Memoria Practica 3 – Geometría computacional

Jorge del Valle Vázquez

1 Introducción

Partiendo de un sistema X de 1000 elementos obtendremos una clasificación en vecidades de Voronoi, clusters, donde la cantidad de estas se obtiene optimizando el coeficiente de Silhouette.

Trabajamos con un algoritmo que clasifica por cercanía de elementos y otro que lo hace por densidad de elementos en una zona.

2 Material usado

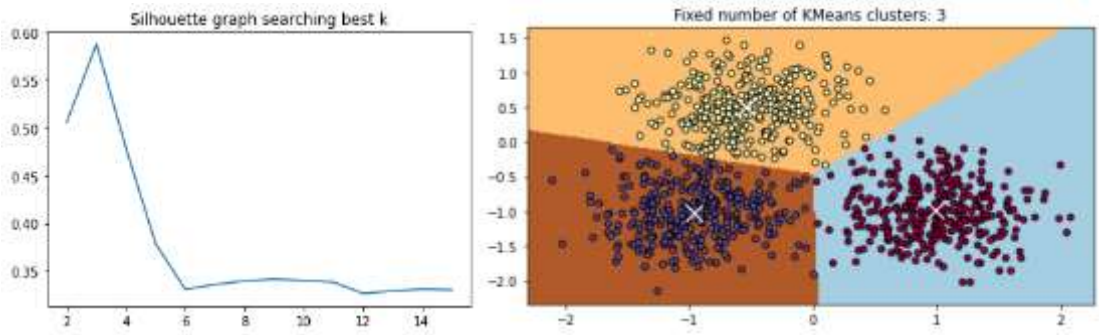
El objetivo principal es explorar la librería **sklearn**, para el uso de *KMEANS* y *DBSCAN*. También se emplea como viene siendo habitual **matplotlib.pyplot** para las gráficas junto numpy para trabaar con arrays, y en la plantilla proporcionada se muesta la librería spicy que permite representar la envoltura convexa.

Como estamos en la busca del numero de vecindades optimo en el caso de *kmeans,* y de épsilon en el caso de *dbscan*, en ambos casos procedemos de la misma forma. Hacemos variar el parámetro a optimizar, k toma valores enteros entre 2 y 15, y épsilon entre 0.1 y 0.4, que en nuestro caso consideramos todos con paso 0.02. La selección del mejor valor se corresponde con la que da lugar a un mayor coeficiente de silhouette. Para apreciarlo, mostramos la grafica del coeficiente de silhouette en función de k o épsilon.

Una vez obtenidos los parámetros clasificamos el sistema X según estos. Mostramos en una gráfica la clasificación , y en el caso de *kmeans* en la misma gráfica se muestra el diagrama de voronoi haciendo uso de imshow de pyplot.

Mediante la función *predict* de la clasificación ya entrenada de *kmeans* podemos predecir el clúster al que pertenecen los puntos a(0,0) y b(0,-1). Para facilitar la interpretación del label asociado, indicamos a su vez el label correspondiente a cada centroide de los clústeres.

3 Resultados



i. En cuanto al algoritmo *Kmeans* tenemos que el valor óptimo **k** de vecindades es **3**, como se aprecia en la gráfica, del que se obtiene un valor 0.588en el coeficiente de

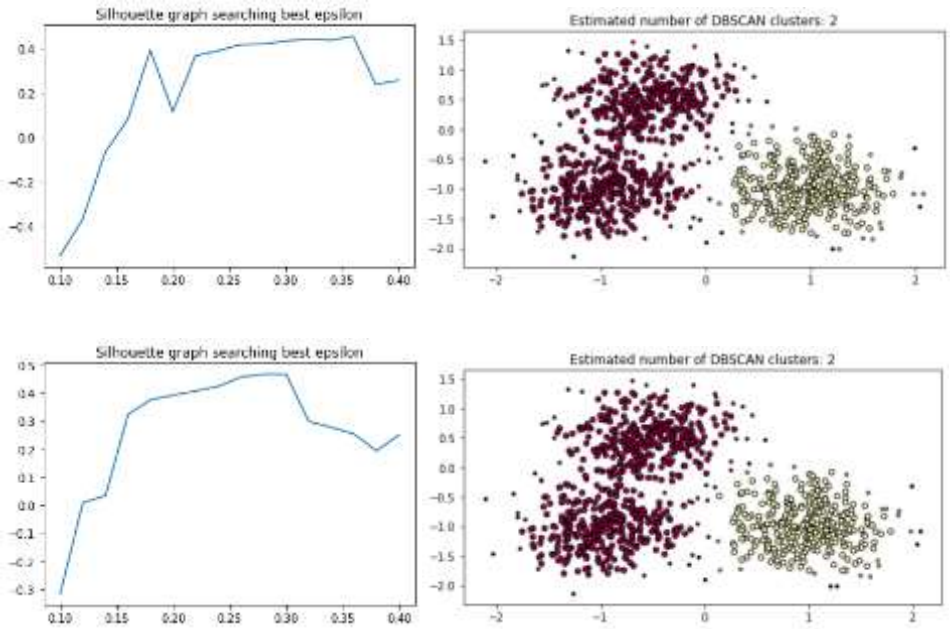
silhouette. Esto se debe a que el algoritmo haciendo uso de la distancia euclidiana favorece las agrupaciones como bolas, en este caso circunferencias, y es el valor k= 3 donde estas quedan más diferenciadas.

Además de la clasificación de cada punto, se muestra como fondo las respectivas celdas de Voronoi, que nos ayudan a interpretar el tercer ejercicio. En este obtenemos que la predicción del punto a (0,0) es el label 1, que se corresponde con el label del clúster que tiene por centroide [-0.5443386 0.50413529], es decir, el de fondo naranja que ocupa la zona superior, siendo sencillo visualizar que el origen se sitúa en esa zona. La predicción del punto b(0,-1) sería el label 2 correspondiente al clúster de centroide [-0.96555004 -1.02042969] con fondo marron en la zona inferior izquierda. Nótese que la separación entre los clusters 0 y 2 es una línea casi vertical, ligeramente inclinada comparándola con la recta {x=0} dejando así en su lado izquierdo el punto b, en otras palabras, en el clúster de label 2.

ii. En el segundo apartado consideramos el algoritmo DBSCAN y lo analizamos con ambas distancias, Manhattan y euclídea.
Para la distancia Manhattan se tiene que el **umbral de distancia** épsilon que optimiza es 0.360, que da lugar a un valor 0.452 del coeficiente de silhouette, mientras que con la euclídea sería 0.28 y un valor de 0.467.

Si queremos comparar los obtenido con el *kmeans* se aprecia fácilmente que aquí tenemos tan solo dos clusters, que principalmente se debe a que los el clúster en la izquierda carece de una zona de baja densidad entre lo que para el kmeans eran los clusters de label 1 y 2, impidiendo de esta forma diferenciarlos como si pude hacer con el otro pues la zona en torno a la recta {y=2x} es de baja densidad.

Vemos los gráficos de por orden, para la distancia manhattan y euclídea:



4 Conclusión

Muchas conclusiones han ido introduciéndose en el anterior desarrollo, como la comparación de los algoritmos, o el análisis sobre las predicciones. Fuera de eso cabe destacar la sencillez de la interpretación en este caso puesto que estamos trabajando en una dimensión fácilmente representable, cosa que no sucede para mayores dimensionalidades.

Además, se puede apreciar en el DBSCAN que la euclídea proporciona una clasificación ligeramente mejor. Como la gráfica muestra con puntos grandes aquellos con más relevancia en el clúster y pequeños los que menos, además de en negro los puntos de ruido, se observa también que para cada distancia los puntos pueden tomar distintos papeles. Por ejemplo, el punto más a la derecha se considera de ruido para la distancia euclídea.

5 Anexo: Código

KMEANS

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.datasets import make_blobs
from scipy.spatial import ConvexHull, convex_hull_plot_2d
#from scipy.spatial import Voronoi, voronoi_plot_2d
import matplotlib.pyplot as plt
# #############################################################################
# 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)
k_=[]
for k in range(2,16):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(X)
    silhouette = metrics.silhouette_score(X, kmeans.labels_)
    #Coeficiente de Silhouette
    k_.append(silhouette)
    print("Silhouette Coefficient: %0.3f" % silhouette)
    #plt.plot(k, silhouette, 'o', markersize=5)
silhouette_best=max(k_)
k_best=k_.index(silhouette_best)+2
print (k_best)
plt.plot(range(2,16), k_)
plt.title('Silhouette graph searching best k')
plt.show()
#Usamos la inicialización aleatoria "random_state=0"
kmeans = KMeans(n_clusters=k_best, random_state=0).fit(X)
labels = kmeans.labels_
```

```
# Índice de los centros de vencindades o regiones de Voronoi para cada elemento (punto)
print(kmeans.cluster_centers_)
# ##############################################################################
# Step size of the mesh. Decrease to increase the quality of the VQ.
h = 0.02  # point in the mesh [x_min, x_max]x[y_min, y_max].
# Plot the decision boundary. For that, we will assign a color to each
x_min, x_max = X[:, 0].min() - 0.2, X[:, 0].max() + 0.2
y_min, y_max = X[:, 1].min() - 0.2, X[:, 1].max() + 0.2
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
# Obtain labels for each point in mesh. Use last trained model.
AUX=np.c_[xx.ravel(), yy.ravel()]
Z = kmeans.predict(AUX)
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(figsize=(8,4))
plt.clf()
plt.imshow(
    Z,
    interpolation="nearest",
    extent=(xx.min(), xx.max(), yy.min(), yy.max()),
    cmap=plt.cm.Paired,
    aspect="auto",
    origin="lower",
)
unique_labels = set(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 = (labels == k)
    xy = X[class_member_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=5)
# Plot the centroids as a white X
centroids = kmeans.cluster_centers_
plt.scatter(
    centroids[:, 0],
    centroids[:, 1],
    marker="x",
    s=169,
    linewidths=3,
    color="w",
    zorder=10,
)
plt.title('Fixed number of KMeans clusters: %d' % k_best)
plt.show()
# ##############################################################################
# Predicción de elementos para pertenecer a una clase:
```

```
problem = np.array([[0,0], [0, -1]])
clases_pred = kmeans.predict(problem)
# mostramos la prediccion de los centroides para clarificar la  pertenencia
print(clases_pred,kmeans.predict(centroids))
```

DBSCAN

```
# -*- coding: utf-8 -*-
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
# #############################################################################
# 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)
e_=[]
s_=[]
for e in np.arange(0.1,0.4,0.02):
    db = DBSCAN(eps=e, min_samples=10, metric='euclidean').fit(X)
    labels = db.labels_
    sil= metrics.silhouette_score(X, labels)
    e_.append(e)
    s_.append(sil)

silhouette_best=max(s_)
e_best=e_[s_.index(silhouette_best)]
print('epsilon %0.3f' %e_best,"Silhouette Coefficient: %0.3f"      %silhouette_best)
plt.plot(np.arange(0.1,0.4,0.02), s_)
plt.title('Silhouette graph searching best epsilon')
plt.show()

db = DBSCAN(eps=e_best, 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_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# #############################################################################
# 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 clusters: %d' % n_clusters_)
plt.show()

# ##################################
# Repetimos con MANHATTAN
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)
e_=[]
s_=[]
for e in np.arange(0.1,0.4,0.02):
    db = DBSCAN(eps=e, min_samples=10, metric='manhattan').fit(X)
    labels = db.labels_
    sil= metrics.silhouette_score(X, labels)
    e_.append(e)
    s_.append(sil)

silhouette_best=max(s_)
e_best=e_[s_.index(silhouette_best)]
print('epsilon %0.3f' %e_best,"Silhouette Coefficient: %0.3f"      %silhouette_best)
plt.plot(np.arange(0.1,0.4,0.02), s_)
plt.title('Silhouette graph searching best epsilon')
plt.show()

db = DBSCAN(eps=e_best, 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_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# ##############################################################################
# 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 clusters: %d' % n_clusters_)
plt.show()
```