

AI and DS-1

Experiment 7

Aim: To implement different clustering algorithms.

Problem statement:

- a) Clustering algorithm for unsupervised classification (K-means, density based (DBSCAN))
- b) Plot the cluster data and show mathematical steps.

Theory:

Clustering is a type of unsupervised learning technique. In unsupervised learning, we work with datasets that do not have predefined labels or outputs. The goal is to uncover hidden patterns, structures, or groupings within the data. Clustering specifically involves grouping a collection of data points into clusters, where points within the same cluster are more similar to each other than to those in other clusters.

The primary objective of clustering is to group data based on similarities, so that elements in the same group share common traits or characteristics.

Real-World Applications of Clustering

1. Marketing: Clustering is used to segment customers based on purchasing behavior or demographics to enable targeted marketing strategies.
2. Biology: It's applied in identifying and grouping various species based on genetic traits or observable features.
3. Library Organization: Books and resources can be grouped based on topics or genres for easier access.

4. Insurance Sector: Helps in grouping policyholders to detect fraudulent claims or tailor insurance plans.
5. Urban Development: Clustering assists in analyzing housing patterns and neighborhood planning based on location, price, and amenities.
6. Seismology: Clustering earthquake-prone zones can aid in identifying risk levels and preparing for natural disasters.

Types of Clustering Algorithms

When selecting a clustering algorithm, scalability and efficiency are crucial—especially with large datasets. Some algorithms compute the similarity between every pair of points, which makes them computationally expensive, with time complexity of $O(n^2)$, making them impractical for millions of data points.

1. Density-Based Clustering

These methods define clusters as dense regions in the data space, separated by areas of lower density. They perform well with irregularly shaped clusters and can handle noise.

Examples: DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS.

2. Hierarchical Clustering

This approach builds a tree of clusters based on hierarchical relationships. It starts either from individual data points (bottom-up) or from a single cluster (top-down).

Types:

- Agglomerative: Merges smaller clusters into larger ones.
- Divisive: Splits large clusters into smaller groups.
Examples: CURE, BIRCH.

3. Partitioning Clustering

This technique divides the dataset into a fixed number of clusters (k). It seeks to optimize a given objective function, often based on distance or similarity.

Examples: K-Means, CLARANS.

4. Grid-Based Clustering

In this method, the data space is divided into a grid structure, and clustering is performed on these grids instead of individual points. It is highly efficient and independent of the data size.

Examples: STING, CLIQUE, WaveCluster.

Dataset description:

For this experiment we have used the Wine Quality dataset which contains physicochemical properties of red wine samples along with a quality score rated by experts.

Its attributes are:


1. fixed acidity – Non-volatile acids (e.g., tartaric acid) contributing to stability and flavor.
2. volatile acidity – Acetic acid content; high levels may result in an unpleasant taste.
3. citric acid – Adds freshness and flavor to wine.
4. residual sugar – Sugar left after fermentation; affects sweetness.
5. chlorides – Salt content in the wine.
6. free sulfur dioxide – SO_2 available to protect wine from microbes and oxidation.
7. total sulfur dioxide – Total concentration of SO_2 (free + bound).
8. density – Affected by sugar and alcohol content.
9. pH – Indicates the acidity or alkalinity of wine.
10. sulphates – Acts as a preservative and antioxidant.
11. alcohol – Percentage of alcohol by volume.
12. quality – Target variable; wine quality score (typically 3 to 8).
13. Id – Unique identifier for each sample (not used in modeling).

Steps performed:

Importing libraries and loading dataset:

```
[ ] import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from google.colab import files

uploaded = files.upload()
```

 Choose Files No file chosen Upload

Saving WineQT.csv to WineQT (1).csv

This is the first step wherein we are importing the necessary libraries and loading the dataset.

Performing Kmeans clustering:

We are primarily selecting 2 columns on which our clustering will be based i.e volatile acidity and alcohol.

Step 1. Scaling the features:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)
```

Here we are creating a new data frame with only the required 2 features and applying scaling to the features. Scaling is applied to standardize the features, i.e., scale them so that: Mean = 0 and Standard Deviation = 1

Step 2. Using PCA (Principal component analysis)

```
from sklearn.decomposition import PCA

# Reduce dimensions to 2 or 3
pca = PCA(n_components=2)
features_pca = pca.fit_transform(features_scaled)

# Check variance retained
print(f"Total variance retained: {sum(pca.explained_variance_ratio_):.2f}")

Total variance retained: 0.46
```

We are using PCA to reduce the number of features to 2 dimensions so we can visualize clusters and simplify the data while still retaining most of the important information (variance).

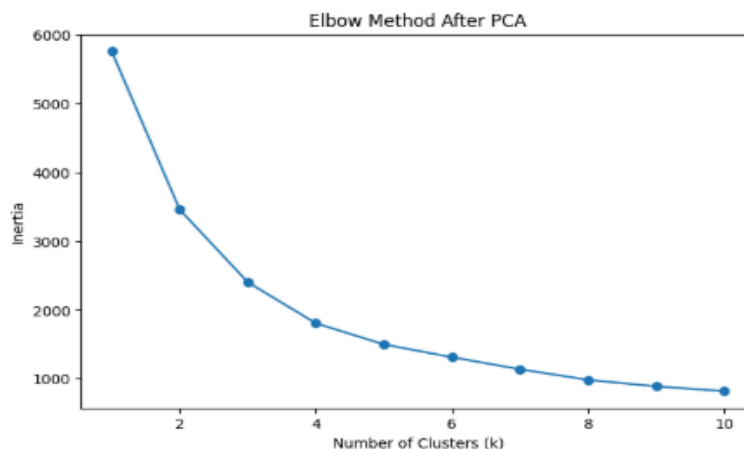
Step 3. Elbow method

```
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

inertia = []
K_range = range(1, 11) # Test k from 1 to 10

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

# Plot the Elbow curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, marker="o")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.title("Elbow Method After PCA")
plt.show()
```



Here we have used the Elbow Method to find the optimal number of clusters for KMeans.

Why the Elbow Method?

The Elbow Method helps determine the optimal number of clusters (k) by plotting:

- X-axis: Number of clusters (k)
- Y-axis: Within-Cluster Sum of Squares (WCSS / Inertia)

We select the “elbow point” – where the decrease in WCSS slows down – as the best k.

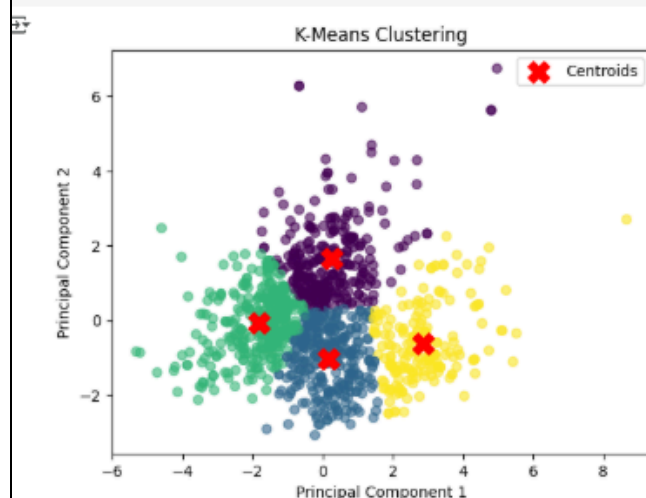
```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
clusters = kmeans.fit_predict(features_pca)
```

Based on the elbow method we have selected k=4 and now we will visualize the clusters

Step 4. Visualizing the clusters

```
# Visualize clusters
plt.scatter(features_pca[:, 0], features_pca[:, 1], c=clusters, cmap="viridis", alpha=0.6)
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], c="red", marker="x", s=200, label="Centroids")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("K-Means Clustering")
plt.legend()
plt.show()
```



The scatter plot visualizes the clusters formed using K-Means after reducing dimensionality with PCA. Five distinct clusters are clearly observed, with red X marks indicating the centroids. The clusters are relatively compact and well-separated, suggesting that K-Means performed effective segmentation of the data. Only minor overlaps and a few outliers are visible.

Step 5. Calculating silhouette score and Davies Bouldin score

Now we will calculate the Silhouette Score, which measures how well the data points fit within their assigned clusters.

- Range: -1 to +1
 - +1 → Perfect clustering
 - 0 → Overlapping clusters
 - Negative → Misclassified points

Here is the calculated silhouette score:

```
from sklearn.metrics import silhouette_score  
  
score = silhouette_score(features_pca, clusters)  
print(f"Silhouette Score: {score:.3f}")  
  
Silhouette Score: 0.372
```

A silhouette score of 0.372 shows that clustering is moderate. Clusters exist, but they overlap or are not well-defined.

Calculating Davies Bouldin score:

```
from sklearn.metrics import davies_bouldin_score  
  
db_score = davies_bouldin_score(features_pca, clusters)  
print(f"Davies-Bouldin Score: {db_score:.3f}")  
  
Davies-Bouldin Score: 0.852
```

A Davies-Bouldin Score of 0.852 indicates that the clusters are reasonably well-separated and moderately compact.

Performing DBSCAN:

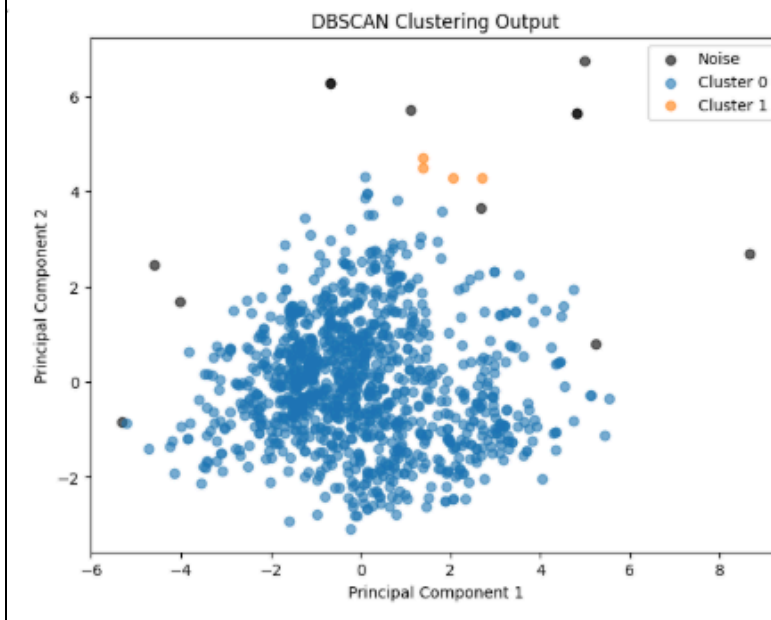
```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot as plt
import numpy as np

dbscan = DBSCAN(eps=0.8, min_samples=5)
clusters_dbscan = dbscan.fit_predict(features_pca)

unique_labels = np.unique(clusters_dbscan)

plt.figure(figsize=(8, 6))
for label in unique_labels:
    if label == -1:
        plt.scatter(features_pca[clusters_dbscan == label, 0],
                    features_pca[clusters_dbscan == label, 1],
                    color='black', label='Noise', alpha=0.6)
    else:
        plt.scatter(features_pca[clusters_dbscan == label, 0],
                    features_pca[clusters_dbscan == label, 1],
                    label=f'Cluster {label}', alpha=0.6)

plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("DBSCAN Clustering Output")
plt.legend()
plt.show()
```



DBSCAN (Density-Based Spatial Clustering of Applications with Noise) was applied on the PCA-reduced features to discover clusters based on density. The parameters used were $\text{eps}=0.8$ and $\text{min_samples}=5$. The scatter plot shows several clusters formed with some points labeled as noise (black), indicating that they didn't belong to any dense region.

What are eps and min_samples in DBSCAN

- eps (epsilon): This defines the maximum distance between two points for them to be considered neighbors.
 - Smaller eps → tighter clusters, more noise points.
 - Larger eps → looser clusters, fewer noise points.
- min_samples: This is the minimum number of points required to form a dense region (a cluster).
 - Smaller min_samples → more, smaller clusters (can lead to noise).
 - Larger min_samples → fewer, larger clusters (may merge nearby clusters).

Together, these parameters control how sensitive DBSCAN is to density. We usually need to try different combinations of eps and min_samples to get the best clustering result.

```
from sklearn.metrics import silhouette_score, davies_bouldin_score
import numpy as np

mask = clusters_dbscan != -1

if np.sum(mask) > 1:
    sil_score = silhouette_score(features_pca[mask], clusters_dbscan[mask])
    db_score = davies_bouldin_score(features_pca[mask], clusters_dbscan[mask])

    print("Silhouette Score for DBSCAN:", round(sil_score, 3))
    print("Davies-Bouldin Score for DBSCAN:", round(db_score, 3))
else:
    print("Not enough valid clusters to calculate evaluation scores.")

Silhouette Score for DBSCAN: 0.455
Davies-Bouldin Score for DBSCAN: 0.495
```

The DBSCAN algorithm achieved a Silhouette Score of 0.455, indicating moderately well-defined clusters. The Davies-Bouldin Score of 0.495 suggests that the clusters are compact and well-separated, confirming that DBSCAN effectively identified meaningful groupings in the data.

Conclusion:

Based on the evaluation metrics, DBSCAN performed better than KMeans for clustering the wine dataset. While KMeans achieved a Silhouette Score of 0.372 and a Davies-Bouldin Score of 0.852, DBSCAN showed improved clustering quality with a Silhouette Score of 0.455 and a lower Davies-Bouldin Score of 0.495. This indicates that DBSCAN formed more compact and well-separated clusters. Additionally, DBSCAN's ability to identify outliers and handle clusters of varying shapes contributed to its superior performance in this context.