Aprendizaje No Supervisado

Clustering

Clase 07

Breve Introducción a Machine Learning

Dr. Ramón Caraballo
SECIU Red Académica Uruguaya
UDELAR

Objetivo

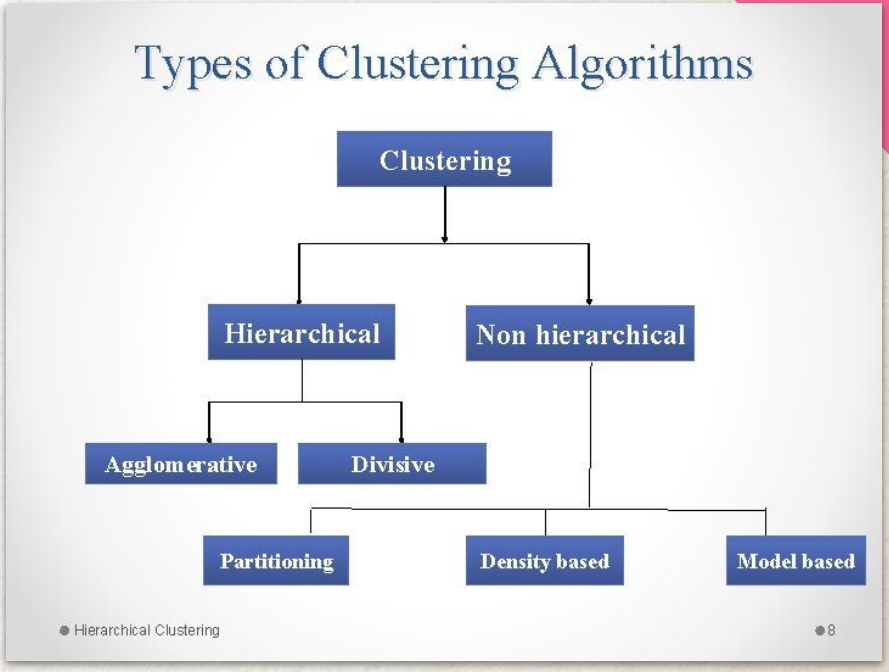
Veremos:

- **Clustering Particional: k-Means**
- **Clustering Aglomerativo o Jerárquico**
- **Density Based Scan (DBScan)**
- **Métricas de evaluación de la performance**

Aprendizaje No Supervisado

Generalidades

- Trabaja con datos sin etiquetar.
- Los algoritmos de aprendizaje no supervisado se encargan de encontrar patrones, estructuras y relaciones dentro de los propios datos, sin ninguna guía explícita.
- El objetivo es comprender la organización inherente de los datos.



Aplicaciones

Segmentación de clientes: Agrupación de clientes según su comportamiento de compra, datos demográficos o intereses para marketing dirigido.

Sistemas de recomendación: Sugerencias de productos, películas o artículos a los usuarios según su comportamiento previo y el de usuarios similares.

Detección de anomalías: Identificación de transacciones fraudulentas, intrusiones en la red o patrones inusuales en los datos.

Agrupación de imágenes y documentos: Organización de grandes colecciones de imágenes o documentos en categorías significativas.

Bioinformática: Análisis de datos de expresión génica para identificar patrones y relaciones.

Procesamiento del lenguaje natural (PLN): Modelado de temas, resumen de documentos y agrupación semántica de datos de texto.
Etc. …

Ventajas y Desventajas

Ventajas:

- **Maneja datos sin etiquetar:**
  Útil cuando los datos etiquetados son escasos o costosos de obtener.
- **Descubre patrones ocultos:**
  Puede identificar relaciones y estructuras previamente desconocidas en los datos.
- **Análisis exploratorio de datos:**
  Ayuda a comprender la estructura subyacente de un conjunto de datos.
- **Escalabilidad:**
  Muchos algoritmos no supervisados pueden manejar grandes conjuntos de datos de forma eficaz.

Desventajas:

- **Dificultad de evaluación:**
  Dado que no existen etiquetas precisas, evaluar el rendimiento de los algoritmos de aprendizaje no supervisado puede ser complicado.
- **Los resultados pueden ser subjetivos:**
  La interpretación de los patrones descubiertos puede depender del usuario.
- **Menos preciso que el aprendizaje supervisado:**
  Sin una guía explícita, los modelos podrían no alcanzar el mismo nivel de precisión que el aprendizaje supervisado para tareas de predicción específicas.

Aprendizaje supervisado vs. no supervisado

- El aprendizaje supervisado encuentra patrones para una tarea de predicción
- P. ej., clasificar tumores como benignos o cancerosos (etiquetas)
- El aprendizaje no supervisado encuentra patrones en los datos
- ... pero sin una tarea de predicción específica en mente

Classification	Clustering
Model predicts a probable classification for a given input	Model maps a given input into one of the data clusters
Uses supervised learning	Uses unsupervised learning
Labeled data required for training	Does not require labeled data
Number of categories is known	Number of groups is unknown
Two-step process to train and predict	Single-step process

Algoritmos Comunes

Clustering (Agrupamiento): Agrupa los datos en conjuntos (clusters) basándose en su similitud. El objetivo es que los puntos de datos dentro de un mismo cluster sean más similares entre sí que con los puntos de datos de otros clusters.

Reducción de dimensionalidad: Reduce el número de variables (características) en un conjunto de datos mientras se preserva la mayor cantidad de información importante posible. Esto facilita la visualización y el análisis de datos de alta dimensión.

Minería de reglas de asociación: Identifica relaciones o dependencias entre diferentes elementos en un conjunto de datos. Un ejemplo clásico es el análisis de la cesta de la compra, donde se busca qué productos se compran juntos con frecuencia.

Detección de anomalías: Identifica puntos de datos que son significativamente diferentes del resto de los datos. Estos puntos "atípicos" pueden indicar errores, fraudes o eventos inusuales.

Basado en Particiones:
Relativamente Eficientes
k-means, k-median, fuzzy means

En base a Jerarquías:
Aglomeración-División
Clustering Jerárquico, Dendrogramas

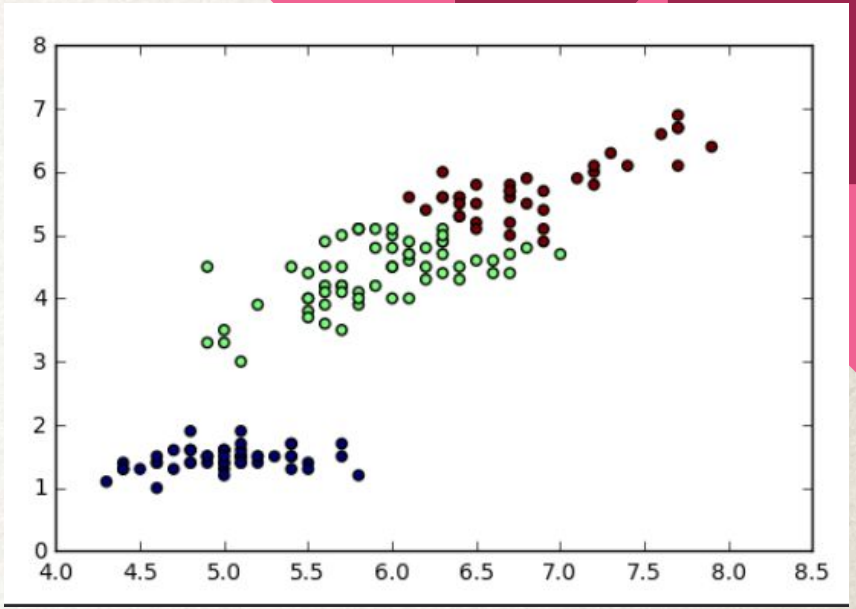
Por Densidad de elementos:
Clusters de forma arbitraria
DBScan

k-Means

- Iris Dataset
- Encuentra grupos de muestras.
- Se debe especificar el número de grupos.
- Implementado en sklearn ("scikit-learn").

k-means clustering with scikit-learn

```
In [1]: print(samples)
[[ 5.   3.3  1.4  0.2]
 [ 5.   3.5  1.3  0.3]
 [ 4.9  2.4  3.3  1. ]
 [ 6.3  2.8  5.1  1.5]
 ...
 [ 7.2  3.2  6.   1.8]]

In [2]: from sklearn.cluster import KMeans

In [3]: model = KMeans(n_clusters=3)

In [4]: model.fit(samples)
Out[4]: KMeans(algorithm='auto', ...)

In [5]: labels = model.predict(samples)

In [6]: print(labels)
[0 0 1 1 0 1 2 1 0 1 ...]
```



- Diagrama de dispersión de la longitud del sépalo frente a la longitud del pétalo.
- Cada punto representa una muestra de iris.
- Colorear los puntos según las etiquetas de los grupos.
- PyPlot (matplotlib.pyplot)

Datos Nuevos

Etiquetas de clusters para nuevas muestras

- Se pueden asignar nuevas muestras a clusters existentes
- k-means recuerda la media de cada clusters (los "centroides")
- Encuentra el centroide más cercano a cada nueva muestra

```
In [7]: print(new_samples)
[[ 5.7  4.4  1.5  0.4]
 [ 6.5  3.   5.5  1.8]
 [ 5.8  2.7  5.1  1.9]]

In [8]: new_labels = model.predict(new_samples)

In [9]: print(new_labels)
[0 2 1]
```

```
In [1]: import matplotlib.pyplot as plt

In [2]: xs = samples[:,0]

In [3]: ys = samples[:,2]

In [4]: plt.scatter(xs, ys, c=labels)

In [5]: plt.show()
```

Evaluación del Clustering

Medidas de Similitud y Disparidad

- Distancia de similitud ----> p. ej Distancia Euclídea $$D(x_1, x_2) = \left(\sum_{i=1}^{m} (x_{1i} - x_{2i})^2 \right)^{\frac{1}{2}}$$
- K-means trata de minimizar las distancias intra-cluster mientras que maximiza las distancias extra-cluster.
- Debemos **normalizar (estandarizar)**, todas las features para tener una medida exacta de disparidad.

Inercia: La distancia de cada pto del cluster a su centroide al cuadrado.

$$SSE_k = \sum_{i=1}^{n} (x_{ik} - c_k)^2$$

Exactitud de k-Means

Evaluación de un clustering
- Permite comprobar la correspondencia con, por ejemplo, especies de iris.
- … ¿pero qué ocurre si no hay especies con las que comparar?
- Medición de la calidad de una agrupación
- Informa sobre la elección de cuántos grupos buscar.

Enfoque externo: Comparar los clusters con datos comprobados si es posible

Enfoque interno: Promediar las distancias entre puntos dentro del mismo cluster

No hay garantía que K-means alcance el óptimo global, (podría llegar a uno local), el resultado dependerá de cómo se definen los clusters iniciales.

Debemos correr el proceso varias veces partiendo con diferentes centroides cada vez.

Tabulación cruzada con Pandas

Iris: grupos vs. especies
- k-means encontró 3 grupos entre las muestras de iris
- ¿Los grupos corresponden a las especies?

species labels	setosa	versicolor	virginica
0	0	2	36
1	50	0	0
2	0	48	14

- La comparación entre grupos y especies es una "**tabulación cruzada**".
- Dar las especies de cada muestra como una lista de especies.

```
In [1]: print(species)
['setosa', 'setosa', 'versicolor', 'virginica', ... ]
```

Alineando Etiquetas y Especies

```
In [2]: import pandas as pd

In [3]: df = pd.DataFrame({'labels': labels, 'species': species})

In [4]: print(df)
     labels     species
0         1      setosa
1         1      setosa
2         2  versicolor
3         2   virginica
4         1      setosa
...
```

Tabulación Cruzada de Etiquetas y Especies

```
In [5]: ct = pd.crosstab(df['labels'], df['species'])

In [6]: print(ct)
species  setosa  versicolor  virginica
labels
0             0           2         36
1            50           0          0
2             0          48         14
```

¿Cómo evaluar un agrupamiento si no hay información sobre las especies?

Medición de la calidad del Clustering

- Utilizando únicamente muestras y sus etiquetas de grupo
- Un buen agrupamiento tiene grupos compactos
- ... y muestras en cada grupo agrupadas

La **inercia** mide la calidad del clustering.

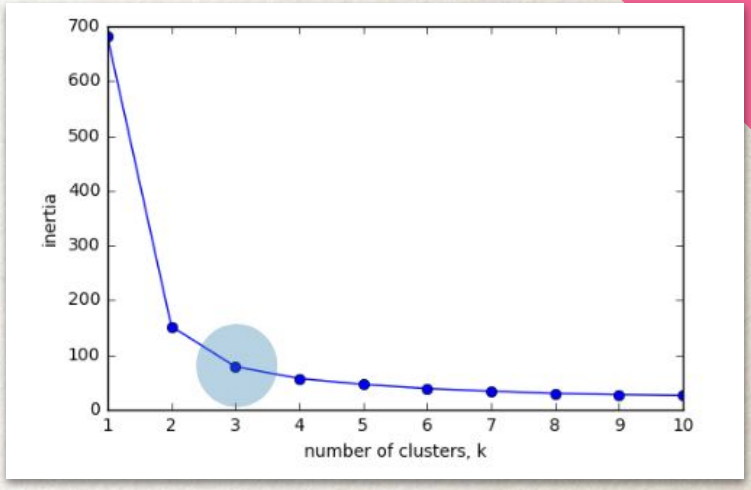
- Mide la dispersión de los clústeres (cuanto menor, mejor).
- Distancia de cada muestra al centroide de su cluster.
- Después de **.fit()**, está disponible el método **.inertia_**
- k-means intenta minimizar la inercia al seleccionar clústeres.

```
In [1]: from sklearn.cluster import KMeans

In [2]: model = KMeans(n_clusters=3)

In [3]: model.fit(samples)

In [4]: print(model.inertia_)
78.9408414261
```

Método Gráfico

- Agrupe el dataset Iris en diferentes números de clústeres
- Más clústeres significan menor inercia
- ¿Cuál es el mejor número de clústeres?

- Un buen agrupamiento tiene clústeres compactos (por lo tanto, baja inercia).
- ... ¡pero no demasiados!
- Elija un "codo" en el gráfico de inercia.
- Donde la inercia comienza a disminuir más lentamente.
- Por ejemplo, para el conjunto de datos de iris, 3 es una buena opción.



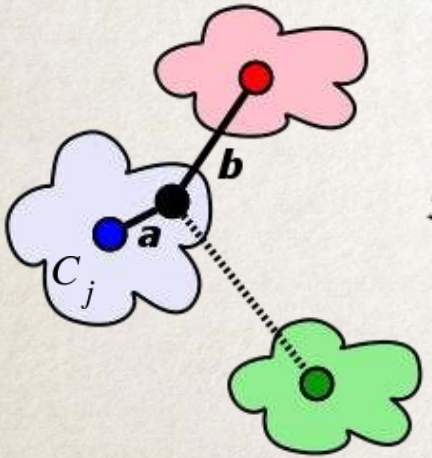
Silhouette Score

Óptimo valor para k

C_i : población del cluster C_j.

b_i : distancia intercluster, distancia promedio al cluster más próximo del i-ésimo punto, exceptuando su propio cluster C_j. Mide cuán bien asignado fue el i-ésimo punto al cluster C_j.

$$b_i = min_{i \neq j} \frac{1}{|C_j|} \sum_{i \in C_j} d(i,j)$$



$$SSI_j = \frac{b_i - a_i}{max(a_i, b_i)}$$

a_i : distancia intracluster, distancia promedio del i-ésimo punto a todos los demás del mismo cluster. Mide cuán bien asignado fue el i-ésimo punto al cluster C_j.

$$a_i = \frac{1}{|C_j| - 1} \sum_{C_j,\, i \neq j} d(i,j)$$

Silhouette Score

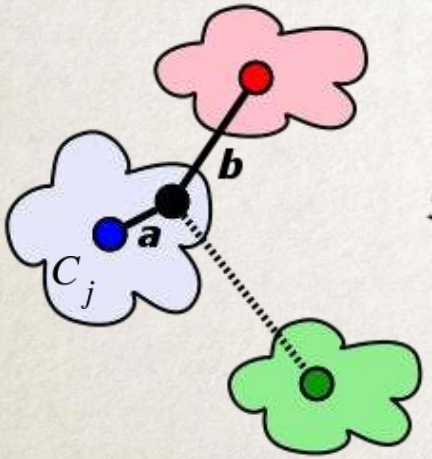
Óptimo valor para k

En gral: $-1 \leq SSI_i \leq \leq 1$

$SSI_i \rightarrow 1$ El punto está muy lejos de los clusters vecinos

$SSI_i \rightarrow 0$ El punto está cerca del borde de decisión entre clusters

$SSI_i \rightarrow 1$ El punto fue asignado al cluster equivocado



$$SSI_i = \frac{b_i - a_i}{max(a_i, b_i)}$$

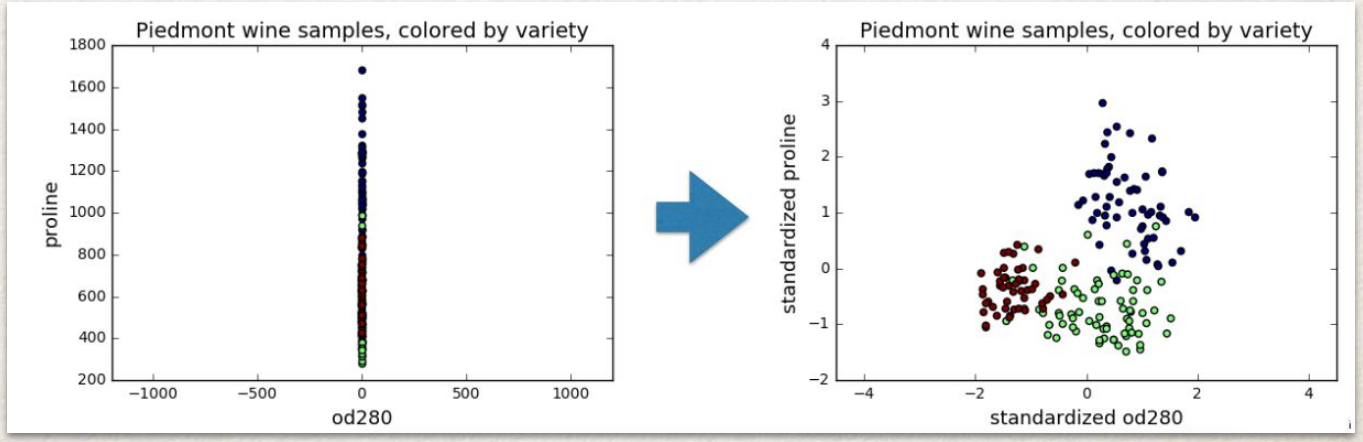
El score de Silhouette general es el promedio de los escores para todos los puntos del dataset

$$SSI = \frac{1}{C_j} \sum_i SSI_i$$

Rol del Estandarizado

StandardScaler

- **En kmeans:** varianza de la característica = influencia de la característica
- **StandardScaler** transforma cada característica para que tenga media 0 y varianza 1
- Se dice que las características están "**estandarizadas**"



Sintaxis

```
In [1]: from sklearn.preprocessing import StandardScaler

In [2]: scaler = StandardScaler()

In [3]: scaler.fit(samples)
Out[3]: StandardScaler(copy=True, with_mean=True, with_std=True)

In [4]: samples_scaled = scaler.transform(samples)
```

Métodos similares

- **StandardScaler** y **KMeans** tienen métodos similares
- Use **fit() / transform()** con **StandardScaler**
- Use **fit() / predict()** con **KMeans**

Proceso de dos pasos

Ejemplo: Clasificación de variedades de vinos

Piedmont wines dataset

- 178 samples from 3 distinct varieties of red wine: *Barolo*, *Grignolino* and *Barbera*
- Features measure chemical composition e.g. alcohol content
- ... also visual properties like “color intensity”

https://archive.ics.uci.edu/ml/datasets/Wine

StandardScaler, luego KMeans

- Se necesitan dos pasos: StandardScaler, luego KMeans
- Usar la canalización de Sklearn para combinar varios pasos
- Los datos fluyen de un paso al siguiente

Pipelines combine multiple steps

```
In [1]: from sklearn.preprocessing import StandardScaler

In [2]: from sklearn.cluster import KMeans

In [3]: scaler = StandardScaler()

In [4]: kmeans = KMeans(n_clusters=3)

In [5]: from sklearn.pipeline import make_pipeline

In [6]: pipeline = make_pipeline(scaler, kmeans)

In [7]: pipeline.fit(samples)
Out[7]: Pipeline(steps=...)

In [8]: labels = pipeline.predict(samples)
```

Feature standardization improves clustering

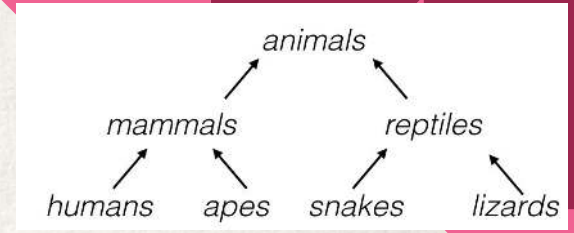
```
In [9]: df = pd.DataFrame({'labels': labels, 'varieties': varieties})

In [10]: ct = pd.crosstab(df['labels'], df['varieties'])

In [11]: print(ct)
varieties  Barbera  Barolo  Grignolino
labels
0                0      59           3
1               48       0           3
2                0       0          65
```

Without **feature standardization was very bad:**

```
varieties  Barbera  Barolo  Grignolino
labels
0               29      13          20
1                0      46           1
2               19       0          50
```

Clustering Jerárquico

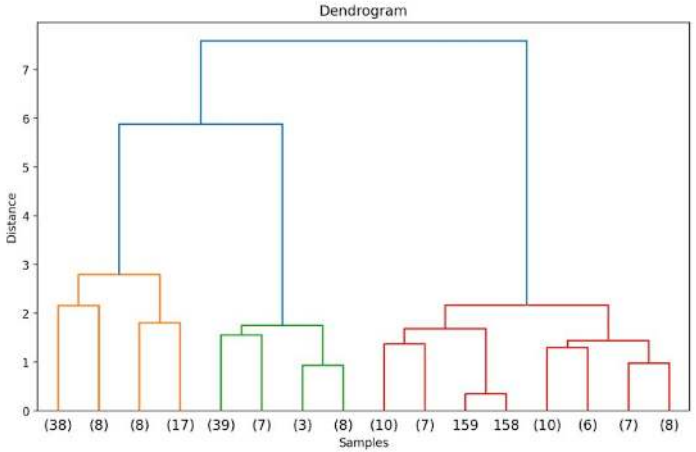
Dendrogramas

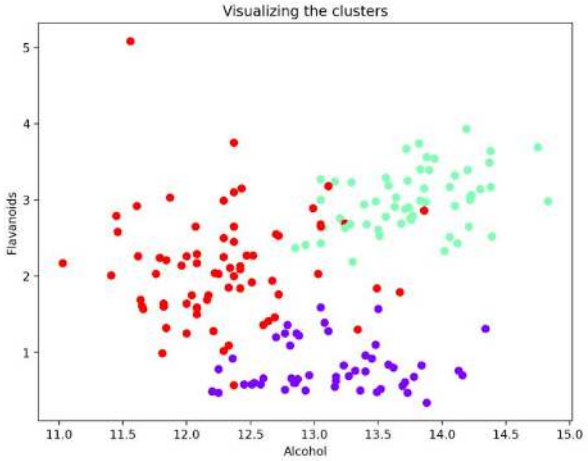


Las visualizaciones transmiten información

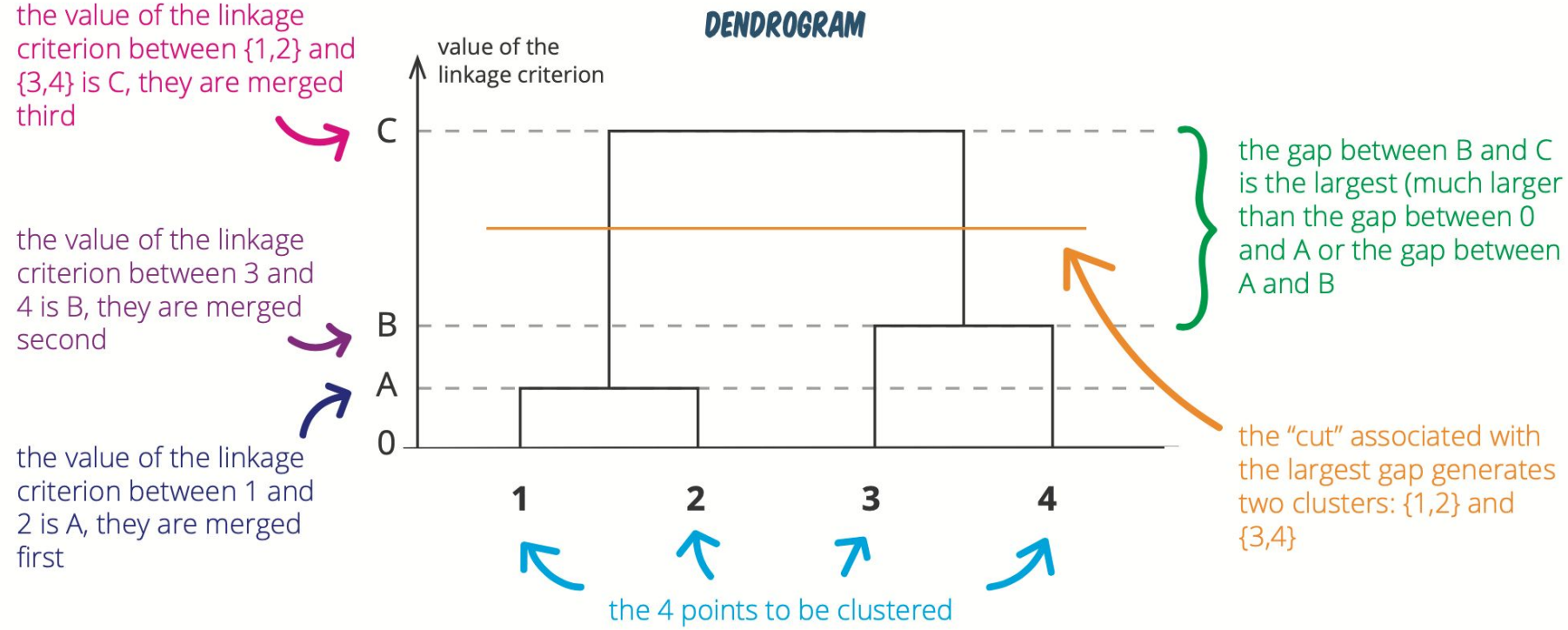
Una jerarquía de grupos

- Los grupos de seres vivos pueden formar una jerarquía
- Los grupos están contenidos unos dentro de otros

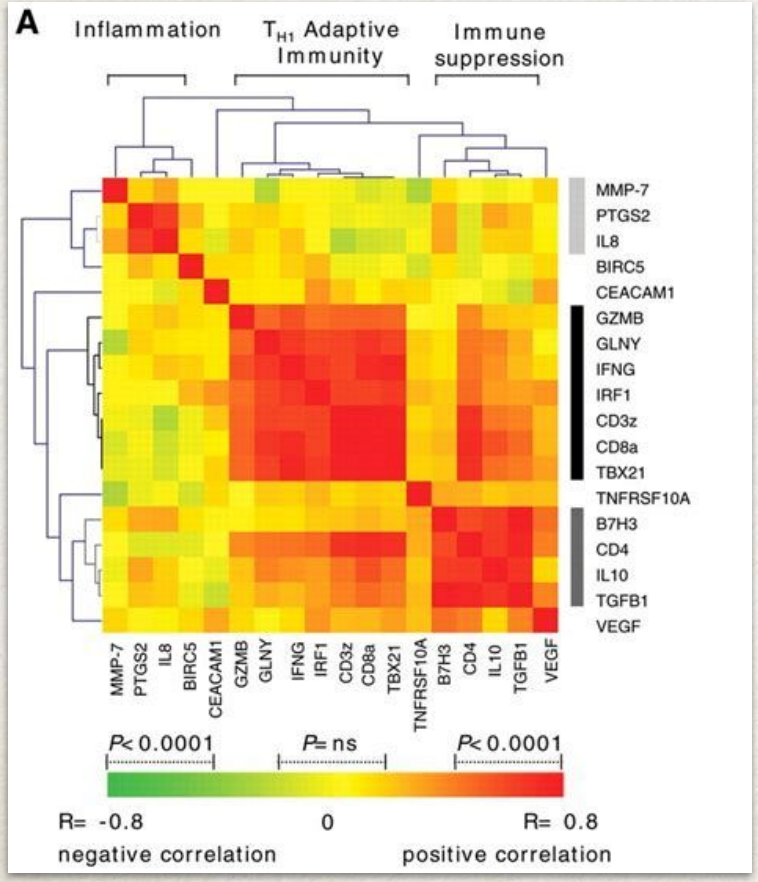




Entendiendo un Dendrograma



Linkage: Gdo. de Asociación



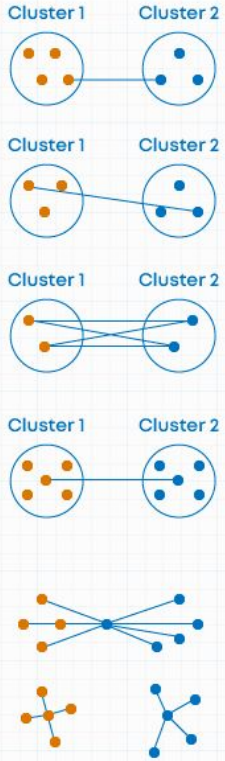
- **Single Linkage**
  $D(c_1,c_2) = \min D(x_1,x_2)$
  Minimum distance or distance between closest elements in clusters

- **Complete Linkage**
  $D(c_1,c_2) = \max D(x_1,x_2)$
  Maximum distance between elements in clusters

- **Average Linkage**
  $D(c_1,c_2) = \frac{1}{|c_1|}\frac{1}{|c_2|}\Sigma\Sigma D(x_1,x_2)$
  Average of the distances of all pairs

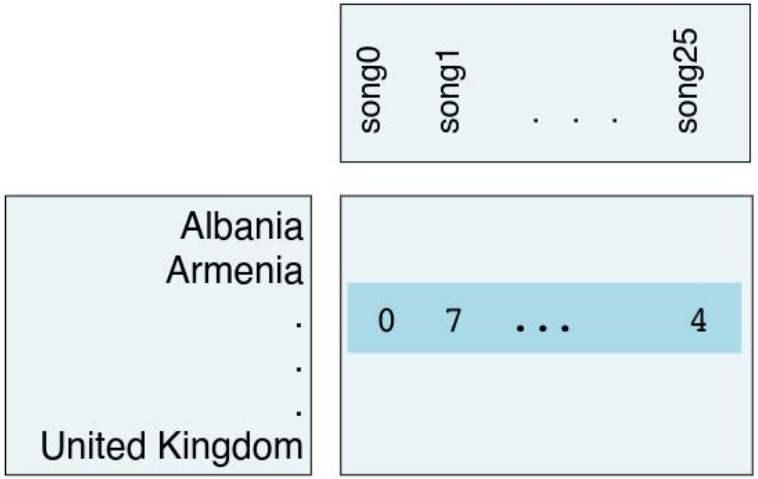
- **Centroid Method**
  Combining clusters with minimum distance between the centroids of the two clusters

- **Ward's Method**
  - Combining clusters where increase in within cluster variance is to the smallest degree.
  - Objective is to minimize the total within cluster vairance



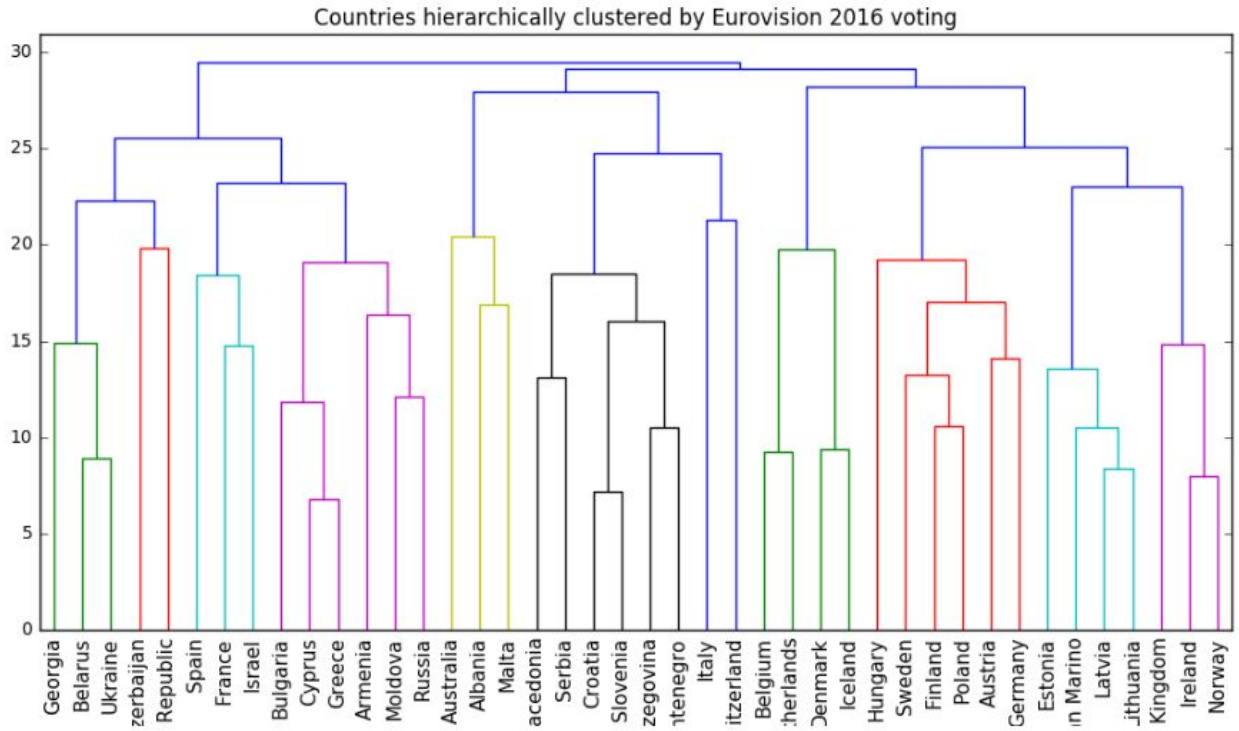
Eurovision scoring dataset

- Countries gave scores to songs performed at the Eurovision 2016
- 2D array of scores
- Rows are countries, columns are songs



http://www.eurovision.tv/page/results

Hierarchical clustering of voting countries

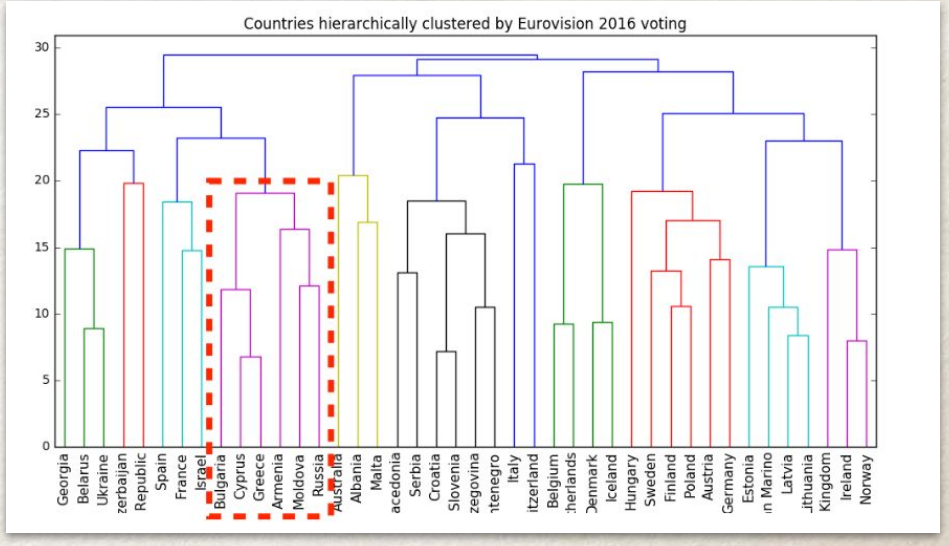


Clustering Jerárquico

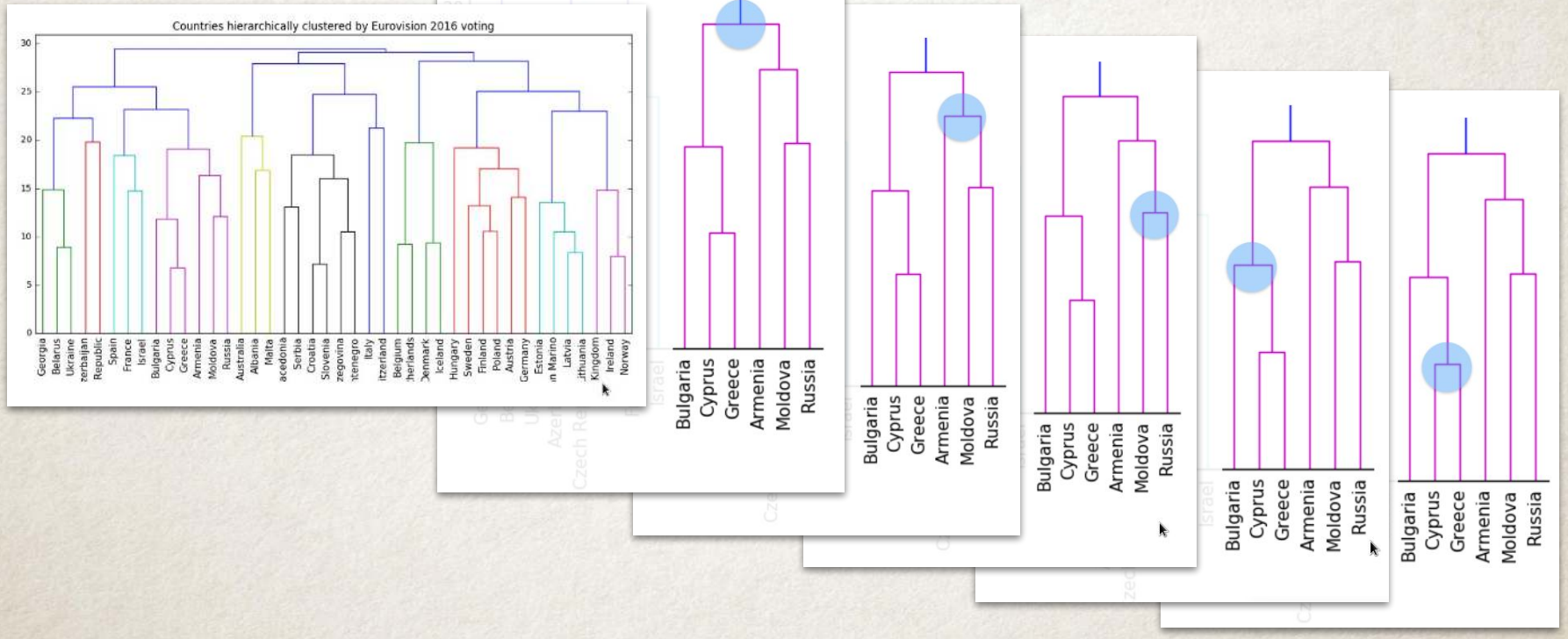
- Cada país comienza en un grupo separado
- En cada paso, se fusionan los dos grupos más cercanos
- Continúe hasta que todos los países estén en un solo grupo
- Esta es la agrupación jerárquica "aglomerativa"

El dendrograma de un cluster jerárquico

- Se Lee de abajo a arriba
- Las líneas verticales representan clústeres



Dendrograma paso a paso



Hierarchical clustering with SciPy

- Given **samples** (the array of scores), and **country_names**

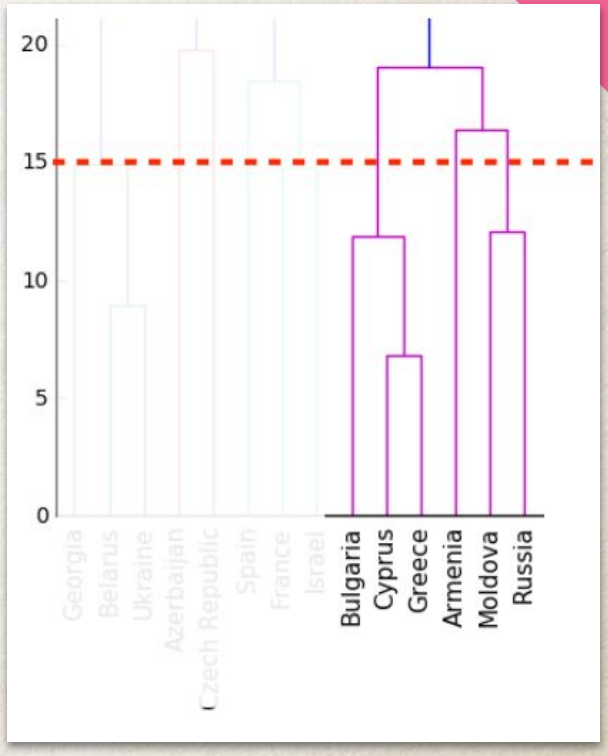
```
In [1]: import matplotlib.pyplot as plt

In [2]: from scipy.cluster.hierarchy import linkage, dendrogram

In [3]: mergings = linkage(samples, method='complete')

In [4]: dendrogram(mergings,
   ...:            labels=country_names,
   ...:            leaf_rotation=90,
   ...:            leaf_font_size=6)

In [5]: plt.show()
```

Etiquetado en el Clustering Jerárquico

- ¡No solo una herramienta de visualización!
- Se pueden recuperar las etiquetas de clúster en cualquier etapa intermedia.
- Para su uso, por ejemplo, en tabulaciones cruzadas.

Agrupamientos intermedios y altura en el dendrograma

- Por ejemplo, a la altura 15: Bulgaria, Chipre y Grecia forman un grupo
- Rusia y Moldavia forman otro
- Armenia forma un grupo independiente

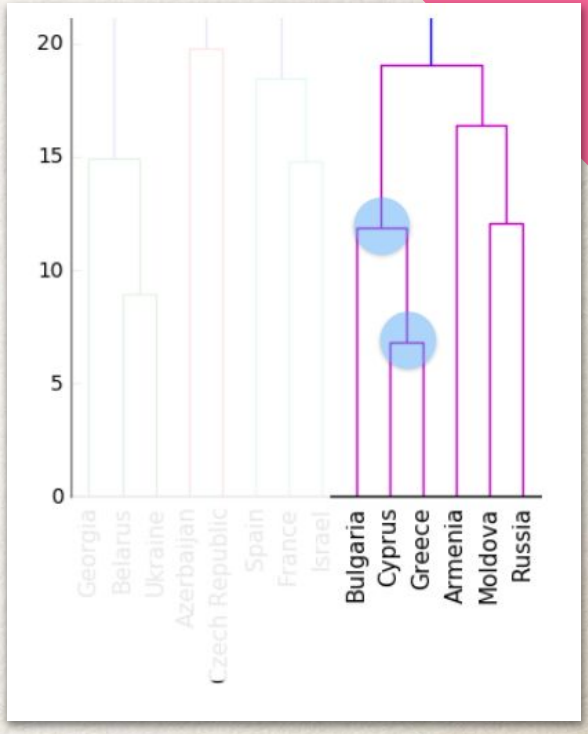


Los dendrogramas muestran las distancias entre los grupos.

- Altura en el dendrograma = distancia entre los grupos fusionados.

- Por ejemplo, los grupos que solo incluían Chipre y Grecia tenían una distancia aproximada de 6.

- Este nuevo grupo se encuentra a una distancia aproximada de 12 del grupo que solo incluía a Bulgaria.

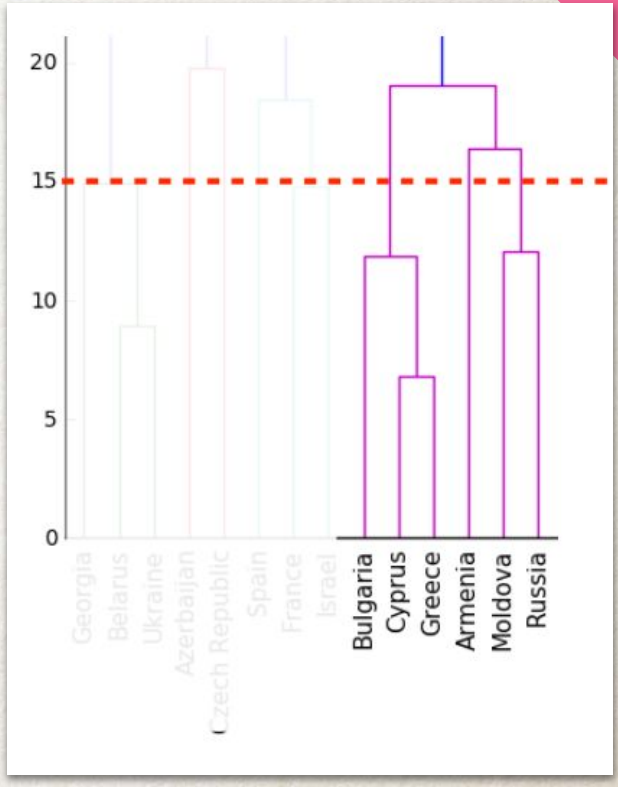


Clústeres intermedios y altura en el dendrograma

- La altura en el dendrograma especifica la distancia máxima entre las agrupaciones que se fusionan.
- No fusionar agrupaciones con una distancia mayor (p. ej., 15).

Distancia entre clústeres

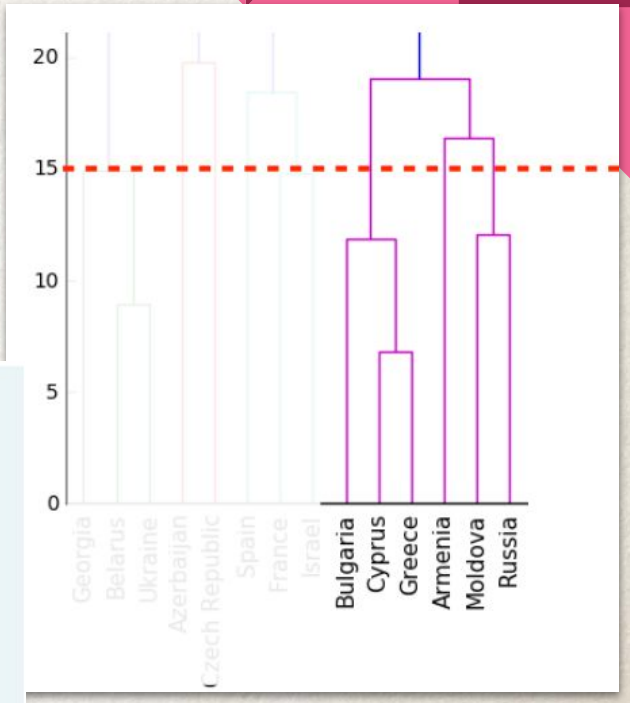
- Definida por un "método de linkage"
- Especificada mediante un parámetro de método, p. ej., linkage(samples, method="complete")
- En un linkage "completo": la distancia entre clústeres es la distancia máxima entre sus muestras
- ¡Diferente método de linkage, diferente agrupamiento jerárquico!



Extracción de etiquetas de clúster

- Usar el método **fcluster**
- Devuelve una matriz NumPy de etiquetas de clúster

```
In [1]: from scipy.cluster.hierarchy import linkage

In [2]: mergings = linkage(samples, method='complete')

In [3]: from scipy.cluster.hierarchy import fcluster

In [4]: labels = fcluster(mergings, 15, criterion='distance')

In [5]: print(labels)
[ 9  8 11 20  2  1 17 14 ... ]
```

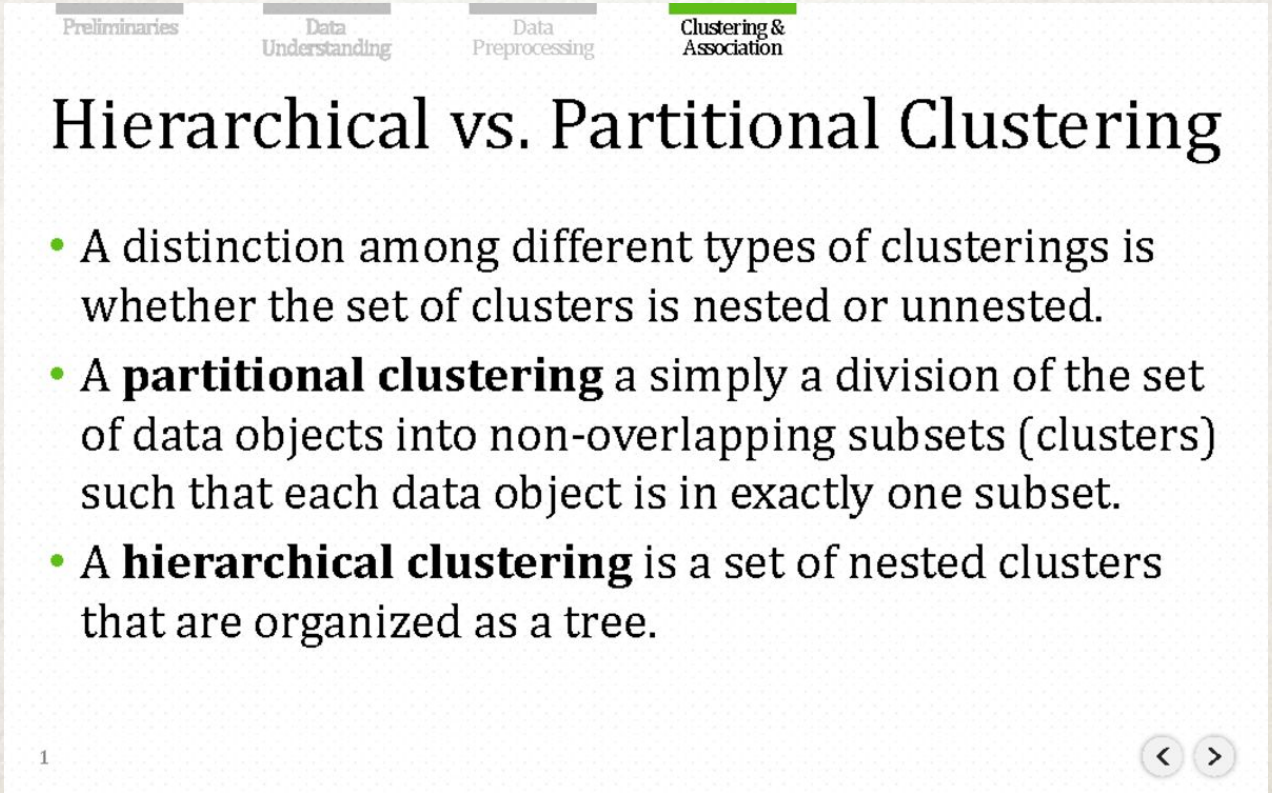


Alineación de las etiquetas de clúster con los nombres de los países

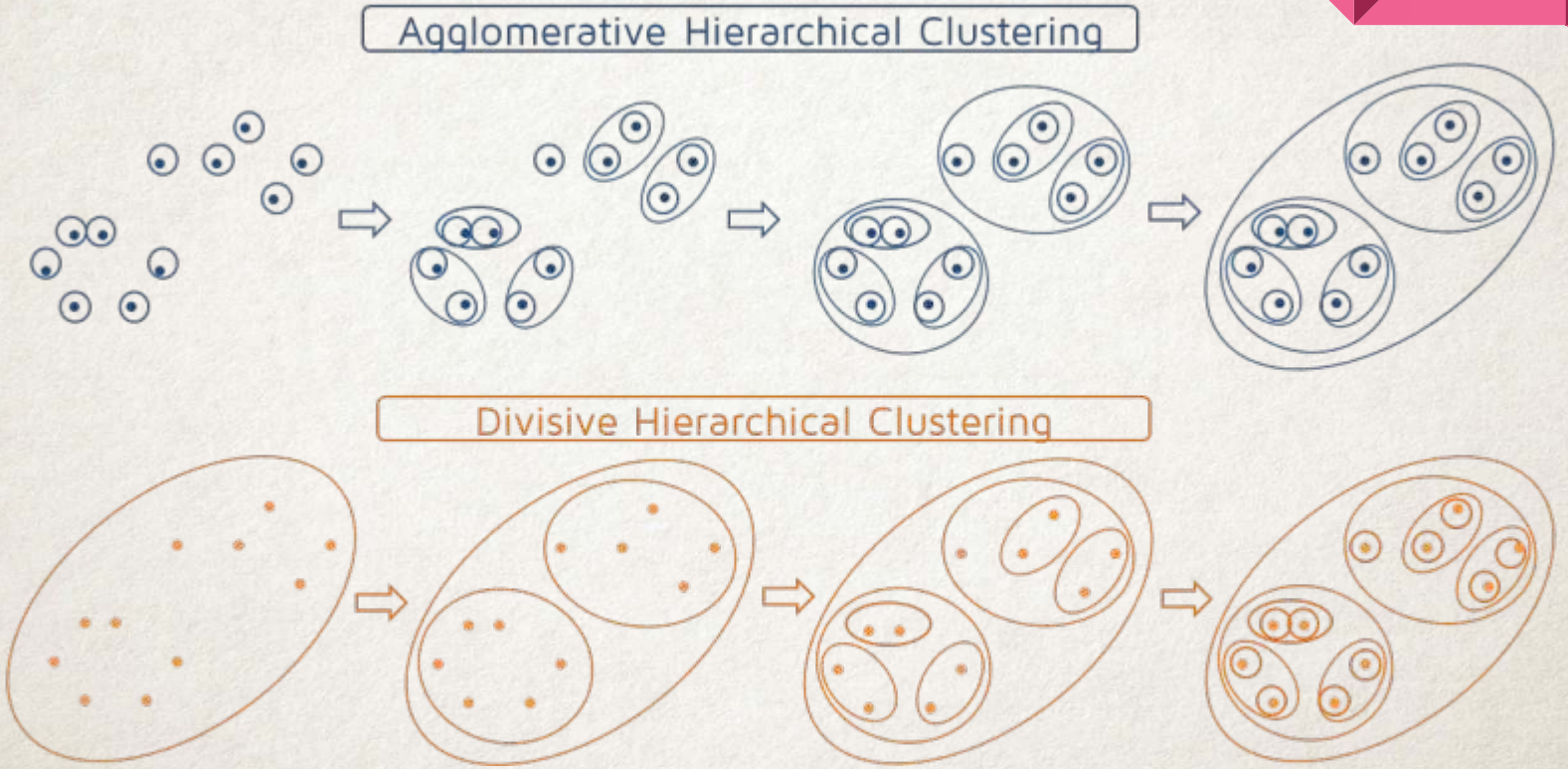
- Dada una lista de strings **country_names**:

```
In [1]: import pandas as pd

In [2]: pairs = pd.DataFrame({'labels': labels,
   ...:                       'countries': country_names})

In [3]: print(pairs.sort_values('labels'))
                countries  labels
5                 Belarus       1
40                Ukraine       1
17                Georgia       1
...
36                  Spain       5
8                Bulgaria       6
19                 Greece       6
10                 Cyprus       6
28                Moldova       7
...
```

Para finalizar

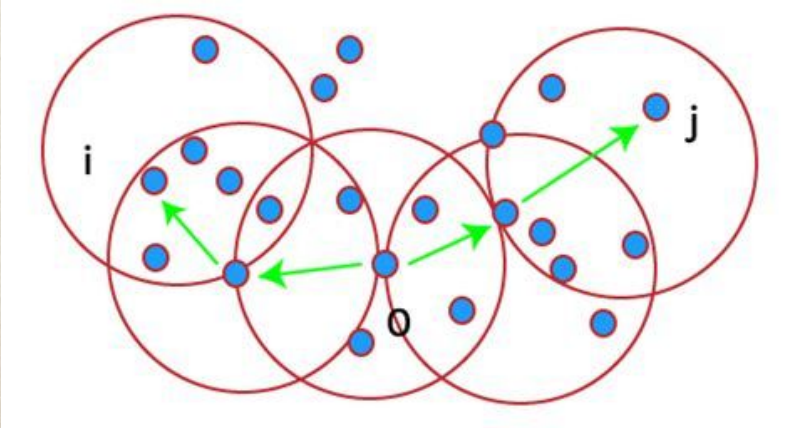


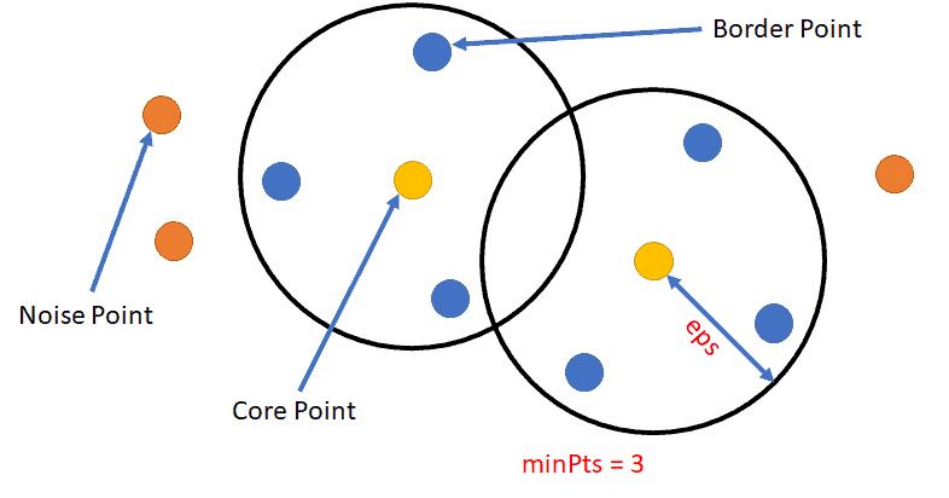
Dos Aproximaciones Diferentes



Density Based Scan: DBScan

El Agrupamiento Espacial Basado en la Densidad (DBSCAN)
Algoritmo de clustering basado en la densidad. A diferencia de los algoritmos basados en centroides como K-Means, DBSCAN agrupa puntos de datos con una alta densidad de datos (muchos vecinos cercanos), marcando como valores atípicos los puntos que se encuentran solos en regiones de baja densidad.





Conceptos Básicos

Épsilon (eps o ε): Define el radio de la vecindad alrededor de un punto de datos. Si la distancia entre dos puntos es menor o igual a ε, se consideran vecinos.

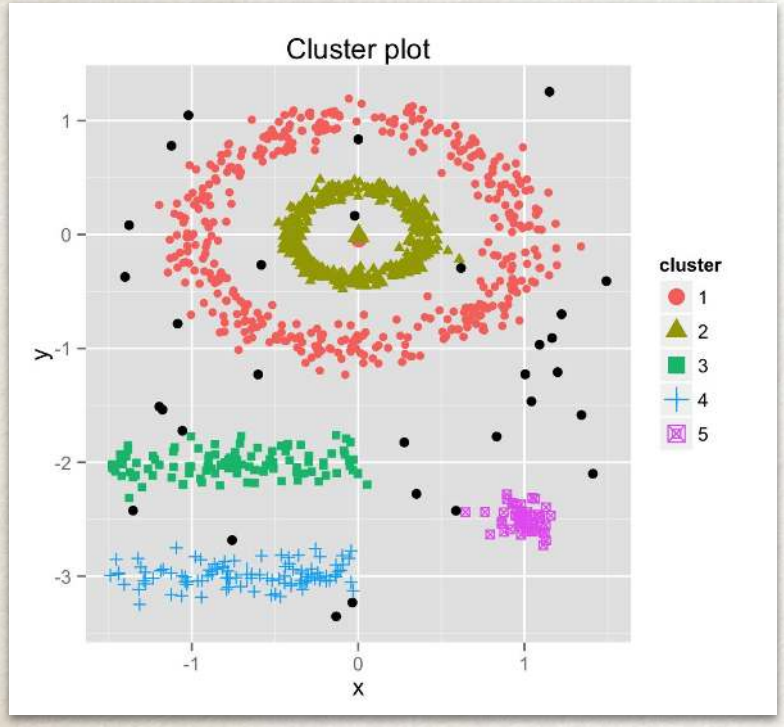
Puntos mínimos (minPts): Especifica el nro mínimo de vecinos que un punto de datos debe tener dentro de su vecindad ε para ser considerado un punto central. Este número incluye el propio punto.

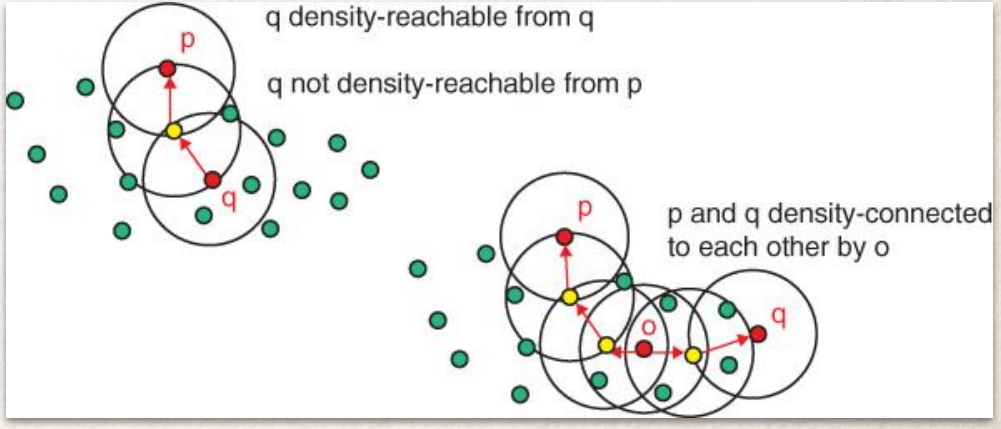
Punto central: Un punto de datos es un punto central si hay al menos minPts (incluido el propio punto) dentro de su vecindad ε. Estos puntos se encuentran en el interior de un clúster.

Punto límite: Un punto de datos es un punto límite si no es un punto central, pero se encuentra dentro de la vecindad ε de un punto central. Se encuentra en el borde de un clúster.

Punto de ruido (valor atípico): Un punto de datos es un punto de ruido si no es un punto central ni un punto límite. No tiene suficientes vecinos dentro de ε y no está dentro del vecindario ε de ningún punto central.

Ejemplos


Cluster plot
cluster
1
2
3
4
5
x
y
1
0
-1
-2
-3
-1
0
1


q density-reachable from q
p
q
q not density-reachable from p
p
p and q density-connected
to each other by o
o
q

Algoritmo

Inicialización: Comienza con un punto arbitrario no visitado en el conjunto de datos.

Consulta de vecindad: Se buscan todos los puntos dentro de la vecindad ε del punto actual.

Comprobación de punto central: Si el número de puntos en la vecindad ε (incluido el punto actual) es mayor o igual a **minPts**, el punto actual se etiqueta como punto central. Se forma un nuevo clúster y todos los puntos en su vecindad ε se añaden a este. El proceso continúa recursivamente para todos estos puntos recién añadidos y sus vecindades ε.
Si el número de puntos en la vecindad ε es menor que **minPts**, el punto actual se etiqueta temporalmente como ruido. Este punto podría encontrarse posteriormente dentro de la vecindad ε de un punto central y convertirse en un punto límite.

Iteración: Repetir los pasos 1 a 3 para todos los puntos no visitados del conjunto de datos.

Asignación de puntos límite: Una vez procesados todos los puntos centrales, visitar los puntos de ruido restantes. Si un punto de ruido se encuentra en la vecindad ε de un punto central, se asigna al grupo de dicho punto central, convirtiéndose en un punto límite.

Puntos de ruido restantes: Los puntos que no son puntos centrales ni accesibles desde ningún punto central permanecen etiquetados como ruido (valores atípicos).

Ventajas

Ventajas de DBSCAN:

- **Descubre clústeres de formas arbitrarias:** A diferencia de K-Means, DBSCAN puede encontrar clústeres no esféricos.
- **Maneja el ruido eficazmente:** Puede identificar y aislar valores atípicos como puntos de ruido.
- **No requiere especificar el número de clústeres de antemano:** El algoritmo determina automáticamente el número de clústeres en función de la densidad de datos.
- **Robusto al orden de los puntos de datos:** Los clústeres resultantes son generalmente consistentes, independientemente del orden en que se procesen los puntos de datos.

Desventajas

Sensible a la selección de parámetros: La elección de ε y **minPts** puede afectar significativamente los resultados de la agrupación. Encontrar valores óptimos puede ser un desafío y, a menudo, requiere conocimiento del dominio o experimentación.

Dificultades con densidades variables: DBSCAN puede tener dificultades para identificar clústeres con densidades significativamente diferentes. Un único conjunto de eps y minPts podría no funcionar correctamente en todas las variaciones de densidad.

Computacionalmente costoso para conjuntos de datos muy grandes: Las consultas de vecindad pueden consumir mucho tiempo para datos de alta dimensión o conjuntos de datos grandes, aunque las estructuras de indexación espacial pueden ayudar a mitigar esto.

Los puntos límite se pueden asignar de forma algo arbitraria: Si un punto límite se encuentra dentro de la vecindad ε de los puntos centrales de varios clústeres, su asignación a un clúster específico podría depender del orden en que se visitaron los puntos centrales.

Parámetros para Ajustar:

- **ε (Épsilon):** Elegir un ε adecuado es crucial.
  - Si es demasiado pequeño, muchos puntos se considerarán ruido.
  - Si es demasiado grande, los clústeres densos podrían fusionarse, o los clústeres dispersos podrían incluir incorrectamente puntos de ruido. Un enfoque común para estimar eps es mediante el gráfico de k-distancia.
- **minPts (Puntos Mínimos):** Este parámetro también influye en la robustez de la agrupación.
  - ***Un minPts mayor*** hace que la definición de un punto central sea más estricta, lo que podría resultar en menos clústeres y más puntos de ruido.
  - ***Un minPts menor*** puede hacer que el algoritmo sea más sensible al ruido y podría fusionar clústeres que deberían estar separados. Una regla general es establecer minPts al menos al doble del número de dimensiones del conjunto de datos.

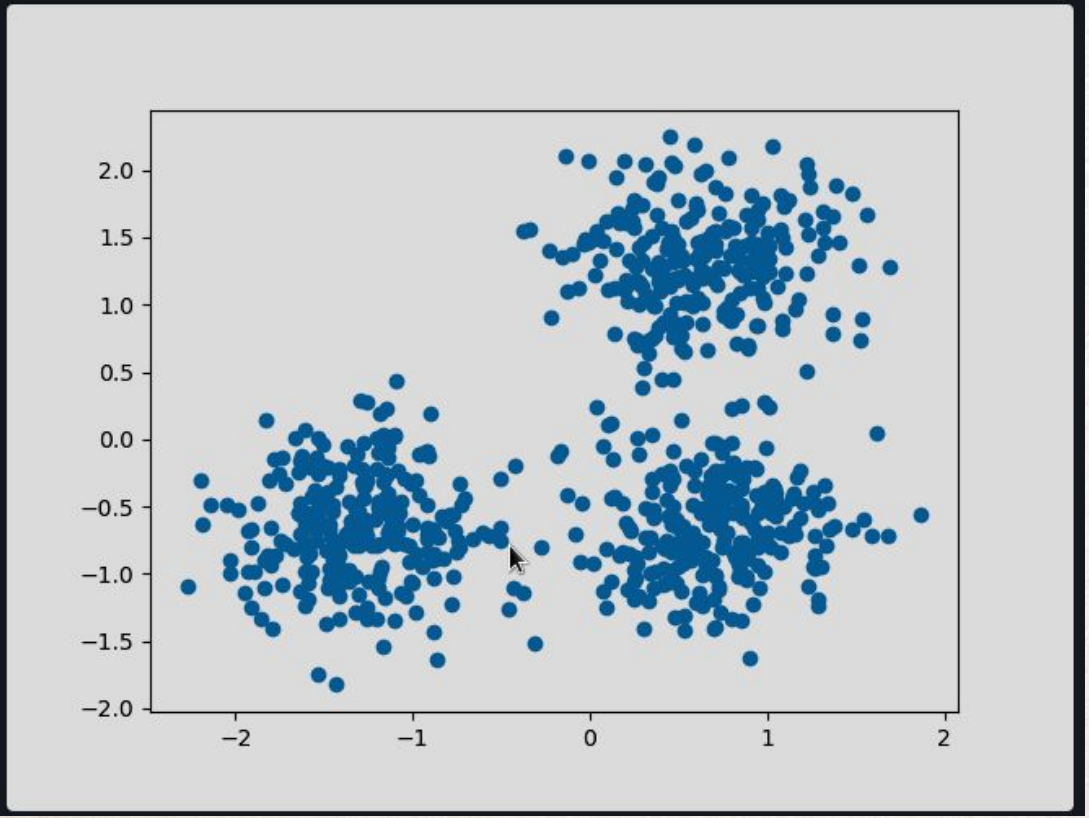
Ejemplo

Data generation

We use `make_blobs` to create 3 synthetic clusters.

```
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler

centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(
    n_samples=750, centers=centers, cluster_std=0.4, random_state=0
)

X = StandardScaler().fit_transform(X)
```

```python
import matplotlib.pyplot as plt

plt.scatter(X[:, 0], X[:, 1])
plt.show()
```



Compute DBSCAN

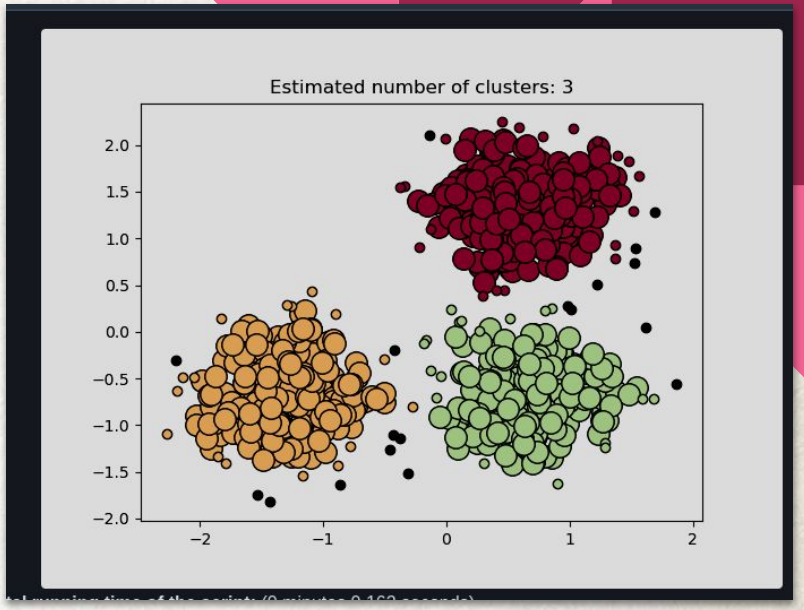
One can access the labels assigned by `DBSCAN` using the `labels_` attribute. Noisy samples are given the label math: `-1`.

```
import numpy as np

from sklearn import metrics
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.3, min_samples=10).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)
n_noise_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
```

Plot results

Core samples (large dots) and non-core samples (small dots) are color-coded according to the assigned cluster. Samples tagged as noise are represented in black.

```
unique_labels = set(labels)
core_samples_mask = np.zeros_like(labels, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True

colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))]
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=14,
    )

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(
        xy[:, 0],
        xy[:, 1],
        "o",
        markerfacecolor=tuple(col),
        markeredgecolor="k",
        markersize=6,
    )

plt.title(f"Estimated number of clusters: {n_clusters_}")
plt.show()
```




```
print(f"Homogeneity: {metrics.homogeneity_score(labels_true, labels):.3f}")
print(f"Completeness: {metrics.completeness_score(labels_true, labels):.3f}")
print(f"V-measure: {metrics.v_measure_score(labels_true, labels):.3f}")
print(f"Adjusted Rand Index: {metrics.adjusted_rand_score(labels_true, labels):.3f}")
print(
    "Adjusted Mutual Information:"
    f" {metrics.adjusted_mutual_info_score(labels_true, labels):.3f}"
)
print(f"Silhouette Coefficient: {metrics.silhouette_score(X, labels):.3f}")
```

Out:

```
Homogeneity: 0.953
Completeness: 0.883
V-measure: 0.917
Adjusted Rand Index: 0.952
Adjusted Mutual Information: 0.916
Silhouette Coefficient: 0.626
```

Compute DBSCAN

One can access the labels assigned by `DBSCAN` using the `labels_` attribute. Noisy samples are given the label math: `-1`.

```
import numpy as np

from sklearn import metrics
from sklearn.cluster import DBSCAN

db = DBSCAN(eps=0.3, min_samples=10).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)
n_noise_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n_clusters_)
print("Estimated number of noise points: %d" % n_noise_)
```

Sumario

- Los algoritmos de Aprendizaje no supervisado trabajan con datos no etiquetados.
- Cada algoritmo trabaja por sí solo identificando relaciones y patrones en los datos.
- Es más difícil de evaluar la performance de los algoritmos de aprendizaje no supervisado hay pocas métricas.
- En general estos algoritmos no usan el train-test splitting
- La normalización o estandarizado de los datos es importante para mejorar la exactitud de los resultados finales.
- k-Means es un ejemplo de Clustering particional
- Clustering Jerárquico es un ejemplo de Clustering aglomerativo/divisivo
- DBScan es un ejemplo de clustering por densidad