Memory updated

K-Means Clustering – A Deep Dive

1. What is Clustering?

Clustering is an **unsupervised learning technique** that groups data points into **clusters** based on similarity.

- Unlike supervised learning, **there are no labels** in the dataset.
- The goal is to **discover hidden structures** in the data.

2. What is K-Means?

K-Means is one of the simplest and most widely used clustering algorithms. It works by:

1. Choosing **K cluster centroids (randomly or heuristically)**.
2. Assigning each data point to the **nearest centroid**.
3. Recomputing centroids as the **average of assigned points**.
4. Repeating until convergence (centroids stop changing).

---

3. When to Use & Avoid K-Means

✅ Use K-Means When:

- You **need fast and scalable clustering**.
- The clusters are **well-separated and convex**.
- You have a **moderate** number of clusters.

❌ Avoid K-Means When:

- The dataset has **non-spherical clusters** (e.g., concentric circles).
- There are **many outliers**, as K-Means is sensitive to them.
- The **value of K is unknown**, and choosing it is difficult.

---

4. Choosing K (Number of Clusters)

Methods to Determine the Optimal K

1. **Elbow Method**
    - Compute the **within-cluster sum of squares (WCSS)** for different values of K.
    - Look for an **"elbow point"** where WCSS stops decreasing significantly.
2. **Silhouette Score**
    - Measures how well-separated the clusters are.
    - Higher scores indicate better clustering.
3. **Gap Statistic**
    - Compares clustering performance against randomly generated datasets.

Mathematical Breakdown of Methods for Choosing K in K-Means Clustering

Selecting the optimal number of clusters K is crucial for **interpretable and efficient clustering**. Here, we cover the most common techniques:

1. The Elbow Method

Intuition

- We compute the **within-cluster sum of squares (WCSS)** for different values of K.
- Plot WCSS vs. K and look for an **"elbow" point**, where WCSS stops decreasing significantly.

Mathematics

The **WCSS (Within-Cluster Sum of Squares)** measures how compact each cluster is:

$$\text{WCSS} = \sum_{i=1}^{K} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

where:

- x_j = data point
- μ_i = centroid of cluster C_i
- $||x_j - \mu_i||^2$ = squared Euclidean distance from x_j to its centroid

Procedure

1. Compute WCSS for different values of K.
2. Plot K vs. WCSS.
3. Identify the "elbow point" where WCSS reduction slows.

Limitations

- The elbow point is **subjective**.
- Doesn't always work well if clusters are not clearly defined.

1. The Elbow Method

Intuition

- We compute the **within-cluster sum of squares (WCSS)** for different values of K.
- Plot WCSS vs. K and look for an **"elbow" point**, where WCSS stops decreasing significantly.

Mathematics

The **WCSS (Within-Cluster Sum of Squares)** measures how compact each cluster is:

$$\text{WCSS} = \sum_{i=1}^{K} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

where:

- x_j = data point
- μ_i = centroid of cluster C_i
- $||x_j - \mu_i||^2$ = squared Euclidean distance from x_j to its centroid

Procedure

1. Compute WCSS for different values of K.
2. Plot K vs. WCSS.
3. Identify the "elbow point" where WCSS reduction slows.

Limitations

- The elbow point is **subjective**.
- Doesn’t always work well if clusters are not clearly defined.

2. Silhouette Score

Intuition

- Measures how well-separated clusters are.
- The score is between **-1 and 1**:
  - **1** → Perfect clustering.
  - **0** → Overlapping clusters.
  - **-1** → Wrong clustering.

Mathematics

For each data point x_i, compute:

1. **Average intra-cluster distance** (how close is x_i to its own cluster centroid?):

$$a(i) = \frac{1}{|C_k| - 1} \sum_{x_j \in C_k, i \neq j} ||x_i - x_j||$$

   where C_k is the cluster to which x_i belongs.

2. **Average inter-cluster distance** (how far is x_i from the nearest different cluster?):

$$b(i) = \min_{C_m \neq C_k} \frac{1}{|C_m|} \sum_{x_j \in C_m} ||x_i - x_j||$$

   where C_m is the nearest cluster.

3. Compute **Silhouette Score** for x_i:

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

4. Compute the **average silhouette score** for all points to evaluate clustering quality.

Procedure

1. Compute silhouette scores for different K.
2. Choose K with the **highest average silhouette score**.

Limitations

- Requires pairwise distance computations → **Computationally expensive** for large datasets.
- Can be misleading if clusters have non-convex shapes.

3. Gap Statistic

Intuition

- Compares clustering performance against **randomly generated datasets**.
- A higher gap means clustering is **better than random noise**.

Mathematics

1. Compute the WCSS for **real data**:

$$W_k = \sum_{i=1}^{K} \sum_{x_j \in C_i} ||x_j - \mu_i||^2$$

2. Generate **B random datasets** (same size as real data, but uniformly distributed).
3. Compute the WCSS for each random dataset:

$$W_k^b = \sum_{i=1}^{K} \sum_{x_j \in C_i^b} ||x_j - \mu_i^b||^2, \quad b = 1, \dots, B$$

4. Compute the expected WCSS over all **B random datasets**:

$$E(W_k) = \frac{1}{B} \sum_{b=1}^{B} W_k^b$$

5. Compute the **Gap Statistic**:

$$G_k = \frac{1}{B} \sum_{b=1}^{B} \log(W_k^b) - \log(W_k)$$

6.

   Choose K where G_k is **maximum**.

Procedure

1. Compute G_k for different K.
2. Choose the **largest** G_k.

Limitations

- **Computationally expensive** due to multiple random dataset generations.
- Assumes clusters are compact and spherical.

4. Davies-Bouldin Index

Intuition

- Measures **inter-cluster similarity** (how well-separated clusters are).
- Lower values indicate **better clustering**.

Mathematics

1.

   Compute cluster dispersion:

$$S_i = \frac{1}{|C_i|} \sum_{x_j \in C_i} ||x_j - \mu_i||$$

2. Compute cluster separation between clusters i and j:

$$R_{ij} = \frac{S_i + S_j}{||\mu_i - \mu_j||}$$

3. Compute **Davies-Bouldin Index**:

$$DB = \frac{1}{K} \sum_{i=1}^{K} \max_{j \neq i} R_{ij}$$

4.

   Choose K with the **smallest DB index**.

Procedure

1. Compute DB index for different K.
2. Select K where DB is **minimum**.

Limitations

- Works well for convex clusters but fails for non-spherical clusters.
- Sensitive to noise.

Comparison of Methods for Choosing K

Method	Works Best When...	Computational Complexity	Key Limitation
Elbow Method	Clusters are well-separated	Low	The "elbow" is subjective
Silhouette Score	Clusters have clear boundaries	High	Expensive for large datasets
Gap Statistic	Random reference comparison is needed	Very High	Computationally expensive
Davies-Bouldin Index	Clusters have equal density	Medium	Fails for irregular cluster shapes

---

Final Takeaways

✅ **Elbow Method is the simplest approach** but is **subjective.**
✅ **Silhouette Score measures separation quality**, but is **expensive.**
✅ **Gap Statistic provides a rigorous test** but is **very slow.**
✅ **Davies-Bouldin Index works well for convex clusters** but fails for complex shapes.

5. Step-by-Step Pseudo Code for K-Means

plaintext

```
# Step 1: Initialize Clusters
1. Choose K (number of clusters).
2. Randomly select K data points as initial centroids.

# Step 2: Assign Points to Clusters
1. For each data point:
    a. Compute distance to each centroid.
    b. Assign point to the nearest centroid.

# Step 3: Recompute Centroids
1. For each cluster:
    a. Compute the mean of all assigned points.
    b. Update the centroid.

# Step 4: Check for Convergence
1. If centroids do not change significantly, stop.
2. Otherwise, repeat Steps 2 and 3.

# Step 5: Output Final Clusters
1. Return cluster assignments and final centroids.
```

6. Mathematical Breakdown

6.1 Objective Function (Minimizing Variance Within Clusters)

The goal of K-Means is to minimize the **intra-cluster variance**, measured by the sum of squared differences:

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

where:

- x_j is a data point.
- μ_i is the centroid of cluster C_i.
- The inner sum computes the variance within a cluster.

6.2 Distance Calculation

The most common distance metric used is **Euclidean distance**:

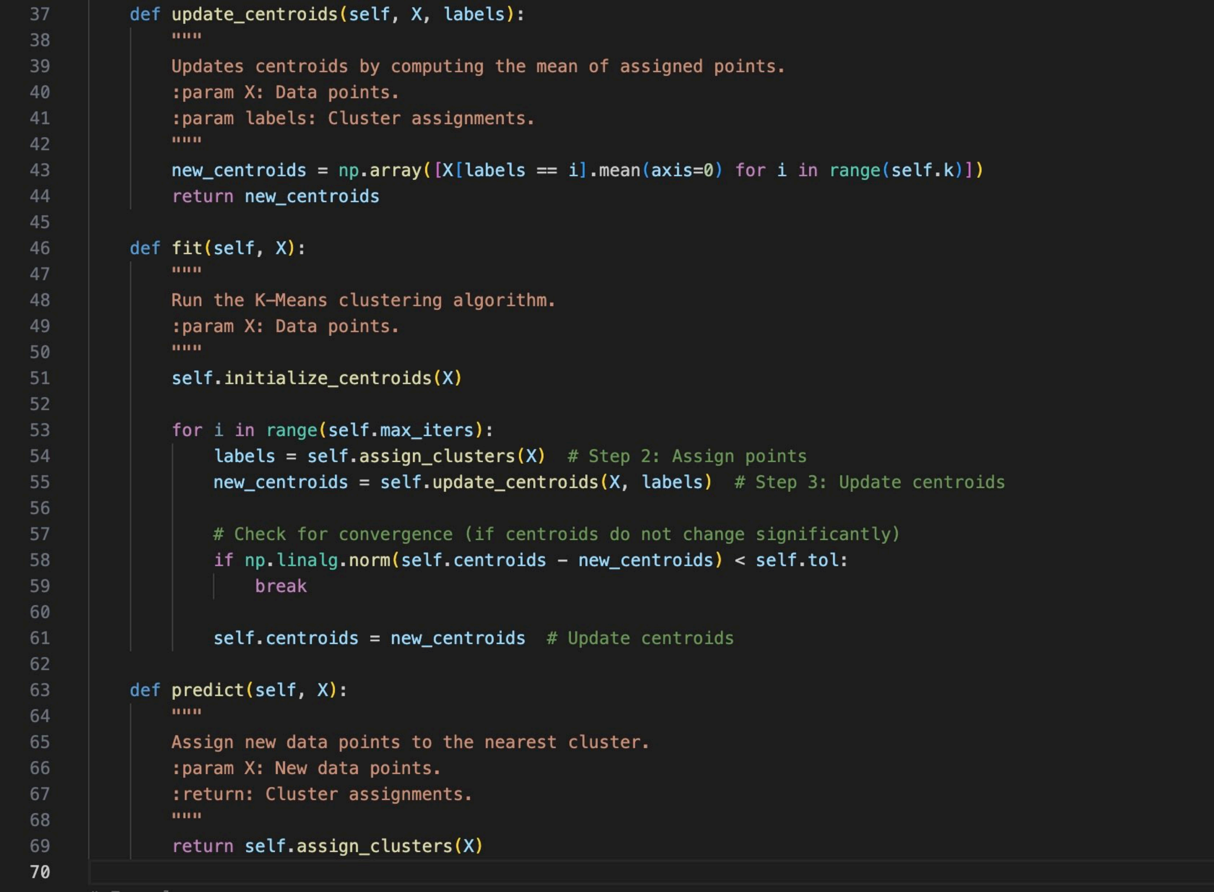
$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

6.3 Updating Centroids

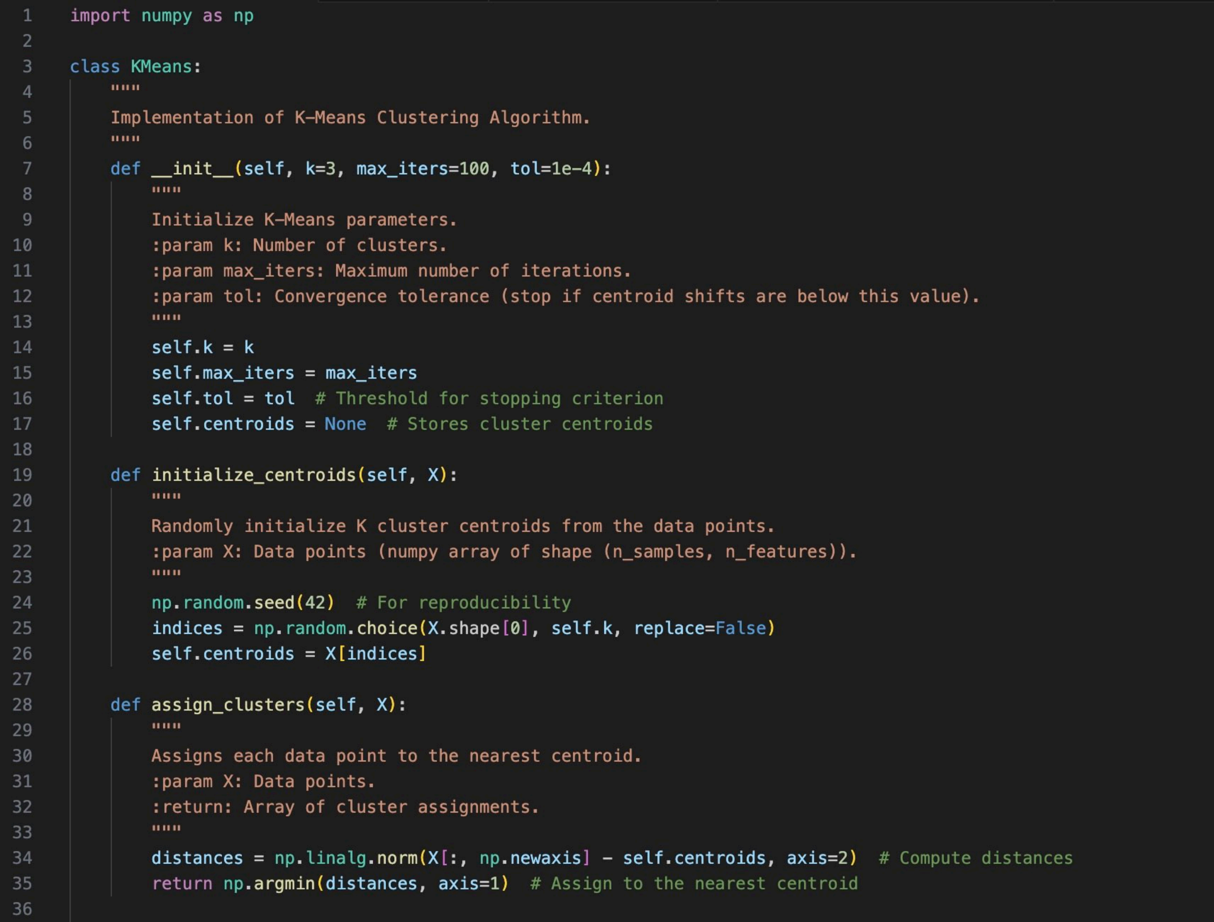
The new centroid μ_i is simply the **mean** of all assigned points:

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

7. Fully Commented Manual Implementation (From Scratch)



```python
    def update_centroids(self, X, labels):
        """
        Updates centroids by computing the mean of assigned points.
        :param X: Data points.
        :param labels: Cluster assignments.
        """
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(self.k)])
        return new_centroids

    def fit(self, X):
        """
        Run the K-Means clustering algorithm.
        :param X: Data points.
        """
        self.initialize_centroids(X)

        for i in range(self.max_iters):
            labels = self.assign_clusters(X)  # Step 2: Assign points
            new_centroids = self.update_centroids(X, labels)  # Step 3: Update centroids

            # Check for convergence (if centroids do not change significantly)
            if np.linalg.norm(self.centroids - new_centroids) < self.tol:
                break

            self.centroids = new_centroids  # Update centroids

    def predict(self, X):
        """
        Assign new data points to the nearest cluster.
        :param X: New data points.
        :return: Cluster assignments.
        """
        return self.assign_clusters(X)

```



```python
import numpy as np

class KMeans:
    """
    Implementation of K-Means Clustering Algorithm.
    """
    def __init__(self, k=3, max_iters=100, tol=1e-4):
        """
        Initialize K-Means parameters.
        :param k: Number of clusters.
        :param max_iters: Maximum number of iterations.
        :param tol: Convergence tolerance (stop if centroid shifts are below this value).
        """
        self.k = k
        self.max_iters = max_iters
        self.tol = tol  # Threshold for stopping criterion
        self.centroids = None  # Stores cluster centroids

    def initialize_centroids(self, X):
        """
        Randomly initialize K cluster centroids from the data points.
        :param X: Data points (numpy array of shape (n_samples, n_features)).
        """
        np.random.seed(42)  # For reproducibility
        indices = np.random.choice(X.shape[0], self.k, replace=False)
        self.centroids = X[indices]

    def assign_clusters(self, X):
        """
        Assigns each data point to the nearest centroid.
        :param X: Data points.
        :return: Array of cluster assignments.
        """
        distances = np.linalg.norm(X[:, np.newaxis] - self.centroids, axis=2)  # Compute distances
        return np.argmin(distances, axis=1)  # Assign to the nearest centroid

```

8. Scikit-Learn Implementation

```python
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

# Generate synthetic dataset
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=1.0, random_state=42)

# Train K-Means
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Get cluster assignments and centroids
labels = kmeans.labels_
centroids = kmeans.cluster_centers_

print("Cluster Centers:\n", centroids)
```

9. Advantages & Disadvantages of K-Means

✅ Advantages

- Simple and easy to implement.
- Scalable to large datasets.
- Fast because it uses simple distance calculations.

❌ Disadvantages

- Requires manual selection of K.
- Fails for non-spherical clusters.
- Sensitive to outliers.

Final Takeaways

✅ K-Means is a fundamental clustering algorithm that partitions data into K groups.

✅ Choosing the right K is crucial for meaningful clustering.

✅ Sensitive to initialization, but optimizations (K-Means++) can improve results.

Comparison of Popular Clustering Algorithms

Algorithm	Intuition (How It Works & What It Does)	When to Use	Pros	Cons
K-Means	Assigns points to the nearest centroid, then updates centroids iteratively to minimize variance.	When clusters are **spherical, well-separated**, and you know **K**.	Simple, fast, works well on large datasets.	Sensitive to **initialization** and **outliers**. Struggles with non-spherical clusters.
Hierarchical Clustering	Recursively merges (agglomerative) or splits (divisive) clusters based on similarity.	When you want a **dendrogram** (cluster hierarchy) and **don't know K**.	No need to predefine K, creates a full hierarchy.	Slow for large datasets, sensitive to **distance metrics**.
DBSCAN (Density-Based Clustering)	Groups points that have a minimum number of neighbors within a radius, marking outliers.	When clusters have **arbitrary shapes** and contain **noise**.	Detects **arbitrary-shaped** clusters, robust to noise.	Struggles with varying densities and **high-dimensional data**.
Gaussian Mixture Model (GMM)	Assumes data is generated from **multiple Gaussian distributions** and assigns probabilities to clusters.	When clusters **overlap** and have **elliptical shapes**.	Soft clustering, flexible cluster shapes.	Computationally expensive, requires **Gaussian assumption**.
Mean Shift	Moves data points toward areas of **highest density** (mode seeking).	When clusters have **varying densities** and you want automatic cluster detection.	No need to specify K, works for **irregular densities**.	Computationally expensive, sensitive to bandwidth selection.
Spectral Clustering	Uses graph-based methods and eigenvalues to separate data points into clusters.	When clusters are **connected but not necessarily spherical**.	Works well for **non-convex clusters**, handles complex structures.	Doesn't scale well, computationally expensive.

Final Takeaways

✅ **K-Means** → Best for **simple, well-separated clusters.**

✅ **DBSCAN** → Best for **arbitrary-shaped clusters with noise.**

✅ **Hierarchical Clustering** → Best when **you don't know K and want a hierarchy.**

✅ **GMM** → Best when **clusters overlap and have elliptical shapes.**

✅ **Mean Shift** → Best when **you want automatic cluster detection without K.**

✅ **Spectral Clustering** → Best for **complex structures and non-spherical clusters.**