

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:

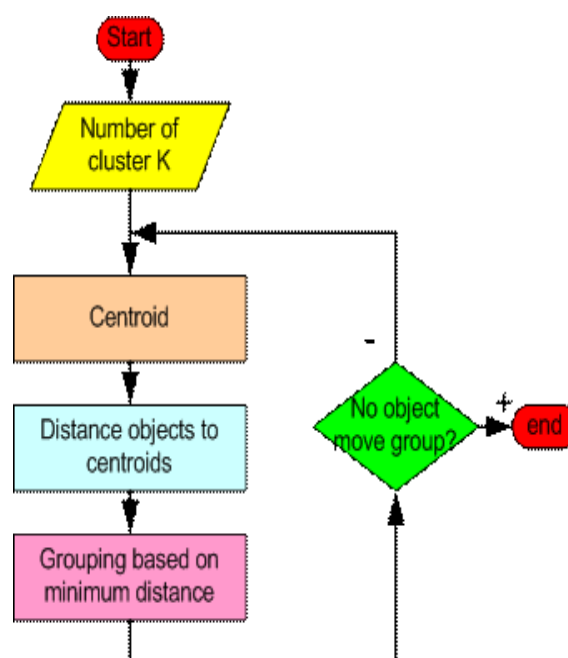
Clustering is a machine learning technique that involves grouping similar data points together. The goal of clustering is to identify patterns in the data that may not be immediately apparent. In this lab, we will be implementing and comparing three popular clustering techniques: K-means, Spectral Clustering,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

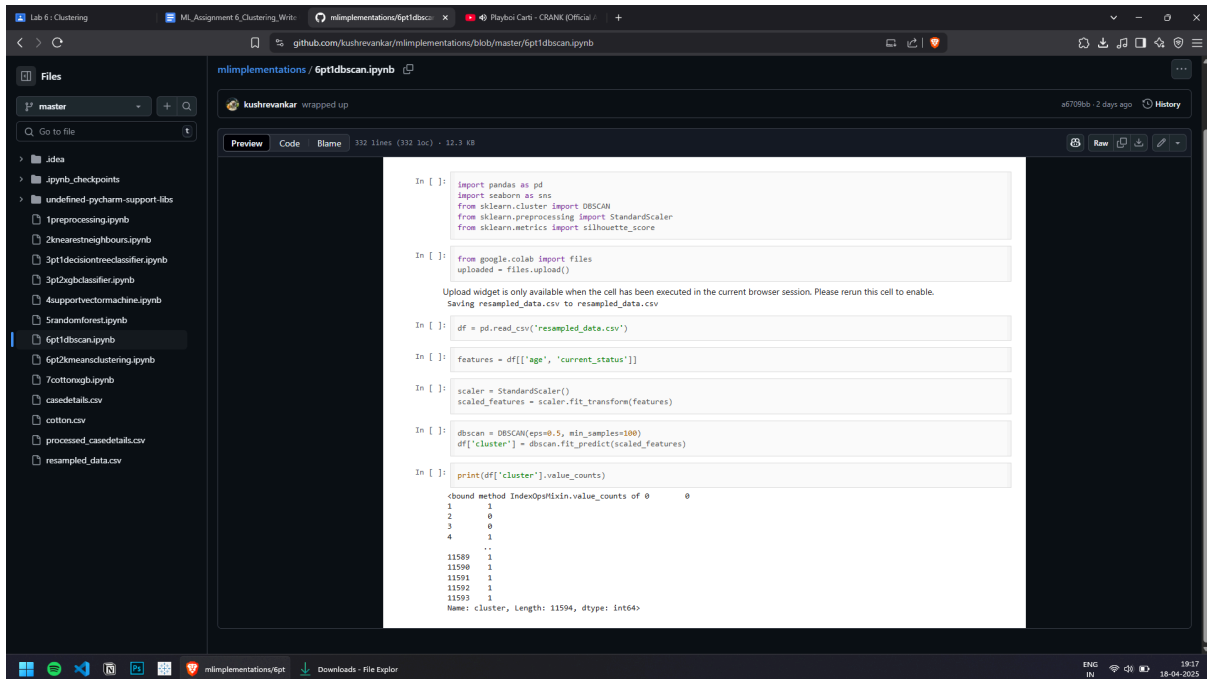
In this lab, we implemented and compared three popular clustering techniques: K-means, Spectral Clustering, and DBSCAN. We found that K-means had the best clustering performance on the Iris dataset, while DBSCAN had the worst performance. However, the performance of each technique may vary depending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



The screenshot shows a Jupyter Notebook titled 'm implementations / 6pt1dbscan.ipynb'. The code in the notebook is as follows:

```
In [ ]: import pandas as pd
import seaborn as sns
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score

In [ ]: from google.colab import files
uploaded = files.upload()

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving resampled_data.csv to resampled_data.csv

In [ ]: df = pd.read_csv('resampled_data.csv')

In [ ]: features = df[['age', 'current_status']]

In [ ]: scaler = StandardScaler()
scaled_features = scaler.fit_transform(features)

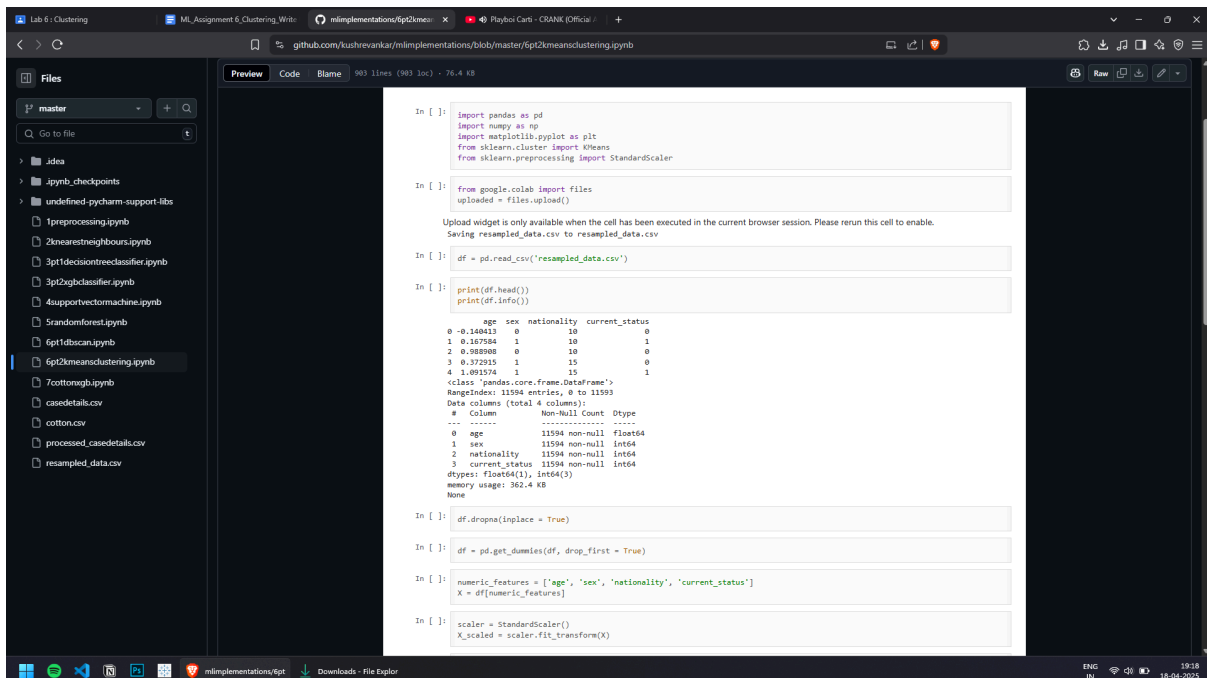
In [ ]: dbscan = DBSCAN(eps=0.5, min_samples=100)
df['cluster'] = dbscan.fit_predict(scaled_features)

In [ ]: print(df['cluster'].value_counts)
```

The output of the notebook is:

```
<bound method IndexOpsMixin.value_counts of 0 0
1 1
2 0
3 0
4 1
..
11589 1
11590 1
11591 1
11592 1
11593 1
Name: cluster, Length: 11594, dtype: int64>
```

K-means clustering



The screenshot shows a Jupyter Notebook titled 'm implementations / 6pt2kmeansclustering.ipynb'. The code in the notebook is as follows:

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

In [ ]: from google.colab import files
uploaded = files.upload()

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving resampled_data.csv to resampled_data.csv

In [ ]: df = pd.read_csv('resampled_data.csv')

In [ ]: print(df.head())
print(df.info())
```

The output of the notebook is:

```
age sex nationality current_status
0 0.140411 0 10 0
1 0.167584 1 10 1
2 0.082068 0 10 0
3 0.372915 1 15 0
4 1.091574 1 15 1
<class 'pandas.core.frame.DataFrame'>
Data columns (total 4 columns):
# Column Non-Null Count Dtype
---  ---
0 age 11594 non-null float64
1 sex 11594 non-null int64
2 nationality 11594 non-null int64
3 current_status 11594 non-null int64
dtypes: float64(1), int64(3)
memory usage: 362.4 KB
None
```

```
In [ ]: df.dropna(inplace = True)

In [ ]: df = pd.get_dummies(df, drop_first = True)

In [ ]: numeric_features = ['age', 'sex', 'nationality', 'current_status']
X = df[numeric_features]

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

Lab 6: ClusteringML_Assignment 6: Clustering: Webmimplementations/Epit2kmeansclustering.ipynb - Playbot Carb - CRANK (Official Asu)

github.com/kushrevanka/mimplementations/blob/master/Epit2kmeansclustering.ipynb

Files

master

Go to file

adca

ipynb_checkpoints

undefined-pycharm-support-files

1preprocessing.ipynb

2knearestneighbour.ipynb

3pt1decisiontreeclassifier.ipynb

3pt2xgbclassifier.ipynb

4supportvectormachine.ipynb

5randomforest.ipynb

6pt1dscan.ipynb

6pt2kmeansclustering.ipynb

7cottonghg.ipynb

case_details.csv

cotton.csv

processed_case_details.csv

resampled_data.csv

mimplementations / Epit2kmeansclustering.ipynb

903 lines (903 loc) · 74.4 KB

PreviewCodeBlame

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

In [ ]: 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 call 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 [ ]: df['Cluster'] = kmeans.labels_

In [ ]: 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 [ ]: 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: 46375.999999999997
Number of clusters: 2, Inertia: 28395.874787584984
Number of clusters: 3, Inertia: 22318.8288322882
Number of clusters: 4, Inertia: 17807.238543180213
Number of clusters: 5, Inertia: 12660.178727973509
Number of clusters: 6, Inertia: 10341.288227371246
Number of clusters: 7, Inertia: 9531.726621772344
Number of clusters: 8, Inertia: 7548.592954906673
Number of clusters: 9, Inertia: 6284.447115939418
Number of clusters: 10, Inertia: 5495.122750840878
Number of clusters: 11, Inertia: 4779.889993762445
Number of clusters: 12, Inertia: 4441.883636316634
Number of clusters: 13, Inertia: 3988.837481938827
Number of clusters: 14, Inertia: 3734.733315334818
Number of clusters: 15, Inertia: 3182.2308644689307
Number of clusters: 16, Inertia: 3019.2868894385835
Number of clusters: 17, Inertia: 2881.353812886119
Number of clusters: 18, Inertia: 2614.783388886743
Number of clusters: 19, Inertia: 2395.2858221653544
Number of clusters: 20, Inertia: 2288.543824197856
```

19:19

18-04-2025