MEMORIA PRÁCTICA 2 - GCOM: Diagrama de Voronói y Clustering

Nombre: Gabriel Alba Serrano
Subgrupo: U1

1. Introducción

La clasificación de grupos ó clustering de un sistema consiste en agrupar objetos por similitud, en grupos o conjuntos de manera que los miembros del mismo grupo tengan características similares. El algoritmo apropiado para dicha clasificación depende del conjunto de datos que se analiza y el uso que se le dará a los resultados.
La idea de un grupo de datos similares resulta incompleta y subjetiva, y por ello existen miles de algoritmos. Alguno de los modelos más conocidos se basan en la conectividad, centroides, distribución, densidad, etc. En esta práctica vamos a estudiar dos algoritmos, uno basado en agrupamiento por centroides (KMeans) y otro basado en agrupamiento por densidad (DBSCAN).
La principal característica del algoritmo KMeans es que clasifica en clusters de aproximadamente tamaño similar (con un número similar de elementos), debido a que siempre asignará cada elemento al centroide más cercano, según la correspondiente métrica. Esto a menudo provoca cortes incorrectos en los bordes de los grupos, ya que el algoritmo optimiza los centroides, no las fronteras. Los clusters en este caso se denominan Diagramas de Voronói.
Por otro lado la principal característica del DBSCAN es que los clusters están definidos como áreas de densidad más alta que en el resto del conjunto de datos. Los objetos en áreas esparcidas son conocidos como ruido o puntos frontera. Este algoritmo no proporciona un buen clustering en un sistema que no tiene bajadas de densidad de sus elementos, es decir, un sistema con una densidad similar en todas sus regiones.

2. Material usado

2.1. Método

Utilizando el software de python3 y su librería sklearn, hemos aplicado los algoritmos de clustering KMeans y DBSCAN a un sistema. Además hemos graficado los diagramas de Voronói de dicho sistema de acuerdo con la clasificación más óptima del algoritmo KMeans, es decir aquella con el coeficiente de Silhouette más próximo al valor de 1.

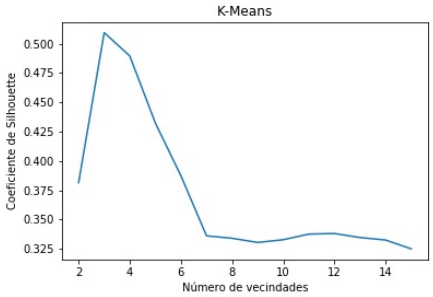
2.2. Datos

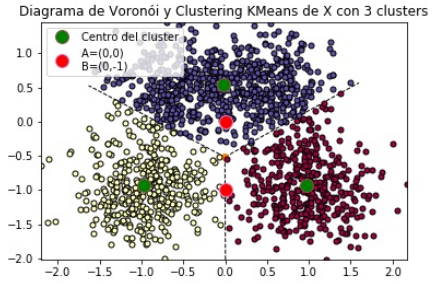
- Los archivos de texto Grados_en_la_facultad_matematicas.txt y Personas_en_la_facultad_matematicas.txt, siendo el primero un sistema de 1500 elementos en el que cada elemento tiene dos estados: X_1='nivel de estrés' y X_2='afición al rock'. Vamos a aplicar los distintos algoritmos de clustering a dicho sistema que denominaremos como X.
- Las plantillas 1 y 2 de Python que aplican un ejemplo de KMeans con métrica euclidiana y otro ejemplo de DBSCAN con métrica de Manhattan, respectivamente.

3. Resultados

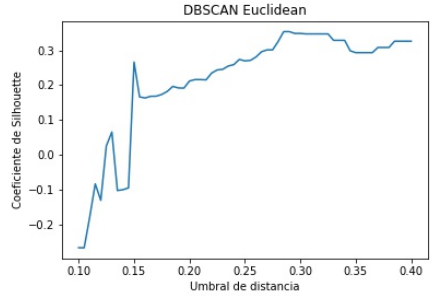
1. Hemos calculado el coeficiente de Silhouette para cada k $\in \{2, 3, ..., 15\}$, siendo k el número de vecindades (clusters) que se utiliza para aplicar el algoritmo KMeans. Para k = 3 , el coeficiente

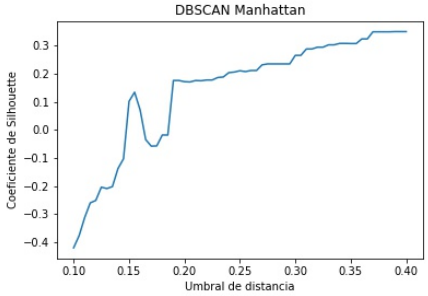
de Silhouette vale 0.510 que es el valor más próximo a 1 de todos los calculados, por lo tanto la clasificación KMeans del sistema X con 3 clusters es la más óptima.

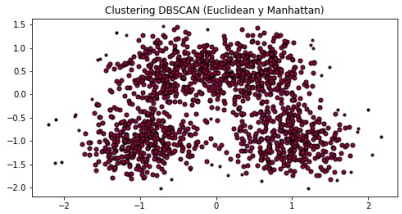




2. Además hemos calculado el coeficiente de Silhouette al aplicar el algoritmo de DBSCAN, primero con métrica euclidiana y luego con métrica de Manhattan. Ahora variando el umbral de distancia $\epsilon \in (0.1,\ 0.4)$ y fijando el número mínimo de elementos $n_0 = 10$. Con métrica 'Euclidean' el umbral de distancia óptimo es $\epsilon = 0.285$ y con métrica 'Manhattan' $\epsilon = 0.390$, es decir, para estos valores, sus respectivos coeficientes de Silhouette son los más altos ($s \approx 0.350$, con ambas métricas). Cabe destacar que para ambas métricas y con el umbral de distancia óptimo hemos obtenido que la clasificación de los elementos del sistema X a través del algoritmo DBSCAN ha sido de en un único cluster.







3. Utilizando la función *kmeans.predict* hemos obtenido que los puntos $a := (0, 0)$ y $b := (0, -1)$ pertenecen a los clusters 2 y 0, respectivamente, generados al aplicar el algoritmo KMeans con 3 clusters ó vecindades.

4. Conclusión

Observamos que el algoritmo KMeans clasifica los elementos de X en 3 clusters, frente al algoritmo DBSCAN que con ambas métricas (Euclidean y Manhattan) clasifica los elementos en un único cluster, es decir, no hace ninguna clasificación de los elementos del sistema, esto se debe a que el sistema X no sufre bajadas de densidad respecto a sus elementos. También hay que destacar que el valor óptimo del coeficiente de Silhouette de KMeans es mayor que el valor óptimo de DBSCAN.
Por lo tanto la clasificación del algoritmo KMeans es más apropiada que la del DBSCAN para este sistema.

5. Anexo con el script/código utilizado

```
"""
GCOM - PRACTICA 2: DIAGRAMA DE VORONOI Y CLUSTERING
NOMBRE: Gabriel Alba Serrano
SUBGRUPO: U1
"""

import numpy as np
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn import metrics
from scipy.spatial import ConvexHull, convex_hull_plot_2d
from scipy.spatial import Voronoi, voronoi_plot_2d
import matplotlib.pyplot as plt

# Aqu  tenemos definido el sistema X de 1500 elementos (personas) con dos
# estados: X1 = "nivel de estr s" y X2 = "afici n al rock"
archivo1 = "C:/Users/Gabriel/Documents/CC-MAT/Geometr a Computacional/Pr ctica 2/
    Personas_en_la_facultad_matematicas.txt"
archivo2 = "C:/Users/Gabriel/Documents/CC-MAT/Geometr a Computacional/Pr ctica 2/
    Grados_en_la_facultad_matematicas.txt"
X = np.loadtxt(archivo1)
Y = np.loadtxt(archivo2)
labels_true = Y[:,0]

#Si quisieramos estandarizar los valores del sistema, har amos:
#from sklearn.preprocessing import StandardScaler
#X = StandardScaler().fit_transform(X)

#Envolvente convexa, envoltura convexa o c psula convexa, de X
hull = ConvexHull(X)
convex_hull_plot_2d(hull)
plt.title('Envolvente convexa del sistema X')
plt.plot(X[:,0],X[:,1],'ro', markersize=1)
plt.show()



""" APARTADO 1 """

""" Aplicamos al sistema X el algoritmo KMeans para cada n mero de vecindades
k = {2,3,...,15}, y comprobamos qu  coeficiente de Silhouette es mayor seg n k,
y de esta forma obtenemos el n mero  ptimo  de celdas de Voronoi. """

neighborhoods = list(range(2,16))
silhouette = []
for k in neighborhoods:
    #Usamos la inicializaci n aleatoria "random_state=0"
    kmeans = KMeans(n_clusters = k, random_state=0).fit(X)
    labels = kmeans.labels_
    silhouette.append( metrics.silhouette_score(X, labels) )

# Gr fica del coeficiente de Silhouette respecto al n mero de vecindades
plt.plot(neighborhoods,silhouette)
plt.xlabel('N mero de vecindades')
plt.ylabel('Coeficiente de Silhouette')
plt.title('K-Means')
plt.savefig('Silhouette KMeans.jpg')
plt.show()

""" Definimos n_clusters := k , tal que max(silhouette) = silhouette[k]
De esta forma, n_clusters es el n mero de vecindades que mejor empareja los
elementos de cada cluster (vecindad), es decir es el n mero de vecindades m s
 ptimo . """
n_clusters = neighborhoods[ silhouette.index(max(silhouette))]
print('El n mero  ptimo  de vecindades es: k = ' + str(n_clusters) + '\n')

""" Definimos la etiqueta que indica a qu  cluster pertenece cada elemento y
los centros de cada cluster, cuando el n mero de vecindades es n_clusters """
```

```
kmeans = KMeans(n_clusters = n_clusters, random_state=0).fit(X)
labels = kmeans.labels_
centers = kmeans.cluster_centers_

""" Representamos la clasificaci n del sistema X, por el algoritmo KMeans
y su respectivo diagrama de Voronoi. """

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

""" Definimos las celdas de Voronoi  seg n los centros de los clusters,y
graficamos el diagrama de Voronoi. """
vor = Voronoi(centers)
voronoi_plot_2d(vor)

""" Graficamos cada elemento del sistema X, seg n al cluster al que pertenezcan. """
for lab, col in zip(unique_labels, colors):
    if lab == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == lab)

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

""" Graficamos los centros de cada cluster. """
plt.plot(centers[:,0], centers[:,1], 'o', markersize=12, markerfacecolor="green",
    label='Centro del cluster')

""" Graficamos los puntos a=(0,0) y b=(0,-1), que los necesitaremos para el
apartado 3. """
problem = np.array([[0, 0], [0, -1]])
clases_pred = kmeans.predict(problem)
plt.plot(problem[:,0],problem[:,1],'o', markersize=12, markerfacecolor="red", label='A
    =(0,0) \nB=(0,-1)')

plt.title('Diagrama de Voron i y Clustering KMeans de X con ' + str(n_clusters) + '
    clusters')
plt.legend()
plt.savefig('Diagrama de Voron i y Clustering KMeans con 3 clusters.jpg')
plt.show()



""" APARTADO 2 """

""" Aplicamos al sistema X el algoritmo DBSCAN para epsilon (umbral de distancia)
perteneciente al intervalo (0.1 , 0.4), y comprobamos qu  coeficiente de Silhouette
es mayor seg n epsilon, y de esta forma obtenemos el n mero  ptimo  de celdas de
    Voronoi. """

# Los posibles valores de epsilon van a ser los siguientes puntos:
epsilon = np.arange(0.1, 0.4, 0.005)


### M trica 'Euclidean'
s_euclidean = []

for e in epsilon:
    db = DBSCAN(eps=e, min_samples=10, metric='euclidean').fit(X)
    labels = db.labels_
    s = metrics.silhouette_score(X, labels)
    s_euclidean.append(s)

# Graficamos
plt.plot(epsilon, s_euclidean)
```

```
plt.xlabel('Umbral de distancia')
plt.ylabel('Coeficiente de Silhouette')
plt.title('DBSCAN Euclidean')
plt.savefig('Silhouette DBSCAN Euclidean.jpg')
plt.show()

# Definimos el epsilon  ptimo  para la m trica Euclidean
epsilon_euclidean = format(epsilon[ s_euclidean.index(max(s_euclidean)) ] , '.3f')
print('El umbral de distancia  ptimo  para la m trica Euclidean es:    = ' +
    epsilon_euclidean + '\n')


### M trica 'Manhattan'
s_man = []

for e in epsilon:
    db = DBSCAN(eps=e, min_samples=10, metric='manhattan').fit(X)
    labels = db.labels_
    s = metrics.silhouette_score(X, labels)
    s_man.append(s)

# Graficamos
plt.plot(epsilon, s_man)
plt.xlabel('Umbral de distancia')
plt.ylabel('Coeficiente de Silhouette')
plt.title('DBSCAN Manhattan')
plt.savefig('Silhouette DBSCAN Manhattan.jpg')
plt.show()

# Definimos el epsilon  ptimo  para la m trica Manhattan
epsilon_man = format( epsilon[ s_man.index(max(s_man)) ] , '.3f' )
print('El umbral de distancia  ptimo  para la m trica Manhattan es:    = ' +
    epsilon_man + '\n')

""" Finalmente en este apartado vamos a comparar gr ficamente el resultado de
aplicar el algoritmo DBSCAN con m trica 'Euclidean' y 'Manhattan' al conjunto X,
es decir, el clustering, frente al clustering obtenido al aplicar KMeans """

# Gr fica DBSCAN Euclidean

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

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

n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

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('DBSCAN Euclidean del sistema X con ' + str(n_clusters_) + ' clusters')
plt.show()
```

```
203
204 # Gr fica DBSCAN Manhattan
205
206 db = DBSCAN(eps=float(epsilon_man), min_samples=10, metric='manhattan').fit(X)
207 labels = db.labels_
208 core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
209 core_samples_mask[db.core_sample_indices_] = True
210
211 unique_labels = set(labels)
212 colors = [plt.cm.Spectral(each)
213           for each in np.linspace(0, 1, len(unique_labels))]
214
215 n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
216 n_noise_ = list(labels).count(-1)
217
218 plt.figure(figsize=(8,4))
219 for k, col in zip(unique_labels, colors):
220     if k == -1:
221         # Black used for noise.
222         col = [0, 0, 0, 1]
223
224     class_member_mask = (labels == k)
225
226     xy = X[class_member_mask & core_samples_mask]
227     plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
228              markeredgecolor='k', markersize=5)
229
230     xy = X[class_member_mask & ~core_samples_mask]
231     plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
232              markeredgecolor='k', markersize=3)
233
234 plt.title('DBSCAN Manhattan del sistema X con ' + str(n_clusters_) + ' clusters')
235 plt.show()
236
237 # Conclusi n
238 print('Como conclusi n observamos que el algoritmo KMeans clasifica los elementos de
      X en 3 clusters, frente al algoritmo DBSCAN que con ambas m tricas (Euclidean y
      Manhattan) clasifica los elementos en un  nico  cluster.')
239 print('Hay que destacar que la clasificaci n del algoritmo KMeans es m s  ptima  que
       la del DBSCAN porque su coeficiente de Silhouette es el que m s se aproxima a 1.
      \n')
240
241
242
243 """ APARTADO 3 """
244
245 problem = np.array([[0, 0], [0, -1]])
246 clases_pred = kmeans.predict(problem)
247 print('Seg n el clustering resultante de aplicar el algoritmo KMeans, los puntos a
      =(0,0) y b=(0,-1) deber an pertenecer a los clusters ' + str(clases_pred[0]) + ' y
       ' + str(clases_pred[1]) + ', respectivamente.')
```