CLUSTERING IN ML

CLUSTERING IN ML

- Clustering is an unsupervised machine learning technique.
- It used to group of data points into clusters based on their similarities or patterns
- Clustering works with unlabeled dataset

COMMON CLUSTERING METHODS

- **Centroid-Based Clustering**: Groups data points around centroids (e.g., K-Means). It is efficient but requires predefining the number of clusters and struggles with irregularly shaped clusters.
- **Density-Based Clustering**: Identifies clusters as dense regions separated by sparse areas (e.g., DBSCAN). It handles noise and irregular shapes well but may struggle with varying densities.
- **Hierarchical Clustering**: Builds a tree-like structure (dendrogram) to represent data relationships. It can be agglomerative (bottom-up) or divisive (top-down) and does not require predefining the number of clusters.
- **Distribution-Based Clustering**: Assumes data points are generated from probability distributions (e.g., Gaussian Mixture Models). It is flexible but computationally expensive.
- **Fuzzy Clustering**: Allows data points to belong to multiple clusters with membership probabilities, making it suitable for overlapping data.

APPLICATION OF CLUSTERING

Market Segmentation: Grouping customers for targeted marketing.

Anomaly Detection: Identifying outliers in datasets, such as fraudulent transactions.

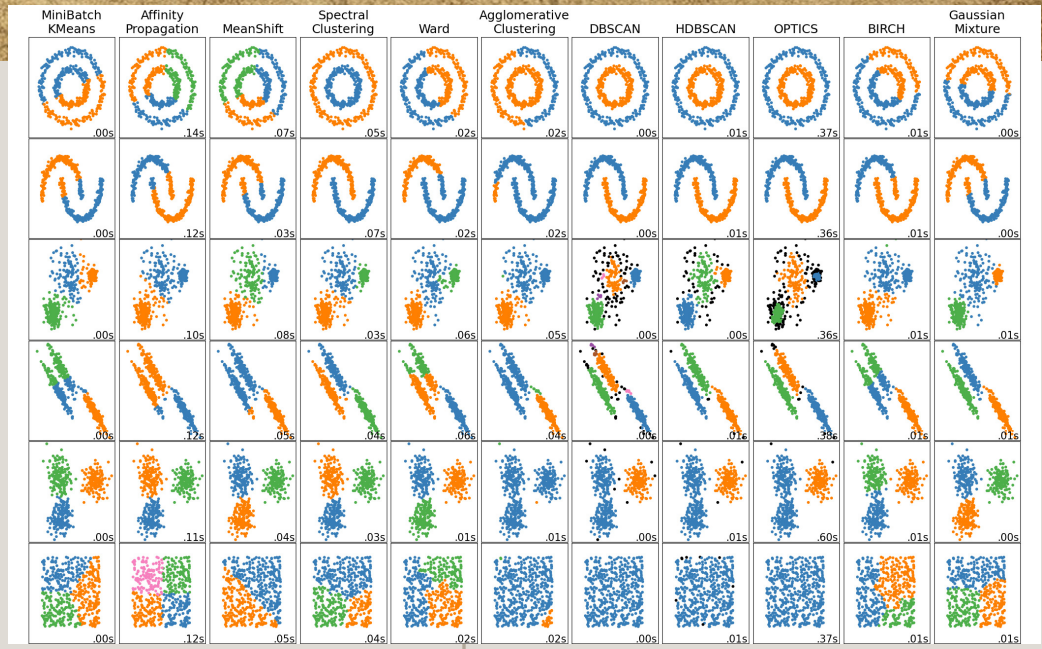
Medical Imaging: Segmenting diagnostic images to detect diseased areas.

Social Network Analysis: Understanding user behavior and recommending connections.

Data Compression: Reducing dataset complexity by replacing features with cluster IDs

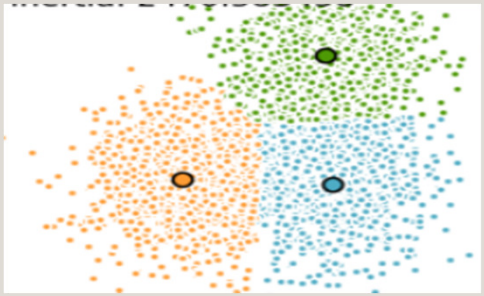
CLUSTERING ALGORITHMS

- K-Means
- Affinity propagation
- Agglomerative clustering
- DBSCAN
- HDBSCAN
- OPTICS
- BIRCH



K-MEANS CLUSTERING

K-Means clustering is a popular **unsupervised machine learning algorithm** used to group data into clusters based on similarity. It is widely applied in fields like customer segmentation, image compression, and pattern recognition.



Choose the Number of Clusters (K):

Decide how many clusters you want to divide the data into.

Initialize Centroids:

Randomly select K data points as the initial cluster centroids (centers).

Assign Data Points to Clusters:

For each data point, calculate its distance to each centroid.

Assign the data point to the cluster with the nearest centroid.

Update Centroids:

Recalculate the centroid of each cluster by taking the mean of all data points assigned to it.

Repeat:

Repeat steps 3 and 4 until the centroids no longer change significantly or a maximum number of iterations is reached.

K-MEANS CLUSTERING

Distance Metric: Typically uses Euclidean distance to measure similarity.

Output: Divides the dataset into K clusters, where each cluster has a centroid representing its center.

Unsupervised: Works without labeled data.

Advantages:

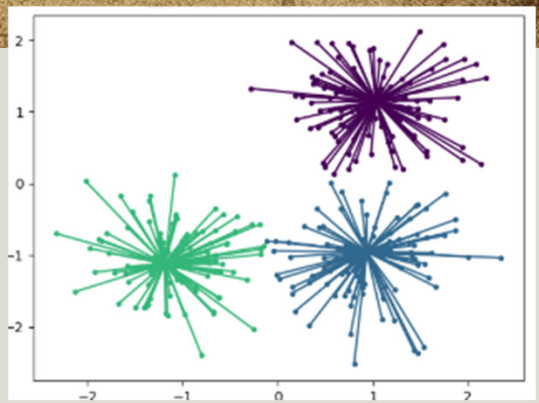
- Simple and easy to implement.
- Scales well to large datasets.
- Works well when clusters are spherical and evenly distributed.

Applications:

- Market segmentation.
- Image compression.
- Document clustering.
- Anomaly detection.

AFFINITY PROPAGATION

Affinity Propagation is a clustering algorithm that groups data points based on their similarity, using a unique approach called "message passing."



Key Features:

Unlike K-Means, you don't need to specify the number of clusters beforehand. The algorithm determines the optimal number of clusters automatically.

Message Passing: Data points exchange two types of messages:

- **Responsibility**: Indicates how well-suited a point is to be the exemplar (cluster center) for another point.
- **Availability**: Reflects how appropriate it is for a point to choose another as its exemplar.

Exemplars: Instead of centroids, the algorithm identifies actual data points as cluster centers (exemplars).

AFFINITY PROPAGATION

Algorithm Steps:

- **Similarity Computation**: Calculate the similarity matrix which quantifies the similarity between pairs of data points.
- **Responsibility Update**: Initialize and update the responsibility matrix iteratively.
- **Availability Update**: Initialize and update the availability matrix iteratively.
- **Net Responsibility Calculation**: Calculate the net responsibility for each data point by summing its responsibility and availability.
- **Exemplar Selection**: Identify exemplars as data points with high net responsibility.
- **Cluster Assignment**: Assign each data point to the nearest exemplar to form clusters.

Advantages:

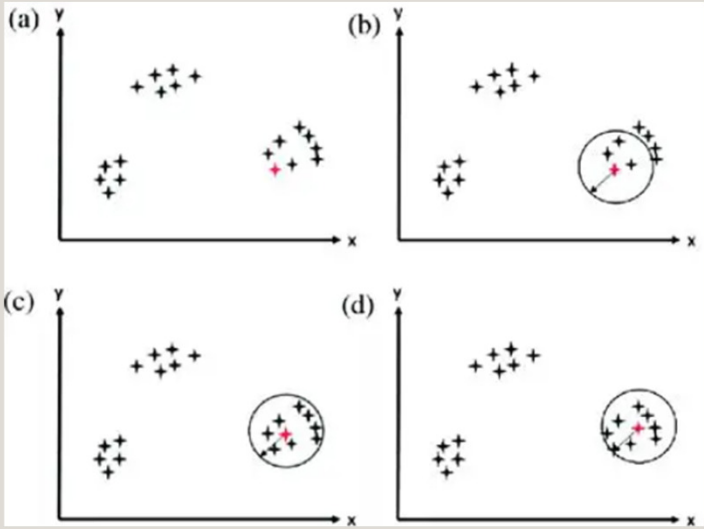
- Automatically determines the number of clusters.
- Works well with non-convex clusters.
- Identifies actual data points as exemplars, making results interpretable.

MEAN SHIFT

The **Mean Shift clustering algorithm** is a powerful, non-parametric, density-based clustering technique used to identify clusters in a dataset. Here's a concise overview:

Key Features

- **Non-Parametric**: Unlike K-Means, it does not require specifying the number of clusters beforehand.
- **Density-Based**: It identifies clusters by locating the modes (peaks) of the data's density distribution.
- **Arbitrary Shapes**: It works well with clusters of arbitrary shapes, making it versatile for complex datasets.



MEAN SHIFT

Advantages

- No need to predefine the number of clusters.
- Handles non-linear and arbitrarily shaped clusters.
- Robust to outliers.

Applications

- Image segmentation.
- Object tracking in computer vision.
- Anomaly detection.
- Customer segmentation in marketing.

```python
from sklearn.cluster import MeanShift
import numpy as np

# Sample data
data = np.array([[1, 2], [1, 4], [1, 0],
                 [10, 2], [10, 4], [10, 0]])

# Mean Shift clustering
mean_shift = MeanShift(bandwidth=2)
mean_shift.fit(data)
```

HIERARCHICAL CLUSTERING

Hierarchical clustering is a method of cluster analysis that seeks to build a hierarchy of clusters. It is widely used in data mining and statistics to group similar data points based on their similarity, creating a tree-like structure known as a dendrogram.

Types of Hierarchical Clustering

- **Agglomerative Clustering**: This is a "bottom-up" approach where each data point starts as its own cluster, and pairs of clusters are merged as one moves up the hierarchy. It does not require a predefined number of clusters.
- **Divisive Clustering**: This is a "top-down" approach where all data points start in one cluster, and splits are performed recursively as one moves down the hierarchy. It also does not require a predefined number of clusters.

HIERARCHICAL CLUSTERING

Workflow for Hierarchical Agglomerative Clustering

- **Start with individual points**: Each data point is its own cluster.
- **Calculate distances between clusters**: Calculate the distance between every pair of clusters.
- **Merge the closest clusters**: Identify the two clusters with the smallest distance and merge them into a single cluster.
- **Update distance matrix**: Recalculate the distances between the new cluster and the remaining clusters.
- **Repeat steps 3 and 4**: Continue merging the closest clusters and updating the distance matrix until only one cluster is left.
- **Create a dendrogram**: Visualize the merging of clusters using a dendrogram

```
from sklearn.cluster import AgglomerativeClustering
import numpy as np

X = np.array([[1, 2], [1, 4], [1, 0], [4, 2], [4, 4], [4, 0]])
clustering = AgglomerativeClustering(n_clusters=2).fit(X)
print(clustering.labels_)
```

HIERARCHICAL CLUSTERING

Workflow for Hierarchical Divisive Clustering

- **Start with all data points in one cluster**: Treat the entire dataset as a single large cluster.
- **Split the cluster**: Divide the cluster into two smaller clusters.
- **Repeat the process**: For each of the new clusters, repeat the splitting process until every data point is its own cluster or the stopping condition is met

Advantages:

- Can create more complex shaped clusters.
- Does not require a predefined number of clusters.
- Provides a structured way to explore data similarity

Disadvantages:

- Heavily driven by heuristics, requiring manual intervention.
- Computationally intensive due to the need to calculate distances between data samples/subclusters.
- Difficult to visually analyze dendrograms with a large number of data samples

DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a powerful and widely used clustering algorithm in machine learning. It is particularly effective for identifying clusters of varying shapes and sizes in datasets with noise.

- **Density-Based Clustering**: Groups points that are closely packed together (high density) and separates regions of lower density.
- **Non-Parametric**: Does not require specifying the number of clusters beforehand, unlike algorithms like K-Means.
- **Outlier Detection**: Marks points in low-density regions as noise or outliers.

How DBSCAN Works

1. Parameters:
- Epsilon (ε): The maximum distance between two points to be considered neighbors.
- MinPts: The minimum number of points required to form a dense region (a cluster).

2. Steps:
- Identify core points: Points with at least MinPts neighbors within a distance of ε.
- Expand clusters: Connect core points and their neighbors to form clusters.
- Label points that are not part of any cluster as noise.

DBSCAN

Advantages

- Handles clusters of arbitrary shapes.
- Robust to noise and outliers.
- No need to predefine the number of clusters.

```python
from sklearn.cluster import DBSCAN
import numpy as np

# Example data
data = np.array([[1, 2], [2, 2], [2, 3], [8, 7], [8, 8], [25, 80]])

# DBSCAN model
dbscan = DBSCAN(eps=3, min_samples=2)
clusters = dbscan.fit_predict(data)

print("Cluster labels:", clusters)
```

OPTICS

OPTICS (Ordering Points To Identify the Clustering Structure) is a **density-based clustering algorithm** that is particularly useful for identifying clusters of varying densities and shapes in large, high-dimensional datasets. It is an extension of the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm, addressing some of its limitations.

- **Core Distance**: The minimum radius required to classify a point as a core point. If a point is not a core point, its core distance is undefined.
- **Reachability Distance**: The maximum of the core distance of a point and the distance between two points. It is undefined if the second point is not a core point.

Algorithm Steps:

- **Define Parameters**: Set the density threshold parameter (Eps) and the minimum number of points (MinPts).
- **Calculate Distances**: For each point, calculate the distance to its k-nearest neighbors.
- **Compute Reachability Distances**: Starting with an arbitrary point, calculate the reachability distance of each point based on the density of its neighbors.
- **Order Points**: Order the points based on their reachability distance and create the reachability plot.
- **Extract Clusters**: Group points that are close to each other and have similar reachability distances.

OPTICS

Advantages

Flexibility: Unlike DBSCAN, OPTICS does not require the number of clusters to be set in advance. It produces a reachability plot, allowing for more flexible cluster selection

Handling Varying Densities: OPTICS can handle clusters of different densities and shapes, making it more versatile than DBSCAN

Memory and Computational Cost: OPTICS requires more memory and computational power due to the use of a priority queue and more complex nearest neighbor queries

Manual Interpretation: Extracting clusters from the reachability plot requires manual interpretation and decision-making

```python
from sklearn.cluster import OPTICS
from sklearn import datasets
import numpy as np

# Load the iris dataset
iris = datasets.load_iris()
X = iris.data

# Fit the OPTICS model
clustering = OPTICS(min_samples=20, xi=.05, min_cluster_size=.05)
clustering.fit(X)

# Extract the clusters
labels = clustering.labels_

# Print the cluster labels
print("Cluster Labels:", labels)
```

OPTICS is a powerful clustering algorithm that extends the capabilities of DBSCAN by handling varying densities and shapes in datasets. It provides a flexible and robust approach to clustering, especially for large and complex datasets

BIRTH

BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) is an unsupervised data mining algorithm designed for hierarchical clustering of large datasets. It is particularly effective for clustering large datasets by creating a compact summary of the data, which can then be clustered using other algorithms like K-means or Agglomerative Clustering

Algorithm Stages

Building the CF Tree: The algorithm compresses the data into CF nodes, which are then organized into a CF tree. This tree structure allows efficient clustering by summarizing the data into smaller, manageable sub-clusters.

Global Clustering: An existing clustering algorithm is applied to the leaves of the CF tree to combine the sub-clusters into final clusters

BIRTH

Advantages:

Efficiency: BIRCH is efficient in terms of memory and time, making it suitable for large datasets.

Incremental Clustering: It can incrementally and dynamically cluster incoming data points

Limitations:

Metric Attributes Only: BIRCH can only process metric attributes, meaning it cannot handle categorical data

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import Birch

# Generating 600 samples using make_blobs
dataset, clusters = make_blobs(n_samples=600, centers=8, cluster_std=0.75,
random_state=0)

# Creating the BIRCH clustering model
model = Birch(branching_factor=50, n_clusters=None, threshold=1.5)

# Fit the data (Training)
model.fit(dataset)

# Predict the same data
pred = model.predict(dataset)

# Creating a scatter plot
plt.scatter(dataset[:, 0], dataset[:, 1], c=pred, cmap='rainbow', alpha=0.7,
edgecolors='b')
plt.show()
```