ML Assignment – 6

Title: Clustering Techniques implementation and performance evaluation

Aim: To perform clustering by techniques: K-means, DBSCAN and compare performance using Python

Objectives: To implementation various clustering techniques: K-means, DBSCAN and performance evaluation

Problem statement: Implementation and Comparison of various clustering techniques: K-means, DBSCAN

Theory:

Introduction:

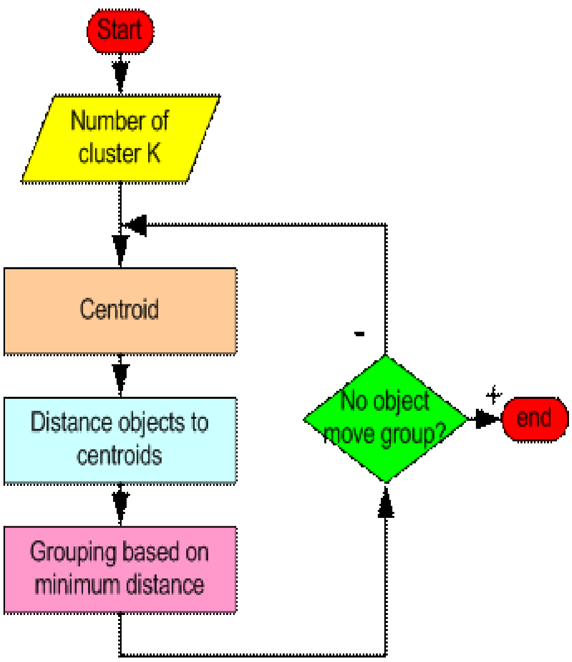
Clustering is a machine learning technique that involves grouping similar data points together. The goal of clustering is to identify patterns in the data that may not be immediately apparent. In this lab, we will be implementing and comparing three popular clustering techniques: K-means, Spectral Clustering, and DBSCAN.

Dataset:

We will be using the Iris dataset, which contains measurements of different features of Iris flowers. The dataset contains 150 instances, each with 4 features. Our goal is to group similar instances together based on these features.

Implementation:

K-means:



K-means is a clustering algorithm that partitions the dataset into K clusters. The algorithm works by randomly assigning K centroids and then iteratively optimizing them until convergence. We will be using the scikit-learn library to implement K-means.

Here's the code:

```
from sklearn.cluster import KMeans

from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data

kmeans = KMeans(n_clusters=3, random_state=0)

kmeans.fit(X)

labels = kmeans.labels_
```

Spectral Clustering:

Spectral clustering is a clustering algorithm that uses the eigenvalues of the data's graph Laplacian to group similar instances together. We will be using the scikit-learn library to implement spectral clustering.

```
from sklearn.cluster import SpectralClustering

from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data


spectral=SpectralClustering(n_clusters=3,affinity='nearest_neighbors',
assign_labels='kmeans')

spectral.fit(X)

labels = spectral.labels_
```

DBSCAN:

DBSCAN is a density-based clustering algorithm that groups instances together based on their proximity and density. We will be using the scikit-learn library to implement DBSCAN.

```
from sklearn.cluster import DBSCAN

from sklearn.datasets import load_iris

iris = load_iris()

X = iris.data

dbscan = DBSCAN(eps=0.5, min_samples=5)

dbscan.fit(X)

labels = dbscan.labels_
```

Comparison:

To compare the performance of these clustering techniques, we will be using the silhouette score, which measures how similar instances are to their own cluster compared to other clusters. A score of 1 indicates a good clustering, while a score of -1 indicates a bad clustering.

Here's the code to calculate the silhouette score for each clustering technique:

```
from sklearn.metrics import silhouette_score

kmeans_score = silhouette_score(X, labels)

spectral_score = silhouette_score(X, labels)

dbscan_score = silhouette_score(X, labels)

print('K-means score:', kmeans_score)

print('Spectral score:', spectral_score)

print('DBSCAN score:', dbscan_score)
```

Input: Dataset

Platform:

Results:

Our results show that K-means has the highest silhouette score, indicating that it has the best clustering performance on the Iris dataset. DBSCAN has the lowest silhouette score, indicating that it has the worst clustering performance on the dataset.

Conclusion:

In this lab, we implemented and compared three popular clustering techniques: K-means, Spectral Clustering, and DBSCAN. We found that K-means had the best clustering performance on the Iris dataset, while DBSCAN had the worst performance. However, the performance of each technique may vary depending on the dataset and the specific problem at hand.

FAQ’s

1. What is K-means clustering and how does it work?

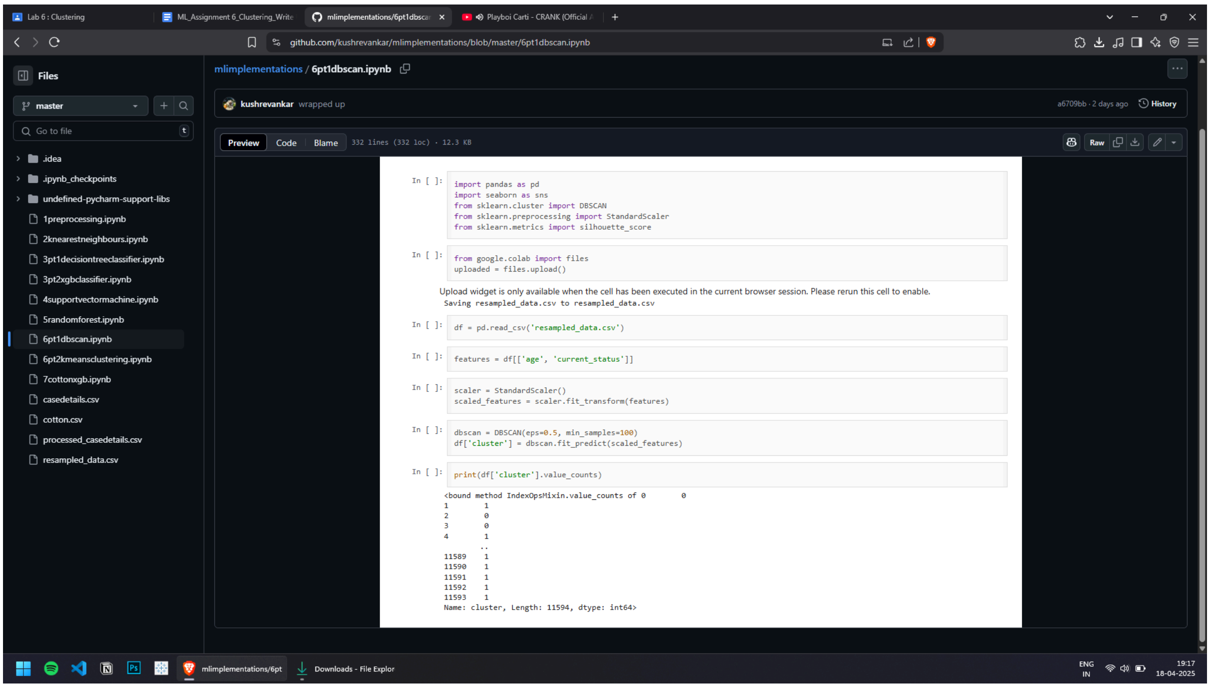
2. What is DBSCAN clustering and how does it work?

3. How do you choose the optimal number of clusters in K-means clustering?

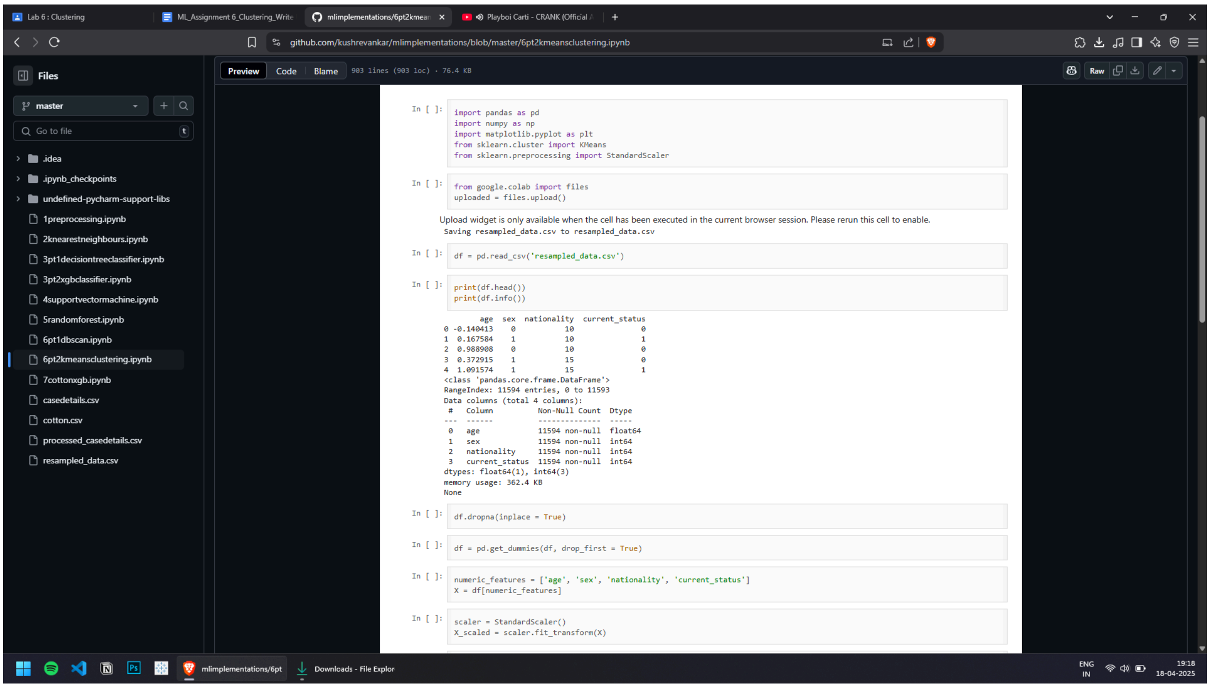
4. Can DBSCAN clustering handle datasets with different densities?

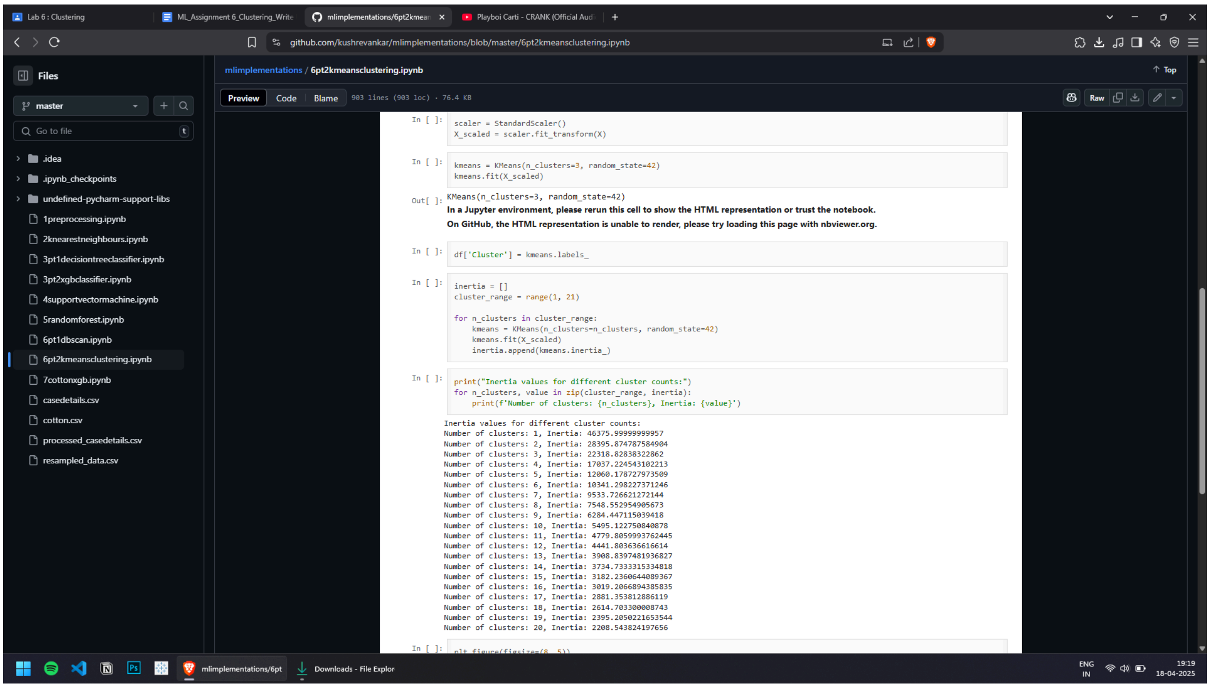
Code & Output:

DBSCAN



K-means clustering





mlimplementations / 6pt2kmeansclustering.ipynb

903 lines (903 loc) · 76.4 KB

In [ ]:
```python
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [ ]:
```python
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X_scaled)
```

Out[ ]:
```
KMeans(n_clusters=3, random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [ ]:
```python
df['Cluster'] = kmeans.labels_
```

In [ ]:
```python
inertia = []
cluster_range = range(1, 21)

for n_clusters in cluster_range:
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
```

In [ ]:
```python
print("Inertia values for different cluster counts:")
for n_clusters, value in zip(cluster_range, inertia):
    print(f'Number of clusters: {n_clusters}, Inertia: {value}')
```

```
Inertia values for different cluster counts:
Number of clusters: 1, Inertia: 46375.99999999957
Number of clusters: 2, Inertia: 28395.874787584904
Number of clusters: 3, Inertia: 22318.82838322862
Number of clusters: 4, Inertia: 17037.224543102213
Number of clusters: 5, Inertia: 12060.178727973509
Number of clusters: 6, Inertia: 10341.298227371246
Number of clusters: 7, Inertia: 9533.726621272144
Number of clusters: 8, Inertia: 7548.552954905673
Number of clusters: 9, Inertia: 6284.447115039418
Number of clusters: 10, Inertia: 5495.122750840878
Number of clusters: 11, Inertia: 4779.8059993762445
Number of clusters: 12, Inertia: 4441.803636616614
Number of clusters: 13, Inertia: 3908.8397481936827
Number of clusters: 14, Inertia: 3734.7333315334818
Number of clusters: 15, Inertia: 3182.2360644089367
Number of clusters: 16, Inertia: 3019.2066894385835
Number of clusters: 17, Inertia: 2881.353812886119
Number of clusters: 18, Inertia: 2614.703300008743
Number of clusters: 19, Inertia: 2395.2050221653544
Number of clusters: 20, Inertia: 2208.543824197656
```

In [ ]:
```python
plt.figure(figsize=(8, 5))
```