Data Science – Fall 2025/2026

INTRODUCTION TO Machine Learning

Lecture 4: Clustering

What is Clustering ?

Clustering is like having hundreds of photos and grouping them automatically by similarity: pictures with faces together, landscapes together, selfies together, etc. We don’t predefine categories. The clusters emerge automatically from the similarities in the data.

Real-World Examples

- **Library organization**: Putting books on similar topics together without knowing the exact categories in advance
- **Social circles**: Noticing that certain people naturally hang out together based on shared interests
- **Grocery store layout**: Grouping fruits together, dairy together, meats together
- **Music playlists**: Automatically grouping songs that have similar tempo, mood, or style

Why do we use Clustering ?

1. **Find Natural Groups**: Discover patterns we didn't know existed

2. **Simplify Data**: Understand millions of data points as just a few groups

3. **Customer Segmentation**: Group similar customers for targeted marketing

4. **Anomaly Detection**: Find unusual items that don't fit any group

5. **Data Exploration**: Get a quick overview of what's in your data

Mathematical Formulation

Clustering is an unsupervised machine learning technique that partitions a dataset into subsets (clusters) such that:

- **Intra-cluster** similarity is maximized
- **Inter-cluster** similarity is minimized

Given a dataset $X = \{x_1, x_2, ., ..x_n\}$ with $x_i \in \mathbb{R}^p$,clustering aims to find:

A partition $C = \{C_1, C_2, ., ..C_k\}$, where $\bigcup_{i=1}^{k} C_i = X$ and $C_i \cap C_j = \emptyset$ for $i \neq j$

Objective Function:

Most clustering algorithms optimize: $$\text{Minimize} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

, where μ_i is the centroid of cluster C_i and

$\|x - \mu_i\|$ is a distance (Euclidian, Mahalanobis, Manhattan, Cosine)

What is a Distance ?

A distance d on a set X is a function defined as follows:

$$d\colon X \times X \to \mathbb{R}_{\geq 0}$$

A distance assigns a non-negative real number to each pair of points in X and satisfies the following properties for all $x, y, z \in X$:

1.Non-negativity:

$$d(x, y) \geq 0$$

2.Identity of indiscernibles:

$$d(x, y) = 0 \text{ if and only if } x = y$$

3. Symmetry:

$$d(x, y) = d(y, x)$$

4. Triangle inequality:

$$d(x, z) \leq d(x, y) + d(y, z)$$

N.B. We use a distance to quantify how similar or different two objects are, allowing us to compare, group, or analyze them meaningfully.

Examples of a Distance

Euclidean Distance

$$d_{\text{Euclid}}(x, y) = \sqrt{\sum_{i=1}^{p}(x_i - y_i)^2}$$

where,

$x = (x_1, x_2, \ldots, x_p)$:first observation (a point in $\mathbb{R}^p$)

$y = (y_1, y_2, \ldots, y_p)$:second observation

p: number of dimensions (features)

x_i, y_i :value of the i-th feature for observations x and y

Mahalanobis Distance

$$d_{Mahalanobis}(x, y) = \sqrt{(x - y)^T \Sigma^{-1} (x - y)}$$

where,

x, y: two observations in $\mathbb{R}^p$

$x - y$:vector of component-wise differences

$(x - y)^T$:transpose of the difference vector

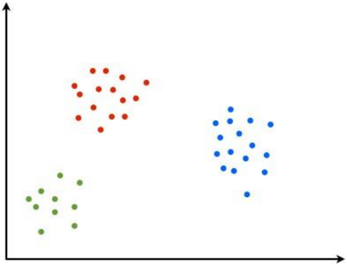
Σ: covariance matrix of the dataset (size $p \times p$)

Σ^{-1} :inverse of the covariance matrix

p: number of dimensions (features)

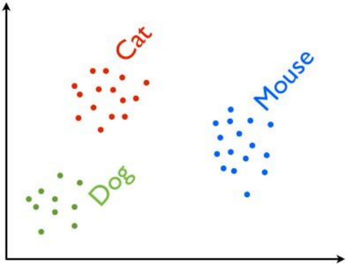
Example of Clustering

Clustering/Partition



"clusters",
"classes",
"blocks (of a partition)"

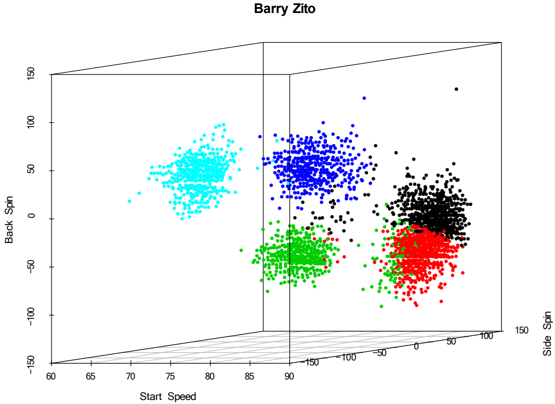
Clustering/Partition



“clusters”,
“classes”,
“blocks (of a partition)”

Note: In unsupervised learning, we work with data that has no predefined labels or categories. Unlike supervised approaches where we know the target classes (like "cat," "dog," or "mouse"), here we begin with completely unlabeled data, unaware of what natural groupings might exist. This is exactly how an unsupervised model operates, it doesn't receive answers or categories to learn from. Instead, its fundamental goal is to automatically discover the inherent separations and natural groupings within the data by identifying patterns and similarities that humans might not have anticipated. The algorithm explores the unknown structure of the data, seeking to organize it into meaningful clusters without any prior guidance about what those clusters should represent.

Clustering of Baseball Pitches

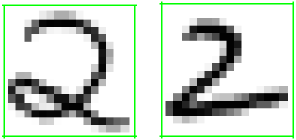


Inferred meaning of clusters: black – fastball, red – sinker, green – changeup, blue – slider, light blue – curveball

(Example from Mike Pane)

Clustering versus Classification

- In classification, we have data for which the class labels are known.
- In clustering, we look at data, where groups are unknown and undefined.
- We try to learn the groups themselves, as well as what differentiates them.



Clustering Algorithms

The two known clustering algorithms that are very simple to understand, visualize, and use are:

The k-means algorithm

Hierarchical clustering

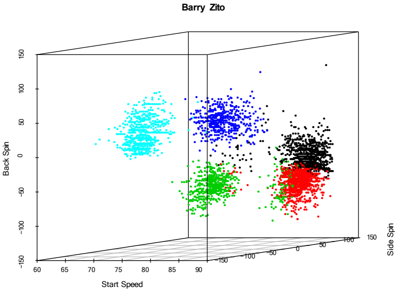
K-means clustering algorithm

- K-means clustering: simple approach for partitioning a dataset into K distinct, non-overlapping clusters.
  1. To perform K-means clustering: specify the desired number of clusters K.
  2. Then the K-means algorithm will assign each observation to exactly one of the K clusters.

Non-overlapping clusters are groups of data points that are separated, with no point belonging to more than one cluster.

K-means clustering: Notations

- Observations $X_1, ..., X_{i,} ,...,X_n$ / $X_i \in \mathbb{R}^p$
- Dissimilarity is defined as a distance and denoted: $d(X_i, X_j)$.
- Let K be the number of clusters (pre-defined).
- A clustering of points $X_1, \ldots X_n$ is a function C that assigns each observation X_i to a group $k \in \{1, \ldots K\}$



K-means clustering: Notations

Let n denote the number of data points in our data set.

Let $C_1, C_{2,}, \ldots, C_j, \ldots, C_k$ denoting sets containing the indices of the observations in each cluster ($C_j \neq \emptyset$, for all $j \in \{1, 2, \ldots, K\}$).

We have:

1. $$C_1 \cup C_2 \cup \cdots C_K = \{1, \ldots, n\}.$$
2. $$C_k \cap C_{k'} = \emptyset \text{ (for all } k \neq k)$$

K-means clustering: : Within-Cluster Variation

$C(i) = k$ means that X_i is assigned to group k, and $|C_k|$ is the number of points in the group k. Also, let $d(i,j) = d(X_i, X_j)$

We define the within-cluster variation as follows:

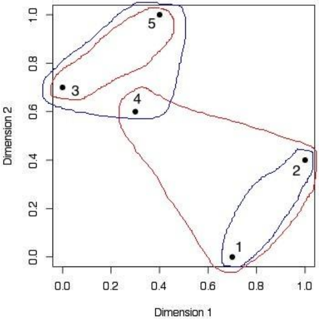
$$W = \sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{C(i)=k,\, C(j)=k} d_{ij}$$

Smaller *W* is better

Within-Cluster Variation: Simple Example

Here $n = 5$ and $K = 2$,
$X_i \in \mathbb{R}^2$ and d_{ij} is the Euclidian distance

	1	2	3	4	5
1	0	0.25	0.98	0.52	1.09
2	0.25	0	1.09	0.53	0.72
3	0.98	1.09	0	0.10	0.25
4	0.52	0.53	0.10	0	0.17
5	1.09	0.72	0.25	0.17	0



- Red clustering: $W_{\text{red}} = (0.25 + 0.53 + 0.52)/3 + 0.25/2 = 0.56$
- Blue clustering:
  $W_{\text{blue}} = 0.25/2 + (0.10 + 0.17 + 0.25)/3 = 0.30$

$$W = W_{\text{red}} + W_{\text{blue}}$$

Finding the best group assignments

Smaller W is better, so why don't we just directly find the clustering C that minimizes W?

Problem: doing so requires trying all possible assignments of the n points into K groups. The number of possible assignments is

$$A(n,K) = \frac{1}{K!}\sum_{k=1}^{K}(-1)^{K-k}\binom{K}{k}k^n$$

Note that $A(10,4) = 34\,105$, and $A(25,4) \approx 5 \times 10^{13}$

See, Jain and Dubes (1998), "Algorithms for Clustering Data"

Most problems we look at are going to have way more than $n = 25$ observations, and potentially more than $K = 4$ clusters too (but $K = 4$ is not unrealistic)

Finding the best group assignments

How do we get around this?

We will end up making an approximation. Let’s walk through all the details now of K-means clustering.

K-means: Basic Steps

- K-means is a simple way to paritition a data set into K distinct, non-overlapping clusters (each data point belongs to EXACTLY ONE cluster).
- To perform K-means clustering, we must:

1. first specify the desired number of clusters K
2. then the K-means algorithm will assign each observation to exactly one of the K clusters.

How does this work?

Intuition

We think of k-means being a good clustering algorithm when the within-cluster variation is as small as possible.

What is this?

The within cluster variation

The within cluster variation of cluster C_k is a measure of $W(C_k)$ of the amount by which the observations within a cluster differ from each other.

Mathematically, we want to solve the following optimization problem:

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k)$$

In words, this means that we want to partition the data points into clusters such that the total within-cluster variation summed over all K clusters is as small as possible.

This seems reasonable, but how do we define the within-cluster variation $W(C_k)$?

The within cluster variation

There are many possible ways to define this concept, but by far the most common choice involves squared **Euclidean distance**. Why ?

- It strongly penalizes points that are very far from the center
- It's mathematically convenient and computationally efficient
- It it possible measures how far points are from their cluster center

Another Distances

1. Manhattan Distance: to reduce the influence of outliers

2. Cosine Distance: Comparing article similarity

3. Mahalanobis Distance: Multivariate statistical analysis

The within cluster variation

Thus, we define the within-cluster variation $W(C_k)$:

$$W(C_k) = \frac{1}{|C_k|} \sum_{\substack{i<i' \\ x_i, x_{i'} \in C_k}} \sum_{j=1}^{j=p} (x_{ij} - x_{i'j})^2,$$

,where $|C_k|$ denotes the number of observations in the kth cluster.

In words, the within-cluster variation for the *k*th cluster is the sum of all of the pairwise squared Euclidean distances between the observations in the *k*th cluster, divided by the total number of observations in the *k*th cluster.

From now on we will replace $\displaystyle\sum_{\substack{i<i' \\ x_i, x_{i'} \in C_k}}$ by $\displaystyle\sum_{i,i'}$

Back to the optimization problem

We return now to the optimization problem that defines K-means clustering (under the Euclidean distance):

$$\min_{C_1,\ldots,C_K} \sum_{k=1}^{K} W(C_k) = \min_{C_1,\ldots,C_K} \{\sum_{k=1}^{K} \frac{1}{|C_k|} \sum_{i,i'} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2\} \quad (1)$$

Now, we would like to find an algorithm to solve equation 1.

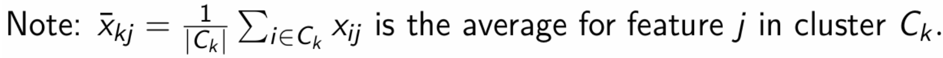
That is, we want a method to partition the observations into K clusters such that the objective of equation 1 is minimized.

Fortunately, a very simple algorithm can be shown to provide a local Optimum.

A pretty good solution.

K-means clustering algorithm

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations.
2. Iterate until the clustering algorithm stops changing:
   a. For each of the K clusters, compute the centroid. The kth cluster centriod is the vector of the p feature averages for the observations in the cluster k.
   b. Assign each observation to the cluster whose centroid is closest (where closest is defined using Euclidean distance).



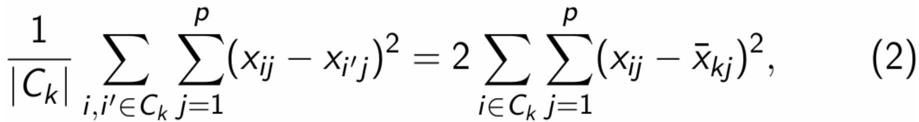
Note: $\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$ is the average for feature j in cluster C_k.

This algorithm is known as **Lloyd's** algorithm. There are improved variants of k-means, such as the Hartigan–Wong method (used by default in R), whereas Python typically uses Lloyd's or Elkan's algorithms.

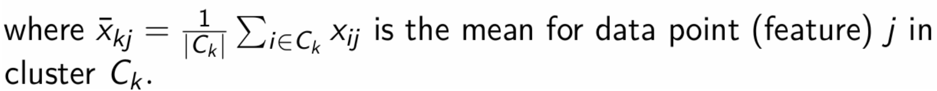
K-means clustering algorithm

The Lloyd's algorithm is guaranteed to decrease the value of the objective function at each step.

Why is this true?



$$\frac{1}{|C_k|}\sum_{i,i'\in C_k}\sum_{j=1}^{p}(x_{ij}-x_{i'j})^2 = 2\sum_{i\in C_k}\sum_{j=1}^{p}(x_{ij}-\bar{x}_{kj})^2, \qquad (2)$$



where $\bar{x}_{kj} = \frac{1}{|C_k|}\sum_{i\in C_k} x_{ij}$ is the mean for data point (feature) j in cluster C_k.

K-means clustering: The algorithm

- In Step 2(a), centroids represent the ideal center of each cluster. By avergaing all points in a cluster, we find its natural center point
- In Step 2(b), reallocating the data points can only improve the the within sum of squares.
- This means that as the algorithm is run, the clustering obtained will continually improve until the result no longer changes and the objective of equation 1 will never increase!
- When the result no longer changes, we reach a local optimum.

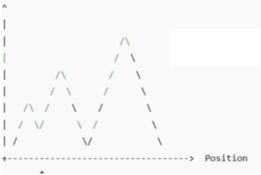
K-means clustering algorithm

Since the algorithm reaches a local optimum and not a global optimum, the results obtained will depend on the initial (random) cluster assignment of each observation in Step 1.

- Due to this, it's crucial to run the algorithm many times and from multiple (random) starting points.
- One should select the best solution: the one where the objective function is the smallest.

A local optimum is a solution where the algorithm cannot improve the objective function by making small changes, even though a better

(global) solution may exist elsewhere in the search space.



Example 1

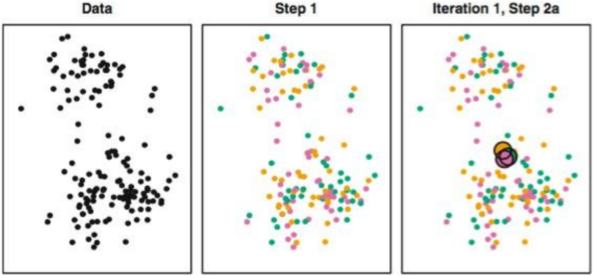


Figure 1: Top left: the data is shown. Top center: in Step 1 of the algorithm, each observation is randomly assigned to a cluster. Top right: in Step 2(a), the cluster centroids are computed. These are shown as large colored disks. Initially the centroids are almost completely overlapping because the initial cluster assignments were chosen at random.

Example 1 (continued)

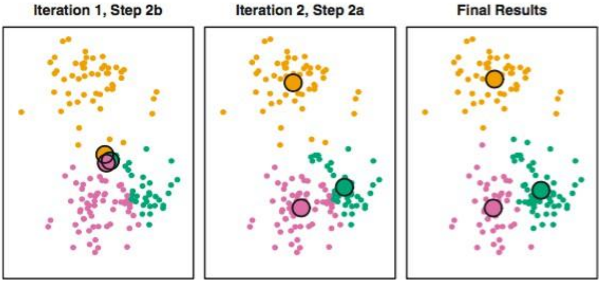


Figure 2: Bottom left: in Step 2(b), each observation is assigned to the nearest centroid. Bottom center: Step 2(a) is once again performed, leading to new cluster centroids. Bottom right: the results obtained after ten iterations.

Example 2

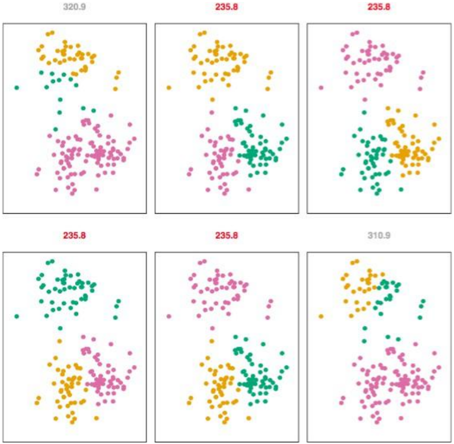


Figure 3: K-means clustering performed six times; K = 3, each time with a different random assignment of the observations in Step 1.

Above each plot is the value of the objective. Those labeled in red (objective value of 235.8) achieved the same best solution.

Application

We begin with a simple simulated example in which there truly are two clusters in the data: the first 25 observations have a mean shift relative to the next 25 observations.

```
# Set seed for reproducibility
np.random.seed(2)

# Generate a 50x2 matrix of random numbers from standard
# normal distribution (mean = 0 , std = 1). We are creating 50 data points,
# each with 2 features (x and y coordinates)
x = np.random.randn(50, 2)

# Modify the first 25 points to create two distinct clusters:
# - Add 3 to the x-coordinate (first column) of first 25 points
# This shifts them 3 units to the RIGHT on x-axis
x[0:25, 0] = x[0:25, 0] + 3

# Subtract 4 from the y-coordinate (second column) of first 25 points
# This shifts them 4 units DOWN on y-axis
x[0:25, 1] = x[0:25, 1] - 4
```

Perform K-means clustering, $K = 2$

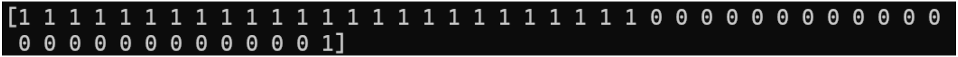
```
# KEY PARAMETER EXPLANATION:
# n_clusters=2: We want to partition data into 2 clusters
# n_init=20: Run K-means 20 times with different random starting points and
# keep the best result. This helps avoid poor local optimum and finds better
# cluster assignments
# random_state=2: Makes results reproducible

km_out = KMeans(n_clusters=2, n_init=20, random_state=2).fit(x)
```

Perform K-means clustering, $K = 2$

The cluster assignments of the 50 observations can be found by the following syntax:

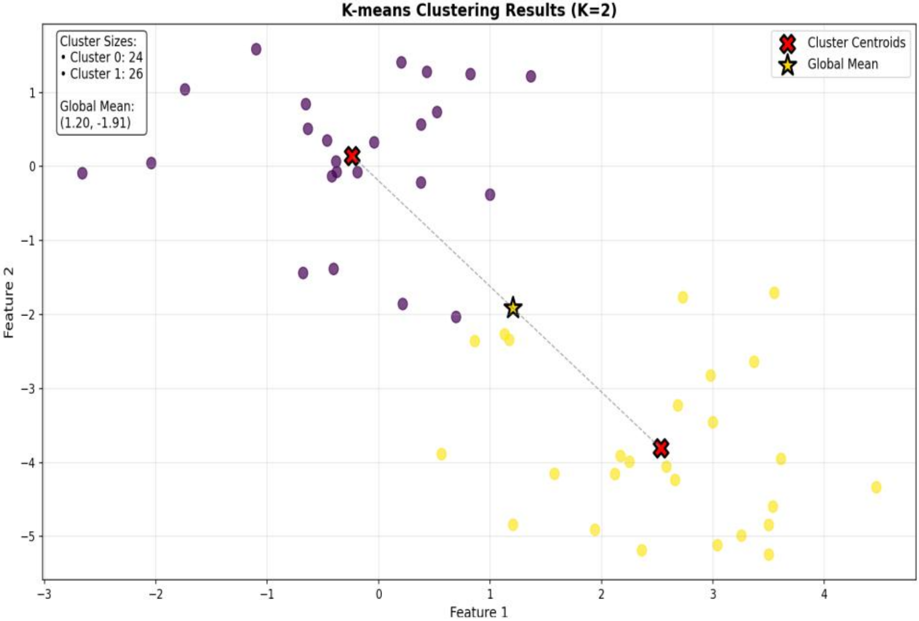
```
cluster_assignments = km_out.labels_
print(cluster_assignments )
```



```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1]
```

The K-means clustering perfectly separated the observations into two clusters even though we did not supply any group information (labels) to kmeans

Plotting with cluster assignment

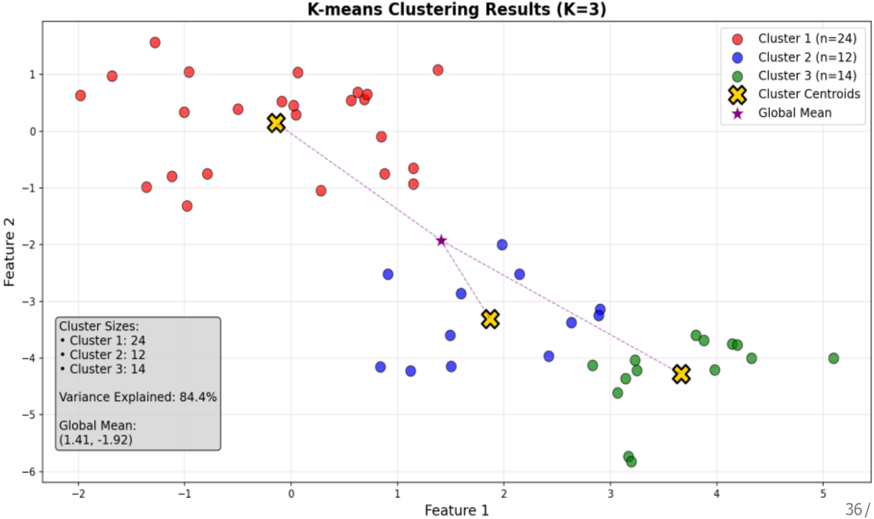


Here the observations can be easily plotted because they are two-dimensional. If there were more than two variables then we could instead perform PCA and plot the first two principal components score vectors.

Other values of K

In general we don't know K, so we need to play around with this value.

What happens if we look at $K = 3$.



Decomposition of Variation in K-Means Clustering

Definitions

Let:

- x_i = the i^{th}data point$\left(\in \mathbb{R}^p\right)$
- $\bar{x}$ = the overall mean of all points $\left(\in \mathbb{R}^p\right)$
- μ_k =the mean (centroid) of cluster k $\left(\in \mathbb{R}^p\right)$
- C_k = the set of points in cluster k
- n_k =number of points in cluster k
- K= total number of clusters

Decomposition of Variation in K-Means Clustering

Total Variation (Total Sum of Squares : TSS)

$$TSS = \sum_{i=1}^{n} \|x_i - \bar{x}\|^2 = \sum_{i=1}^{n} \sum_{j=1}^{p} (x_{ij} - \bar{x}_j)^2$$

, where:

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$$

is the mean of feature *j* across all observations.

Decomposition of Variation in K-Means Clustering

Within-Cluster Variation (Within Sum of Squares : WSS)

$$WSS = \sum_{k=1}^{K} \sum_{x_i \in C_k} \|x_i - \mu_k\|^2 = \sum_{k=1}^{K} \sum_{x_i \in C_k} \sum_{j=1}^{p} (x_{ij} - \mu_{kj})^2$$

, where:

$$\mu_{kj} = \frac{1}{n_k} \sum_{x_i \in C_k} x_{ij}$$

is the mean of feature j within cluster k.

Decomposition of Variation in K-Means Clustering

Between-Cluster Variation (Between Sum of Squares : BSS)

$$BSS = \sum_{k=1}^{K} n_k \|\mu_k - \bar{x}\|^2 = \sum_{k=1}^{K} n_k \sum_{j=1}^{p} (\mu_{kj} - \bar{x}_j)^2$$

Decomposition of Variation in K-Means Clustering

K-Means splits total variation into between-cluster and within-cluster parts:

$$TSS = BSS + WSS$$

The explained variance (called clustering efficiency or R²) is the fraction of total variation explained by the clusters structure:

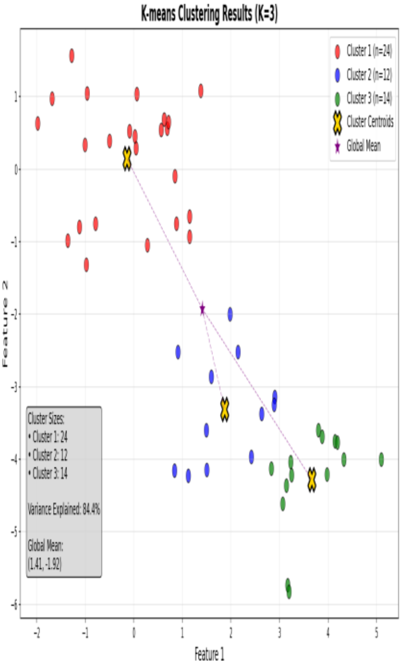
$$EV = \frac{BSS}{TSS} = 1 - \frac{WSS}{TSS}$$

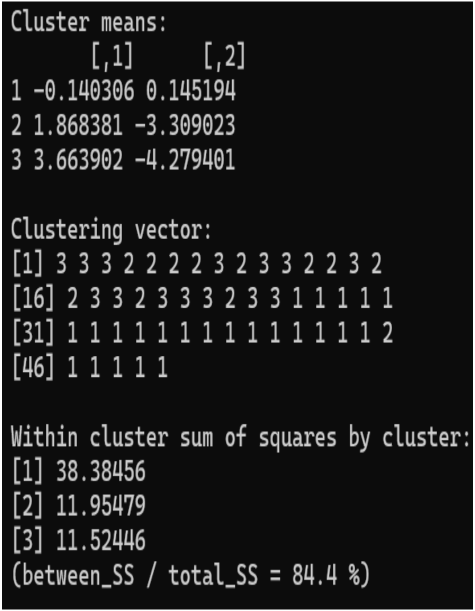
Interpretation

A higher ***EV*** means clusters are well-separated (large between clusters variance)

Goal of K-Means

Maximize EV ⇔ Minimize WSS and Maximize BSS





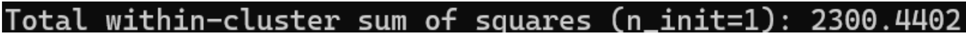
```
Cluster means:
       [,1]      [,2]
1 -0.140306 0.145194
2 1.868381 -3.309023
3 3.663902 -4.279401

Clustering vector:
[1] 3 3 3 2 2 2 2 3 2 3 3 2 2 3 2
[16] 2 3 3 2 3 3 3 2 3 3 1 1 1 1 1
[31] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
[46] 1 1 1 1 1

Within cluster sum of squares by cluster:
[1] 38.38456
[2] 11.95479
[3] 11.52446
(between_SS / total_SS = 84.4 %)
```

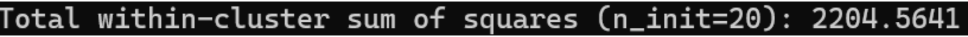
Varying the n_init value

```
# K-means with n_init=1
km_out = KMeans(n_clusters=3, n_init=1, random_state=3)
km_out.fit(x)
print(f"Total within-cluster sum of squares (n_init=1): {km_out.inertia_:.4f}")
```



```
Total within-cluster sum of squares (n_init=1): 2300.4402
```

```
# K-means with n_init=20
km_out = KMeans(n_clusters=3, n_init=20, random_state=3)
km_out.fit(x)
print(f"Total within-cluster sum of squares (n_init=20): {km_out.inertia_:.4f}")
```



```
Total within-cluster sum of squares (n_init=20): 2204.5641
```

- km.out$inertia is the total within-cluster sum of squares, which we seek to minimize by performing K-means clustering

Recommended settings

- Recommend always running K-means clustering with a large value of n_init, such as 20 or 50, since otherwise an undesirable local optimum may be obtained.
- Make sure you always set a random seed as well so that you can reproduce your results.

K-means: Discussions

- The main question that is posed by k-means is how many clusters K should we choose?
- Anytime we make such a choice regarding K, this can have a strong impact on the results obtained.
- In practice, we try several different choices, and look for the one with the most useful or interpretable solution.
- With these methods, there is no single right answer: any solution that exposes some interesting aspects of the data should be considered.

Hierarchical Clustering

Hierarchical clustering is a grouping algorithm that builds a hierarchy of clusters by progressively merging similar items together. It works step by step, starting with individual data points and combining the most similar ones at each stage, much like building a family tree for your data.

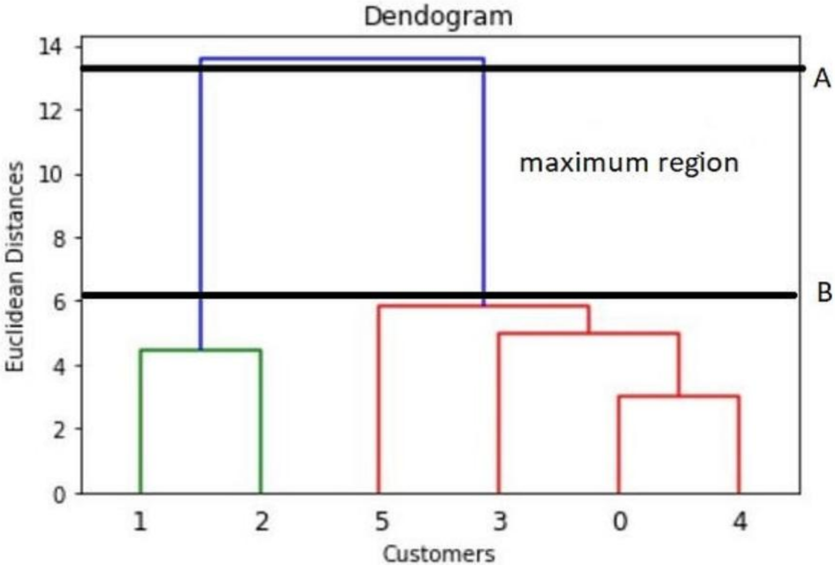
Steps of the Algorithm:

- Start by treating every item as its own little “family.”
- Combine the two most similar ones.
- Combine the next closest groups.
- Keep doing this until everything is merged into one big group.

The result is a tree-like diagram (a *dendrogram*) that shows not just the final clusters but also how they formed along the way.

It’s like watching friendships form at a party from individuals to small groups to bigger groups, while keeping track of who teamed up first.

Dendrogram Example



Hierarchical Clustering

Suppose we have n points ($x_1, \dots, x_n \in \mathbb{R}^p$) such as feature vectors describing customer behavior or attributes. Our goal is to cluster these points hierarchically in order to capture the complex, multi-scale structure present in real datasets.

Divisive Approach

The **divisive** approach, starts with all points grouped into a single cluster and then repeatedly splits this cluster into smaller ones. For example, we might first apply k-means with $k = 2$ to obtain two clusters, then apply the same procedure within each cluster, and continue recursively. In practice, this top-down strategy often produces less meaningful results than the **agglomerative** approach.

Agglomerative Approach

The **agglomerative** approach begins with each point in its own cluster and then repeatedly merges clusters until only one remains. At each step, the algorithm identifies the two closest clusters (according to some metric d) and merges them. The meric function d does not need to be a true distance; it only needs to be non-negative and symmetric.

Agglomerative Approach : The Algorithm

The algorithm proceeds as follows:

1. Initialization:

1) Start with the set of clusters $C = \{ \{1\}, \dots, \{n\} \}$.

2) Initialize the list of merges $L = \emptyset$.

2. Agglomeration:

for $t = 1, \dots, n-1$:

1) Find the two distinct clusters A and B such that $A, B \leftarrow \underset{\substack{a,b \in C \\ a \neq b}}{\arg\min} \; d(a, b)$

2) Merge them into a new cluster: $C_{new} \leftarrow A \cup B$

3) Remove A and B from the set of clusters $C \leftarrow C \setminus \{A, B\}$

4) Add the new cluster C to the set: $C \leftarrow C \cup \{C_{new}\}$

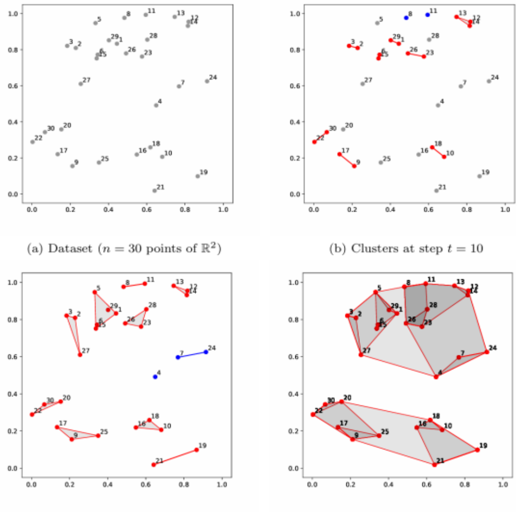
5) Record the merge by appending (A, B) to L: $L \leftarrow L + (A, B)$

3. Output:

Return the list of merges L, which encodes the hierarchy.

Agglomerative Approach : The Algorithm

Note that after step t, there are exactly $n - t$ clusters in C. An example of this process is illustrated in Figure below (for $n = 30$ points in $\mathbb{R}^2$).



(a) Dataset ($n = 30$ points of $\mathbb{R}^2$)

(b) Clusters at step $t = 10$

(c) Clusters at step $t = 20$

(d) Final clustering

Agglomerative Approach : Application

Consider the four points in $\mathbb{R}^2$:

$$x_1 = (0,0), x_2 = (1,0), x_3 = (5,0), x_4 = (6,0)$$

We will use **Euclidean distance** and **single linkage** (distance between clusters = min distance between any pair of points).

Step 0 — Initialization

Each point starts as its own cluster:

$$C = \{ \{x_1\}, \{x_2\}, \{x_3\}, \{ x_4\} \} = \{ \{1\}, \{2\}, \{3\}, \{4\} \}$$

Merge list: $L = \emptyset$

Step 1: First Merge (t=1)

$$d(\{1\},\{2\}) = \| x_1 - x_2 \| = \sqrt{(1-0)^2 + (0-0)^2} = 1$$

Minimum distance = 1
Two such pairs exist: $(1,2)$ and $(3,4)$.
Select randomly one: $(1,2)$

$$d(\{1\},\{3\}) = 5$$
$$d(\{1\},\{4\}) = 6$$
$$d(\{2\},\{3\}) = 4$$
$$d(\{2\},\{4\}) = 5$$
$$d(\{3\},\{4\}) = 1$$

Agglomerative Approach : Application

Merge:

$$A = \{1\}, B = \{2\}, C_{new} = \{1, 2\}$$

Update:

$$C = \{ \{1, 2\}, \{3\}, \{4\} \}$$

Record merge with its merge-distance *d*= *1*:

$$L = [(\{1\}, \{2\}; dist = 1)]$$

Nbr of clusters remaining in C : $4 - 1 = 3$

Agglomerative Approach : Application

Step 2: Second Merge ($t = 2$)

Compute distances between remaining clusters:

$$d(\{1,2\},\{3\}) = \min\{d(1,3), d(2,3)\} = \min\{5,4\} = 4$$
$$d(\{1,2\},\{4\}) = \min\{6,5\} = 5$$
$$d(\{3\},\{4\}) = 1$$

Minimum distance = 1. Merge $\{3\}$ and $\{4\}$.

Merge:

$$A = \{3\}, B = \{4\}, C_{new} = \{3, 4\}$$

Update:

$$C = \{\{1, 2\}, \{3, 4\}\}$$

Record:

$$L = [(\{1\}, \{2\}; dist = 1), (\{3\}, \{4\}; dist = 1)]$$

Clusters remaining: $4 - 2 = 2$

Agglomerative Approach : Application

Step 3: Third Merge ($t = 3$)

Only two clusters left:

$$d(\{1,2\},\{3,4\}) = \min\{d(1,3), d(1,4), d(2,3), d(2,4)\} = \min\{5,6,4,5\} = 4$$

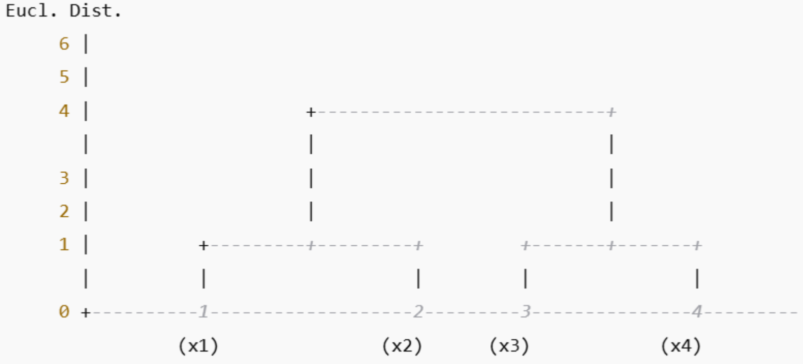
Merge them into $\{1,2,3,4\}$ with merge-distance 4.

Final:

$$C = \{\{1,2,3,4\}\}, L = [(\{1\},\{2\},1),(\{3\},\{4\},1),(\{1,2\},\{3,4\},4)].$$

Clusters remaining: $4 - 3 = 1$

Agglomerative Approach : The Dendrogram



```
Eucl. Dist.
      6 |
      5 |
      4 |                 +---------------------------+
        |                 |                           |
      3 |                 |                           |
      2 |                 |                           |
      1 |      +----------+---------+        +--------+-------+
        |      |                    |        |                |
      0 +------1--------------------2--------3----------------4--------
              (x1)                (x2)     (x3)             (x4)
```

Interpretation

At distance = 1

x_1 merges with x_2 →cluster $\{1, 2\}$

x_3 merges with x_4 →cluster $\{3, 4\}$

At distance = 4

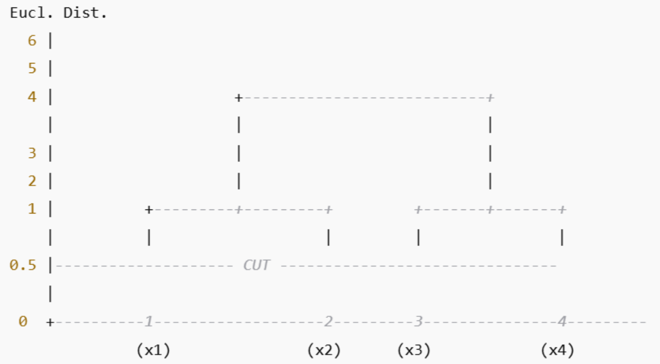
The two $\{1, 2\}$ and $\{3, 4\}$merge

Agglomerative Approach : Final Clusters

- Hierarchical agglomerative clustering builds a tree of merges (Dendrogram)

- Final clusters are determined by choosing a **cut height** on the dendrogram.

cut = 0.5

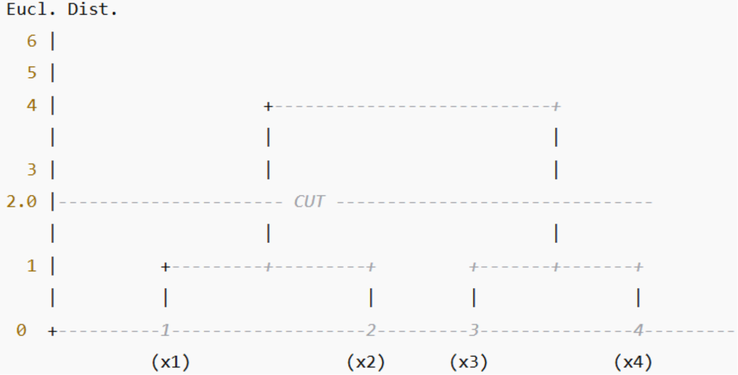


```
Eucl. Dist.
  6 |
  5 |
  4 |                     +---------------------------+
    |                     |                           |
  3 |                     |                           |
  2 |                     |                           |
  1 |           +---------+---------+         +-------+-------+
    |           |                   |         |               |
0.5 |-------------------- CUT  --------------------------------
    |
  0 +-----------1-------------------2---------3---------------4--------
              (x1)                (x2)      (x3)            (x4)
```

Resulting clusters: {1}, {2}, {3}, {4}
(the cut is below the first merges at distance 1, so no points have joined)

Agglomerative Approach : Final Clusters

cut = 2

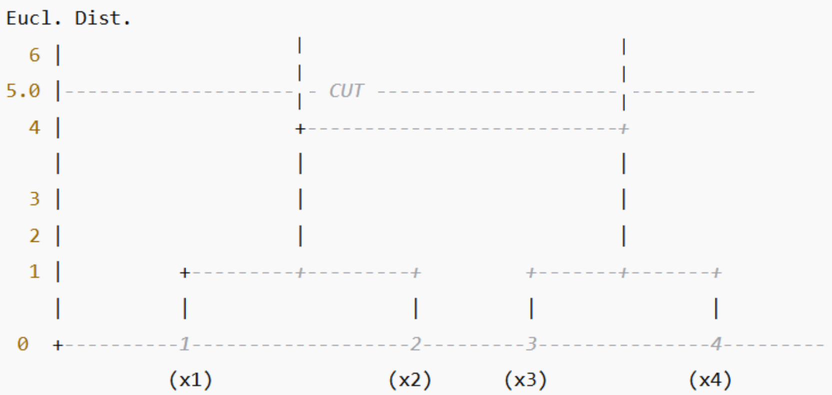


```
Eucl. Dist.
  6 |
  5 |
  4 |                  +---------------------------+
    |                  |                           |
  3 |                  |                           |
2.0 |--------------------- CUT -----------------------------
    |                  |                           |
  1 |          +-------+--------+        +---------+-------+
    |          |                |        |                 |
 0  +----------1----------------2--------3-----------------4--------
              (x1)             (x2)     (x3)              (x4)
```

Resulting clusters: `{1,2}`, `{3,4}`
(the cut is above the merges at 1, so (1,2) and (3,4) are formed

Agglomerative Approach : Final Clusters

cut = 5



```
Eucl. Dist.
  6 |                    |                           |
                         |                           |
5.0 |--------------------|- CUT ---------------------| -----------
  4 |                    +---------------------------+
    |                    |                           |
  3 |                    |                           |
  2 |                    |                           |
  1 |          +---------+---------+         +-------+-------+
    |          |                   |         |               |
 0  +----------1-------------------2---------3---------------4---------
              (x1)               (x2)      (x3)            (x4)
```

Resulting cluster: `{1,2,3,4}`
(the cut is above the final merge at distance 4. Everything is connected into one cluster.)

Agglomerative Approach : Linkage Choices

- The metric $d(A, B)$ used in the agglomerative algorithm (step 2.1) can change.

- In the previous example, we used **minimum linkage** (also called single linkage), where:

$$d(A, B) = \min\{\, d(x_i, x_j) : x_i \in A,\ x_j \in B \,\}$$

- This means that clusters merge when **any** pair of points (one in each cluster) are sufficiently close.

- This produces the *chaining effect*.

Agglomerative Approach : Maximum Linkage

Maximum linkage (complete linkage)

$$d(A,B) = \max\{\, d(x_i, x_j) : x_i \in A,\ x_j \in B \,\}$$

Clusters merge only when all points between the two clusters are close. This avoids chaining and tends to produce compact, concentrated clusters.

Complete linkage clusters are formed only if the worst-case (farthest) distance between the groups is small.

Clusters are more spherical and small.

Agglomerative Approach : Average Linkage

Average Linkage

$$d(A,B) = \frac{1}{|A|\,|B|} \sum_{x_i \in A} \sum_{x_j \in B} d(x_i, x_j)$$

We look at the average distance between points in the two clusters.

Produces an intermediate behavior:
- less chaining than single linkage
- less compact than complete linkage

The Average Linkage is a balanced compromise.

Agglomerative Approach : Ward's Linkage

Ward's linkage (variance minimization)

Ward's method does not use distances directly. It merges the two clusters whose merger increases the within-cluster variance the least.

For a cluster C, we have:
μ_C = the mean of the points in cluster C
The within-cluster variance is:

$$WSS(C) = \sum_{x \in C} \| x - \mu_C \|^2$$

$WSS(C) = 0$ → a cluster with one point.

$WSS(C)$ **increases** as a cluster becomes larger or more spread out.

Agglomerative Approach Ward’s Linkage

What happens when merging two clusters A and B?

Let the means be:

- μ_A =mean of cluster A
- μ_B = mean of cluster B
- $\mu_{A \cup B}$ =mean of the merged cluster

Ward’s method cares about how much the **total variance** increases after merging:

$$\Delta(A, B) = WSS(A \cup B) - WCSS(A) - WCSS(B)$$

This is the *increase in within-cluster variance* caused by merging A and B.

Ward’s method chooses the pair A, B that minimizes this increase.

$$\Delta(A, B) = \frac{|A| \quad |B|}{|A|+|B|} \| \mu_A - \mu_B \|^2$$

Beyond Euclidean Distance

So far, whether using minimum, maximum, average, or Ward’s linkage, we used the Euclidean distance to compute the distance between points. Euclidean distance is natural and easy to interpret in most cases, but it is not the only option.

In practice, we can replace it with any other distance or dissimilarity measure, depending on the structure of the data. For example, the Mahalanobis distance takes into account the correlations and different scales of the variables, making it useful when features have very different variances or are correlated. Using a different distance metric changes the dendrogram and the resulting clusters, but the agglomerative algorithm itself (the way clusters are merged based on the chosen linkage) remains exactly the same.

Validating the Clusters Obtained

Whenever we run a clustering algorithm (K-Means or Hierarchical), we will always obtain some clusters.

The key question is whether these clusters represent real subgroups in the data or are simply the result of random noise (random fluctuations that do not reflect any true structure).

To evaluate this, we might ask: *if we collected a new, independent dataset, would we observe the same clustering pattern again?*

Several methods exist to assign a p-value to clusters, helping us determine whether a cluster is stronger than what would be expected by chance.

However, there is no single universally accepted method for this assessment (see Hastie et al., 2009 for more details).

Other Considerations in Clustering

- Both K-means and hierarchical clustering will assign each observation to a cluster.
- However, sometimes this might not be appropriate.
- For instance, suppose that most of the observations truly belong to a small number of (unknown) subgroups, and a small subset of the observations (outliers) are quite different from each other and from all other observations.
- Then since K-means and hierarchical clustering force every observation into a cluster, the clusters found may be heavily skewed due to the presence of outliers that do not belong to any cluster.
- Mixture models are an attractive approach for accommodating the presence of such outliers.
- These amount to a soft version of K-means clustering, and are described in Hastie et al. (2009).