Unsupervised Learning (Detailed)

Nipun Batra

IIT Gandhinagar

October 27, 2025

The need for Unsupervised Learning

- Aids the search of patterns in data.
- Find features for categorization.
- Easier to collect unlabeled data.

The need for Unsupervised Learning

- Aids the search of patterns in data.
- Find features for categorization.
- Easier to collect unlabeled data.

Places where you will see unsupervised learning

- It can be used to segment the market based on customer preferences.
- A data science team reduces the number of dimensions in a large dataset to simplify modeling and reduce file size.
- Anomaly detection in network traffic or fraud detection
- Document clustering and topic modeling
- Image segmentation in computer vision
- Gene expression analysis in bioinformatics

Real-World Examples of Unsupervised Learning

- **E-commerce**: Grouping customers by purchasing behavior for personalized recommendations
- **Social Networks**: Detecting communities and user groups
- **Healthcare**: Identifying disease subtypes from patient data
- **Astronomy**: Classifying galaxies and celestial objects
- **Climate Science**: Identifying weather patterns and climate zones
- **Manufacturing**: Quality control by detecting anomalous products

Clustering

Clustering

AIM: To find groups/subgroups in a dataset.

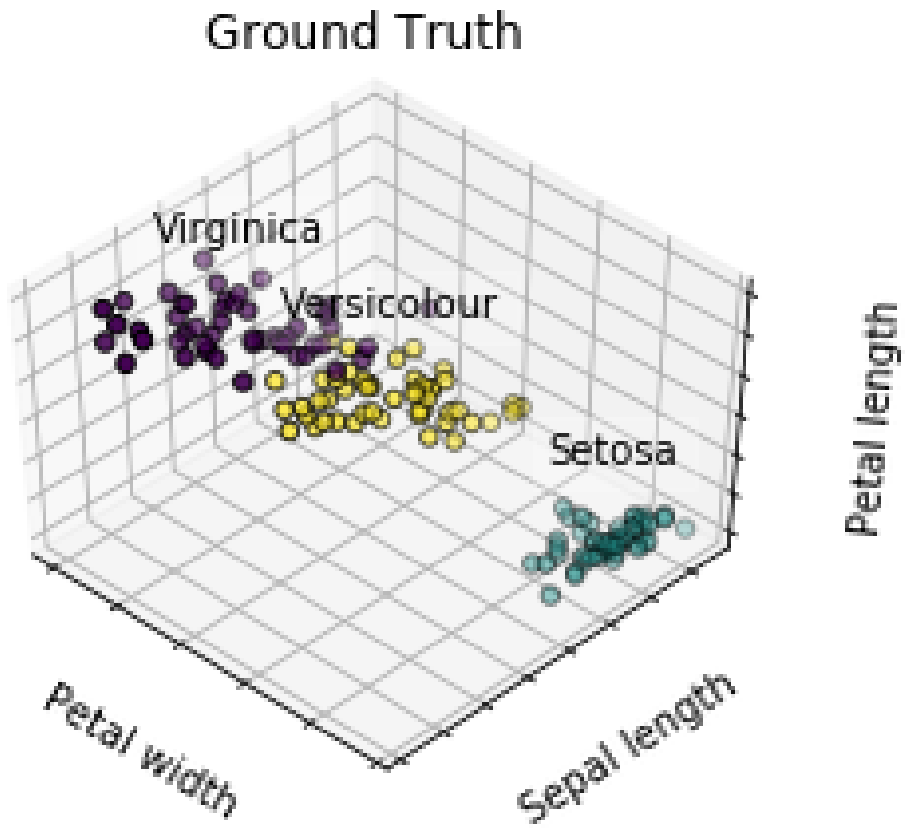
Clustering

AIM: To find groups/subgroups in a dataset.
REQUIREMENTS: A predefined notion of similarity/dissimilarity.

Clustering

AIM: To find groups/subgroups in a dataset.
REQUIREMENTS: A predefined notion of similarity/dissimilarity. **Examples:**
Market Segmentation: Customers with similar preferences in the same groups. This would aid in targeted marketing.

Clustering



Iris Data Set with ground truth

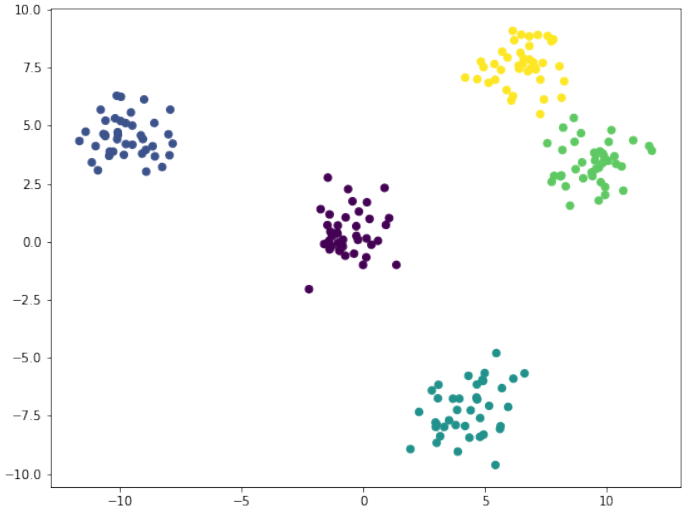
Types of Clustering Algorithms

- **Partitioning Methods**: K-Means, K-Medoids
- **Hierarchical Methods**: Agglomerative, Divisive
- **Density-Based Methods**: DBSCAN, OPTICS
- **Model-Based Methods**: Gaussian Mixture Models
- **Grid-Based Methods**: STING, CLIQUE

K-Means Clustering

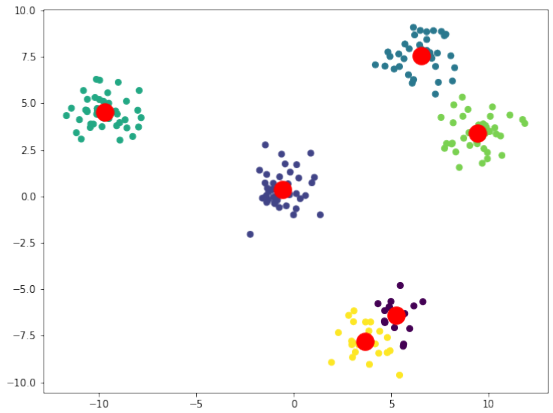
K-Means Clustering

- N points in a R^d space.
- C_i: set of points in the i^{th} cluster.
- $C_1 \cup C_2 \cup \ldots C_k = \{1, \ldots, n\}$
- $C_i \cap C_j = \{\phi\}$ for $i \neq j$

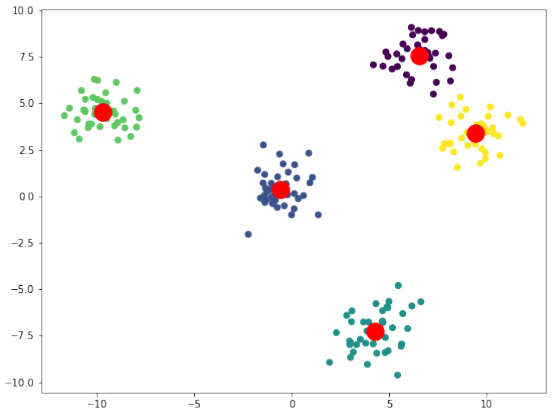


Dataset with 5 clusters

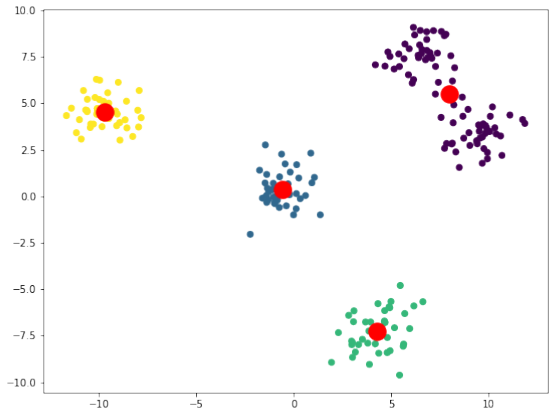
K-Means Clustering



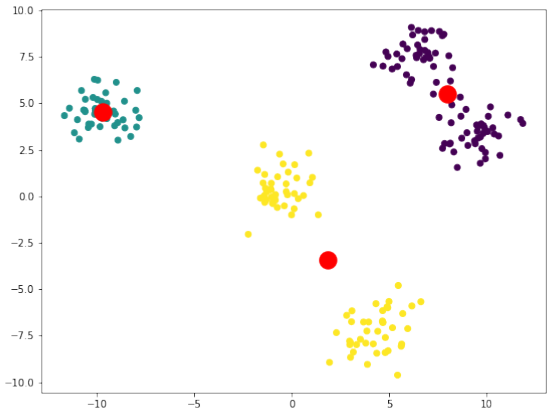
K=6



K=5



K=4



K=3

K-Means Intuition

- Good Clustering: Within the cluster the variation (WCV) is small.

K-Means Intuition

- Good Clustering: Within the cluster the variation (WCV) is small.
- Objective:

$$\min_{C_1,\ldots,C_k} \left(\sum_{i=1}^{k} WCV\left(C_i\right) \right)$$

K-Means Intuition

- Good Clustering: Within the cluster the variation (WCV) is small.
- Objective:

$$\min_{C_1,\ldots,C_k} \left(\sum_{i=1}^{k} WCV\left(C_i\right) \right)$$

  Minimize the WCV as much as possible

K-Means Intuition

Objective:

$$\min_{C_1,\ldots,C_k} \left(\sum_{i=1}^{k} WCV\left(C_i\right) \right)$$

K-Means Intuition

Objective:

$$\min_{C_1,\ldots,C_k} \left(\sum_{i=1}^{k} WCV\left(C_i\right) \right)$$

$$WCV\left(C_i\right) = \frac{1}{|C_i|} \texttt{ (Distance between all points)}$$

$$WCV\left(C_i\right) = \frac{1}{|C_i|} \sum_{a \in C_i} \sum_{b \in C_i} ||x_a - x_b||_2^2$$

where $|C_i|$ is the number of points in C_i

K-Means Algorithm

1. Randomly assign a cluster number i to every point (where $i \in \{1, \ldots k\}$)

K-Means Algorithm

1. Randomly assign a cluster number i to every point (where $i \in \{1, \ldots k\}$)

    1) For each cluster C_i compute the centroid (mean of all points in C_i over d dimensions)

Convergence: Algorithm stops when assignments no longer change (or changes are minimal)

K-Means Algorithm

1. Randomly assign a cluster number i to every point (where $i \in \{1, \ldots k\}$)

    1) For each cluster C_i compute the centroid (mean of all points in C_i over d dimensions)
    2) Assign each observation to the cluster whose centroid is closest.

Convergence: Algorithm stops when assignments no longer change (or changes are minimal)

K-Means Algorithm

1. Randomly assign a cluster number i to every point (where $i \in \{1, \ldots k\}$)
2. Iterate until convergence:
   1) For each cluster C_i compute the centroid (mean of all points in C_i over d dimensions)
   2) Assign each observation to the cluster whose centroid is closest.

Convergence: Algorithm stops when assignments no longer change (or changes are minimal)

Working of K-Means Algorithm

K-Means Convergence: Intuitive Proof (1/3)

Key Idea: The objective function (total within-cluster variance) decreases at each step and is bounded below.

K-Means Convergence: Intuitive Proof (1/3)

Key Idea: The objective function (total within-cluster variance) decreases at each step and is bounded below.
Objective Function:

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

where μ_i is the centroid of cluster C_i

K-Means Convergence: Intuitive Proof (2/3)

Step 1: Update Centroids

- Fix cluster assignments, optimize centroids
- For each cluster C_i, the centroid that minimizes $\sum_{x \in C_i} ||x - \mu_i||^2$ is $\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$
- This is the mean of the points in the cluster
- This step cannot increase J (optimal choice given fixed assignments)

K-Means Convergence: Intuitive Proof (3/3)

Step 2: Update Assignments

- Fix centroids, reassign each point to nearest centroid
- For each point x, assign to $\arg\min_i ||x - \mu_i||^2$
- This step cannot increase J (each point moves to minimize its contribution)

K-Means Convergence: Intuitive Proof (3/3)

Step 2: Update Assignments

- Fix centroids, reassign each point to nearest centroid
- For each point x, assign to $\arg\min_i ||x - \mu_i||^2$
- This step cannot increase J (each point moves to minimize its contribution)

Conclusion:

- J decreases (or stays same) at each iteration
- $J \geq 0$ (bounded below)
- Finite number of possible assignments
- Therefore, algorithm must converge!

K-Means Convergence: Intuitive Proof (3/3)

Step 2: Update Assignments

- Fix centroids, reassign each point to nearest centroid
- For each point x, assign to $\arg\min_i ||x - \mu_i||^2$
- This step cannot increase J (each point moves to minimize its contribution)

Conclusion:

- J decreases (or stays same) at each iteration
- $J \geq 0$ (bounded below)
- Finite number of possible assignments
- Therefore, algorithm must converge!

Note: Converges to a **local minimum**, not necessarily global

Why does K-Means work?

$$\text{Let, } \mu_i \in R^d = \text{Centroid for } i^{th} \text{ cluster}$$

$$= \frac{1}{|C_i|} \sum_{a \in C_i} x_a$$

Why does K-Means work?

$$\text{Let, } \mu_i \in R^d = \text{Centroid for } i^{th} \text{ cluster}$$
$$= \frac{1}{|C_i|} \sum_{a \in C_i} x_a$$

Then,

$$WCV\left(C_i\right) = \frac{1}{|C_i|} \sum_{a \in C_i} \sum_{b \in C_i} ||x_a - x_b||_2^2$$
$$= 2 \sum_{a \in C_i} ||x_a - \mu_i||_2^2$$

Why does K-Means work?

$$\text{Let, } \mu_i \in R^d = \text{Centroid for } i^{th} \text{ cluster}$$
$$= \frac{1}{|C_i|} \sum_{a \in C_i} x_a$$

Then,

$$WCV\left(C_i\right) = \frac{1}{|C_i|} \sum_{a \in C_i} \sum_{b \in C_i} ||x_a - x_b||_2^2$$
$$= 2 \sum_{a \in C_i} ||x_a - \mu_i||_2^2$$

This shows that K-Means gives the **local minima**.

K-Means: Time and Space Complexity

Time Complexity:

- Per iteration: $O(nkd)$ where
  - n = number of points
  - k = number of clusters
  - d = number of dimensions
- Total: $O(nkdt)$ where t = number of iterations

K-Means: Time and Space Complexity

Time Complexity:

- Per iteration: $O(nkd)$ where
  - n = number of points
  - k = number of clusters
  - d = number of dimensions
- Total: $O(nkdt)$ where t = number of iterations

Space Complexity: $O(n + k)d$

- Store data points and centroids

K-Means: Time and Space Complexity

Time Complexity:

- Per iteration: $O(nkd)$ where
  - n = number of points
  - k = number of clusters
  - d = number of dimensions
- Total: $O(nkdt)$ where t = number of iterations

Space Complexity: $O(n + k)d$

- Store data points and centroids

Practical Note: Usually converges quickly, but worst-case can be exponential

K-Means Failure Modes

K-Means Failure Mode 1: Non-Spherical Clusters

- K-Means assumes spherical (isotropic) clusters
- Fails with elongated, elliptical, or irregular shapes
- Uses Euclidean distance, which favors spherical boundaries

K-Means Failure Mode 1: Non-Spherical Clusters

- K-Means assumes spherical (isotropic) clusters
- Fails with elongated, elliptical, or irregular shapes
- Uses Euclidean distance, which favors spherical boundaries

Example: Two elongated, parallel clusters will be incorrectly split

K-Means Failure Mode 2: Different Cluster Sizes

- K-Means tends to create equal-sized clusters
- Struggles when true clusters have vastly different sizes
- Large clusters may be split, small clusters may be merged

K-Means Failure Mode 3: Different Densities

- Assumes similar density across all clusters
- Fails when clusters have different densities
- Dense clusters may be over-segmented
- Sparse clusters may absorb nearby points incorrectly

K-Means Failure Mode 4: Non-Convex Shapes

- K-Means creates convex decision boundaries
- Cannot handle non-convex shapes (e.g., crescents, rings)
- Example: Two interleaving half-moon shapes

K-Means Failure Mode 4: Non-Convex Shapes

- K-Means creates convex decision boundaries
- Cannot handle non-convex shapes (e.g., crescents, rings)
- Example: Two interleaving half-moon shapes

Solution: Use kernel K-Means or other methods (DBSCAN, spectral clustering)

K-Means Failure Mode 5: Sensitive to Initialization

- Random initialization can lead to poor local minima
- Different runs can give very different results
- Outliers can "capture" centroids

K-Means Failure Mode 5: Sensitive to Initialization

- Random initialization can lead to poor local minima
- Different runs can give very different results
- Outliers can "capture" centroids

Example: If initial centroid placed in low-density region, may create poor clustering

K-Means Failure Mode 5: Sensitive to Initialization

- Random initialization can lead to poor local minima
- Different runs can give very different results
- Outliers can "capture" centroids

Example: If initial centroid placed in low-density region, may create poor clustering
Solution: K-Means++ initialization (discussed next!)

K-Means Failure Mode 6: Outliers

- Sensitive to outliers (uses squared distances)
- Single outlier can significantly shift centroid
- Can create spurious clusters for noise points

K-Means Failure Mode 6: Outliers

- Sensitive to outliers (uses squared distances)
- Single outlier can significantly shift centroid
- Can create spurious clusters for noise points

Solutions:

- Pre-process to remove outliers
- Use K-Medoids (more robust)
- Use trimmed K-Means

K-Means Failure Mode 7: Choosing K

- Requires pre-specifying number of clusters k
- Wrong k leads to over/under-segmentation
- No automatic way to determine optimal k

K-Means Failure Mode 7: Choosing K

- Requires pre-specifying number of clusters k
- Wrong k leads to over/under-segmentation
- No automatic way to determine optimal k

Methods to choose K:

- Elbow method (plot inertia vs k)
- Silhouette analysis
- Gap statistic
- Domain knowledge

K-Means++

K-Means++: Smarter Initialization

Problem: Random initialization can lead to poor results

K-Means++: Smarter Initialization

Problem: Random initialization can lead to poor results
K-Means++ Solution: Choose initial centroids that are far apart from each other

K-Means++: Smarter Initialization

Problem: Random initialization can lead to poor results
K-Means++ Solution: Choose initial centroids that are far apart from each other
Key Idea:

- First centroid: chosen uniformly at random
- Subsequent centroids: chosen with probability proportional to squared distance from nearest existing centroid
- Spreads out initial centroids

K-Means++ Algorithm

1. Choose first centroid μ_1 uniformly at random from data points

K-Means++ Algorithm

1. Choose first centroid μ_1 uniformly at random from data points
2. For $i = 2, 3, \ldots, k$:
   1) For each point x, compute $D(x) =$ distance to nearest centroid already chosen
   2) Choose next centroid μ_i with probability $\propto D(x)^2$

K-Means++ Algorithm

1. Choose first centroid μ_1 uniformly at random from data points
2. For $i = 2, 3, \ldots, k$:
   1) For each point x, compute $D(x) =$ distance to nearest centroid already chosen
   2) Choose next centroid μ_i with probability $\propto D(x)^2$
3. Proceed with standard K-Means algorithm

K-Means++: Why Does It Work?

Intuition:

- Points far from existing centroids are more likely to be chosen
- Ensures good coverage of the data space
- Reduces chance of multiple centroids in same cluster

K-Means++: Why Does It Work?

Intuition:

- Points far from existing centroids are more likely to be chosen
- Ensures good coverage of the data space
- Reduces chance of multiple centroids in same cluster

Theoretical Guarantee:

- K-Means++ initialization is $O(\log k)$-competitive with optimal clustering
- Expected objective value is at most $O(\log k)$ times optimal
- Much better than random initialization (no guarantees)

K-Means++ Example (1/3)

Step 1: Choose first centroid randomly

- Say we pick point A
- $\mu_1 = A$

K-Means++ Example (2/3)

Step 2: Choose second centroid

- Compute $D(x)^2$ for all points (distance to A)
- Point B very far from A has high $D(B)^2$
- Point C close to A has low $D(C)^2$
- Probability of choosing B as μ_2 is much higher than C

K-Means++ Example (3/3)

Step 3: Choose third centroid (if $k = 3$)

- Now compute $D(x)^2 =$ min distance to μ_1 or μ_2
- Points far from both A and B have higher probability
- Likely to choose point from third cluster region

K-Means++ Example (3/3)

Step 3: Choose third centroid (if $k = 3$)

- Now compute $D(x)^2 =$ min distance to μ_1 or μ_2
- Points far from both A and B have higher probability
- Likely to choose point from third cluster region

Result: Initial centroids spread across data, one per true cluster (ideally)

K-Means++ vs Standard K-Means

Advantages of K-Means++:

- Better convergence to global optimum
- Fewer iterations typically needed
- More consistent results across runs
- Theoretical guarantees

K-Means++ vs Standard K-Means

Advantages of K-Means++:

- Better convergence to global optimum
- Fewer iterations typically needed
- More consistent results across runs
- Theoretical guarantees

Disadvantages:

- Slightly more complex initialization
- Sequential process (harder to parallelize initialization)
- Still inherits other K-Means limitations

K-Means++ vs Standard K-Means

Advantages of K-Means++:

- Better convergence to global optimum
- Fewer iterations typically needed
- More consistent results across runs
- Theoretical guarantees

Disadvantages:

- Slightly more complex initialization
- Sequential process (harder to parallelize initialization)
- Still inherits other K-Means limitations

Practical Note: K-Means++ is now the default in most libraries (scikit-learn, etc.)

Hierarchical Clustering

Hierarchical Clustering

Gives a clustering of all the clusters

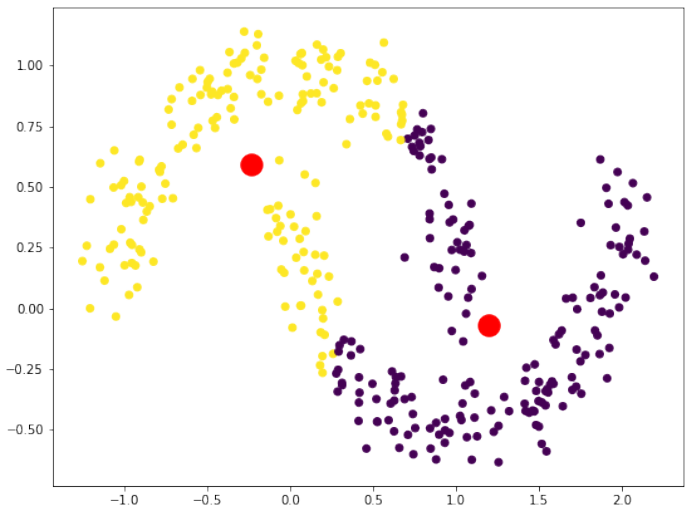
Hierarchical Clustering

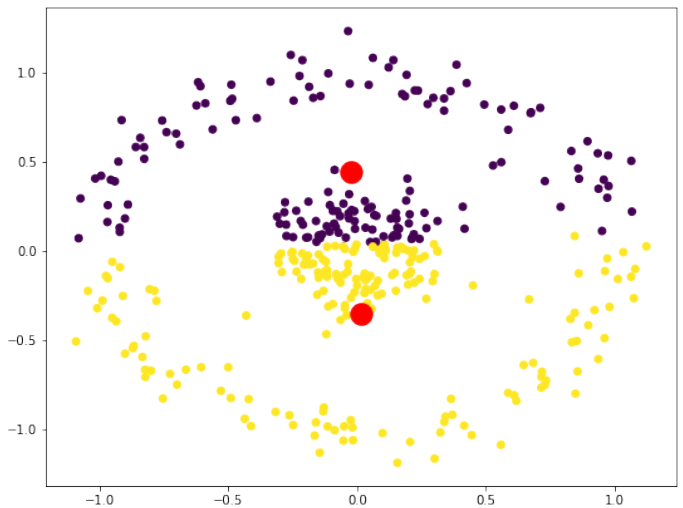
Gives a clustering of all the clusters
There is no need to specify K at the start

Hierarchical Clustering

Gives a clustering of all the clusters

There is no need to specify K at the start



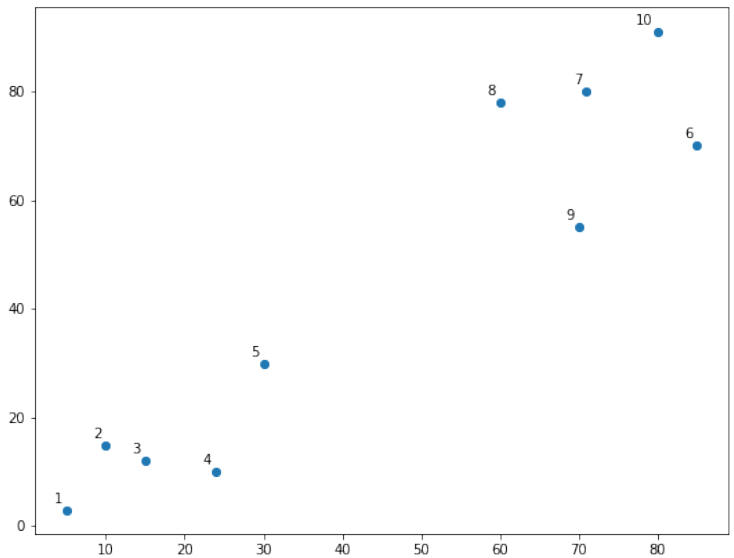


Examples where K-Means fails

Algorithm for Hierarchical Clustering

Agglomerative (Bottom-Up):

1. Start with each point in its own cluster (n clusters)



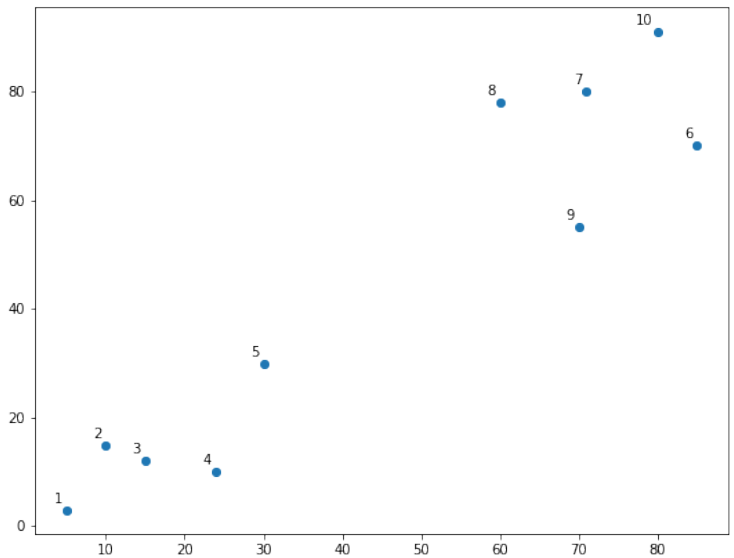
Example Dataset

Algorithm for Hierarchical Clustering

Agglomerative (Bottom-Up):

1. Start with each point in its own cluster (n clusters)

    1) Identify the 2 closest clusters



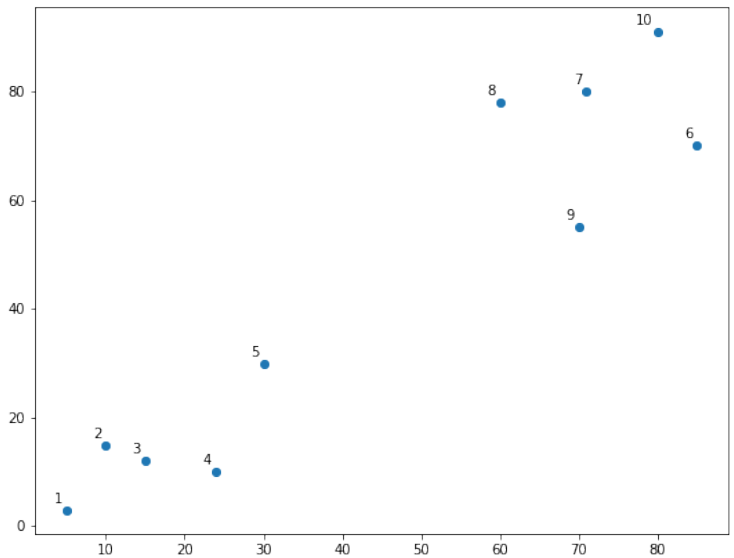
Example Dataset

Algorithm for Hierarchical Clustering

Agglomerative (Bottom-Up):

1. Start with each point in its own cluster (n clusters)

    1) Identify the 2 closest clusters
    2) Merge them

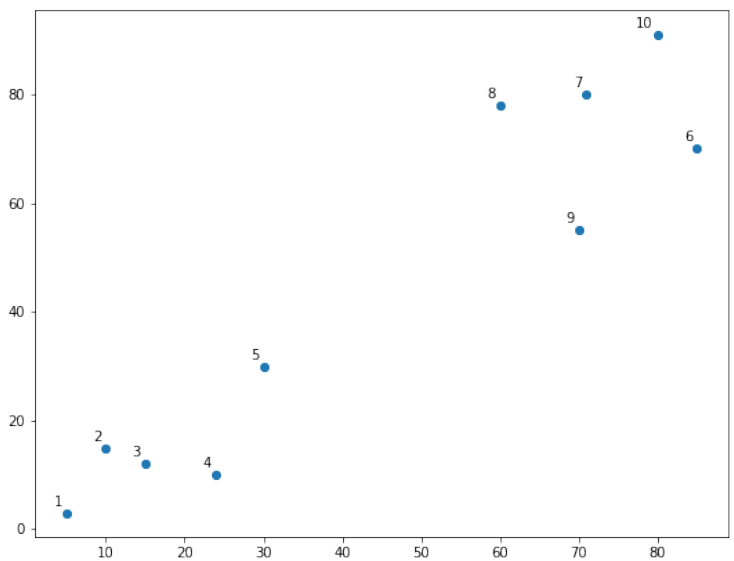


Example Dataset

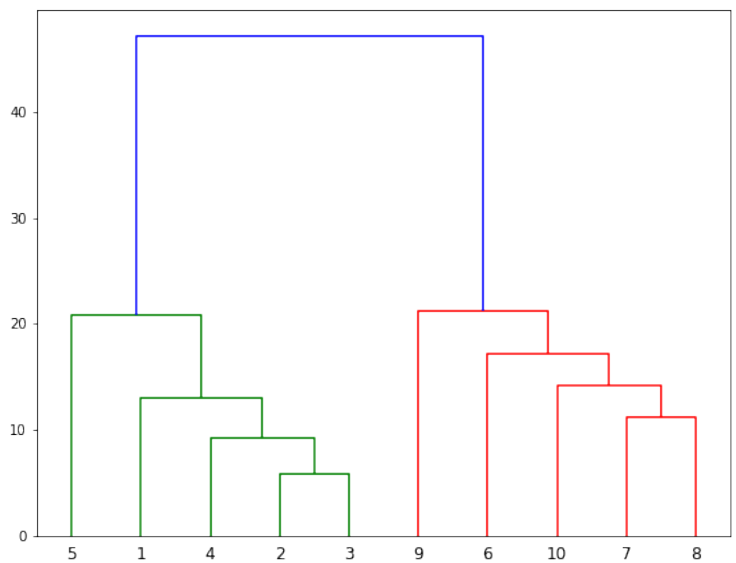
Algorithm for Hierarchical Clustering

Agglomerative (Bottom-Up):

1. Start with each point in its own cluster (n clusters)
2. Repeat until all points are in a single cluster:
   1) Identify the 2 closest clusters
   2) Merge them



Example Dataset



Final Clustering

Joining Clusters/Linkages

Complete
Max inter-cluster similarity

Single
Min inter-cluster similarity

Centroid
Dissimilarity between cluster centroids

Joining Clusters/Linkages

Complete
Max inter-cluster similarity

Single
Min inter-cluster similarity

Centroid
Dissimilarity between cluster centroids

Average Linkage: Average distance between all pairs of points

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

Joining Clusters/Linkages

Complete
Max inter-cluster similarity

Single
Min inter-cluster similarity

Centroid
Dissimilarity between cluster centroids

Average Linkage: Average distance between all pairs of points

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

Ward's Method: Minimizes within-cluster variance when merging

Density-Based Clustering

Density-Based Clustering: Motivation

Problem with K-Means and Hierarchical:

- Assume clusters are convex and isotropic
- Struggle with arbitrary shapes
- Sensitive to noise and outliers

Density-Based Clustering: Motivation

Problem with K-Means and Hierarchical:

- Assume clusters are convex and isotropic
- Struggle with arbitrary shapes
- Sensitive to noise and outliers

Density-Based Idea:

- Clusters are dense regions separated by sparse regions
- Can find arbitrarily shaped clusters
- Can identify noise/outliers

DBSCAN: Density-Based Spatial Clustering

Key Concepts:

- ϵ-**neighborhood**: Points within distance ϵ of a point p
- **Core point**: Has at least `MinPts` points in its ϵ-neighborhood
- **Border point**: Not a core point, but in ϵ-neighborhood of core point
- **Noise point**: Neither core nor border

DBSCAN Algorithm

Parameters: ϵ (radius), `MinPts` (minimum points)

DBSCAN Algorithm

Parameters: ϵ (radius), `MinPts` (minimum points)
Algorithm:

1. Mark all points as unvisited
2. For each unvisited point p:
   1) Mark p as visited
   2) Find all points in ϵ-neighborhood of p
   3) If $|\text{neighborhood}| \geq$ `MinPts`:
      - Create new cluster
      - Add p and expand cluster (recursively add neighbors)
   4) Else: mark p as noise (may be changed later)

DBSCAN: Advantages

- Can find arbitrarily shaped clusters
- Robust to outliers (marks them as noise)
- No need to specify number of clusters
- Only two parameters: ϵ and `MinPts`
- Deterministic (same result each run)

DBSCAN: Disadvantages

- Sensitive to parameter choice (ϵ, `MinPts`)
- Struggles with varying density clusters
- Not suitable for high-dimensional data (curse of dimensionality)
- Cannot cluster datasets with large differences in densities
- Time complexity: $O(n^2)$ (can be $O(n \log n)$ with spatial index)

DBSCAN Example

Dataset: Points in 2D with two clusters and noise

DBSCAN Example

Dataset: Points in 2D with two clusters and noise
Parameters: $\epsilon = 0.5$, $\texttt{MinPts} = 4$

DBSCAN Example

Dataset: Points in 2D with two clusters and noise
Parameters: $\epsilon = 0.5$, $\texttt{MinPts} = 4$
Result:

- Dense regions become clusters
- Sparse points between clusters marked as noise
- Can handle non-convex shapes (e.g., crescents, rings)

Choosing DBSCAN Parameters

ϵ **(radius)**:

- Plot k-nearest neighbor distance graph
- Look for "elbow" where distance increases sharply
- Points after elbow are likely noise

Choosing DBSCAN Parameters

ϵ **(radius)**:

- Plot k-nearest neighbor distance graph
- Look for "elbow" where distance increases sharply
- Points after elbow are likely noise

`MinPts`:

- Rule of thumb: $\texttt{MinPts} \geq d + 1$ where d is dimensionality
- Larger values: more robust to noise, but may merge clusters
- Smaller values: more sensitive, may create many small clusters

Density-Based Methods: Variants

OPTICS (Ordering Points To Identify Clustering Structure):

- Extension of DBSCAN
- Produces reachability plot showing cluster structure
- Can extract clusters at different densities

Density-Based Methods: Variants

OPTICS (Ordering Points To Identify Clustering Structure):

- Extension of DBSCAN
- Produces reachability plot showing cluster structure
- Can extract clusters at different densities

HDBSCAN (Hierarchical DBSCAN):

- Builds hierarchy of clusters
- Automatically selects stable clusters
- More robust parameter selection

Comparing Clustering Methods

Method	**Shape**	**Outliers**	**K?**	**Speed**
K-Means	Spherical	Sensitive	Yes	Fast
K-Means++	Spherical	Sensitive	Yes	Fast
Hierarchical	Any	Sensitive	No	Slow
DBSCAN	Arbitrary	Robust	No	Medium

Comparing Clustering Methods

Method	Shape	Outliers	K?	Speed
K-Means	Spherical	Sensitive	Yes	Fast
K-Means++	Spherical	Sensitive	Yes	Fast
Hierarchical	Any	Sensitive	No	Slow
DBSCAN	Arbitrary	Robust	No	Medium

Choosing a Method:

- Known k, spherical clusters: K-Means++
- Arbitrary shapes, noise: DBSCAN
- Hierarchy needed: Hierarchical
- Large data: K-Means++ or Mini-Batch K-Means

More Code

Google Colab Link