DAY 22 **[19 - JULY - 2025]**

UNSUPERVISED MACHINE LEARNING

Clustering techniques: k-means, hierarchical clustering, DBSCAN

Introduction to Clustering Techniques

Clustering is an unsupervised machine learning technique used to group similar data points into clusters based on their features, without prior knowledge of class labels. This document introduces three popular clustering algorithms: k-Means, Hierarchical Clustering, and DBSCAN, followed by their Python implementations using scikit-learn.

k-Means Clustering

k-Means is a centroid-based clustering algorithm that partitions data into k predefined clusters. It iteratively assigns data points to the nearest cluster centroid and updates the centroids to minimize the within-cluster variance. Key characteristics:

- **Simple and fast**: Suitable for large datasets with well-separated clusters.
- **Requires predefined k**: The number of clusters must be specified in advance.
- **Sensitive to initialization**: Results depend on initial centroid placement.

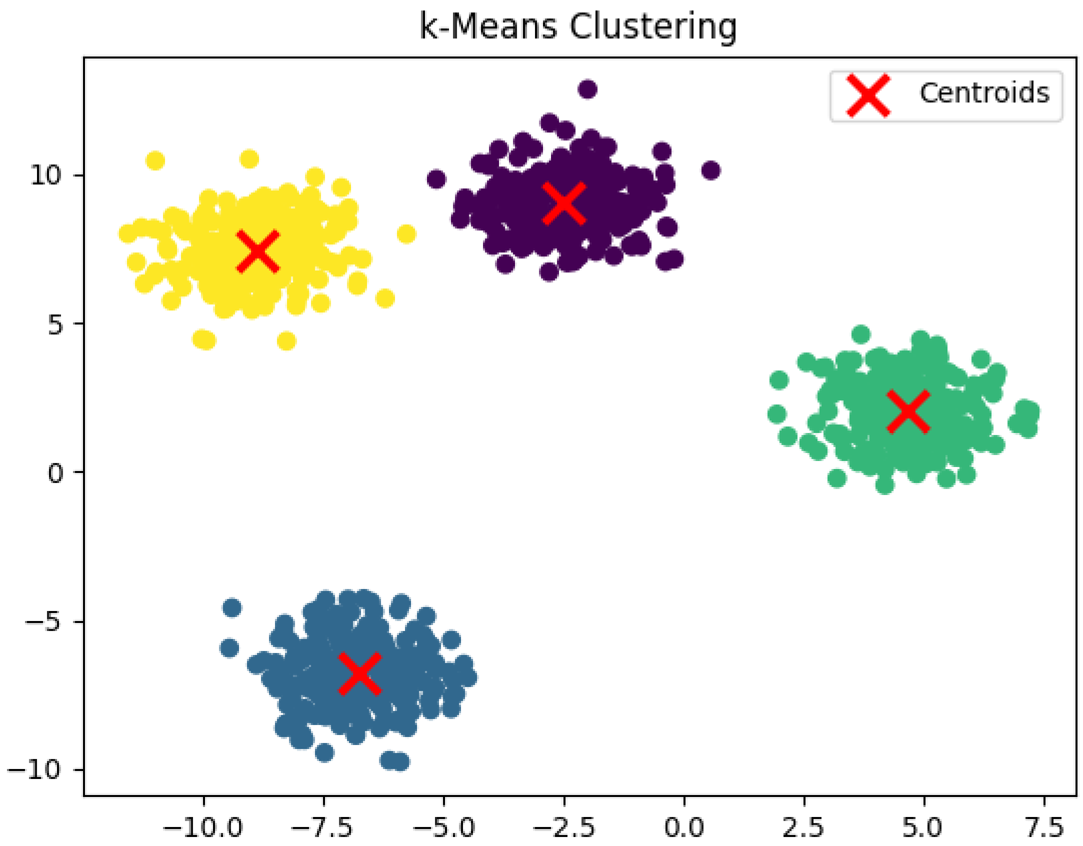
k-Means Clustering Implementation

```
# Import necessary libraries
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic dataset
X, _ = make_blobs(n_samples=1000, centers=4, random_state=42)

# k-Means Clustering
def kmeans_clustering(X, n_clusters=4):
    # Initialize k-Means
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)

    # Fit and predict clusters
    labels = kmeans.fit_predict(X)

    # Calculate silhouette score
    silhouette = silhouette_score(X, labels)

    print(f"\nk-Means Clustering (k={n_clusters}):")
    print(f"Silhouette Score: {silhouette:.4f}")

    # Visualize results
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
    plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,
1],
               c='red', marker='x', s=200, linewidths=3, label='Centroids')
```

```
    plt.title('k-Means Clustering')
    plt.legend()
    plt.show()

    return kmeans, labels

# Run k-Means
if __name__ == "__main__":
    kmeans_model, kmeans_labels = kmeans_clustering(X)
```

Output:

```
k-Means Clustering (k=4):
Silhouette Score: 0.7916
```



Hierarchical Clustering

Hierarchical Clustering builds a hierarchy of clusters either by merging smaller clusters (agglomerative) or splitting larger ones (divisive). Agglomerative is more common, using a linkage criterion (e.g., ward, single, complete) to determine cluster proximity. Key characteristics:

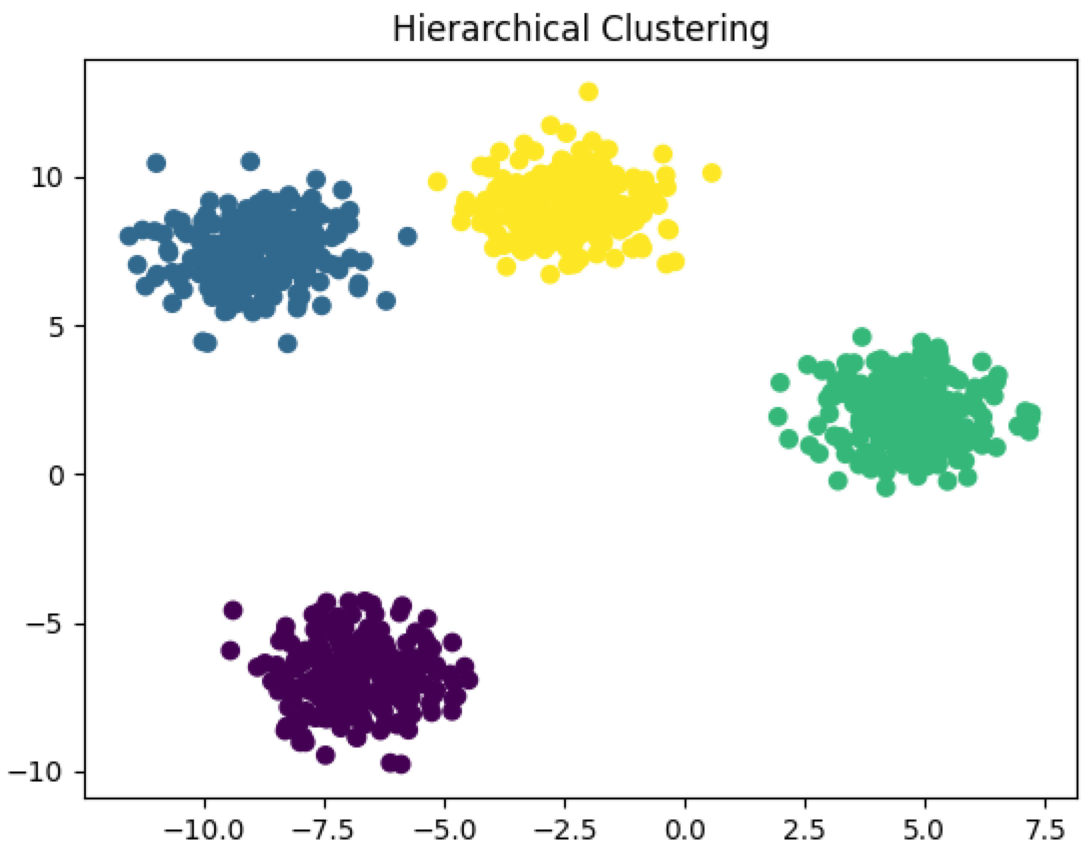
- **No need for predefined k**: Produces a dendrogram showing cluster relationships.
- **Computationally intensive**: Less efficient for large datasets.
- **Flexible linkage options**: Allows different distance metrics and merging strategies.

Hierarchical Clustering Implementation

```
# Import necessary libraries
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Generate synthetic dataset
X, _ = make_blobs(n_samples=1000, centers=4, random_state=42)

# Hierarchical Clustering
def hierarchical_clustering(X, n_clusters=4):
    # Initialize Agglomerative Clustering
    hc = AgglomerativeClustering(n_clusters=n_clusters, linkage='ward')

    # Fit and predict clusters
    labels = hc.fit_predict(X)

    # Calculate silhouette score
    silhouette = silhouette_score(X, labels)

    print(f"\nHierarchical Clustering (k={n_clusters}):")
    print(f"Silhouette Score: {silhouette:.4f}")

    # Visualize results
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
    plt.title('Hierarchical Clustering')
    plt.show()

    # Plot dendrogram
    Z = linkage(X, method='ward')
    plt.figure(figsize=(10, 5))
    dendrogram(Z)
    plt.title('Dendrogram for Hierarchical Clustering')
    plt.xlabel('Sample Index')
    plt.ylabel('Distance')
    plt.show()

    return hc, labels

# Run Hierarchical Clustering
if __name__ == "__main__":
    hc_model, hc_labels = hierarchical_clustering(X)
```

Output:

```
Hierarchical Clustering (k=4):
Silhouette Score: 0.7916
```



DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN groups data points based on density, identifying clusters as regions of high density separated by low-density areas. It marks outliers as noise. Key characteristics:

- **Handles arbitrary shapes**: Can find non-spherical clusters.
- **No need to specify k**: Automatically determines clusters based on parameters (eps and min_samples).
- **Robust to outliers**: Identifies noise points effectively.

Below are separate Python implementations for each clustering technique, using a synthetic dataset generated with scikit-learn.

DBSCAN Clustering Implementation

```
# Import necessary libraries
from sklearn.cluster import DBSCAN
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_score
import numpy as np
import matplotlib.pyplot as plt

# Generate synthetic dataset
X, _ = make_blobs(n_samples=1000, centers=4, random_state=42)

# DBSCAN Clustering
def dbscan_clustering(X, eps=0.5, min_samples=5):
```

```
    # Initialize DBSCAN
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)

    # Fit and predict clusters
    labels = dbscan.fit_predict(X)

    # Calculate silhouette score (excluding noise points)
    if len(np.unique(labels)) > 1:
        silhouette = silhouette_score(X[labels != -1], labels[labels != -
1])
    else:
        silhouette = -1  # No valid clusters
        print("Warning: Only noise points detected or single cluster
formed")

    print(f"\nDBSCAN Clustering (eps={eps}, min_samples={min_samples}):")
    print(f"Silhouette Score: {silhouette:.4f}")
    print(f"Number of clusters: {len(np.unique(labels[labels != -1]))}")
    print(f"Number of noise points: {np.sum(labels == -1)}")

    # Visualize results
    plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
    plt.title('DBSCAN Clustering')
    plt.show()

    return dbscan, labels

# Run DBSCAN
if __name__ == "__main__":
    dbscan_model, dbscan_labels = dbscan_clustering(X)
```

Explanation of the Code

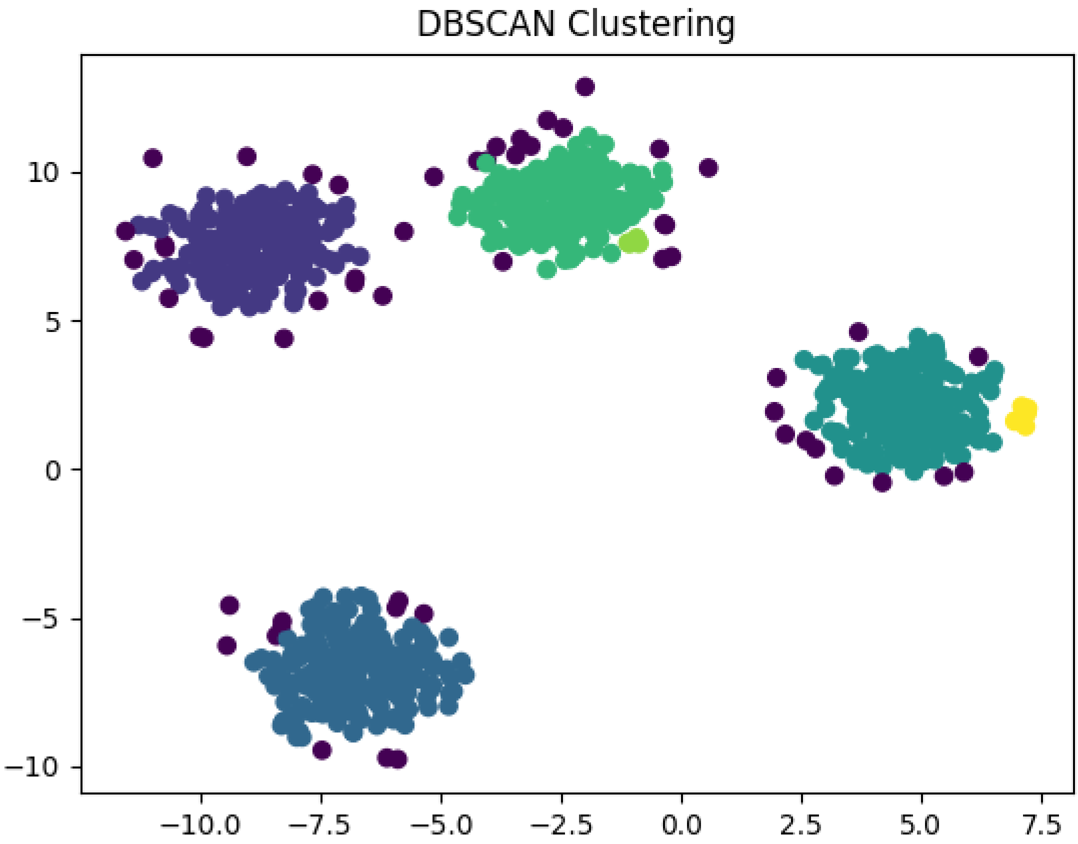
Each code section:

1. Imports required libraries (scikit-learn, numpy, matplotlib).
2. Generates a synthetic dataset with 1000 samples and 4 centers using `make_blobs`.
3. Implements a clustering algorithm (k-Means, Hierarchical, or DBSCAN).
4. Computes the silhouette score to evaluate clustering quality.
5. Visualizes the clusters using scatter plots, with additional dendrogram visualization for Hierarchical Clustering.
6. Outputs performance metrics (e.g., silhouette score, number of clusters, noise points for DBSCAN).

To run these scripts, install the required libraries (`pip install scikit-learn numpy matplotlib scipy`). Each script can be run independently to visualize clustering results on the synthetic dataset. The parameters (e.g., k, eps, min_samples) can be adjusted to explore different clustering behaviors.

Output:

```
DBSCAN Clustering (eps=0.5, min_samples=5):
Silhouette Score: 0.5732
Number of clusters: 6
Number of noise points: 56
```


DBSCAN Clustering
10
5
0
−5
−10
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5