K-Means

🎓 Topic: K-Means Clustering – Detailed Theory

🧠 What is K-Means?

K-Means is an unsupervised machine learning algorithm used to **partition a dataset into `K` distinct clusters**, where each data point belongs to the cluster with the **nearest mean (centroid)**.

It's one of the most widely used clustering techniques for:

- Pattern recognition
- Market segmentation
- Data compression
- Anomaly detection

🚀 The Goal

Given:

- A dataset of `n` observations, each with `d` features.
- A desired number of clusters, `K`.

Find:

- `K` cluster centroids
- An assignment of each data point to one of the `K` clusters

Objective:

Minimize the **Within-Cluster Sum of Squares (WCSS)** — i.e., the sum of squared distances between each point and its cluster's centroid.

$$\arg\min_{C} \sum_{i=1}^{K} \sum_{x \in C_i} \|x - \mu_i\|^2$$

Breakdown:

- KK: Number of clusters
- CiC_i: The set of points assigned to cluster ii
- μi\mu_i: The centroid (mean) of cluster ii
- ∥x−μi∥2\|x - \mu_i\|^2: Squared Euclidean distance between a point xx and the centroid of its cluster

This function minimizes the **within-cluster sum of squared distances**, ensuring points are as close as possible to their assigned centroids.

🔁 Step-by-Step Algorithm

1. **Initialization**

   Randomly choose `K` data points as initial centroids.

2. **Assignment Step**

   For each data point, assign it to the **nearest** centroid (based on Euclidean distance).

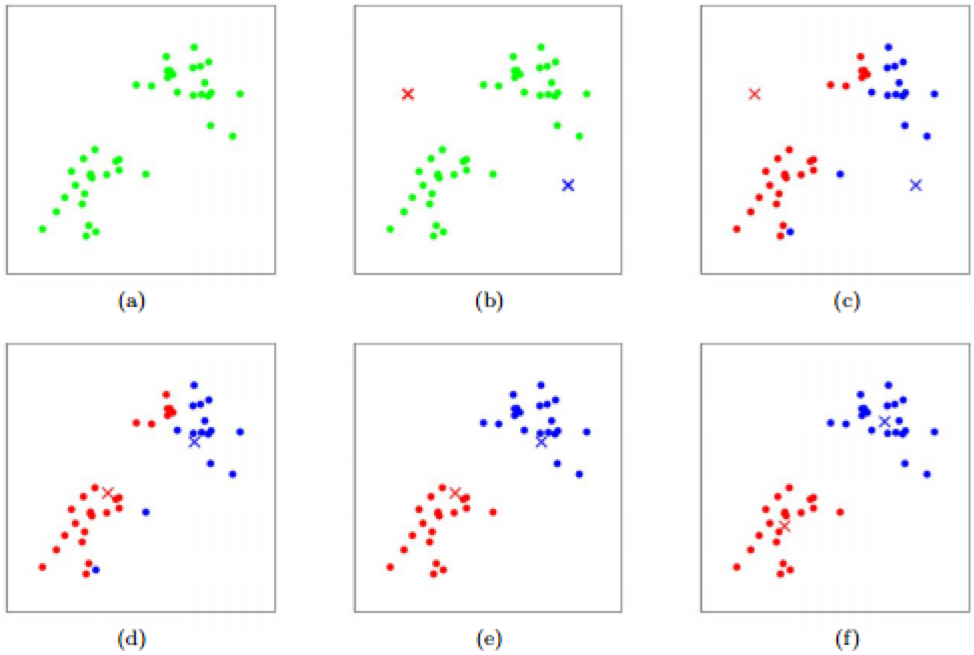
3. **Update Step**

   Recalculate each centroid as the **mean** of all data points assigned to that cluster.

4. **Repeat**

   Alternate between assignment and update until convergence:

   - No (or minimal) change in cluster assignments
   - Or centroids stop moving significantly



🧮 Why “Means”?

The algorithm uses the **arithmetic mean** of the points in a cluster to determine the centroid. That’s why it’s called *K-Means*.

🪄 Real-World Analogies

🛒 Market Segmentation (Retail)

Imagine a supermarket analyzing customer behavior:

- Features: Age, Income, Annual Spend
- Goal: Segment customers into groups like "budget-conscious", "luxury", "family-oriented", etc.
- K-Means finds clusters in this feature space to enable **personalized marketing**.

🎓 University Admissions

A university wants to group applicants based on academic performance:

- Features: GPA, standardized test scores, extracurriculars

- K-Means might reveal natural groupings like "High GPA/Low Extracurriculars" or "Moderate GPA/Strong Leadership"

🛰️ Image Compression

An image has thousands of colors (pixels). K-Means is used to group similar colors:

- Each pixel = a point in RGB space
- Cluster pixels into `K` colors → Replace pixel color with its centroid → Reduce storage size

---

⚠️ Assumptions & Limitations

Assumption	Description
Spherical Clusters	Assumes clusters are convex and roughly equal in size.
Euclidean Distance	Sensitive to scale; features must be normalized.
Fixed K	You must pre-specify the number of clusters.
Random Initialization	May converge to **local minima**. Use K-Means++ for better seeding.

💣 When K-Means Fails

1. **Clusters with different densities** – K-Means tends to split evenly, ignoring actual density.
2. **Non-spherical shapes** – It will poorly cluster moon-shaped or concentric ring data.
3. **Outliers** – One outlier can shift a centroid significantly.

💬 When to Use K-Means

✅ You have numeric data and you suspect the existence of *natural groups*

✅ You want **interpretability** (centroids can represent cluster "types")

✅ You need **scalability** (K-Means is fast and efficient on large datasets)

Avoid when:

- Clusters are expected to be non-spherical
- Outliers or noise are prevalent

🧠 Thought Questions for Class

1. Why is it important to scale/normalize features before applying K-Means?

Short answer:

Because K-Means uses **Euclidean distance**, and features with larger numeric ranges can **dominate** the distance calculation.

🔍 Explanation:

K-Means computes the distance between points and centroids using this formula:

$$||x - \mu|| = \sqrt{(x_1 - \mu_1)^2 + (x_2 - \mu_2)^2 + \ldots + (x_d - \mu_d)^2}$$

If one feature (e.g., income in dollars: 0–100,000) has a much larger scale than another (e.g., age: 0–100), then:

- The large-scale feature will **overpower** the distance calculation
- Clustering will be biased toward that feature

✅ Solution:

Use normalization techniques such as:

- **Min-Max Scaling** (scales features to [0, 1])
- **Standardization** (zero mean, unit variance)

---

2. Can K-Means work with categorical data? Why or why not?

Short answer:

No, not directly. K-Means is designed for **continuous numerical data**, not for **categorical features**.

❌ Why not:

- **Centroids are means** of data points. But the **mean of categories like {'red', 'blue', 'green'} is undefined.**
- **Euclidean distance** isn't meaningful for categories. For example:
  - Is "cat" closer to "dog" than to "car"? Euclidean metrics can't answer that.

✅ Alternatives:

If your data is **purely categorical**, consider:

- **K-Modes**: Uses **mode** instead of mean; appropriate for categorical features.
- **K-Prototypes**: Handles **mixed** numerical + categorical data.
- **Hierarchical clustering** with appropriate **distance metrics** like Hamming or Jaccard.

---

3. What would happen if your initial centroids are very poor?

Short answer:

You can end up with:

- **Suboptimal clusters** (poor local minimum)
- **Empty clusters**
- **Slow convergence**

🌀 Why this happens:

K-Means starts by **randomly selecting** `K` **initial centroids**, and then:

- All future steps depend on these initial points
- If two centroids are too close, one might dominate
- If a centroid starts in a sparse area, it may attract very few (or zero) points

🛠️ Solutions:

- Use **K-Means++**: A smarter initialization that spreads centroids apart based on distance probabilities

- Run K-Means **multiple times with different seeds**, and choose the best result (lowest WCSS)
