PRÁCTICA 2:
Diagrama de Voronoi y Clustering

Laura Cano Gómez (U2)

1 de marzo de 2024

1. Introducción

El objetivo de esta práctica es aplicar a un caso práctico lo que hemos aprendido sobre los algoritmos DBSCAN y Kmeans, como técnicas de clustering. Ambos métodos tienen como objetivo separar los datos en grupos con características comunes, para así obtener una información más relevante sobre el problema en cuestión, y poder actuar en consecuencia.

En particular, este trabajo consiste en determinar el número ideal de franjas de un sistema (Villa Laminera) a partir del número óptimo de clusters o vecindades de Voronoi, que se determina con el coeficiente de Silhouette. Todo ello, tanto con el algoritmo Kmeans, como con el algoritmo DBSCAN (con la métrica *euclídea* y la *manhattan*), para comparar gráficamente los resultados.

2. Material utilizado

Los recursos utilizados en este trabajo se han obtenido del campus virtual de la asignatura, en concreto, el fichero *Personas_de_villa_laminera.txt*, del que se obtienen los datos del sistema a estudiar. Contiene dos variables de estado, X_1 = "nivel de estrés" y X_2 = "afición a los dulces", para un conjunto $A = \{a_i, \{X_j\}_{j=1}^{2}\}_{i=1}^{1500}$ de 1500 personas, la población ficticia "Villa Laminera". Además de las plantillas *GCOM2024-practica2_plantilla1*, *GCOM2024-practica2_plantilla2* de las que se han extraído código e ideas para la implementación de esta práctica.

Por otro lado, se han utilizado las librerías de Python *numpy* para estructuras de datos, *sklearn* para el algoritmo K-Means, *matplotlib* para representar los resultados, *scipy.spatial* para los cálculos de las vecindades de Voronoi y *sklearn.cluster* para el algoritmo DBSCAN.

A continuación, se explican las funciones incorporadas:

- **silhouette_coeff()**: Determina el coeficiente de Sihouette para un número entre 2 y 15 de vecindades, y devuelve el número óptimo de vecindades de Voronoi para ese problema.
- **show_Voronoi_cells()**: Relaciona gráficamente el coeficiente de Silhouette con el número (k = 2, ..., 15) de vecindades.
- **clusters_and_Voronoi_Kmeans()**: Representa gráficamente el diagrama de Voronoi para el número k de vecindades óptimo, obtenido previamente con la comparación de coeficientes de Silhouette, para el algoritmo K-means.
- **dbscan_metric()**: Obtiene los coeficientes de Silhouette (con unos Épsilon fijados) para el algoritmo DBSCAN y su determinada métrica (*euclidean* o *manhattan*).
- **compare_euc_man()**: Relaciona gráficamente los coeficientes de Silhouette, obtenidos con la función anterior, para el algoritmo DBSCAN, utilizando la métrica 'euclidean' y 'manhattan' en un intervalo dado de distancias.
- **compare_sil_clus()**: Compara graficamente los coeficientes de Silhouette obtenidos con el algoritmo DBSCAN, con los obtenidos con K-means, en función del número de clusters.
- **compare_sil_clus()**: Compara graficamente los coeficientes de Silhouette obtenidos con el algoritmo DBSCAN, con los obtenidos con K-means, en función del número de clusters.
- **distance_to_centroid()**: Dado un punto, calcula la distancia a cada centroide para determinar a qué cluster pertenece (aquella que sea menor).

- **age_with_predict()**: Obtiene la franja de edad (el cluster) al que pertenece un punto dado utilizando la función *kmeans.predict*.
- **main()**: Devuelve el resultado de los 3 apartados pedidos en la práctica al ejecutar el *.py*, imprimiendo por pantalla los enunciados y sus resultados.

3. Resultados

3.1. Apartado i)

Obtenemos el coeficiente $\bar{s}$ del sistema A para $k \in \{2, 3, ..., 15\}$ vecindades, usando el algoritmo KMeans: [0.4948997184698838, 0.6107764206090893, 0.6002303886854918, 0.4997897657421682, 0.41683732024105485, 0.38050501209709, 0.3877052697872178, 0.37549593770255707, 0.3286374019958081, 0.3292475033907043, 0.33834443071212617, 0.33459069445192996, 0.3312576378633709, 0.3335814148829196]

Representamos gráficamente los datos obtenidos, *Figura 1*, y concluimos que debemos tomar 3 vecindades, ya que es el número con mayor coeficiente de Silhouette. Así mismo, representamos la clasificación (clusters) obtenida junto con el diagrama de Voronoi en la *Figura 2*.

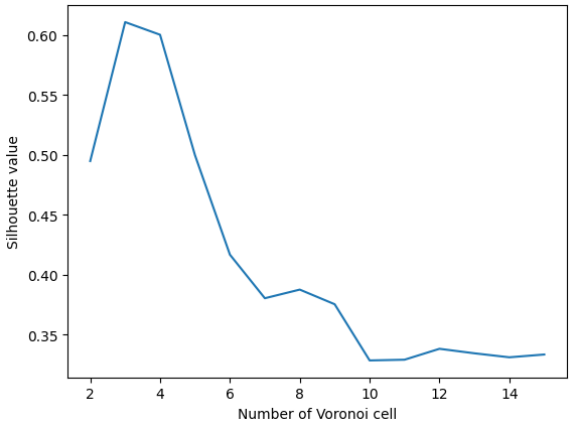

Figura 1: Coef. Silhouette vs. Clusters

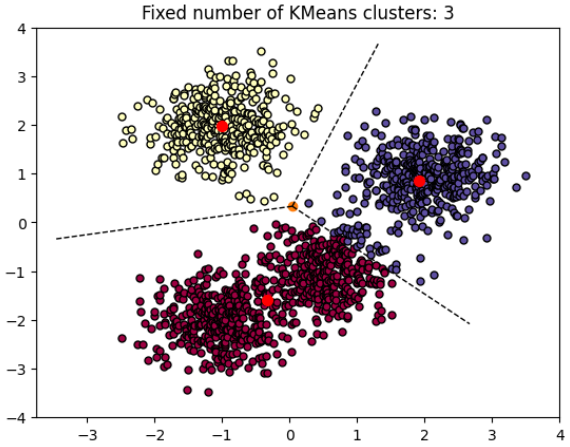

Figura 2: Diagrama de Voronoi con K-means

3.2. Apartado ii)

Obtenemos el coeficiente $\bar{s}$ del sistema A usando el algoritmo DBSCAN con la métrica 'euclidean' y 'manhattan', con un umbral de distancia $\epsilon \in (0.1, 0.4)$ y fijando el número de elementos mínimo en cada cluster a $n_0 = 10$. Comparamos gráficamente el coeficiente de Silhouette en funcion del umbral fijado y obtenemos la *Figura 3*. A continuación, en la *Figura 4*, comparamos gráficamente estos resultados con los obtenidos metiante el algoritmo K-Means.

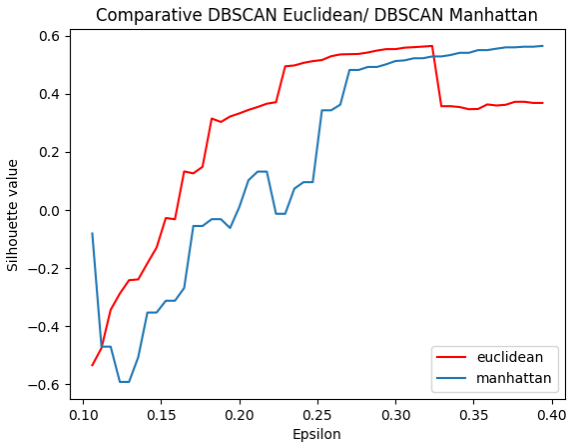

Figura 3: Comparativa DBSCAN Euc. vs. Man.

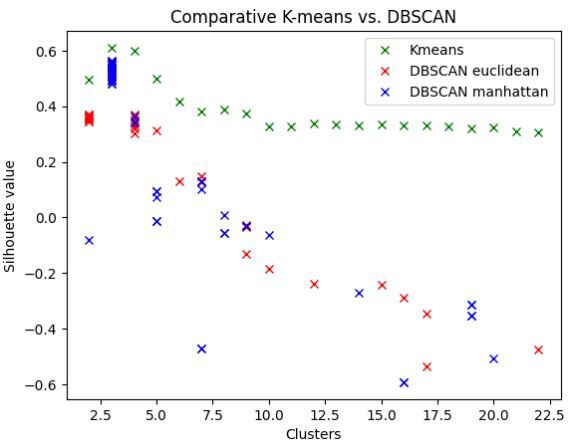

Figura 4: Comparativa K-Means vs. DBSCAN

3.3. Apartado iii)

Calculamos la franja de edad, es decir, el cluster al que pertenecen, las personas con cordenadas $a := (1/2, 0)$ y $b := (0, -3)$ comparando las distancias de los puntos a los centroides y los visualizamos gráficamente en la *Figura 5*. Los puntos dados pertenecerán al centroide más cercano.

El punto $(1/2, 0)$ pertenece al cluster 2, es decir, al morado.
El punto $(0, -3)$ pertenece al cluster 0, es decir, al rojo.

Obteniendo el mismo resultado aplicando la función *kmeans.predict*.

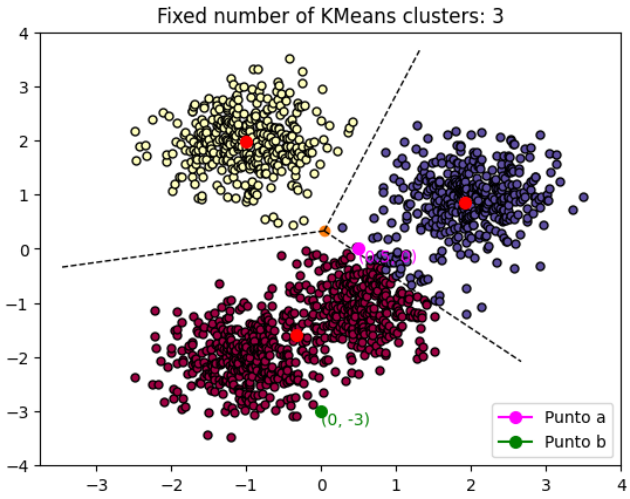

Figura 5: Representación de los puntos a y b

4. Conclusión

Hemos podido comprobar la utilidad de los algoritmos K-means y DBSCAN para realizar estudios de clustering. En este caso práctico, el coeficiente de Silhouette óptimo se alcanza para 3 clusters. Por otro lado, en la *Figura 3* observamos que el coeficiente de Silhouette máximo para la métrica Euclídea se alcanza en el extremo derecho del intervalo solicitado, por tanto, convendría aumentar el intervalo de estudio, o bien aumentar la densidad mínima de cada cluster, con el fin de obtener una información más completa.

5. Anexo con el script/código utilizado

```
"""
Práctica 2. DIAGRAMA DE VORONÓI Y CLUSTERING

Alumna: Laura Cano Gómez
Subgrupo: U2
"""

import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
from sklearn.cluster import DBSCAN


'''
Determina el número ideal de franjas de Villa Laminera (sistema A) a partir del número óptimo
de clusters o vecindades de Voronói. Para ello, utiliza el coeficiente de Silhouette (s), que puede
emplearse directamente desde la librería sklearn:
'''

'''
APARTADO 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 (clusters) resultante con diferentes
```

```
colores y representa el diagrama de Voronoi en esa misma gráfica.
'''

def silhouette_coeff(X, clus_range):
    """
    Determina el coeficiente de Sihouette para k = 2, ..., 15 vecindades, y devuelve el numero
    optimo de vecindades de Voronoi.

    Returns a list object
        list object (silhouette coefficients for k = [clus_range])
        int object

    Arguments:
        X          -> ndarray object
        clus_range -> list object

    """
    coef_s = []
    for k in range(clus_range[0], clus_range[1] + 1):
        kmeans = KMeans(n_clusters= k, random_state= 0).fit(X)  # Inicializamos k aleatoriamente
        labels = kmeans.labels_
        silhouette = metrics.silhouette_score(X, labels)        # Obtiene el c. de Silhouette
        coef_s.append(silhouette)                               # Añade el coef. a la lista

    index = coef_s.index(max(coef_s))                           # Posicion del coef. de Silhouette
                                                                # mayor en la lista

    return [coef_s, index +2]                                   # Empezamos en k = 2 asi que
                                                                # ajustamos el indice

def show_Voronoi_cells(coef_s, clus_range):
    """
    Representa los coeficientes de Silhouette en funcion del numero (k = 2, ..., 15) vecindades.

    Arguments:
        coef_s -> list object (Silhouette values)

    """
    plt.xlabel("Number of Voronoi cell")                            # Representamos graficamente
    plt.ylabel("Silhouette value")
    plt.plot(list(range(clus_range[0], clus_range[1] + 1)), coef_s)

    plt.show()


def clusters_and_Voronoi_Kmeans(X, k, pto1= None, pto2= None):
    """
    Representa graficamente la clasificación (clusters) resultante junto con el diagrama de
    Voronoi.

    Arguments:
        k -> int object (number of clusters)

    """
    kmeans = KMeans(n_clusters= k, random_state= 0).fit(X)
    labels = kmeans.labels_

    unique_labels = set(labels)
    colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]

    plt.figure(figsize=(8,4))                       # Representamos graficamente con las pautas
                                                    # de la Plantilla 1
```

```
    centroids = kmeans.cluster_centers_
    vor=Voronoi(centroids)
    voronoi_plot_2d(vor)

    for i, col in zip(unique_labels, colors):
        if i == -1:
            col = [0, 0, 0, 1]                          # Black used for noise.

        class_member_mask = (labels == i)

        xy = X[class_member_mask]
        plt.plot(xy[ : , 0], xy[ : , 1], 'o', markerfacecolor= tuple(col), \
            markeredgecolor= 'k', markersize= 5)

    plt.xlim(-3.75,4)
    plt.ylim(-4,4)

    plt.title('Fixed number of KMeans clusters: %d' % k)

    # Representamos los centroides:
    for i in range(len(centroids)):
        plt.plot(centroids[i][0], centroids[i][1], 'ro', markersize= 7)

    # Representamos pts concretos si nos lo han pedido (para el apartado iii) )
    if pto1 is not None:
        x = pto1[0]
        y = pto1[1]
        plt.plot(x, y, marker= 'o', markersize = 7, color='magenta', label= "Punto a")
        plt.text(x, y, f'({x}, {y})', verticalalignment='top', horizontalalignment='left',
        color = 'magenta')

    if pto2 is not None:
        x = pto2[0]
        y = pto2[1]
        plt.plot(x, y, marker= 'o', markersize = 7, color='green', label= "Punto b")
        plt.text(x, y, f'({x}, {y})', verticalalignment='top', horizontalalignment='left', \
            color = 'green')

    plt.legend(loc= 'lower right')
    plt.show()


'''
APARTADO 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   (0.1, 0.4), fijando el número de elementos mínimo en n0 = 10.
Comparad gráficamente con el resultado del apartado anterior.
'''

def dbscan_metric(X, metric_type, epsilon, n_min ):
    """
    Obtiene los coef. de Silhouette con el algoritmo DBSCAN
    para una metrica concreta dada.

    Returns
        list object (epsilon distances)
        list object (Silhouette values)

    Arguments:
        metric_type -> string object ('euclidean' / 'manhattan')
        epsilon     -> list object (epsilon range)
        n_min       -> int object (minimum number of elements)
```

```
    """
    distances = []
    coef_s = []
    clusters = [] # Numeros de cluster para cada coef de silouette

    for e in np.linspace(epsilon[0], epsilon[1], 50 + 2)[1:-1]:
        distances.append(e)

        db = DBSCAN(eps= e, min_samples= n_min, metric= metric_type).fit(X)
        core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
        core_samples_mask[db.core_sample_indices_] = True
        labels = db.labels_

        coef_s.append(metrics.silhouette_score(X, labels))

        n_clus = len(set(labels)) - (1 if -1 in labels else 0)
        clusters.append(n_clus)


    return [distances, coef_s, clusters]


def compare_euc_man(dist, sil_euc, sil_man):
    """
    Compara graficamente los coef. de Silhouette obtenidos con el algoritmo DBSCAN
    utilizando la metrica 'euclidean' y  'manhattan', para un intervalo dado de distancias.

    Arguments:
        epsilon -> list object (epsilon range)
        n_min   -> int object (minimum number of elements)

    """
    plt.plot(dist, sil_euc, 'r', label= 'euclidean')
    plt.plot(dist, sil_man, label= 'manhattan')
    plt.legend(loc= 'lower right')

    plt.title('Comparative DBSCAN Euclidean vs. DBSCAN Manhattan')
    plt.xlabel('Epsilon')
    plt.ylabel('Silhouette value')

    plt.show()


def compare_sil_clus(clus_range, sil_kmeans, sil_euc, clusters_euc, sil_man, clusters_man):
    """
    Compara graficamente los coef. de Silhouette obtenidos con el algoritmo DBSCAN
    con los obtenidos con Kmeans, en funcion del numero de clusters

    Arguments:
        epsilon -> list object (epsilon range)
        n_min   -> int object (minimum number of elements)

    """
    plt.plot(list(range(clus_range[0], clus_range[1] + 1)), sil_kmeans, 'gx', label = 'Kmeans')
    plt.plot(clusters_euc, sil_euc, 'rx', label = 'DBSCAN euclidean')
    plt.plot(clusters_man, sil_man, 'bx', label = 'DBSCAN manhattan')
    plt.legend(loc= 'upper right')

    plt.title('Comparative K-means vs. DBSCAN')
    plt.xlabel('Clusters')
    plt.ylabel('Silhouette value')
```

```
    plt.show()



'''
APARTADO iii)
¿De qué franja de edad diríamos que son las personas con coordenadas a := (1/2, 0) y b :=
(0, -3)? Comprueba tu respuesta con la función kmeans.predict.
'''


def distance_to_centroid(X, n_clus, pto):
    """
    Dado un punto, devuelve a que centroide corresponde, es decir, aquel con el que tiene
    una distancia menor.

    Returns an int object

    Arguments:
        X      -> ndarray object
        n_clus -> int object
        a      -> list object
        b      -> list object

    """
    kmeans = KMeans(n_clusters= n_clus, random_state= 0).fit(X)
    centroids = kmeans.cluster_centers_

    distances = []
    for i in centroids:
        distances.append(np.linalg.norm(i-pto))    # Norma del vector Pto-Centroide

    return distances.index(min(distances))


def age_with_predict(X, a, b):
    """
    Obtiene la franja de edad (el cluster) al que pertenece un pto dado utilizando
    la funcion kmeans.predict.

    Returns an array object

    Arguments:
        a -> list object
        b -> list object

    """
    kmeans = KMeans(n_clusters= 3, random_state= 0).fit(X)

    problem = np.array([a, b])
    clases_pred = kmeans.predict(problem)

    return clases_pred



def main():
    archivo1 = r"C:\Users\Documents\Gcom\Practicas\Practica_2\Personas_de_villa_laminera.txt"
    # archivo2 = r"C:\Users\Documents\Gcom\Practicas\Practica_2\Franjas_de_edad.txt"

    X = np.loadtxt(archivo1,skiprows=1)
    # Y = np.loadtxt(archivo2,skiprows=1)
```

```
# Apartado I
print("APARTADO I")
print("- Obtén el coeficiente s de A para diferente número de vecindades k  {2, 3, ..., 15}"
+ "usando el algoritmo KMeans. \n- Muestra en una gráfica el valor de s en función de k y decide"
+ "con ello cuáles el número óptimo de vecindades. \n- En una segunda gráfica, muestra la"
+ "clasificación (clusters) resultante con diferentes colores y representa el diagrama"
+ "de Voronoi en esa misma gráfica.")

clus_range = [2, 15]
[sil_kmeans, better_k] = silhouette_coeff(X, clus_range)
show_Voronoi_cells(sil_kmeans, clus_range)

print(f"S es: {sil_kmeans}")

print(f"El numero optimo de vecindades de Voronoi es aquel en el que el coeficiente de"
+ "Silhouette es mayor, por tanto, segun la gráfica obtenida, debemos tomar {better_k} "
+ "vecindades.\n")

clusters_and_Voronoi_Kmeans(X, better_k)

# Apartado II
print("APARTADO II")
print("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   (0.1, 0.4), fijando el número de elementos "
+ "mínimo en n0 = 10. Comparad gráficamente con el resultado del apartado anterior.")

print(f"\nObtenemos el coeficiente de  Silhouette para el mismo sistema usando el algoritmo "
+ "DBSCAN")
# Nota: las distancias son iguales para ambas metricas
[dist, sil_euc, clusters_euc] = dbscan_metric(X, 'euclidean', [0.1, 0.4], 10)
[dist, sil_man, clusters_man] = dbscan_metric(X, 'manhattan', [0.1, 0.4], 10)

compare_euc_man(dist, sil_euc, sil_man)

clus_range = [2, 22]  # Visualizamos la comparacion para k = 2, ... , 22 clusters
[sil_kmeans, better_k] = silhouette_coeff(X, clus_range)
compare_sil_clus(clus_range, sil_kmeans, sil_euc, clusters_euc, sil_man, clusters_man)

# Apartado III
print("APARTADO III")
print("¿De qué franja de edad diríamos que son las personas con coordenadas a:=(1/2, 0) y "
+ "b:=(0, -3)? Comprueba tu respuesta con la función kmeans.predict.")

pto1 = [1/2, 0]
pto2 = [0, -3]

cluster_pto1= distance_to_centroid(X, better_k, [1/2, 0])
cluster_pto2 = distance_to_centroid(X, better_k, [0, -3])

colors = ['rojo', 'amarillo', 'morado']
print(f"El pto {pto1} pertenece al cluster {cluster_pto1}, es decir, al {colors[cluster_pto1]}.")
print(f"El pto {pto2} pertenece al cluster {cluster_pto2}, es decir, al {colors[cluster_pto2]}.")

print("Representacion grafica:")
clusters_and_Voronoi_Kmeans(X, better_k, pto1, pto2)


print(f"\nComprobamos el resultado con kmeans.predict:")
cluster_predict = age_with_predict(X, pto1, pto2)

print(f"El pto {pto1} pertenece al cluster {cluster_predict[0]}, es decir,"
+ f"al {colors[cluster_predict[0]]}.")
```

```
    print(f"El pto {pto2} pertenece al cluster {cluster_predict[1]}, es decir,"
    + f"al {colors[cluster_predict[1]]}.")


if __name__ == '__main__':
    main()
```