



Actividad Individual:

Clustering.

APRENDIZAJE AUTOMÁTICO

Universidad de Huelva

*Grado en Ingeniería Informática
Especialidad en Computación
Curso 2024/25
Manuel Ramírez Ballesteros*

- Introducción:

En esta actividad se ha empleado un *dataset* con datos de 3685 estaciones que podemos encontrar en el archivo *Stations_2024.csv*:

Archivo	Edición	Formato	Ver	Ayuda
"id","station_id","latitude","longitude","created_at","updated_at","deleted_at","error"				
1,1,50.46254,4.86960,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
2,2,50.46424,4.86520,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
3,3,50.47536,4.84661,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
4,4,50.46060,4.84284,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
5,5,50.46675,4.84834,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
6,6,50.46387,4.86114,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
7,7,50.46687,4.88484,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
8,8,50.45507,4.87558,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
9,9,50.45786,4.86697,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
10,10,50.46452,4.83985,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
11,11,50.46786,4.87352,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
12,12,50.46155,4.87609,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
13,13,50.44822,4.85951,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
14,14,50.46274,4.83720,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
15,15,50.46448,4.84929,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
16,16,50.46446,4.85644,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
17,17,50.47069,4.85280,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
18,18,50.46697,4.86546,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
19,19,50.46204,4.86499,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				
20,20,50.46364,4.87353,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0				

Al analizar los datos, observamos que 2 de estas estaciones contienen valores anómalos en el atributo *longitud* ya que, en términos de coordenadas geográficas, la latitud se mide entre -90° y 90° con 0° en el ecuador, mientras que la longitud se mide entre -180° y 180°, con 0° en el meridiano de Greenwich. En el *dataset* encontramos estos dos registros:

3322,1558,37.40707,-6002969.00000,2023-01-20 06:50:03.000,2023-01-20 07:02:04.000,,1
3323,3242,43.65586,-79.38941,2023-01-20 07:55:36.000,2023-01-26 06:51:07.000,,0
3324,3243,37.37585,-5925090.00000,2023-01-20 08:54:40.000,2023-01-20 09:15:39.000,,1

Observando los decimales, podemos entender que los registros pueden ser correctos al considerar los valores -60.02969 y -59.25090 ya que, en el resto de estaciones, el atributo *longitud* contiene 5 decimales, por lo que se ha decidido no excluir estos registros completos y generar un nuevo fichero *Stations_Bueno.csv* con los dos valores corregidos, que es el que emplearemos para realizar el *clustering*.

- **Ejercicio 1: Probar distintos números de clusters usando la medida entre centroides, justificar de cuantas ciudades está compuesto el dataset y etiquetar cada estación con un número de ciudad.**

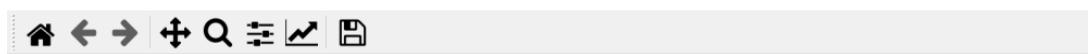
Primero se ha empleado una librería externa para aplicar K-Means para agrupar las estaciones geográficas por latitud y longitud y determinar el número óptimo de clusters basándonos en el Silhouette Score, que es una medida de la similitud de un objeto a su propio grupo (cohesión) en comparación con otros (separación). En el fichero *Clustering1_libreria.py* se ha definido la función *encontrar_clusters* para determinar el número óptimo de clusters empleando dicha métrica que recibe los datos con las coordenadas a agrupar, el número máximo de clusters a evaluar y un umbral para guardar los clusters que lo superan. Se realiza un bucle para evaluar los distintos valores de k creando un modelo de K-Means para cada valor, entrenando el modelo con los datos y midiendo la calidad del agrupamiento. Si el Silhouette Score supera el umbral establecido (0.97), se guarda el modelo y la información para graficar los puntos agrupados por clusters. También se genera un gráfico con el número de clusters y la métrica empleada. Finalmente se devuelve una lista con los k evaluados (rangoClusters), una lista con las diferentes métricas obtenidas (puntuaciones) y los modelos y puntuaciones que han superado el umbral (clustersValidos).

```

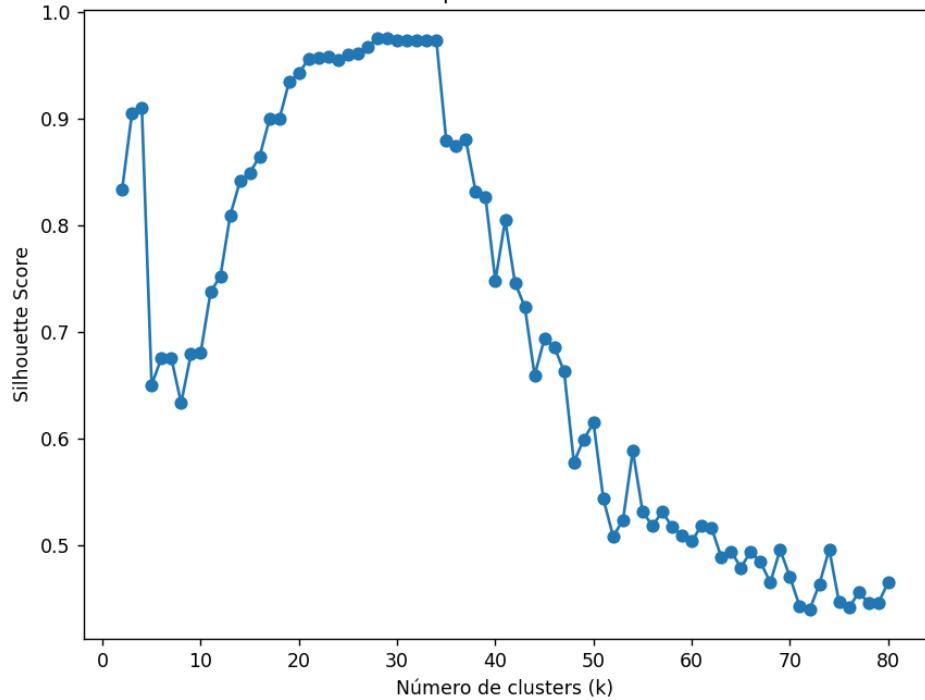
1  import pandas as pd
2  from sklearn.cluster import KMeans
3  from sklearn.metrics import silhouette_score
4  import matplotlib.pyplot as plt
5
6  # Función para determinar el número óptimo de clusters basado en el Silhouette Score
7  def encontrar_clusters(data, max_clusters=80, silhouette_threshold=0.97):
8      puntuaciones = []
9      rangoClusters = range(2, max_clusters + 1)
10     clustersValidos = []
11
12     for k in rangoClusters:
13         kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
14         kmeans.fit(data)
15         silhouette = silhouette_score(data, kmeans.labels_)
16         puntuaciones.append(silhouette)
17
18         # Guardar los clusters con silhouette score > threshold
19         if silhouette > silhouette_threshold:
20             |   clustersValidos.append((k, kmeans, silhouette))
21
22     # Graficar el Silhouette Score
23     plt.figure(figsize=(8, 6))
24     plt.plot(rangoClusters, puntuaciones, marker='o')
25     plt.title('Silhouette Score para diferentes valores de k')
26     plt.xlabel('Número de clusters (k)')
27     plt.ylabel('Silhouette Score')
28     plt.show()
29
30     # Graficar los clusters para valores de k con silhouette > threshold
31     for k, modelo, silhouette in clustersValidos:
32         print(f"Graficando clusters para k={k} con Silhouette Score={silhouette:.2f}")
33         plt.figure(figsize=(8, 6))
34         plt.scatter(data['longitude'], data['latitude'], c=modelo.labels_, cmap='viridis', s=50)
35         plt.scatter(modelo.cluster_centers_[:, 0], modelo.cluster_centers_[:, 1], c='red', marker='x', s=200)
36         plt.title(f"Clusters para k={k} (Silhouette Score={silhouette:.2f})")
37         plt.xlabel('Latitud')
38         plt.ylabel('Longitud')
39         plt.xlim(-90, 90)
40         plt.ylim(-180, 180)
41         plt.show()
42
43     return rangoClusters, puntuaciones, clustersValidos
44

```

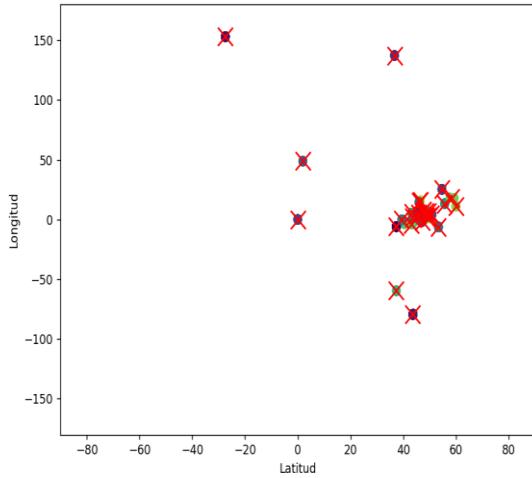
Figure 1



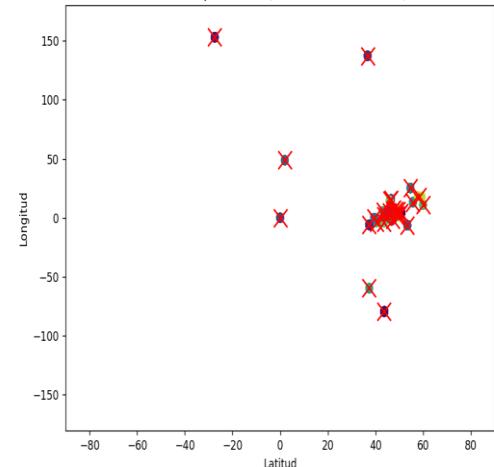
Silhouette Score para diferentes valores de k

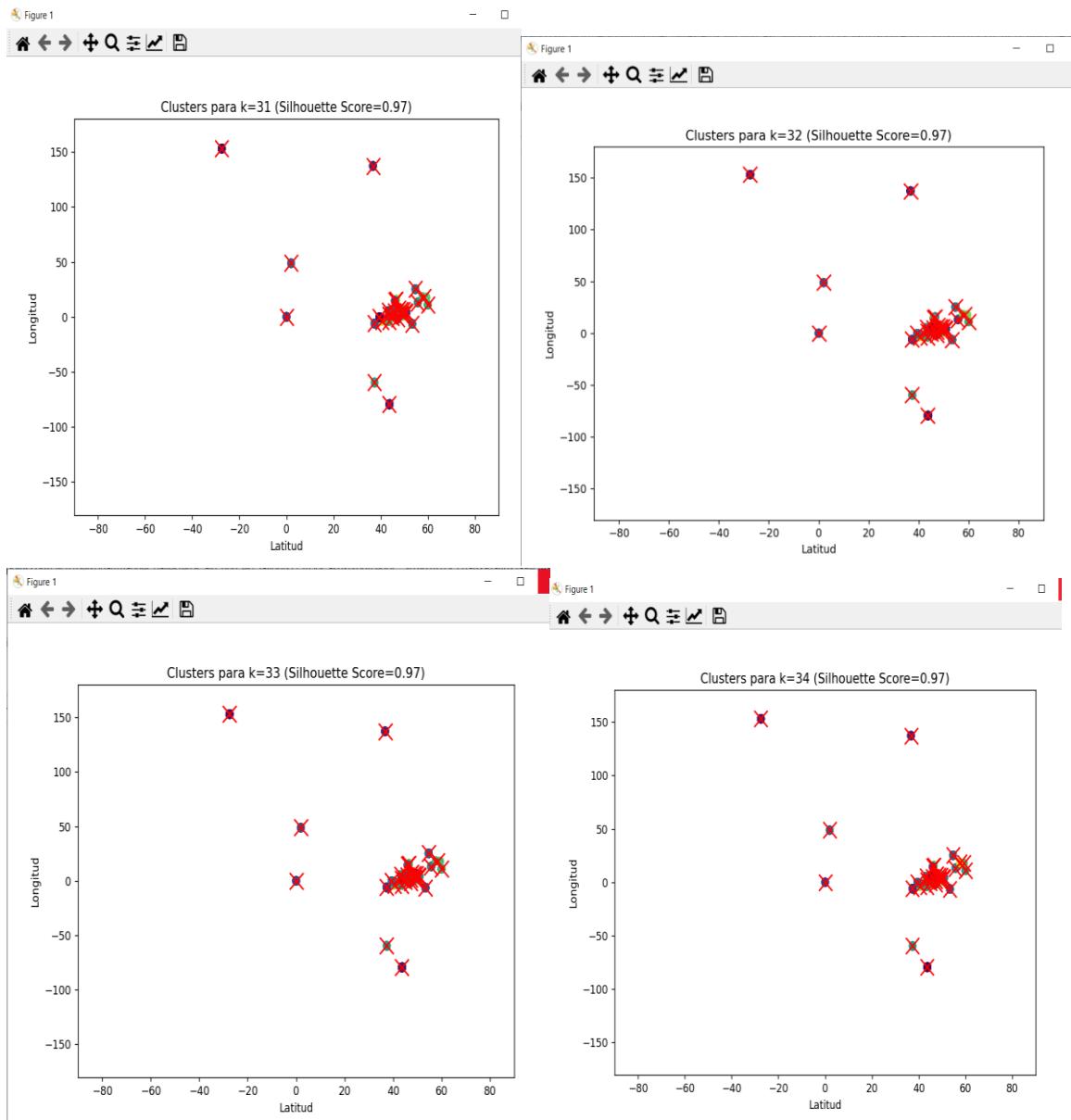


Clusters para k=28 (Silhouette Score=0.98)



Clusters para k=29 (Silhouette Score=0.98)





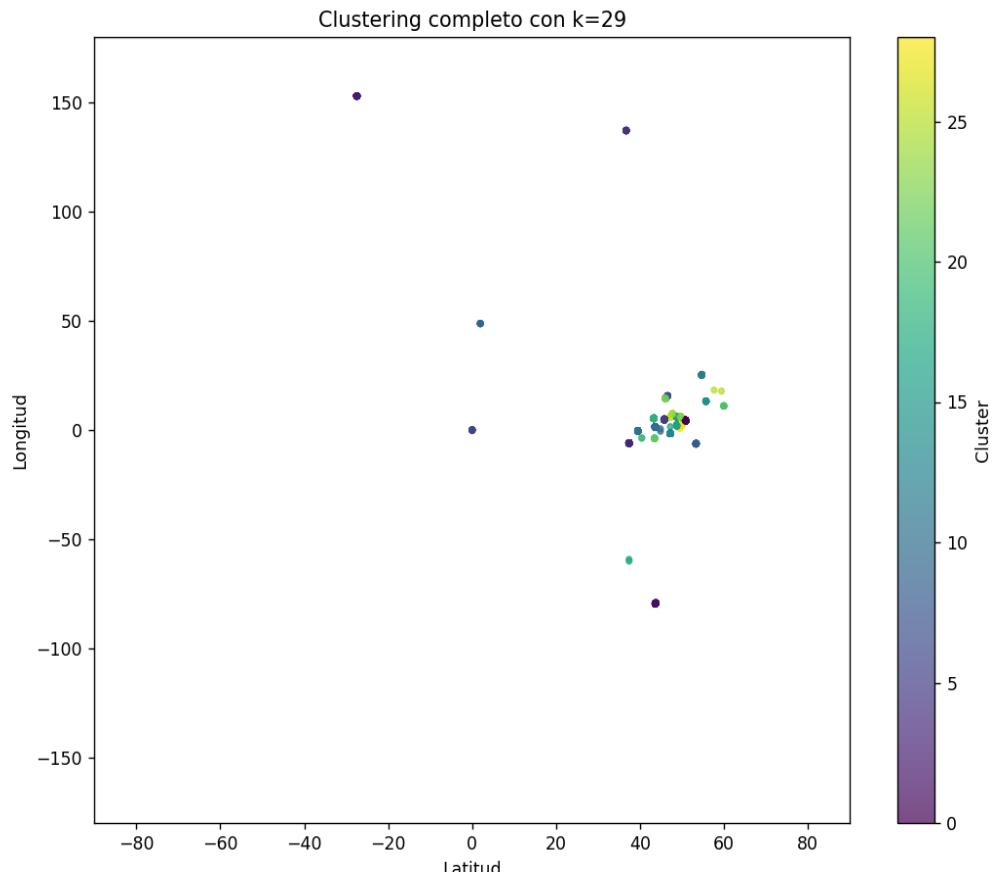
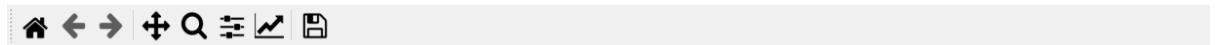
Para ejecutar dicha función se han cargado los datos desde el csv y se ha llamado a `encontrar_clusters` evaluando desde 2 a 80 clusters. El número óptimo de clusters será quien tenga el índice más alto en puntuaciones y para ese valor se aplica K-Means y se le asigna a cada grupo su etiqueta de cluster (`city_cluster`). Finalmente se muestran los puntos agrupados por su etiqueta de cluster y guardando los datos con sus nuevas etiquetas en un nuevo archivo `Stations_Clustered_library.csv`.

```

47 # Leer el archivo
48 df = pd.read_csv('Stations_Bueno.csv')
49 coordenadas = df[['latitude', 'longitude']]
50
51 # Determinar el número óptimo de clusters
52 rangoClusters, puntuaciones, clustersValidos = encontrar_clusters(coordenadas)
53
54 # Seleccionar el número óptimo de clusters según silhouette score
55 kOptimo = puntuaciones.index(max(puntuaciones)) + 2
56 print(f"Número óptimo de clusters: {kOptimo}")
57
58 # Aplicar K-Means con el número óptimo de clusters
59 kmeans = KMeans(n_clusters=kOptimo, random_state=42, n_init=10)
60 coordenadas['city_cluster'] = kmeans.fit_predict(coordenadas)
61
62 # Mapear los clusters al DataFrame original
63 df['city_cluster'] = coordenadas['city_cluster']
64
65 # Visualizar todos los puntos con los clusters asignados
66 plt.figure(figsize=(10, 8))
67 plt.scatter(coordenadas['latitude'], coordenadas['longitude'], c=coordenadas['city_cluster'], cmap='viridis', s=10, alpha=0.7)
68 plt.title(f"Clustering completo con k={kOptimo}")
69 plt.xlabel('Latitud')
70 plt.ylabel('Longitud')
71 plt.xlim([-90, 90])
72 plt.ylim([-180, 180])
73 plt.colorbar(label='Cluster')
74 plt.show()
75
76 # Guardar los datos con los clusters asignados en un archivo CSV
77 df.to_csv('Stations_Clustered_library.csv', index=False)
78 print("Etiquetas de cluster guardadas en 'Stations_Clustered_library.csv'")

```

Figure 1



A continuación, se ha hecho una implementación propia del algoritmo K-Means en el fichero *Clustering1.py*. Para ello se ha diseñado una función para inicializar los centroides escogiendo el primero aleatoriamente del conjunto de datos y

seleccionar los demás iterativamente calculando la distancia mínima de cada punto al centroide más cercano y seleccionando el siguiente centroide con una probabilidad proporcional al cuadrado de la distancia (los puntos más alejados tienen mayor probabilidad de ser seleccionados), lo que nos proporciona una mayor dispersión de los centroides. Esta función nos devuelve un array con los centroides inicializados.

```
6 # Función para inicialización K-Means
7 def inicializar_centroides(data, clusters, random_state=42):
8     np.random.seed(random_state)
9     centroides = []
10
11     # Seleccionar el primer centroide aleatoriamente
12     centroides.append(data[np.random.choice(len(data))])
13
14     for _ in range(1, clusters):
15         # Calcular la distancia mínima al centroide más cercano
16         distancias = np.min(np.linalg.norm(data[:, np.newaxis] - np.array(centroides), axis=2), axis=1)
17         # Seleccionar un nuevo centroide proporcional a la distancia al cuadrado
18         prob = distancias ** 2 / np.sum(distancias ** 2)
19         centroide = data[np.random.choice(len(data), p=prob)]
20         centroides.append(centroide)
21
22     return np.array(centroides)
```

Se ha diseñado una función para implementar el algoritmo K-Means que recibe la matriz de datos (*data*), el número de clusters deseados (*clusters*), un número máximo de iteraciones (*max_iter*), un valor de tolerancia para detectar la convergencia basada en los cambios de los centroides (*tol*) y una semilla para una inicialización que pueda reproducirse (*random_state*). En primer lugar, se llama a la función *inicializar_centroides* para obtener los centroides iniciales y se itera hasta alcanzar la convergencia o superar el número máximo de iteraciones de manera que asignamos los distintos puntos al centroide más cercano mediante la distancia euclídea. Para cada cluster, se calcula la media de los puntos asignados salvo si está vacío (no tiene puntos asignados), en cuyo caso se vuelve a recolocar dicho centroide aleatoriamente. Por último, se verifica la convergencia de dos formas: Si las etiquetas no cambian (etiquetas estables) o si los centroides no cambian significativamente (centroides estables), devolviendo los centroides finales y las etiquetas de los puntos.

```

24  # Implementación del algoritmo K-Means con múltiples inicializaciones
25  def kmeans(data, clusters, max_iter=300, tol=1e-4, random_state=42):
26      # Inicializar centroides con K-Means
27      centroides = inicializar_centroides(data, clusters, random_state=random_state)
28      prevEtiquetas = None
29
30      for iter in range(max_iter):
31          # Asignar puntos al centroide más cercano
32          distancias = np.linalg.norm(data[:, np.newaxis] - centroides, axis=2) # Distancia euclídea
33          etiquetas = np.argmin(distancias, axis=1)
34
35          # Recalculiar centroides
36          NuevosCentroides = []
37          for i in range(clusters):
38              puntosCluster = data[etiquetas == i]
39              if len(puntosCluster) > 0:
40                  # Calcular el nuevo centroide como la media de los puntos
41                  NuevosCentroides.append(puntosCluster.mean(axis=0))
42              else:
43                  # Cluster vacío: Recolocar el centroide aleatoriamente
44                  NuevosCentroides.append(data[np.random.choice(len(data))])
45
46          NuevosCentroides = np.array(NuevosCentroides)
47
48          # Verificar convergencia
49          if prevEtiquetas is not None and np.array_equal(etiquetas, prevEtiquetas):
50              print(f"Convergencia alcanzada en la iteración {iter} (etiquetas estables)")
51              break
52          if np.all(np.abs(NuevosCentroides - centroides) < tol):
53              print(f"Convergencia alcanzada en la iteración {iter} (centroides estables)")
54              break
55
56          centroides = NuevosCentroides
57          prevEtiquetas = etiquetas
58
59      return centroides, etiquetas

```

También se ha diseñado una función para ejecutar el algoritmo K-Means varias veces con distintas inicializaciones, ya que con las funciones anteriores no se conseguían resultados similares a los obtenidos empleando la librería *KMeans*. A esta función tenemos que pasarle como parámetro también el número de inicializaciones que queremos hacer (*n_init*) que serán 10 al igual que empleamos en la librería externa, ejecutando el algoritmo esa cantidad de veces con diferentes semillas y calculando la métrica para evaluar la calidad del clustering. Se selecciona la mejor opción y se devuelven los mejores centroides, etiquetas y métricas:

```

61  # Función para ejecutar K-Means múltiples veces y seleccionar el mejor resultado
62  def kmeans_multiple(data, clusters, n_init=10, max_iter=300, tol=1e-4, random_state=42):
63      mejoresCentroides = None
64      mejoresEtiquetas = None
65      mejorSilhouette = -1 # Comenzamos con el peor valor posible para el silhouette score
66
67      for i in range(n_init):
68          centroides, etiquetas = kmeans(data, clusters, max_iter=max_iter, tol=tol, random_state=random_state + i)
69
70          # Calcular el silhouette score
71          silhouette = silhouette_score(data, etiquetas)
72
73          if silhouette > mejorSilhouette:
74              mejorSilhouette = silhouette
75              mejoresCentroides = centroides
76              mejoresEtiquetas = etiquetas
77
78      return mejoresCentroides, mejoresEtiquetas, mejorSilhouette
79

```

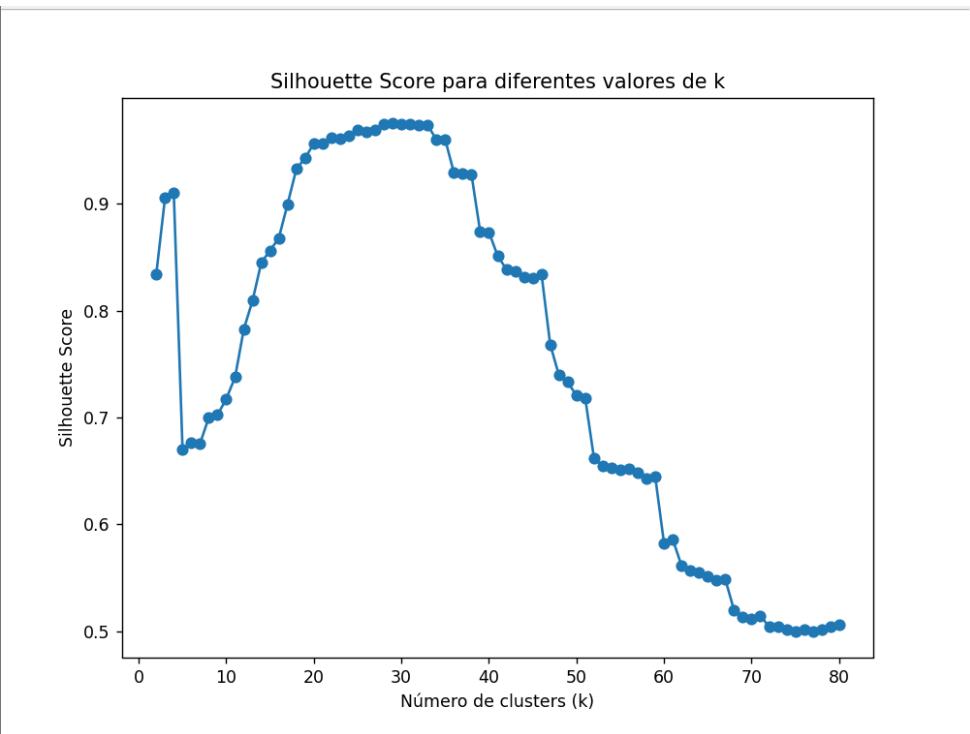
La función *determinar_clusters* es similar a la empleada en el primer fichero .py donde se determina el número óptimo de clusters en base a la métrica para

diferentes valores de k ejecutando *kmeans_multiple* para obtener el mejor modelo y guardando aquellos que superan cierto valor umbral. Se visualiza la relación entre el número de clusters y la métrica, así como los clusters válidos y se devuelve el rango de clusters evaluados, las puntuaciones, los clusters válidos según el umbral y el mejor valor de k con sus centroides y etiquetas para evitar repetir estos cálculos posteriormente.

```

80  def determinar_clusters(data, max_clusters=80, silhouette_threshold=0.97, n_init=10):
81      puntuaciones = []
82      rangoClusters = range(2, max_clusters + 1)
83      clustersValidos = []
84      mejor_k = None
85      mejores_centroides = None
86      mejores_etiquetas = None
87      mejor_silhouette = -1
88
89      for k in rangoClusters:
90          centroides, etiquetas, silhouette = kmeans_multiple(data, clusters=k, n_init=n_init)
91          puntuaciones.append(silhouette)
92
93          if silhouette > silhouette_threshold:
94              clustersValidos.append((k, centroides, etiquetas, silhouette))
95
96          if silhouette > mejor_silhouette: # Guardar el mejor k
97              mejor_k = k
98              mejores_centroides = centroides
99              mejores_etiquetas = etiquetas
100             mejor_silhouette = silhouette
101
102     # Visualizar el Silhouette Score
103     plt.figure(figsize=(8, 6))
104     plt.plot(rangoClusters, puntuaciones, marker='o')
105     plt.title('Silhouette Score para diferentes valores de k')
106     plt.xlabel('Número de clusters (k)')
107     plt.ylabel('Silhouette Score')
108     plt.show()
109
110     # Graficar los clusters para valores de k con silhouette > threshold
111     for k, centroides, etiquetas, silhouette in clustersValidos:
112         print(f"Graficando clusters para k={k} con Silhouette Score={silhouette:.2f}")
113         plt.figure(figsize=(8, 6))
114         plt.scatter(data[:, 0], data[:, 1], c=etiquetas, cmap='viridis', s=50)
115         plt.scatter(centroides[:, 0], centroides[:, 1], c='red', marker='x', s=200)
116         plt.title(f"Clusters para k={k} (Silhouette Score={silhouette:.2f})")
117         plt.xlabel('Latitud')
118         plt.ylabel('Longitud')
119         plt.xlim([-50, 90])
120         plt.ylim([-180, 180])
121         plt.show()
122
123     return rangoClusters, puntuaciones, clustersValidos, mejor_k, mejores_centroides, mejores_etiquetas
124

```



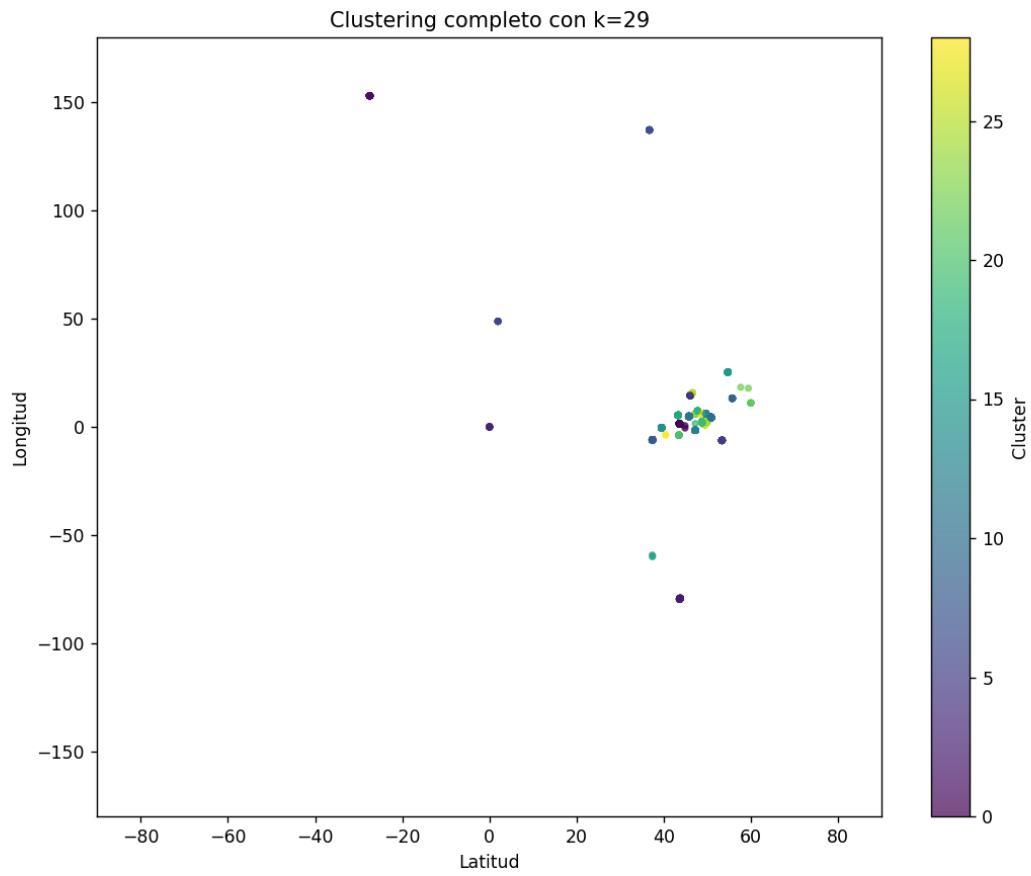
Finalmente, se lee el archivo csv y se determina el número óptimo de clusters llamando a la función *determinar_clusters*, visualizando los puntos y guardando las etiquetas en *Stations_Clustered*.

```

126 # Leer el dataset
127 df = pd.read_csv('Stations_Bueno.csv')
128 coordenadas = df[['latitude', 'longitude']].values
129
130 # Llamar a determinar_clusters y recuperar valores directamente
131 rangoClusters, puntuaciones, clustersValidos, kOptimo, centroides, etiquetas = determinar_clusters(coordenadas)
132
133 print(f"Número óptimo de clusters: {kOptimo}")
134
135 # Asignar clusters al DataFrame original
136 df['city_cluster'] = etiquetas
137
138 # Visualizar los puntos
139 plt.figure(figsize=(10, 8))
140 plt.scatter(coordenadas[:, 0], coordenadas[:, 1], c=etiquetas, cmap='viridis', s=10, alpha=0.7)
141 plt.title(f"Clustering completo con k={kOptimo}")
142 plt.xlabel('Latitud')
143 plt.ylabel('Longitud')
144 plt.xlim([-90, 90])
145 plt.ylim([-180, 180])
146 plt.colorbar(label='Cluster')
147 plt.show()
148
149 # Guardar las etiquetas en el archivo
150 df.to_csv('Stations_Clustered.csv', index=False)
151 print("Etiquetas de cluster guardadas en 'Stations_Clustered.csv'")
152

```

Como vemos, los resultados obtenidos son similares a los que se consiguen empleando la librería externa, donde el número óptimo de clusters es 29.



Stations_Clustered: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
id,station_id,latitude,longitude,created_at,updated_at,deleted_at,error,city_cluster
1,1,50.46254,4.8696,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
2,2,50.46424,4.8652,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
3,3,50.47536,4.84661,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
4,4,50.4606,4.84284,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
5,5,50.46675,4.84834,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
6,6,50.46387,4.86114,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
7,7,50.46687,4.88484,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
8,8,50.45507,4.87558,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
9,9,50.45786,4.86697,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
10,10,50.46452,4.83985,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
11,11,50.46786,4.87352,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
12,12,50.46155,4.87609,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
13,13,50.44822,4.85951,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
14,14,50.46274,4.8372,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
15,15,50.46448,4.84929,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
16,16,50.46446,4.85644,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
17,17,50.47069,4.8528,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
18,18,50.46697,4.86546,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
19,19,50.46204,4.86499,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
20,20,50.46364,4.87353,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
21,21,50.45189,4.86884,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
22,22,50.46735,4.86872,2023-01-19 12:26:48.000,2023-05-20 02:02:22.000,,0,10
```

- ***Ejercicio 2: Elegir cualquiera de las ciudades y, usando la distancia entre los elementos más cercanos, quedarnos con 10 clusters.***

Para este ejercicio se ha generado el archivo *Clustering2.py*, en el que se carga el archivo generado anteriormente (*Stations_Clustered.csv*) con los datos de las estaciones y sus clusters correspondientes y se realiza un clustering jerárquico sobre las estaciones dentro de una ciudad específica (*idCiudad*) filtrando las filas correspondientes a dicho cluster y las columnas relevantes. También se genera un dendrograma (árbol binario donde ejemplos en el mismo grupo en un nivel k permanecerán en el mismo grupo en niveles más altos mostrando la similitud de los grupos creados). Utiliza el método de enlace *Single Linkage* que agrupa clusters en base a la distancia mínima entre puntos.

Se aplica el algoritmo de clustering aglomerativo que comienza con cada punto como un cluster individual y los combina iterativamente en clusters mayores, dividiendo las estaciones en un número de subclusters especificados (numClusters). El resultado (subcluster al que pertenece cada estación) se guarda en una nueva columna (*sub_cluster*) y se muestran los resultados graficando las estaciones con colores distintos para cada subcluster.

```

1  import pandas as pd
2  import scipy.cluster.hierarchy as sch
3  from sklearn.cluster import AgglomerativeClustering
4  import matplotlib.pyplot as plt
5
6  # Leer el dataset con etiquetas de clusters
7  df = pd.read_csv('Stations_Clustered.csv')
8
9  # Función para clustering jerárquico dentro de una ciudad
10 def clustering_jerarquico_ciudad(idCiudad, numClusters=80):
11     # Filtrar estaciones en la ciudad seleccionada
12     datosCiudad = df[df['city_cluster'] == idCiudad]
13     coordenadasCiudad = datosCiudad[['latitude', 'longitude']]
14
15     if coordenadasCiudad.empty:
16         print(f"No hay estaciones en la ciudad con ID {idCiudad}.")
17         return
18
19     # Crear dendrograma
20     plt.figure(figsize=(10, 5))
21     sch.dendrogram(sch.linkage(coordenadasCiudad, method='single'))
22     plt.title(f"Dendrograma - Ciudad {idCiudad}")
23     plt.xlabel('Estaciones')
24     plt.ylabel('Distancia')
25     plt.show()
26
27     # Aplicar clustering jerárquico aglomerativo
28     hc = AgglomerativeClustering(n_clusters=min(numClusters, len(coordenadasCiudad)), affinity='euclidean', linkage='single')
29     datosCiudad['sub_cluster'] = hc.fit_predict(coordenadasCiudad)
30
31     # Mostrar resultados
32     print(f"SubClusters dentro de la ciudad {idCiudad}:")
33     print(datosCiudad[['station_id', 'sub_cluster']])
34
35     # Graficar los subclusters
36     plt.scatter(coordenadasCiudad['latitude'], coordenadasCiudad['longitude'], c=datosCiudad['sub_cluster'], cmap='viridis', s=50)
37     plt.title(f"Subclusters en la ciudad {idCiudad}")
38     plt.xlabel('Latitud')
39     plt.ylabel('Longitud')
40     plt.show()
41
42     # Ejemplo: Subdividir la ciudad con ID 5 en 10 clusters
43     clustering_jerarquico_ciudad(idCiudad=5, numClusters=10)
```

Para la ciudad 5 se obtiene:

Subclusters dentro de la ciudad 5:		
station_id	sub_cluster	
205	206	2
206	207	1
207	208	1
208	209	1
209	210	3
...
3488	3311	3
3492	3315	1
3572	207	1
3584	3356	1
3631	3362	0

