Cluster Analysis: Methods and Applications

Statistical Data Analysis

Prof. Dr. Jörg Osterrieder

Outline

Learning Objectives

After completing this lesson, you will be able to:

1. Apply k-Means and k-Medoids algorithms to partition data into clusters
2. Select appropriate distance metrics and understand when standardization is needed
3. Perform hierarchical clustering and interpret dendrograms with different linkage methods
4. Determine the optimal number of clusters using elbow, silhouette, and gap statistic methods

---

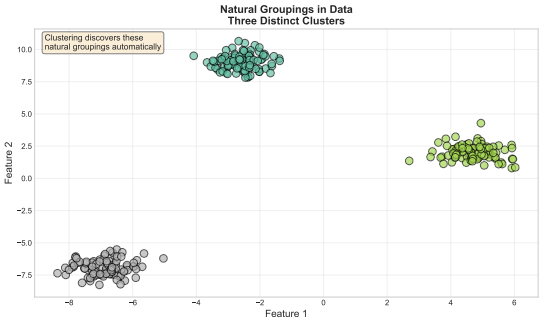
These four skills cover the complete clustering workflow from algorithm selection to validation

Introduction to Clustering

What is Clustering?

Core Concept

- Clustering groups similar observations together without predefined labels
- It is an unsupervised learning method — the algorithm discovers structure on its own
- Goal: maximize similarity within clusters, maximize dissimilarity between clusters
- Applications: customer segmentation, image compression, document clustering, anomaly detection

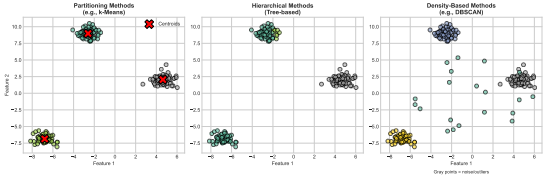


Clustering discovers natural groupings — the data speaks for itself without labels

Types of Clustering Methods

Method Families

- Partitional methods (k-Means, k-Medoids): divide data into k non-overlapping groups
- Hierarchical methods: build a tree of nested clusters (agglomerative or divisive)
- Density-based methods (DBSCAN): find clusters as high-density regions separated by low density
- The choice depends on data structure, cluster shapes, and whether k is known in advance



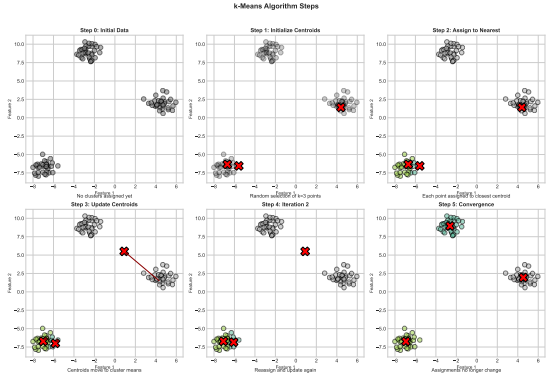
Partitional methods need k in advance; hierarchical methods let you explore first

k-Means Clustering

k-Means: The Core Idea

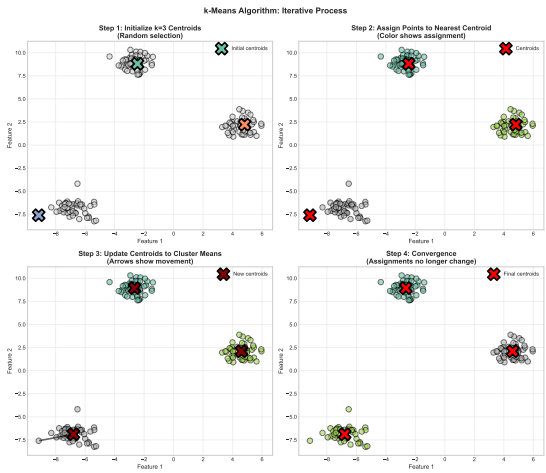
Algorithm Overview

- Partition n observations into k clusters, each represented by its centroid (mean)
- Assign every point to the cluster with the nearest centroid
- Recompute centroids as the mean of all assigned points
- Iterate assignment and update until centroids stabilize (convergence)



k-Means alternates between assigning points and updating centroids until convergence

k-Means Algorithm in Action



Algorithm Steps:

- Step 1: Randomly initialize k centroids
- Step 2: Assign each point to nearest centroid
- Step 3: Recompute centroids as cluster means
- Step 4: Repeat until convergence

Each iteration reduces WCSS — the algorithm is guaranteed to converge in finite steps

k-Means: Objective Function

Within-Cluster Sum of Squares

- WCSS measures cluster compactness
- k-Means minimizes WCSS by alternating assignment and update steps
- Guaranteed to converge but may find a local (not global) minimum

Objective Function

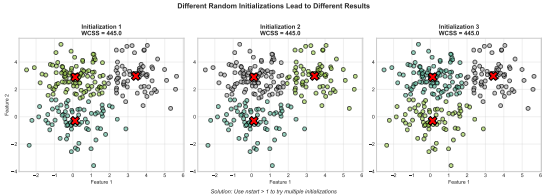
$$\min_{C_1,\ldots,C_k} \sum_{j=1}^{k} \sum_{x_i \in C_j} \|x_i - \mu_j\|^2$$

where μ_j = centroid of cluster C_j

WCSS measures cluster compactness: lower is better, but always decreases as k increases

Initialization Strategies

- Random initialization can lead to poor local optima — different starts give different results
- k-Means++ selects initial centroids spread apart: first center random, subsequent centers chosen with probability proportional to squared distance
- Use `nstart = 25` in R to run k-Means 25 times and keep the best result
- k-Means++ is the default initialization in most modern implementations



k-Means++ solves the initialization problem by spreading initial centroids apart

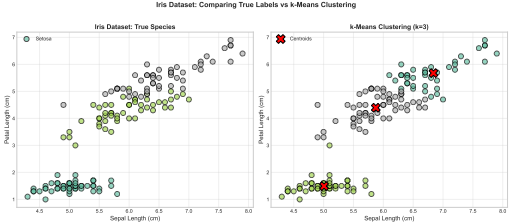
k-Means: Strengths and Limitations

Strengths

+ Fast and scalable — $O(nkd)$ per iteration, works on millions of points
+ Simple to implement and interpret — cluster centers are meaningful summaries

Limitations

− Assumes spherical, equally-sized clusters — fails on elongated or irregular shapes
− Sensitive to outliers — a single extreme point can pull the centroid away



k-Means is fast and simple but assumes spherical clusters — always check your assumptions

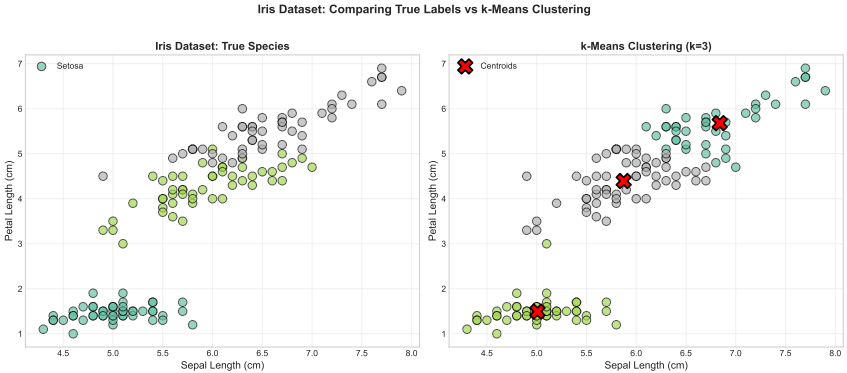
k-Means in R

```r
# Generate example data
set.seed(123)
x <- rbind(
  matrix(rnorm(100, mean=0, sd=0.3), ncol=2),
  matrix(rnorm(100, mean=1, sd=0.3), ncol=2),
  matrix(rnorm(100, mean=c(1,0), sd=0.3), ncol=2)
)

# Apply k-Means
km_result <- kmeans(x, centers=3, nstart=25)

# Extract results
centers <- km_result$centers
clusters <- km_result$cluster
wcss <- km_result$tot.withinss
```

```r
# Visualize results
plot(x, col=clusters, pch=19,
     main="k-Means Clustering",
     xlab="Feature 1", ylab="Feature 2")
points(centers, col=1:3, pch=3, cex=3, lwd=3)
legend("topright", legend=c("Cluster 1","Cluster 2","Cluster 3")
       ,
       col=1:3, pch=19)

# Print summary
cat("Within-cluster SS:", wcss, "\n")
cat("Between-cluster SS:", km_result$betweenss, "\n")
cat("Cluster sizes:", km_result$size, "\n")
```

nstart=25 runs k-Means 25 times with different seeds and keeps the best result

k-Means on the Iris Dataset



Key Observations:

- k-Means correctly separates Iris setosa but struggles with versicolor/virginica overlap
- The algorithm assigns hard boundaries — points near cluster borders may be misclassified
- Comparing known labels with k-Means clusters reveals both strengths and failure modes

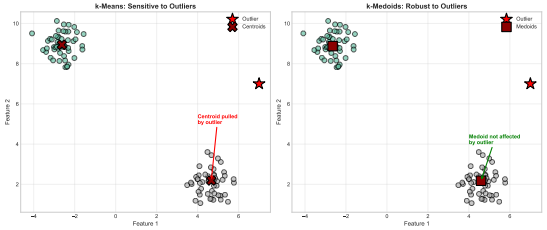
Real data like Iris shows both the power and the limits of k-Means — overlapping classes are hard

k-Medoids and Distance Metrics

k-Medoids: Robust Alternative to k-Means

How k-Medoids Works

- k-Medoids chooses actual data points (medoids) as cluster centers, not means
- PAM (Partitioning Around Medoids) is the standard algorithm — swaps medoids to minimize total dissimilarity
- Robust to outliers: an extreme point cannot pull the center away from the data
- Works with any distance metric, not just Euclidean — more flexible than k-Means



k-Medoids uses actual data points as centers — making it robust to outliers

k-Medoids (PAM) in R

```r
# Load required package
library(cluster)

# Generate example data with outlier
set.seed(123)
x <- rbind(
  matrix(rnorm(100, mean=0, sd=0.3), ncol=2),
  matrix(rnorm(100, mean=1, sd=0.3), ncol=2),
  c(5, 5)  # outlier
)

# Apply PAM
pam_result <- pam(x, k=2)

# Extract results
medoids <- pam_result$medoids
clusters <- pam_result$clustering
```

```r
# Visualize
plot(x, col=clusters, pch=19,
     main="PAM Clustering",
     xlab="Feature 1", ylab="Feature 2")
points(medoids, col=1:2, pch=8, cex=3, lwd=2)

# Compare with k-Means
km_result <- kmeans(x, centers=2, nstart=25)
plot(x, col=km_result$cluster, pch=19,
     main="k-Means Clustering")
points(km_result$centers, col=1:2, pch=3,
       cex=3, lwd=3)
```

PAM is available in the cluster package; use it when data contains outliers

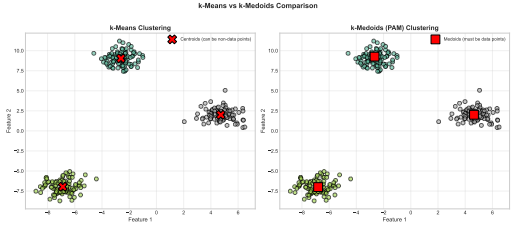
k-Means vs. k-Medoids

k-Means Advantages

+ Faster — $O(nkd)$ vs $O(n^2k)$ for PAM; better for large datasets
+ Centroids give compact cluster summaries even for continuous features

k-Medoids Advantages

– More robust to outliers — medoids are real data points, not pulled by extremes
– Works with any distance metric — Manhattan, cosine, or custom dissimilarity



k-Means is faster; k-Medoids is more robust — choose based on data quality

Euclidean and Manhattan Distance

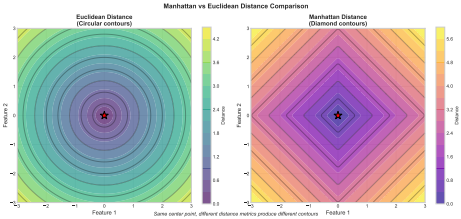
Distance Formulas

- Euclidean measures straight-line distance — sensitive to large differences in any dimension
- Manhattan sums absolute differences — more robust to outliers in individual features
- Choice of metric affects which clusters emerge — always consider your data's geometry

Definitions

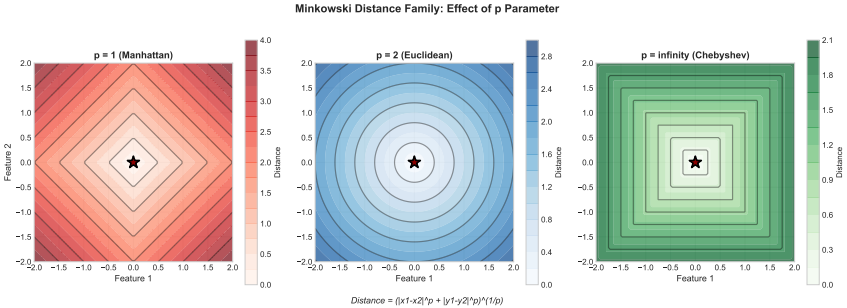
$$d_E(x, y) = \sqrt{\sum_{i=1}^{p} (x_i - y_i)^2}$$

$$d_M(x, y) = \sum_{i=1}^{p} |x_i - y_i|$$



Euclidean measures straight-line distance; Manhattan sums absolute differences along axes

Minkowski Distance Family



Key Observations:

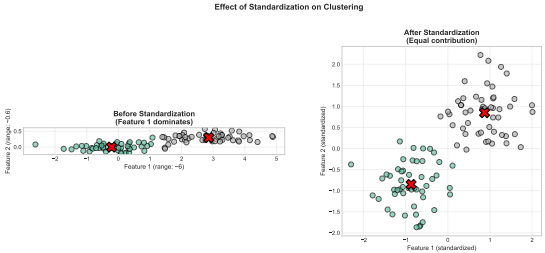
- Minkowski distance generalizes both: $d_p(x, y) = (\sum |x_i - y_i|^p)^{1/p}$
- $p = 1$ gives Manhattan, $p = 2$ gives Euclidean, $p \to \infty$ gives Chebyshev (max difference)
- Higher p emphasizes the single largest coordinate difference

The Minkowski parameter p controls the balance between Manhattan (p=1) and Chebyshev ($p \to \infty$)

Why Standardization Matters

The Scale Problem

- Variables on different scales (e.g., age in years vs. income in thousands) dominate distance calculations
- Standardization (z-score: subtract mean, divide by SD) puts all variables on equal footing
- Without standardization, the variable with the largest range controls the clustering
- Always standardize before clustering unless all variables share the same unit and scale



Unstandardized variables with large ranges dominate distance calculations — always standardize first

Distance Metrics in R

```r
# Example data
x <- matrix(c(1,2,4,5,2,3,6,7), ncol=2, byrow=TRUE)

# Euclidean (default)
d_euclidean <- dist(x, method="euclidean")

# Manhattan
d_manhattan <- dist(x, method="manhattan")

# Minkowski with p=3
d_minkowski <- dist(x, method="minkowski", p=3)

# Maximum (Chebyshev)
d_max <- dist(x, method="maximum")
```

```r
# Standardization
x_scaled <- scale(x)

# Clustering with different distances
km_euclidean <- kmeans(x, centers=2)
km_after_scaling <- kmeans(x_scaled, centers=2)

# PAM with Manhattan distance
library(cluster)
pam_manhattan <- pam(x, k=2, metric="manhattan")

# Custom distance matrix
custom_dist <- as.dist(my_distance_function(x))
hclust_result <- hclust(custom_dist)
```

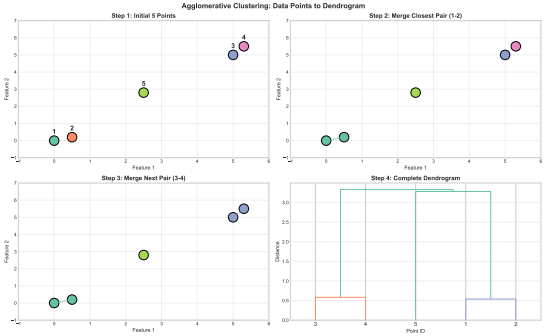
dist() supports multiple metrics; scale() standardizes data before clustering

Hierarchical Clustering

Agglomerative Algorithm

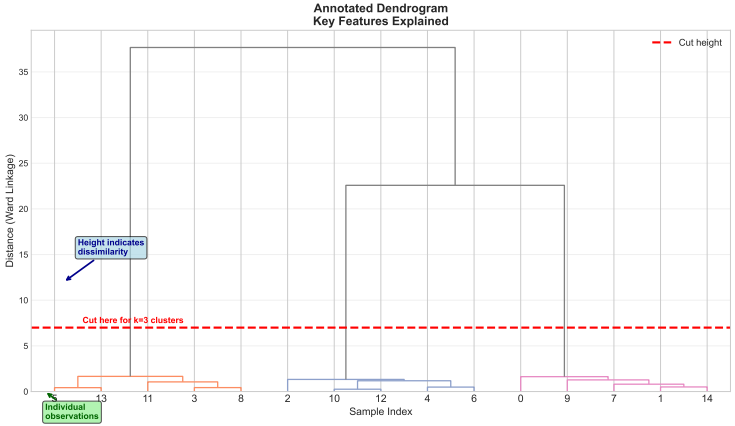
Bottom-Up Construction

- Start with n individual observations, each forming its own cluster
- Find the two closest clusters and merge them into one
- Update the distance matrix using the chosen linkage rule
- Repeat until all observations belong to a single cluster — record the merge history



Agglomerative clustering builds the tree bottom-up: merge the two closest clusters at each step

Reading Dendrograms



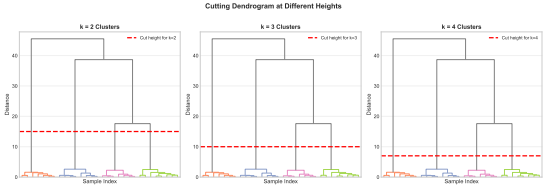
Key Observations:

- The y-axis shows the distance (or dissimilarity) at which clusters were merged
- Tall branches indicate well-separated clusters; short branches suggest similar groups
- Cut the dendrogram horizontally at a chosen height to obtain k clusters
- The structure is hierarchical: every split is nested within the one above it

Cutting Dendrograms to Form Clusters

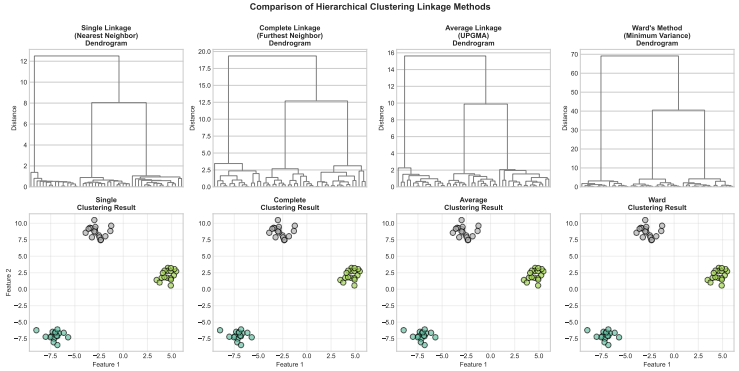
Cutting Strategies

- Cut at a fixed height h: all merges below h form separate clusters
- Cut to obtain exactly k clusters: choose the height that produces k groups
- In R: `cutree(hc, k=3)` or `cutree(hc, h=5)` — both approaches work
- The optimal cut balances cluster compactness (low WCSS) against interpretability



Cut the dendrogram at a height or specify k directly — both approaches give cluster assignments

Linkage Methods Compared



Key Observations:

- Single linkage (minimum distance): tends to produce elongated, chain-like clusters
- Complete linkage (maximum distance): produces compact, spherical clusters
- Average linkage (mean distance): a compromise between single and complete
- Ward's method (minimum variance increase): most popular — produces similarly-sized, compact clusters

Linkage choice determines cluster shape: Ward's for compact, single for chains, average for balance

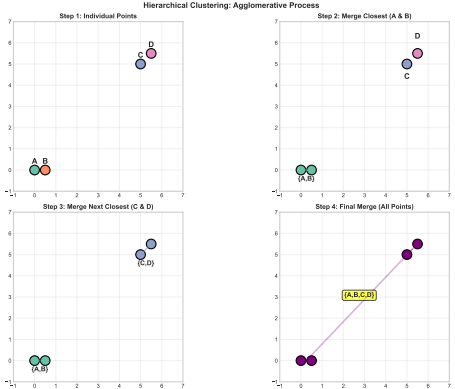
Hierarchical: Strengths and Limitations

Strengths

+ No need to specify k in advance — explore the dendrogram to decide
+ Produces a full hierarchy — reveals multi-scale structure in the data

Limitations

– $O(n^2)$ memory and $O(n^3)$ time — impractical for datasets larger than ∼5,000 observations
– Merges are irreversible — an early bad merge propagates through the entire tree



Hierarchical clustering reveals structure at all scales but is expensive for large datasets

Hierarchical Clustering in R

```r
# Generate example data
set.seed(123)
x <- rbind(
  matrix(rnorm(50, mean=0), ncol=2),
  matrix(rnorm(50, mean=3), ncol=2)
)

# Compute distance matrix
d <- dist(x, method="euclidean")

# Hierarchical clustering with different linkages
hc_single <- hclust(d, method="single")
hc_complete <- hclust(d, method="complete")
hc_average <- hclust(d, method="average")
hc_ward <- hclust(d, method="ward.D2")
```

```r
# Plot dendrogram
plot(hc_ward, main="Dendrogram␣(Ward's)",
     xlab="", sub="", cex=0.7)

# Cut tree to get k clusters
clusters <- cutree(hc_ward, k=2)

# Draw rectangles around clusters
rect.hclust(hc_ward, k=2, border="red")

# Visualize clusters in original space
plot(x, col=clusters, pch=19,
     main="Hierarchical␣Clustering␣Results")
```

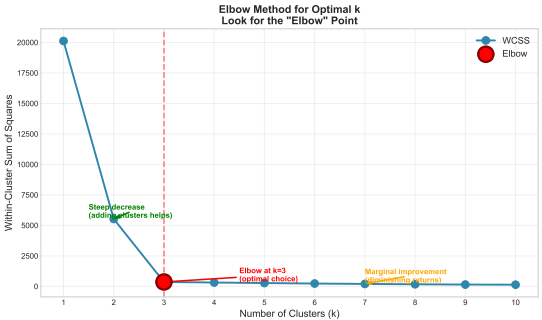
hclust() with Ward's linkage and cutree() for cluster extraction is the standard R workflow

Choosing the Number of Clusters

Elbow Method: WCSS

The Elbow Heuristic

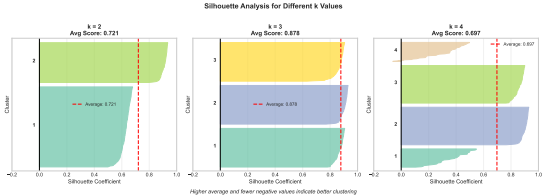
- Compute WCSS (Within-Cluster Sum of Squares) for $k = 1, 2, \ldots, K$
- Plot WCSS against k — the curve decreases as k increases
- The "elbow" is the point where adding more clusters yields diminishing returns
- Simple and widely used, but the elbow can be ambiguous — combine with other methods



The elbow in the WCSS plot marks where adding clusters gives diminishing returns

Measuring Cluster Quality

- Silhouette measures how well each point fits its assigned cluster vs. the nearest other cluster
- Values range from -1 (wrong cluster) through 0 (boundary) to $+1$ (well-matched)
- Average silhouette width across all points summarizes overall clustering quality
- Choose k that maximizes the average silhouette width



Silhouette width near +1 means well-clustered; near 0 means ambiguous; negative means misplaced

Silhouette Intuition

Interpreting Silhouette Values

- $s(i) \approx 1$: point is well inside its cluster — good assignment
- $s(i) \approx 0$: point is on the boundary between two clusters
- $s(i) < 0$: point may be in the wrong cluster — investigate reassignment

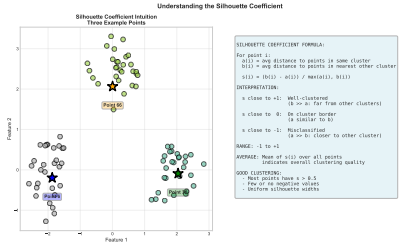
Silhouette Coefficient

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

where:

$a(i)$ = average distance to points in the same cluster (cohesion)

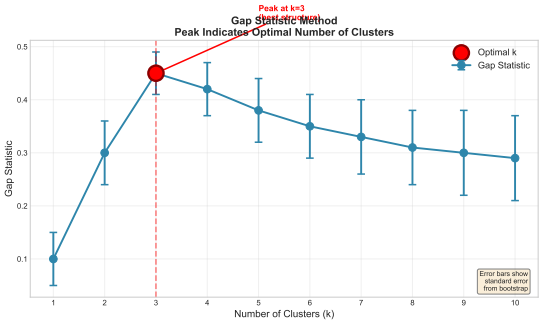
$b(i)$ = average distance to points in the nearest other cluster (separation)



The silhouette formula balances within-cluster cohesion against between-cluster separation

Statistical Test for k

- Compare observed WCSS to WCSS of random uniform reference data (no structure)
- $\text{Gap}(k) = E[\log(\text{WCSS}^*)] - \log(\text{WCSS})$ — larger gap means more structure than expected by chance
- Choose the smallest k such that $\text{Gap}(k) \geq \text{Gap}(k{+}1) - \text{SE}(k{+}1)$
- Provides a statistical test (not just a visual heuristic) — more principled than the elbow



Gap statistic provides a statistical test — not just a visual heuristic — for choosing k

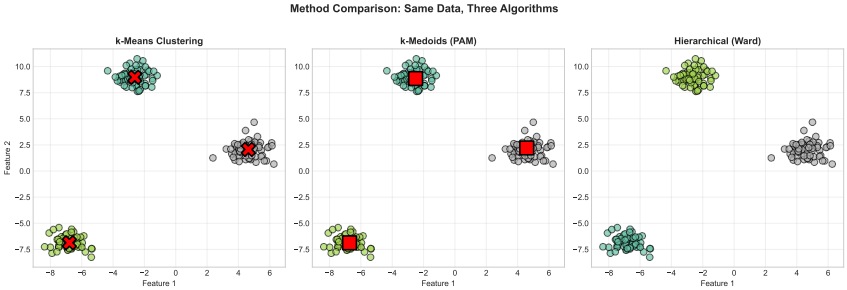
Cluster Validation in R

```r
library(cluster)
library(factoextra)

# Compute silhouette for k=3
km <- kmeans(x, centers=3, nstart=25)
sil <- silhouette(km$cluster, dist(x))

# Average silhouette width
mean(sil[,3])

# Plot silhouette
plot(sil, col=1:3, border=NA,
     main="Silhouette Plot (k=3)")

# Or with factoextra
fviz_silhouette(sil)
```

```r
# Compare different k values
k_values <- 2:8
avg_sil <- sapply(k_values, function(k) {
  km <- kmeans(x, centers=k, nstart=25)
  sil <- silhouette(km$cluster, dist(x))
  mean(sil[,3])
})

# Plot average silhouette vs k
plot(k_values, avg_sil, type="b",
     xlab="Number of Clusters",
     ylab="Average Silhouette Width",
     main="Silhouette Method")
abline(v=k_values[which.max(avg_sil)],
       lty=2, col="red")
```

Combine silhouette analysis with elbow and gap statistic for robust cluster validation

Comparing k-Selection Methods



Key Observations:

- Elbow method is fast and intuitive but the elbow is often subjective
- Silhouette analysis provides a clear optimum (max average width) and per-point diagnostics
- Gap statistic offers statistical rigor but is computationally expensive (bootstrap resampling)

No single method is perfect — use multiple k-selection criteria and let them vote

Comparison and Best Practices

Complete Algorithm Comparison

	k-Means	**k-Medoids**	**Single Linkage**	**Ward's**
Centers	Means	Medoids	N/A	N/A
k required	Yes	Yes	No	No
Outliers	Sensitive	Robust	Sensitive	Moderate
Shape	Spherical	Flexible	Elongated	Spherical
Speed	Fast	Slow	Medium	Medium
Best for	Large data, speed	Outliers, interpret	Chains	Compact

This comparison matrix is your quick reference — match method to data characteristics

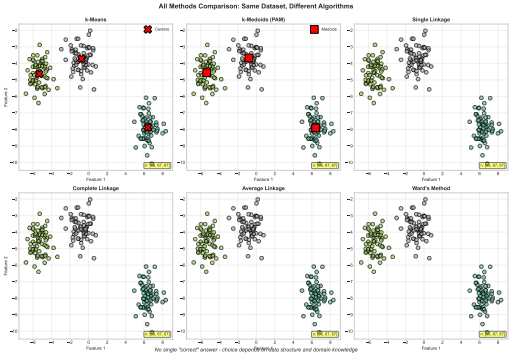
When to Use Which Method

Partitional Methods

+ k-Means/k-Medoids: Fast, scalable, best when k is known and clusters are roughly spherical
+ Use PAM when outliers are present; always combine statistical k with domain knowledge

Hierarchical Methods

− Reveals multi-scale structure via dendrograms — best for exploration when $n < 5{,}000$
− Ward's is the most popular linkage; statistical k may differ from the practical choice — let the data inform but let the domain decide

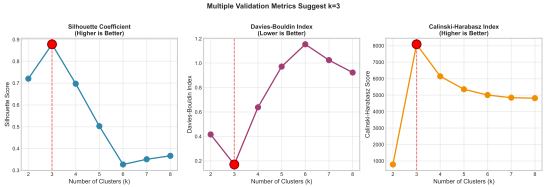


Start with k-Means, try hierarchical for exploration, use PAM if outliers are a concern

Summary and Key Takeaways

What You Have Learned

- k-Means is the default starting point: fast, simple, works well for spherical clusters with `nstart` $\geq$ 25
- k-Medoids (PAM) is preferred when outliers are present or non-Euclidean distances are needed
- Hierarchical clustering with Ward's linkage reveals nested structure — always visualize the dendrogram
- Validate k with multiple methods: elbow, silhouette, gap statistic, and domain knowledge combined



Clustering is exploratory — validate with domain knowledge. Next: density-based methods (DBSCAN) and model-based clustering (GMM)