Name: Ritesh Maurya Exp No.7 ID:VU4F2223112

AIM: Clustering

a. **Clustering algorithms for unsupervised classification**
b. **Plot the cluster data**

THEORY:
Clustering Algorithms:

Clustering algorithms are unsupervised machine learning techniques used to group similar data points into clusters based on their inherent characteristics. Unlike classification, clustering does not rely on labeled data but instead discovers hidden patterns in the dataset.

Types of Clustering Algorithms

1. **Partitioning Clustering**
    - Divides data into non-overlapping groups.
    - Example: K-Means (minimizes the distance between points and their cluster center).
2. **Hierarchical Clustering**
    - Builds a tree-like structure of clusters.
    - Example: Agglomerative Hierarchical Clustering (merges smaller clusters into larger ones).
3. **Density-Based Clustering**
    - Identifies clusters based on dense regions of data.
    - Example: DBSCAN (detects clusters of arbitrary shape and filters out noise).
4. **Grid-Based Clustering**
    - Divides the data space into a finite number of cells and performs clustering.
    - Example: STING (Statistical Information Grid-based method).
5. **Model-Based Clustering**
    - Assumes a distribution model and fits data to it.
    - Example: Gaussian Mixture Model (GMM).

Applications of Clustering Algorithms

- Customer Segmentation in marketing.

Name: Ritesh Maurya Exp No.7 ID:VU4F2223112

- Anomaly Detection in cybersecurity.
- Image Segmentation in computer vision.
- Genomic Data Analysis in bioinformatics.

K-Means Model:

K-Means is a partitioning-based clustering algorithm that groups data into K clusters by minimizing the distance between data points and their assigned cluster center (centroid). It is widely used for pattern recognition, data segmentation, and anomaly detection.

Working of K-Means Algorithm

1. **Choose the Number of Clusters (K):**
    - The user specifies the number of clusters to form.
2. **Initialize Centroids:**
    - Randomly select K points as initial cluster centers.
3. **Assign Data Points to Clusters:**
    - Each data point is assigned to the nearest centroid using Euclidean distance.
4. **Update Centroids:**
    - The new centroid of each cluster is calculated as the mean of all points in that cluster.
5. **Repeat Until Convergence:**
    - Steps 3 and 4 are repeated until centroids no longer change significantly or a maximum number of iterations is reached.

Advantages of K-Means

- Simple and computationally efficient.
- Works well on large datasets.
- Produces well-separated clusters when data is clearly defined.

Limitations of K-Means

- Requires pre-specifying K.

Name: Ritesh Maurya Exp No.7 ID:VU4F2223112

- Sensitive to initial centroid selection.
- Struggles with non-spherical and overlapping clusters.

Applications of K-Means

- Customer Segmentation in marketing.
- Image Compression and color quantization.
- Anomaly Detection in fraud detection.

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

# Example data (2D)
X = np.array([[1, 2], [1.5, 1.8], [5, 8], [8, 8], [1, 0.6], [9, 11]])

# Fit the KMeans model with k=2 (two clusters)
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(X)

# Get the labels (cluster assignments) for each data point
labels = kmeans.labels_

# Print the labels and cluster centers
print("Labels:", labels)
print("Cluster Centers:", kmeans.cluster_centers_)
```

✓ 0.0s Python

```
Labels: [1 1 0 0 1 0]
Cluster Centers: [[7.33333333 9.        ]
 [1.16666667 1.46666667]]
```

STEP 1: Load the data and define the number of clusters

```python
import pandas as pd

df = pd.read_csv('hair_loss.csv')

df.head()
```

✓ 4.2s Python

	total_protein	total_keratine	hair_texture	vitamin	manganese	iron	calcium	body_water_content	stress_level	liver_data	hair_fall
0	312	100	14	249	87	55	333	44	41	368	4
1	52	207	3	425	387	1	182	26	65	41	1
2	170	197	11	140	199	91	414	30	54	90	4
3	256	334	19	358	120	3	35	48	45	65	2
4	309	185	58	207	329	301	345	23	90	346	4

Name: Ritesh Maurya Exp No.7 ID:VU4F2223112

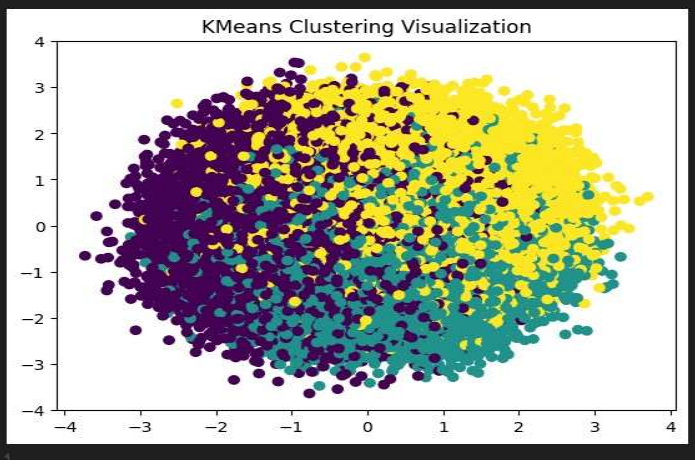
```python
from sklearn.cluster import KMeans
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load and preprocess data (use your dataset)
df = pd.read_csv('hair_loss.csv')  # Update with your file path
df = df.dropna()  # Handle missing values
X = df.drop(columns=['target_column'], errors='ignore')  # Update target_column if needed
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply KMeans clustering
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)

# Get cluster labels
kmeans_labels = kmeans.labels_

# Evaluate KMeans clustering
from sklearn.metrics import silhouette_score, davies_bouldin_score

silhouette_kmeans = silhouette_score(X_scaled, kmeans_labels)
print("Silhouette Score for KMeans:", silhouette_kmeans)

db_kmeans = davies_bouldin_score(X_scaled, kmeans_labels)
print("Davies-Bouldin Index for KMeans:", db_kmeans)
```

STEP 2: Perform Visualization using Matplotlib if needed

```python
# Visualization using PCA (optional)
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis')
plt.title("KMeans Clustering Visualization")
plt.show()
```

✓ 1m 27.2s Python

```
Silhouette Score for KMeans: 0.05778147026558065
Davies-Bouldin Index for KMeans: 3.2757213581776075
Matplotlib is building the font cache; this may take a moment.
```



Name: Ritesh Maurya Exp No.7 ID:VU4F2223112

STEP 3: Plot MiniBatchK-Means and DBSCAN using K-Means and Matplotlib

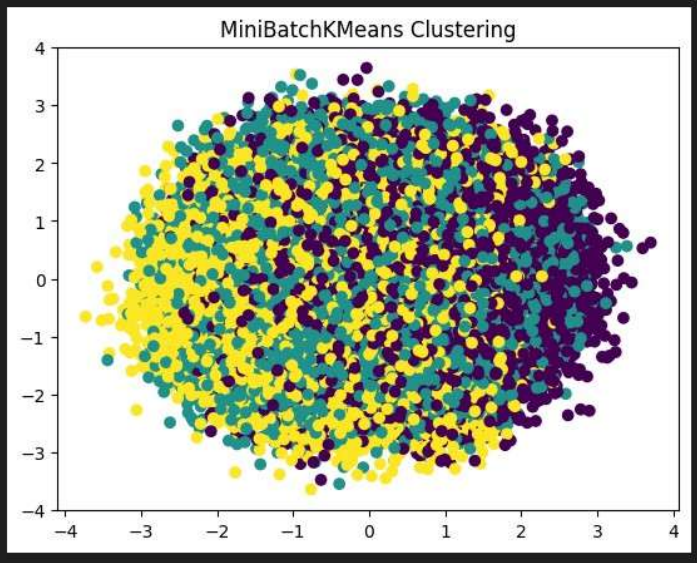
```python
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN, MiniBatchKMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv('hair_loss.csv')  # Update with your file path

# Preprocessing (handling missing values)
df = df.dropna()  # Drop rows with missing values
X = df.drop(columns=['target_column'], errors='ignore')  # Replace 'target_column' with actual target column name

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply MiniBatchKMeans Clustering (Memory-efficient KMeans)
minibatch_kmeans = MiniBatchKMeans(n_clusters=3, random_state=42, batch_size=1000)
minibatch_kmeans.fit(X_scaled)
minibatch_kmeans_labels = minibatch_kmeans.labels_

# Apply DBSCAN Clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(X_scaled)

# Evaluate Silhouette Score
print("Silhouette Score for MiniBatchKMeans:", silhouette_score(X_scaled, minibatch_kmeans_labels))
print("Silhouette Score for DBSCAN:", silhouette_score(X_scaled, dbscan_labels) if len(set(dbscan_labels)) > 1 else 'Undefine

# Evaluate Davies-Bouldin Index
print("Davies-Bouldin Index for MiniBatchKMeans:", davies_bouldin_score(X_scaled, minibatch_kmeans_labels))
print("Davies-Bouldin Index for DBSCAN:", davies_bouldin_score(X_scaled, dbscan_labels) if len(set(dbscan_labels)) > 1 else '

# Visualize using PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Plot MiniBatchKMeans Clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=minibatch_kmeans_labels, cmap='viridis')
plt.title("MiniBatchKMeans Clustering")
plt.show()

# Plot DBSCAN Clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels, cmap='viridis')
plt.title("DBSCAN Clustering")
plt.show()
```

1m 20.2s Python

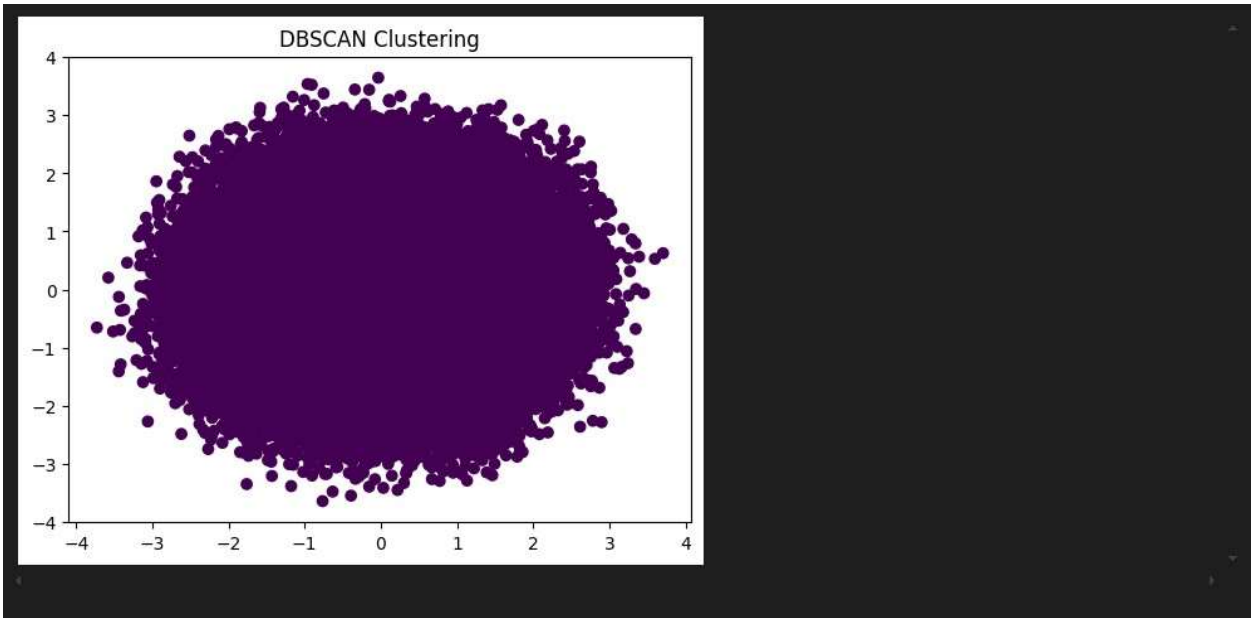
MiniBatchK-Means Plot:

Name: Ritesh Maurya Exp No.7 ID:VU4F2223112

```
Silhouette Score for MiniBatchKMeans: 0.05305746507521844
Silhouette Score for DBSCAN: Undefined
Davies-Bouldin Index for MiniBatchKMeans: 3.4335642339169508
Davies-Bouldin Index for DBSCAN: Undefined
```



DBSCAN Plot:



CONCLUSION: Hence, we have successfully implemented

a. **Clustering algorithms for unsupervised classification**
b. **Plot the data.**