Clase16 IMA539

Mg. Alejandro Ferreira Vergara

May 22, 2023

1 Clustering Jerárquico y DBSCAN

1.1 Organizar los clusters como un Árbol Jerárquico

Una ventaja de este tipo de algoritmos es que no es necesario especificar el número de clusters a priori.

Los dos enfoques principales del clustering jerárquico son el **aglomerativo** y el **jerárquico divisivo**.

En el **clustering jerárquico divisivo**, comenzamos con un clúster que abarca todas las muestras, y dividimos iterativamente en clusters más pequeños hasta que cada uno de estos contenga solo una muestra.

La **agrupación aglomerativa** adopta el enfoque opuesto al enfoque divisivo. Comenzamos con cada muestra como un clúster individual y fusionamos los pares de clusters más cercanos hasta que sólo queda un clúster.

1.1.1 Agrupación Jarárquica Aglomerativa

Los dos algoritmos estándar para la agrupación jerárquica aglomerativa son la **vinculación simple** y la **vinculación completa**.

En el caso de la **vinculación simple**, se calculan las distancias entre los miembros más similares de cada par de conglomerados y se fusionan los dos conglomerados cuya distancia entre los miembros más similares sea la menor.

El enfoque de **vinculación completa** es similar al de vinculación simple, pero en lugar de comparar los miembros más similares de cada par de conglomerados, comparamos los miembros más disímiles para realizar la fusión.

Otros algoritmos comúnmente utilizados para la agrupación jerárquica aglomerativa son la **vinculación promedio** y la **vinculación de Ward**.

En la vinculación promedio, se fusionan los pares de conglomerados basándose en las distancias medias mínimas entre todos los miembros del grupo en los dos conglomerados. En la vinculación de Ward, se fusionan los dos clusters que conducen al mínimo incremento de la SSE total dentro del cluster.

1.1.2 Algoritmo para clustering jerárquico con vinculación completa

1. Calcular la **matriz de distancias** de todas las muestras.

2. Representar cada punto de datos como un clúster único.
3. Combinar los dos clusters más cercanos en función de la distancia entre los miembros más disímiles (distantes).
4. Actualizar la matriz de similitud.
5. Repetir los pasos 2-4 hasta que quede un único cluster.

[ ]:
```
import pandas as pd
import numpy as np

np.random.seed(123)

variables = ['X', 'Y', 'Z']
labels = ['ID_0', 'ID_1', 'ID_2', 'ID_3', 'ID_4']

X = np.random.random_sample([5, 3])*10
df = pd.DataFrame(X, columns=variables, index=labels)
df
```

Paso 1

[ ]:
```
from scipy.spatial.distance import pdist, squareform

row_dist = pd.DataFrame(squareform(pdist(df, metric='euclidean')),␣
 ↪columns=labels, index=labels)
row_dist
```

[ ]:
```
from scipy.cluster.hierarchy import linkage

row_clusters = linkage(pdist(df, metric='euclidean'), method='complete')

pd.DataFrame(row_clusters,
             columns=['row label 1', 'row label 2','distance', 'no. of items in␣
 ↪clust.'],
             index=['cluster %d' % (i + 1) for i in range(row_clusters.
 ↪shape[0])])
```

[ ]:
```
row_clusters = linkage(df.values, method='complete', metric='euclidean')

pd.DataFrame(row_clusters,
             columns=['row label 1', 'row label 2','distance', 'no. of items in␣
 ↪clust.'],
             index=['cluster %d' % (i + 1) for i in range(row_clusters.
 ↪shape[0])])
```

[ ]:
```
from scipy.cluster.hierarchy import dendrogram
import matplotlib.pyplot as plt
```

```
row_dendr = dendrogram(row_clusters, labels=labels)
plt.tight_layout()
plt.ylabel('Euclidean distance')
#plt.savefig('images/11_11.png', dpi=300,bbox_inches='tight')
plt.show()
```

1.1.3 Dendrograma y Mapa de calor

[ ]:
```
# plot row dendrogram
fig = plt.figure(figsize=(8, 8), facecolor='white')
axd = fig.add_axes([0.09, 0.1, 0.2, 0.6])

# note: for matplotlib < v1.5.1, please use orientation='right'
row_dendr = dendrogram(row_clusters, orientation='left')

# reorder data with respect to clustering
df_rowclust = df.iloc[row_dendr['leaves'][::-1]]

axd.set_xticks([])
axd.set_yticks([])

# remove axes spines from dendrogram
for i in axd.spines.values():
    i.set_visible(False)

# plot heatmap
axm = fig.add_axes([0.23, 0.1, 0.6, 0.6])  # x-pos, y-pos, width, height
cax = axm.matshow(df_rowclust, interpolation='nearest', cmap='hot_r')
fig.colorbar(cax)
axm.set_xticklabels([''] + list(df_rowclust.columns))
axm.set_yticklabels([''] + list(df_rowclust.index))

#plt.savefig('images/11_12.png', dpi=300)
plt.show()
```

1.1.4 Ahora con scikit-learn

Fuente: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html

[ ]:
```
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=None,affinity='euclidean',linkage='complete',distance_th
)

labels = ac.fit_predict(X)
```

```
print('Cluster labels: %s' % labels)
```

[ ]:
```
ac =␣
 ↪AgglomerativeClustering(n_clusters=None,affinity='euclidean',linkage='complete',distance_th
 ↪)

labels = ac.fit_predict(X)
print('Cluster labels: %s' % labels)
```

[ ]:
```
ac =␣
 ↪AgglomerativeClustering(n_clusters=3,affinity='euclidean',linkage='complete')

labels = ac.fit_predict(X)
print('Cluster labels: %s' % labels)
```

1.2 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

- **Algoritmo de clustering basado en densidades**
- **No se requiere especificar el número de clusters**

No hace suposiciones sobre clusters esféricos como K-means, ni particiona el conjunto de datos en jerarquías que requieren un punto de corte manual.

El clustering basado en la densidad asigna etiquetas de cluster basadas en regiones densas de puntos.

En DBSCAN, la noción de densidad se define como el número de puntos dentro de un radio determinado llamado ε.

El algoritmo DBSCAN asigna una etiqueta especial a cada muestra (punto) utilizando los siguientes criterios:

- Un punto se considera un **punto central o núcleo** si al menos un número determinado de puntos (MinPts) vecinos se encuentra dentro del radio especificado ε.
- Un **punto fronterizo** es un punto que tiene menos vecinos que MinPts dentro de ε, pero se encuentra dentro del radio ε de un punto central.
- Todos los demás puntos que no son ni núcleo ni frontera se consideran **puntos de ruido**.

Tras etiquetar los puntos como núcleo, frontera o ruido, el algoritmo DBSCAN puede resumirse en dos pasos:

1. Formar una agrupación separada para cada punto central o grupo conectado de puntos centrales (los puntos centrales están conectados si no están más lejos que ε).
2. Asignar cada punto fronterizo a la agrupación de su correspondiente punto central.

[ ]:
```
from sklearn.datasets import make_moons

X, y = make_moons(n_samples=200, noise=0.05, random_state=0)
plt.scatter(X[:, 0], X[:, 1])
plt.tight_layout()
```

```
#plt.savefig('images/11_14.png', dpi=300)
plt.show()
```

[ ]:
```
from sklearn.cluster import KMeans

f, (ax1, ax2) = plt.subplots(1, 2, figsize=(8, 3))

km = KMeans(n_clusters=2, random_state=0)
y_km = km.fit_predict(X)

ax1.scatter(X[y_km == 0, 0], X[y_km == 0, 1],
            edgecolor='black',
            c='lightblue', marker='o', s=40, label='cluster 1')
ax1.scatter(X[y_km == 1, 0], X[y_km == 1, 1],
            edgecolor='black',
            c='red', marker='s', s=40, label='cluster 2')
ax1.set_title('K-means clustering')

ac =␣
 ↪AgglomerativeClustering(n_clusters=2,affinity='euclidean',linkage='complete')
y_ac = ac.fit_predict(X)

ax2.scatter(X[y_ac == 0, 0], X[y_ac == 0, 1], c='lightblue',
            edgecolor='black',
            marker='o', s=40, label='cluster 1')
ax2.scatter(X[y_ac == 1, 0], X[y_ac == 1, 1], c='red',
            edgecolor='black',
            marker='s', s=40, label='cluster 2')
ax2.set_title('Agglomerative clustering')

plt.legend()
plt.tight_layout()
# plt.savefig('images/11_15.png', dpi=300)
plt.show()
```

[ ]:
```
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.1, min_samples=3, metric='euclidean')
y_db = db.fit_predict(X)

np.unique(y_db)
```

[ ]:
```
db = DBSCAN(eps=0.1, min_samples=5, metric='euclidean')
y_db = db.fit_predict(X)

np.unique(y_db)
```

[ ]:
```
db = DBSCAN(eps=0.2, min_samples=5, metric='euclidean')
y_db = db.fit_predict(X)

np.unique(y_db)
```

[ ]:
```
plt.scatter(X[y_db == 0, 0], X[y_db == 0, 1],
            c='lightblue', marker='o', s=40,
            edgecolor='black',
            label='cluster 1')
plt.scatter(X[y_db == 1, 0], X[y_db == 1, 1],
            c='red', marker='s', s=40,
            edgecolor='black',
            label='cluster 2')
plt.legend()
plt.tight_layout()
#plt.savefig('images/11_16.png', dpi=300)
plt.show()
```

1.3 Actividad Final

Correr algoritmos de clustering para los siguientes datos:

[ ]:
```
X1, y1 = make_moons(n_samples=200, noise=0.12, random_state=0)
plt.scatter(X1[:, 0], X1[:, 1])
plt.tight_layout()
#plt.savefig('images/11_14.png', dpi=300)
plt.show()
```

[ ]:
```

```