sumary

April 21, 2023

Clustering project JUAN PABLO MONTOYA VALLEJO EVERY POINT OF THIS PROJECT IS DONE IN THIS NOTEBOOK , YOU CAN USE THIS NOTEBOOK AS A SUMMARY OF THE WORK DONE YOU CAN CONTRIBUTE THIS PROJECT BY A PULL REQUEST! FEEL FREE TO USE THIS IN YOUR CONVENIENCE

[43]:
```
import warnings
warnings.filterwarnings('ignore')
```

0.1 1. Research about the Spectral Clustering method, and answer the following questions:

```
a.  In which cases might it be more useful to apply?
b.  What are the mathematical fundamentals of it?
c.  What is the algorithm to compute it?
d.  Does it hold any relation to some of the concepts previously mentioned in class? Which, and
```

0.2 ANSWER 1

a. In which cases might it be more useful to apply?

makes no assumptions on the shapes of clusters

b. What are the mathematical fundamentals of it?

1. preprocessing: construct a similarity graph (e.g. knn graph) for all the data points
2. decomposition: embed data points in a low-dimensional space(spectral embedding) in wich the clusters are more obvious, with the use of the eigenvectors of the graph laplacian and make easier to identify the clusters by other methods
3. grouping: a classical clustering algorith (e.g. kmeans) is applied to partition the embedding

c. What is the algorithm to compute it?

1. make the matrix of laplacian is calculated by the degree matrix and de adjacency matrix

$$L = D - A$$

2. the find the eigevectors and eigenvalues of the laplacian matrix

$$Lv = \Lambda v$$

3. grouping with a clustering method of the preference

$$KMEANS, KMEDOIDS, DBSCAN$$

d. Does it hold any relation to some of the concepts previously mentioned in class? Which, and how?

it uses the eigenvectors and eigenvalues that we saw in SVD, because the need to decompose the matrix,and the you can use kmeans, kmedoids as we saw in class to create the clusters

0.3 2. Research about the DBSCAN method, and answer the following questions:

```
a.  In which cases might it be more useful to apply?
b.  What are the mathematical fundamentals of it?
c.  Is there any relation between DBSCAN and Spectral Clustering? If so, what is it?
```

0.4 ANSWER 2

a. In which cases might it be more useful to apply? when the cluster are near or emmbedded in other cluster or nested clusters trying to connect the points based on 'seing' the density of the points like the human does

b. What are the mathematical fundamentals of it?

1. is based on the density of the records in the area of the matrix starts to see the points near to each point
2. core point is a point that contains equal or more neighbors as defined by the user of DBSCAN to define a cluster
3. non core point is a point that doesnt have as many neighbors as the number set in the algorithm
4. select a random core point as a cluster and add all the core points neighbors to the cluster then add the noncore points that are close to the cluster's core points
5. repeat the process until there are no more core points
6. any remain non core points are asigned as outliers

c. Is there any relation between DBSCAN and Spectral Clustering? If so, what is it?

dbscan and be used in the spectral clustering as a second stage part, you can create and spectral visualization of the clusters and then used dbscan to identify all the clusters. this can improve the dbscan performance

0.5 3. What is the elbow method in clustering? And which flaws does it pose to assess quality?

1 ANSWER 3 Elbow method in clustering

The elbow method is a technique commonly used to determine the optimal number of clusters in a clustering algorithm. It works by plotting the sum of squared distances (also known as the Within-Cluster Sum of Squares or WCSS) of the data points to their closest cluster center for different values of k (the number of clusters). The elbow method suggests that the optimal number of clusters is at the point where the decrease in WCSS starts to level off or form an "elbow" shape.

Flaws of the elbow method

1. The elbow point is subjective: There is no clear or objective way to identify the elbow point on the WCSS curve, and different analysts may have different opinions on the optimal number of clusters.

It assumes linear relationships: The elbow method assumes that the relationship between the number of clusters and WCSS is linear. However, in practice, the relationship may be more complex, and the elbow point may not be clearly visible.

It does not account for cluster validity: The elbow method only considers the WCSS as a measure of clustering quality, but it does not take into account other important aspects such as cluster validity, cluster coherence, or cluster separability.

It can be affected by the initial conditions: The optimal number of clusters obtained using the elbow method may depend on the initial conditions of the clustering algorithm, such as the choice of the initial centroids or the random selection of data points.

2 4. Remember the unsupervised Python package you created in the previous unit? It's time for an upgrade.

a. Implement the k-means module using Python and Numpy
b. Implement the k-medoids module using Python and Numpy
c. Remember to keep consistency with Scikit-Learn API as high as possible

3 ANSWER 4

IN THE REPOSITORY THE MODULES ARE LOCATED IN THE FOLLOWING PATH: Unsupervised/clustering

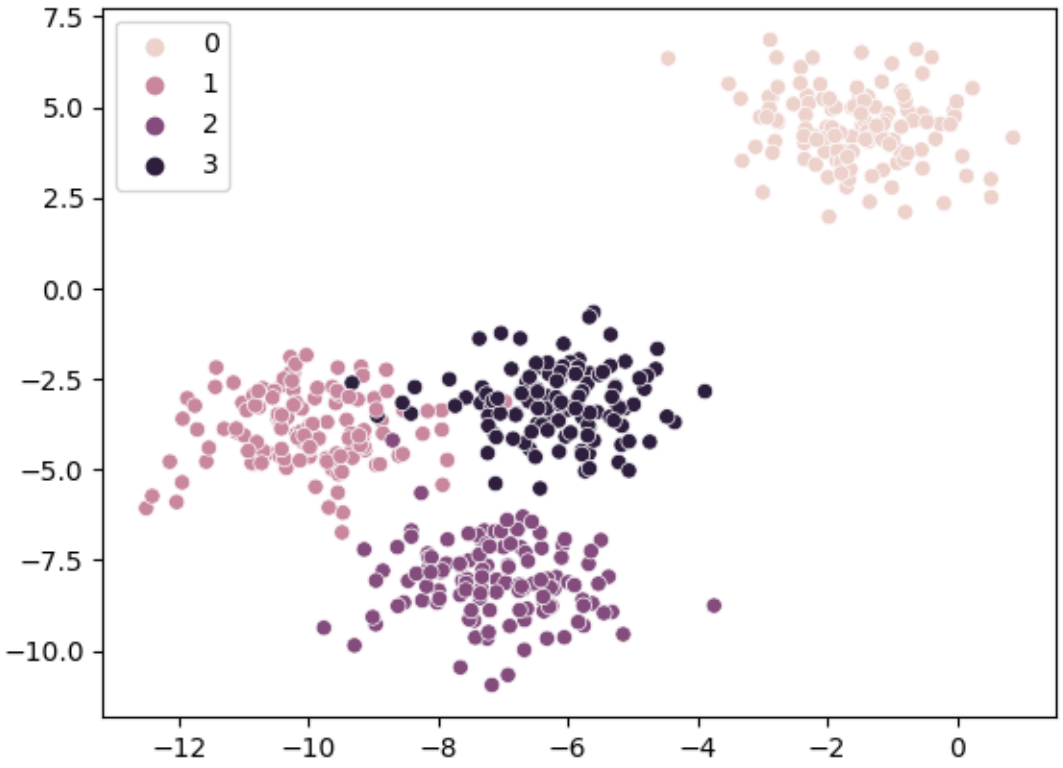
4 5. Let's use the newly created modules in unsupervised to cluster some toy data.

a. Use the following code snippet to create scattered data X from sklearn.datasets import mak
   X, y = make_blobs(n_samples=500, n_features=2, centers=4, cluster_std=1,
   center_box=(-10.0, 10.0), shuffle=True, random_state=1,
)
b. Plot the resulting dataset. How many clusters are there? How far are they from one another
c. For both k-means and k-medoids (your implementations), calculate the silhouette plots and
d. What number of K got the best silhouette score? What can you say about the figures? Is thi

[44]:
```
from sklearn.datasets import make_blobs
X, y = make_blobs( n_samples=500, n_features=2, centers=4, cluster_std=1,
center_box=(-10.0, 10.0), shuffle=True, random_state=1,)
```

[45]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(x=X[:,0], y=X[:,1], hue=y)
```

```
plt.show()
```



[46]:
```
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_samples, silhouette_score
from Unsupervised.clustering.kmeans import KMeans_unsupervised
from Unsupervised.clustering.kmedoids import KMedoids_unsupervised
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import numpy as np

# Generating the sample data from make_blobs
# This particular setting has one distinct cluster and 3 clusters placed close
# together.
X, y = make_blobs(
    n_samples=500,
    n_features=2,
    centers=4,
    cluster_std=1,
    center_box=(-10.0, 10.0),
    shuffle=True,
    random_state=1,
)  # For reproducibility
```

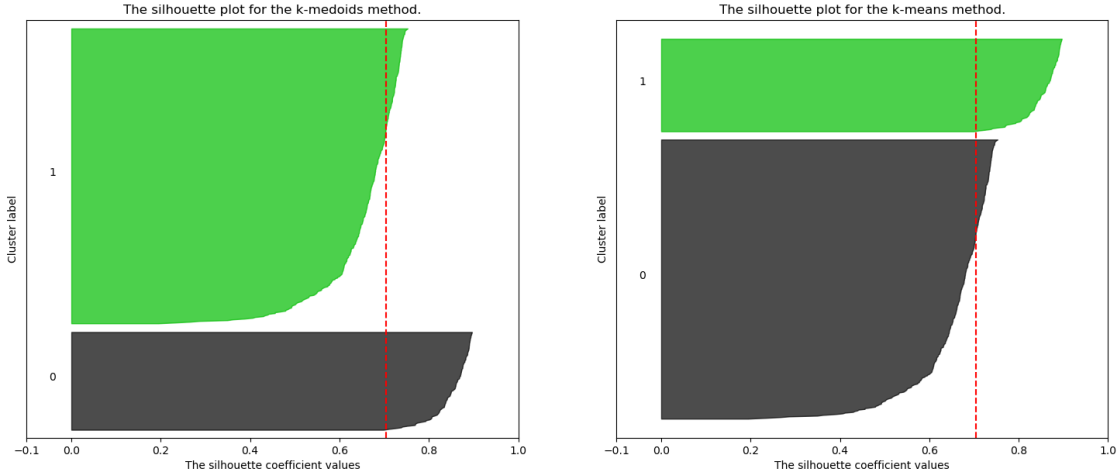
```
range_n_clusters = [2, 3, 4, 5]

for n_clusters in range_n_clusters:
    # Create a subplot with 1 row and 2 columns
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # The 1st subplot is the silhouette plot
    # The silhouette coefficient can range from -1, 1 but in this example all
    # lie within [-0.1, 1]
    ax1.set_xlim([-0.1, 1])
    # The (n_clusters+1)*10 is for inserting blank space between silhouette
    # plots of individual clusters, to demarcate them clearly.
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator
    # seed of 10 for reproducibility.
    clusterer_kmeans = KMeans_unsupervised(K=n_clusters)
    kmeans_cluster_labels = clusterer_kmeans.fit_predict(X)
    clusterer_kmedoids = KMedoids_unsupervised(K=n_clusters)
    kmedoids_cluster_labels = clusterer_kmedoids.fit_predict(X)
    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg_kmeans = silhouette_score(X, kmeans_cluster_labels)
    silhouette_avg_kmedoids = silhouette_score(X, kmedoids_cluster_labels)
    print(
        "For n_clusters =",
        n_clusters,
        "The average silhouette_score for kmeans is :",
        silhouette_avg_kmeans,
        "The average silhouette_score for kmedoids is :",
        silhouette_avg_kmedoids,
    )

    # Compute the silhouette scores for each sample
    sample_silhouette_values_kmeans = silhouette_samples(X,␣
↪kmeans_cluster_labels)
    sample_silhouette_values_kmedoids = silhouette_samples(X,␣
↪kmedoids_cluster_labels)

    y_lower_kmeans = 10
    y_lower_kmedoids = 10
    for i in range(n_clusters):
        # Aggregate the silhouette scores for samples belonging to
        # cluster i, and sort them
```

```
        kmeans_ith_cluster_silhouette_values =␣
↪sample_silhouette_values_kmeans[kmeans_cluster_labels == i]
        kmeans_ith_cluster_silhouette_values.sort()
        kmedoids_ith_cluster_silhouette_values =␣
↪sample_silhouette_values_kmedoids[kmedoids_cluster_labels == i]
        kmedoids_ith_cluster_silhouette_values.sort()


        size_cluster_i_kmeans = kmeans_ith_cluster_silhouette_values.shape[0]
        size_cluster_i_kmedoids = kmedoids_ith_cluster_silhouette_values.
↪shape[0]
        y_upper_kmeans = y_lower_kmeans + size_cluster_i_kmeans
        y_upper_kmedoids = y_lower_kmedoids + size_cluster_i_kmedoids

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(
            np.arange(y_lower_kmeans, y_upper_kmeans),
            0,
            kmeans_ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )
        ax2.fill_betweenx(
            np.arange(y_lower_kmedoids, y_upper_kmedoids),
            0,
            kmedoids_ith_cluster_silhouette_values,
            facecolor=color,
            edgecolor=color,
            alpha=0.7,
        )

        # Label the silhouette plots with their cluster numbers at the middle
        ax1.text(-0.05, y_lower_kmeans + 0.5 * size_cluster_i_kmeans, str(i))

        # Compute the new y_lower for next plot
        y_lower_kmeans = y_upper_kmeans + 10  # 10 for the 0 samples
        # Label the silhouette plots with their cluster numbers at the middle
        ax2.text(-0.05, y_lower_kmedoids + 0.5 * size_cluster_i_kmedoids,␣
↪str(i))

        # Compute the new y_lower for next plot
        y_lower_kmedoids = y_upper_kmedoids + 10  # 10 for the 0 samples

    ax1.set_title("The silhouette plot for the k-medoids method.")
    ax1.set_xlabel("The silhouette coefficient values")
    ax1.set_ylabel("Cluster label")
```

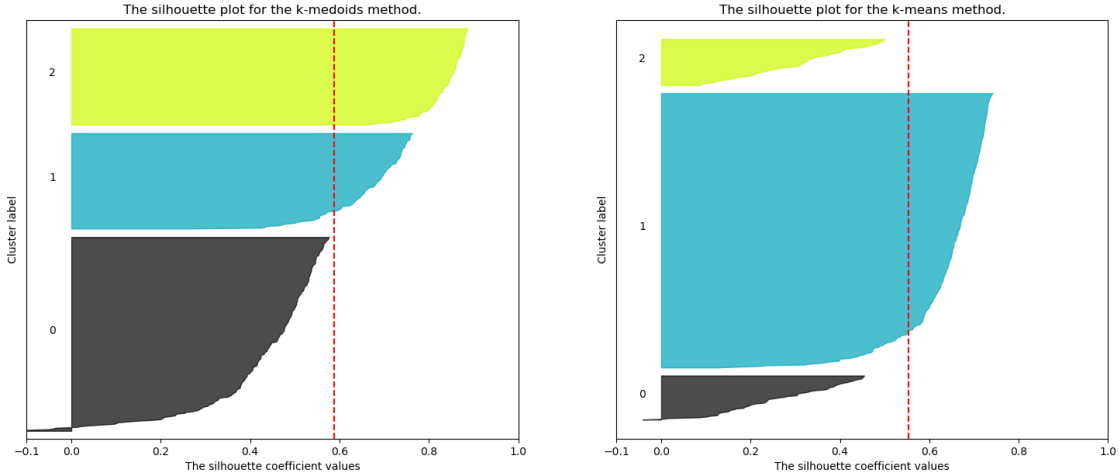
```
    # The vertical line for average silhouette score of all the values
    ax1.axvline(x=silhouette_avg_kmeans, color="red", linestyle="--")

    ax1.set_yticks([])  # Clear the yaxis labels / ticks
    ax1.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])


    ax2.set_title("The silhouette plot for the k-means method.")
    ax2.set_xlabel("The silhouette coefficient values")
    ax2.set_ylabel("Cluster label")

    # The vertical line for average silhouette score of all the values
    ax2.axvline(x=silhouette_avg_kmedoids, color="red", linestyle="--")

    ax2.set_yticks([])  # Clear the yaxis labels / ticks
    ax2.set_xticks([-0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

    plt.suptitle(
        "Silhouette analysis for KMeans clustering on sample data with␣
 ↪n_clusters = %d"
        % n_clusters,
        fontsize=14,
        fontweight="bold",
    )

plt.show()
```

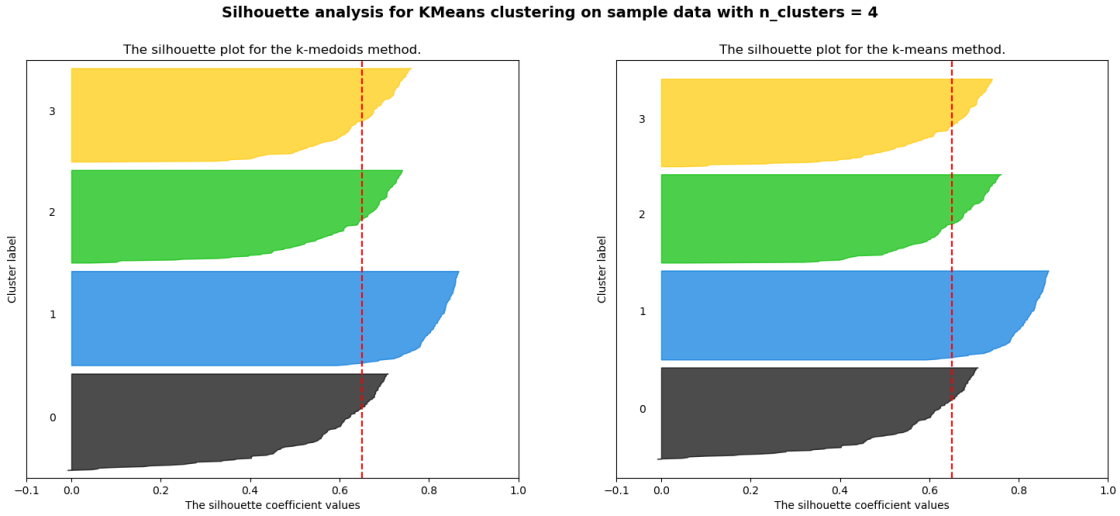
```
For n_clusters = 2 The average silhouette_score for kmeans is :
0.7049787496083262 The average silhouette_score for kmedoids is :
0.7049787496083262
For n_clusters = 3 The average silhouette_score for kmeans is :
0.5882004012129721 The average silhouette_score for kmedoids is :
0.5527698429665097
For n_clusters = 4 The average silhouette_score for kmeans is :
0.6505186632729437 The average silhouette_score for kmedoids is :
0.6505186632729437
For n_clusters = 5 The average silhouette_score for kmeans is :
0.555806786741096 The average silhouette_score for kmedoids is :
0.5750773491409008
```

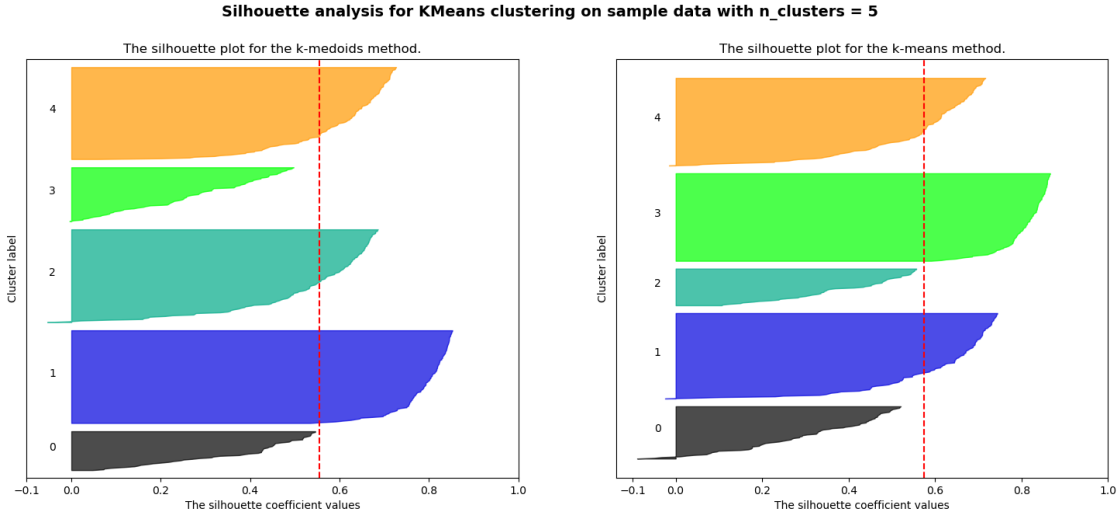
Silhouette analysis for KMeans clustering on sample data with n_clusters = 2



Silhouette analysis for KMeans clustering on sample data with n_clusters = 3







the best distribution overall was for the kmedoids in 2 clusters. that corresponds to the best distribution in the shape of the silouette and this is expected because the fact that 3 of the 4 initial clusters are to close from each other so the next best option would be 4 clusters

5 6. Use the following code snippet to create different types of scattered data:

```
import numpy as np
from sklearn import cluster, datasets, mixture

# ============
# Generate datasets. We choose the size big enough to see the scalability # of the algorithms,
```

```
# ============
n_samples = 500
noisy_circles = datasets.make_circles(n_samples=n_samples, factor=0.5, noise=0.05)
noisy_moons = datasets.make_moons(n_samples=n_samples, noise=0.05) blobs = datasets.make_blobs


# Anisotropically distributed data random_state = 170
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state) transformation = [[
X_aniso = np.dot(X, transformation)
aniso = (X_aniso, y)

# blobs with varied variances
varied = datasets.make_blobs(
n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], random_state=random_state
)
```

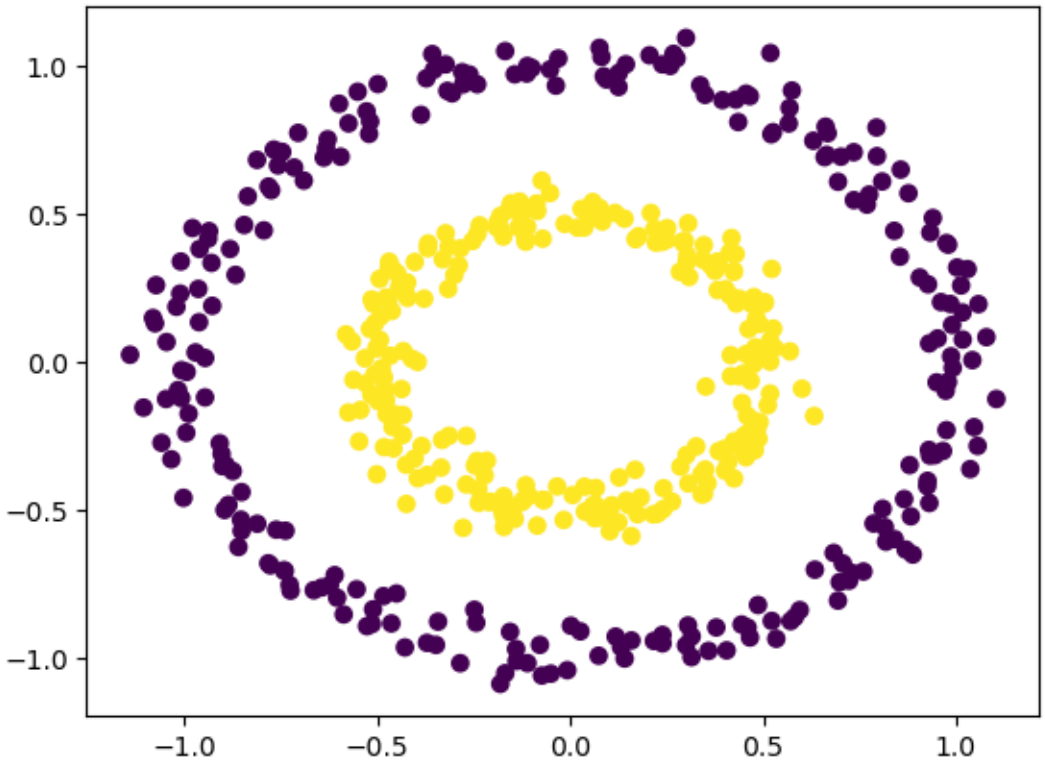
a. Plot the different datasets in separate figures. What can you say about them?

b. Apply k-means, k-medoids, DBSCAN and Spectral Clustering from Scikit-Learn over each datas

[47]:
```
import numpy as np
from sklearn import cluster, datasets, mixture
# ============
# Generate datasets. We choose the size big enough to see the scalability # of␣
 ↪the algorithms, but not too big to avoid too long running times
# ============
n_samples = 500
noisy_circles = datasets.make_circles(n_samples=n_samples, factor=0.5, noise=0.
 ↪05)
noisy_moons = datasets.make_moons(n_samples=n_samples, noise=0.05)
blobs = datasets.make_blobs(n_samples=n_samples, random_state=8)
no_structure = np.random.rand(n_samples, 2), None


# Anisotropically distributed
random_state = 170
X, y = datasets.make_blobs(n_samples=n_samples, random_state=random_state)
transformation = [[0.6, -0.6], [-0.4, 0.8]]
X_aniso = np.dot(X, transformation)
aniso = (X_aniso, y)

# blobs with varied variances
varied = datasets.make_blobs(
n_samples=n_samples, cluster_std=[1.0, 2.5, 0.5], random_state=random_state
)
```

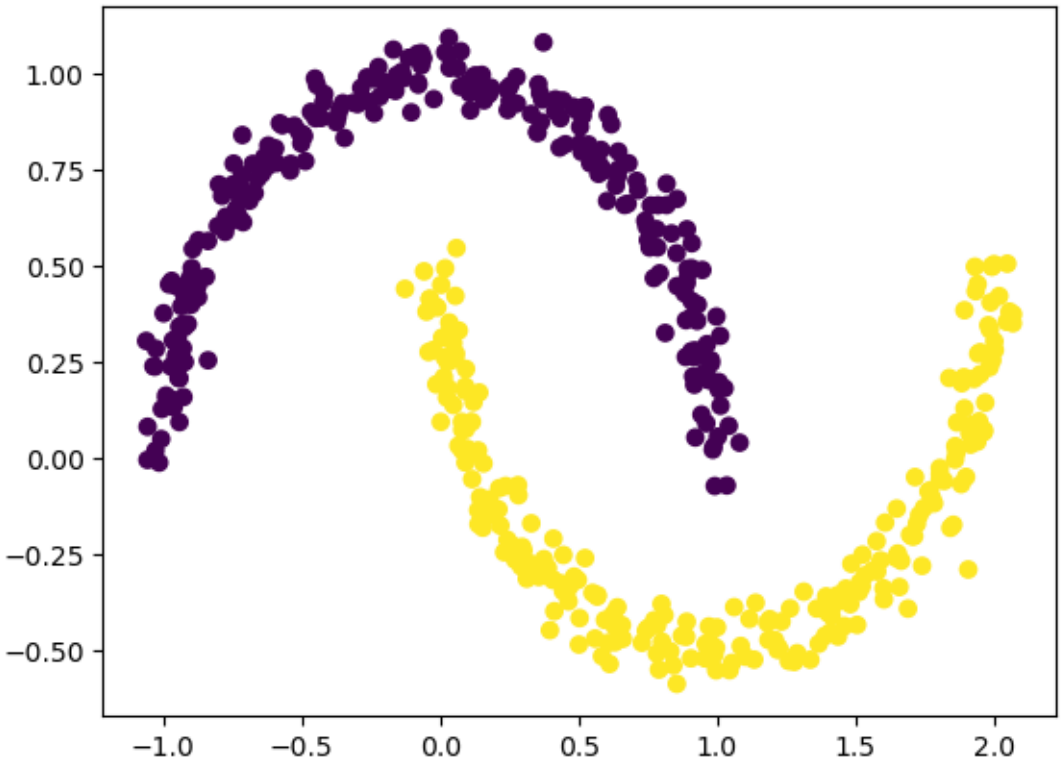
[48]:
```
plt.scatter(noisy_circles[0][:, 0],noisy_circles[0][:, 1], c=noisy_circles[1])
```

[48]:
```
<matplotlib.collections.PathCollection at 0x24746c35f40>
```

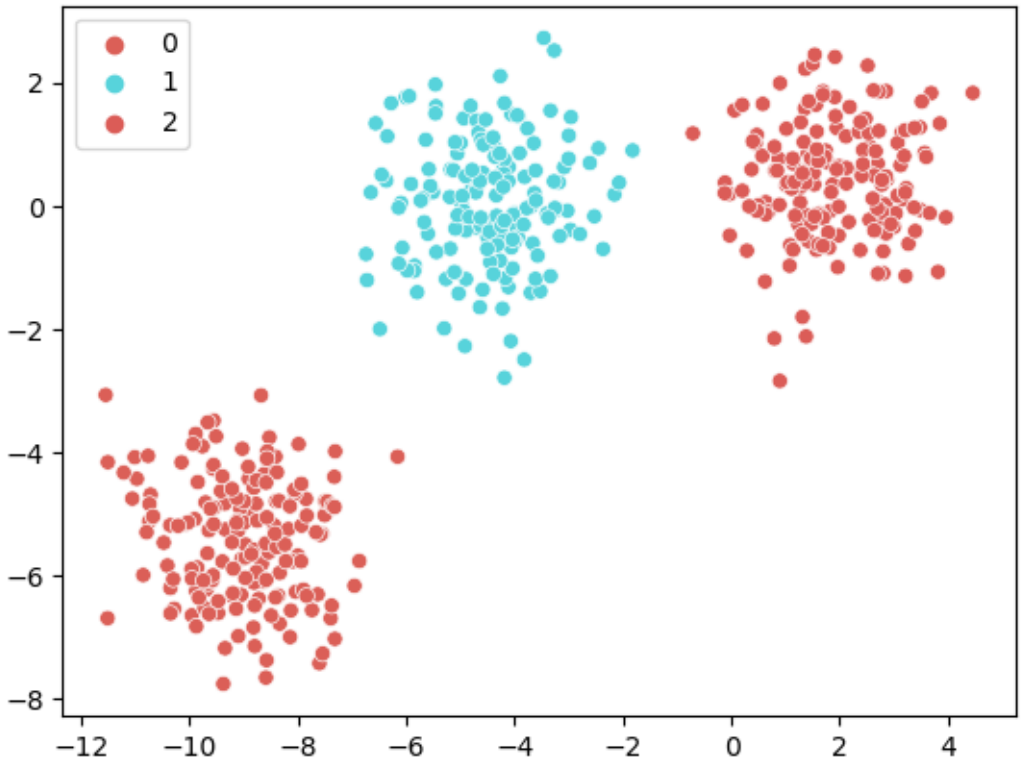


[49]:
```
plt.scatter(noisy_moons[0][:, 0],noisy_moons[0][:, 1], c=noisy_moons[1])
```

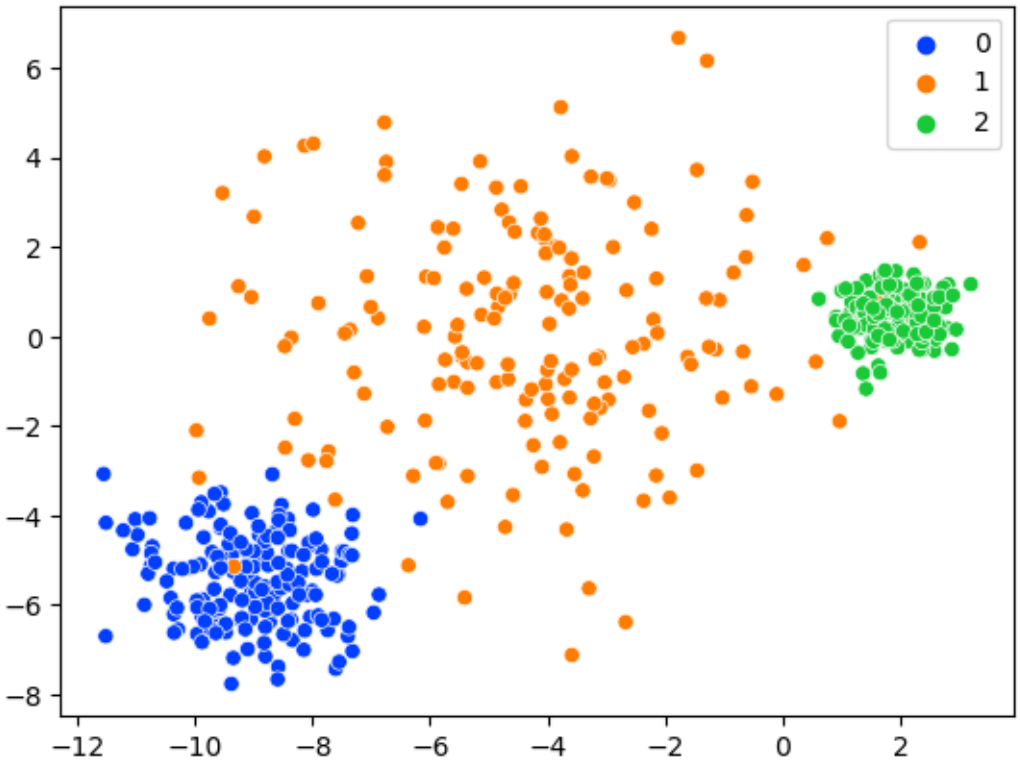
[49]: <matplotlib.collections.PathCollection at 0x24747c993a0>



[50]:
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.scatterplot(x=X[:,0], y=X[:,1], hue=y,palette="hls")
plt.show()
```

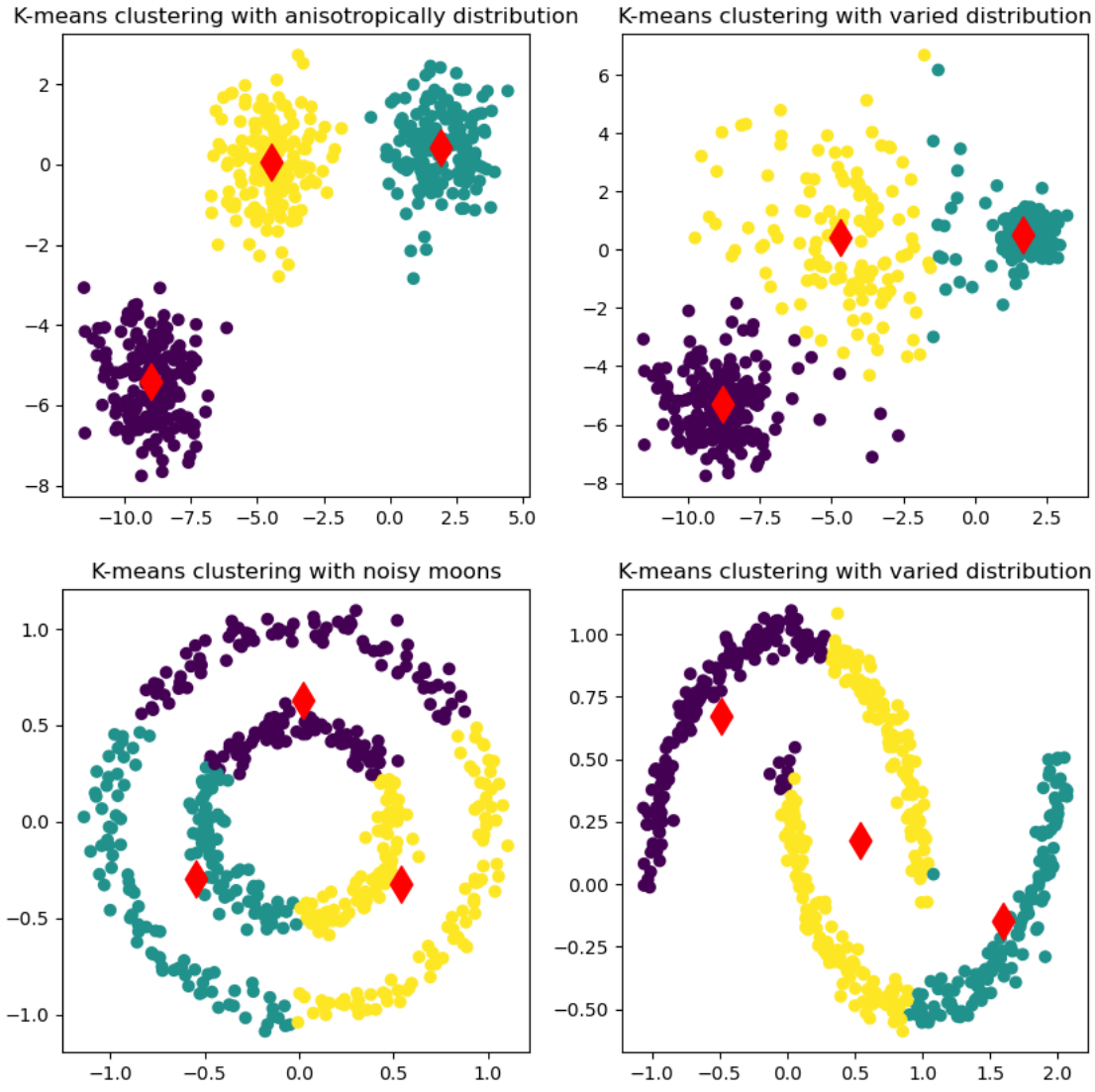


[51]:
```
import seaborn as sns
import matplotlib.pyplot as plt
sns.scatterplot(x=varied[0][:,0], y=varied[0][:,1], hue=varied[1],␣
↪palette='bright')
plt.show()
```



[52]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import DBSCAN
from sklearn.cluster import KMeans


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

fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize=(10, 10))

y_pred_Anisotropically_distributed = kmeans.fit_predict(X)
ax1.scatter(X[:, 0], X[:, 1], c=y_pred_Anisotropically_distributed)
ax1.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
 ↪marker='d', s=200, c='r')
ax1.set_title('K-means clustering with anisotropically distribution')

y_pred_varied = kmeans.fit_predict(varied[0])
ax2.scatter(varied[0][:,0], varied[0][:,1], c=y_pred_varied)
ax2.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
 ↪marker='d', s=200, c='r')
ax2.set_title('K-means clustering with varied distribution')
```

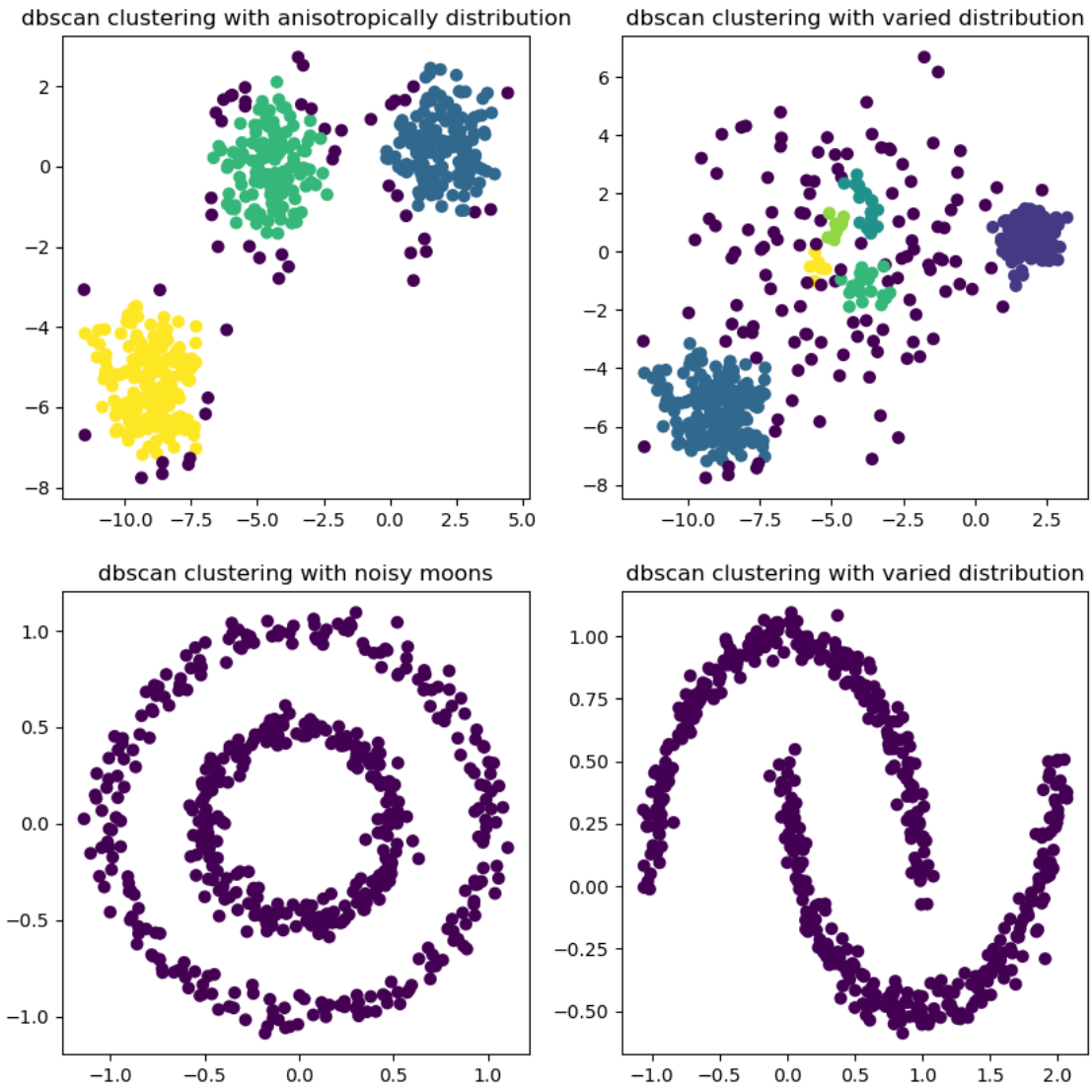
```
y_pred_noisy_circles=kmeans.fit_predict(noisy_circles[0])
ax3.scatter(noisy_circles[0][:,0], noisy_circles[0][:,1],␣
 ↪c=y_pred_noisy_circles)
ax3.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
 ↪marker='d', s=200, c='r')
ax3.set_title('K-means clustering with noisy moons')



y_pred_noisy_moons=kmeans.fit_predict(noisy_moons[0])
ax4.scatter(noisy_moons[0][:,0], noisy_moons[0][:,1], c=y_pred_noisy_moons)
ax4.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],␣
 ↪marker='d', s=200, c='r')
ax4.set_title('K-means clustering with varied distribution')

plt.show()
```



kmeans method has a good performance in linear separation datasets, but when it comes to non linear separation it has a lower performance but acceptable in some cases.

[53]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import DBSCAN


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


fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize=(10, 10))
```

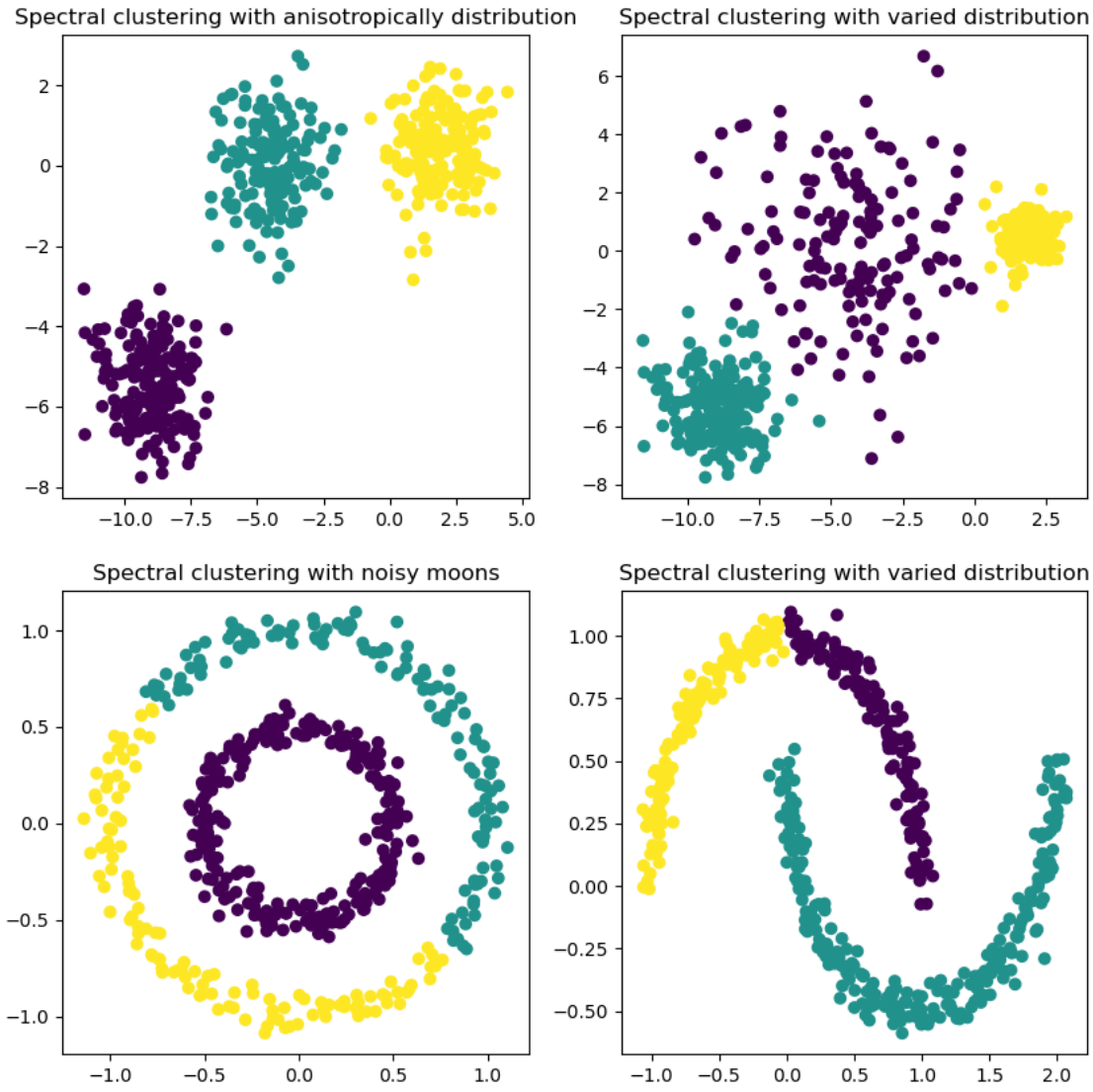
```
dbscan.fit(X)
y_pred_Anisotropically_distributed = dbscan.labels_
ax1.scatter(X[:, 0], X[:, 1], c=y_pred_Anisotropically_distributed)
ax1.set_title('dbscan clustering with anisotropically distribution')

dbscan=DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(varied[0])
y_pred_varied = dbscan.labels_
ax2.scatter(varied[0][:,0], varied[0][:,1], c=y_pred_varied)
ax2.set_title('dbscan clustering with varied distribution')

dbscan=DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(noisy_circles[0])
y_pred_noisy_circles=dbscan.labels_
ax3.scatter(noisy_circles[0][:,0], noisy_circles[0][:,1],␣
 ↪c=y_pred_noisy_circles)
ax3.set_title('dbscan clustering with noisy moons')

dbscan=DBSCAN(eps=0.5, min_samples=5)
dbscan.fit(noisy_moons[0])
y_pred_noisy_moons=dbscan.labels_
ax4.scatter(noisy_moons[0][:,0], noisy_moons[0][:,1], c=y_pred_noisy_moons)
ax4.set_title('dbscan clustering with varied distribution')

plt.show()
```



dbscan method has a good performance in linear separation datasets, but when it comes to non linear separation it has a poor performance

[54]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import SpectralClustering

spectral_clustering = SpectralClustering(n_clusters=3,␣
 ↪affinity='nearest_neighbors', assign_labels='kmeans')


fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize=(10, 10))
```

```
y_pred_Anisotropically_distributed = spectral_clustering.fit_predict(X)
ax1.scatter(X[:, 0], X[:, 1], c=y_pred_Anisotropically_distributed)
ax1.set_title('Spectral clustering with anisotropically distribution')

y_pred_varied = spectral_clustering.fit_predict(varied[0])
ax2.scatter(varied[0][:,0], varied[0][:,1], c=y_pred_varied)
ax2.set_title('Spectral clustering with varied distribution')

y_pred_noisy_circles=spectral_clustering.fit_predict(noisy_circles[0])
ax3.scatter(noisy_circles[0][:,0], noisy_circles[0][:,1],␣
 ↪c=y_pred_noisy_circles)
ax3.set_title('Spectral clustering with noisy moons')



y_pred_noisy_moons=spectral_clustering.fit_predict(noisy_moons[0])
ax4.scatter(noisy_moons[0][:,0], noisy_moons[0][:,1], c=y_pred_noisy_moons)
ax4.set_title('Spectral clustering with varied distribution')

plt.show()
```

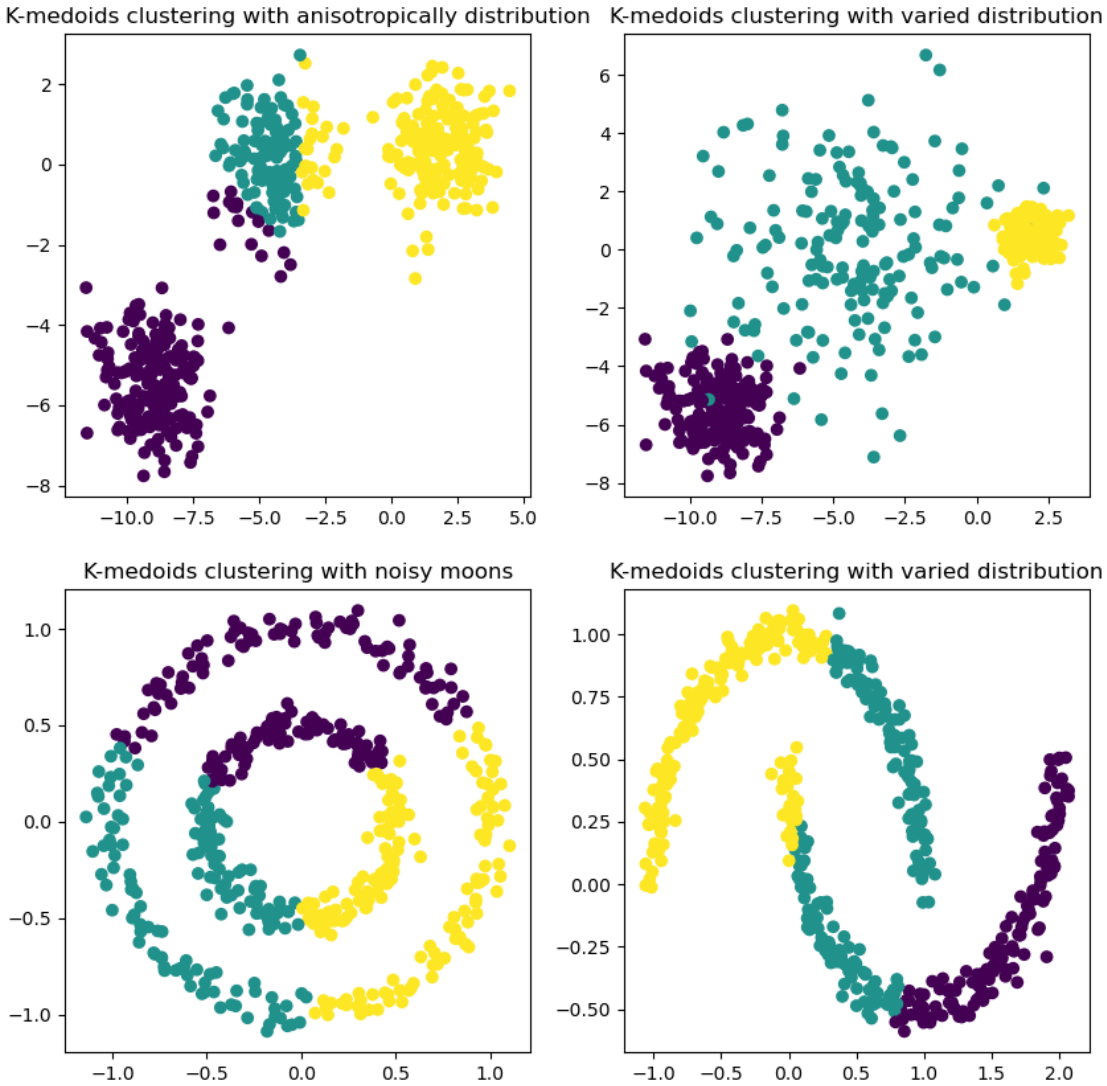


Spectral clustering methods performed well in every case. it is the best method of the 4 with this datasets

[55]:
```
import numpy as np
import matplotlib.pyplot as plt
from pyclustering.cluster.kmedoids import kmedoids




# Initialize KMedoids algorithm
n_clusters = 3
initial_medoids = np.random.choice(X.shape[0], n_clusters, replace=False)
```

```
kmedoids_varied = kmedoids(X, initial_medoids)
kmedoids_Anisotropically_distributed = kmedoids(varied[0], initial_medoids)
kmedoids_noisy_circles = kmedoids(noisy_circles[0], initial_medoids)
kmedoids_noisy_moons = kmedoids(noisy_moons[0], initial_medoids)



# Run KMedoids algorithm
kmedoids_varied.process()
clusters_varied = kmedoids_varied.get_clusters()
medoids_varied = kmedoids_varied.get_medoids()

kmedoids_Anisotropically_distributed.process()
clusters_Anisotropically_distributed = kmedoids_Anisotropically_distributed.
 ↪get_clusters()
medoids_Anisotropically_distributed = kmedoids_Anisotropically_distributed.
 ↪get_medoids()

kmedoids_noisy_circles.process()
clusters_noisy_circles = kmedoids_noisy_circles.get_clusters()
medoids_noisy_circles = kmedoids_noisy_circles.get_medoids()

kmedoids_noisy_moons.process()
clusters_noisy_moons = kmedoids_noisy_moons.get_clusters()
medoids_noisy_moons = kmedoids_noisy_moons.get_medoids()


# Assign cluster labels to each point
y_pred_varied = np.zeros(X.shape[0])
for i, cluster in enumerate(clusters_varied):
    y_pred_varied[cluster] = i

y_pred_Anisotropically_distributed = np.zeros(X.shape[0])
for i, cluster in enumerate(clusters_Anisotropically_distributed):
    y_pred_Anisotropically_distributed[cluster] = i

y_pred_noisy_circles = np.zeros(X.shape[0])
for i, cluster in enumerate(clusters_noisy_circles):
    y_pred_noisy_circles[cluster] = i

y_pred_noisy_moons = np.zeros(X.shape[0])
for i, cluster in enumerate(clusters_noisy_moons):
    y_pred_noisy_moons[cluster] = i


fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize=(10, 10))
```

```
ax1.scatter(X[:, 0], X[:, 1], c=y_pred_Anisotropically_distributed)
ax1.set_title('K-medoids clustering with anisotropically distribution')

ax2.scatter(varied[0][:,0], varied[0][:,1], c=y_pred_varied)
ax2.set_title('K-medoids clustering with varied distribution')

ax3.scatter(noisy_circles[0][:,0], noisy_circles[0][:,1],␣
 ↪c=y_pred_noisy_circles)
ax3.set_title('K-medoids clustering with noisy moons')

ax4.scatter(noisy_moons[0][:,0], noisy_moons[0][:,1], c=y_pred_noisy_moons)
ax4.set_title('K-medoids clustering with varied distribution')

plt.show()
```



kmedoids method has a good performance in linear separation datasets not as accurate as kmeans and spectral clustering but acceptable, but when it comes to non linear separation it has a lower performance but acceptable in some cases.