MINERÍA DE DATOS

Maximiliano Ojeda

muojeda@uc.cl

IIC-2433

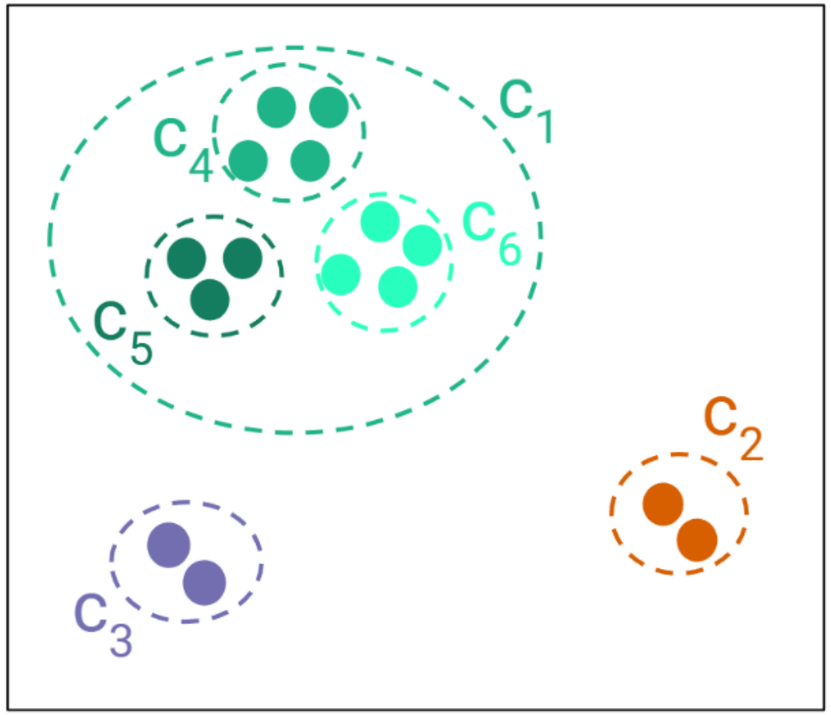
Clustering Jerarquico

Clustering Jerarquico

Es un método de agrupamiento **no supervisado** que busca **organizar objetos en una jerarquía** de grupos o clústeres, según su similitud o distancia.

Construye una jerarquía multinivel de grupos al unir clústeres más pequeños en otros más grandes o dividir un clúster grande en otros más pequeños.

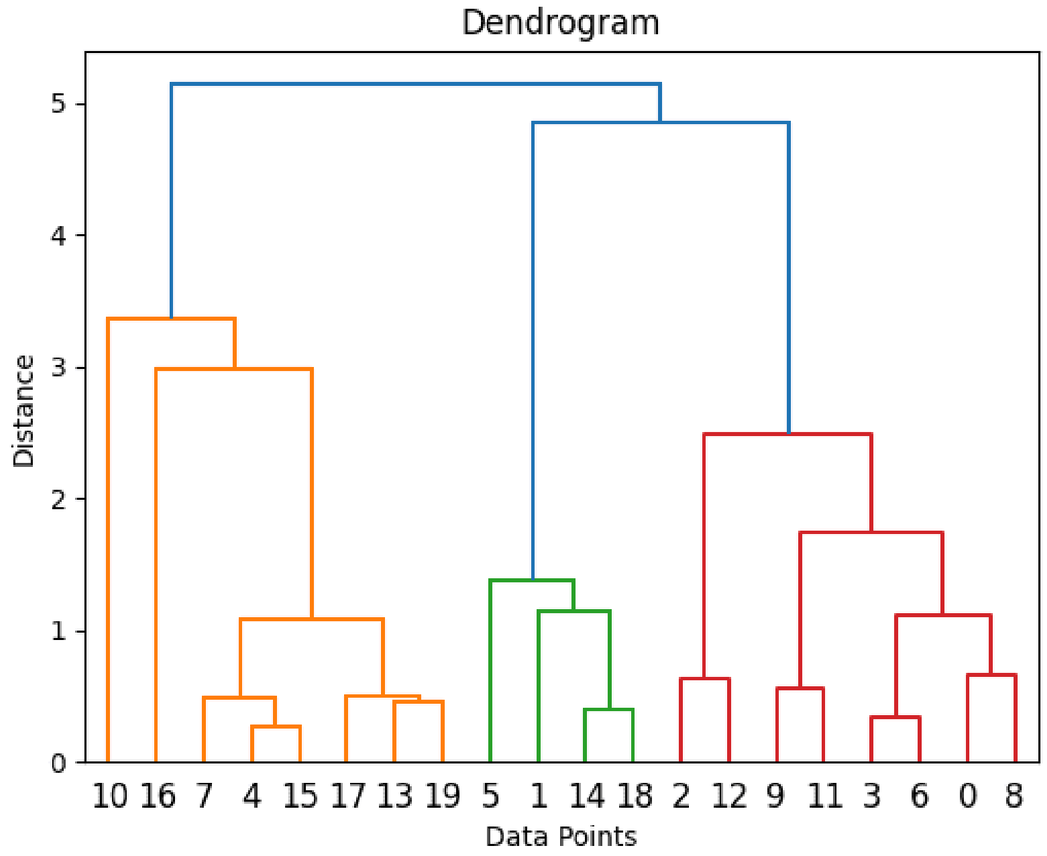
Esto da como resultado una estructura en forma de árbol conocida como **dendrograma**.



Dendograma

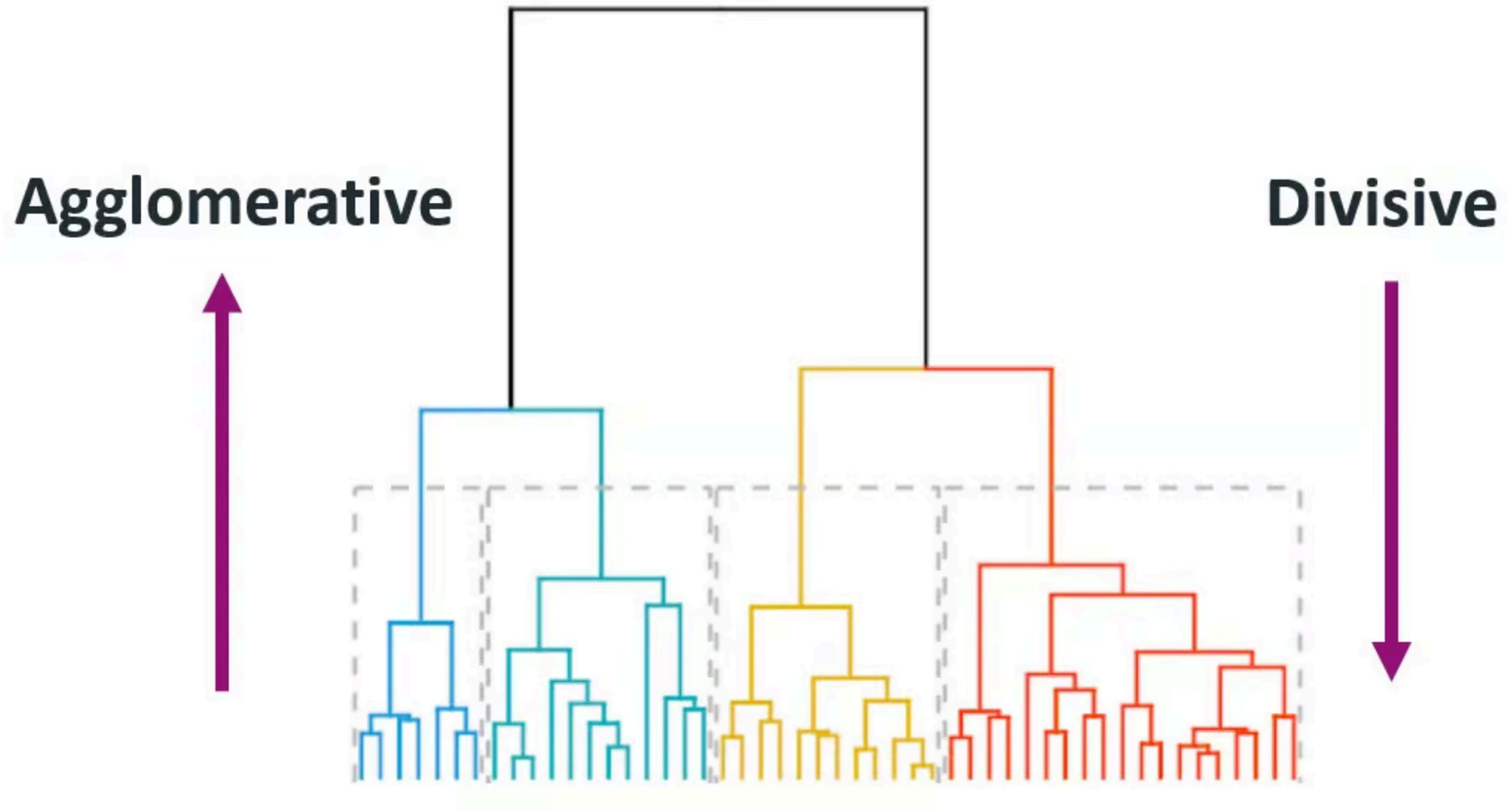
En lugar de asignar cada punto directamente a un grupo (como hace K-Means), el clustering jerárquico construye un **dendrograma**, que muestra cómo **los grupos se van formando o dividiendo paso a paso**.

- La altura de las ramas en un dendrograma representa la distancia o disimilitud a la que los clústeres se unen.
- Las alturas más bajas indican clústeres que se combinan a distancias menores, lo que representa una mayor similitud.



Tipos de clustering jerárquico

Aglomerativo (bottom-up): Empieza con cada punto separado y va uniendo los más similares hasta formar un clúster
Divisivo (top-down): Empieza con todos los puntos juntos y va dividiendo el grupo en partes más pequeñas.



Himanshu Sharma on Medium

HDBSCAN

HDBSCAN

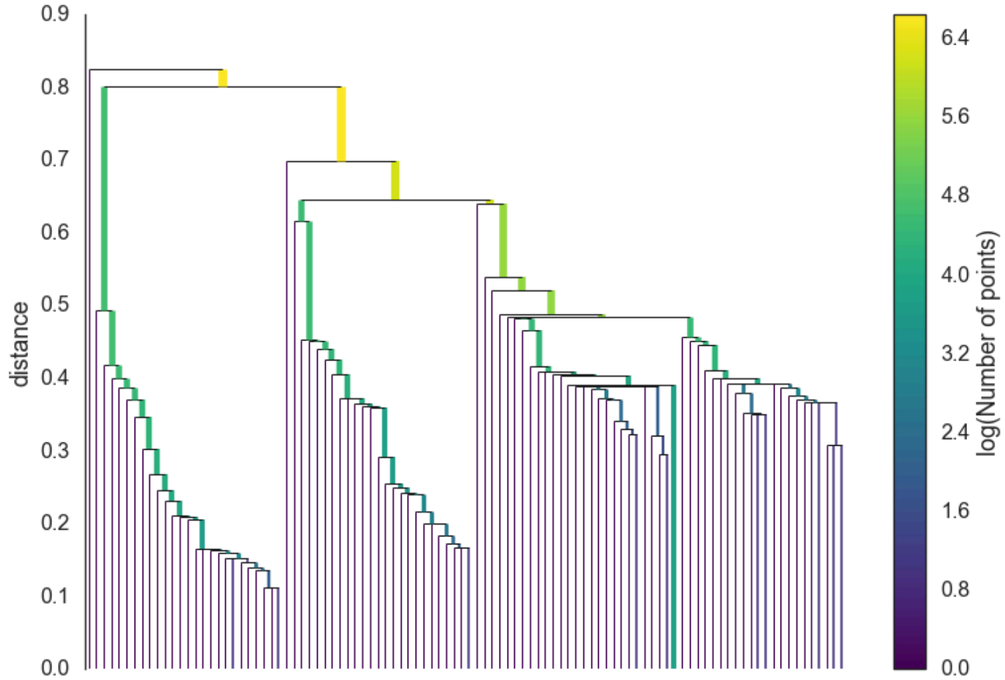
HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) es una versión mejorada de DBSCAN que combina la idea de densidad con la jerarquía de clústeres.

Diferencia con DBSCAN

DBSCAN usa un solo **valor de densidad (eps)** para todo el dataset, mientras que HDBSCAN construye una jerarquía de densidades y selecciona los clústeres más estables.

Ventajas

- Detecta clústeres con **distintas formas y densidades**
- No necesita especificar eps.
- Más robusto al ruido que DBSCAN.
- Produce también una medida de confianza para cada punto.



HDBSCAN

1. Mide la densidad local

Primero, estima **qué tan denso es el entorno de cada punto**. Para eso, calcula la distancia al vecino número k. Esa distancia se llama distancia de alcance principal (core distance):

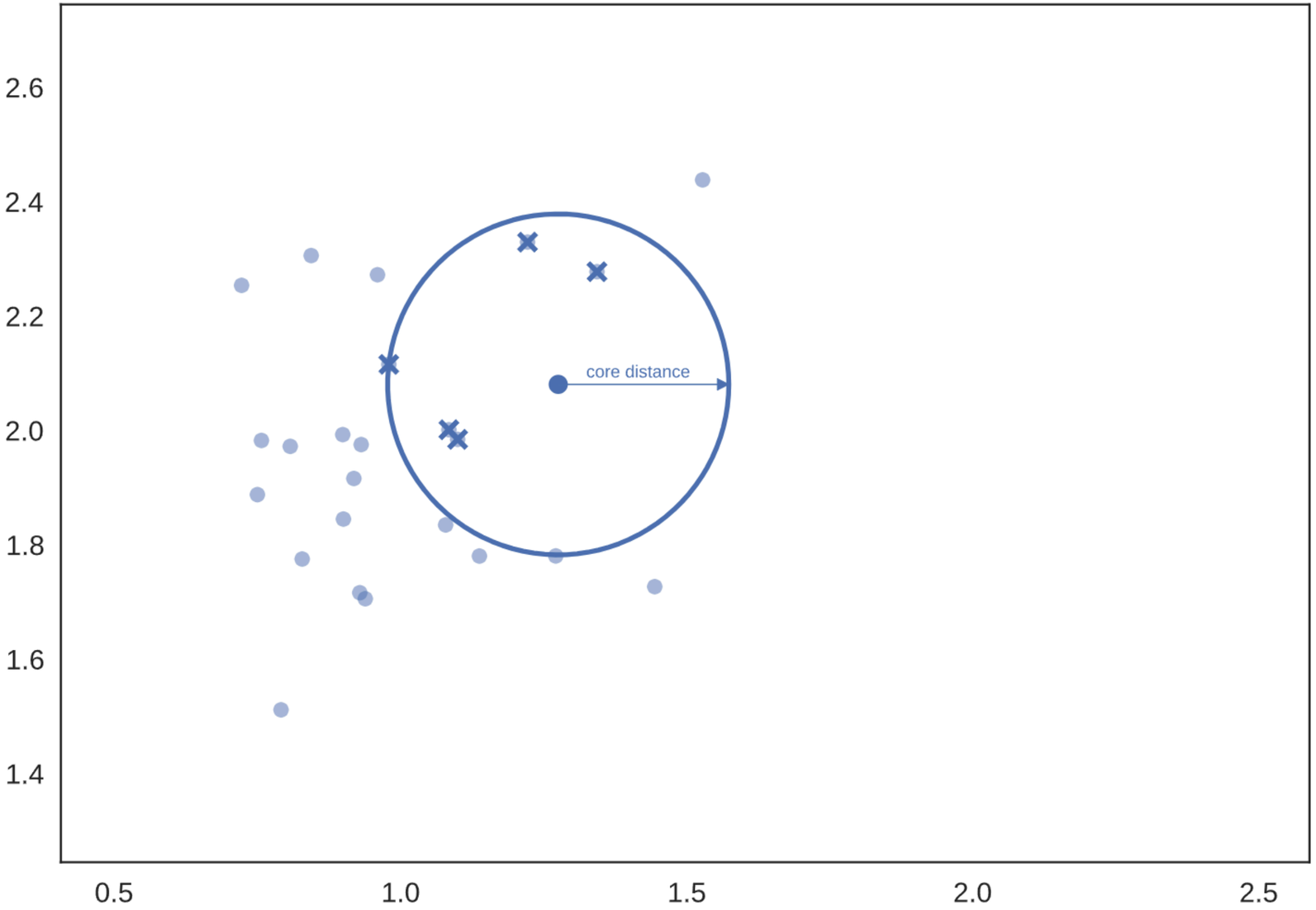
$$core_k(i) = \text{distancia hasta su k-esimo vecino mas cercano}$$

2. Mutual reachability distance

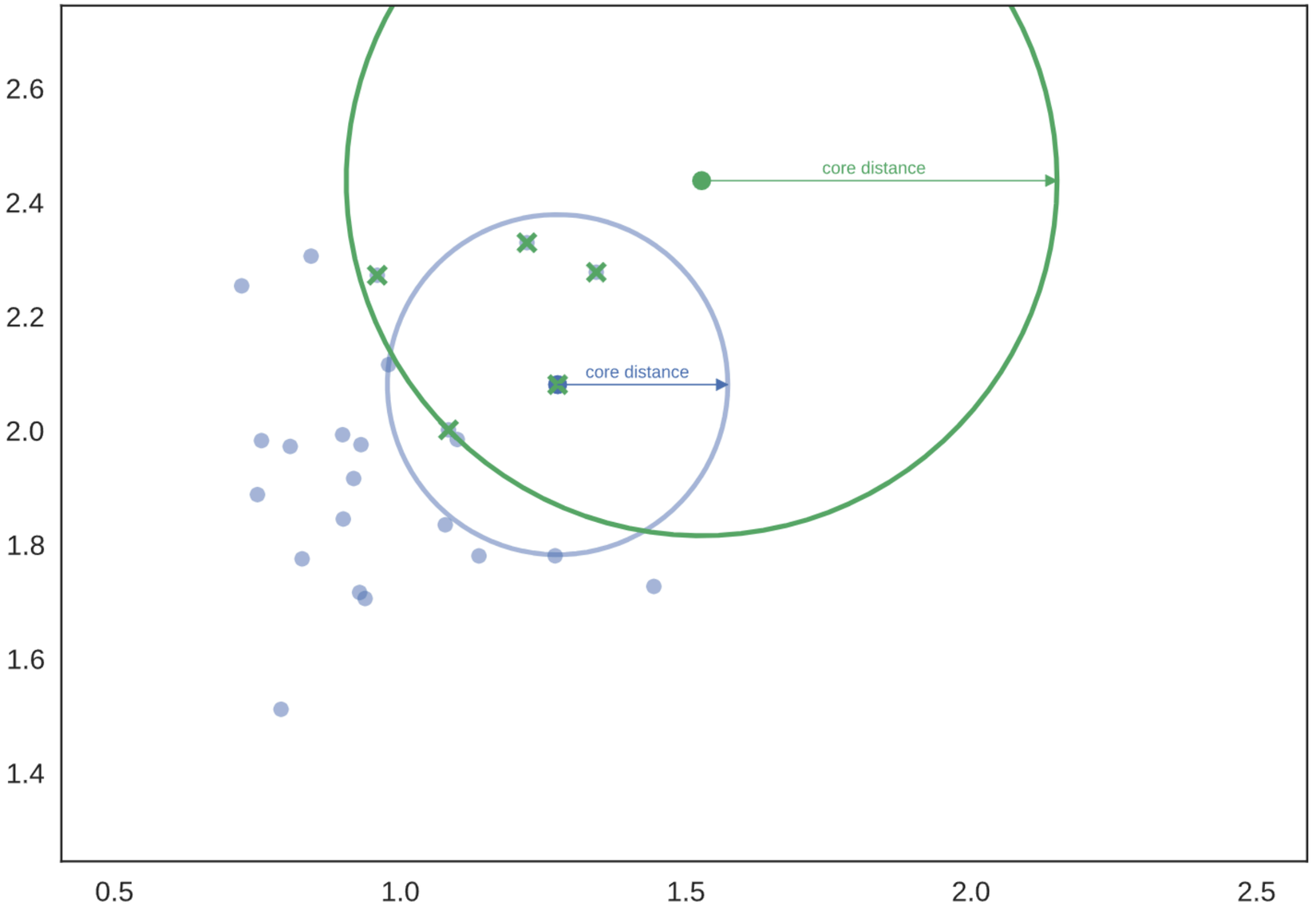
Define la distancia de alcance mutuo entre dos puntos i y j:

$$d_{mreach}(i,j) = max\{core_k(i), core_k(j), dist(i,j)\}$$

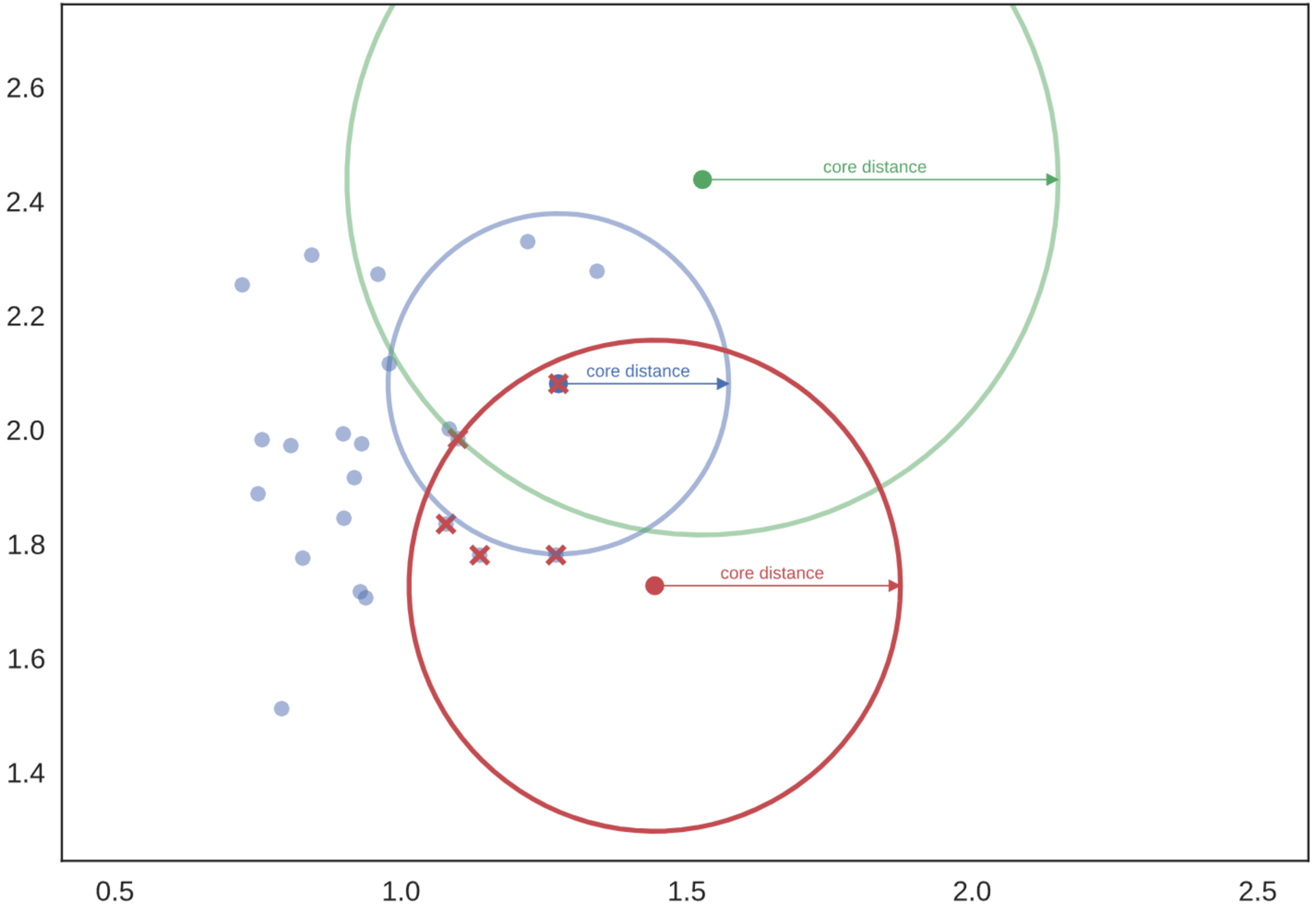
HDBSCAN



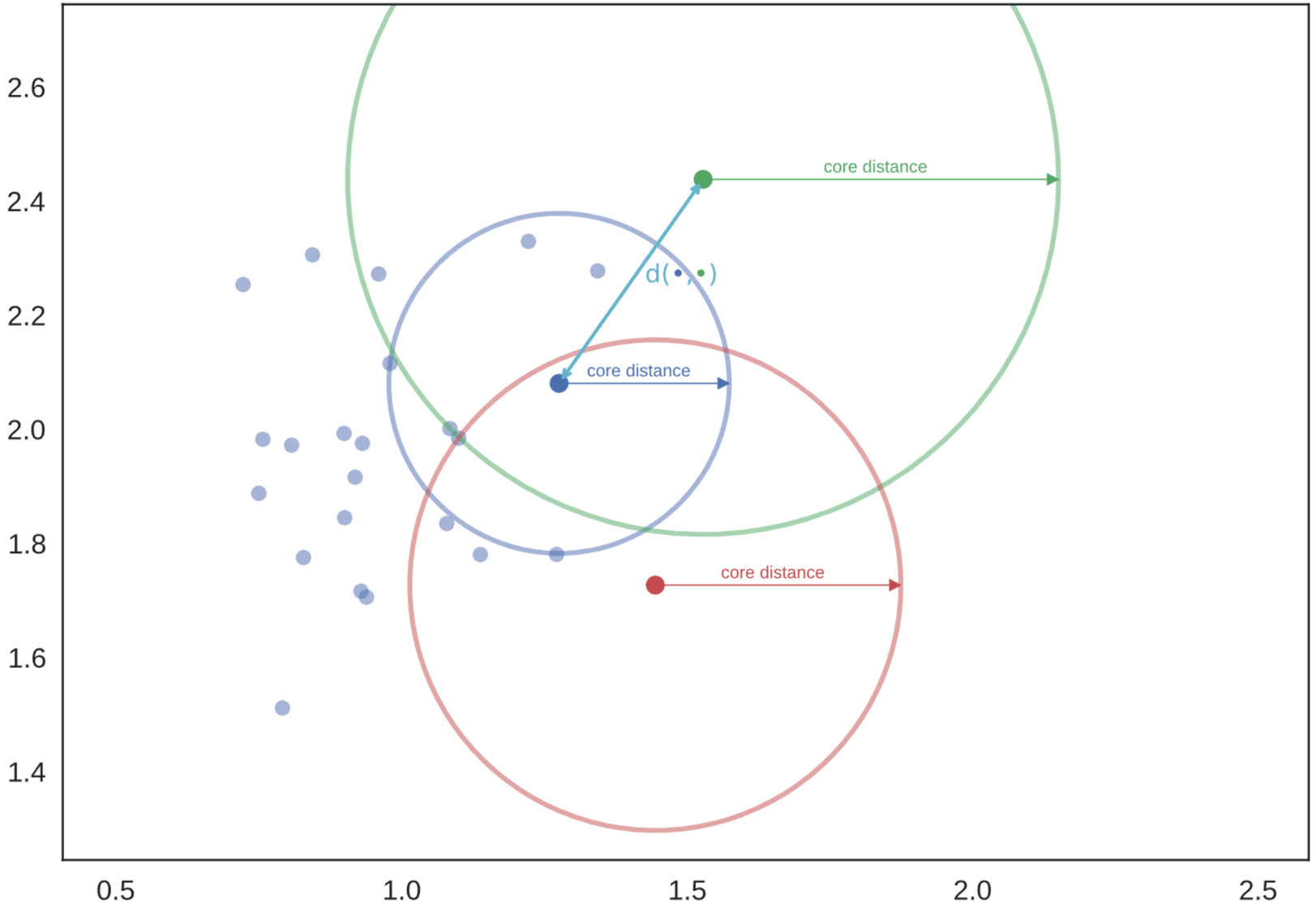
HDBSCAN



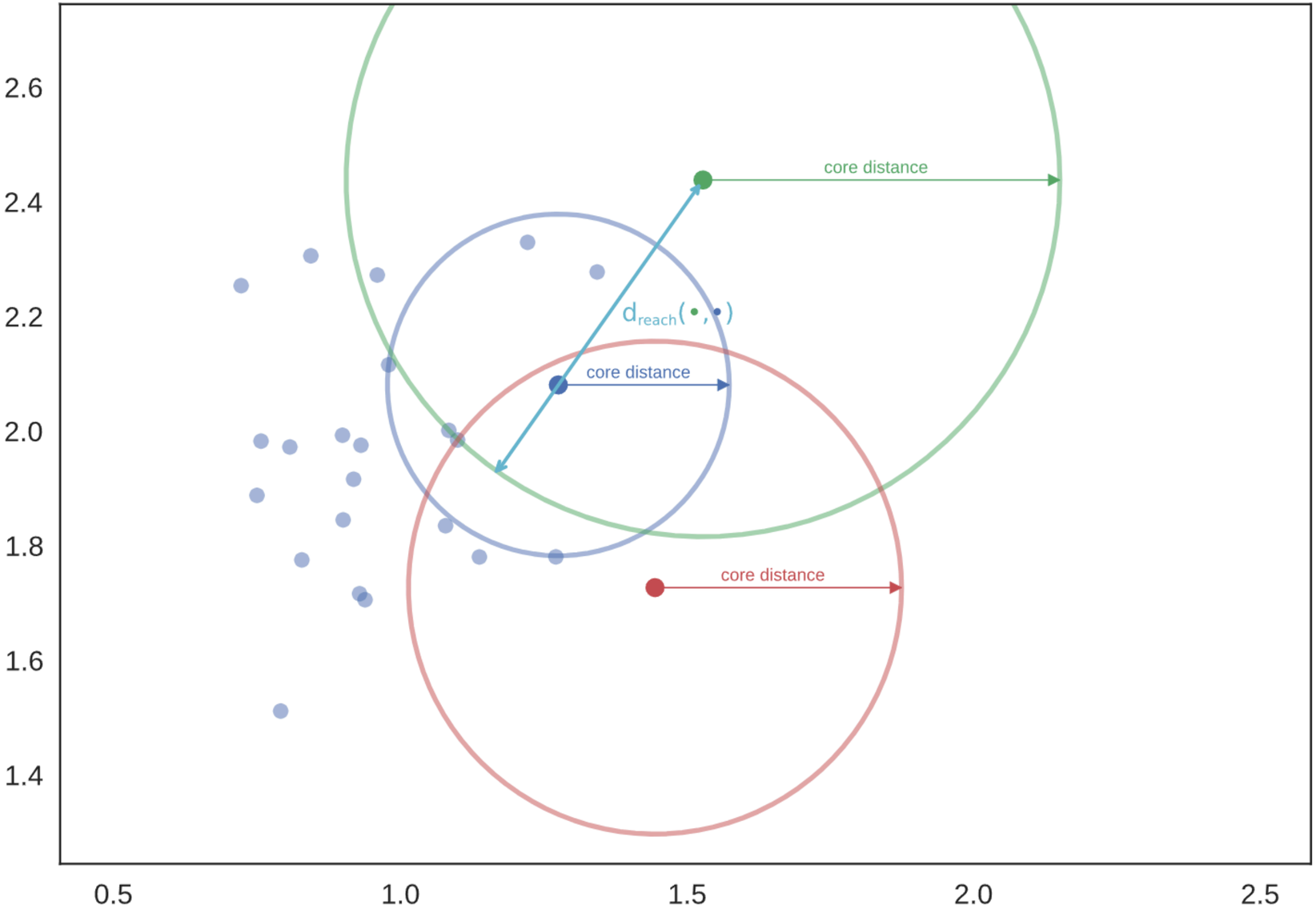
HDBSCAN


core distance
core distance
core distance
2.6
2.4
2.2
2.0
1.8
1.6
1.4
0.5
1.0
1.5
2.0
2.5

HDBSCAN


2.6
2.4
2.2
2.0
1.8
1.6
1.4
0.5
1.0
1.5
2.0
2.5
core distance
d(,)
core distance
core distance

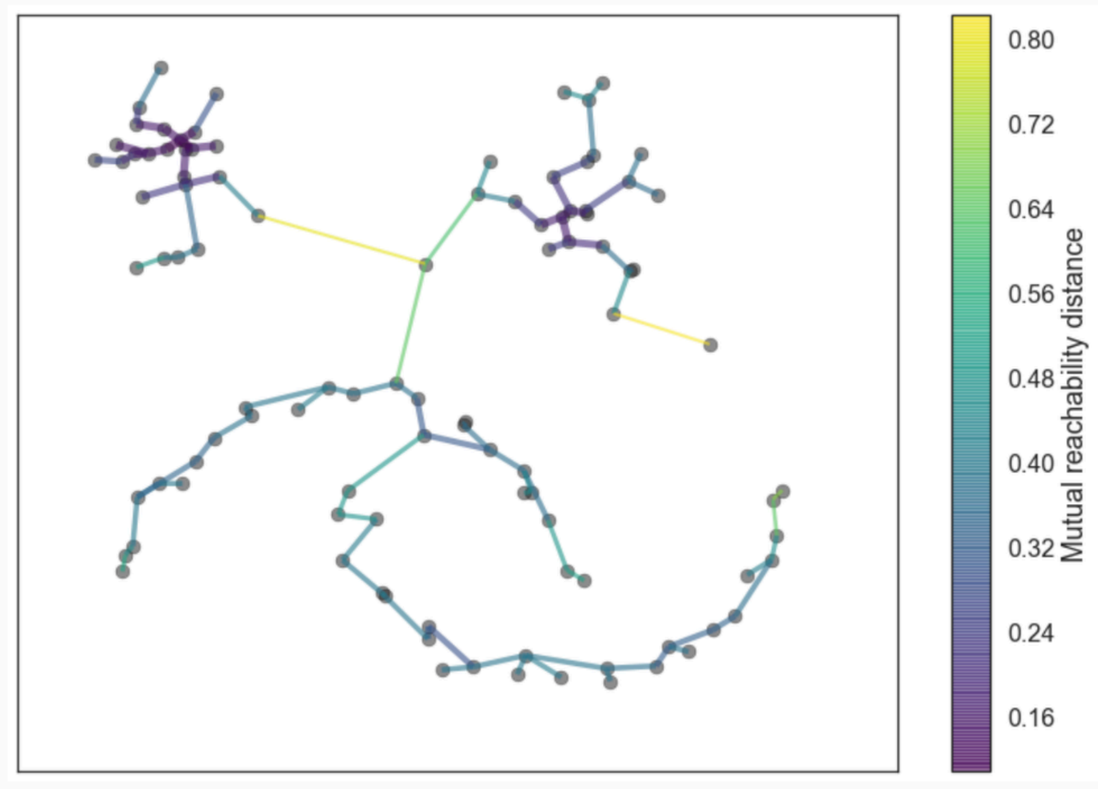
HDBSCAN


2.6
2.4
2.2
2.0
1.8
1.6
1.4
0.5
1.0
1.5
2.0
2.5
core distance
$d_{reach}(\bullet,\bullet)$
core distance
core distance

HDBSCAN

3. Construye un grafo y un árbol jerárquico

- Vamos a construir un grafo de conectividad sobre la base de la mutual reachability distance.
- Para esto usamos el **minimum spanning tree** usando el **algoritmo de Prim**. El invariante del algoritmo de Prim es agregar la arista de menor peso que conecta al árbol actual un nodo que no está conectado.

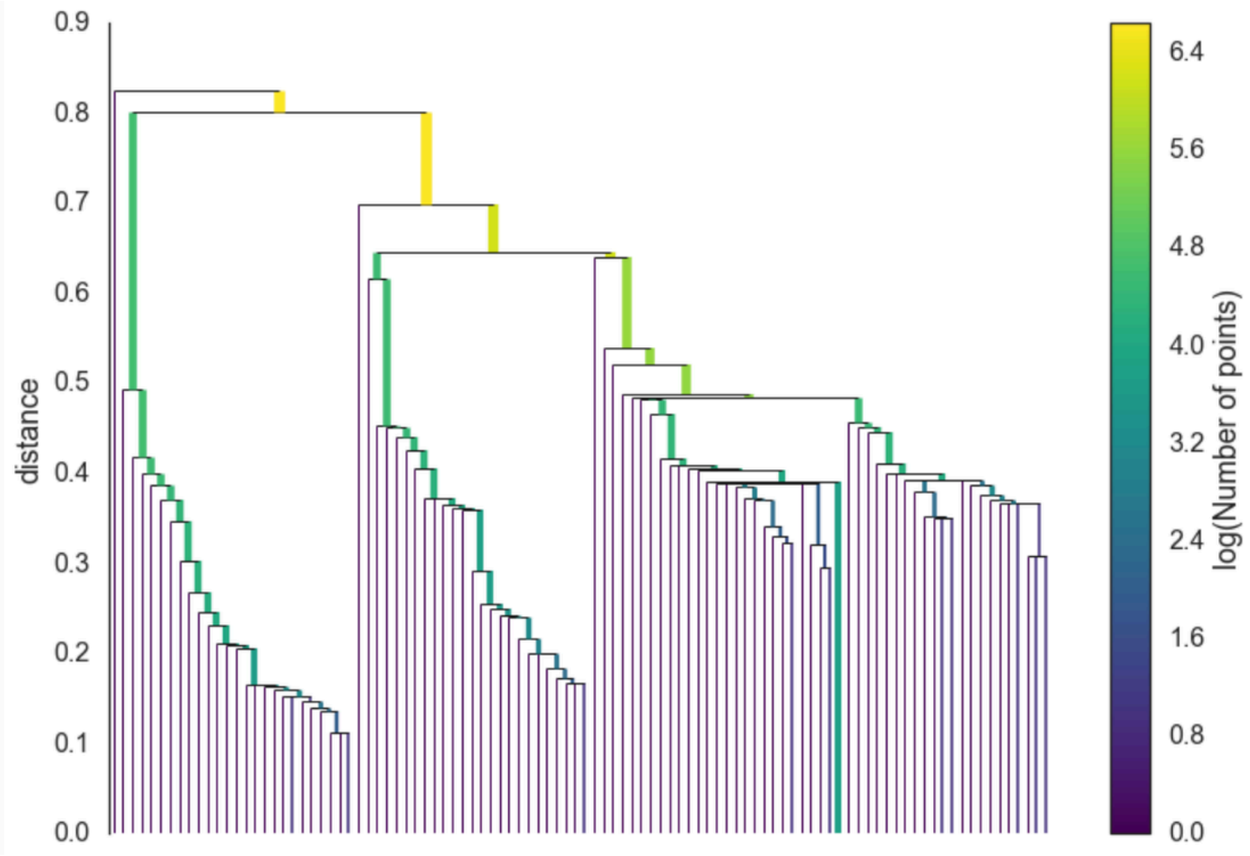


HDBSCAN

3. Construye un grafo y un árbol jerárquico

Dado el árbol de expansión mínima, el siguiente paso es **convertirlo en una jerarquía de componentes conectados**.

Esto se hace más fácilmente en orden inverso: se ordenan las aristas del árbol por distancia y luego se itera sobre ellas, **creando un nuevo clúster fusionado por cada arista**.

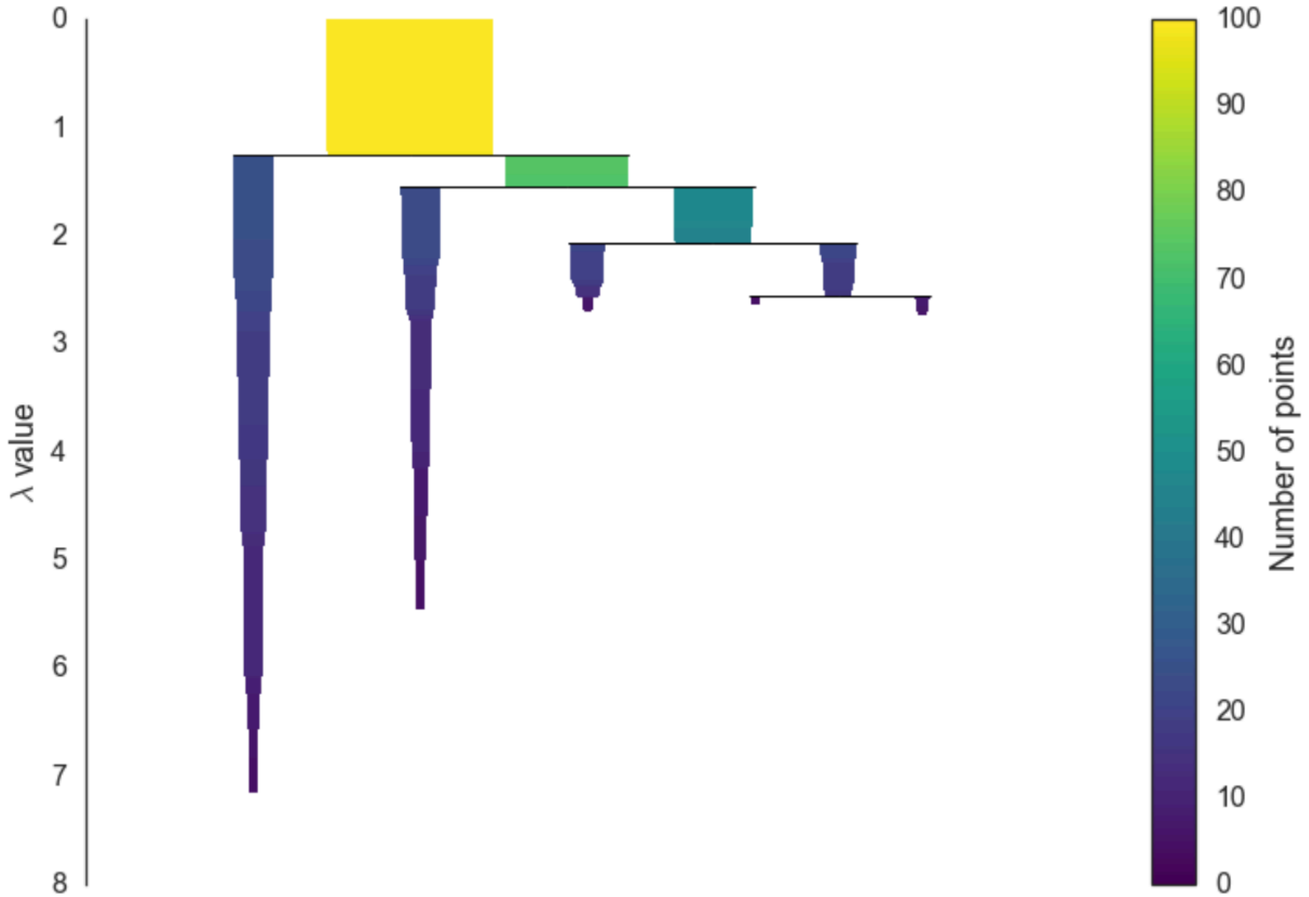


HDBSCAN

4. Condensar árbol

Condensaremos bottom-up el dendrograma con una condición de **minimum cluster size**.

- Si un cluster tiene menos datos, se mezcla con su cluster padre (merge-up).
- Si un split produce dos clusters que cumplen con la condición de tamaño, el split se mantiene



HDBSCAN

5. Extraer clusters

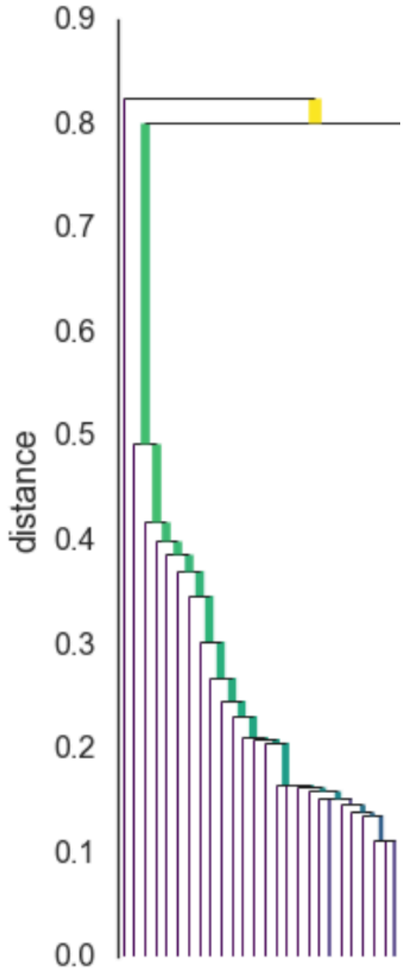
Se usa el concepto de **cluster estable** para indicar cuando un cluster sobrevive a los splits definidos por un umbral de distancia en el dendrograma. Es inverso a la distancia, por lo que se define la variable lambda como:

$$\lambda = \frac{1}{\text{distancia}}$$

Hacemos un recorrido top-down del dendrograma. Vamos a tener un lambda de nacimiento del cluster, y en la medida que d disminuya (cortamos más abajo) en algún momento el cluster va a morir.

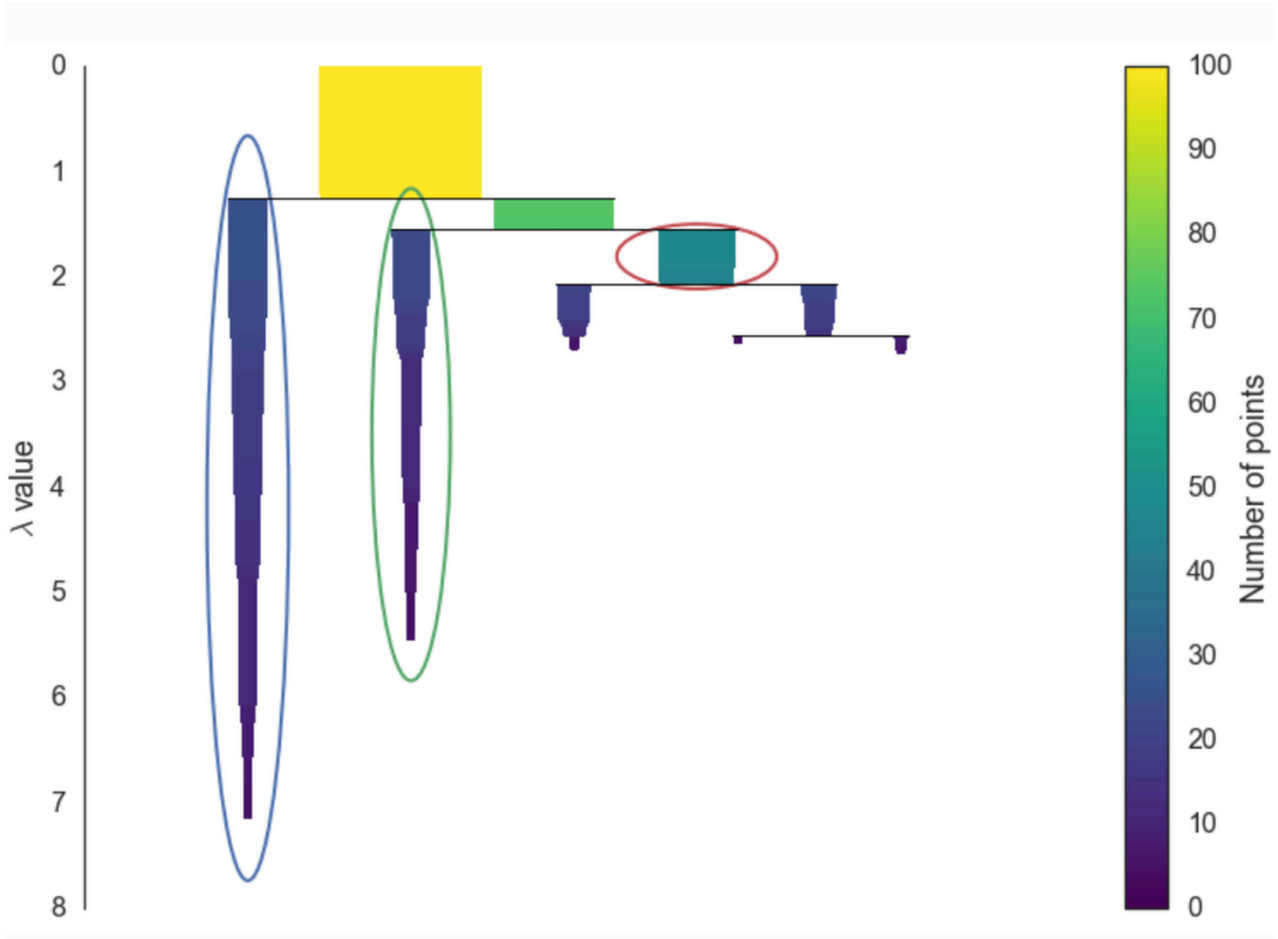
Para cada dato del cluster vamos a medir cuando sale del cluster (lambda del dato), y calcularemos la estabilidad del cluster según:

$$\text{Stability}(C) = \sum_{p \in C} (\lambda_{\text{death}}(p) - \lambda_{\text{birth}}(p))$$



HDBSCAN

5. Extraer clusters



HDBSCAN

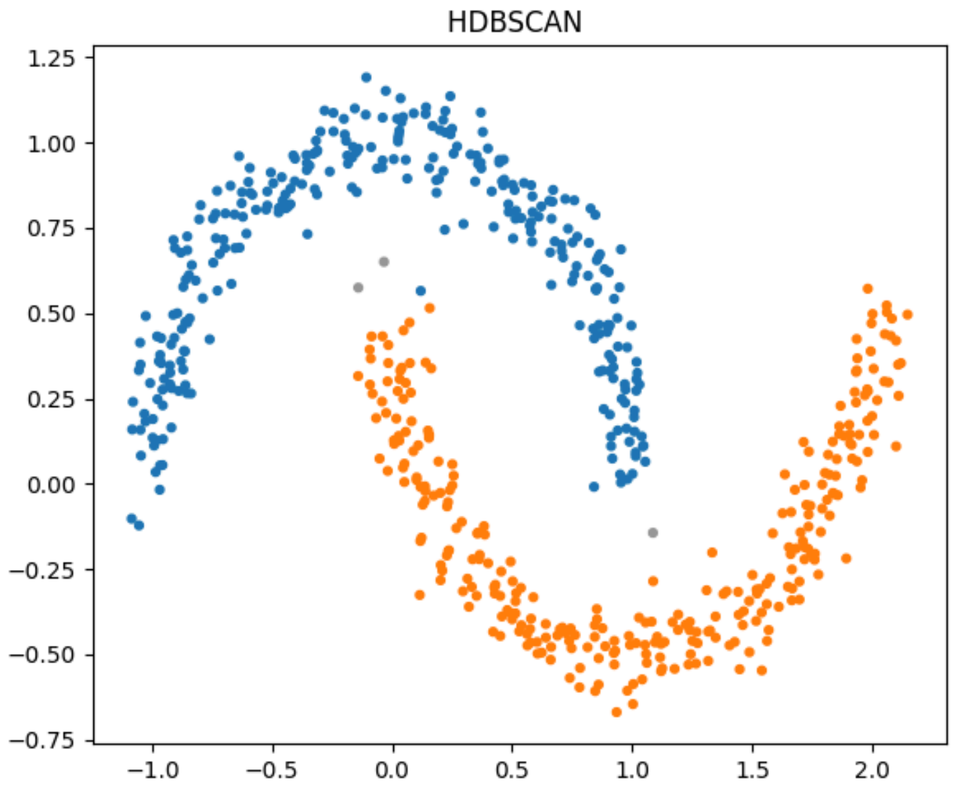
```
from sklearn.datasets import make_moons
import hdbscan

X, _ = make_moons(n_samples=600, noise=0.07, random_state=42)

# HDBSCAN
clusterer = hdbscan.HDBSCAN(min_cluster_size=20, min_samples=5)
labels = clusterer.fit_predict(X)
probs  = clusterer.probabilities_

plt.scatter(X[:,0], X[:,1], s=12, c=colors)
plt.title("HDBSCAN ")
plt.tight_layout(); plt.show()

array([1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       0.76670022, 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 1.        ,
       1.        , 1.        , 1.        , 1.        , 0.97069817,
       1.        , 1.        , 1.        , 1.        , 0.74991879])
```

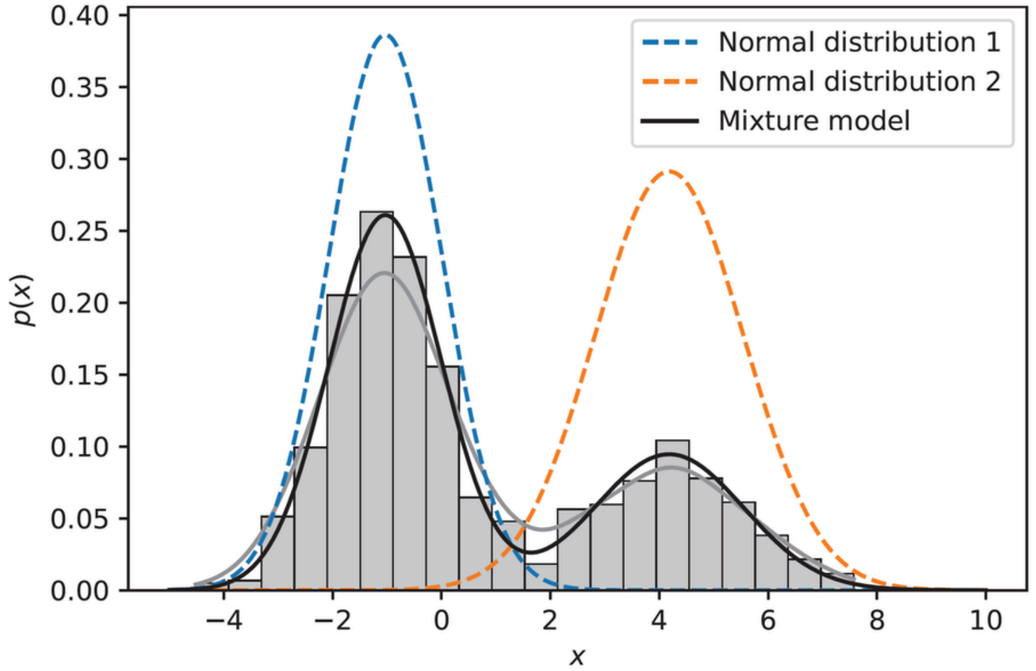


Gaussian Mixtures Model

GMM

Es un **modelo probabilístico** que asume que los puntos de datos se generan a partir de una mezcla de varias distribuciones gaussianas con **parámetros desconocidos**.

A diferencia de los métodos de clustering duros como K-Means, que asignan cada punto a un único clúster según el centroide más cercano, el GMM realiza un clustering suave **asignando a cada punto una probabilidad de pertenecer a múltiples clústeres**.



Recordatorio: Verosimilitud

La verosimilitud mide qué tan bien un conjunto de parámetros explica los datos que observamos. Mide la plausibilidad de diferentes valores de parámetros, dados los datos observados. Es una función de parámetros para datos fijos.

Si tenemos un modelo probabilístico $P(X \mid \theta)$

$X = \{x_1, x_2, \ldots, x_n\}$: son los datos observados

θ : son los parámetros del modelo (por ejemplo, media y varianza)

entonces la **verosimilitud** se define como:

$$L(\theta \mid X) = P(X \mid \theta) = \prod_{i=1}^{n} P(x_i \mid \theta)$$

y su versión logarítmica (más usada) es:

$$\log L(\theta \mid X) = \sum_{i=1}^{n} \log P(x_i \mid \theta)$$

MLE

La **estimación de máxima verosimilitud** (MLE) es un método estadístico que se utiliza para estimar los parámetros de una distribución de probabilidad mediante **maximizar la función de verosimilitud**.

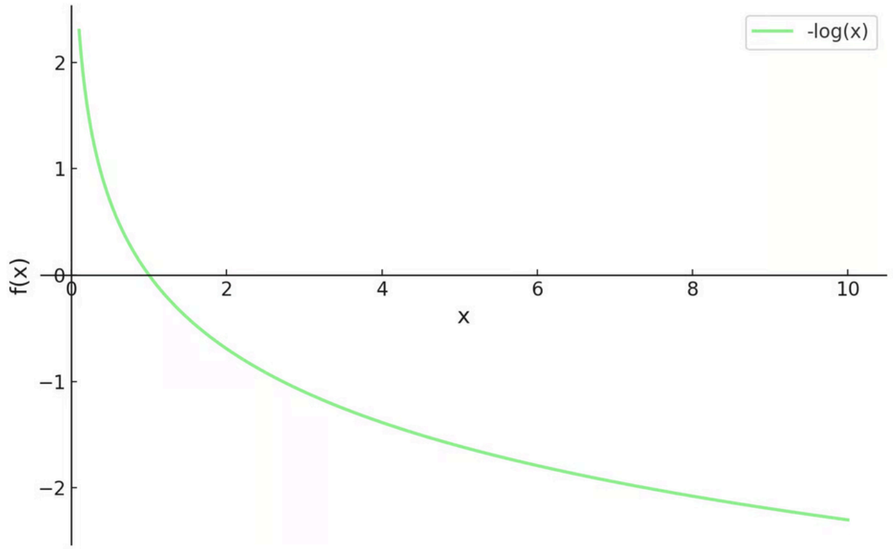
$$\hat{\theta} = \arg\max_{\theta} L(\theta)$$

o bien,

$$\hat{\theta} = \arg\max_{\theta} \sum_{i=1}^{n} \log P(x_i \mid \theta)$$

en general se suele utilizar la log-verosimilitud negativa

$$\hat{\theta} = \arg\min_{\theta} \left[-\sum_{i=1}^{n} \log P(x_i \mid \theta) \right]$$



Ejemplo: MLE

Supongamos que lanzamos un dado 12 veces y anotamos los resultados. Queremos modelar estos datos utilizando una distribución categórica , pero centrémonos en **estimar la probabilidad θ de sacar un seis**. En este ejemplo:

- **Parámetro (θ):** El valor que deseas estimar: probabilidad de sacar un seis.
- **Data (x):** Los resultados que observamos: 4 seis en 12 tiradas.

Ahora calculamos la función de verosimilitud, que, dado que obtuvimos 4 seises en 12 tiradas, sería:

$$L(\theta) = \theta^4 \times (1-\theta)^8$$

Tomamos la **log-verosimilitud negativa**, lo que nos da esta ecuación:

$$\ell(\theta) = -4\ln\theta - 8\ln(1-\theta)$$

Derivado la ecuación e igualando a 0:

$$\ell'(\theta) = \frac{8}{1-\theta} - \frac{4}{\theta} = 0 \quad \Rightarrow \quad \theta = \frac{1}{3}$$

Mixture Models

Es un modelo probabilístico que se utiliza para representar **datos que pueden provenir de categorías diferentes**, **cada una de las cuales se modela mediante una distribución de probabilidad distinta.**

Formalmente, si X es una variable aleatoria cuya distribución es una mezcla de K distribuciones componentes, la función de densidad de probabilidad de X puede escribirse como:

$$p(x) = \sum_{k=1}^{K} w_k f_k(x; \theta_k)$$

K : es el número de distribuciones componentes en la mezcla.

$f_k()$: es la función de densidad de la distribución componente k-ésima.

w_k : es el peso de mezcla del componente k-ésimo

θ_k : representa los parámetros del componente k-ésimo

Gaussian Mixture Models

Es un modelo probabilístico que asume que los puntos de datos se generan a partir de una **mezcla de varias distribuciones gaussianas con parámetros desconocidos**.

$$p(\mathbf{x}) = \sum_{k=1}^{K} w_k \mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$$

K : es el número de distribuciones componentes en la mezcla.

$\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k)$: es la función de densidad normal multivariante para el componente k-ésimo

Σ_k : es la matriz de covarianza del componente gaussian

μ_k : es el vector de medias del componente gaussiano

w_k : es el peso de mezcla del componente k-ésimo

$$\mathcal{N}(\mathbf{x}; \mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^d |\Sigma_k|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_k)^\top \Sigma_k^{-1} (\mathbf{x} - \mu_k)\right)$$

Aprender Parámetros GMM

El objetivo es **encontrar los parámetros del GMM** (medias, covarianzas y coeficientes de mezcla) que mejor expliquen los datos observados

Verosimilitud,

$$\mathcal{L}(\theta \mid X) = \prod_{i=1}^{n} \left[\sum_{k=1}^{K} w_k \mathcal{N}(x_i; \mu_k, \Sigma_k) \right]$$

Log-Verosimilitud,

$$\ell(\theta \mid X) = \sum_{i=1}^{n} \log \left[\sum_{k=1}^{K} w_k \mathcal{N}(x_i; \mu_k, \Sigma_k) \right]$$

No podemos aplicar directamente MLE para estimar los parámetros de un GMM:

- La función de log-verosimilitud es **altamente no lineal** y compleja de maximizar analíticamente
- El modelo **posee variables latentes** (los pesos de mezcla) que **no son directamente observables en los datos**

Expectation-Maximization (EM)

Es un método para encontrar las **estimaciones de máxima verosimilitud** de los parámetros en modelos estadísticos que dependen de variables latentes no observadas, como GMM.

El algoritmo comienza inicializando aleatoriamente los parámetros del modelo, y luego itera entre dos pasos :

1. Expectación (E-step):

Calcula la **log-verosimilitud esperada** del modelo con respecto a la distribución de las variables latentes, dadas las observaciones y los valores actuales de los parámetros del modelo. En este paso, **se estima la probabilidad de las variables latentes** (la probabilidad de que cada punto pertenezca a cada componente).

2. Maximización (M-step):

Actualiza los parámetros del modelo para maximizar la log-verosimilitud de los datos observados, utilizando las probabilidades de las variables latentes obtenidas en el paso E.

Estos dos pasos se **repiten hasta la convergencia**, la cual suele determinarse mediante un umbral en el cambio de la log-verosimilitud o tras alcanzar un número máximo de iteraciones.

Asginación de Clusters

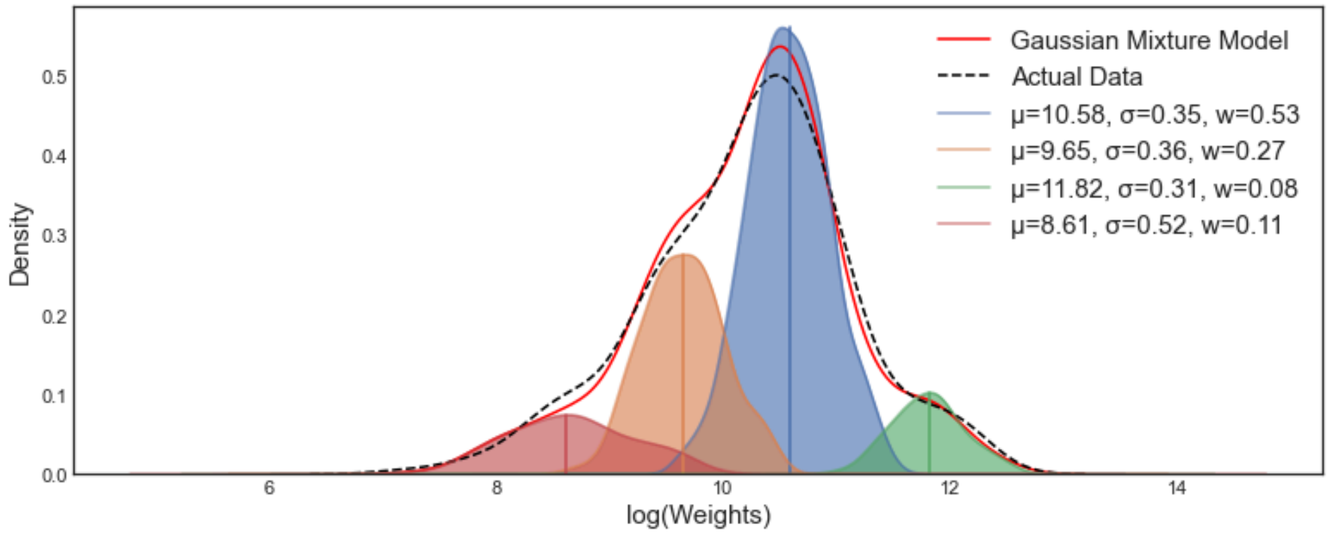
Una vez que el modelo aprende los parámetros de cada gaussiana, puede calcular **la probabilidad de que un punto x_i haya sido generado por cada componente k**.

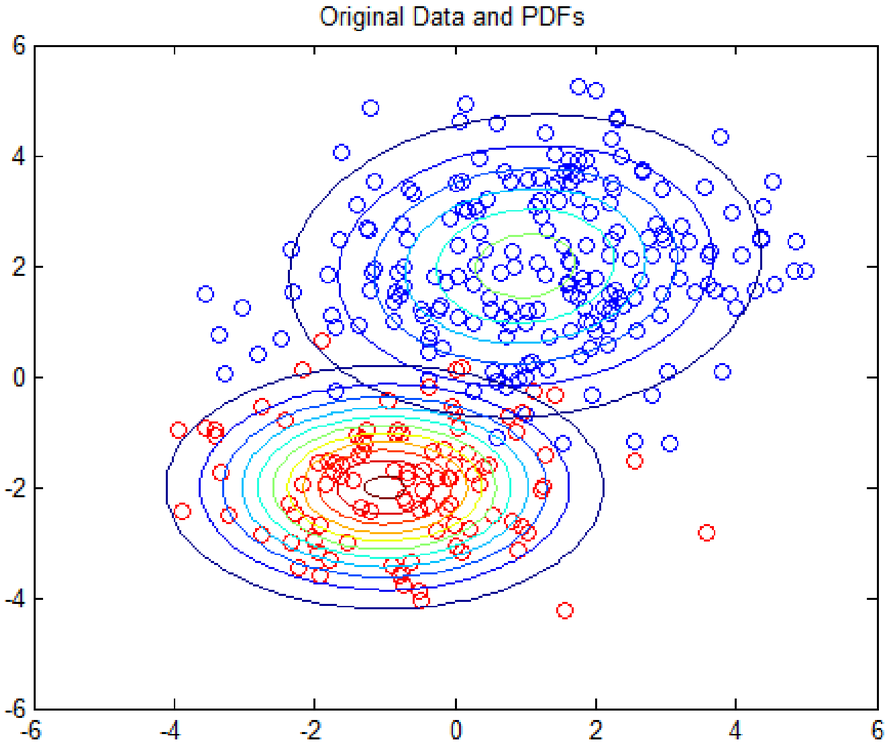
Para cada punto x_i, se calcula su responsabilidad o probabilidad posterior de pertenecer al clúster k:

$$\gamma_{ik} = P(k \mid x_i) = \frac{w_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} w_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

γ_{ik} son las probabilidades de pertenencia, es decir, cuánto contribuye cada gaussiana a explicar el punto

GMM


Gaussian Mixture Model
Actual Data
μ=10.58, σ=0.35, w=0.53
μ=9.65, σ=0.36, w=0.27
μ=11.82, σ=0.31, w=0.08
μ=8.61, σ=0.52, w=0.11
Density
log(Weights)
0.0
0.1
0.2
0.3
0.4
0.5
6
8
10
12
14


Original Data and PDFs
6
4
2
0
-2
-4
-6
-6
-4
-2
0
2
4
6

GMM

El único parámetro que debemos definir es la cantidad de clusters: K

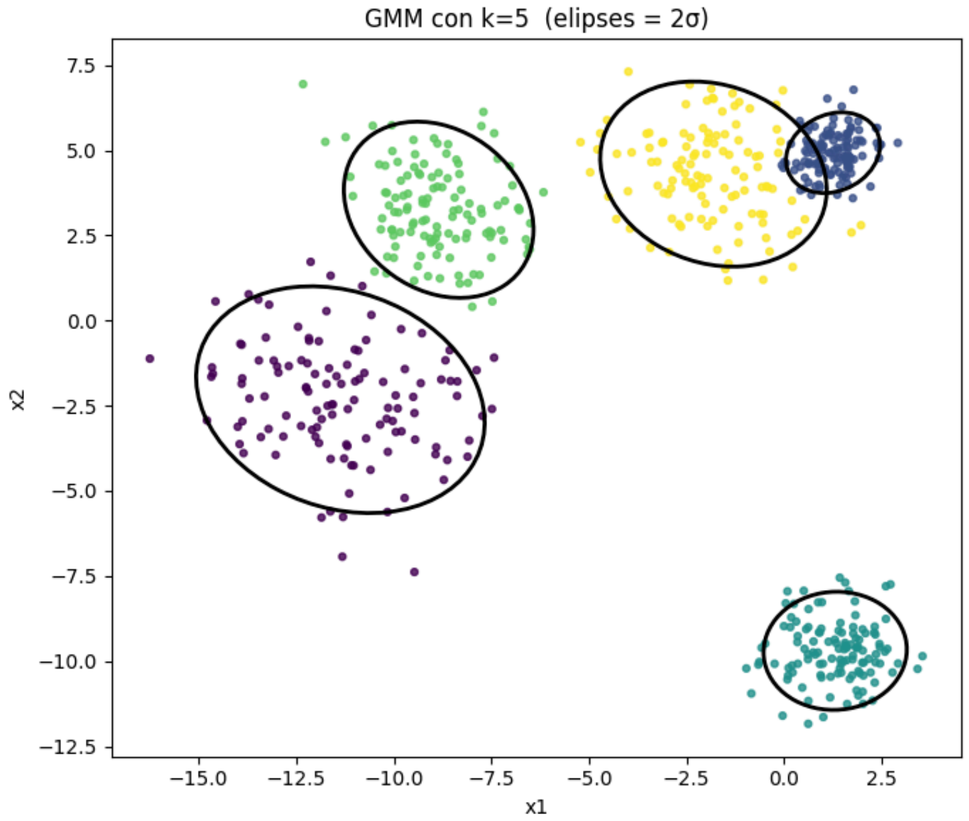
```python
from sklearn.mixture import GaussianMixture
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

X, _ = make_blobs(n_samples=300, centers=5, cluster_std=[1.3, 0.6, 0.9, 1.9, 1.4])

# Crear y ajustar un modelo GMM con 5 componentes
gmm = GaussianMixture(n_components=5, random_state=42)
gmm.fit(X)

# Predecir a qué componente pertenece cada punto
labels = gmm.predict(X)

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.title("Clustering con Gaussian Mixture Model (GMM)")
plt.show()
```



MINERÍA DE DATOS

Maximiliano Ojeda

muojeda@uc.cl

IIC-2433