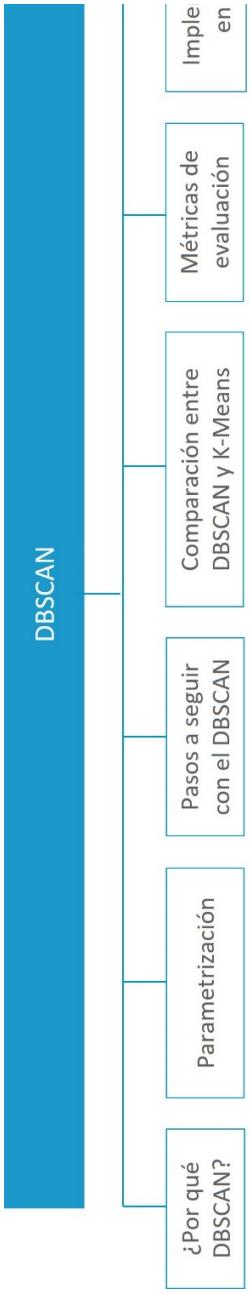
Aprendizaje Automático No Supervisado

# Tema 5. Clustering basado en densidad DBSCAN

# Índice

# Esquema



## 5.1. Introducción y objetivos

El **algoritmo DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) es una técnica de agrupamiento no supervisada ampliamente utilizada en el campo de la inteligencia artificial y el análisis de datos. Desarrollado por Martin Ester, Hans-Peter Kriegel, Jörg Sander y Xiaowei Xu en 1996. DBSCAN se destaca por su capacidad para identificar clústeres de forma arbitraria en conjuntos de datos que contienen ruido y distribuciones no lineales.

A diferencia de otros algoritmos de agrupamiento, como K-Means, que requieren que el usuario especifique el número de clústeres, DBSCAN trabaja identificando regiones densas de puntos y expandiéndose a partir de estos. Utiliza dos parámetros principales: ε (épsilon), que define el radio de vecindad de un punto, y MinPts, que determina el número mínimo de puntos necesarios para formar un clúster. Un punto se considera parte de un clúster si al menos MinPts puntos caen dentro de su radio ε. Si un punto no cumple con este criterio y no pertenece a ningún clúster, se clasifica como ruido.

DBSCAN es particularmente útil en aplicaciones donde los datos tienen estructuras complejas y no se conoce de antemano el número de clústeres. Además, su capacidad para manejar ruido y su eficiencia con grandes conjuntos de datos lo hacen ideal para tareas de minería de datos, reconocimiento de patrones y detección de anomalías.

En este tema, exploraremos en detalle cómo funciona DBSCAN, sus ventajas y desventajas, y presentaremos casos de uso prácticos que demuestran su aplicabilidad en el mundo real.

Como **objetivos** de este tema nos planteamos:

- Explicar en detalle los conceptos y mecanismos detrás del algoritmo DBSCAN,

incluyendo la definición de los parámetros ε y MinPts, y cómo estos afectan el proceso de agrupamiento.

- Analizar las ventajas y limitaciones de DBSCAN en comparación con otros algoritmos de agrupamiento, destacando su capacidad para manejar ruido y su flexibilidad en la identificación de clústeres de formas arbitrarias.

## 5.2. ¿Por qué DBSCAN?

Los métodos basados en K-Means y agrupación jerárquica encuentran grupos que tienen forma esférica o convexa; es decir, solamente son adecuados para conjuntos de datos cuyas clases están bien separadas. Otro problema de los algoritmos más convencionales es que se ven muy afectados por la presencia de ruido o por valores atípicos.

DBSCAN puede encontrar clústeres de forma arbitraria y compleja. Esto es gracias a que se basa en el cálculo de la densidad de los puntos para formar un grupo. Además, supera uno de los grandes obstáculos de los algoritmos que hemos visto y es que no es necesario especificar o conocer de antemano el número de clústeres. En su lugar, utiliza dos parámetros  (épsilon), que define la distancia máxima entre dos puntos para considerarlos vecinos, y MinPts, que es el número mínimo requerido para formar un clúster.

En la vida real los datos tienen muchas irregularidades, los grupos tienen diferentes formas, contienen ruido y los valores atípicos son muy comunes. Basándose en la idea de que un grupo en el espacio de datos es una región contigua de alta densidad de puntos, separada de otros grupos similares por regiones contiguas de baja densidad de puntos, DBSCAN tiene la capacidad de identificar puntos de datos que no pertenecen a ningún clúster, y los etiqueta fácilmente como posible ruido.

Otra característica del DBSCAN es su eficiencia en conjuntos de grandes cantidades de datos, utilizando sus variantes KD-Trees o R-Trees para buscar vecinos de manera eficiente.

## 5.3. Parametrización

Para utilizar DBSCAN es importante entender y ajustar sus dos parámetros clave:

- **Épsilon (ϵ):** es la distancia máxima entre dos puntos para que se consideren vecinos el uno del otro. Es decir, si la distancia entre dos puntos es menor o igual a ϵ se consideran vecinos. Si este valor es muy pequeño, una gran parte de los datos se considerarán como valor atípico. En cambio, si el valor es muy grande, los grupos se fusionarán y la mayoría de los puntos de datos estarán en los mismos grupos.

- **Mínimo número de puntos:** el número mínimo de puntos que debe formar un grupo denso. Como regla general, los mínimos se pueden derivar del número de dimensiones D en el conjunto de datos como mínimo número de puntos >= D+1. El valor debe ser elegido, y como sugerencia es que sea 3 como mínimo.

En este algoritmo tenemos tres tipos de puntos de datos:

- **Punto central:** es considerado punto central si tiene más puntos MinPts dentro de los eps (ϵ).

- **Punto fronterizo:** un punto que tiene menos de MinPts dentro de los eps (ϵ), pero que está en las proximidades de un punto central.

- **Ruido o valor atípico:** un punto que no es un punto central o un punto fronterizo.

Estos parámetros se pueden ajustar utilizando prueba y error o a través de la gráfica del codo. También se puede usar la matriz de distancia, de esta forma se pueden estimar valores razonables.

## 5.4. Pasos para seguir con el DBSCAN

Los pasos que sigue el algoritmo son los siguientes:

- Encuentra todos los puntos vecinos dentro de eps (ϵ) e identifica los puntos centrales o visitados con más vecinos teniendo en cuenta el número.
- Para cada punto central, si aún no le han asignado grupo, se crea un nuevo grupo.
- De manera recursiva encuentra todos los puntos conectados por densidad y asígnalos al mismo grupo que el punto central. Un punto a y b están conectados por densidad si existe un punto c que tiene un número suficiente de puntos en sus vecinos y ambos puntos a y b están dentro de la distancia ϵ. Es un proceso de encadenamiento; entonces, si a es vecino de c, c es vecino de d y d es vecino de e, lo que a su vez indica que d es vecino de a, además implica que b es vecino de a.
- Repetir con todos los puntos restantes no visitados en el conjunto de datos. Aquellos puntos que no pertenecen a ningún grupo son considerados ruido o *outliers.*

**Pseudocódigo** (GeeksforGeeks, 2023):

```
DBSCAN(conjunto de datos, eps, MinPts){
# índice de clúster
C = 1
para cada punto no visitado p en el conjunto de datos {
         marcar p como visitado
         # encontrar vecinos
         Vecinos N = encontrar los puntos vecinos de p

         si |N|>=Ptos Mínimos:
             N = NU N'
             si p' no es miembro de ningún grupo:
                 agregue p' al grupo C
```

Veamos un ejemplo de aplicación utilizando datos de peso y talla:

Asumamos que hemos recopilado datos de peso y talla de un grupo de personas:

| | Peso | Talla |
|---|---|---|
| Dato1 | 87 | 188 |
| Dato2 | 74 | 156 |
| Dato3 | 79 | 157 |
| - | - | - |

Tabla 1. Recopilación de datos. Fuente: elaboración propia.

Dibujamos los puntos originales (Figura 1).

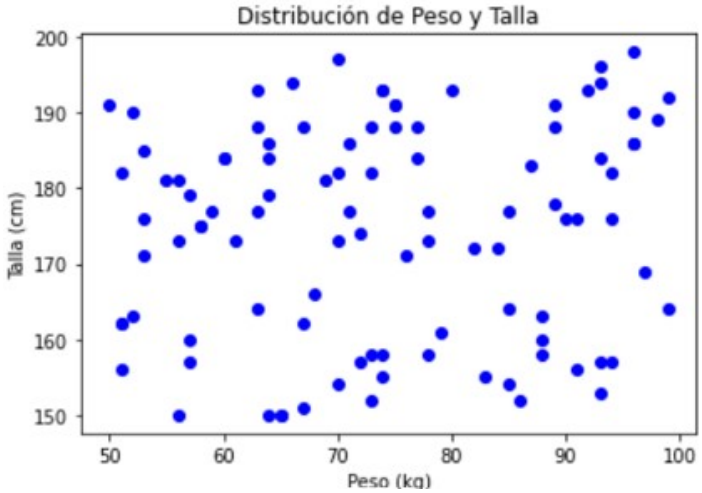



Figura 1. Puntos originales de peso y talla. Fuente: elaboración propia.

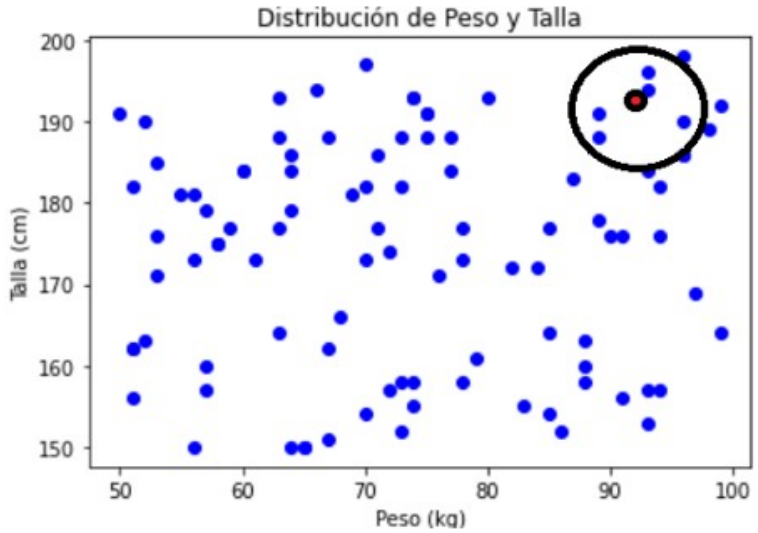



Figura 2. Identificación de un punto y sus vecinos. Fuente: elaboración propia.

El punto rojo está cerca de otros seis puntos (Figura 2).

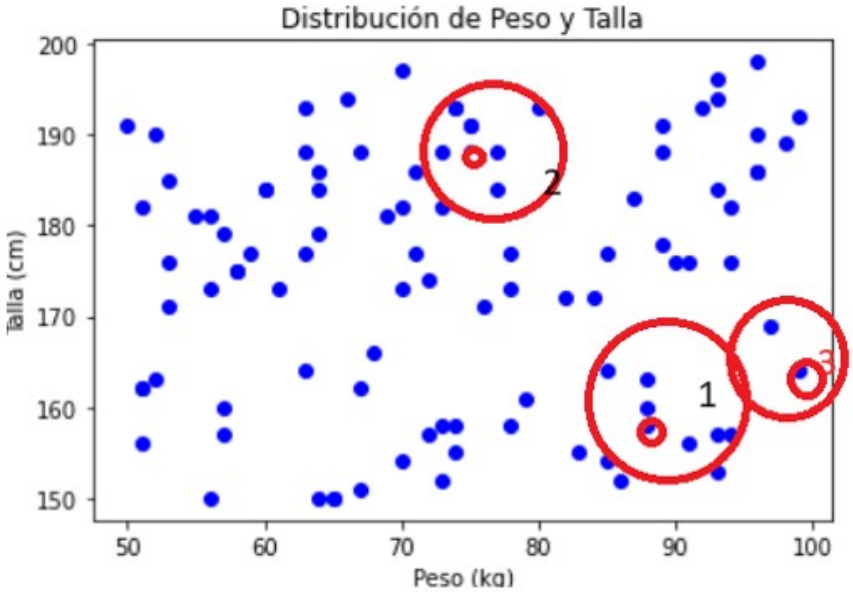



Figura 3. Identificación de varios puntos y sus vecinos. Fuente: elaboración propia.

En la Figura 3 vemos dos puntos que tienen seis puntos cerca y un punto (el número 3) está cerca solo de 1 punto y no hay otros puntos en sus proximidades. Así se

repite el proceso para cada uno de los puntos del conjunto de datos.

En este ejemplo estamos considerando que un punto central debe tener cerca al menos otros cinco puntos. Sin embargo, debemos tener en cuenta que el número de puntos cercanos para un punto central está definido por el usuario.

Para el ejemplo de la Figura 3, los puntos 1 y 2 son puntos centrales, pero el punto 3 no lo es, no cumple la condición.

Elegimos aleatoriamente un punto central y lo asignamos al primer grupo como se puede observar en la Figura 4.

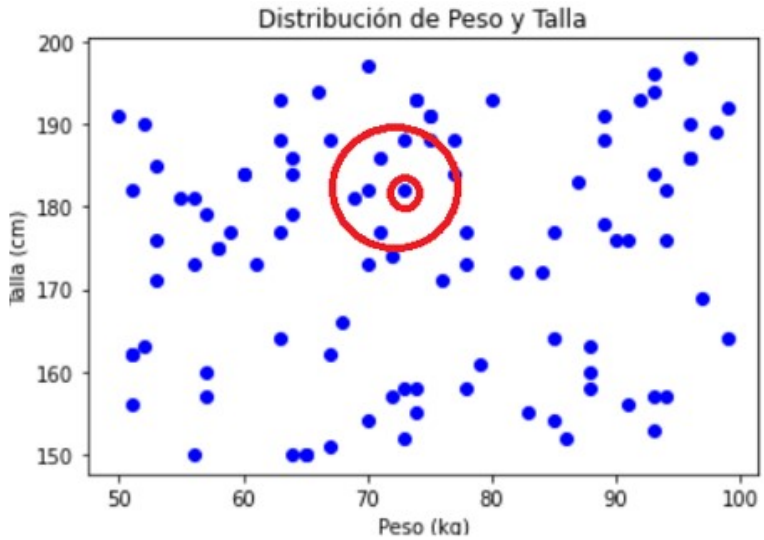



Figura 4. Empezamos a crear un clúster a partir de la elección aleatoria de un punto. Fuente: elaboración propia.

Luego todos los puntos centrales que están cerca de este primer grupo creado se unen a él y lo extienden a los otros puntos centrales que están cerca.

Existen puntos cercanos a puntos centrales, pero no cumplen la condición de tener un número de puntos cercanos. Dichos puntos no pueden unirse al grupo; haciendo que el grupo no pueda extenderse más.

Veamos el código y el resultado de utilizar DBSCAN en este ejemplo:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generar datos aleatorios
np.random.seed(42) # Fijar la semilla para reproducibilidad
peso = np.random.randint(50, 100, 100)
talla = np.random.randint(150, 200, 100)

# Crear un conjunto de datos combinando peso y talla
X = np.column_stack((peso, talla))

# Escalar los datos
X = StandardScaler().fit_transform(X)

# Aplicar DBSCAN
db = DBSCAN(eps=0.5, min_samples=5).fit(X)
labels = db.labels_

# Número de clusters en los datos, ignorando el ruido si está presente
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)

# Graficar los resultados
unique_labels = set(labels)
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1,
len(unique_labels))]

# Graficar cada cluster con colores diferentes
for k, col in zip(unique_labels, colors):
    if k == -1:
        # El color negro es para el ruido.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

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

plt.title(f'Número estimado de clusters: {n_clusters_}')
plt.xlabel('Peso (escalado)')
plt.ylabel('Talla (escalado)')
plt.show()
```

Finalmente, todos los puntos centrales se han asignado a un grupo. Hemos terminado de crear nuevos grupos (Figura 5).

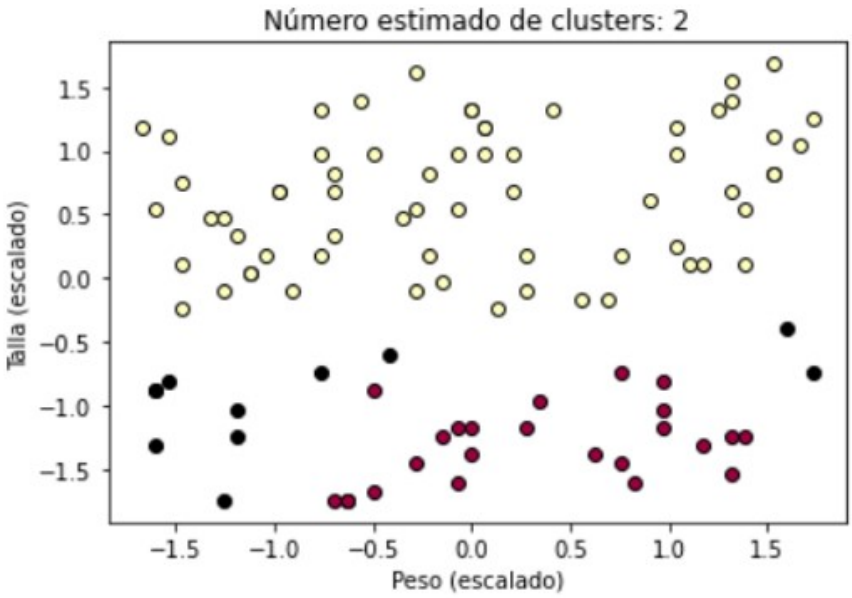



Figura 5. Resultado final al aplicar DBSCAN a los datos de peso y talla. Fuente: elaboración propia.

Al utilizar DBSCAN, podemos agrupar los datos de manera más precisa y eficiente. El DBSCAN puede identificar clústeres anidados en grandes dimensiones. DBSCAN imita la identificación de grupos por las densidades de los puntos en los que se encuentran los grupos, mientras que los valores atípicos tienden a estar en áreas de baja densidad.

## 5.5. Métricas de evaluación DBSCAN

Veamos dos métricas que nos permiten evaluar y comparar algoritmos de *clustering:* índice Rand y la puntuación silueta.

### Índice Rand

Este índice mide la similitud entre dos agrupaciones (clústeres) de un conjunto de datos: una agrupación «predicha» y una agrupación de referencia que puede ser una agrupación hecha por otro algoritmo de *clustering.*

El índice Rand se basa en la idea de contar las decisiones de pares correctas y las incorrectas hechas por el algoritmo de *clustering.* Se consideran las siguientes cuatro cantidades:

- a: número de pares de puntos que están en el mismo clúster en ambos agrupamientos (predicho y real).
- b: número de pares de puntos que están en clústeres diferentes en ambos agrupamientos.
- c: número de pares de puntos que están en el mismo clúster en el agrupamiento real, pero en clústeres diferentes en el agrupamiento predicho.
- d: número de pares de puntos que están en el mismo clúster en el agrupamiento predicho, pero en clústeres diferentes en el agrupamiento real.

El índice Rand se calcula usando la siguiente fórmula:

$$índiceRand = \frac{a+b}{a+b+c+d}$$

Donde:

- a+ba + ba+b es el número total de acuerdos (pares que están en el mismo clúster

en ambos agrupamientos o en diferentes clústeres en ambos agrupamientos).

- a+b+c+da + b + c + da+b+c+d es el número total de pares de puntos en el conjunto de datos.

Interpretación del índice Rand:

- **Valor de 1:** indica una coincidencia perfecta entre los dos agrupamientos.
- **Valor de 0:** indica que no hay acuerdo entre los agrupamientos (esto es teóricamente posible, pero improbable).
- **Valor cercano a 1:** indica un alto grado de similitud entre los agrupamientos.
- **Valor cercano a 0.5:** indica que la similitud entre los agrupamientos es similar a la que se esperaría al azar.

## Puntuación silueta

La puntuación de silueta mide qué tan similar es un punto a los puntos de su propio clúster (cohesión) en comparación con los puntos del clúster más cercano (separación). Se define para cada punto y luego se puede promediar para todos los puntos en el conjunto de datos.

**Cálculo de la puntuación de silueta**

Para un punto i:

- **Cohesión ( $a_i$ ):** la distancia promedio entre el punto i y todos los demás puntos en el mismo clúster.
- **Separación ( $b_i$ ):** la distancia promedio entre el punto i y todos los puntos en el clúster más cercano diferente del suyo.

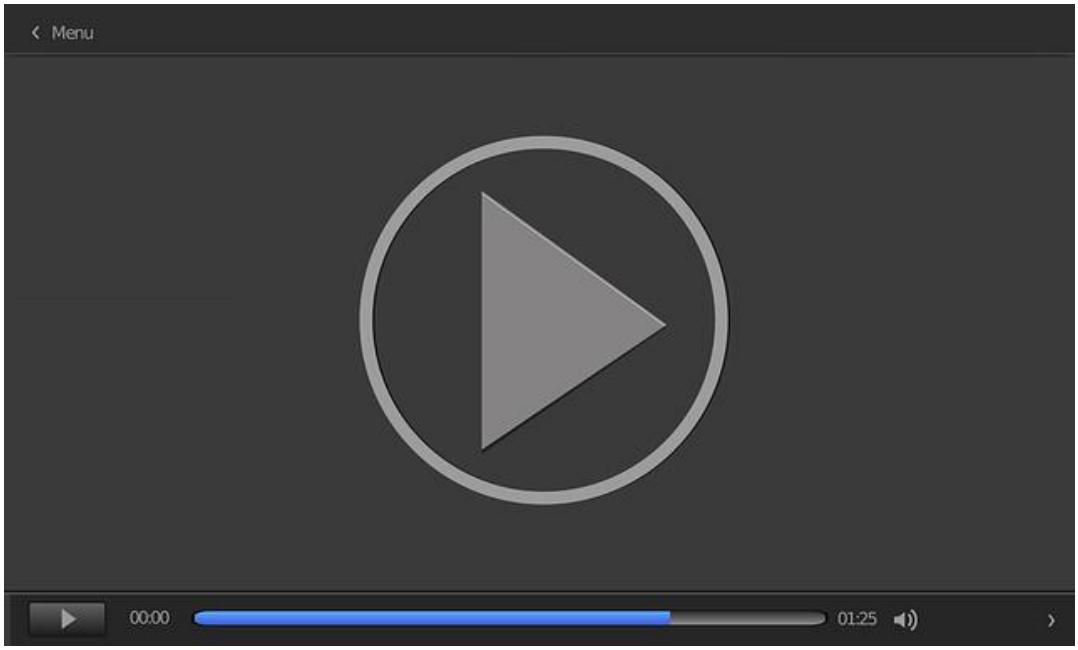
La puntuación de silueta para el punto i se define como:

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

Donde:

- $s(i)$ puede variar entre -1 y 1.
- Un valor cercano a 1 indica que el punto está bien agrupado.
- Un valor cercano a -1 indica que el punto está mal agrupado.
- Un valor cercano a 0 indica que el punto está en el límite entre dos clústeres.

A continuación, veremos el vídeo ***Comparación de dos algoritmos de clustering utilizando el índice Rand y puntuación silueta.***




Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=906d592f-2449-4f71-aacf-b1be01528d68

## 5.6. Implementación en Python

El DBSCAN utiliza el KD-Tree y el R-Tree para optimizar la búsqueda de clústeres. Vamos a ver cómo construir e implementar un KD-Tree, su aplicación en el algoritmo DBSCAN para mejorar el rendimiento de la búsqueda de puntos centrales y su implementación en Python.

### KD-Tree

Un **KD-Tree** (K-Dimensional Tree) es una estructura de datos en forma de árbol utilizada para organizar puntos en un espacio k-dimensional. Es particularmente útil para operaciones como la búsqueda de vecinos más cercanos y la agrupación de datos en algoritmos de aprendizaje automático y minería de datos.

**Estructura del árbol**

Un KD-Tree es un árbol binario en el que cada nodo representa una partición de los puntos de datos en el espacio k-dimensional.

Cada nodo tiene una «dimensión de corte» asociada y divide los puntos según su valor en esa dimensión.

**Construcción del KD-Tree**

- La construcción del KD-Tree es un proceso recursivo.
- Comenzamos con todos los puntos y elegimos una dimensión para dividirlos.
- Encontramos el punto mediano en esa dimensión y usamos ese punto para dividir el conjunto de datos en dos subconjuntos: uno con puntos a la izquierda del mediano y otro con puntos a la derecha.
- Recursivamente repetimos el proceso para cada subconjunto, alternando las dimensiones en cada nivel del árbol (primera dimensión, segunda dimensión, ..., k-

ésima dimensión, y luego volvemos a la primera dimensión).

**Búsqueda en el KD-Tree**

Para buscar los vecinos más cercanos de un punto dado, navegamos el árbol comparando el punto con los nodos del árbol y decidiendo si ir a la izquierda o a la derecha.

Podemos podar (descartar) ramas del árbol que no pueden contener un vecino más cercano que el mejor encontrado hasta ahora, reduciendo así el número de comparaciones necesarias.

Supongamos que tenemos un conjunto de puntos en un espacio 2D (x, y):

- **Primera división (dimensión x):**
  - Ordenamos los puntos por la coordenada x.
  - Elegimos el punto mediano (mediano en x) como la raíz del árbol.
  - Los puntos a la izquierda del mediano se convertirán en el subárbol izquierdo y los puntos a la derecha en el subárbol derecho.
- **Segunda división (dimensión y):**
  - En el nivel siguiente, ordenamos los puntos en cada subárbol por la coordenada y.
  - Elegimos el punto mediano en y para dividir nuevamente los puntos en subárboles izquierdo y derecho.
- **Continuamos recursivamente:**
  - Alternamos las dimensiones x y y en cada nivel del árbol.

> Si tenemos los siguientes puntos: (3, 6), (17, 15), (13, 15), (6, 12), (9, 1), (2, 7), (10, 19).

Ordenamos los puntos teniendo en cuenta la coordenada X: (2, 7), (3, 6), (6, 12), (9, 1), (10, 19), (13, 15), (17, 15).

**Primera división (x):**

Mediana en x: (9, 1).

Raíz: (9, 1).

Izquierda: (3, 6), (2, 7), (6, 12)

Derecha: (10, 19), (13, 15), (17, 15).

Ordenamos los puntos teniendo en cuenta la coordenada Y del subárbol izquierdo que obtuvimos con la coordenada x: (3, 6), (2, 7), (6, 12).

**Primera división (y):**

Mediana en x: (2,7).

Raíz: (2, 7).

Izquierda: (3, 6)

Derecha: (6,12).

Y así sucesivamente alternando las dimensiones x y y en cada nivel del árbol.

**Ventajas del KD-Tree**

- **Eficiencia en búsquedas:** reduce el número de comparaciones necesarias para buscar puntos cercanos.

**Desventajas**

- **Desequilibrio:** si los datos no están distribuidos uniformemente, el árbol puede volverse desequilibrado, afectando el rendimiento.
- **Actualizaciones:** insertar o eliminar puntos puede requerir la reconstrucción del árbol.

Una de las partes críticas del algoritmo DBSCAN es la consulta de vecindad: determinar los puntos que están dentro de un cierto radio (eps) de un punto dado. Este proceso puede ser computacionalmente costoso, especialmente para grandes conjuntos de datos, debido a la necesidad de calcular distancias entre muchos pares de puntos.

El Kd-Tree se puede utilizar para acelerar estas consultas de vecindad. En lugar de comparar cada punto con todos los demás para encontrar vecinos dentro del radio eps, se puede utilizar el Kd-Tree para reducir significativamente el número de comparaciones necesarias.

## R-Tree

E l **R-Tree** es una estructura de datos en forma de árbol utilizada para indexar información espacial o multidimensional. Es particularmente útil para realizar consultas espaciales eficientes, como encontrar todos los objetos dentro de una región dada. A diferencia de otros árboles, como el KD-Tree, el R-Tree está diseñado para manejar datos que no solo se representan como puntos, sino también como objetos de mayor dimensión (rectángulos, polígonos, etc.).

### Estructura del árbol

Un R-Tree está compuesto por nodos que contienen un número variable de entradas.

Cada entrada en un nodo no hoja es un par formado por un identificador de nodo hijo y el MBR *(Minimum Bounding Rectangle)* de todos los objetos contenidos en ese nodo hijo.

Las hojas del árbol contienen las entradas reales, que son los MBR de los objetos almacenados.

***Minimum Bounding Rectangle* (MBR)**

Un MBR es el rectángulo más pequeño que puede contener un conjunto de objetos en el espacio.

Cada nodo del R-Tree almacena el MBR de sus hijos, proporcionando una jerarquía de MBRs que permite la indexación eficiente.

**Construcción**

- Los datos se insertan en el R-Tree uno a uno.
- Durante la inserción, se elige la hoja que requerirá la menor expansión de su MBR para incluir el nuevo objeto.
- Si una hoja excede su capacidad, se divide en dos nuevas hojas, y se ajustan los MBRs de los nodos padre.

**Consultas**

Las consultas pueden buscar todos los objetos dentro de un MBR dado o encontrar el vecino más cercano.

La estructura del árbol permite que muchas ramas se podan durante la búsqueda, reduciendo significativamente el número de comparaciones.

**Ventajas**

- **Eficiencia en búsquedas espaciales:** el R-Tree permite realizar búsquedas espaciales de manera eficiente, reduciendo el número de comparaciones necesarias.
- A diferencia del KD-Tree, el R-Tree puede manejar objetos de diferentes tamaños y

formas, no solo puntos.

**Desventajas**

- La implementación y ajuste del R-Tree pueden ser más complejos que otras estructuras de datos.
- En ciertos casos, el rendimiento puede verse afectado si los MBRs de los nodos superiores tienen mucha superposición.

## ¿Cómo funciona el KD-Tree y el R-Tree con DBSCAN?

- Construimos un KD-Tree o un R-Tree a partir del conjunto de datos. Esto organiza los puntos de manera que las consultas espaciales sean más eficientes.
- Para cada punto en el conjunto de datos, en lugar de calcular la distancia a todos los otros puntos, utilizamos el árbol generado por el KD-Tree o el R-Tree para encontrar rápidamente todos los puntos dentro del radio eps.
- El KD-Tree y R-Tree permiten búsquedas eficientes en el espacio, lo que reduce el número de comparaciones y, por lo tanto, acelera el proceso de encontrar vecinos.
- Una vez que tenemos los vecinos, el algoritmo DBSCAN puede clasificar el punto actual como parte de un clúster, expandir el clúster o marcar el punto como ruido, según corresponda.

Veamos un ejemplo de implementación en Python con KD-Tree:

```
import numpy as np
from sklearn.cluster import DBSCAN
from sklearn.neighbors import KDTree
import matplotlib.pyplot as plt

# Generamos datos de ejemplo
np.random.seed(0)
n_points_per_cluster = 250
C1 = [-5, -2] + .8 * np.random.randn(n_points_per_cluster, 2)
C2 = [4, -1] + .1 * np.random.randn(n_points_per_cluster, 2)
```

```
C3 = [1, -2] + .2 * np.random.randn(n_points_per_cluster, 2)
C4 = [-2, 3] + .3 * np.random.randn(n_points_per_cluster, 2)
X = np.vstack((C1, C2, C3, C4))

# Creamos un árbol KD
tree = KDTree(X, leaf_size=30)

# Configuramos DBSCAN con los parámetros deseados y el árbol KD
db = DBSCAN(eps=0.3, min_samples=10, algorithm='kd_tree', leaf_size=30)
db.fit(X)

# Obtenemos las etiquetas de los clusters
labels = db.labels_

# Número de clusters en los datos, ignorando ruido si hay (-1).
n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
n_noise = list(labels).count(-1)

print(f'Número estimado de clusters: {n_clusters}')
print(f'Número estimado de puntos de ruido: {n_noise}')

# Visualizamos los resultados
unique_labels = set(labels)
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:
        # Negro se utiliza para ruido en la visualización.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask &amp;amp; ~np.isnan(X).any(axis=1)]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title(f'Número estimado de clusters: {n_clusters}')
plt.show()
```

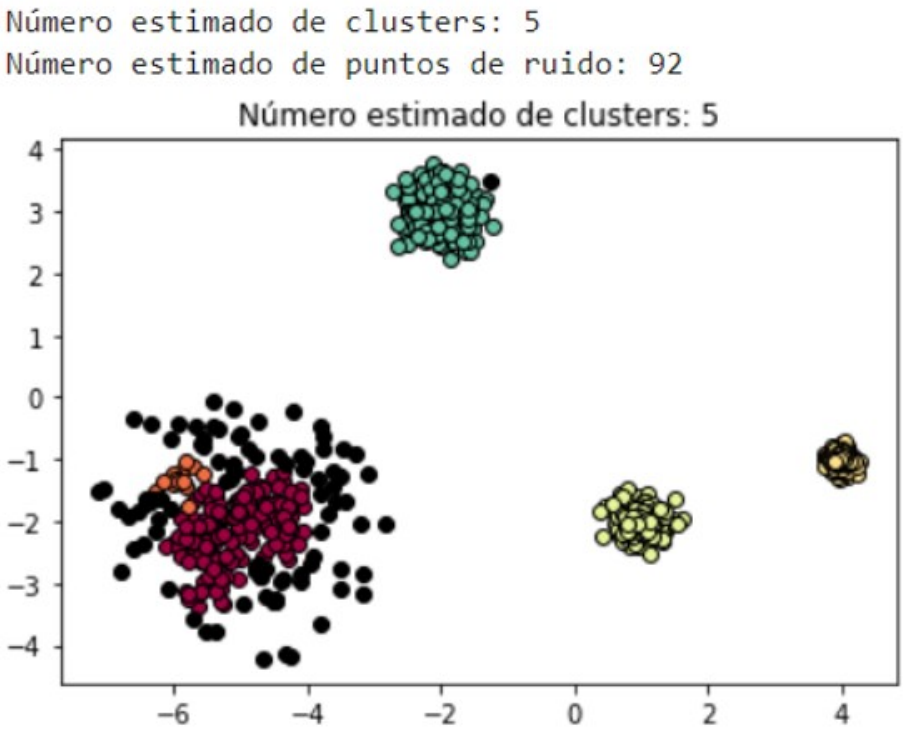



Figura 6. Resultado final al aplicar DBSCAN utilizando KD-Tree. Fuente: elaboración propia.

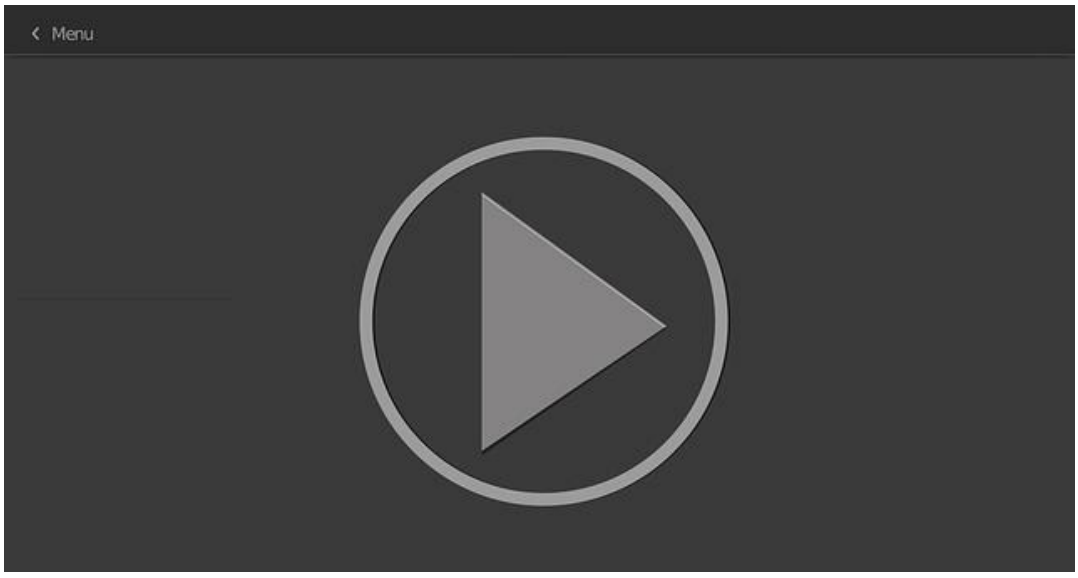
## 5.7. Comparación entre DBSCAN y K-Means

En la Tabla 2 vemos las diferencias entre DBSCAN y K-Means:

| DBSCAN | K-Means |
|---|---|
| En DBSCAN no necesitamos especificar el número de clústeres. | K-Means es muy sensible al número de grupos, por lo que necesita ser definido inicialmente. |
| Los grupos formados en DBSCAN pueden tener cualquier forma. | Los grupos formados en K-Means son esféricos o de forma convexa. |
| DBSCAN puede funcionar bien con conjuntos de datos que tienen ruido y valores atípicos. | K-Means no funciona bien con datos atípicos. Valores atípicos puede sesgar en gran medida los grupos en K-Means. |
| En DBSCAN se requieren dos parámetros para entrenar el modelo. | En K-Means solo se requiere un parámetro para el entrenamiento el modelo. |

Tabla 2. Diferencias entre DBSCAN y K-Means. Fuente: basado en GeeksforGeeks, 2023.

A continuación, veremos el vídeo ***DBSCAN: aplicaciones avanzadas y optimización en Python.***



# Ideas clave


Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=5f972c9e-00bc-4478-a1c6-b1be015c8563

## 5.8. Cuaderno de ejercicios

### Ejercicio 1. Conceptos básicos

Explica en tus propias palabras qué es el algoritmo DBSCAN y cuáles son sus principales ventajas en comparación con otros algoritmos de *clustering* como K-Means. Asegúrate de mencionar al menos tres ventajas clave y dar ejemplos de situaciones en las que DBSCAN sería más adecuado que K-Means.

**Solución**

El alumno debería explicar que DBSCAN (Density-Based Spatial Clustering of Applications with Noise) es un algoritmo de *clustering* que agrupa puntos de datos en función de la densidad de puntos en una región.

Ventajas:

- No requiere especificar el número de clústeres: a diferencia de K-Means, que necesita que se defina el número de clústeres antes de ejecutarse.
- Forma arbitraria de clústeres: puede identificar clústeres de cualquier forma, no solo esféricos como K-Means.
- Manejo del ruido: DBSCAN puede detectar y manejar ruido y valores atípicos, etiquetándolos como ruido en lugar de forzarlos a un clúster.

### Ejercicio 2. Parametrización

Describe los dos parámetros principales de DBSCAN, épsilon ( ) y MinPts, y explica cómo afectan al resultado del *clustering.* Proporciona un ejemplo de cómo ajustar estos parámetros en un conjunto de datos específico.

**Solución**

es la distancia máxima entre dos puntos para que se consideren vecinos. Si es

muy pequeño, muchos puntos serán considerados ruido. Si es muy grande, clústeres diferentes pueden fusionarse.

MinPts es el número mínimo de puntos que debe tener una región para ser considerada un clúster denso. Por ejemplo, en un conjunto de datos de dos dimensiones, MinPts debería ser al menos 3 o 4. Ajustar estos parámetros puede implicar el uso de una gráfica de distancia para encontrar un buen valor de y establecer MinPts basado en el conocimiento del dominio.

## Ejercicio 3. Implementación DBSCAN en Python

Describe los pasos básicos para implementar el algoritmo DBSCAN en Python utilizando la biblioteca Scikit-learn. Incluye un breve código de ejemplo y explica cada parte del código.

**Solución**

- Importar las bibliotecas necesarias: numpy , matplotlib , sklearn.cluster.DBSCAN .
- Generar o cargar el conjunto de datos.
- Preprocesar los datos: normalización o estandarización.
- Aplicar el algoritmo DBSCAN: configurar los parámetros ϵ y MinPts.
- Visualizar los resultados: graficar los clústeres identificados.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler

# Generar datos aleatorios
np.random.seed(42)
X = np.random.rand(100, 2)

# Escalar los datos
X = StandardScaler().fit_transform(X)
```

```
# Aplicar DBSCAN
db = DBSCAN(eps=0.3, min_samples=5).fit(X)
labels = db.labels_

# Graficar los resultados
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.show()
```

## Ejercicio 4. Evaluación de clustering con DBSCAN

Explica cómo se puede evaluar la calidad de un *clustering* realizado con DBSCAN utilizando la puntuación de la silueta y la puntuación de Rand. Proporciona ejemplos de cómo interpretar estos valores en un conjunto de datos.

**Solución**

La silueta mide la cohesión y separación de los clústeres, con valores que van de -1 a 1. Una puntuación cercana a 1 indica clústeres bien definidos. La puntuación de Rand compara la similitud entre las agrupaciones generadas y una verdad conocida, con valores entre 0 y 1. Un valor superior a 0.9 indica un modelo muy bueno.

Ejemplo de interpretación: si la puntuación de la silueta es 0.75 y la puntuación de Rand es 0.85, se puede concluir que el *clustering* es de alta calidad, con buena separación entre clústeres y consistencia con las etiquetas reales, si existen.

## 5.9. Referencias bibliográficas

GeeksforGeeks. (2023, mayo 23). *Agrupación DBSCAN en ML | agrupación basada en densidad.* https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/

## Vídeo explicativo sobre DBSCAN

StatQuest with Josh Starmer. (2022, enero 10). *Clustering with DBSCAN, Clearly Explained!!!* [Vídeo]. YouTube. https://www.youtube.com/watch?v=RDZUdRSDOok

Este vídeo ofrece una explicación clara y detallada del algoritmo DBSCAN, abordando sus conceptos básicos, cómo funciona y ejemplos prácticos de implementación en Python.

## Tutorial de codificación de DBSCAN en Python

Greg Hogg. (2022, agosto 22). *DBSCAN Clustering Coding Tutorial in Python & Scikit-Learn* [Vídeo]. YouTube. https://www.youtube.com/watch?v=VO_uzCU_nKw

Este tutorial se centra en la implementación práctica del algoritmo DBSCAN utilizando Python y la biblioteca Scikit-learn. Cubre desde la configuración de los parámetros hasta la visualización de los resultados, proporcionando una guía paso a paso.

# Test

1. ¿Qué significa DBSCAN?

   A. Density-Based Spatial Clustering of Applications with Noise.

   B. Data-Based Spatial Clustering Algorithm.

   C. Density-Balanced Spatial Clustering Algorithm.

   D. Distance-Based Spatial Clustering of Applications with Noise.

2. ¿Cuál de los siguientes parámetros es clave en DBSCAN?

   A. K.

   B. Épsilon (ϵ).

   C. Sigma (σ).

   D. Alpha (α).

3. ¿Qué tipo de clústeres puede identificar el DBSCAN?

   A. Solamente esféricos.

   B. Solamente convexos.

   C. De cualquier forma.

   D. Solamente cóncavos.

4. ¿A qué hace referencia el parámetro MinPts?

   A. El número mínimo de puntos en el *dataset.*

   B. El número mínimo de clústeres.

   C. El número mínimo de puntos para formar un clúster denso.

   D. La distancia mínima entre dos puntos.

**5.** ¿Qué ocurre si el valor de épsilon (ϵ) es muy pequeño?

A. Todos los puntos se considerarán clústeres.

B. La mayoría de los puntos serán considerados ruido.

C. Todos los puntos se unirán en un solo clúster.

D. Se formarán clústeres de forma esférica.

**6.** ¿Cómo se denomina un punto que tiene menos de MinPts dentro de la distancia ϵ pero que está cerca de un punto central?

A. Punto central.

B. Punto de ruido.

C. Punto aislado.

D. Punto fronterizo.

**7.** ¿Cuál es una ventaja importante de DBSCAN frente a K-Means?

A. Requiere menos parámetros.

B. Identifica clústeres de forma esférica.

C. Es más rápido.

D. No necesita especificar el número de clústeres de antemano.

**8.** ¿Qué métrica de evaluación se puede usar para evaluar la calidad de un *clustering* con DBSCAN?

A. Puntaje de la silueta.

B. Error cuadrático medio.

C. Puntaje de Z-Score.

D. Prueba t de Student.

**9.** En el contexto de DBSCAN, ¿para qué se utiliza un KD-Tree?

A. Acelerar la búsqueda de vecinos.

B. Reducir el número de puntos en el *dataset.*

C. Mejorar la precisión del *clustering.*

D. Aumentar el número de clústeres.

**10.** ¿Qué hace DBSCAN con los puntos que no pertenecen a ningún clúster?

A. Los redistribuye a clústeres existentes.

B. Los etiqueta como ruido.

C. Los elimina del *dataset.*

D. Los marca como puntos centrales.