Clustering Tutorial – A Hands-on Approach

Introduction to Clustering

Clustering is an **unsupervised machine learning** technique used to **group similar data points** based on their features. Unlike classification, clustering doesn't have predefined labels but finds patterns in the data.

Common Applications of Clustering

- **Customer Segmentation:** Grouping customers based on purchasing behavior.
- **Anomaly Detection:** Detecting fraudulent transactions.
- **Document Clustering:** Organizing similar documents or news articles.
- **Image Segmentation:** Grouping similar regions in images.

---

1. Types of Clustering Algorithms

1.1 K-Means Clustering

- Most widely used clustering algorithm.
- Assigns data points to **K** clusters based on **centroid distance**.
- Uses **Euclidean distance** for similarity measurement.

1.2 Hierarchical Clustering

- Creates a **tree of clusters** (dendrogram).
- Two types:
    - **Agglomerative (Bottom-Up):** Starts with individual points and merges them.
    - **Divisive (Top-Down):** Starts with one large cluster and splits it.

1.3 DBSCAN (Density-Based Spatial Clustering)

- Groups points **based on density**.
- Can **detect noise and outliers**.
- Works well with **irregularly shaped clusters**.

1.4 Gaussian Mixture Model (GMM)

- Uses **probability distributions** to model clusters.
- More flexible than K-Means.
- Assigns a **probability score** to each data point belonging to a cluster.

---

2. Hands-on Implementation in Python

Let's implement **K-Means, Hierarchical Clustering, and DBSCAN** using Python.

2.1 Install Required Libraries

```
!pip install numpy pandas matplotlib seaborn scikit-learn
```

2.2 Import Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.datasets import make_blobs
from scipy.cluster.hierarchy import dendrogram, linkage
```

---

3. Generating a Synthetic Dataset

We will create **3 clusters** of data using `make_blobs()`:

```
# Generate synthetic dataset
X, y = make_blobs(n_samples=300, centers=3, cluster_std=1.0,
random_state=42)

# Scatter plot of data
plt.scatter(X[:, 0], X[:, 1], s=50, alpha=0.6)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Generated Data with 3 Clusters")
plt.show()
```

---

4. K-Means Clustering

4.1 Applying K-Means

```
# Apply K-Means with K=3
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
y_kmeans = kmeans.fit_predict(X)

# Plot Clusters
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, cmap='viridis', s=50, alpha=0.6)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
s=200, c='red', marker="X", label="Centroids")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("K-Means Clustering")
plt.legend()
```

```
plt.show()
```

4.2 Choosing Optimal K using the Elbow Method

```
inertia = []
K_range = range(1, 10)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertia.append(kmeans.inertia_)

plt.plot(K_range, inertia, marker='o')
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia (Within-cluster Sum of Squares)")
plt.title("Elbow Method for Optimal K")
plt.show()
```

- The **optimal K** is at the "elbow" where the inertia decreases sharply.

---

5. Hierarchical Clustering

5.1 Dendrogram Visualization

```
# Create linkage matrix
linked = linkage(X, method='ward')

plt.figure(figsize=(10, 5))
dendrogram(linked)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Data Points")
plt.ylabel("Distance")
plt.show()
```

- The **best number of clusters** is determined by cutting the dendrogram at the longest vertical line without crossing a horizontal line.

5.2 Applying Agglomerative Clustering

```
# Apply Agglomerative Clustering with 3 clusters
agglo = AgglomerativeClustering(n_clusters=3)
y_agglo = agglo.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_agglo, cmap='coolwarm', s=50, alpha=0.6)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Agglomerative Clustering")
```

```
plt.show()
```

---

6. DBSCAN (Density-Based Clustering)

6.1 Applying DBSCAN

```
# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
y_dbscan = dbscan.fit_predict(X)

plt.scatter(X[:, 0], X[:, 1], c=y_dbscan, cmap='Set1', s=50, alpha=0.6)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("DBSCAN Clustering (Outlier Detection)")
plt.show()
```

- **Outliers** are labeled as `-1`.

---

7. Comparing Clustering Techniques

Algorithm	Strengths	Weaknesses
K-Means	Fast, simple, efficient	Sensitive to outliers and cluster shape
Hierarchical	No need to specify K	Computationally expensive for large datasets
DBSCAN	Detects noise, finds arbitrarily shaped clusters	Struggles with varying density clusters

---

8. Summary

- **K-Means** is great for **well-separated clusters**.
- **Hierarchical Clustering** provides a **dendrogram for visualization**.
- **DBSCAN** is useful for **density-based clustering** and detecting **outliers**.