📘 COMPLETE NOTES: K-MEANS, ELBOW METHOD & SILHOUETTE SCORE

(Concept-first • Zero gaps • Exam + Interview ready)

PART 0 — WHY CLUSTERING EXISTS

In Machine Learning, not all datasets have labels.

Examples

- Customers without categories
- Students without grades
- Documents without topics

So we ask:

> **“Can the data organize itself into meaningful groups?”**

This is called **Unsupervised Learning**.

PART 1 — WHAT IS A CLUSTER?

A **cluster** is:

> A group of data points that are **more similar to each other** than to points in other groups.

Computers cannot understand “similarity” directly, so we define similarity using **distance**.

PART 2 — DISTANCE (MOST FUNDAMENTAL)

Distance used in K-Means: Euclidean Distance

Formula (copyable):

- `Distance(x, y) = √[(x₁ - y₁)² + (x₂ - y₂)² + …]`

Why distance matters:

- Smaller distance → more similar
- Larger distance → less similar

⚠️ **Distance depends on scale**

So:

- Features **must be scaled**
- Otherwise distance becomes meaningless

That's why we use:

- `X_train_scaled`

PART 3 — WHAT IS A CENTROID?

A **centroid** is:

> The **mean position** of all points in a cluster.

Example

Points:

- `(1,2), (3,4), (5,6)`

Centroid:

- `((1+3+5)/3 , (2+4+6)/3) = (3,4)`

📌 Important:

- Centroid is **not always a real data point**
- It is a **mathematical average**

PART 4 — GOAL OF K-MEANS (KEY STATEMENT)

> **K-Means places K centroids such that the total squared distance of all points to their nearest centroid is minimized.**

That total squared distance has a name.

PART 5 — WCSS / INERTIA (SOUL OF K-MEANS)

Full name:

Within-Cluster Sum of Squares (WCSS)

Meaning:

> "How tightly packed are the clusters?"

Mathematical definition (copyable):

- `WCSS = Σ (over i = 1 to K) Σ (over x ∈ Cᵢ) || x - μᵢ ||²`

Where:

- C_i = i-th cluster
- μ_i = centroid of cluster i

📌 In **scikit-learn**:

- `kmeans.inertia_`

So:

- `Inertia = WCSS`

PART 6 — WHY SQUARED DISTANCE?

Distance is squared because it:

- Removes negative values
- Penalizes far points more
- Makes optimization stable
- Gives a convex objective per cluster

This is why both **K-Means** and **k-means++** use squared distance.

PART 7 — K-MEANS ALGORITHM (STEP-BY-STEP)

STEP 1: Choose K

- You must specify the number of clusters
- K-Means **cannot decide K itself**

STEP 2: Initialize centroids (k-means++)

Why initialization matters

- Bad initialization → bad clusters

k-means++ logic

1. Choose first centroid randomly
2. Compute distance of each point to nearest centroid
3. Square the distance
4. Choose next centroid with probability ∝ squared distance
5. Repeat until K centroids are chosen

📌 Effect:

- Centroids start far apart
- Faster convergence
- Better clustering

STEP 3: Assignment step

For each data point:

- Compute distance to all centroids
- Assign to nearest centroid

STEP 4: Update step

For each cluster:

- Recompute centroid = mean of assigned points

STEP 5: Repeat

Repeat steps 3 & 4 until:

- Centroids stop changing
- WCSS stops decreasing

This is called **convergence**.

PART 8 — WHY INERTIA ALWAYS DECREASES

Because:

- More clusters → points closer to centroids
- Squared distances shrink

Extreme cases

- K = 1 → very large inertia
- K = N → inertia = 0

So inertia is **monotonically decreasing**.

PART 9 — PROBLEM: HOW TO CHOOSE K?

Since inertia always decreases:

- We **cannot** choose K by minimum inertia

Instead, we look for:

> **Point of diminishing returns**

This leads to the **Elbow Method**.

PART 10 — ELBOW METHOD (CONCEPT)

Core idea:

> Plot **WCSS vs K** and find the point where improvement slows down.

Before elbow

- Large drop in WCSS
- Meaningful structure captured

After elbow

- Very small improvement
- Overfitting / noise

PART 11 — ELBOW METHOD (CODE)

```
wcss = []

for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(X_train_scaled)
    wcss.append(kmeans.inertia_)
```

Plot:

- `plt.plot(range(1,11), wcss)`
- `plt.xticks(range(1,11))`
- `plt.xlabel("Number of Clusters")`
- `plt.ylabel("WCSS")`
- `plt.show()`

📌 `wcss` does **not** affect training
📌 It is used **only for analysis**

PART 12 — AUTOMATIC ELBOW DETECTION (KNEELOCATOR)

```
from kneed import KneeLocator

kl = KneeLocator(
    range(1,11),
    wcss,
    curve="convex",
    direction="decreasing"
)

kl.elbow
```

Output:

```
3
```

📌 This automatically finds the elbow point.

PART 13 — SILHOUETTE SCORE (BETTER METRIC)

What it measures

For each point:

- How close it is to its own cluster
- How far it is from other clusters

Silhouette coefficient range:

- **+1** → excellent clustering
- **0** → overlapping clusters
- **−1** → wrong clustering

Formula (conceptual)

- `s(i) = (b(i) - a(i)) / max[a(i), b(i)]`

Where:

- `a(i)` = average distance to own cluster
- `b(i)` = average distance to nearest other cluster

Code

```
from sklearn.metrics import silhouette_score

silhouette_coefficients = []

for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, init="k-means++")
    kmeans.fit(X_train_scaled)
```

- `    score = silhouette_score(X_train_scaled, kmeans.labels_)`
- `    silhouette_coefficients.append(score)`

Example output:

- `[0.5791, ...]`

📌 Starts from `k=2` because silhouette is undefined for `k=1`.

PART 14 — LIMITATIONS (VERY IMPORTANT)

K-Means fails when:

- Clusters are non-spherical
- Cluster sizes differ significantly
- Data contains strong outliers

Elbow method fails when:

- Curve is smooth
- No clear elbow exists

PART 15 — FINAL LOCK-IN SUMMARY

- K-Means minimizes **WCSS**
- Inertia = WCSS
- Squared distance penalizes far points

- k-means++ improves initialization
- Elbow method selects K using diminishing returns
- KneeLocator automates elbow detection
- Silhouette score measures clustering quality
- Feature scaling is mandatory
-