TICS-411 Minería de Datos

Clase 5: DBSCAN

Alfonso Tobar-Arancibia
alfonso.tobar.a@edu.uai.cl

Clustering por Densidad

Clustering: Densidad

> Se basan en la idea de continuar el crecimiento de un cluster a medida que la densidad (número de objetos o puntos) en el vecindario sobrepase algún umbral.

ⓘ En nuestro caso utilizaremos **DBSCAN** (Density-Based Spatial Clustering Applications with Noise).

DBSCAN: Definiciones

Hiperparámetros del Modelo

- `eps`: Radio de análisis
- `MinPts`: Corresponde al mínimo de puntos necesarios en un Radio `eps`.

Densidad

> Densidad es el número de puntos dentro del radio `eps`.

Core Point/Punto Central

> Un punto central/core es aquel que tiene al menos `MinPts` puntos dentro de la esfera definida por `eps` (se incluye él mismo).

Border Point/Punto Borde

> Un punto de borde tiene menos puntos que `MinPts` del `eps`, pero está dentro de la esfera de un punto central.

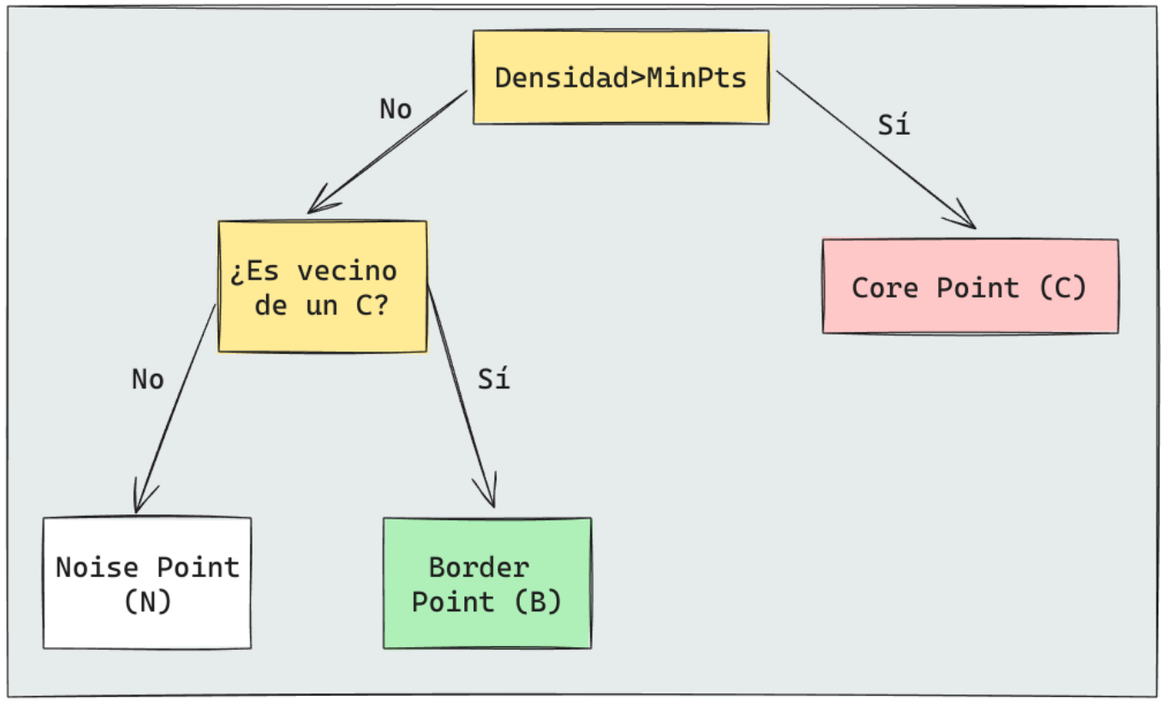
Noise Point/Punto Ruido

> Un punto de ruido es todo aquel que no es punto central ni de borde.

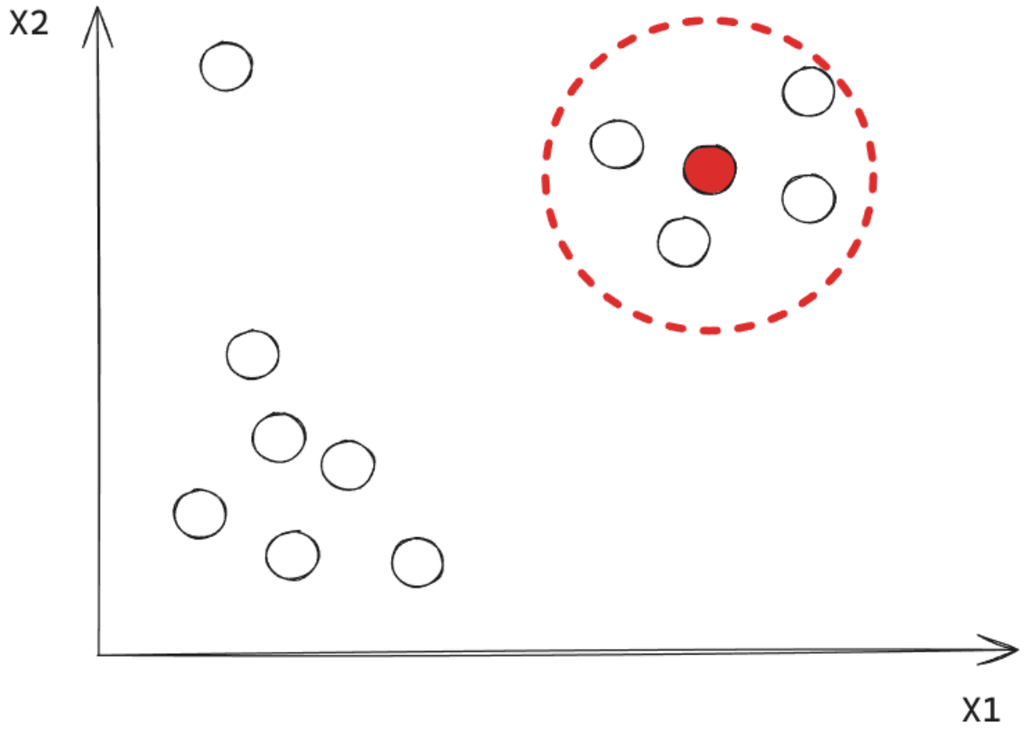
DBSCAN: Algoritmo categorización de puntos

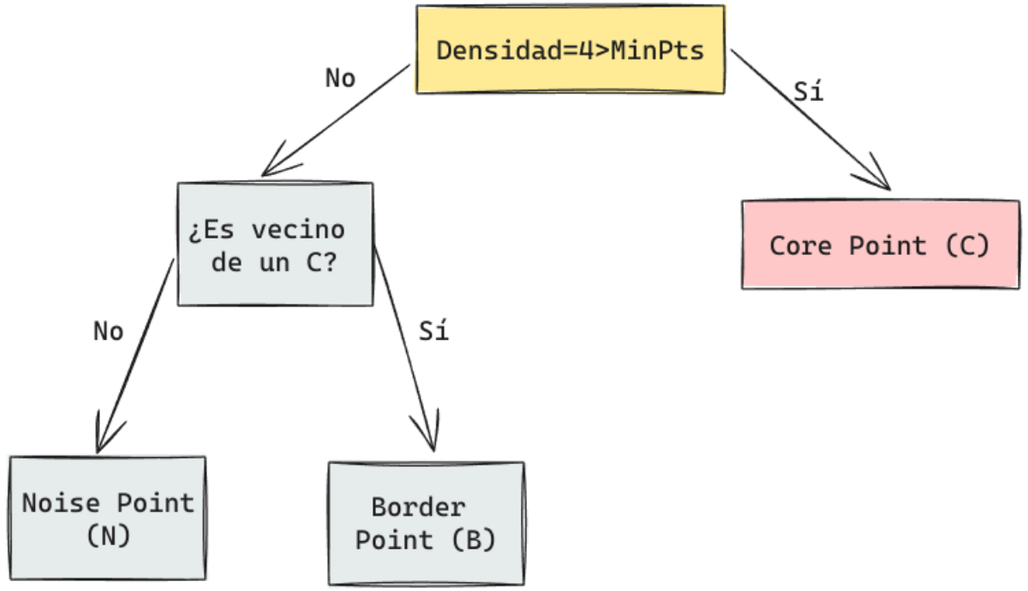
> Primeramente se aplica un algoritmo para categorizar cada punto de acuerdo a las definiciones anteriores.

- Para cada punto en el espacio:
  - Calcular su densidad en EPS y aplicar el siguiente algoritmo:



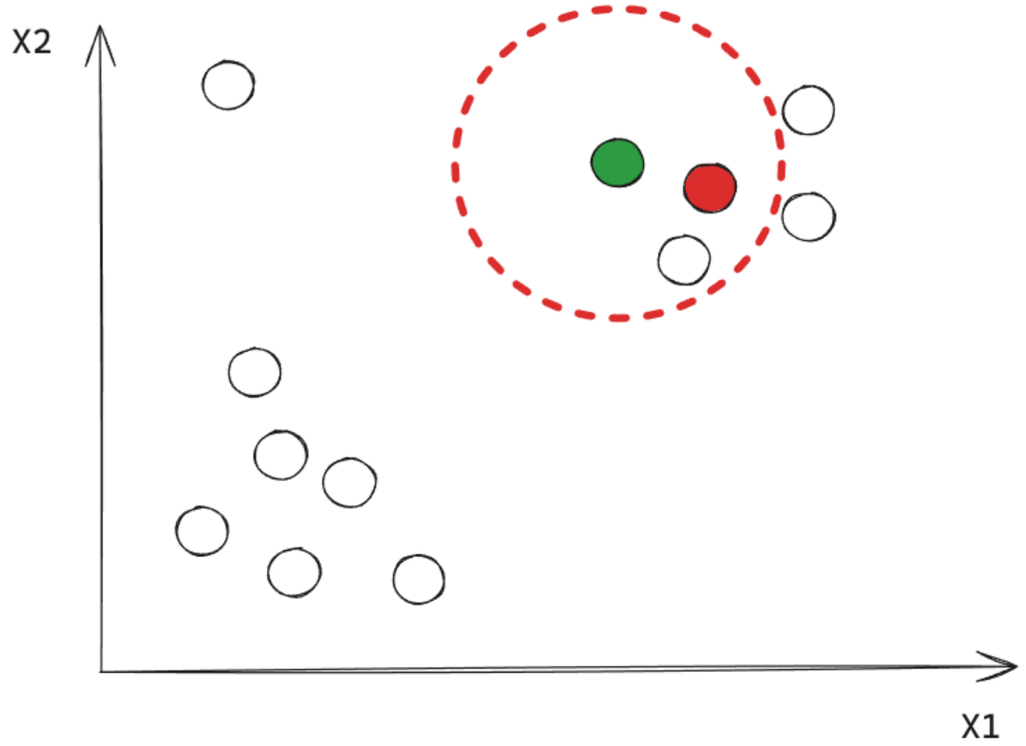
Ejemplo: Iteración 1

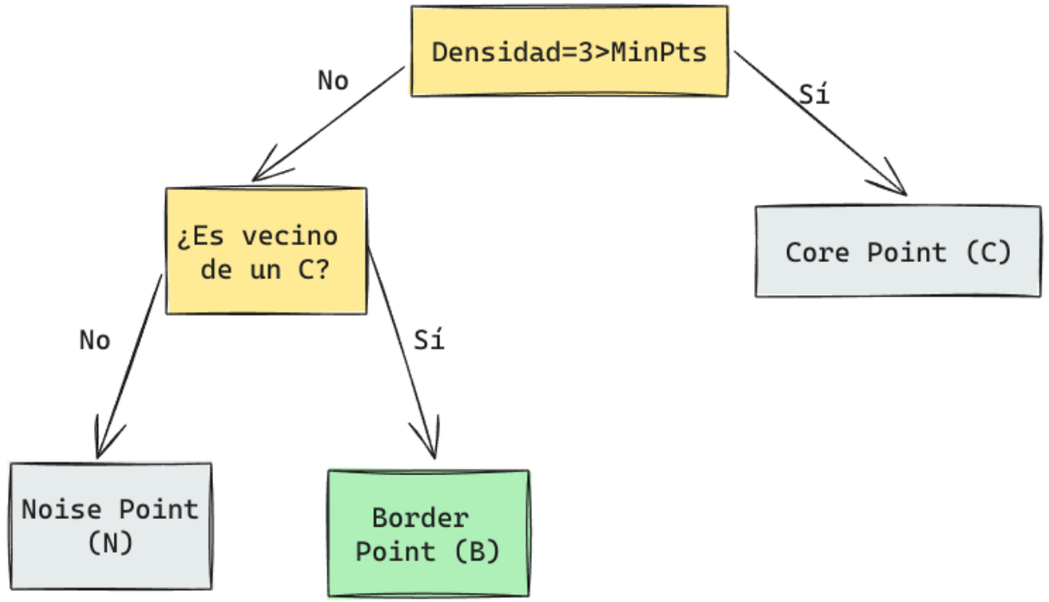




ⓘ Este punto corresponde a un **Core Point**.

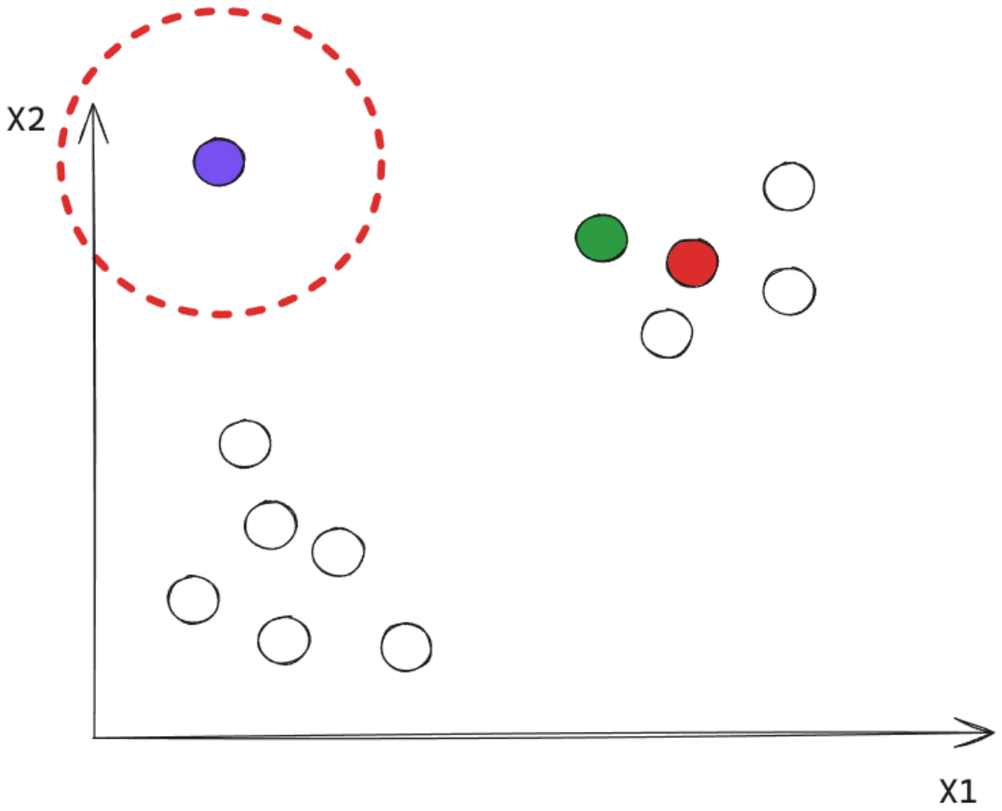
Ejemplo: Iteración 2

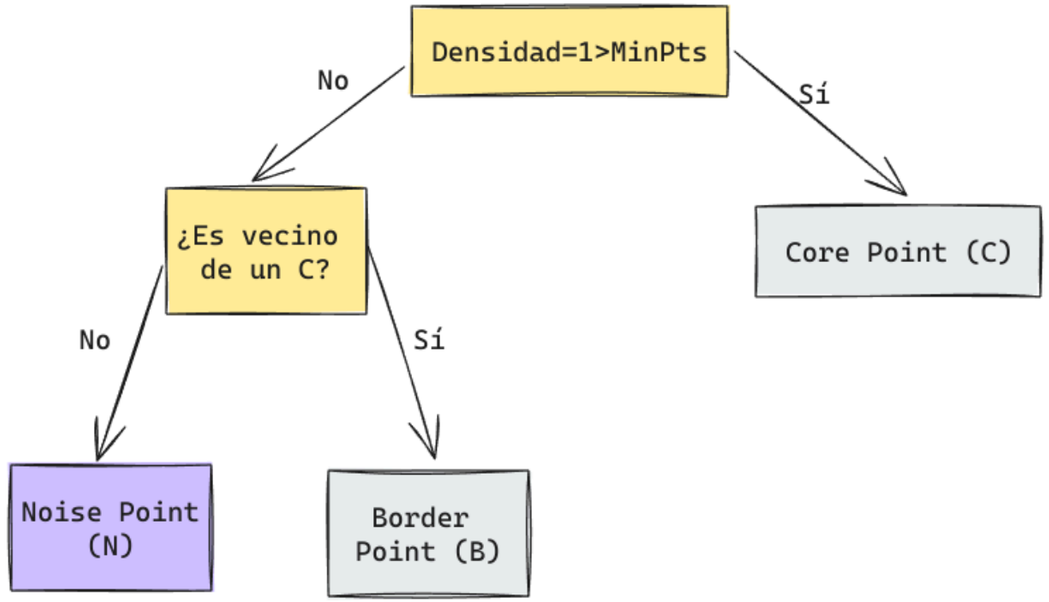




ⓘ Este punto corresponde a un **Border Point.**

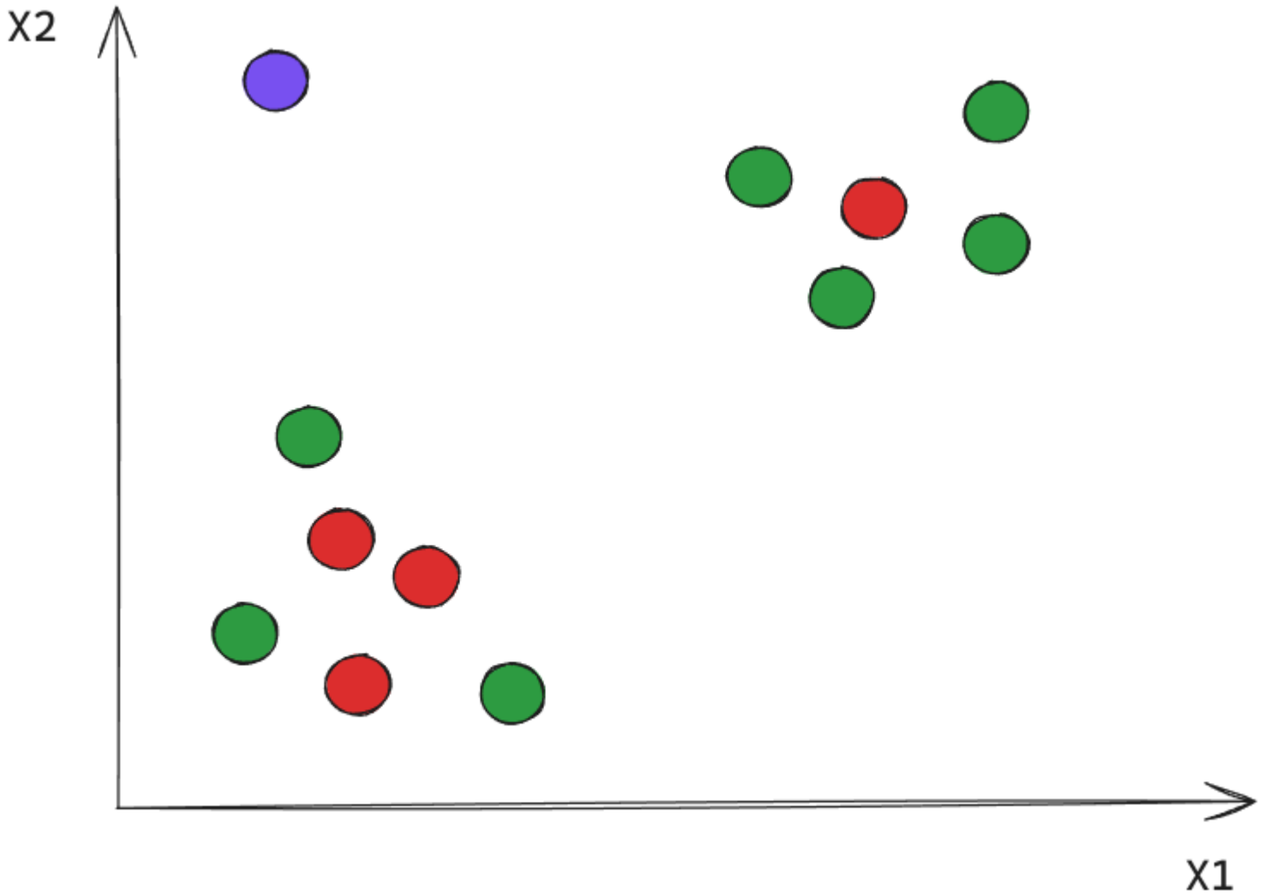
Ejemplo: Iteración 3





ⓘ Este punto corresponde a un **Noise Point**.

Ejemplo: Iteración Final



Ahora, **¿Cómo definimos que partes son clusters o no?**

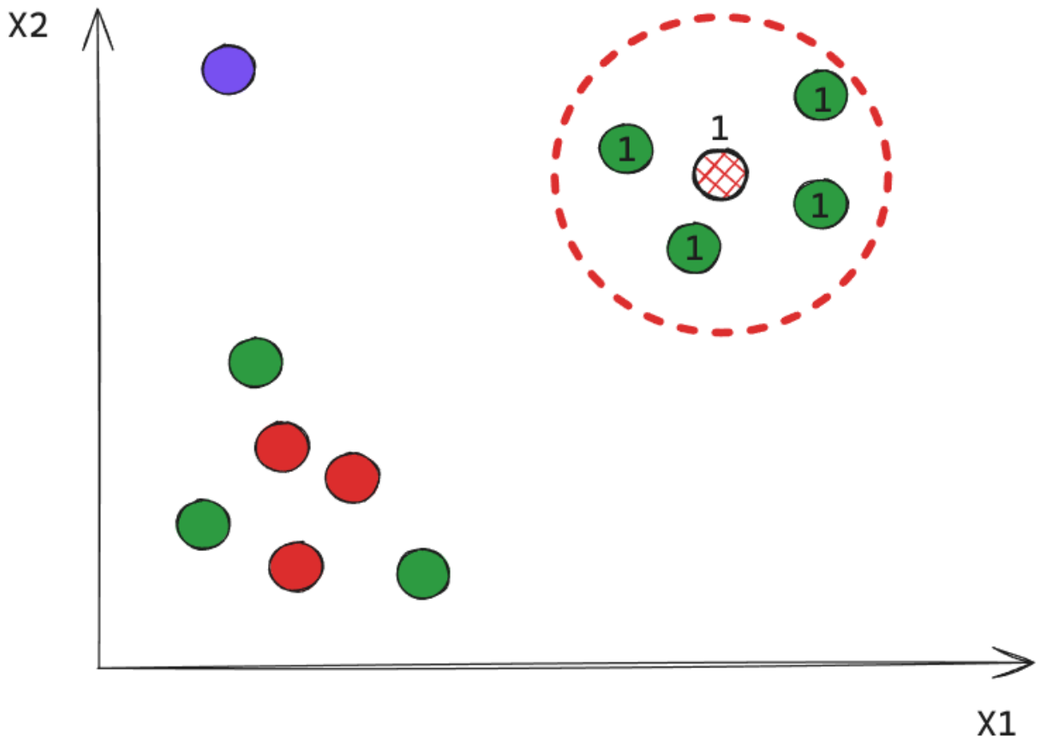
Algoritmo de Clustering

Se aplica el siguiente algoritmo para calcular clusterings.

> Antes de aplicar se desechan los Noise Points ya que no serán considerados. **(Veremos luego que ocurre con estos puntos)**.

```
label=0
for punto_c in corePoints:
    if punto_c no tiene etiqueta:
        label += 1
        punto_c = label
    for point_eps dentro de eps:
        if punto_eps no tiene etiqueta:
            punto_eps = label
```

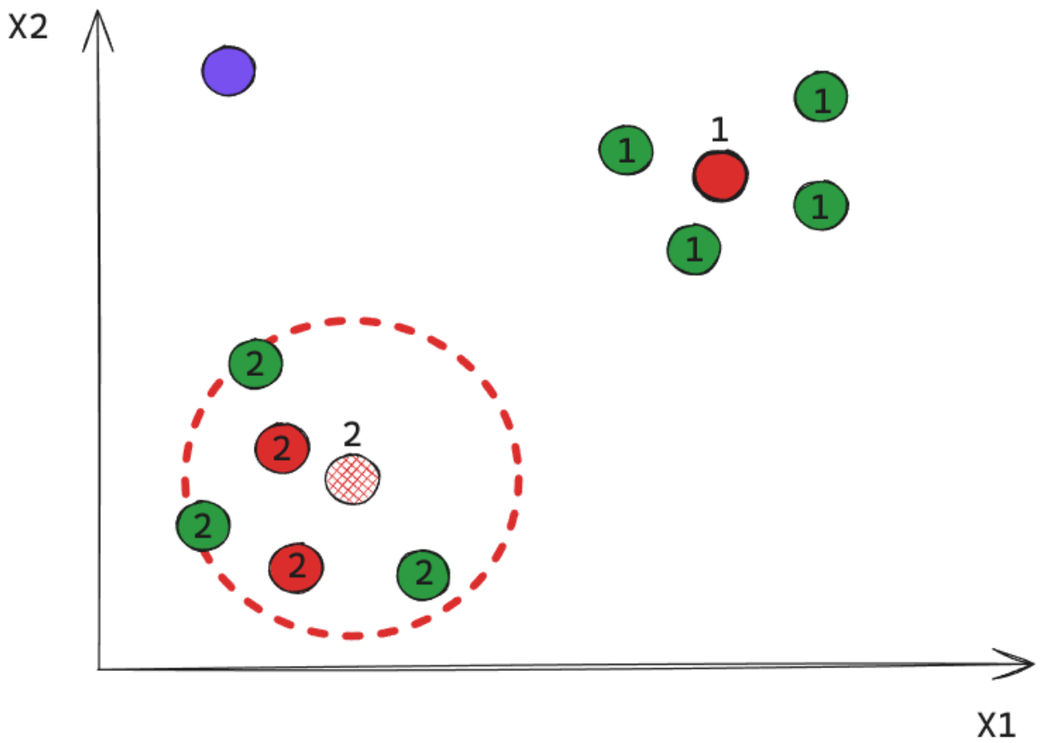
Iteración 1




```
label=0
for punto_c in corePoints:
    if punto_c no tiene etiqueta:
        label += 1
        punto_c = label
    for point_eps dentro de eps:
        if punto_eps no tiene etiqueta:
            punto_eps = label
```

Todos los puntos cercanos a un Core reciben la misma etiqueta.

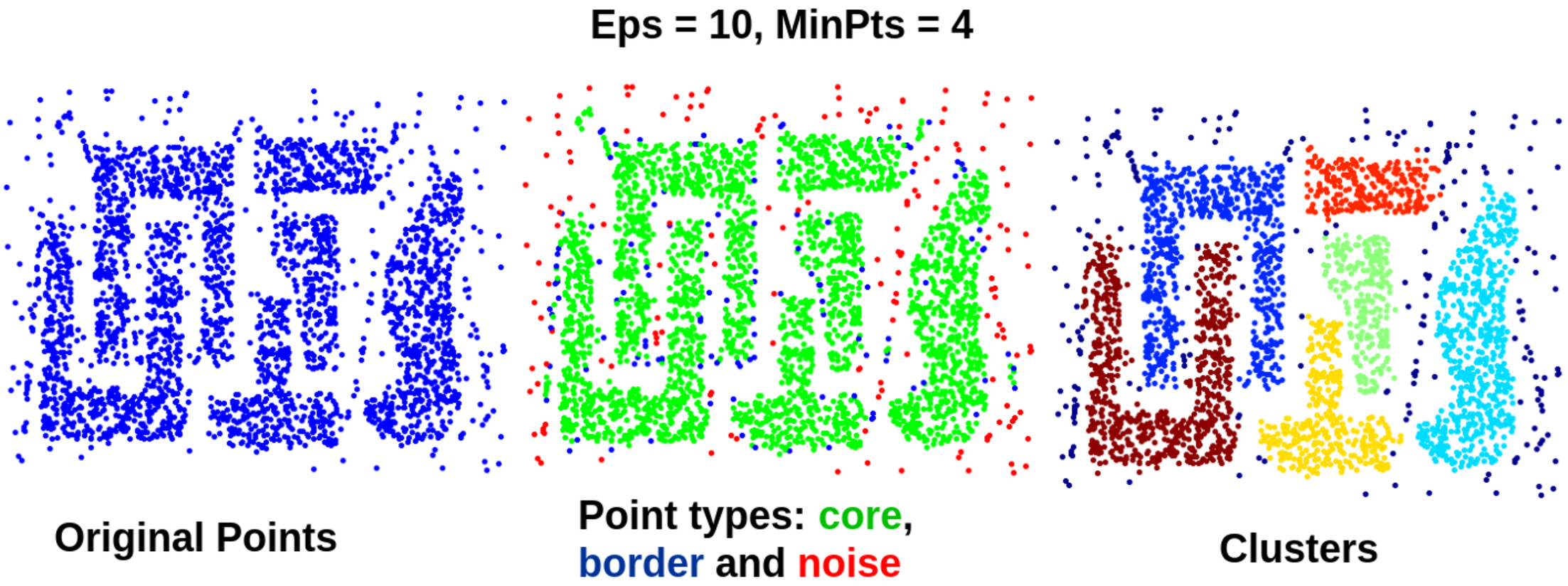
Iteración 2




```
## label ya está en 1
for punto_c in corePoints:
    if punto_c no tiene etiqueta:
        label += 1
        punto_c = label
    for point_eps dentro de eps:
        if punto_eps no tiene etiqueta:
            punto_eps = label
```

En este caso obtuvimos 2 clusters, e indirectamente un 3er de puntos ruido.

DBSCAN



¿Sería posible replicar un proceso de Clustering similar utilizando K-Means? ¿Por qué?

DBSCAN: Detalles Técnicos

Fortalezas

- Resistente al ruido.
- Puede lidiar con clusters de diferentes formas y tamaños.
- No es necesario especificar cuántos clusters encontrar.

Debilidades

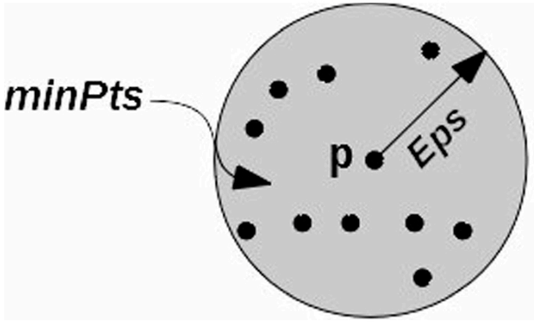
- Algoritmo de alta complejidad computacional que puede llegar $O(n^2)$ en el peor caso.
- Se ve afectado por densidad de los datos y por datos con una alta dimensionalidad.
- Su óptimo resultado depende específicamente de sus Hiperparámetros.
- No puede generalizar en datos no usados en entrenamiento.

¿Cómo encontrar los Hiperparámetros?

minPts

Para datasets multidimensionales grandes, la regla es:

$$minPts \geq dim + 1$$



Otras recomendaciones:

- Para dos dimensiones: $minPts = 4$ (Ester et al., 1996)
- Para más de 2 dimensiones: $minPts = 2 \cdot dim$ (Sander et al., 1998)

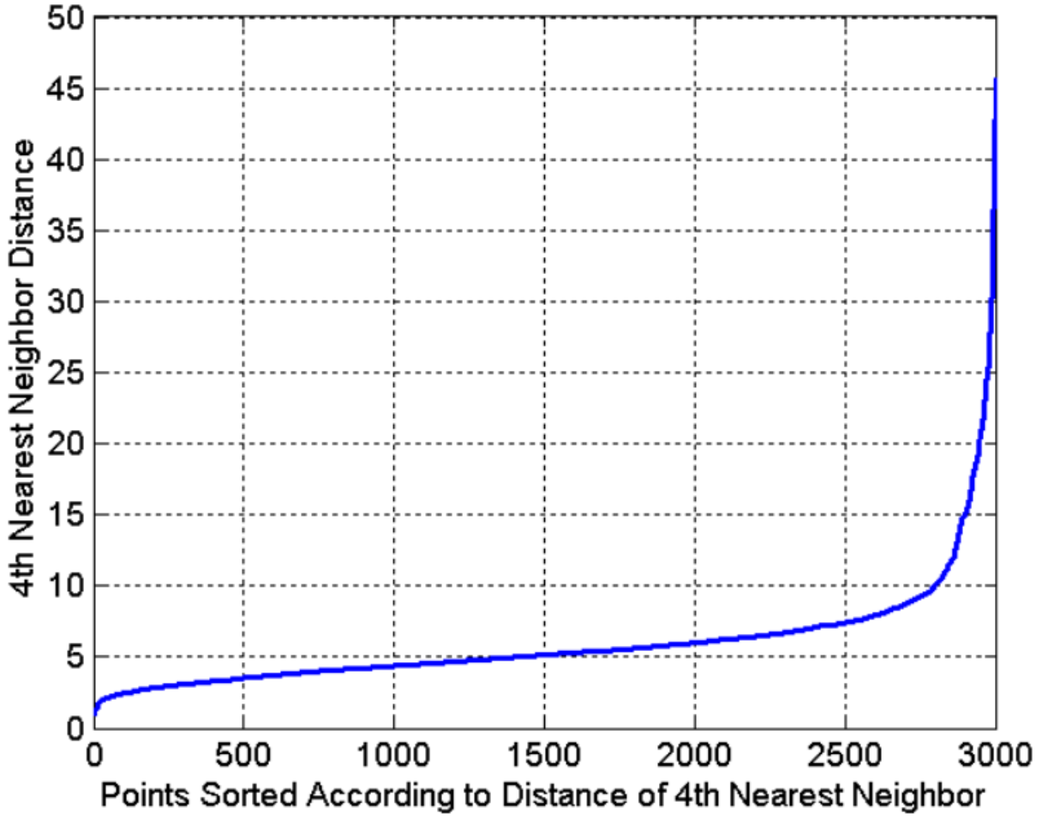
¿Cómo encontrar los Hiperparámetros?

> Para encontrar EPS se suele utilizar el método de Vecinos más cercanos.

Idea

- La distancia de los puntos dentro de un cluster a su k-ésimo vecino deberían ser similares.
- Luego, los puntos atípicos (o ruidosos) tienen el k-ésimo vecino a una mayor distancia.

💡 Podemos plotear la distancia ordenada de cada punto a su k-ésimo vecino y seleccionar un `eps` cercano al crecimiento exponencial (codo).



Implementación en Scikit-Learn

```python
from sklearn.cluster import DBSCAN

dbs = DBSCAN(min_samples = 5, eps = 0.5, metric = "euclidean")

## Se entrena y se genera la predicción
dbs.fit_predict(X)
```

- **min_samples**: Corresponde a minPts. Por defecto 5.
- **eps**: Corresponde al radio de la esfera en la que se buscan los puntos cercanos. Por defecto 0.5.
- **metric**: Corresponde a la distancia utilizada para medir la distancia. Permite todas las distancias mencionadas acá.
- `.fit_predict()`: Entrenará el modelo en los datos suministrados e inmediatamente genera el cluster asociado a cada elemento. Adicionalmente los puntos ruidosos se etiquetarán como -1.

👀 Veamos un ejemplo.

It’s over!!