Práctica 3: KMeans y DBSCAN

Guillermo Martín Sánchez

20/03/20

1 Introducción

El objetivo de esta práctica es el estudio de dos métodos de clasificación por análogos: KMeans y DBSCAN. Para ello, estudiamos cómo de bien clasifican unos datos según el valor del coeficiente de Silhouette que se define como:

$$\bar{s} = \frac{1}{N} \sum_{q=1}^{k} \sum_{i \in V_q} \frac{b_i - a_i}{\max\{a_i, b_i\}} \tag{1}$$

donde, a_i es la p-distancia media del elemento i a su vecinidad V_q, y b_i es la p-distancia mínima al resto de vecindades:

$$a_i = \frac{1}{|V_q| - 1} \sum_{j \in V_q} d_p(i,j) \qquad\qquad b_i = \min_{k \neq q} \frac{1}{|V_k|} \sum_{j \in V_k} d_p(i,j) \tag{2}$$

El valor de este coeficiente, por lo tanto, se encuentra en $[-1, 1]$ y, cuanto mayor sea el coeficiente, mejor se considera la clasificación.

2 Material usado

2.1 Ejercicio 1

Obtén el coeficiente $\bar{s}$ de X para diferente número de vecindades $k \in \{1, 2, 3, ..., 15\}$ utilizando el algoritmo KMeans. Muestra gráficamente $\bar{s}$ en función de k y decide cuál es el número óptimo de vecindades.

Hemos utilizado el algoritmo $KMeans$ de la librería de Python *sklearn.cluster*. Hemos calculado el coeficiente de Silhouette para cada k con *sklearn.metrics.silhouette.score*. Para el caso $k = 1$, no contemplado en esta función, hemos establecido que $b_i = 0$ (debido a que no hay otras vecindades) y, por lo tanto, $\bar{s} = -1$.

2.2 Ejercicio 2

Obtén el coeficiente $\bar{s}$ para el mismo sistema X usando ahora el algoritmo DBSCAN con la métrica 'euclidean' y luego con 'manhattan'. En este caso, el parámetro que debemos explorar es el umbral de distancia $\epsilon \in (0.1, 1)$, fijando el número de elementos mínimo en $n_0 = 10$. Comparad gráficamente con el resultado del apartado anterior

En este caso usamos el algoritmo $DBSCAN$ de *sklearn.cluster*. Igual que en el ejercicio anterior añadimos al coeficiente el valor $\bar{s} = -1$ cuando sólo hay un cluster.

3 Resultados y discusión

3.1 Ejercicio 1

En la Figura 1 podemos ver, para cada valor del cluster k, el correspondiente valor de $\bar{s}$ para el algoritmo $KMeans$ sobre los datos proporcionados. Como se puede ver, el valor para el cuál la clasificación es óptima (el valor de $\bar{s}$ es máximo) es con $k = 3$ clusters. En la Figura 2 vemos cómo este algoritmo clasifica los datos con ese número de clusters.

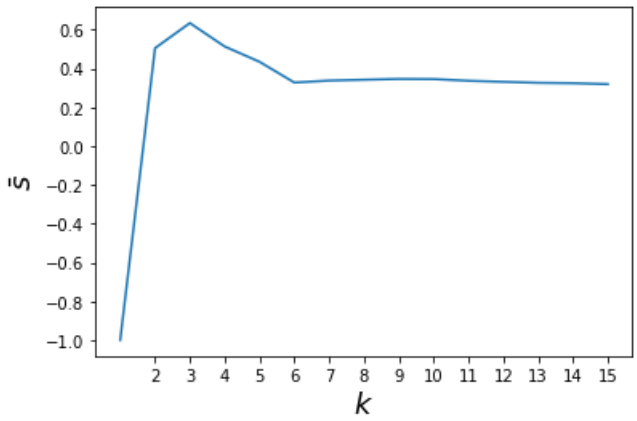

Figure 1: Coeficiente de Silhouette $\bar{s}$ en KMeans para distinto número de clusters k

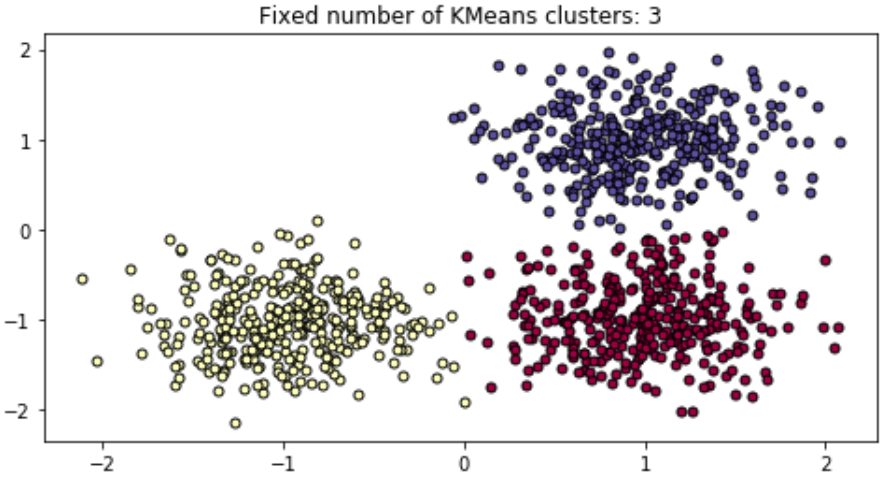

Figure 2: Puntos del dataset clasificados en tres clusters por KMeans

3.2 Ejercicio 2

Ahora en la Figura 3 tenemos el valor de $\bar{s}$ para cada valor de ϵ para la clasificación con DBSCAN con la métrica euclídea y la métrica manhattan. Como se puede ver ambas tienen un máximo en puntos distintos ($\epsilon = 0.2$ y $\epsilon = 0.3$ respectivamente). Además vemos que la métrica manhattan da un resultado mejor en su máximo. Por ello, usamos esta métrica y $\epsilon = 0.3$ para obtener la clasificación que se muestra en la Figura 4.

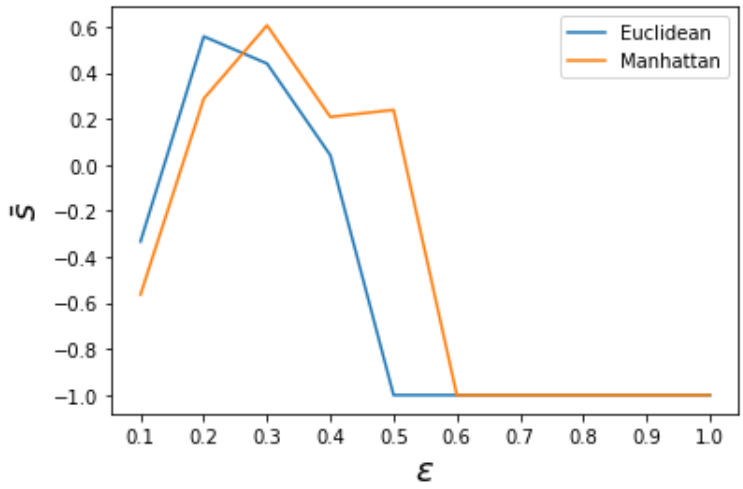

Figure 3: Coeficiente de Silhouette $\bar{s}$ en DBSCAN para distintos valores de ϵ en la métrica euclídea y la métrica manhattan

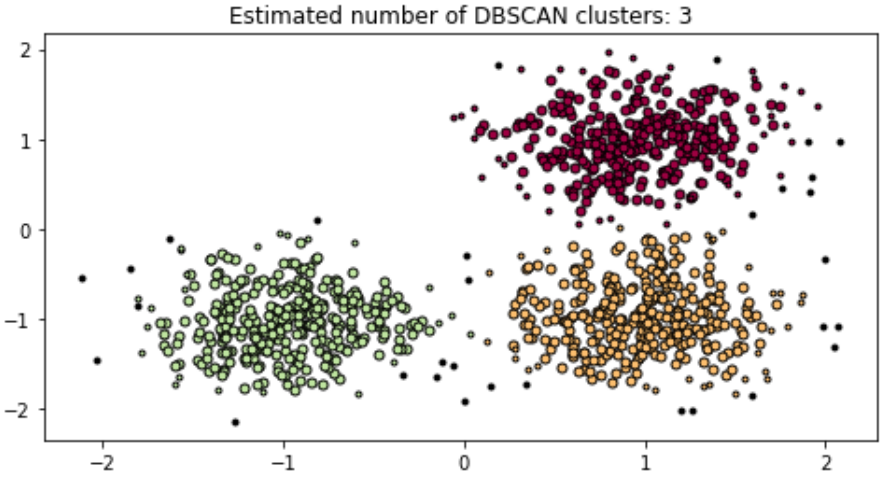

Figure 4: Puntos del dataset clasificados en tres clusters ($\epsilon = 0.3$) por DBSCAN. Los puntos negros son puntos considerados como ruido

Vemos que es muy parecida a la clasificación que hace KMeans, salvo por la existencia de los puntos de ruido. Además en ambos casos es una clasificación que visualmente parece la más acertada.

4 Conclusión

Hemos estudiado dos algoritmos de clasificación no supervisada. Ambos dan resultados parecidos sobre este dataset en sus clasificaciones óptimas. Además hemos visto como usar el coeficiente de Silhouette para encontrar estos parámetros óptimos.

5 Código

```
"""

Coursework 3: Clustering

References:
    https://scikit-learn.org/stable/modules/clustering.html
    https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
    https://docs.scipy.org/doc/scipy/reference/spatial.distance.html
"""

import numpy as np

from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs

import matplotlib.pyplot as plt

# #############################################################################
# Load data
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=1000, centers=centers, cluster_std=0.4,
                            random_state=0)

plt.plot(X[:,0],X[:,1],'ro', markersize=1)
plt.show()

#%%
"""
Exercise 1: KMeans
"""

# Test different number of clusters
n_clusters = range(2,16)
silhouette = [-1]
```

```
for n in n_clusters:
    kmeans = KMeans(n_clusters=n, random_state=0).fit(X)
    labels = kmeans.labels_
    silhouette.append(metrics.silhouette_score(X, labels))


# Plot Silhouette coefficients
plt.plot(range(1,16), silhouette);
plt.xticks(n_clusters);
plt.xlabel('$k$',fontsize = 18)
plt.ylabel(r'$\bar{s}$', fontsize = 18)

# Compute optimum value for the number of clusters
n_clusters = np.argmax(silhouette)+2

# Train optimum model of KMeans
kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
labels = kmeans.labels_

# Plot optimum model of KMeans
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]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=5)

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

#%%

"""
Exercise 2: DBSCAN
"""

# Test different number of epsilon
```

```
epsilon = np.linspace(0.1,1,10)

silhouette_euc = []
silhouette_man = []
for eps in epsilon:
    db = DBSCAN(eps=eps, min_samples=10, metric='euclidean').fit(X)
    labels = db.labels_
    if(len(set(labels)) == 1):
        silhouette_euc.append(-1)
    else:
        silhouette_euc.append(metrics.silhouette_score(X, labels))

    db = DBSCAN(eps=eps, min_samples=10, metric='manhattan').fit(X)
    labels = db.labels_
    if(len(set(labels)) == 1):
        silhouette_man.append(-1)
    else:
        silhouette_man.append(metrics.silhouette_score(X, labels))

# Plot Silhouette coefficients
p1, = plt.plot(epsilon, silhouette_euc, label = 'Euclidean');
p2, = plt.plot(epsilon, silhouette_man, label = 'Manhattan');
plt.xlabel('$\epsilon$', fontsize = 18)
plt.ylabel(r'$\bar{s}$', fontsize = 18)
plt.legend(handles = [p1,p2])
plt.xticks(epsilon);

# Compute optimum value of epsilon
eps_euc = epsilon[np.argmax(silhouette_euc)]
eps_man = epsilon[np.argmax(silhouette_man)]

# Train optimum model of DBSCAN
db = DBSCAN(eps=eps_man, min_samples=10, metric='manhattan').fit(X)
labels = db.labels_

# Plot different characteristics of the model

# 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)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(labels_true, labels))
print("Silhouette Coefficient: %0.3f"
```

```
      % metrics.silhouette_score(X, labels))

core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True

# Plot optimum model of DBSCAN
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()
```