



## SESIÓN ALGORITMOS DE CLUSTERIZACIÓN

### CONTENIDOS:

- Agrupamiento jerárquico: qué es el agrupamiento jerárquico.
- Tipos de agrupamiento jerárquico.
  - Aglomerativo.
  - Divisivo.
    - Qué es un dendrograma y para qué sirve.
    - Agrupación jerárquica utilizando el dendrograma
    - Etapas para la realización de agrupamiento jerárquico.
    - Elección de la cantidad de clusters.
    - Ventajas y desventajas de esta técnica.
    - Implementación en Python.
- Algoritmo K-Means: En qué consiste el algoritmo K-Means.
- Elección del valor de  $K$ .
  - Método del codo.
    - Coeficiente de silueta.
    - Ventajas y desventajas de utilizar el algoritmo.
    - Implementación en Python.
- Algoritmo DBSCAN: en qué consiste el algoritmo DBSCAN.
  - Parámetros del algoritmo.
  - Elección del valor de  $\epsilon$ .
  - Ventajas y desventajas de utilizar el algoritmo.
  - Implementación en Python.

### QUÉ ES EL AGRUPAMIENTO JERÁRQUICO

El agrupamiento jerárquico es una técnica de agrupación que organiza los datos en una estructura similar a un árbol, permitiendo analizar las relaciones entre los diferentes grupos a diferentes niveles de granularidad.



A diferencia de métodos como **K-Means**, donde debes definir de antemano el número de clusters ( $K$ ), en el agrupamiento jerárquico los clusters se forman de manera progresiva mediante un proceso de fusión o división.

Los resultados suelen representarse con un **dendrograma**, que es un diagrama en forma de árbol donde los datos más similares se agrupan en ramas cercanas y los más diferentes en ramas más separadas.

## TIPOS DE AGRUPAMIENTO JERÁRQUICO

Existen dos enfoques principales en el agrupamiento jerárquico:

- Algoritmo Aglomerativo.
- Algoritmo Divisivo.

### Algoritmo Aglomerativo (Hierarchical Agglomerative Clustering, HAC)

Es el más utilizado y se basa en una estrategia "**bottom-up**" (*de abajo hacia arriba*).

#### ¿Cómo funciona?

1. Se comienza con cada punto de datos como un cluster individual.
2. En cada paso, los dos clusters más similares se fusionan en uno solo.
3. Este proceso continúa hasta que todos los puntos de datos pertenecen a un solo cluster o hasta alcanzar un número deseado de grupos.

#### Métodos para medir la similitud entre clusters:

Para decidir qué clusters unir, se utilizan distintas métricas de distancia, como:

- **Vinculación completa (complete linkage):** distancia entre los puntos más alejados de cada cluster.
- **Vinculación simple (single linkage):** distancia entre los puntos más cercanos de cada cluster.

- **Vinculación promedio (average linkage):** promedio de todas las distancias entre los puntos de los clusters.
- **Centroide (centroid linkage):** distancia entre los centroides de los clusters.

### Ejemplo Algoritmo Aglomerativo

Imagina que tienes un grupo de personas y quieres agruparlas según su similitud en características como peso y altura. Primero, cada persona es un cluster. Luego, las dos más parecidas se unen en un solo grupo, y el proceso sigue hasta formar una jerarquía completa.

### Algoritmo Divisivo (Divisive Hierarchical Clustering, DHC)

Es menos común y sigue una estrategia "**top-down**" (*de arriba hacia abajo*).

### ¿Cómo funciona?

1. Se comienza con todos los datos en un solo cluster grande.
2. En cada paso, el cluster se divide en dos subgrupos basándose en diferencias entre los datos.
3. Este proceso continúa hasta que cada punto de datos es su propio cluster o hasta alcanzar un número deseado de grupos.

### Métodos para dividir los clusters:

- Se pueden usar técnicas como **K-Means** o **división basada en la distancia**.
- La separación se basa en encontrar la mayor heterogeneidad dentro del grupo y dividirlo en consecuencia.

## Ejemplo Algoritmo Divisivo

Imagina que tienes una cesta llena de frutas de diferentes tipos. Primero, consideras toda la cesta como un solo grupo. Luego, separas las frutas en dos grandes grupos (por ejemplo, frutas redondas y frutas alargadas). Después, sigues dividiendo hasta que cada tipo de fruta queda en su propio grupo.

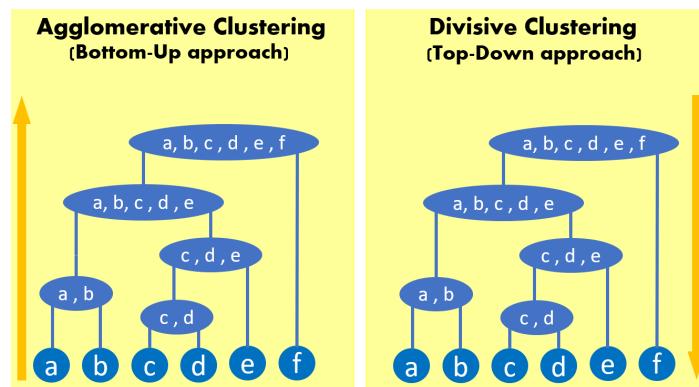


Ilustración 1 Imagen de Reddit: Algoritmo de Clustering Jerárquico

## ¿Cuál es mejor: Aglomerativo o Divisivo?

Depende del caso:

- ⇒ **Aglomerativo** es más usado porque es más eficiente computacionalmente.
- ⇒ **Divisivo** puede ser mejor cuando hay grandes diferencias en los datos y queremos dividirlos de forma clara.

Ambos producen un **dendrograma**, que permite elegir el número de clusters analizando la estructura del árbol.



## QUÉ ES UN DENDROGRAMA Y PARA QUÉ SIRVE

Un **dendrograma** es un **diagrama en forma de árbol** que muestra cómo se agrupan los datos en un proceso de agrupamiento jerárquico.

En este gráfico, los datos más similares están conectados en ramas cercanas, mientras que los datos más distintos están en ramas separadas.

### ¿Para qué sirve un dendrograma?

⇒ **Visualizar la jerarquía de los clusters**

- Permite ver cómo se forman los grupos en diferentes niveles de similitud.
- Se puede analizar qué elementos están más relacionados entre sí.

⇒ **Determinar el número óptimo de clusters**

- Al cortar el dendrograma en un determinado nivel de altura, se pueden identificar los grupos más significativos.
- Se observa dónde hay grandes saltos en la distancia de fusión (esto indica que unir más clusters haría los grupos menos homogéneos).

⇒ **Comparar la similitud entre elementos**

- Elementos que se agrupan en una misma rama antes que otros son más similares entre sí.
- Cuanto más tarde dos puntos se fusionan en un solo cluster, mayor es su diferencia.

### ¿Cómo interpretar un dendrograma?

El dendrograma tiene dos ejes:

- ◊ **Eje horizontal (X):** Representa los elementos que se están agrupando.
- ◊ **Eje vertical (Y):** Representa la distancia o diferencia entre los clusters cuando se fusionan.

### Ejemplo de interpretación:

- Si el dendrograma muestra un salto grande en la altura de la fusión, significa que la unión de esos clusters introduce una gran diferencia.
- Un buen corte en el dendrograma se hace antes de estos grandes saltos, asegurando clusters más homogéneos.

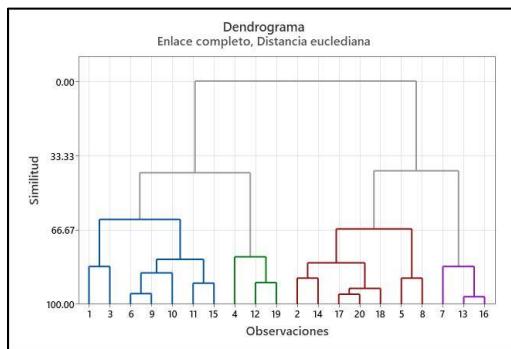


Ilustración 2 Ejemplo dendrograma. Support.minitab.com

## AGRUPACIÓN JERÁRQUICA UTILIZANDO EL DENDROGRAMA

Como vimos antes, el dendrograma es una representación visual del agrupamiento jerárquico. Ahora, veamos cómo usarlo para agrupar datos de manera efectiva.

### Etapas para la realización de agrupamiento jerárquico

Para realizar un agrupamiento jerárquico, se siguen los siguientes pasos clave:

#### 1.- Calcular la matriz de distancias

Se mide la similitud entre los datos utilizando una métrica de distancia, como la distancia euclídea o manhattan.



## 2.- Aplicar un método de enlace (linkage)

Esto define cómo se combinan los clusters en cada iteración:

- **Enlace simple:** Usa la distancia mínima entre los elementos de dos clusters.
- **Enlace completo:** Usa la distancia máxima entre elementos de los clusters.
- **Enlace promedio:** Usa el promedio de las distancias entre todos los elementos de ambos clusters.
- **Enlace por centroide:** Usa la distancia entre los centroides de los clusters.

## 3.- Construcción del dendrograma

Se representa gráficamente la jerarquía de clusters para analizar cómo se agrupan los datos.

## 4.- Elección del número de clusters

Se analiza el dendrograma y se elige un punto de corte para definir cuántos clusters habrá.

## 5.- Asignación final de clusters

Se agrupan los datos en los clusters definidos.

### Elección de la cantidad de clusters

Elegir el número correcto de clusters es **fundamental** para una buena segmentación de los datos. Existen distintos métodos que nos permiten determinar el número óptimo de cluster y los veremos a continuación:

### Métodos para determinar el número óptimo de clusters en agrupamiento jerárquico:

#### 1. Corte del dendrograma:

- ▶ Se observa en qué punto del dendrograma hay un gran salto en la altura de fusión.
- ▶ Se corta antes de este gran cambio para obtener clusters más homogéneos.



## 2. Coeficiente de silueta:

- ▶ Mide la cohesión y separación entre clusters.
- ▶ Valores cercanos a 1 indican mejores agrupaciones.

## 3. Método de inconsistencia:

- ▶ Calcula el cambio en la distancia entre clusters conforme avanzan las fusiones.
- ▶ Si hay una gran inconsistencia en la fusión, indica un buen punto de corte.

### Ventajas y desventajas de esta técnica

#### Ventajas:

- ✓ No requiere especificar el número de clusters de antemano, a diferencia de K-Means.
- ✓ Proporciona una estructura jerárquica útil para interpretar relaciones entre datos.
- ✓ Puede manejar datos con formas de clusters no esféricos, a diferencia de K-Means.

#### Desventajas:

- ✗ Es computacionalmente costoso, especialmente con grandes volúmenes de datos.
- ✗ Sensibilidad a los valores atípicos, lo que puede afectar la formación de clusters.
- ✗ No permite re-asignación de puntos una vez que un dato ha sido asignado a un cluster.

### Implementación en Python

En el siguiente ejemplo implementaremos el **agrupamiento jerárquico** paso a paso en Python, utilizando la librería **scipy** para generar un dendrograma y **sklearn** para aplicar el clustering.

El objetivo es:

1. **Obtener un conjunto de datos** (usaremos el dataset *Iris*).
2. **Calcular la matriz de distancias** y representar los clusters en un dendrograma.
3. **Elegir el número de clusters** óptimo con base en el dendrograma.
4. **Aplicar el algoritmo de clustering jerárquico** para asignar cada punto a un cluster.
5. **Visualizar los resultados**.



Utilizaremos el **conjunto de datos Iris**, que es un dataset clásico de Machine Learning con características de flores. Usaremos solo dos columnas (sepal length y sepal width) para visualizar mejor los clusters.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import scipy.cluster.hierarchy as sch
from sklearn.cluster import AgglomerativeClustering
from sklearn import datasets

# 1 Cargar el dataset Iris
iris = datasets.load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)

# Usamos solo las dos primeras columnas para facilitar la visualización
data = df.iloc[:, [0, 1]].values # 'sepal length' y 'sepal width'
```

Ilustración 3 Primera parte del código de ejemplo

```
# 2 Construcción del dendrograma para analizar agrupaciones
plt.figure(figsize=(10, 5))
linkage_matrix = sch.linkage(data, method='ward') # 'ward' minimiza la varianza dentro de los clusters
sch.dendrogram(linkage_matrix)
plt.title("Dendrograma del Agrupamiento Jerárquico")
plt.xlabel("Puntos de datos")
plt.ylabel("Distancia")
plt.show()

# 3 Elección del número de clusters basado en el dendrograma
num_clusters = 3 # Se elige observando el dendrograma

# 4 Aplicación del clustering jerárquico
hc = AgglomerativeClustering(n_clusters=num_clusters, affinity='euclidean', linkage='ward')
y_clusters = hc.fit_predict(data)

# 5 Visualización de los clusters resultantes
plt.figure(figsize=(8, 5))
for i in range(num_clusters):
    plt.scatter(data[y_clusters == i, 0], data[y_clusters == i, 1], label=f'Cluster {i+1}')

plt.xlabel("Sepal Length (cm)")
plt.ylabel("Sepal Width (cm)")
plt.title("Agrupamiento Jerárquico - Clusters Encontrados")
plt.legend()
plt.show()

# 6 Mostrar los resultados en la terminal
print("Asignación de clusters para los primeros 10 datos:")
print(y_clusters[:10]) # Mostramos solo los primeros 10 resultados
```

Ilustración 4 Segunda parte del código de ejemplo

## ALGORITMO K MEANS: EN QUÉ CONSISTE

El **algoritmo K-Means** es uno de los métodos más utilizados para agrupamiento de datos en Machine Learning. Su objetivo es dividir un conjunto de datos en  $K$  grupos (clusters), donde cada dato pertenece al cluster con el centroide más cercano.

Es un **algoritmo no supervisado**, lo que significa que no necesita etiquetas previas en los datos. Se basa en una idea simple pero poderosa: minimizar la distancia entre los puntos y el centroide del cluster al que pertenecen.

### ¿Cómo funciona el algoritmo K-Means?

El algoritmo K-Means sigue estos pasos fundamentales:

#### 1. Elegir el número de clusters $K$

El usuario define cuántos clusters quiere encontrar en los datos. Elegir el valor correcto de  $K$  es clave (veremos métodos para esto más adelante).

#### 2. Inicializar $K$ centroides aleatorios

Se seleccionan  $K$  puntos aleatorios del conjunto de datos para ser los centroides iniciales. Estos puntos representan temporalmente el centro de cada cluster.

#### 3. Asignar cada punto de datos al cluster más cercano

Cada dato se asigna al cluster cuyo centroide esté más cerca, utilizando una métrica de distancia, generalmente la distancia euclídea.

#### 4. Recalcular los centroides

Una vez que todos los datos han sido asignados a un cluster, se calcula el nuevo centroide de cada cluster como el promedio de todos los puntos dentro de él.

#### 5. Repetir los pasos 3 y 4 hasta converger

Se repiten los pasos de asignación y actualización de centroides hasta que los centroides no cambien significativamente o se alcance un número máximo de iteraciones.

## Aplicaciones del Algoritmo K-Means

- ⇒ Agrupamiento de clientes en marketing.
- ⇒ Detección de anomalías en seguridad informática.
- ⇒ Compresión de imágenes.
- ⇒ Agrupación de documentos o noticias similares.

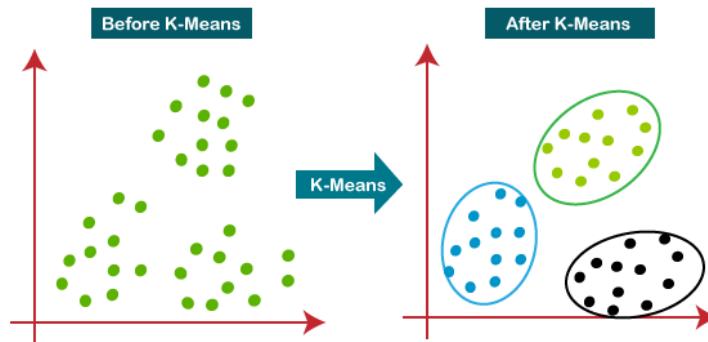


Ilustración 5 Ejemplo antes y después de K-Means. Lea Setruk

## Elección del valor de K

Elegir el número correcto de clusters ( $K$ ) es clave para un buen desempeño del algoritmo K-Means. Si elegimos un  $K$  muy pequeño, los clusters serán demasiado generales y no representarán bien los datos. Si  $K$  es demasiado grande, el modelo puede sobreajustarse y detectar clusters artificiales.

Para determinar el  $K$  óptimo, existen varios métodos. Aquí veremos los dos más utilizados:

### Método del codo (Elbow Method)

El método del codo se basa en calcular la **suma de los errores cuadrados (WCSS - Within-Cluster Sum of Squares)** para diferentes valores de  $K$ .

## ¿Qué es WCSS?

Es la suma de las distancias de cada punto dentro de un cluster a su centroide. Matemáticamente:

$$WCSS = \sum_{i=1}^K \sum_{x \in C_i} \|x - \mu_i\|^2$$

Ilustración 6 Operación Matemática

donde  $C_i$  es un cluster y  $\mu_i$  es su centroide.

## ¿Cómo funciona el método?

1. Se ejecuta K-Means con diferentes valores de  $K$ .
2. Se calcula el **WCSS** para cada  $K$ .
3. Se grafica el WCSS vs.  $K$ .
4. Se elige el "**punto de inflexión**", donde la disminución de WCSS se hace menos pronunciada.

## Coeficiente de silueta

El coeficiente de silueta mide qué tan bien separados están los clusters y qué tan coherentes son internamente.

## ¿Cómo se calcula?

Para cada punto, se calcula:

1.  $\alpha(i)$  = Distancia promedio entre el punto  $i$  y los demás puntos de su cluster.
2.  $b(i)$  = Distancia promedio entre el punto  $i$  y los puntos del cluster más cercano al que no pertenece.



### 3. Coeficiente de silueta:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Ilustración 7 Operación Matemática

- ⇒  $s(i)$  cercano a 1 → Punto bien agrupado.
- ⇒  $s(i)$  cercano a 0 → Punto en el límite entre clusters.
- ⇒  $s(i)$  negativo → Punto mal asignado.

#### Cómo interpretarlo:

Para diferentes valores de  $K$ , se calcula el coeficiente de silueta promedio. El  $K$  que maximiza esta métrica es el mejor.

#### Comparación de ambos métodos

Método	¿Qué Mide?	¿Cómo se usa?	Mejor $K$
<b>Codo</b>	Reducción de WCSS	Buscar el "punto de inflexión"	Donde el WCSS deja de disminuir rápidamente
<b>Silueta</b>	Separación y compactación de clusters	Elegir $K$ donde el coeficiente de silueta es mayor	Donde la silueta es más alta



## Ventajas y desventajas de utilizar el algoritmo

Cómo cualquier método, el algoritmo K-Means tiene ventajas y desventajas que veremos a continuación:

### Ventajas de K-Means

#### ► Eficiencia computacional

- Es mucho más rápido que otros algoritmos de clustering como **DBSCAN** o el agrupamiento jerárquico.
- Su complejidad es aproximadamente  $O(n \cdot k \cdot i)$ , donde  $n$  es el número de puntos,  $k$  la cantidad de clusters y  $i$  el número de iteraciones.

#### ► Fácil de entender e implementar

- Su funcionamiento es intuitivo: asigna puntos a un cluster y recalcula centroides iterativamente.
- Está disponible en casi todas las bibliotecas de Machine Learning.

#### ► Escalabilidad

- Funciona bien con grandes volúmenes de datos.
- Puede manejar millones de muestras sin perder eficiencia.

#### ► Funciona bien en datos con clusters bien definidos

- Si los clusters tienen una forma esférica o compacta, K-Means los detecta con alta precisión.

### Desventajas de K-Means

#### ► Sensibilidad al número de clusters $K$

- Se debe elegir manualmente el valor de  $K$ , lo que puede ser difícil sin una técnica adecuada (como el método del codo o el coeficiente de silueta).

- 
- ▶ **No maneja bien clusters de diferentes formas y densidades**
    - K-Means asume que los clusters son esféricos y tienen tamaños similares.
    - No es ideal para datasets donde los clusters tienen formas irregulares o densidades muy distintas.
  - ▶ **Sensibilidad a valores atípicos (outliers)**
    - Un solo outlier puede afectar el cálculo del centroide, lo que provoca clusters incorrectos.
    - Métodos como **DBSCAN** son más resistentes a outliers.
  - ▶ **No garantiza el óptimo global**
    - K-Means depende de la inicialización de los centroides y puede quedar atrapado en un mínimo local.
    - Para mitigar esto, se suele ejecutar varias veces con diferentes inicializaciones (**n\_init** en Scikit-Learn).
  - ▶ **No detecta ruido o puntos sin cluster**
    - Cada punto **debe pertenecer** a un cluster, lo que puede ser un problema en datos con ruido.
    - DBSCAN, en cambio, puede dejar puntos sin asignar si no cumplen ciertos criterios.

## Implementación en Python

En el siguiente ejemplo, implementaremos el algoritmo K-Means paso a paso.

### ¿Qué hará este código?

- Cargaremos un conjunto de datos (Iris) y seleccionaremos solo dos características para poder visualizar los clusters.
- Aplicaremos el **algoritmo K-Means** con un valor de  $K$  elegido previamente.
- Graficaremos los clusters resultantes y sus centroides.



Usaremos el conjunto de datos **Iris**, disponible en la librería **sklearn.datasets**. Este dataset contiene información sobre tres tipos de flores (**Setosa**, **Versicolor**, **Virginica**) y sus características (largo del pétalo, ancho del pétalo, etc.).

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn import datasets

# Cargar el dataset Iris
iris = datasets.load_iris()
X = iris.data[:, :2] # Usamos solo las dos primeras características para visualizar mejor

# Aplicar K-Means con K=3 (porque sabemos que hay 3 clases en Iris)
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
labels = kmeans.fit_predict(X) # Asigna cada punto a un cluster
centroids = kmeans.cluster_centers_ # Obtiene los centros

# Graficar los puntos y los clusters
plt.figure(figsize=(8, 5))
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', marker='o', edgecolor='k', label="Datos")
plt.scatter(centroids[:, 0], centroids[:, 1], c='red', marker='X', s=200, label="Centroídes")
plt.xlabel("Longitud del Sépalo")
plt.ylabel("Ancho del Sépalo")
plt.title("Clustering K-Means en Iris (K=3)")
plt.legend()
plt.show()
```

Ilustración 8 Código ejemplo

## ALGORITMO DBSCAN: EN QUÉ CONSISTE EL ALGORITMO DBSCAN

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** es un **algoritmo de clustering basado en densidad**, ideal para identificar estructuras en conjuntos de datos con formas irregulares y para detectar **outliers** (datos ruidosos).

### ¿En qué consiste el algoritmo DBSCAN?

DBSCAN clasifica los puntos en tres categorías principales:

1. **Puntos centrales:** Tienen suficientes vecinos cercanos y forman la estructura del cluster.
2. **Puntos frontera:** Están cerca de un punto central, pero no tienen suficientes vecinos para ser centrales.
3. **Ruido (outliers):** No tienen suficientes vecinos cercanos y no pertenecen a ningún cluster.

## Parámetros del Algoritmo

DBSCAN utiliza dos parámetros clave:

- **$\epsilon$  (eps - radio de vecindad)**
- Es la distancia máxima en la que un punto es considerado vecino de otro.
- Valores pequeños: Generan muchos clusters pequeños.
- Valores grandes: Unifican clusters que podrían ser diferentes.
- **MinPts (mínima cantidad de puntos en una región densa)**
- Un valor recomendado es  $MinPts \geq 2 \times dim$ , donde  $dim$  es la cantidad de características del dataset.
- Valores pequeños: Generan más clusters y detectan ruido con mayor precisión.
- Valores grandes: Agrupan menos puntos y pueden etiquetar demasiados puntos como ruido.

## Funcionamiento de DBSCAN

1. Se elige un punto aleatorio no visitado.
2. Se verifica cuántos puntos están en su vecindad (definida por  $\epsilon$ ).
3. Si hay suficientes puntos vecinos (**mínimo min\_samples**), se forma un cluster.
4. Los puntos dentro del radio  $\epsilon$  se agregan al cluster y el proceso continúa recursivamente.
5. Si un punto no tiene suficientes vecinos, se clasifica como **ruido (outlier)**.
6. El proceso se repite hasta que todos los puntos sean clasificados.

**Importante:** DBSCAN **no requiere especificar la cantidad de clusters** como K-Means, ya que los clusters emergen de la estructura de los datos.



## Parámetros del algoritmo

DBSCAN tiene dos parámetros principales:

1.  **$\epsilon$  (epsilon):** Define la distancia máxima entre dos puntos para considerarlos vecinos.
2. **min\_samples:** Cantidad mínima de puntos dentro del radio  $\epsilon$  para que un punto sea considerado **central**.

El ajuste correcto de estos parámetros es clave para un clustering eficiente.

### Elección del valor de $\epsilon$ (Epsilon)

El valor de  $\epsilon$  determina cuán cerca deben estar los puntos para ser considerados del mismo cluster.

#### Método para elegir $\epsilon$ :

##### 1. Usar la gráfica K-Distance:

- Se calcula la distancia de cada punto a su  $k$ -ésimo vecino más cercano.
- Se ordenan las distancias en orden ascendente.
- Se busca un "codo" en la curva para encontrar un valor adecuado de  $\epsilon$ .

##### 2. Elegir un $\epsilon$ razonable basado en el contexto de los datos.

- Valores muy pequeños generan muchos outliers.
- Valores muy grandes unen clusters diferentes.

Un buen punto de partida es probar con  $\epsilon$  cercano al valor donde hay un cambio brusco en la curva K-Distance.

## Ventajas y desventajas de utilizar el algoritmo

Este algoritmo, igual que los otros métodos, tiene ventajas y desventajas que podremos ver a continuación:

## Ventajas

- ✓ No requiere especificar el número de clusters  $K$ .
- ✓ Identifica clusters de diferentes formas y densidades.
- ✓ Detecta puntos ruido y outliers automáticamente.
- ✓ Robusto ante valores atípicos.

## Desventajas

- ✗ Sensibilidad a los parámetros  $\varepsilon$  y  $MinPts$ .
- ✗ Manejo ineficiente de datos con variaciones en densidad.
- ✗ Requiere cálculos de distancia entre todos los puntos → Costoso en grandes datasets.

## Implementación en Python

### ¿Qué hará este código?

- Generaremos datos de prueba con **make\_moons()**.
- Aplicaremos **DBSCAN** con diferentes valores de **eps**.
- Compararemos los clusters resultantes con el algoritmo **K-Means**.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN, KMeans
from sklearn.datasets import make_moons
from sklearn.preprocessing import StandardScaler

# 1 Generar un conjunto de datos con forma de luna creciente
X, _ = make_moons(n_samples=300, noise=0.1, random_state=42)
X = StandardScaler().fit_transform(X) # Normalizar datos para mejorar resultados
```

Ilustración 9 Primera parte código ejemplo

```

# 2 Aplicar DBSCAN con eps=0.3 y min_samples=5
dbSCAN = DBSCAN(eps=0.3, min_samples=5)
labels_dbSCAN = dbSCAN.fit_predict(X)

# 3 Aplicar K-Means con K=2 para comparar resultados
kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)
labels_kmeans = kmeans.fit_predict(X)

# 4 Graficar los resultados
fig, axs = plt.subplots(1, 2, figsize=(12, 5))

# Gráfico DBSCAN
axs[0].scatter(X[:, 0], X[:, 1], c=labels_dbSCAN, cmap='viridis', marker='o', edgecolor='k')
axs[0].set_title("Clustering con DBSCAN")

# Gráfico K-Means
axs[1].scatter(X[:, 0], X[:, 1], c=labels_kmeans, cmap='viridis', marker='o', edgecolor='k')
axs[1].set_title("Clustering con K-Means")

plt.show()

```

*Ilustración 10 Segunda parte código ejemplo*

#### Salida esperada:

- ▶ **Gráfico 1 (DBSCAN):** Clusters de diferentes **formas** detectados correctamente.
- ▶ **Gráfico 2 (K-Means):** Clusters **forzados a ser esféricos**, no representando bien la estructura real.