Juan Carlos Díaz Rodríguez
Subgrupo U1
juancdia@ucm.es

1. Introducción

En el segundo tema del curso hemos visto algoritmos que nos permiten clasificar un conjunto de datos según distintos criterios, todos ellos fuertemente relacionados con las distancias que definamos. En esta práctica nos centramos en el algoritmo de *k-means* y *DB-SCAN*. Además, para los resultados obtenidos con el primero de ellos representaremos el diagrama de Voronoi asociado a las etiquetas obtenidas.

2. Material utilizado

Los datos empleados son aquellos proporcionados en el archivo *Personas_en_la_facultad_matematicas.txt*. En él tenemos un conjunto de 1500 datos que relacionan el nivel del estrés con la afición al rock.

Los algoritmos descritos anteriormente no se han programados directamente, sino que se ha hecho uso de las funcionalidades ofrecidas por la librería `sklearn`. Además, la parte gráfica se ha obtenido a raíz de la librería `matplotlib`. Se describe a continuación la resolución de los distintos apartados.

Apartado i)

En este apartado se pide hallar el número de *clusters* óptimo para el algoritmo de *k-means*. Esta optimalidad se determina en función del coeficiente de *Silhouette*, que recordemos viene dado por

$$\overline{s} \coloneqq \frac{1}{N} \sum_{q=1}^{k} \sum_{q \in V_1} \frac{b_i - a_i}{\text{máx}\{a_i, b_i\}},$$

donde $a_i \coloneqq \frac{1}{|V_q|-1} \sum_{j \in V_q} d_p(i,j)$, $b_i \coloneqq \min_{k \neq q} \frac{1}{|V_k|} \sum_{j \in V_k} d_p(i,j)$. Este cálculo se realiza a través de la función en el módulo `sklearn.metrics`. Este coeficiente da una idea de la compacidad de las regiones determinadas por las vecindades $\{V_q\}_{q=1}^{k}$, por lo que tendremos que seleccionar aquel valor de k que maximice $\overline{s}$.

Para completar el análisis, se representan los datos según las etiquetas dadas por *k-means* en un diagrama de Voronoi. Más tarde se empleará este diagrama para determinar visualmente el apartado iii).

Apartado ii)

A continuación, se estudia el algoritmo *DBSCAN*, cuya implementación se encuentra dentro del módulo `sklearn.clusters`. Para estudiar el mejor ϵ a tomar en el intervalo $(0{,}1, 0{,}4)$ se toman valores aleatorios en dicho intervalo y se calcula el coeficiente de *Silhouette*. Con ello se comparan los *clusters* obtenidos en cada caso, el ϵ usado y el coeficiente obtenido. Todas estas comparaciones se harán empleando la distancia euclídea por un lado y la distancia de Manhattan por otro.

Apartado iii)

Volviendo a los datos obtenidos a partir del apartado i), se obtiene una gráfica donde se pueda ver a qué grupo de datos pertenecen los puntos $a \coloneqq (0,0)$ y $b \coloneqq (0,-1)$ y se compara con la predicción del método `predict`.

3. Resultados

Se presentan a continuación los resultados referentes a los anteriores apartados.

En primer lugar, se observa en Figura 1 que el número óptimo de *clusters* es 3 para el algoritmo de *k-means*. En Figura 2 se encuentra el diagrama de Voronoi para este número de *clusters*.

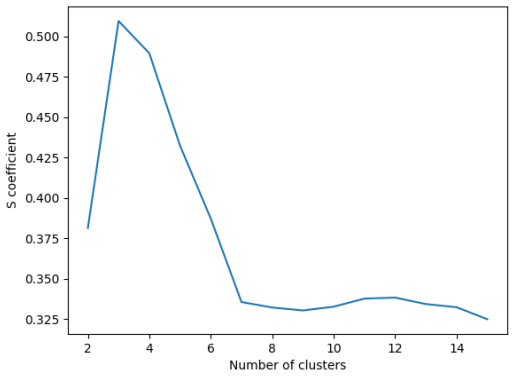


Figura 1: Búsqueda del número óptimo de clusters

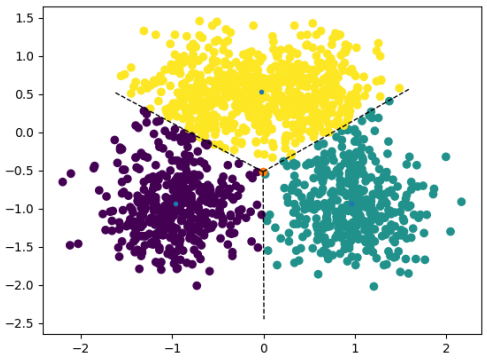


Figura 2: Diagrama de Voronoi para k=3

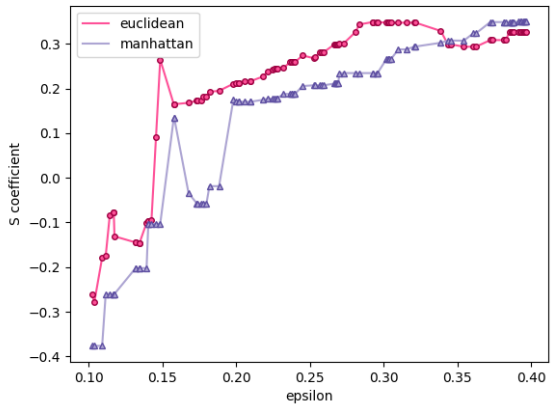


Figura 3: Búsqueda del ϵ óptimo

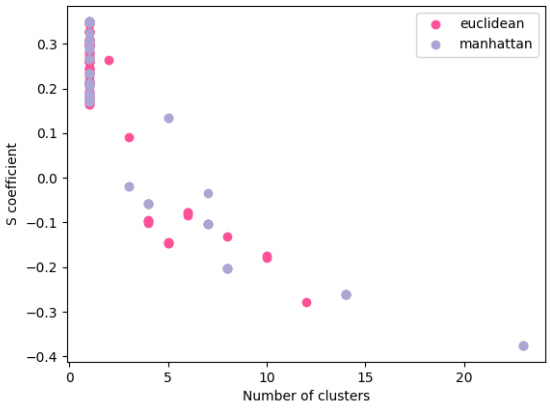
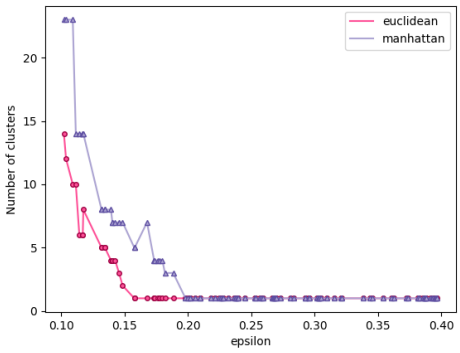


Figura 4: Relación número de *clusters*, coeficiente S obtenido

En el algoritmo *DBSCAN* se observa un valor óptimo de ϵ de 0.3 y 0.4 para cada una de las distancias en Figura 3. El resultado óptimo para Manhattan es engañoso debido a haber fijado n_0 a un valor demasiado bajo para un conjunto de datos disperso. Para valores mayores de n_0 se obtendrían resultados más positivos. Se observa también en Figura 4, reforzando la idea de que los resultados no son adecuados debido a n_0 que la mayoría de ϵ reportan un número de *clusters* óptimo en 1. Por último, en Figura 5 se aprecia como únicamente para valores menores de ϵ el algoritmo opta por aumentar el número de *clusters*, como es más lógico.

Para el último apartado basta hacer zoom sobre el diagrama para ver los puntos a y b pertenecen a las clases 2 y 1 repectivamente. Esto puede ser corroborado con la función `pred` de *k-means*.

Figura 5: Número de *clusters* obtenidos para cada ϵ



4. Conclusión

De esta práctica podemos sacar las siguientes conclusiones:

- El coeficiente de *Silhouette* es un buen indicador para elegir el número de *clusters* en algoritmos de clasificación.
- Los resultados del algoritmo *DBSCAN* sufren si elegimos un n_0 demasiado bajo para conjuntos de datos no especialmente densos.

5. Código

```
import numpy as np

from sklearn.cluster import KMeans
from sklearn import metrics
import matplotlib.pyplot as plt

markers = ['o', '^', 's', 'D', 'v' ,'+']
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']

def lighten_color(color, amount=0.5):
    """
    Lightens the given color by multiplying (1-luminosity) by the given amount.
    Input can be matplotlib color string, hex string, or RGB tuple.

    Examples:
    >> lighten_color('g', 0.3)
    >> lighten_color('#F034A3', 0.6)
    >> lighten_color((.3,.55,.1), 0.5)
    """
    import matplotlib.colors as mc
    import colorsys
    try:
        c = mc.cnames[color]
    except:
        c = color
    c = colorsys.rgb_to_hls(*mc.to_rgb(c))
    return colorsys.hls_to_rgb(c[0], 1 - amount * (1 - c[1]), c[2])

# Carga de datos
file1 = "Personas_en_la_facultad_matematicas.txt"
file2 = "Grados_en_la_facultad_matematicas.txt"
X = np.loadtxt(file1, encoding='latin1')
Y = np.loadtxt(file2, encoding='latin1')
labels_true = Y[:,0]

header = open(file1).readline()

# Apartado i)

results = []
s_coefficients = []

for k in range(2,16):
    results.append(KMeans(n_clusters=k, random_state=0, n_init=10).fit(X))
    labels = results[k-2].labels_
    s_coefficients.append(metrics.silhouette_score(X, labels))

fig, ax = plt.subplots()
ax.plot(range(2,16), s_coefficients)
ax.set_xlabel('Number of clusters')
ax.set_ylabel('S coefficient')
plt.show()

print('El mejor valor de k es :', np.argmax(s_coefficients) + 2)

from scipy.spatial import Voronoi, voronoi_plot_2d

# Obtenemos los centros para k = 3
k = 3
```

```
kmeans = KMeans(n_clusters=k, random_state=0, n_init=10).fit(X)
fig, ax = plt.subplots()
centers = kmeans.cluster_centers_
voronoi_gen = Voronoi(centers)
voronoi_plot_2d(voronoi_gen, ax=ax)

# Now plot the rest of the points
unique_labels = set(kmeans.labels_)
colors = [plt.cm.Spectral(each)\
          for each in np.linspace(0, 1, len(unique_labels))]

ax.scatter(X[:, 0], X[:, 1], c=kmeans.labels_)
problem = np.array([[0, 0], [0, -1]])

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

# Apartado ii)
from sklearn.cluster import DBSCAN

n0 = 10
N = 100
random_epsilons = np.sort(np.random.uniform(0.1, 0.4, (N, )))
euclidean_s = []
euclidean_k = []
for epsilon in random_epsilons:
  db = DBSCAN(eps=epsilon, min_samples=n0, metric='euclidean').fit(X)
  labels = db.labels_
  n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
  euclidean_s.append(metrics.silhouette_score(X, labels))
  euclidean_k.append(n_clusters_)

manhattan_s = []
manhattan_k = []
for epsilon in random_epsilons:
  db = DBSCAN(eps=epsilon, min_samples=n0, metric='manhattan').fit(X)
  labels = db.labels_
  n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
  manhattan_s.append(metrics.silhouette_score(X, labels))
  manhattan_k.append(n_clusters_)



fig, ax = plt.subplots()
ax.plot(random_epsilons, euclidean_k, label='euclidean', color=lighten_color(colors[0]))
ax.plot(random_epsilons , euclidean_k, markers[0], markersize = 4, markerfacecolor = \
        lighten_color(colors[0]), markeredgecolor = colors[0])

ax.plot(random_epsilons, manhattan_k, label='manhattan', color=lighten_color(colors[2]))
ax.plot(random_epsilons , manhattan_k, markers[1], markersize = 4, markerfacecolor = \
        lighten_color(colors[2]), markeredgecolor = colors[2])
ax.set_ylabel('Number of clusters')
ax.set_xlabel('epsilon')
ax.legend()
plt.show()

fig, ax = plt.subplots()
ax.plot(random_epsilons, euclidean_s, label='euclidean', color=lighten_color(colors[0]))
ax.plot(random_epsilons, euclidean_s, markers[0], markersize = 4, markerfacecolor = \
        lighten_color(colors[0]), markeredgecolor = colors[0])
```

```
ax.plot(random_epsilons, manhattan_s, label='manhattan', color=lighten_color(colors[2]))
ax.plot(random_epsilons, manhattan_s, markers[1], markersize = 4, markerfacecolor = \
        lighten_color(colors[2]), markeredgecolor = colors[2])
ax.set_ylabel('S coefficient')
ax.set_xlabel('epsilon')
ax.legend()
plt.show()
best_eps_manhattan = random_epsilons[np.argmax(manhattan_s)]
best_eps_euclidean = random_epsilons[np.argmax(euclidean_s)]
print('Mejor epsilon para Manhattan: ', best_eps_manhattan)
print('Mejor epsilon para Euclidean: ', best_eps_euclidean)

fig, ax = plt.subplots()
ax.scatter(euclidean_k, euclidean_s, label='euclidean', color=lighten_color(colors[0]))

ax.scatter(manhattan_k, manhattan_s, label='manhattan', color=lighten_color(colors[2]))
ax.set_ylabel('S coefficient')
ax.set_xlabel('Number of clusters')
ax.legend()
plt.show()

# Apartado iii)
problem = np.array([[0, 0], [0, -1]])
clases_pred = kmeans.predict(problem)
print(clases_pred)
```