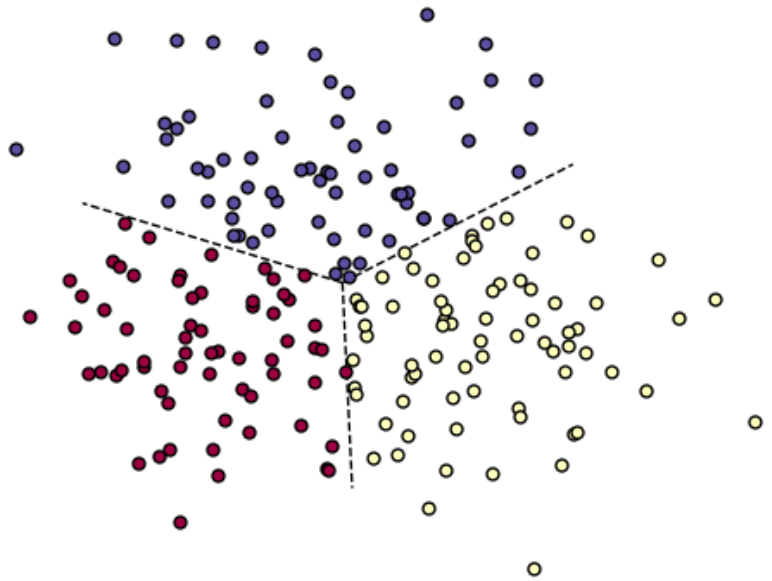
Práctica 3 - GCOMP

Celia Rubio Madrigal

16 de marzo de 2022

Índice



1. Introducción

En esta práctica aplicaremos algoritmos de clasificación no supervisada a un sistema dado. Suponemos que los elementos del sistema pueden agruparse en un número desconocido de clases en base a la distancia de sus estados frente a los del resto, y queremos responder las siguientes preguntas: ¿Cuántas clases, o clústeres, forman los elementos? ¿A qué clúster pertenece cada elemento? ¿A qué clúster pertenecería un elemento nuevo?

2. Material usado y metodología

Partimos de un sistema de 1000 elementos con 2 estados cada uno, que generamos mediante la función `make_blobs` a partir de 3 centros predefinidos: $(-0.5, 0.5)$, $(-1, -1)$ y $(1, -1)$.

2.1. Apartado *i*)

El primer algoritmo para agrupar los elementos del sistema es la clasificación por k-medias. Delimitará como clases las *regiones de Voronoi* de unos puntos (los *centroides*), que se deben determinar iterativamente.

La región de Voronoi de un centroide c será el conjunto de puntos del plano cuyo centroide más próximo sea c. Para poder definir la proximidad de los puntos usaremos la métrica euclídea sin ponderar, que es la p-norma con $p = 2$ y pesos $w_i = 1$: $d_2(x, y) = \sqrt{\sum_{i=1}^{2} |x_i - y_i|^2}$. Así dividimos el espacio en regiones, y clasificamos los elementos del sistema según en qué región se encuentran sus estados.

El algoritmo recibe como parámetros el número de centroides k —igual al de clases— y los puntos del sistema. Mediante la clase `KMeans` de *sklearn*, y su método `fit`, obtenemos en el atributo `labels_` la clasificación. Para averiguar cuál es el mejor k debemos probar con valores en $[2, 15]$ y calcular el coeficiente de Silhouette de cada clasificación. El coeficiente de Silhouette calcula si cada punto está bien clasificado en su clase con respecto a si estuviera en las otras clases. Nos quedaremos con el máximo, y dibujaremos los puntos ya clasificados junto a las fronteras de las regiones de Voronoi a las que pertenecen.

2.2. Apartado *ii*)

El segundo algoritmo de clasificación es DBSCAN, que basa sus agrupamientos en separar regiones de alta densidad de puntos mediante fronteras de baja densidad. Para ello, se toman bolas de un radio, o umbral de distancia, ε, y se van agregando puntos entre sí que son alcanzables mediante esas bolas. Se ha de tomar un límite para el que un elemento forma parte de la frontera de un clúster y no de su interior; será cuando no es alcanzable por otros $n_0 = 10$ puntos.

El algoritmo recibe los puntos y el parámetro ε. Mediante la clase `DBSCAN` de *sklearn*, y su método `fit`, obtenemos en `labels_` la clasificación. Para averiguar cuál es el mejor ε debemos probar con valores en $(0.1, 0.4)$ y calcular el coeficiente de Silhouette de cada clasificación, quedándonos de nuevo con el máximo. Las bolas pueden tomarse con distintas métricas. En nuestro caso serán la euclídea y la Manhattan (p-norma con $p = 1$).

2.3. Apartado *iii*)

Con las celdas de Voronoi del apartado i), podemos comprobar a cuáles pertenecerían otros elementos que no forman parte de nuestro sistema. Lo calculamos para $(0, 0)$ y $(0, -1)$.

3. Resultados y conclusiones

3.1. Apartado *i*)

Para los distintos valores de k, en la figura 1 se encuentran sus valores del coeficiente de Silhouette. El valor máximo es $k = 3$ ($\bar{s} = 0.588 \pm 0.001$), que representamos en la figura 2. Los centroides resultantes son $(-0.54, 0.50)$, $(-0.97, -1.02)$ y $(0.98, -0.98)$ ± 0.01, que son muy cercanos a los tres puntos originales generadores del sistema.

3.2. Apartado *ii*)

Para los distintos valores de ε, en la figura 3 se encuentran sus valores del coeficiente de Silhouette para la métrica euclídea (máximo en $\varepsilon = 0.280 \pm 0.001$, $\bar{s} = 0.467 \pm 0.001$), y en la figura 5 para la Manhattan (máximo en $\varepsilon = 0.320 \pm 0.001$, $\bar{s} = 0.440 \pm 0.001$). Las gráficas de sus clasificaciones están, respectivamente, en las figuras 4 y 6.

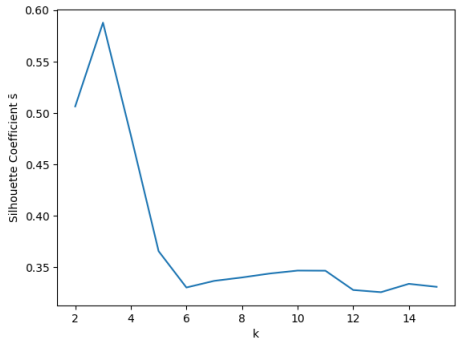


Figura 1: Coefs. de Silhouette para KMeans con $k \in [2, 15]$.

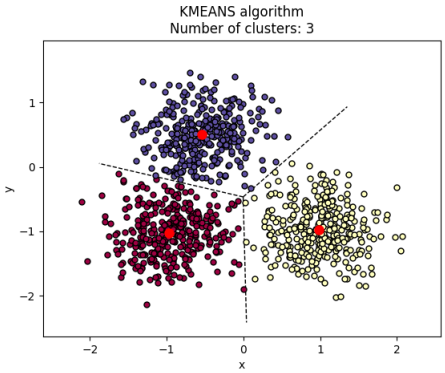


Figura 2: Mejor clasificación del sistema mediante KMeans.

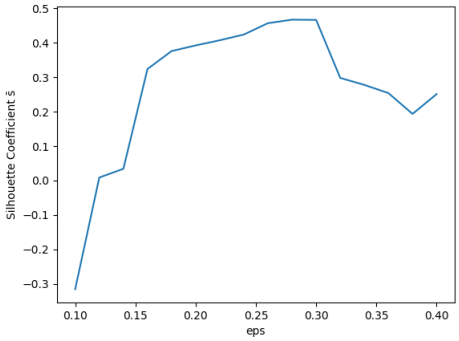


Figura 3: Coefs. de Silhouette para DBSCAN euclídeo con $\varepsilon \in (0.1, 0.4)$.

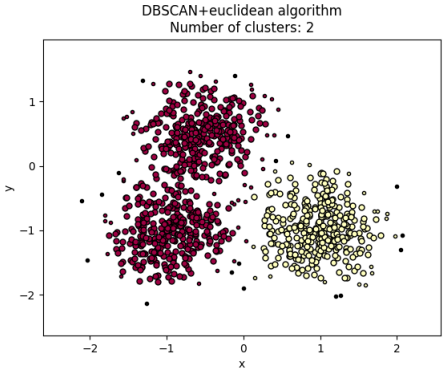


Figura 4: Mejor clasificación mediante DBSCAN euclídeo.

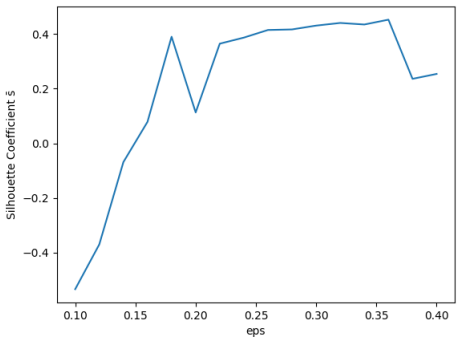


Figura 5: Coefs. de Silhouette para DBSCAN Manhattan con $\varepsilon \in (0.1, 0.4)$.

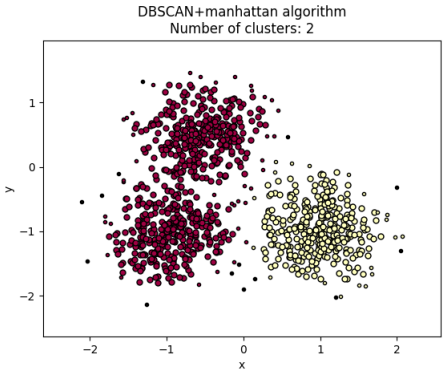


Figura 6: Mejor clasificación mediante DBSCAN Manhattan.

Hay 2 clases en vez de 3, ya que la frontera entre las dos a la izquierda es demasiado densa como para distinguirse mediante este algoritmo. También observamos que con este algoritmo algunos puntos son clasificados directamente como ruido en negro.

3.3. Apartado *iii)*

Volviendo a mirar la figura 1, un elemento con estado $(0, 0)$ queda clasificado en el clúster azul. Sin embargo, $(0, -1)$ queda en la frontera de dos celdas de Voronoi. Por tanto, en algunas ejecuciones del comando `kmeans.predict` sale el clúster rojo y en otras el amarillo.

4. Código

```
"""
Práctica 3
"""

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

Figure = 1

""" Apartado i) """

# #############################################################################
# Aquí tenemos definido el sistema X de 1000 elementos de dos estados
# construido a partir de una muestra aleatoria entorno a unos centros:
centers = [[-0.5, 0.5], [-1, -1], [1, -1]]
X, labels_true = make_blobs(
    n_samples=1000, centers=centers, cluster_std=0.4, random_state=0
)

# #############################################################################
# Los clasificamos mediante el algoritmo KMeans
def plot_silhouette(param, xs, alg):
    global Figure
    ss = []
    for x in xs:
        m = alg(x).fit(X)
        silhouette = metrics.silhouette_score(X, m.labels_)
        k = len(set(m.labels_)) - (1 if -1 in m.labels_ else 0)
        print(param,"= %0.3f" % x, "\tk =", k, "\ts̄ = %0.3f" % silhouette)
        ss += [silhouette]
    plt.xlabel(param)
    plt.ylabel("Silhouette Coefficient s̄")
    plt.plot(xs, ss)
    plt.savefig("Figure_%d" % Figure)
    Figure += 1
    plt.show()


plot_silhouette("k",range(2, 16), lambda x: KMeans(n_clusters=x, random_state=0))
kmeans = KMeans(n_clusters=3, random_state=0).fit(X)
print("\nCentros:\n", kmeans.cluster_centers_)

print("-" * 10)

# #############################################################################
# Representamos el resultado con un plot


def plot_points(alg, core_samples_mask=None):
    if type(core_samples_mask) is not np.ndarray:
        core_samples_mask = np.zeros_like(alg.labels_, dtype=bool)
        core_samples_mask[:] = True

    plt.xlim([X[:, 0].min() - 0.5, X[:, 0].max() + 0.5])
```

```
    plt.ylim([X[:, 1].min() - 0.5, X[:, 1].max() + 0.5])

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

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

        class_member_mask = alg.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.xlabel("x")
    plt.ylabel("y")


vor_regions = Voronoi(kmeans.cluster_centers_)
voronoi_plot_2d(vor_regions, show_points=False, show_vertices=False)

plot_points(kmeans)
plt.plot(
    kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], "ro", markersize=8
)
plt.title("KMEANS algorithm\nNumber of clusters: %d" % len(kmeans.cluster_centers_))
plt.savefig("Figure_%d" % Figure)
Figure += 1
plt.show()


""" Apartado ii) """

# ##############################################################################
# Los clasificamos mediante el algoritmo DBSCAN

plot_silhouette("eps",
    np.arange(0.1, 0.4, 0.02),
    lambda x: DBSCAN(eps=x, min_samples=10, metric="euclidean"),
)
db = DBSCAN(eps=0.28, 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
```

```
plot_points(db, core_samples_mask)

n_clusters_ = len(set(db.labels_)) - (1 if -1 in db.labels_ else 0)
plt.title("DBSCAN+euclidean algorithm\nNumber of clusters: %d" % n_clusters_)
plt.savefig("Figure_%d" % Figure)
Figure += 1
plt.show()

print("-" * 10)

plot_silhouette("eps",
    np.arange(0.1, 0.4, 0.02),
    lambda x: DBSCAN(eps=x, min_samples=10, metric="manhattan"),
)
db = DBSCAN(eps=0.36, 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
plot_points(db, core_samples_mask)

n_clusters_ = len(set(db.labels_)) - (1 if -1 in db.labels_ else 0)
plt.title("DBSCAN+manhattan algorithm\nNumber of clusters: %d" % n_clusters_)
plt.savefig("Figure_%d" % Figure)
Figure += 1
plt.show()

print("-" * 10)


""" Apartado iii) """

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

vor_regions = Voronoi(kmeans.cluster_centers_)
voronoi_plot_2d(vor_regions, show_points=False, show_vertices=False)

plot_points(kmeans)
plt.plot(problem[:, 0], problem[:, 1], "ro", markersize=8)

plt.title("Class Predictions")
plt.savefig("Figure_%d" % Figure)
Figure += 1
plt.show()
```