Name: Kaushik Kotian Class:D15B Roll No. : 30 Batch-B

DS Using Python Lab

Experiment: 07

Aim: To implement different clustering algorithms

Theory:

Dataset: https://www.kaggle.com/code/codebreaker619/employee-download
It is an employee dataset which contains 23000 rows and around 12 columns.

(a) Clustering algorithm for unsupervised classification (K-means, density based (DBSCAN), Hierarchical clustering).

1. K-means Clustering:
K-means is one of the simplest and most commonly used clustering algorithms. It partitions data into K clusters, where each data point belongs to the cluster with the nearest mean (centroid). Here's how it works:

Initialization: Randomly initialize K cluster centroids.
Assignment Step: Assign each data point to the nearest centroid based on Euclidean distance.
Update Step: Recalculate the centroids based on the mean of all data points assigned to each cluster.
Repeat Assignment and Update Steps: Iterate until convergence (when centroids no longer change significantly) or a maximum number of iterations is reached. K-means is sensitive to the initial centroids, and different initializations may lead to different results. It's also suited for spherical clusters and works well when clusters are well-separated.

2. Density-based Spatial Clustering of Applications with Noise (DBSCAN): DBSCAN is a density-based clustering algorithm that identifies clusters of arbitrary shapes. It groups together closely packed points based on two parameters: epsilon (ε) and the minimum number of points (MinPts) required to form a dense region. Here's how it works:

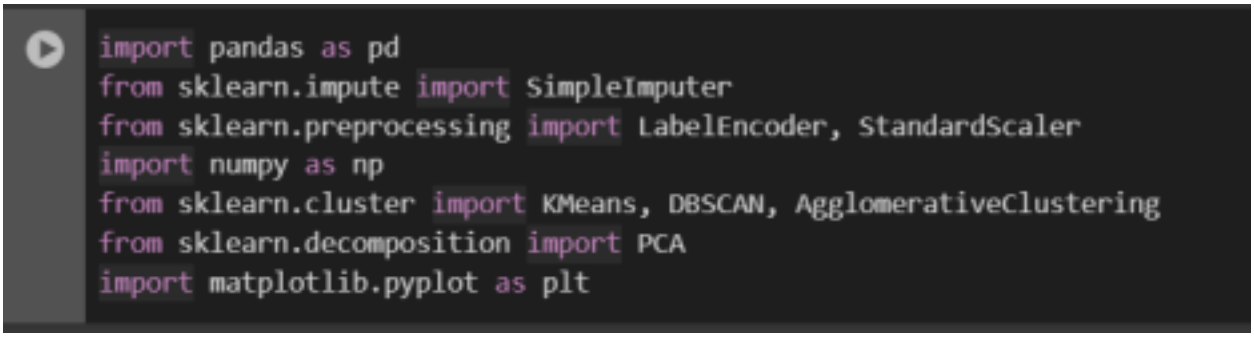
Core Points: A point is considered a core point if it has at least MinPts points (including itself) within distance ε.
Border Points: A point that is within distance ε of a core point but does not have MinPts points within distance ε.
Noise Points: Points that are neither core nor border points. DBSCAN identifies clusters by expanding clusters from core points. It does not require the number of clusters to be specified beforehand, and it can handle noise and outliers effectively.

3. Hierarchical Clustering:
Hierarchical clustering builds a hierarchy of clusters by either iteratively merging smaller clusters into larger ones (agglomerative) or by iteratively splitting clusters into smaller ones (divisive). Here's how agglomerative hierarchical clustering works:

Initialization: Start with each data point as a singleton cluster. Merge Step: Compute the pairwise distances between all clusters and merge the two closest clusters. Update Distance Matrix: Recalculate the distance matrix based on the new clusters. Repeat Merge Step: Iterate until all data points are in a single cluster or until a predefined number of clusters is reached. Hierarchical clustering produces a dendrogram, a tree-like structure representing the merging process. It does not require the number of clusters to be specified beforehand, and it allows for visual inspection of cluster relationships.

(b) Plot the cluster data and show mathematical steps:

1. Importing all the essential functions and libraries



```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder, StandardScaler
import numpy as np
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

2. Loading and preprocessing the dataset

```
data = pd.read_csv("Employee.csv")
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23490 entries, 0 to 23489
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   employee_id           23490 non-null  int64
 1   department            23490 non-null  object
 2   region                23490 non-null  object
 3   education             22456 non-null  object
 4   gender                23490 non-null  object
 5   recruitment_channel   23490 non-null  object
 6   no_of_trainings       23490 non-null  int64
 7   age                   23490 non-null  int64
 8   previous_year_rating  21678 non-null  float64
 9   length_of_service     23490 non-null  int64
 10  awards_won?           23490 non-null  int64
 11  avg_training_score    23490 non-null  int64
dtypes: float64(1), int64(6), object(5)
memory usage: 2.2+ MB
```

```
imputer = SimpleImputer(strategy='most_frequent')
data['education'] = imputer.fit_transform(data[['education']])
data['previous_year_rating'] = imputer.fit_transform(data[['previous_year_rating']])

# encoding categorical variables
label_encoder = LabelEncoder()
data['department'] = label_encoder.fit_transform(data['department'])
data['region'] = label_encoder.fit_transform(data['region'])
data['education'] = label_encoder.fit_transform(data['education'])
data['gender'] = label_encoder.fit_transform(data['gender'])
data['recruitment_channel'] = label_encoder.fit_transform(data['recruitment_channel'])

# Standardizing numerical variables
scaler = StandardScaler()
data[['no_of_trainings', 'age', 'previous_year_rating', 'length_of_service', 'avg_training_score']] = scaler.fit_transform(data[['no_
```

```
pca=PCA(n_components=2)
X_pca=pca.fit_transform(data)
```

3. Selecting the features for clustering

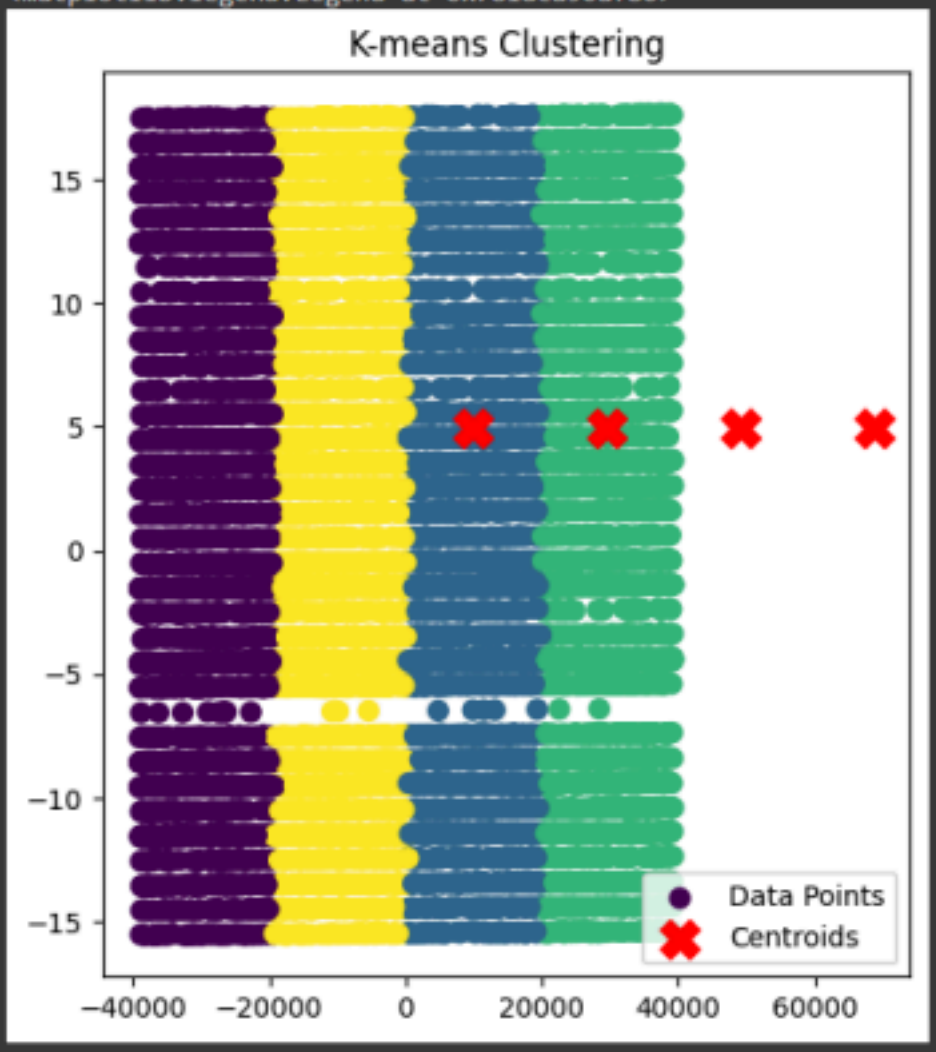
```
features=['employee_id','department']
X=data[features]
```

4. K-means clustering

```
kmeans=KMeans(n_clusters=4,random_state=42)
kmeans_labels=kmeans.fit_predict(X)
kmeans_centroids=kmeans.cluster_centers_
```

```
plt.figure(figsize=(18, 6))
# Plotting K-means clusters with centroids
plt.subplot(131)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=kmeans_labels, cmap='viridis',
s=50, label='Data Points')
plt.scatter(kmeans_centroids[:, 0], kmeans_centroids[:, 1], c='red',
marker='X', s=200, label='Centroids')
plt.title('K-means Clustering')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x781dca90af80>
```

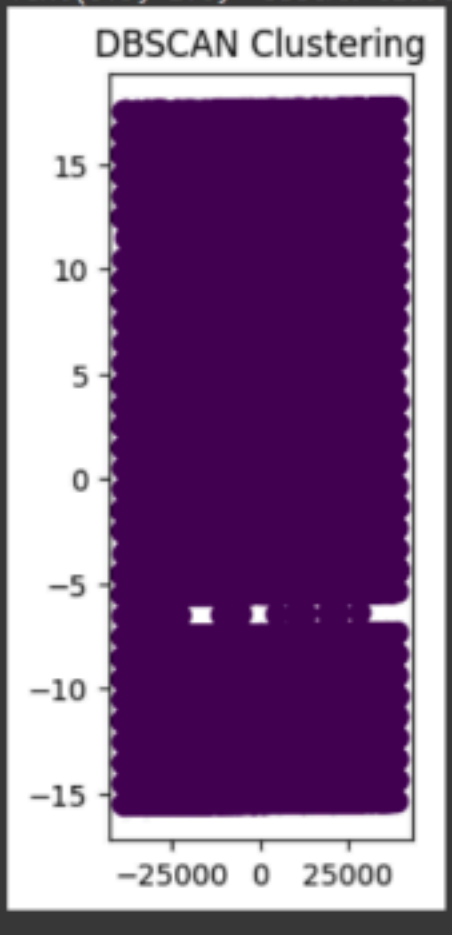


5. DBSCAN

```
[ ]  dbscan = DBSCAN(eps=0.5, min_samples=10)
     dbscan_labels = dbscan.fit_predict(X)
```

```
# Plotting DBSCAN clusters
plt.subplot(132)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=dbscan_labels, cmap='viridis',
s=50)
plt.title('DBSCAN Clustering')
```

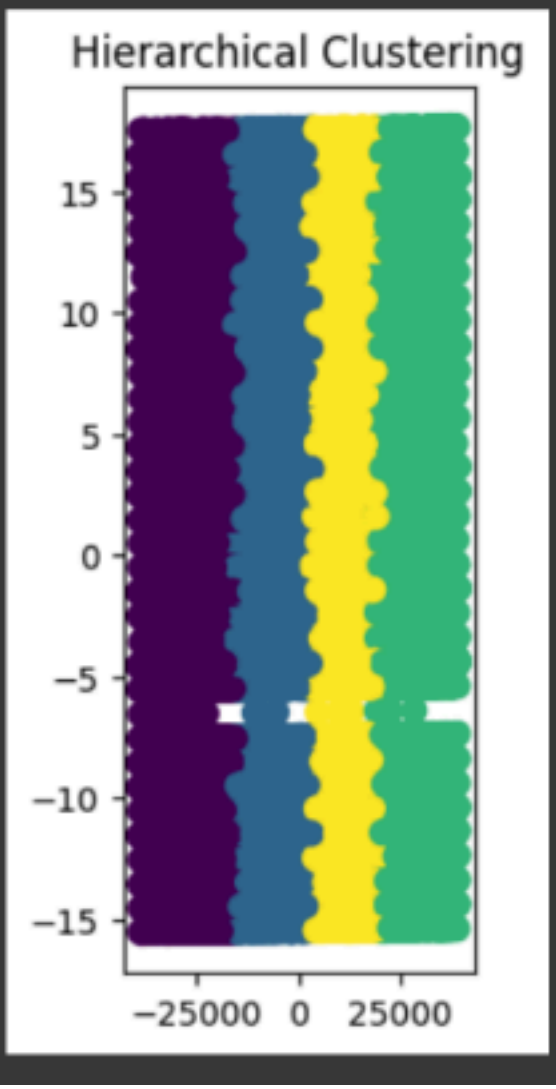
```
Text(0.5, 1.0, 'DBSCAN Clustering')
```



6. Hierarchical clustering

```
hierarchical = AgglomerativeClustering(n_clusters=4)
hierarchical_labels = hierarchical.fit_predict(X)
```

```
# Plotting Hierarchical clustering
plt.subplot(133)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=hierarchical_labels,
cmap='viridis', s=50)
plt.title('Hierarchical Clustering')
plt.show()
```



Conclusion:

Hence, we understood different clustering algorithms for unsupervised classification by implementing them on our dataset.