rule of thumb: $\text{MinPts} \geq d + 1$ where d is dimensionality
- For 2D data: $\text{MinPts} = 3$ or 4
- Larger MinPts: More robust to noise, but may merge clusters

Comparing Clustering Methods

Clustering Algorithms: Summary Table

Algorithm	**Shape**	K**?**	**Outliers**	**Complexity**
K-Means	Spherical	Yes	Sensitive	$O(nKdt)$
K-Means++	Spherical	Yes	Sensitive	$O(nKd(t+K))$
Hierarchical	Flexible	No	Sensitive	$O(n^2 \log n)$
DBSCAN	Arbitrary	No	Robust	$O(n \log n)$*

*With spatial indexing; $O(n^2)$ naive

Clustering Algorithms: Summary Table

Algorithm	**Shape**	K**?**	**Outliers**	**Complexity**
K-Means	Spherical	Yes	Sensitive	$O(nKdt)$
K-Means++	Spherical	Yes	Sensitive	$O(nKd(t+K))$
Hierarchical	Flexible	No	Sensitive	$O(n^2 \log n)$
DBSCAN	Arbitrary	No	Robust	$O(n \log n)$*

*With spatial indexing; $O(n^2)$ naive

Additional Properties:

Algorithm	**Deterministic**	**Scalability**	**High-D OK?**
K-Means	No	Excellent	Yes
K-Means++	No	Excellent	Yes
Hierarchical	Yes	Poor	Moderate
DBSCAN	Yes	Good	No

Decision Tree: Which Algorithm to Use?

Choosing a Clustering Algorithm

1. **Do you know K?**
   - Yes $\rightarrow$ Consider K-Means++
   - No $\rightarrow$ Consider Hierarchical or DBSCAN

Decision Tree: Which Algorithm to Use?

Choosing a Clustering Algorithm

1. **Do you know K?**
   - Yes $\rightarrow$ Consider K-Means++
   - No $\rightarrow$ Consider Hierarchical or DBSCAN
2. **Are clusters roughly spherical?**
   - Yes $\rightarrow$ K-Means++ is great
   - No $\rightarrow$ DBSCAN or Hierarchical

Decision Tree: Which Algorithm to Use?

Choosing a Clustering Algorithm

1. **Do you know K?**
   - Yes $\rightarrow$ Consider K-Means++
   - No $\rightarrow$ Consider Hierarchical or DBSCAN
2. **Are clusters roughly spherical?**
   - Yes $\rightarrow$ K-Means++ is great
   - No $\rightarrow$ DBSCAN or Hierarchical
3. **Are there outliers?**
   - Yes $\rightarrow$ DBSCAN (marks as noise)
   - No $\rightarrow$ Any method works

Decision Tree: Which Algorithm to Use?

Choosing a Clustering Algorithm

1. **Do you know K?**
   - Yes $\rightarrow$ Consider K-Means++
   - No $\rightarrow$ Consider Hierarchical or DBSCAN
2. **Are clusters roughly spherical?**
   - Yes $\rightarrow$ K-Means++ is great
   - No $\rightarrow$ DBSCAN or Hierarchical
3. **Are there outliers?**
   - Yes $\rightarrow$ DBSCAN (marks as noise)
   - No $\rightarrow$ Any method works
4. **How large is your dataset?**
   - Large ($n > 10000$) $\rightarrow$ K-Means++ (fastest)
   - Medium ($1000 < n < 10000$) $\rightarrow$ DBSCAN or K-Means++

Decision Tree: Which Algorithm to Use?

Choosing a Clustering Algorithm

1. **Do you know K?**
   - Yes $\rightarrow$ Consider K-Means++
   - No $\rightarrow$ Consider Hierarchical or DBSCAN
2. **Are clusters roughly spherical?**
   - Yes $\rightarrow$ K-Means++ is great
   - No $\rightarrow$ DBSCAN or Hierarchical
3. **Are there outliers?**
   - Yes $\rightarrow$ DBSCAN (marks as noise)
   - No $\rightarrow$ Any method works
4. **How large is your dataset?**
   - Large ($n > 10000$) $\rightarrow$ K-Means++ (fastest)
   - Medium ($1000 < n < 10000$) $\rightarrow$ DBSCAN or K-Means++

Practical Recommendations

Default Choice: K-Means++

- Fast, simple, works well in practice
- Use elbow method to find K
- Run multiple times (non-deterministic)

Practical Recommendations

Default Choice: K-Means++

- Fast, simple, works well in practice
- Use elbow method to find K
- Run multiple times (non-deterministic)

When to use alternatives:

- **Arbitrary shapes + outliers** $\rightarrow$ DBSCAN
- **Need full hierarchy** $\rightarrow$ Hierarchical
- **Probabilistic model** $\rightarrow$ Gaussian Mixture Models
- **Varying densities** $\rightarrow$ HDBSCAN
- **Very large scale** $\rightarrow$ Mini-Batch K-Means

Practical Recommendations

Default Choice: K-Means++

- Fast, simple, works well in practice
- Use elbow method to find K
- Run multiple times (non-deterministic)

When to use alternatives:

- **Arbitrary shapes + outliers** $\rightarrow$ DBSCAN
- **Need full hierarchy** $\rightarrow$ Hierarchical
- **Probabilistic model** $\rightarrow$ Gaussian Mixture Models
- **Varying densities** $\rightarrow$ HDBSCAN
- **Very large scale** $\rightarrow$ Mini-Batch K-Means

General Advice:

Evaluating Clustering Quality

Problem: Unlike supervised learning, we don't have "true" labels!

Evaluating Clustering Quality

Problem: Unlike supervised learning, we don't have "true" labels!
Internal Metrics (no ground truth needed):

- **Silhouette Score**: Measures compactness vs separation
$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$
Range: $[-1, 1]$, higher is better
- **Davies-Bouldin Index**: Ratio of within-cluster to between-cluster distances Lower is better
- **Calinski-Harabasz Index**: Ratio of between-cluster to within-cluster variance Higher is better

Evaluating Clustering Quality

Problem: Unlike supervised learning, we don't have "true" labels!

Internal Metrics (no ground truth needed):

- **Silhouette Score**: Measures compactness vs separation

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

  Range: $[-1, 1]$, higher is better

- **Davies-Bouldin Index**: Ratio of within-cluster to between-cluster distances Lower is better
- **Calinski-Harabasz Index**: Ratio of between-cluster to within-cluster variance Higher is better

External Metrics (if ground truth available):

- **Adjusted Rand Index (ARI)**: Similarity to true labeling

Practical Considerations

Feature Scaling for Clustering

Problem: Features with different scales dominate distance calculations!

Feature Scaling for Clustering

Problem: Features with different scales dominate distance calculations!
Example:

- Age: 20-80 years
- Income: $20,000 - $200,000
- Distance dominated by income!

Feature Scaling for Clustering

Problem: Features with different scales dominate distance calculations!
Example:

- Age: 20-80 years
- Income: \$20,000 - \$200,000
- Distance dominated by income!

Solution: Always normalize features before clustering!
StandardScaler (Z-score normalization):

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

Transforms to mean $= 0$, std $= 1$

Feature Scaling for Clustering

Problem: Features with different scales dominate distance calculations!
Example:

- Age: 20-80 years
- Income: \$20,000 - \$200,000
- Distance dominated by income!

Solution: Always normalize features before clustering!
StandardScaler (Z-score normalization):

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

Transforms to mean $= 0$, std $= 1$
MinMaxScaler:

$$x_{\text{scaled}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

Transforms to range $[0, 1]$

Feature Scaling for Clustering

Problem: Features with different scales dominate distance calculations!
Example:

- Age: 20-80 years
- Income: \$20,000 - \$200,000
- Distance dominated by income!

Solution: Always normalize features before clustering!
StandardScaler (Z-score normalization):

$$x_{\text{scaled}} = \frac{x - \mu}{\sigma}$$

Transforms to mean $= 0$, std $= 1$
MinMaxScaler:

$$x_{\text{scaled}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

Transforms to range $[0, 1]$

High-Dimensional Data

Curse of Dimensionality:

- In high dimensions, all distances become similar!
- Nearest and farthest neighbors have similar distances
- Clustering becomes meaningless

High-Dimensional Data

Curse of Dimensionality:

- In high dimensions, all distances become similar!
- Nearest and farthest neighbors have similar distances
- Clustering becomes meaningless

Solutions:

1. **Dimensionality Reduction first**:
   - PCA: Project to lower dimensions
   - t-SNE: Nonlinear reduction (for visualization)
   - Autoencoders: Neural network-based
2. **Feature Selection**:
   - Remove irrelevant features
   - Use domain knowledge

Handling Categorical Data

Problem: K-Means uses Euclidean distance (assumes numerical features)

Handling Categorical Data

Problem: K-Means uses Euclidean distance (assumes numerical features)
Solutions:

1. **One-Hot Encoding**:
   - Color $= \{$Red, Blue, Green$\} \rightarrow$ 3 binary features
   - Then cluster normally
   - Can use Hamming distance instead of Euclidean

Handling Categorical Data

Problem: K-Means uses Euclidean distance (assumes numerical features)
Solutions:

1. **One-Hot Encoding**:
   - Color $= \{$Red, Blue, Green$\} \rightarrow$ 3 binary features
   - Then cluster normally
   - Can use Hamming distance instead of Euclidean
2. **K-Modes**:
   - Variant of K-Means for categorical data
   - Uses mode instead of mean
   - Uses Hamming distance

Handling Categorical Data

Problem: K-Means uses Euclidean distance (assumes numerical features)
Solutions:

1. **One-Hot Encoding**:
   - Color $= \{$Red, Blue, Green$\} \rightarrow$ 3 binary features
   - Then cluster normally
   - Can use Hamming distance instead of Euclidean
2. **K-Modes**:
   - Variant of K-Means for categorical data
   - Uses mode instead of mean
   - Uses Hamming distance
3. **K-Prototypes**:
   - Handles mixed numerical $+$ categorical

Handling Categorical Data

Problem: K-Means uses Euclidean distance (assumes numerical features)
Solutions:

1. **One-Hot Encoding**:
   - Color = {Red, Blue, Green} $\rightarrow$ 3 binary features
   - Then cluster normally
   - Can use Hamming distance instead of Euclidean
2. **K-Modes**:
   - Variant of K-Means for categorical data
   - Uses mode instead of mean
   - Uses Hamming distance
3. **K-Prototypes**:
   - Handles mixed numerical + categorical

Summary

What We Learned Today

1. **Fundamentals**:

- Unsupervised learning finds patterns without labels
- Clustering groups similar objects together
- Need to define similarity/distance measure

What We Learned Today

1. **Fundamentals**:

- Unsupervised learning finds patterns without labels
- Clustering groups similar objects together
- Need to define similarity/distance measure

2. **K-Means**:

- Minimizes within-cluster sum of squares
- Two-step: Assignment $\rightarrow$ Update $\rightarrow$ Repeat
- K-Means++ initialization helps a lot!
- Use elbow method to choose K

What We Learned Today

1. **Fundamentals**:

- Unsupervised learning finds patterns without labels
- Clustering groups similar objects together
- Need to define similarity/distance measure

2. **K-Means**:

- Minimizes within-cluster sum of squares
- Two-step: Assignment $\rightarrow$ Update $\rightarrow$ Repeat
- K-Means++ initialization helps a lot!
- Use elbow method to choose K

3. **Hierarchical**:

- Builds tree of clusters (dendrogram)

What We Learned Today (cont.)

4. DBSCAN:

- Density-based: finds arbitrarily shaped clusters
- Identifies outliers as noise
- Parameters: ε (radius), MinPts (threshold)
- Use k-distance graph to choose ε

What We Learned Today (cont.)

4. DBSCAN:

- Density-based: finds arbitrarily shaped clusters
- Identifies outliers as noise
- Parameters: ε (radius), MinPts (threshold)
- Use k-distance graph to choose ε

5. Practical Tips:

- Always scale/normalize features first!
- Visualize results (PCA for high-D)
- Try K-Means++ as baseline
- Choose algorithm based on data properties
- Use multiple metrics to evaluate

What We Learned Today (cont.)

4. DBSCAN:

- Density-based: finds arbitrarily shaped clusters
- Identifies outliers as noise
- Parameters: ε (radius), MinPts (threshold)
- Use k-distance graph to choose ε

5. Practical Tips:

- Always scale/normalize features first!
- Visualize results (PCA for high-D)
- Try K-Means++ as baseline
- Choose algorithm based on data properties
- Use multiple metrics to evaluate

Next Steps

Practice:

- Implement K-Means from scratch (HW/Lab)
- Try scikit-learn: `KMeans`, `DBSCAN`, `AgglomerativeClustering`
- Experiment with real datasets (Iris, customer data, images)

Next Steps

Practice:

- Implement K-Means from scratch (HW/Lab)
- Try scikit-learn: `KMeans`, `DBSCAN`, `AgglomerativeClustering`
- Experiment with real datasets (Iris, customer data, images)

Further Topics:

- Gaussian Mixture Models (probabilistic clustering)
- Spectral clustering (graph-based)
- Topic modeling (Latent Dirichlet Allocation)
- Deep clustering (autoencoders + K-Means)

Next Steps

Practice:

- Implement K-Means from scratch (HW/Lab)
- Try scikit-learn: `KMeans`, `DBSCAN`, `AgglomerativeClustering`
- Experiment with real datasets (Iris, customer data, images)

Further Topics:

- Gaussian Mixture Models (probabilistic clustering)
- Spectral clustering (graph-based)
- Topic modeling (Latent Dirichlet Allocation)
- Deep clustering (autoencoders + K-Means)

Code and Resources:

Google Colab Notebook

Questions?