06-advanced-clustering

April 23, 2025

1 Topic 3: Advanced Clustering Exercise

I'm following along the **Clustering** implementation from this YouTube.

This notebook can be accessed via GitHub repository here.

About the Dataset Dataset used in this exercise is using built-in data set from scikit-learn.

The list of the dataset can be accessed here.

In this exercise, I'll be using iris dataset.

Import necessary libraries & dataset

[26]:
```
# core data manipulation and visualization libraries

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib
import math
import numpy as np


# scikit-learn libraries

from sklearn import datasets
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# setting to make numbers easier to read on display (this is new for me)
pd.options.display.float_format = "{:20.2f}".format

# show all columns on output (this also new for me)
pd.set_option("display.max_columns", 999)
```

Load & display the dataset

[3]:
```
iris = datasets.load_iris()
features = iris.data
```

Standardize features

[6]:
```
scaler = StandardScaler()
features_std = scaler.fit_transform(features)
```

Create k-means object

[8]:
```
cluster = KMeans(n_clusters=3, random_state=0)
```

Train model, view predict class & view true class

[10]:
```
model = cluster.fit(features_std)
```

[11]:
```
model.labels_
```

[11]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 2, 2, 2, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2,
       0, 0, 0, 0, 2, 0, 0, 0, 0, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 2, 2, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 2, 2, 2, 2,
       2, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0], dtype=int32)
```

[12]:
```
iris.target
```

[12]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Create new observation & predict observation's cluster

[13]:
```
new_observation = [[0.8, 0.8, 0.8, 0.8]]
model.predict(new_observation)
```

[13]:
```
array([2], dtype=int32)
```

View cluster centers

[14]:
```
model.cluster_centers_
```

[14]:
```
array([[-0.05021989, -0.88337647,  0.34773781,  0.2815273 ],
       [-1.01457897,  0.85326268, -1.30498732, -1.25489349],
       [ 1.13597027,  0.08842168,  0.99615451,  1.01752612]])
```

Using MiniBatchKMeans How it works: Instead of using the full dataset to update the cluster centers each time (like regular KMeans), it uses small, random samples called "mini-batches."

This reduces the computation time and memory usage, especially for big data.

[15]:
```python
from sklearn.cluster import MiniBatchKMeans
```

[16]:
```python
iris = datasets.load_iris()
features = iris.data
scaler = StandardScaler()
features_std = scaler.fit_transform(features)
cluster = MiniBatchKMeans(n_clusters=3, random_state=0, batch_size=100)
model = cluster.fit(features_std)
```

1.0.1 Meanshift Clustering

[ ]:
```python
from sklearn.cluster import MeanShift
```

[18]:
```python
iris = datasets.load_iris()
features = iris.data
scaler = StandardScaler()
features_std = scaler.fit_transform(features)
cluster = MeanShift(n_jobs=1)
model = cluster.fit(features_std)
```

[19]:
```python
model.labels_
```

[19]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Hierachical Merging Clustering

[20]:
```python
from sklearn.cluster import AgglomerativeClustering
```

[21]:
```python
iris = datasets.load_iris()
features = iris.data
scaler = StandardScaler()
features_std = scaler.fit_transform(features)
cluster = AgglomerativeClustering(n_clusters=3)
model = cluster.fit(features_std)
```

[22]:
```python
model.labels_
```

[22]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 0, 2, 0,
       2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 2, 0, 0, 2,
       2, 2, 2, 0, 2, 2, 2, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

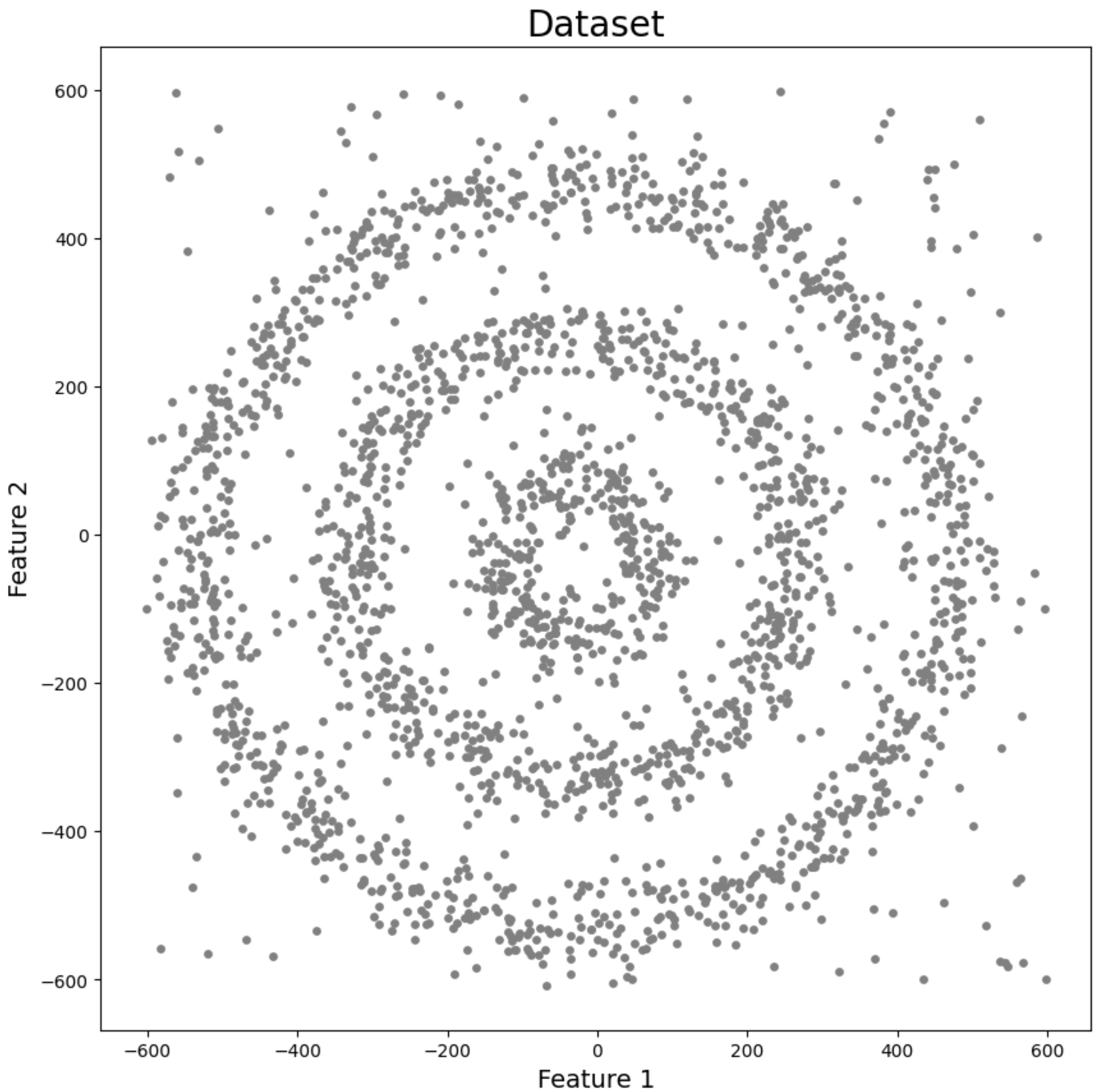
1.0.2 DBSCAN

[27]:
```
np.random.seed(42)
```

[28]:
```
# a function for creating datapoints in the form of a circle

def PointsInCircum(r, n = 100):
    return [(math.cos(2*math.pi/n*x)*r+np.random.normal(-30,30),math.sin(2*math.
 ↪pi/n*x)*r+np.random.normal(-30,30)) for x in range(1,n+1)]
```

[30]:
```
# Generate your circles
circle1 = pd.DataFrame(PointsInCircum(500, 1000))
circle2 = pd.DataFrame(PointsInCircum(300, 700))
circle3 = pd.DataFrame(PointsInCircum(100, 300))

# Generate noise
noise = pd.DataFrame([(np.random.randint(-600,600), np.random.
 ↪randint(-600,600)) for i in range(300)])

# Combine all at once
df = pd.concat([circle1, circle2, circle3, noise], ignore_index=True)
```

[31]:
```
plt.figure(figsize=(10,10))
plt.scatter(df[0],df[1],s=15,color='grey')
plt.title('Dataset',fontsize=20)
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```

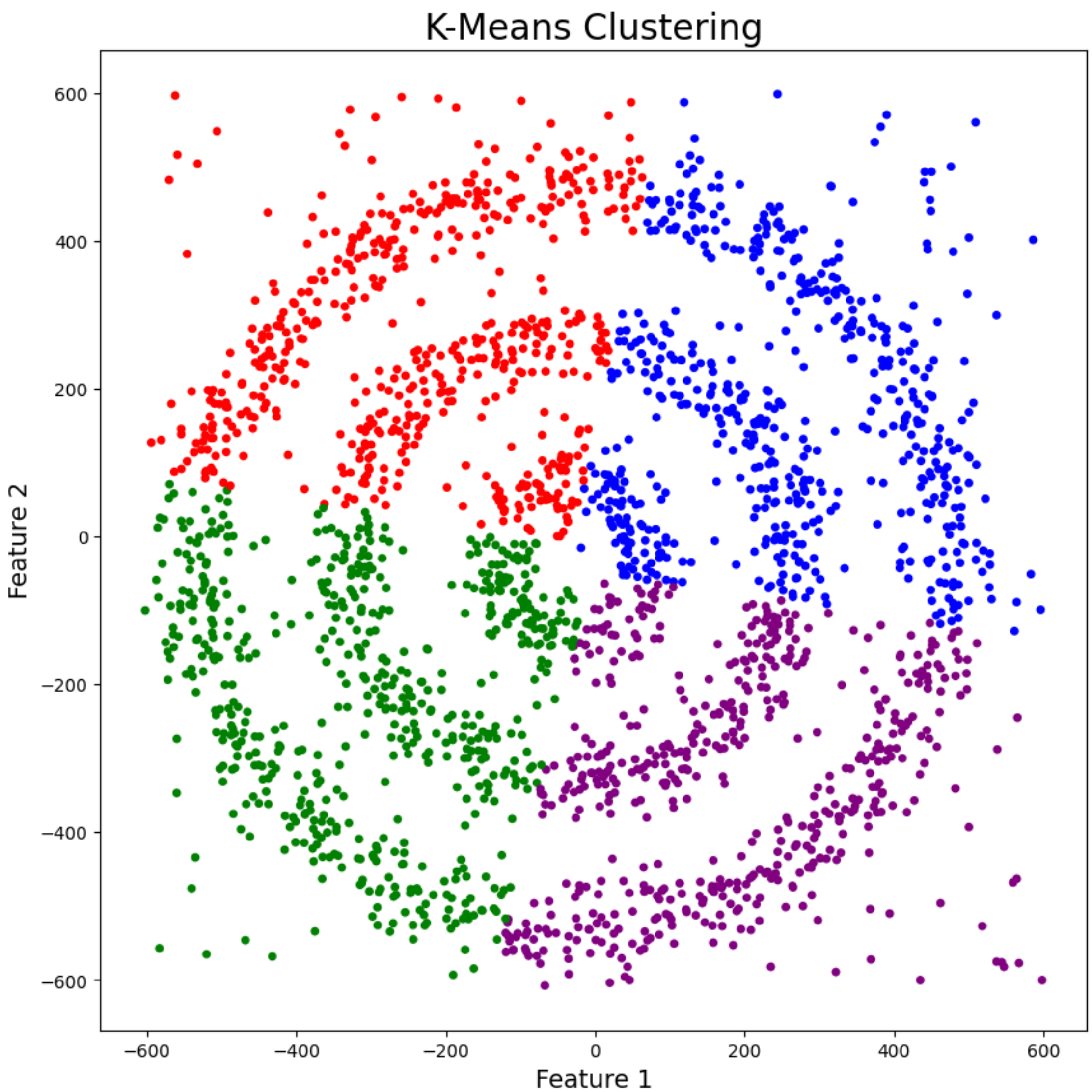


[36]:
```
k_means=KMeans(n_clusters=4,random_state=42)
k_means.fit(df[[0,1]])
```

[36]: KMeans(n_clusters=4, random_state=42)

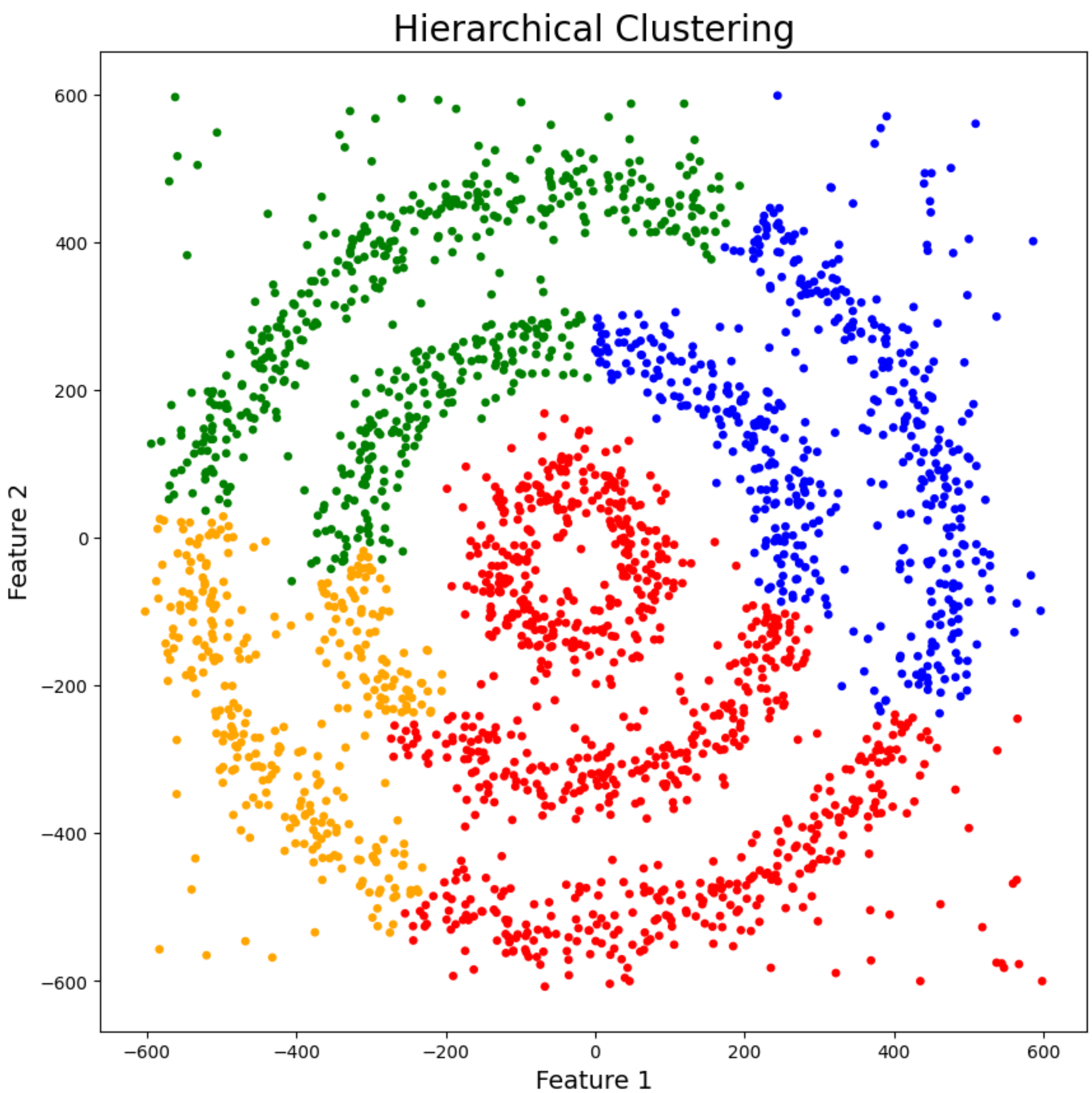
[33]:
```
df['KMeans_labels']=k_means.labels_

# Plotting resulting clusters
colors=['purple','red','blue','green']
plt.figure(figsize=(10,10))
plt.scatter(df[0],df[1],c=df['KMeans_labels'],cmap=matplotlib.colors.
 ↪ListedColormap(colors),s=15)
plt.title('K-Means Clustering',fontsize=20)
```

```
plt.xlabel('Feature 1',fontsize=14)
plt.ylabel('Feature 2',fontsize=14)
plt.show()
```



[39]:
```
# Define colors for up to 4 clusters
colors = ['red', 'green', 'blue', 'orange']

# Clustering
model = AgglomerativeClustering(n_clusters=4, metric='euclidean')
model.fit(df[[0, 1]])
df['HR_labels'] = model.labels_
```

```
# Plot
plt.figure(figsize=(10, 10))
plt.scatter(df[0], df[1], c=df['HR_labels'], cmap=matplotlib.colors.
 ↪ListedColormap(colors), s=15)
plt.title('Hierarchical Clustering', fontsize=20)
plt.xlabel('Feature 1', fontsize=14)
plt.ylabel('Feature 2', fontsize=14)
plt.show()
```



[ ]:
```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=30, min_samples=5)
```

```
dbscan.fit(df[[0,1]])

df['DBSCAN_labels'] = dbscan.labels_

plt.figure(figsize=(10,10))
plt.scatter(df[0], df[1], c=df['DBSCAN_labels'], cmap='tab10', s=15)
plt.title('DBSCAN Clustering', fontsize=20)
plt.xlabel('Feature 1', fontsize=14)
plt.ylabel('Feature 2', fontsize=14)
plt.show()
```