PRÁCTICA 2 – GEOMETRÍA COMPUTACIONAL – 2023

PABLO FERNÁNDEZ DEL AMO

1. INTRODUCCIÓN:

El objetivo de estos ejercicios es utilizar técnicas de clustering para agrupar a las personas de la Facultad de Matemáticas en diferentes clústeres o grupos en base a sus niveles de estrés y afición al rock, y así poder determinar patrones en su comportamiento y características. Además, se busca encontrar el número óptimo de clústeres o vecindades de Voronoi utilizando el coeficiente de Silhouette y los algoritmos KMeans y DBSCAN con diferentes métricas y umbrales de distancia. Por último, se analiza a qué grado podrían pertenecer dos personas en base a sus coordenadas y se verifica esta respuesta utilizando la función kmeans.predict. En resumen, el objetivo es explorar diferentes técnicas de clustering para obtener información útil sobre las características de las personas en la Facultad de Matemáticas.

2. MATERIAL USADO:

Los ejercicios utilizan datos de dos archivos: "Personas en la facultad matematicas.txt", que contiene las variables de estado "nivel de estrés" y "afición al rock" para 1500 personas de la Facultad de Matemáticas, y "Grados en la facultad matematicas.txt", que contiene el listado de pertenencia de cada persona a uno de los 4 grados o doble-grados principales de la Facultad de Matemáticas.
Para ello, hemos utilizado diferentes bibliotecas de Python, como sklearn, pandas, numpy, scipy y matplotlib. En particular, hemos explorado el coeficiente de Silhouette y hemos aplicado el algoritmo KMeans y DBSCAN con diferentes parámetros para determinar el número óptimo de clusters y la clasificación resultante. Además, hemos representado gráficamente los resultados obtenidos, incluyendo la visualización del diagrama de Voronoi. Finalmente, para determinar a qué Grado pertenecen las personas con coordenadas a := (0,0) y b := (0,-1) utilizando el algoritmo KMeans y la función predict.

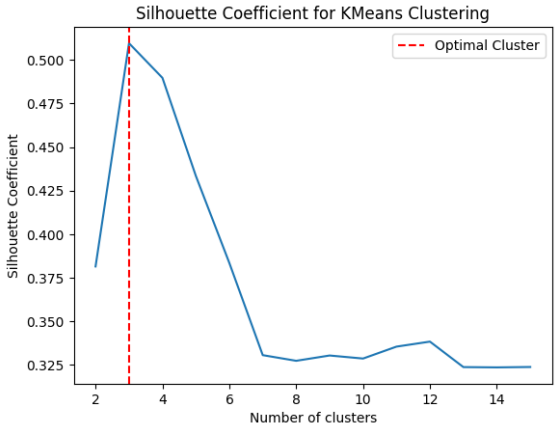
3. RESULTADOS:

i) Obtén el coeficiente s de A para diferente número de vecindades k ∈ {2,3, …, 15} usando el algoritmo KMeans. Muestra en una gráfica el valor de s en función de k y decide con ello cuál es el número óptimo de vecindades. En una segunda gráfica, muestra la clasificación (clústeres) resultante con diferentes colores y representa el diagrama de Voronói en esa misma gráfica

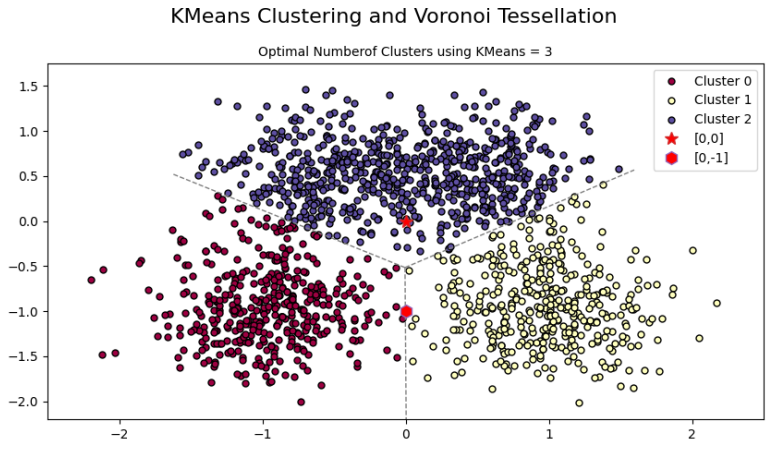
El coeficiente de Silhouette para las vecindades pedidas aparecen en la siguiente lista, siendo el primer elemento de esta el coeficiente para dos vecindades y el último para 15: [0.382, 0.51, 0.49, 0.433, 0.384, 0.331, 0.327, 0.33, 0.329, 0.336, 0.338, 0.324, 0.324, 0.324].

Por tanto, y reforzado por la Gráfica 1, el número óptimo de vecindades es 3.

Esta clasificación en 3 clúster, junto a las celdas de Voronói, son representadas en la Gráfica 2.



Gráfica 1



Gráfica 2

ii) Obtén el coeficiente s para el mismo sistema A usando ahora el algoritmo DBSCAN con la métrica 'euclidean' y luego con 'manhattan'. En este caso, el parámetro que debemos explorar es el umbral de distancia $\in$ (0.1, 0.4), fijando el número de elementos mínimo en $n_0 = 10$. Comparad gráficamente con el resultado del apartado anterior.

Se han calculado el coeficiente s para cada umbral de distancia entre 0.1 y 0.4, empezando en 0.1 y avanzando hasta 0.4 con saltos de 0.01. Al final, en la gráfica 4, se pueden comprobar las diferencias entre cada métrica y cada algoritmo en relación con su número de clústeres y el coeficiente de Silhouette asociado.

La Gráfica 5 expone la relación que existe entre el número de clústeres y el umbral de distancia para ambas métricas.

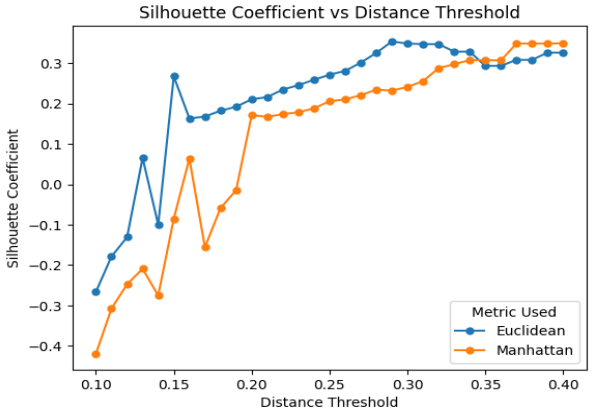
La siguiente lista contiene los coeficientes para cada salto, siendo el primer elemento de esta el umbral 0.1 y el resto los saltos ordenados:

Utilizando la métrica 'euclidean': (Figura 3)

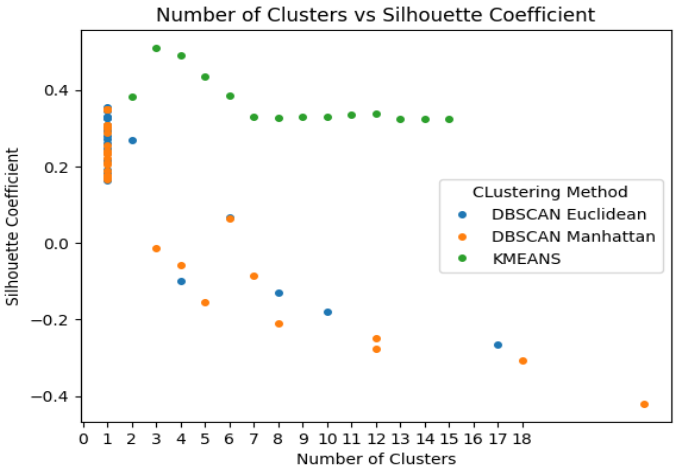
[-0.266, -0.179, -0.131, 0.066, -0.1, 0.267, 0.163, 0.168, 0.182, 0.192, 0.211, 0.216, 0.235, 0.246, 0.259, 0.271, 0.281, 0.301, 0.326, 0.354, 0.349, 0.347, 0.347, 0.329, 0.329, 0.294, 0.294, 0.308, 0.308, 0.326, 0.326]

Utilizando la métrica 'manhattan': (Figura 3)

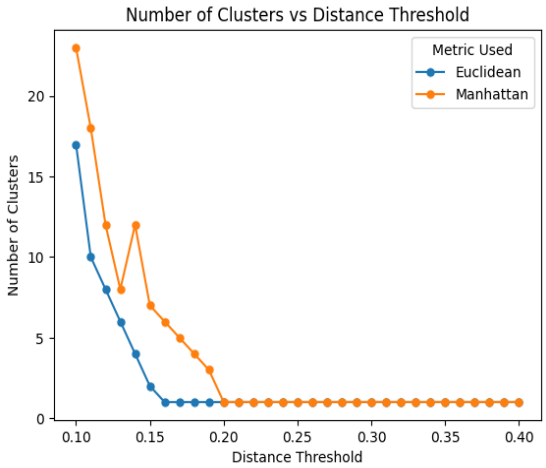
[-0.42, -0.308, -0.248, -0.209, -0.276, -0.085, 0.064, -0.156, -0.059, -0.015, 0.172, 0.167, 0.174, 0.178, 0.188, 0.206, 0.21, 0.221, 0.235, 0.232, 0.241, 0.256, 0.288, 0.298, 0.307, 0.307, 0.307, 0.349, 0.349, 0.349, 0.349]



Gráfica 3



Gráfica 4



Gráfica 5

iii) ¿De qué Grado diríamos que son las personas con coordenadas a := (0, 0) y b := (0, -1)? Comprueba tu respuesta con la función kmeans.predict.

Las dos personas para predecir aparecen como un hexágono y una estrella en la Figura 2. Atendiendo a los clústeres obtenidos visibles en esa gráfica, se determina que la persona a pertenece al clúster 2 y la b al 1. Utilizando kmeans.predict obtenemos el mismo resultado, tal y como se comprueba al correr el código del anexo.

4. CONCLUSIÓN:

En conclusión, en estos ejercicios se aplicaron técnicas de clustering y diagramas de Voronoi para analizar un conjunto de datos de la Facultad de Matemáticas. Se utilizaron las bibliotecas de Python Scikit-learn, Pandas, Numpy, Matplotlib y Scipy. Se determinó el número óptimo de vecindades utilizando el coeficiente de Silhouette y se comparó el rendimiento de dos algoritmos de clustering. Se encontró que el algoritmo KMEANS resultó en una mejor agrupación de datos, calculando con este la pertenencia de dos personas al clúster 1 y 2.. En futuros trabajos se podría aplicar técnicas de reducción de dimensionalidad para mejorar la visualización de los datos y explorar diferentes parámetros para los algoritmos de clustering

5. **ANEXO:**

Para la mejor visualización de estos gráficos y resultados, se recomienda correr el código.
Código de Python utilizado:

```
# PABLO FERNÁNDEZ DEL AMO      GEOMETRÍA COMPUTACIONAL PRÁCTICA 2

import numpy as np
from sklearn.cluster import KMeans, DBSCAN #importing KMeans from scikit-learn library
from sklearn import metrics
from scipy.spatial import ConvexHull, convex_hull_plot_2d #importing ConvexHull and
convex_hull_plot_2d from scipy.spatial library
from scipy.spatial import Voronoi, voronoi_plot_2d #importing Voronoi and voronoi_plot_2d
from scipy.spatial library
import matplotlib.pyplot as plt #importing pyplot from matplotlib library

# ###############################################################################
# Here we define the system X of 1500 elements (people) with two states
archivo1 =
"D:\Documents\MATEMATICAS_UCM\MATEMATICAS_22_23\CUATRI_2\GEOMETRIA_COMPUTACIONAL\Practicas\Pr
actica_2\Personas_en_la_facultad_matematicas.txt" #path to file containing data
archivo2 =
"D:\Documents\MATEMATICAS_UCM\MATEMATICAS_22_23\CUATRI_2\GEOMETRIA_COMPUTACIONAL\Practicas\Pr
actica_2\Grados_en_la_facultad_matematicas.txt" #path to file containing labels
X = np.loadtxt(archivo1) #loading data into numpy array
Y = np.loadtxt(archivo2) #loading labels into numpy array

#If we wanted to standardize the values of the system, we would:
#from sklearn.preprocessing import StandardScaler
#X = StandardScaler().fit_transform(X)


# ###############################################################################
# Clustering using the KMeans algorithm
n_clusters=np.arange(2,16) #defining the range of clusters to test
list_silhouette=[]
for k in n_clusters:
   kmeans=KMeans(n_clusters=k, n_init = 'auto', random_state=0).fit(X) #clustering data using
KMeans algorithm with k clusters
   labels = kmeans.labels_ #assigning labels to each data point
   silhouette = metrics.silhouette_score(X, labels) #computing silhouette score for the
clustering
   print(f'For {k} clusters using KMEANS the Silhouette Coefficient is: {silhouette:.3f}')
#printing silhouette score
   list_silhouette.append(silhouette) #adding silhouette score to a list

optimal_cluster = n_clusters[np.argmax(list_silhouette)]
plt.plot(n_clusters, list_silhouette)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Coefficient')
plt.title('Silhouette Coefficient for KMeans Clustering')
```

```
plt.axvline(x=optimal_cluster, linestyle='--', color='red', label='Optimal Cluster')
plt.legend()
plt.show()

#Random initialization of centroids
#Usamos la inicialización aleatoria "random_state=0"

# ##############################################################################
# Predicting elements to belong to a cluster:
kmeans = KMeans(n_clusters=optimal_cluster, n_init ='auto', random_state=0).fit(X)
#performing KMeans clustering with 3 clusters
problem = np.array([[0, 0], [0, -1]]) #defining two new data points to be classified
clases_pred = kmeans.predict(problem) #predicting the cluster to which each new data point
belongs
print(f'Los clusters usando KMEANS a los que pertenecen la persona con valores [0,0] y [0,-1]
son respectivamente: {clases_pred}') #
vor=Voronoi(X)

# ##############################################################################
# Representamos el resultado con un plot
labels = kmeans.labels_
unique_labels = set(labels)

fig, ax = plt.subplots(figsize=(8, 4))
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(range(len(unique_labels)), colors):
    if k == -1:
        col = [0, 0, 0, 1] # black for noise points
    class_member_mask = (labels == k)
    xy = X[class_member_mask]
    ax.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=5, label = f'Cluster {k}')
    ax.set_title('Fixed number of KMeans clusters: 3' )

# Plot Voronoi diagram
voronoi_plot_2d(vor, ax=ax, show_points=False, show_vertices=False, line_width=1,
line_colors='gray')
for i, reg in enumerate(vor.regions):
    if -1 not in reg:
        polygon = [vor.vertices[j] for j in reg]
        plt.fill(*zip(*polygon),alpha=0.3, edgecolor='none', color=colors[labels[i]])

plt.xlim(vor.min_bound[0] - 0.1, vor.max_bound[0] + 0.1)
plt.ylim(vor.min_bound[1] - 0.1, vor.max_bound[1] + 0.1)
plt.plot(problem[0,0],problem[0,1],'*', markersize=10, markerfacecolor="red", label =
'[0,0]')
plt.plot(problem[0,1],problem[1,1], 'h', markersize=10, markerfacecolor="red", label = '[0,-
1]')

plt.title("KMeans Clustering and Voronoi Tessellation")
plt.legend(loc='best')
plt.show()
```

```python
# Apartado 2:

# define the range of epsilon values to try
epsilon = np.arange(0.1, 0.4, 0.01)

# initialize empty lists to store the results
sil_coeff_eucl = []
sil_coeff_manh = []
nclusters_eucl = []
nclusters_manh = []

# loop through each epsilon value
for i in epsilon:

    # fit DBSCAN models using euclidean and manhattan distance metrics
    dbeuc = DBSCAN(eps=i, min_samples=10, metric='euclidean').fit(X)
    dbmanh = DBSCAN(eps=i, min_samples=10, metric='manhattan').fit(X)

    # find the core samples for each model
    core_samples_mask = np.zeros_like(dbeuc.labels_, dtype=bool)
    core_samples_mask[dbeuc.core_sample_indices_] = True
    core_samples_mask = np.zeros_like(dbmanh.labels_, dtype=bool)
    core_samples_mask[dbmanh.core_sample_indices_] = True

    # extract the cluster labels from each model
    labelseucl = dbeuc.labels_
    labelsmanh = dbmanh.labels_

    # determine the number of clusters for each model, ignoring noise if present
    nclusters_eucl.append(len(set(labelseucl)) - (1 if -1 in labelseucl else 0))
    nclusters_manh.append(len(set(labelsmanh)) - (1 if -1 in labelsmanh else 0))

    # count the number of noise points for each model
    n_noise_ = list(labelseucl).count(-1)
    n_noise_ = list(labelsmanh).count(-1)

    # calculate the silhouette score for each model
    silhouette_eucl = metrics.silhouette_score(X, labelseucl)
    sil_coeff_eucl.append(silhouette_eucl)
    silhouette_manh = metrics.silhouette_score(X, labelsmanh)
    sil_coeff_manh.append(silhouette_manh)
    print(f'DBSCAN: The silhouette coefficient using the euclidean and manhattan metrics  for
the umbral threshold {i:.3f} are:{silhouette_eucl:.3f}, {silhouette_manh:.3f}')


# Plot Silhouette Coefficient
fig, ax = plt.subplots()
ax.plot(epsilon, sil_coeff_eucl, marker='o', markersize=5, label='Euclidean')
ax.plot(epsilon, sil_coeff_manh, marker='o', markersize=5, label='Manhattan')
ax.set_xlabel('Distance Threshold')
ax.set_ylabel('Silhouette Coefficient')
ax.set_title('Silhouette Coefficient vs Distance Threshold')
ax.legend(title='Metric Used')
plt.show()
```

```
# Plot number of clusters
fig, ax = plt.subplots()
ax.plot(epsilon, nclusters_eucl, marker='o', markersize=5, label='Euclidean')
ax.plot(epsilon, nclusters_manh, marker='o', markersize=5, label='Manhattan')
ax.set_xlabel('Distance Threshold')
ax.set_ylabel('Number of Clusters')
ax.set_title('Number of Clusters vs Distance Threshold')
ax.legend(title='Metric Used')
plt.show()

# Plot Clusters vs Silhouette each method:
fig, ax = plt.subplots()
ax.plot(nclusters_eucl, sil_coeff_eucl, 'o', markersize=4,  label = 'DBSCAN Euclidean')
ax.plot(nclusters_manh, sil_coeff_manh, 'o', markersize=4, label = 'DBSCAN Manhattan')
ax.plot(n_clusters, list_silhouette, 'o',  markersize=4, label = 'KMEANS')
ax.legend(title='CLustering Method')
ax.set_xlabel('Number of Clusters')
ax.set_ylabel('Silhouette Coefficient')
ax.set_title('Number of Clusters vs Silhouette Coefficient')
plt.xticks(range(0, 19, 1))
plt.show()

# Find the maximum silhouette coefficient value
optimo = max(sil_coeff_eucl)

# Apply DBSCAN clustering algorithm with epsilon parameter determined from the optimal
silhouette coefficient value
# Set minimum number of samples to be considered a core point to 10
# Use Euclidean distance as the distance metric
db = DBSCAN(eps=epsilon[sil_coeff_eucl.index(optimo)], min_samples=10,
metric='euclidean').fit(X)

# Obtain the labels assigned to each data point by DBSCAN
labels = db.labels_

# Get the unique cluster labels
unique_labels = set(labels)

# Assign a color to each cluster label
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]

# Create a figure to visualize the clusters
plt.figure(figsize=(8, 4))

# For each cluster label, plot the corresponding data points with the assigned color
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Use black color for noise points
        col = [0, 0, 0, 1]

    # Find the data points that belong to the current cluster label
    class_member_mask = (labels == k)
```

```
    # Plot the core data points with a larger marker size
    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=5)

    # Plot the non-core data points with a smaller marker size
    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=3)

# Set the title of the plot
plt.title('Estimated number of optimal DBSCAN clusters: 1')

# Display the plot
plt.show()
```