Diagrama de Voronói y clustering

Sergio González Montero (U2)

5 de marzo de 2024

1. Objetivo

A partir del sistema proporcionado por los archivos txt se pretende clasificar su contenido en *clusters* o celdas de Voronói utilizando el coeficiente de Silhouette, estimando con él un número óptimo de vecindades $k \in \{1, 2, \ldots, 15\}$. También se explorará el número óptimo a través del umbral de distancia $\epsilon \in (0{,}1, 0{,}4)$. Además, se hará una predicción de pertenencia a grupos de ciertos puntos concretos especificados en apartado de resultados.

2. Material y datos

Se usaron los siguientes apartados de las notas de Robert Monjo de la asignatura de Geometría Computacional, a día 04/03/2024: Definición 2.2.5, Diagramas de Voronói; Algoritmo 2.2.2, Clustering o clasificación por k-medias / k-medoids; Definición 2.2.6, Coeficiente de Silhouette y Algoritmo 2.2.3, Algoritmo DBSCAN.
Se usaron los archivos *Personas_de_villa_laminera.txt* y *Franjas_de_edad.txt* que contienen los elementos que conforman el sistema de datos a analizar.
En cuanto al código, se usaron como base las plantillas *GCOM2024-practica2_plantilla1* y *GCOM2024-practica2_plantilla2*. En él, se han utilizado las librerías siguientes:

1. sklearn: aporta algoritmos de agrupamiento como KMeans o DBSCAN, valoración de dichos agrupamientos con Silhouette
2. numpy: operaciones matemáticas y cálculos numéricos
3. matplotlib.pyplot: para visualización de gráficos
4. scypy.spatial: añade el cálculo y representación del diagrama de Voronói así como las métricas euclidiana y de Manhattan

3. Resultados

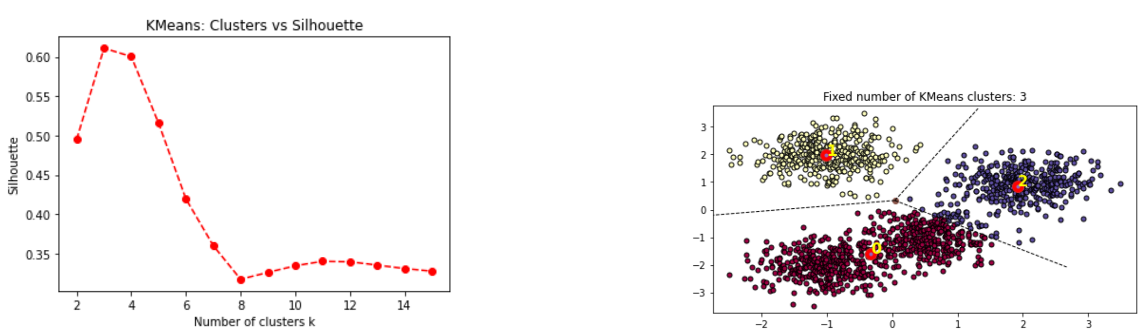

Figura 1: **Optimum clusters: 3, Silhouette 0.611**

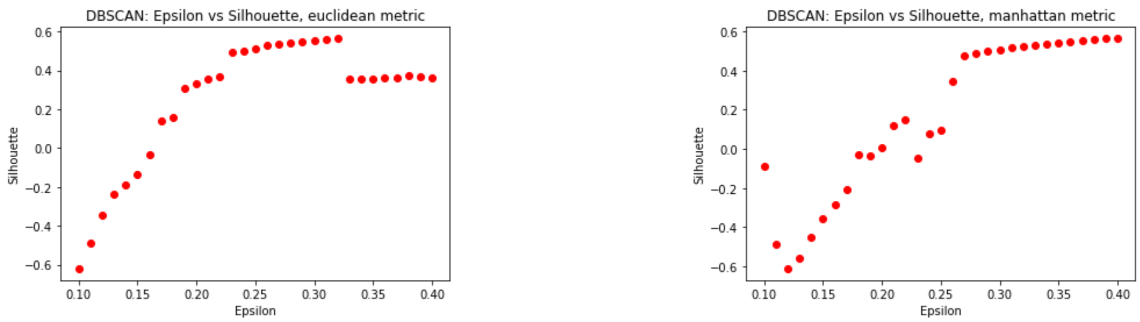

Figura 2: **Optimum euclidean epsilon: 0.32, Silhouette: 0.56357**
Optimum manhattan epsilon: 0.4, Silhouette: 0.56424

Figura 3: **Clusters según métrica**

En cuanto a las predicciones, siendo los puntos a = [0.5, 0], b = [0, -3], y los pares (etiqueta, centroide) (0, [-0.32559490084985876, -1.5872946175637381]), en color rojo, (1, [-1.0077659574468085, 1.9790159574468091]), en color amarillo y (2, [1.922177033492823, 0.8609330143540673]), en color verde, el punto a pertenece al cluster verde con la etiqueta 2, tanto por la métrica euclidiana como por la de manhattan, mientras que para el punto b se da una predicción de pertenencia al cluster rojo etiquetado como 0. Ambos resultados son consistentes con el dado por *kmeans.predict()*.

4. Conclusión

Como se ha podido comprobar la solución no es única pues, si nos guiamos únicamente por el coeficiente de Silhouette, podríamos decir que el algoritmo DBSCAN es más preciso con la métrica manhattan que con la euclidiana. Sin embargo, dependerá de la naturaleza de los datos, como su densidad o distribución, la elección de una u otra métrica para el cálculo de ϵ. De igual manera sucede con KMeans que, por defecto, usa la euclidiana y, aun así, difiere del resultado dado por DBSCAN con la misma métrica. Por tanto, se deduce que el coeficiente de Silhouette es una buena forma de deducir el número de clusters aunque en algunos casos puntuales haya que contrastar con otros métodos, como el índice de Calinski-Harabasz o la varianza explicada .

5. Código

Programa 1: practica2.py

```python
# Sergio Gonzalez Montero
# Victor Martin Martin

import numpy as np

from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn import metrics
import matplotlib.pyplot as plt
from scipy.spatial import Voronoi, voronoi_plot_2d
from scipy.spatial.distance import euclidean, cityblock

print("----------------Dataset----------------\n")
# Definimos el sistema A
archivo1 = "Personas_de_villa_laminera.txt" # Asignar archivos
archivo2 = "Franjas_de_edad.txt"
X = np.loadtxt(archivo1,skiprows=1) # Carga datos de un de un txt
Y = np.loadtxt(archivo2,skiprows=1)
labels_true = Y[:,0]

header = open(archivo1).readline()
print(header)
print(X)

# Pinta el dataset
plt.plot(X[:,0],X[:,1],'ro', markersize=1)
plt.title("Stress vs Sweets")
plt.xlabel("Stress")
plt.ylabel("Sweets")
plt.show()

print("\n----------------KMeans----------------\n")
# Calculo del numero de vecindades optimo
n_clusters = []
sil_k = []
for k in range(2, 16):
    n_cluster = k
    n_clusters.append(k)

    # Usamos la inicializacion aleatoria "random_state=0"
    kmeans = KMeans(n_clusters=k, random_state=0).fit(X)
    labels = kmeans.labels_
```

```
    silhouette = metrics.silhouette_score(X, labels)
    sil_k.append(silhouette)

plt.plot(n_clusters, sil_k, 'ro--')
plt.title("KMeans: Clusters vs Silhouette")
plt.xlabel("Number of clusters k")
plt.ylabel("Silhouette")
plt.show()
max_s_index = sil_k.index(max(sil_k))
# Numero de clusters asociado al Silhouette maximo
k_optimo = n_clusters[max_s_index]
print(f"Optimum clusters: {k_optimo} \n\
Silhouette {round(max(sil_k), 3)}\n")

# Se vuelve a calcular para la representacion particular con k_optimo
kmeans = KMeans(n_clusters=k_optimo, random_state=0).fit(X)
labels = kmeans.labels_
centroids = kmeans.cluster_centers_
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]

fig = plt.figure(figsize=(8,4))
ax = fig.add_subplot(111)

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)

# Graficado de centroides
plt.plot(centroids[:,0],centroids[:,1],'o',
         markersize=12, markerfacecolor="red")
for i in range(len(centroids)):
    plt.text(centroids[i,0],centroids[i,1],str(i),
             color='yellow',fontsize=16,fontweight='black')
# Diagrama de Voronoi
vor = Voronoi(centroids)
voronoi_plot_2d(vor,ax=ax)
# Acomodamiento de los ejes respecto al dataset
```

```python
plt.xlim([min(X[:,0])-0.25,max(X[:,0])+0.25])
plt.ylim([min(X[:,1])-0.25,max(X[:,1])+0.25])

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

print("\n----------------DBSCAN----------------\n")
e = []
sil_e = []
def dbscan_silhouette(metric):
    """
    metric : str, metrica a usar por DBSCAN
    Grafica epsilon vs Silhouette y calcula el
    epsilon optimo para cada metrica
    """
    for epsilon in np.arange(0.10,0.4,0.01):

        # Utilizamos el algoritmo de DBSCAN para minimo 10 elementos
        db = DBSCAN(eps=epsilon, min_samples=10, metric=metric).fit(X)
        labels = db.labels_
        # Number of clusters in labels, ignoring noise if present.
        n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
        silhouette=metrics.silhouette_score(X,labels)if n_clusters_!=1 else -1
        sil_e.append(silhouette)
        e.append(round(epsilon, 2))
        plt.plot(round(epsilon, 2), silhouette, 'ro--')
        plt.title(f"DBSCAN: Epsilon vs Silhouette, \
{metric} metric")
    plt.xlabel("Epsilon")
    plt.ylabel("Silhouette")
    plt.show()

    max_s_index = sil_e.index(max(sil_e))
    # Numero de clusters asociado al Silhouette maximo
    e_optimo = e[max_s_index]
    printed = print(f"Optimum {metric} epsilon: {e_optimo}\nSilhouette: \
{round(max(sil_e), 3)}")
    return printed, e_optimo

dbscan_silhouette('euclidean')[0]
dbscan_silhouette('manhattan')[0]

def plot_dbscan(metric):
    """
    metric : str, metrica a usar por DBSCAN
    Grafica los diagramas correspondientes segun metrica
```

```
    """
    db = DBSCAN(eps=dbscan_silhouette(metric)[1],
                min_samples=10, metric=metric).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)

    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)

    # Graficado de centroides
    plt.plot(centroids[:,0],centroids[:,1],'o',
             markersize=12, markerfacecolor="red")
    for i in range(len(centroids)):
        plt.text(centroids[i,0],centroids[i,1],str(i),
                 color='yellow',fontsize=16,fontweight='black')
    # Diagrama de Voronoi
    vor = Voronoi(centroids)
    voronoi_plot_2d(vor,ax=ax)
    # Acomodamiento de los ejes respecto al dataset
    plt.xlim([min(X[:,0])-0.25,max(X[:,0])+0.25])
    plt.ylim([min(X[:,1])-0.25,max(X[:,1])+0.25])
    plt.title(f'Estimated number of DBSCAN clusters\
({metric}): %d' % n_clusters_)
    plt.show()
```

```python
plot_dbscan('euclidean')
print('\n')
plot_dbscan('manhattan')

print("\n----------------PREDICTION----------------\n")
a, b = [1/2,0], [0,-3]
print(f"Points: a = {a}, b = {b}")
print("(Label, centroid)")
for i in range(len(centroids)):
    print(f"({i}, {list(centroids[i])})")

a_d2 = [euclidean(a,centroid) for centroid in centroids]
b_d2 = [euclidean(b,centroid) for centroid in centroids]
a_dmanhattan = [cityblock(a,centroid) for centroid in centroids]
b_dmanhattan = [cityblock(b,centroid) for centroid in centroids]
cluster_a_e = a_d2.index(min(a_d2))
cluster_b_e = b_d2.index(min(b_d2))
cluster_a_m = a_dmanhattan.index(min(a_dmanhattan))
cluster_b_m = b_dmanhattan.index(min(b_dmanhattan))

print(f"\nPoint {a} belongs to cluster {cluster_a_e},\
 green, by euclidean metric")
print(f"Point {b} belongs to cluster {cluster_b_e},\
 red, by euclidean metric")
print(f"Point {a} belongs to {cluster_a_m},\
 green, by manhattan metric")
print(f"Point {b} belongs to {cluster_b_m},\
 red, by manhattan metric")

print("\nPrediction by kemeans.predict()")
a_predict = kmeans.predict([a])[0]
b_predict = kmeans.predict([b])[0]
print(f"Point a belongs to cluster {a_predict},\
 green, by kmeans.predict()")
print(f"Point b belongs to cluster {b_predict},\
 red, by kmeans.predict()")
```