Clustering Algorithms in Machine Learning

K-Means, Affinity Propagation, Mean Shift, Spectral Clustering,

Hierarchical, DBSCAN, OPTICS, BIRCH

Introduction to Clustering

- Clustering is an unsupervised learning method.
- Groups similar data points into clusters.
- Used in market segmentation, anomaly detection, image processing.

Applications of Clustering

- Market Segmentation
- Social Network Analysis
- Image Compression
- Document Clustering
- Fraud Detection
- Bioinformatics

Types of Clustering Algorithms

1. Partitioning Methods (K-Means)
2. Propagation-Based (Affinity Propagation)
3. Centroid-Based (Mean Shift)
4. Graph-Based (Spectral Clustering)
5. Hierarchical Methods
6. Density-Based (DBSCAN, OPTICS)
7. Tree-Based (BIRCH)

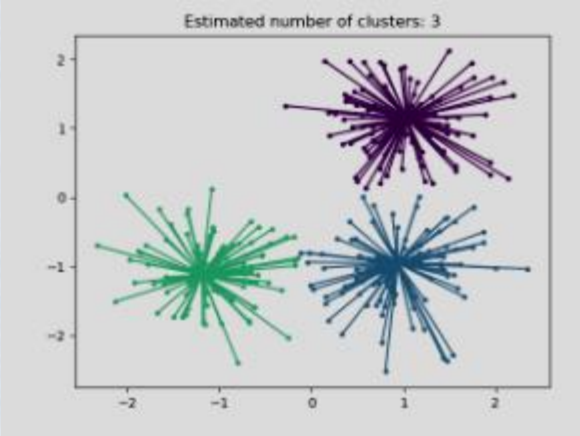
K-Means

- Partitions data into K clusters.
- Python:
- from sklearn.cluster import KMeans
- kmeans = KMeans(n_clusters=3).fit(X)
- labels = kmeans.labels_

- Pros: Simple, fast, scalable
- Cons: Need K, sensitive to noise

Affinity Propagation

- Message-passing between points, no need to choose K.
- Python:
- from sklearn.cluster import AffinityPropagation
- clust = AffinityPropagation().fit(X)
- labels = clust.labels_

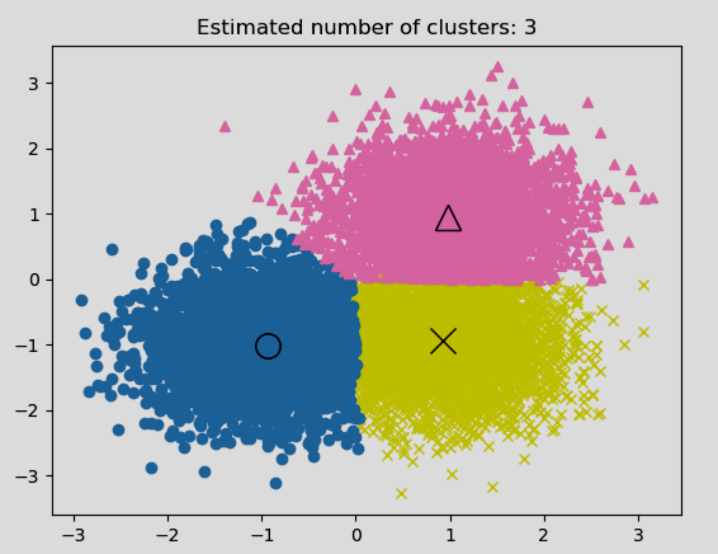


- Pros: No need K, finds exemplars
- Cons: High memory use, slower

Mean Shift

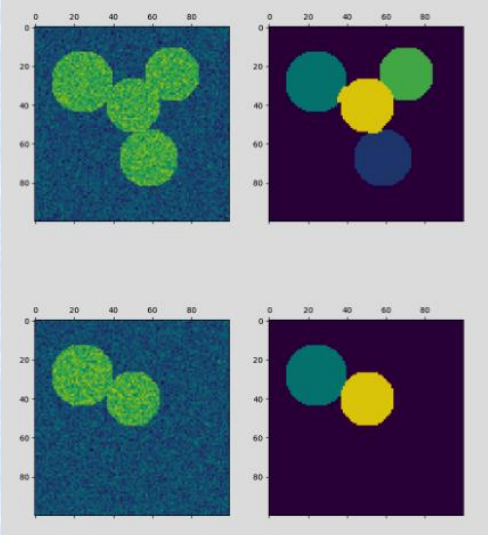
- Shifts centroids towards high-density regions.
- Python:
- from sklearn.cluster import MeanShift
- ms = MeanShift().fit(X)
- labels = ms.labels_

- Pros: No need K, flexible shapes
- Cons: Expensive on large data



Spectral Clustering

- Uses graph Laplacian and eigenvalues.
- Python:
- from sklearn.cluster import SpectralClustering
- sc = SpectralClustering(n_clusters=3).fit(X)
- labels = sc.labels_

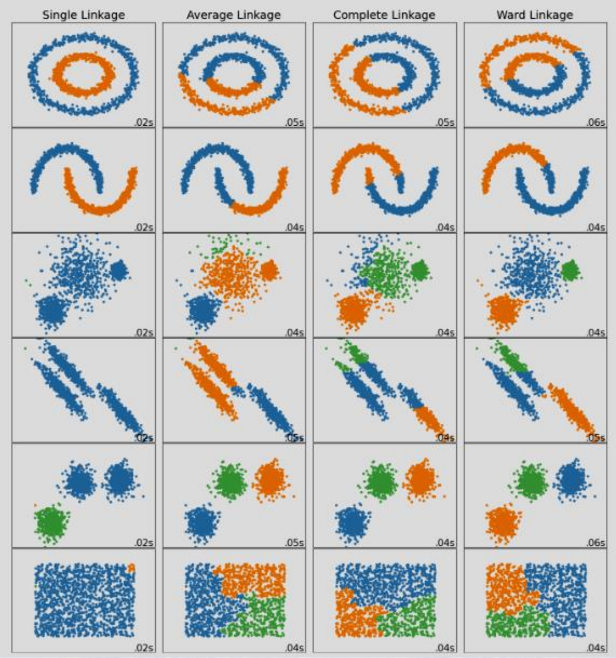


- Pros: Works on non-convex clusters
- Cons: Expensive on large datasets

Hierarchical Clustering

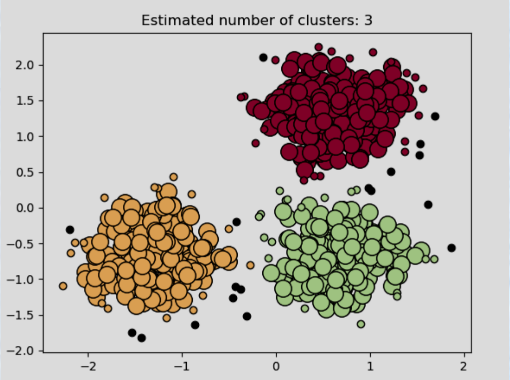
- Agglomerative/Divisive approaches with dendrogram.
- Python:
- from scipy.cluster.hierarchy import linkage, dendrogram
- Z = linkage(X, 'ward')

- Pros: No need K, interpretable
- Cons: Expensive for big data



DBSCAN

- Groups points with many neighbors, detects noise.
- Python:
- from sklearn.cluster import DBSCAN
- db = DBSCAN(eps=0.5, min_samples=5).fit(X)
- labels = db.labels_

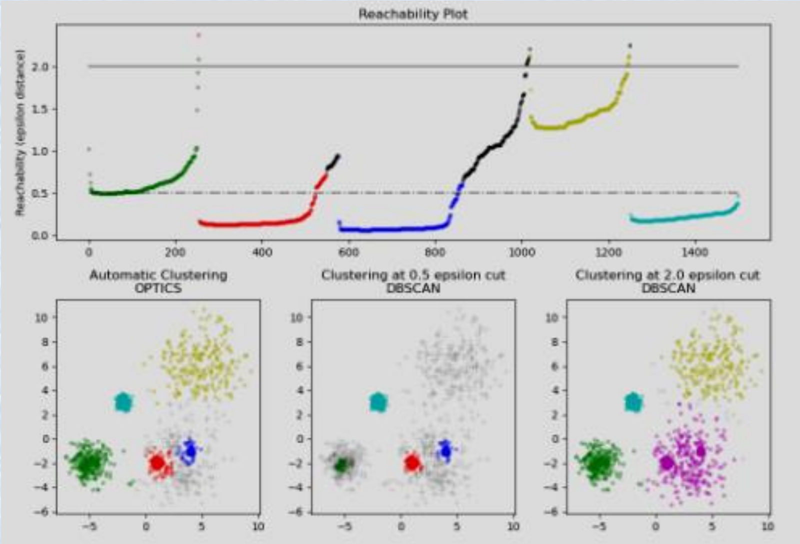


- Pros: Arbitrary shapes, noise handling
- Cons: Struggles with varying density

OPTICS

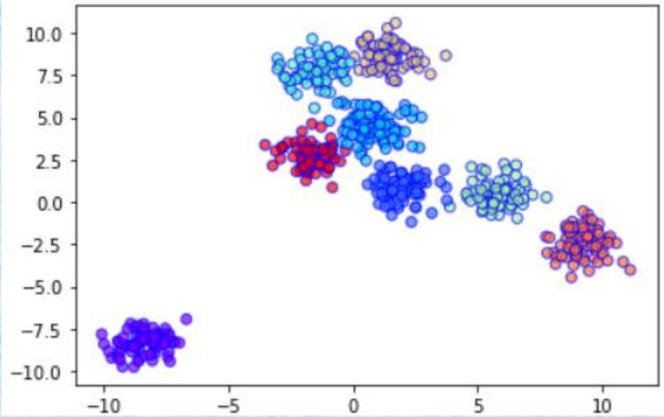
- Orders points to handle varying density.
- Python:
- from sklearn.cluster import OPTICS
- opt = OPTICS(min_samples=5).fit(X)
- labels = opt.labels_

- Pros: Handles varying density
- Cons: Slower than DBSCAN



BIRCH

- Tree structure (CF Tree) for large datasets.
- Python:
- from sklearn.cluster import Birch
- brc = Birch(n_clusters=3).fit(X)
- labels = brc.labels_



- Pros: Scales to large data
- Cons: Works best with spherical clusters

Comparison of Clustering Algorithms

- K-Means: Fast, needs K, sensitive to noise
- Affinity Propagation: No K, slower, memory heavy
- Mean Shift: Flexible, expensive
- Spectral: Good for non-convex, expensive
- Hierarchical: Dendrogram, expensive for large data
- DBSCAN: Arbitrary shapes, struggles with varying density
- OPTICS: Handles varying density, slower
- BIRCH: Scales well, works best on spherical clusters

Conclusion

- Clustering groups data without labels.
- Many algorithms exist, each with trade-offs.
- Choice depends on data size, shape, density, and noise.
- No single best algorithm for all problems.