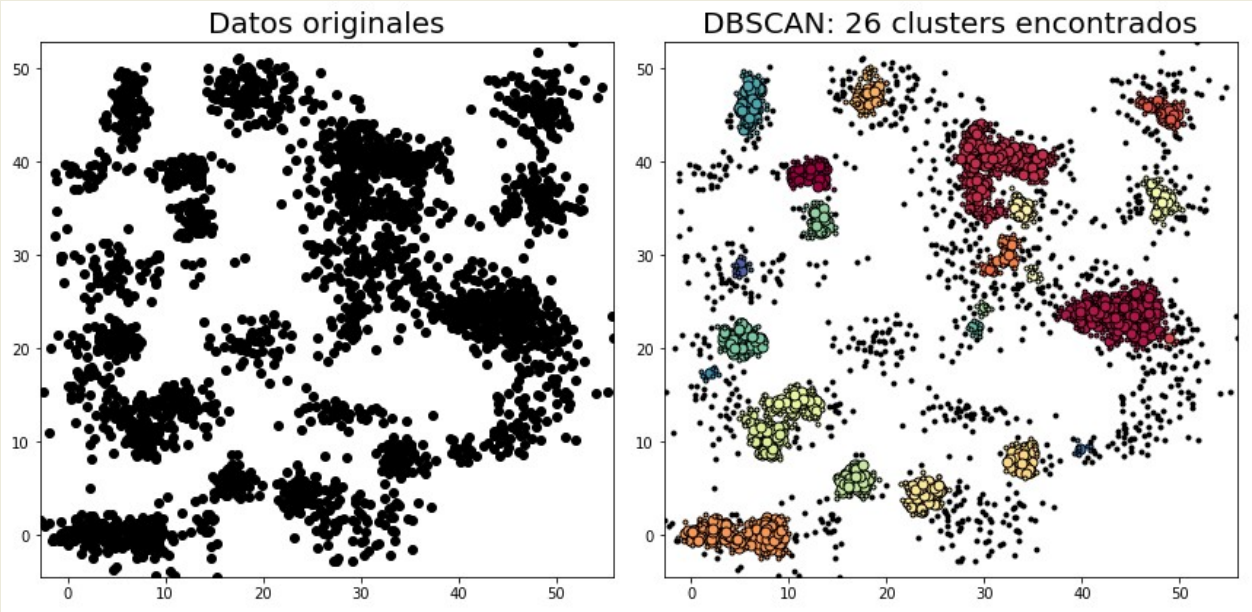
ML_CLUSTERING_DBSCAN_02	Segmentación en clusters utilizando DBScan	ML

En esta práctica se utiliza el algoritmo de clustering DBScan (basado en criterios de densidad). Se define una función para generar de forma aleatoria diferentes nubes de puntos y se aplica el algoritmo DBScan visualizando los resultados con los clusters encontrados.



SOLUCIÓN

Definir las librerías a utilizar

```
# Importar librerías
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
```

Definir una función para generar las nubes de puntos

```
# Definir una función para generar clusters
def cluster_gen(n_clusters, pts_minmax=(10, 100), x_mult=(1, 4), y_mult=(1, 3),
                x_off=(0, 50), y_off=(0, 50)):

    # n_clusters = número de clusters a generar
    # pts_minmax = rango del número de puntos por cluster
    # x_mult = multiplicador para modificar el tamaño del cluster en la dirección 'x'
    # y_mult = multiplicador para modificar el tamaño del cluster en la dirección 'y'
    # x_off = desplazamiento de la posición del cluster en la dirección 'x'
    # y_off = desplazamiento de la posición del cluster en la dirección 'y'
```

```
    # Inicializar listas
    clusters_x = []
    clusters_y = []
    # Generar valores aleatorios para los parámetros
    n_points = np.random.randint(pts_minmax[0], pts_minmax[1], n_clusters)
    x_multipliers = np.random.randint(x_mult[0], x_mult[1], n_clusters)
    y_multipliers = np.random.randint(y_mult[0], y_mult[1], n_clusters)
    x_offsets = np.random.randint(x_off[0], x_off[1], n_clusters)
    y_offsets = np.random.randint(y_off[0], y_off[1], n_clusters)

    # Generar los valores de los clusters en 'x' e 'y'
    for idx, npts in enumerate(n_points):

        xpts = np.random.randn(npts) * x_multipliers[idx] + x_offsets[idx]
        ypts = np.random.randn(npts) * y_multipliers[idx] + y_offsets[idx]
        clusters_x.append(xpts)
        clusters_y.append(ypts)

    # Devolver las posiciones de los clusters
    return clusters_x, clusters_y
```

Generar el conjunto de datos X y modificar el formato de los datos para aplicar DBSCan

```
# Generar algunos conjuntos de datos para aplicar clustering
n_clusters = 50
clusters_x, clusters_y = cluster_gen(n_clusters)

# Convertir los datos en formato 'float32' para utilizar en DBScan
data = np.float32((np.concatenate(clusters_x), np.concatenate(clusters_y))).transpose()
```

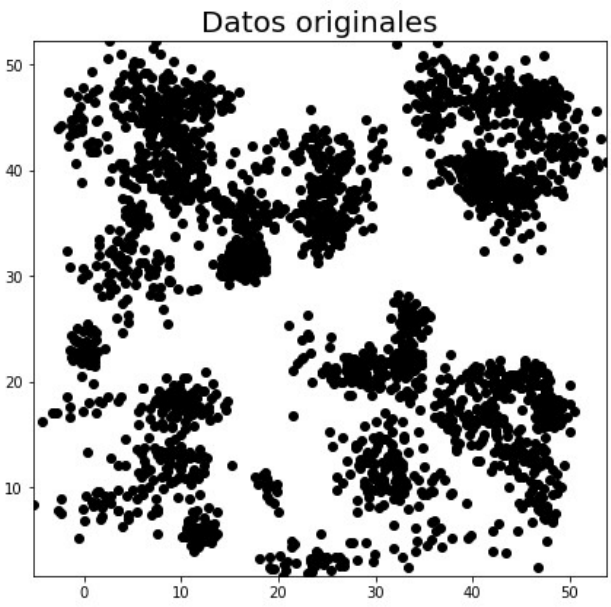
Definir el radio, aplicar DBSCan, determinar etiquetas y el número de clusters resultante

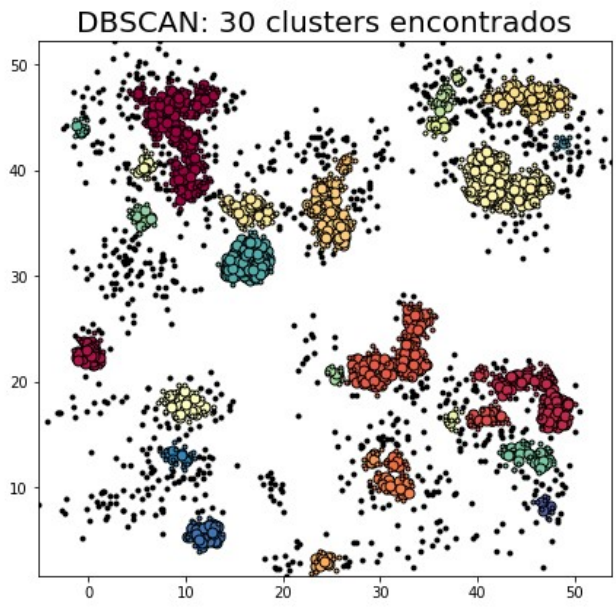
```
# Definir el radio a utilizar
radio = 1
db = DBSCAN(eps=radio, min_samples=10).fit(data)

# Extraer la máscara de los núcleos
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True

# Extraer las etiquetas (-1 para el ruido 'outliers')
labels = db.labels_

# Determinar el número de clusters
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
unique_labels = set(labels)
```

Visualizar los resultados

```
# Visualizar los resultados

# Definir los rangos mínimo y máximo para 'x' e 'y'
min_x = np.min(data[:, 0])
max_x = np.max(data[:, 0])
min_y = np.min(data[:, 1])
max_y = np.max(data[:, 1])

# Definir parámetros del gráfico
fig = plt.figure(figsize=(12,6))
plt.subplot(121)
```

```
plt.plot(data[:,0], data[:,1], 'ko')
plt.xlim(min_x, max_x)
plt.ylim(min_y, max_y)
plt.title('Datos originales', fontsize = 20)




plt.subplot(122)
plt.xlim(min_x, max_x)
plt.ylim(min_y, max_y)
plt.title('DBSCAN: %d clusters encontrados' % n_clusters, fontsize = 20)
fig.tight_layout()
plt.subplots_adjust(left=0.03, right=0.98, top=0.9, bottom=0.05)

# Visualizar los núcleos, bordes y ruido

colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # El color negro se utiliza para el ruido
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    # Núcleos (con tamaño más grande)
    xy = data[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=7)

    # Puntos de borde (con tamaño más pequeño)
    xy = data[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=3)
```


Datos originales
50
40
30
20
10
0
10
20
30
40
50


DBSCAN: 30 clusters encontrados
50
40
30
20
10
0
10
20
30
40
50