Íñigo Iriondo Delgado
Geometría Computacional

Memoria Práctica 2

(1) Introducción

La práctica consiste en determinar el número ideal de grados de la facultad calculando cuántas vecindades de Voronói hay en la muestra. Lo hacemos encontrando el coeficiente de Silhouette más alto dados un número de clusters entre 2 y 15 para el algoritmo KMeans y umbrales de distancia de 0.1 a 0.4 para DBSCAN utilizando la métrica de Manhattan y la euclídea, y viendo cuántas vecindades obtenemos para esos valores.

(2) Material usado

Programamos en Python usando gran parte de las plantillas del campus virtual. Importamos *numpy*, *KMeans*, *metrics*, *pyplot* y *DBSCAN* y abrimos los archivos del campus virtual que serán nuestras muestras.

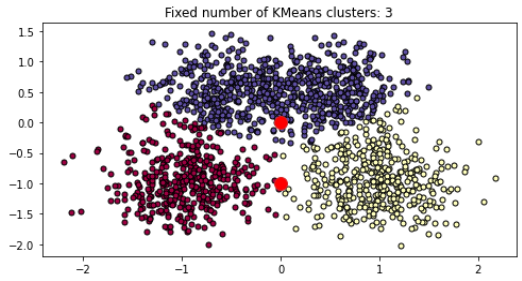
Para el apartado 1 dibujamos el diagrama de Voronói para 3 clusters (después calcularemos que es el número entre 2 y 15 de vecindades más eficiente) con el algoritmo KMeans de la plantilla. También añadimos los dos puntos del tercer apartado en rojo. Entonces sacamos el coeficiente de Silhouette s para cada KMeans de k (entre 2 y 15) clusters y los guardamos en una lista para hacer una gráfica k vs s. La función *maximo* devuelve la posición del mayor elemento de una lista y la usamos para hallar el valor de k para obtener el s máximo.

En el apartado 2 dibujamos los diagramas de Voronói para un umbral de distancia igual a 0.397 y 0.298 (después calcularemos que son los más eficiente de 10 valores de 0.1 a 0.4) con el algoritmo DBSCAN con métrica Manhattan y euclídea de la plantilla, fijando el número de elementos mínimo en 10. De manera análoga al apartado anterior hacemos la gráfica umbral de distancia vs coeficiente de Silhouette.

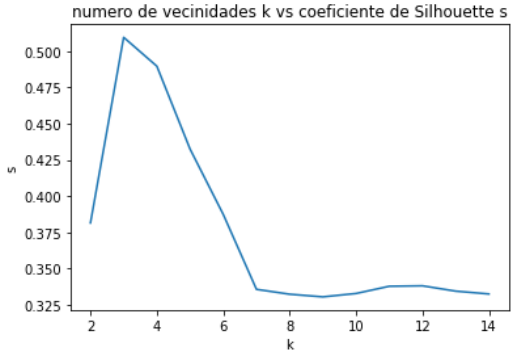
Finalmente en el último apartado utilizamos *kmeans.predict* para obtener la vecindad a la que pertenece cada punto.

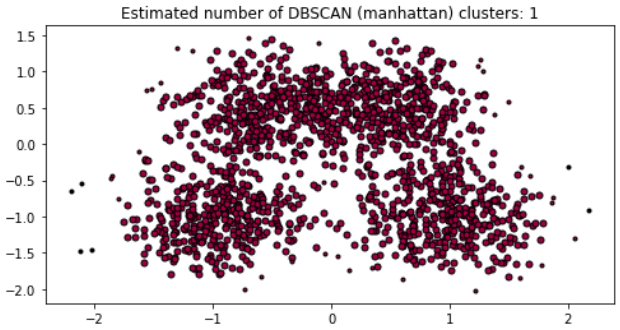
(3) Resultados

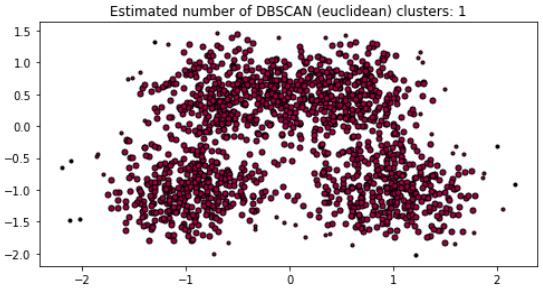
Si ejecutamos el programa obtenemos:



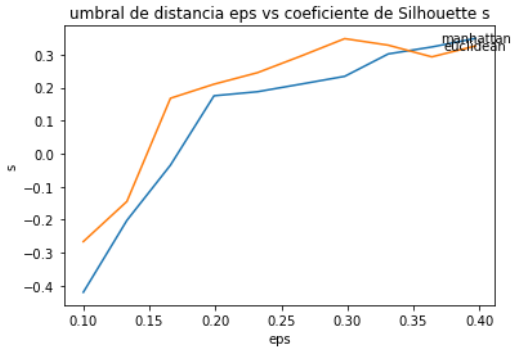
El numero optimo de vecindades es 3 con un coeficiente de Silhouette de 0.51







El umbral de distancia optimo para manhattan DBSCAN es 0.4 con un coeficiente de Silhouette de 0.35
El umbral de distancia optimo para euclidean DBSCAN es 0.3 con un coeficiente de Silhouette de 0.35
El punto a pertenece al cluster 2
El punto b pertenece al cluster 0



(4) Conclusión

En el algoritmo KMeans, el número óptimo entre 2 y 15 de vecindades es 3 con un coeficiente de Silhouette de 0.51, es decir, tendría que haber 3 grados en la facultad.

El umbral de distancia óptimo para DBSCAN con métrica Manhattan es aproximadamente 0.4 y 0.3 para la euclídea, con el mismo coeficiente de Silhouette: 0.35. Dado que el número de elementos mínimo n0 es 10, obtenemos un solo cluster (debería haber un único grado). Si escogieramos un n0 mayor, la gráfica de umbral de distancia vs coeficiente de Silhouette tendría el máximo para un umbral de distancia menor, lo cual probablemente sería más óptimo.

En cuanto al apartado tercero, tenemos que los puntos pertenecen a distintos clusters (en clase te enseñamos que por alguna razón nos sale un número distinto cada vez).

(5) Anexo con el script
Código compartido con David Diez Roshan

```
# -*- coding: utf-8 -*-
"""
Created on Wed Mar  1 18:14:11 2023

@author: Iñigo Iriondo Delgado
"""

#Imports
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN



# Aquí tenemos definido el sistema X de 1500 elementos (personas) con dos estados
archivo1 = "C:/Users/mriri/Personas_en_la_facultad_matematicas.txt"
archivo2 = "C:/Users/mriri/Grados_en_la_facultad_matematicas.txt"
X = np.loadtxt(archivo1)
Y = np.loadtxt(archivo2)
labels_true = Y[:,0]

#Apartado 1

# Los clasificamos mediante el algoritmo KMeans
n_clusters=3

#Usamos la inicialización aleatoria "random_state=0"
kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
labels = kmeans.labels_
silhouette = metrics.silhouette_score(X, labels)

# Predicción de elementos para pertenecer a una clase:
problem = np.array([[0, 0], [0, -1]])

# Representamos el resultado con un plot
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]

plt.figure(figsize=(8,4))
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
```

```
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

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

plt.plot(problem[:,0],problem[:,1],'o', markersize=12, markerfacecolor="red")

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

#Calculamos el coeficiente de Silhouette para cada numero de vecindades
silhouette_list = [0]*13
k_list = [0]*13
for i in range(2,15):
    k_list[i-2] = i
    n_clusters = i
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
    labels = kmeans.labels_
    silhouette_list[i-2] = metrics.silhouette_score(X, labels)
plt.plot(k_list,silhouette_list)
plt.xlabel('k')
plt.ylabel('s')
plt.title('numero de vecinidades k vs coeficiente de Silhouette s')

def maximo(lista):
    for i in range(len(lista)):
        if lista[i] == max(lista):
            return i + 2

print("El numero optimo de vecindades es",maximo(silhouette_list),"con un coeficiente de Silhouette
de",round(max(silhouette_list),2))


#Apartado 2

# Los clasificamos mediante el algoritmo manhattan DBSCAN
epsilon = 0.397

db = DBSCAN(eps=epsilon, min_samples=10, metric='manhattan').fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
```

```
n_noise_ = list(labels).count(-1)

# Representamos el resultado con un plot
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
        for each in np.linspace(0, 1, len(unique_labels))]

plt.figure(figsize=(8,4))
for k, col in zip(unique_labels, colors):
   if k == -1:
      # Black used for noise.
      col = [0, 0, 0, 1]

   class_member_mask = (labels == k)

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

   xy = X[class_member_mask & ~core_samples_mask]
   plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
         markeredgecolor='k', markersize=3)

plt.title('Estimated number of DBSCAN (manhattan) clusters: %d' % n_clusters_)
plt.show()

# Los clasificamos mediante el algoritmo euclidean DBSCAN
epsilon = 0.298

db = DBSCAN(eps=epsilon, min_samples=10, metric='euclidean').fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

# Representamos el resultado con un plot
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
        for each in np.linspace(0, 1, len(unique_labels))]

plt.figure(figsize=(8,4))
for k, col in zip(unique_labels, colors):
   if k == -1:
      # Black used for noise.
      col = [0, 0, 0, 1]
```

```
    class_member_mask = (labels == k)

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

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=3)

plt.title('Estimated number of DBSCAN (euclidean) clusters: %d' % n_clusters_)
plt.show()

#Calculamos el coeficiente de Silhouette para 10 umbrales de distancia en [0.1,0.4]
silhouette_list1 = [0]*10
epsilon_list = [0]*10
for i in range(10):
    epsilon_list[i] = 0.1 + i*0.033
    epsilon = epsilon_list[i]
    db = DBSCAN(eps=epsilon, min_samples=10, metric='manhattan').fit(X)
    core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
    core_samples_mask[db.core_sample_indices_] = True
    labels = db.labels_
    silhouette_list1[i] = metrics.silhouette_score(X, labels)
plt.plot(epsilon_list,silhouette_list1)
plt.text(epsilon_list[-1], silhouette_list1[-1], 'manhattan', ha='center', va='center')
plt.xlabel('eps')
plt.ylabel('s')
plt.title('umbral de distancia eps vs coeficiente de Silhouette s')

silhouette_list2 = [0]*10
epsilon_list = [0]*10
for i in range(10):
    epsilon_list[i] = 0.1 + i*0.033
    epsilon = epsilon_list[i]
    db = DBSCAN(eps=epsilon, min_samples=10, metric='euclidean').fit(X)
    core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
    core_samples_mask[db.core_sample_indices_] = True
    labels = db.labels_
    silhouette_list2[i] = metrics.silhouette_score(X, labels)
plt.plot(epsilon_list,silhouette_list2)
plt.text(epsilon_list[-1], silhouette_list2[-1], 'euclidean', ha='center', va='center')

def maximo(lista):
    for i in range(len(lista)):
        if lista[i] == max(lista):
            return epsilon_list[i]
```

```
print("El umbral de distancia optimo para manhattan DBSCAN
es",round(maximo(silhouette_list1),2),"con un coeficiente de Silhouette
de",round(max(silhouette_list1),2))

print("El umbral de distancia optimo para euclidean DBSCAN
es",round(maximo(silhouette_list2),2),"con un coeficiente de Silhouette
de",round(max(silhouette_list2),2))


#Apartado 3

kmeans = KMeans(n_clusters=3)

kmeans.fit(X)

apred = kmeans.predict(np.array([[0, 0]]))
bpred = kmeans.predict(np.array([[0, -1]]))

print("El punto a pertenece al cluster", apred[0])
print("El punto b pertenece al cluster", bpred[0])
```