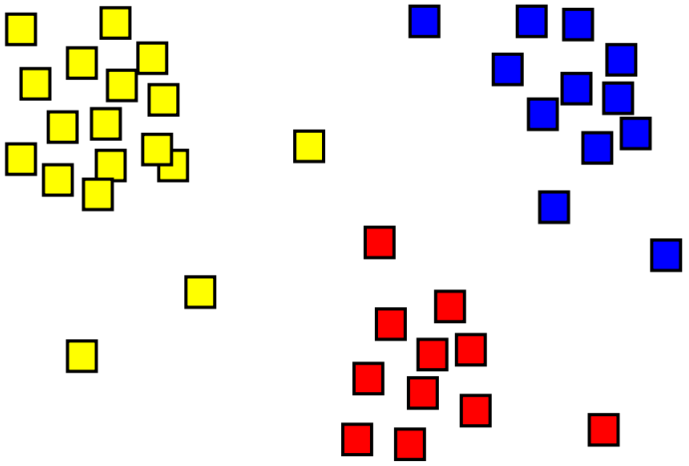
GEOMETRÍA COMPUTACIONAL

CLUSTERING



LUCAS DE TORRE

Índice

1. Introducción

Nuestro objetivo es clasificar un sistema X de 1000 elementos (teniendo cada uno de ellos dos estados) a partir de un determinado número de Vecindades de Vorónoi. Para determinar el número de de vecindades (o clusters), utilizaremos como medida el coeficiente de Silhouette ($\overline{s}$). Lo calcularemos primero mediante el algoritmo K-Means y después mediante el algoritmo DBSCAN, el cual usaremos con la métrica euclídea y con la métrica manhattan.

2. Material

Primero, haciendo uso del algoritmo de K-Means, buscamos el número de vecindades de Vorónoi (en el conjunto $\{1,2,..,15\}$) que maximiza el coeficiente de Silhoutte.

Para ello, para cada número k de vecindades, clasificamos los elementos de nuestro sistema en k clusters mediante la librería *sklearn*. Esta librería tiene un método que nos permite clasificar nuestros elementos aplicando el algoritmo de K-Means, para lo cual únicamente necesita nuestro sistema X y el número de vecindades que queremos.

Una vez clasificados los elementos en las k vecindades, esta misma librería nos permite calcular el coeficiente de Silhouette asociado a la distribución de nuestros elementos en los clusters, siempre y cuando el número k de clusters sea mayor que 1 (en caso de haber una única vecindad, el coeficiente de Silhouette es -1).

Después de calcular el valor de $\overline{s}$ para los 15 valores posibles de vecindades, dibujamos una gráfica que muestra como varía $\overline{s}$ en función del número de vecindades de Vorónoi y, por último, calculamos el máximo de estos 15 valores, que es el valor óptimo del coeficiente de Silhouette.

Por otro lado, buscamos el valor óptimo del coeficiente de Silhouette mediante el algoritmo DBSCAN, primero con la métrica euclídea y después con la Manhattan.

Para aplicar este algoritmo volvemos a hacer uso de la librería sklearn, a la que hay que indicarle la métrica utilizada (euclídea o Manhattan). Además, necesita el número mínimo de elementos en una vecindad (n_0) y el umbral de distancia (ϵ).

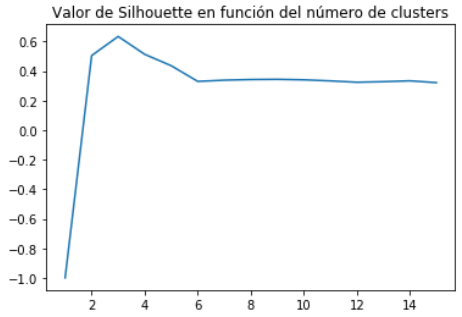
En nuestro caso, n_0 es 10, y, para cada una de las dos métricas, aplicamos el algoritmo DBSCAN con 18 valores distintos para el umbral de distancia (desde 0,1 a 1 en intervalos de 0,05).

De esta manera, una vez aplicado el algoritmo todas las veces (18 para cada métrica), dibujamos una gráfica para cada una de las dos métricas en las que se muestran cómo varía el coeficiente de Silhouette en función de cómo varía el umbral de distancia.

Por último, para cada una de las dos métricas indicamos el valor óptimo de $\overline{s}$ obtenido y mostramos el número de clusters obtenidos.

3. Resultados

KMeans



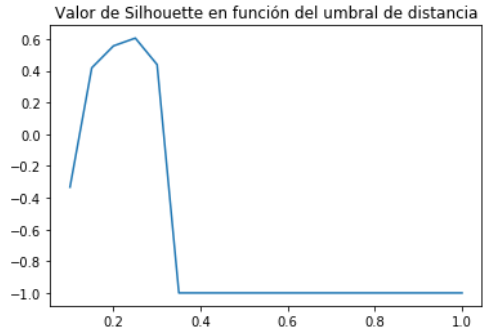
El valor óptimo de Silhouette con el algoritmo KMeans es 0.634 y se obtiene con 3 clusters

Con el algotritmo K-Means obtenemos que el valor óptimo del coeficiente de Silhoutte se obtiene con 3 clusters y es $\overline{s} = 0{,}634$.

Por otro lado, con el algoritmo DBSCAN con la métrica euclídea obtenemos el valor óptimo del coeficiente de Silhouette con un umbral de distancia $\epsilon = 0{,}25$ y es $\overline{s} = 0{,}606$. El número de clusters es 3.

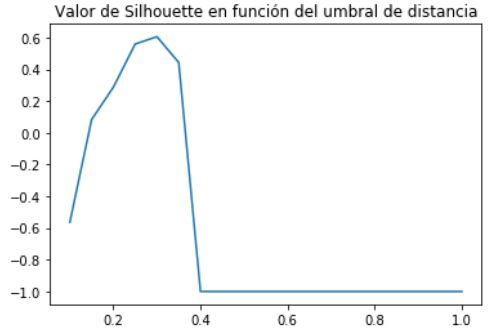
Por último, con el algoritmo DBSCAN con la métrica Manhattan obtenemos el valor óptimo del coeficiente de Silhouette con un umbral de distancia $\epsilon = 0{,}30$ y es $\overline{s} = 0{,}605$. El número de clusters es 3.

DBSCAN euclideo



El valor óptimo de Silhouette es 0.606 y se obtiene con umbral de distancia 0.25 que da lugar a 3 clusters

DBSCAN Manhattan



El valor óptimo de Silhouette es 0.605 y se obtiene con umbral de distancia 0.30 que da lugar a 3 clusters

4. Conclusiones

Observamos que el coeficiente de Silhouette más alto se obtiene con el algoritmo K-Means. En los tres casos, el valor óptimo se da con tres clusters y en todos ellos es muy parecido.

Llama especialmente la atención lo similares que son los valores obtenidos con el algoritmo de DBSCAN con las dos métricas, variando el coeficiente de Silhouette en tan solo una milésima (aunque varía el umbral de distancia).

5. Anexo A: Código

```
# -*- coding: utf-8 -*-
"""
Plantilla 1 de la pr ctica 3
Referencia:
    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 DBSCAN
from sklearn.cluster import KMeans
from sklearn import metrics
from sklearn.datasets import make_blobs

import matplotlib.pyplot as plt

#
    ################################################################################

# Aqu  tenemos definido el sistema X de 1000 elementos de dos estados
# construido a partir de una muestra aleatoria entorno a unos centros:
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=1000, centers=centers, cluster_std=0.4,
                            random_state=0)
#Si quisieramos estandarizar los valores del sistema, har amos:
#from sklearn.preprocessing import StandardScaler
#X = StandardScaler().fit_transform(X)

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

'''
A partir de aqu , c digo escrito por
-Jorge Sainero Valle
-Lucas de Torre Barrio
utilizando el c digo proporcionado
'''

# Representamos el resultado con un plot
def graficaKMeans(n_clusters):
    kmeans = KMeans(n_clusters=n_clusters, random_state=0).fit(X)
    labels = kmeans.labels_
    unique_labels = set(labels)
    colors = [plt.cm.Spectral(each)
              for each in np.linspace(0, 1, len(unique_labels))]
```

```
48
49      plt.figure(figsize=(8,4))
50      for k, col in zip(unique_labels, colors):
51          if k == -1:
52              # Black used for noise.
53              col = [0, 0, 0, 1]
54
55          class_member_mask = (labels == k)
56
57          xy = X[class_member_mask]
58          plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
59                   markeredgecolor='k', markersize=5)
60
61      plt.title('Fixed number of KMeans clusters: %d' % n_clusters)
62      plt.show()
63
64  def kMeans(k):
65      global X
66      kmeans = KMeans(n_clusters=k, random_state=0).fit(X)
67      labels = kmeans.labels_
68      #para que no falle cuando solo hay un cluster
69      if k==1:
70         silhouette = -1
71      else:
72          silhouette = metrics.silhouette_score(X, labels)
73      return silhouette
74
75  def bestSilhouetteKmeans():
76      print('KMeans')
77      sils=np.zeros(15)
78      maxi=-1
79      for i in range(1,16):
80          sils[i-1]=kMeans(i)
81          if sils[i-1]>maxi:
82              maxi=sils[i-1]
83              j=i
84      plt.title('Valor de Silhouette en funci n del n mero de clusters')
85      plt.plot(np.linspace(1,15,15),sils)
86      plt.show()
87      print("El valor  ptimo  de Silhouette con el algoritmo KMeans es %0.3f"
        % maxi, "y se obtiene con", j, "clusters")
88      graficaKMeans(j);
89      print('\n\n\n')
90
91  def graficaDBSCAN(epsilon,metrica):
92      db = DBSCAN(eps=epsilon, min_samples=10, metric=metrica).fit(X)
93      core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
94      core_samples_mask[db.core_sample_indices_] = True
95      labels = db.labels_
96      unique_labels = set(labels)
97      colors = [plt.cm.Spectral(each)
98                for each in np.linspace(0, 1, len(unique_labels))]
99
```

```
100     plt.figure(figsize=(8,4))
101     for k, col in zip(unique_labels, colors):
102         if k == -1:
103             # Black used for noise.
104             col = [0, 0, 0, 1]
105 
106         class_member_mask = (labels == k)
107 
108         xy = X[class_member_mask]
109         plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
110                  markeredgecolor='k', markersize=5)
111     #Restamos uno para no contar el ruido
112     plt.title('Fixed number of DBSCAN '+metrica+' clusters: %d' % (len(
    unique_labels)-1))
113     plt.show()
114 
115 def dbscan(epsilon,metrica):
116     db = DBSCAN(eps=epsilon, min_samples=10, metric=metrica).fit(X)
117 
118     core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
119     core_samples_mask[db.core_sample_indices_] = True
120     labels = db.labels_
121 
122     # Number of clusters in labels, ignoring noise if present.
123     n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
124     #para que no falle cuando solo hay un cluster
125     if n_clusters_<=1:
126         silhouette=-1
127     else:
128         silhouette=metrics.silhouette_score(X, labels)
129     return [n_clusters_,silhouette]
130 
131 def bestSilhouetteDBSCAN():
132     print('DBSCAN euclideo')
133     silsEuc=[[dbscan(i,'euclidean'),i] for i in np.arange(0.1,1.05,0.05)]
134     silsMan=[[dbscan(i,'manhattan'),i] for i in np.arange(0.1,1.05,0.05)]
135     n_clusters_euc=0
136     maxieuc=-1
137     epsilon_euc=0
138     n_clusters_man=0
139     maximan=-1
140     epsilon_man=0
141 
142     for sil in silsEuc:
143         if (sil[0][1]>maxieuc):
144             maxieuc = sil[0][1]
145             n_clusters_euc = sil[0][0]
146             epsilon_euc=sil[1]
147     plt.title('Valor de Silhouette en funci n del umbral de distancia')
148     plt.plot([silsEuc[i][1] for i in range(len(silsEuc))],[silsEuc[i][0][1]
     for i in range(len(silsEuc))])
149     plt.show()
150     print("El valor  ptimo  de Silhouette es %0.3f" % maxieuc, "y se
```

```
    obtiene con umbral de distancia %0.2f" % epsilon_euc, "que da lugar a",
    n_clusters_euc,"clusters")
151     graficaDBSCAN(epsilon_euc,'euclidean')
152     print('\n\n\n')
153 
154     print('DBSCAN Manhattan')
155     for sil in silsMan:
156         if (sil[0][1]>maximan):
157             maximan = sil[0][1]
158             n_clusters_man = sil[0][0]
159             epsilon_man=sil[1]
160     plt.title('Valor de Silhouette en funci n del umbral de distancia')
161     plt.plot([silsMan[i][1] for i in range(len(silsMan))],[silsMan[i][0][1]
        for i in range(len(silsMan))])
162     plt.show()
163     print("El valor  ptimo  de Silhouette es %0.3f" % maximan, "y se
    obtiene con umbral de distancia %0.2f" % epsilon_man, "que da lugar a",
    n_clusters_man,"clusters")
164     graficaDBSCAN(epsilon_euc,'manhattan')
165 
166 bestSilhouetteKmeans()
167 bestSilhouetteDBSCAN()
```