Base matematica

$cond(A) = ||A||_2||A^{-1}||_2, \quad ||\delta d|| = ||A^{-1}\delta p||_2, \quad A^TA > 0$
$\mu(A) = \{\mathbf{x} \,/\, A\mathbf{x} = \mathbf{0}\}, \quad \text{cols}(A) \quad L.I. \iff \mu(A) = \{\mathbf{0}\}$
$rank(A) = \text{cols}(A) \quad L.I., \quad \text{cols}(A) = rank(A) - \mu(A)$
Matriz ortogonal: $A^T = A^{-1}$, columnas U_i:
$U_i^TU_j = 0$ si $i \neq j$, y $U_i^TU_i = ||U_i||^2$ si $i = j$
Matriz ortonormal: Matriz ortogonal con $||U_i||^2 = 1$
Gramm-Schmidt: $\{v_1, \dots, v_n\} \to \{e_1, \dots, e_n\}$ ortonormal:
$u_1 = v_1, \quad u_k = v_k - \sum_{i=1}^{k-1} \frac{v_k \cdot u_i}{u_i \cdot u_i} u_i, \quad e_k = \frac{u_k}{||u_k||}$
Diagonalizacion: $A = CDC^{-1}$, Q ortonormal, D diagonal con autovalores λ_i en la diagonal, $rank(A) = r = \#\lambda_i > 0$
Matriz simetrica: $A = A^T$, $A = Q\Lambda Q^T$, Q ortonormal, Λ diagonal con autovalores λ_i, $rank(A) = r = \#\lambda_i > 0$
Propiedades útiles de matrices
$(A^{-1})^T = (A^T)^{-1}$; $(A^T)^T = A$; $(A^TB^T) = (BA)^T$; $\det(A^T) = \det(A)$, $(AB)^{-1} = B^{-1}A^{-1}$
SVD: $A = U\Sigma V^T$, U y V ortonormales, Σ diagonal con valores singulares σ_i en la diagonal, $rank(A) = r = \#\sigma_i > 0$
Pseudoinversa: $A^\dagger = (A^TA)^{-1}A^T$
Conjunto convexo: $C \in R^n$ es convexo si $\forall x, y \in C$, $tx + (1-t)y \in C$, $t \in [0, 1]$
Funcion convexa: $\forall x, y \in C$ convexo,
$f(tx + (1-t)y) \leqslant tf(x) + (1-t)f(y)$, $t \in [0, 1]$
Sea $g : R^n \to R$ y $h : R \to R$: $f(x) = h(g(x))$ es convexo si: (g y h son convexos, y h es creciente) $\vee$ (g concavo, h convexo, y h es decreciente)
Min-Max a $[a, b]$: $x' = a + (b-a) \times \frac{x - x_{\min}}{x_{\max} - x_{\min}}$
Diferencias Finitas: $\nabla_w \mathcal{L}(w) \approx \frac{\mathcal{L}(w+\epsilon) - \mathcal{L}(w)}{\epsilon}$, ϵ pequeño.
Derivadas de matrices:
$\nabla(U(x)^TV(x)) = \nabla(U(x))^TV(x) + \nabla(V(x))^TU(x)$, $\nabla(a^Tw) = a$;
$\nabla(w^Ta) = a$; $\nabla(w^Tw) = 2w$; $\nabla(w^TAw) = (A + A^T)w$;
$\nabla(||y - Xw||^2) = -2X^T(y - Xw)$; $\nabla(w^TX^Ty) = X^Ty$;
$\nabla(\text{tr}(w^TA)) = A$; $\frac{\partial|A|}{\partial A} = |A|(A^{-1})^T$, $\frac{\partial(\mathbf{z}^TA\mathbf{z})}{\partial A} = \mathbf{z}\mathbf{z}^T$
$\nabla_A \ln|A| = A^{-T}, \quad \nabla_A\left(x^TA^{-1}x\right) = -A^{-T}x\,x^TA^{-T}$
Normal: $\mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$
Teorema de Bayes: $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$

Multiplicadores de Lagrange

$\min_x f(x)$ s.a. $g(x) = 0$. $\mathcal{L}(x, \lambda) = f(x) + \lambda g(x)$.
$\nabla_x\mathcal{L}(x, \lambda) = 0$, $\nabla_\lambda\mathcal{L}(x, \lambda) = 0$.

Likelihood, MLE, NLL y Fisher Information

Sea $\boldsymbol{\theta}$ el parámetro de un modelo y muestras iid.
$\mathcal{L}(\boldsymbol{\theta}) = p(\mathcal{D}, \boldsymbol{\theta}) = p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) = p(\boldsymbol{\theta}) \prod_{i=1}^N p(y_i|x_i, \boldsymbol{\theta})$
$\ell(\boldsymbol{\theta}) = \log\mathcal{L}(\boldsymbol{\theta}) = p(\boldsymbol{\theta}) \sum_{i=1}^N p(y_i|x_i, \boldsymbol{\theta})$, $\text{NLL}(\boldsymbol{\theta}) = -\ell(\boldsymbol{\theta})$
MLE: $\boldsymbol{\theta}^* = \text{argmax}_{\boldsymbol{\theta}}\, \mathcal{L}(\boldsymbol{\theta}) = \text{argmin}_{\boldsymbol{\theta}}\, \text{NLL}(\boldsymbol{\theta})$

ELBO + EM Algorithm

MLE: $\max \sum_{i=1}^N \log p(\boldsymbol{X}|\boldsymbol{\theta})$, $p(\text{X}|\boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$
$p(\boldsymbol{X}|\boldsymbol{\theta})$ dificil, $p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$ fácil de calcular. $\implies$
$\ln p(\boldsymbol{X}|\boldsymbol{\theta}) = \mathcal{L}(q, \boldsymbol{\theta}) + \text{KL}(q||p)$, $\text{KL} \geqslant 0$ y $\text{KL} = 0 \iff q = p$
$\mathcal{L}(q, \boldsymbol{\theta}) = \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \ln \frac{p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})}{q(\boldsymbol{Z})}$, $\text{KL}(q||p) = \sum_{\boldsymbol{Z}} q(\boldsymbol{Z}) \ln \frac{q(\boldsymbol{Z})}{p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta})}$
$\mathcal{L}(q, \boldsymbol{\theta}) \leqslant \ln p(\boldsymbol{X}|\boldsymbol{\theta})$ (ELBO), $\max_q \mathcal{L}(q, \boldsymbol{\theta}) \iff \text{KL} = 0$

E Step: $\max_q \mathcal{L}(q, \boldsymbol{\theta}^{\text{old}}) = p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$
$\mathcal{L}(q^*, \boldsymbol{\theta}) = Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) + \text{C}(\theta^{\text{old}})$
$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{z \in \boldsymbol{Z}} p(z|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\boldsymbol{X}, z|\boldsymbol{\theta})$
$\text{C}(\boldsymbol{\theta}^{\text{old}}) = -\sum_{z \in \boldsymbol{Z}} p(z|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(z|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})$
Q es expected complete data log-likelihood:
$Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \mathbb{E}_{p(z|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}})}[\ln p(\boldsymbol{X}, z|\boldsymbol{\theta})]$

M Step: $\boldsymbol{\theta}^{\text{new}} = \max_{\boldsymbol{\theta}} Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$
Repetir E y M hasta convergencia.

Gaussian Mixture Model (GMM)

$\boldsymbol{X} = \{x_1, \dots, x_N\}$, $\boldsymbol{Z} = \{z_1, \dots, z_N\}$, $\boldsymbol{\theta} = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$
$p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) = \prod_{i=1}^N p(z_i|\boldsymbol{\pi})p(x_i|z_i, \boldsymbol{\mu}, \boldsymbol{\Sigma})$
$= \prod_{i=1}^N \sum_{k=1}^K p(z_i = k|\boldsymbol{\pi})p(x_i|z_i = k, \boldsymbol{\mu}, \boldsymbol{\Sigma})$
$p(x_i|z_i = k, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}(x_i|\mu_k, \Sigma_k)$, $p(z_i = k|\boldsymbol{\pi}) = \pi_k$, $\sum_{k=1}^K \pi_k = 1$
EM para GMM:
E Step: Calcular responsabilidades
$\gamma_{ik} = p(z_i = k|x_i, \boldsymbol{\theta}^{\text{old}}) = \frac{\pi_k^{\text{old}}\mathcal{N}(x_i|\mu_k^{\text{old}}, \Sigma_k^{\text{old}})}{\sum_{j=1}^K \pi_j^{\text{old}}\mathcal{N}(x_i|\mu_j^{\text{old}}, \Sigma_j^{\text{old}})}$
$Q = \sum_{i=1}^N \sum_{k=1}^K \gamma_{ik}(\log\pi_k + \log\mathcal{N}(x_i|\mu_k, \Sigma_k))$

M Step (Update): $N_k = \sum_i \gamma_{ik}$
$\nabla_{\pi_k}(Q + \lambda(\sum_k \pi_k - 1)) = N_k/\pi_k + \lambda = 0 \;\to\; \pi_k^{\text{new}} = N_k/N$
$\nabla_{\mu_k}Q = \boldsymbol{\Sigma}_k^{-1}\sum_i \gamma_{ik}(x_i - \mu_k) = 0 \;\to\; \mu_k^{\text{new}} = {N_k}^{-1}\sum_i \gamma_{ik}x_i$
$\nabla_{\Sigma_k}Q = \frac{1}{2}\boldsymbol{\Sigma}_k^{-1}\left[\sum_i \gamma_{ik}(x_i - \mu_k)(x_i - \mu_k)^\top - N_k\boldsymbol{\Sigma}_k\right]\boldsymbol{\Sigma}_k^{-1} = 0$
$\to \Sigma_k^{\text{new}} = \dfrac{1}{N_k}\sum_i \gamma_{ik}(x_i - \mu_k^{\text{new}})(x_i - \mu_k^{\text{new}})^\top$

K-Means Algorithm

K Means sólo permite tener clusters lineales (círculos), y no da una noción de la probabilidad de que la muestra pertenezca a una clase.
Supuestos desde GMM:
1. $\pi_k = \frac{1}{K}$ 2. $\Sigma_k = \sigma^2\boldsymbol{I}$ 3. $\sigma^2 \to 0$ $\Rightarrow$ asignaciones duras
$\boldsymbol{\theta} = \{\mu_k, r_{ik}\}_{k=1}^K$, $\sum_{k=1}^K r_{ik} = 1$
$p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) = \prod_{i=1}^N \sum_{k=1}^K p(z_i = k|\boldsymbol{\theta})p(x_i|z_i = k, \boldsymbol{\theta})$
$p(z_i = k|\boldsymbol{\theta}) = \frac{1}{K}$, $p(x_i|z_i = k, \boldsymbol{\theta}) = \mathcal{N}(x_i|\mu_k, \sigma^2\boldsymbol{I})$

EM para K-Means:
E Step: Asignación dura a clúster más cercano
$r_{ik} = p(z_i = k|x_i, \boldsymbol{\theta}^{\text{old}}) = \mathbb{I}[k = \arg\min_j ||x_i - \mu_j^{\text{old}}||^2]$
$Q = \sum_{i=1}^N \sum_{k=1}^K r_{ik} \log\left(\frac{1}{K}\mathcal{N}(x_i|\mu_k, \sigma^2\boldsymbol{I})\right)$

M Step: Actualizar centros de clústeres
$\nabla_{\mu_k}Q = \frac{1}{\sigma^2}\sum_{i=1}^N r_{ik}(x_i - \mu_k) = 0$
$\mu_k^{\text{new}} = \frac{1}{N_k}\sum_{i=1}^N r_{ik}x_i$, donde $N_k = \sum_{i=1}^N r_{ik}$

Usos para Reducción de Dim

Reducir datasets con muestras con alta dimensionalidad; Compresión de datos con pérdida (lossy-compression); Especie de Feature Engineering.; Eliminamos ruido (es una manera, idealizado); Visualización; Modelo Generativos.

PCA

$\boldsymbol{X} \in \mathbb{R}^{N \times D}$, $\tilde{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{\mu}$, $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^N x_i$, $E[\tilde{\boldsymbol{X}}] = \mathbf{0}$
Supuestos: $\exists\mathbf{Z} \in \mathbb{R}^{N \times M}$, $M < D$, $\boldsymbol{Z} = \boldsymbol{V}^T\tilde{\boldsymbol{X}}$, $\boldsymbol{V} \in \mathbb{R}^{D \times M}$
La transformación de $\tilde{\boldsymbol{X}}$ a $\boldsymbol{Z}$ es **LINEAL**.
Objetivo: Maximizar varianza de $\boldsymbol{Z}$ con cols de $\boldsymbol{V}$ versores
Demo con M=1:
$\boldsymbol{Z} = v^T\tilde{\boldsymbol{X}}$, $v \in \mathbb{R}^D$, $||v||_2^2 = 1$
$\text{Var}(\boldsymbol{Z}) = \frac{1}{N}\sum_{i=1}^N(v^T\tilde{x}_i)^2 = v^T\left(\frac{1}{N}\sum_{i=1}^N \tilde{x}_i\tilde{x}_i^T\right)v = v^TCv$,
donde $C = \frac{1}{N}\sum_{i=1}^N \tilde{x}_i\tilde{x}_i^T$ es la matriz de covarianza
Maximizar: $\max_v v^TCv$ s.a. $||v||_2^2 = 1$
Lagrangiano: $\mathcal{L}(v, \lambda) = v^TCv - \lambda(v^Tv - 1)$
$\nabla_v\mathcal{L}(v, \lambda) = 2Cv - 2\lambda v = 0 \implies Cv = \lambda v$
Solución: v es autovector de C con autovalor λ, $\lambda = \text{Var}(\boldsymbol{Z})$.
El mayor autovalor de C es la solución y su autovector es v^*.
Generalización a M > 1 por inducción.

AutoEncoders

Autoencoder: Red neuronal con **Encoder** (realiza la reducción) y **Decoder** (realiza la resconstrucción).
Permite modelar un espacio latente **NO LINEAL**.
Con una capa oculta sin función de activación, llegás a PCA!!
Objetivo: $\min_{\boldsymbol{w}_e, \boldsymbol{w}_d} ||\boldsymbol{X} - \hat{\boldsymbol{X}}||^2$, $\boldsymbol{Z} = \phi_e(\boldsymbol{X}, \boldsymbol{w}_e)$, $\hat{\boldsymbol{X}} = \phi_d(\boldsymbol{Z}, \boldsymbol{w}_d)$
Overfittea si no se regulariza.
Sparse Autoencoder: Objetivo $+\lambda\sum_{i=1}^N\sum_{l \in L}||a_i^{(l)}||_1$.
Penaliza la pre-activación de las neuronas en las capas ocultas, muchos pesos se van a cero.
Denoising Autoencoder: Le metemos ruido al input y le pedimos que prediga lo mismo, entonces es equivalente a sacarle el ruido al dataset.

Variational Autoencoder (VAE)

VAE: Autoencoder con un espacio latente contínuo. Permite generar datos sintéticos. Entra x_i, pasa por el encoder y sale $\boldsymbol{\mu}$ y $\boldsymbol{\sigma}^2$, luego se muestrea z_i (en inferencia se utiliza $\boldsymbol{\mu}$) y se pasa por el decoder para obtener $\tilde{x} \sim \mathcal{N}(g(z_i, w), \sigma_x^2 \mathcal{I})$ (en inferencia se utiliza $g(z_i, w)$).

We've seen that for latent-variable models, the likelihood function is: $p(x|w) = \int p(x|z,w)p(z)dz$, intractable when $p(x|z,w)$ is defined by a deep neural network (and you also can't integrate over z xd). VAEs work with an approximation using three key ideas: 1. Evidence Lower Bound (ELBO) to approximate the likelihood. 2. Amortized inference using an encoder network to approximate posterior distributions. 3. Reparameterization trick to make training tractable.

The Evidence Lower Bound

$\ln p(x|w) = L(q,w) + \mathrm{KL}(q(z)\|p(z|x,w))\ \forall q(z)$

where L is the ELBO: $L(w) = \int q(z) \ln\left(\frac{p(x|z,w)p(z)}{q(z)}\right) dz$

and KL div. is: $\mathrm{KL}(q(z)\|p(z|x,w)) = -\int q(z)\ln\left(\frac{p(z|x,w)}{q(z)}\right) dz$

Since $\mathrm{KL}(q\|p) > 0$, we have $\ln p(x|w) > L$, making L a lower bound on $\ln p(x|w)$.

LL: $\ln p(D|w) = \sum_{n=1}^N L_n + \sum_{n=1}^N \mathrm{KL}(q_n(z_n)\|p(z_n|x_n,w))$

where: $L_n = \int q_n(z_n) \ln\left(\frac{p(x_n|z_n,w)p(z_n)}{q_n(z_n)}\right) dz_n$

Each data point x_n has its own latent variable z_n with distribution $q_n(z_n)$. The exact posterior distribution is: $p(z_n|x_n,w) = \frac{p(x_n|z_n,w)p(z_n)}{p(x_n|w)}$

But the denominator is our intractable likelihood.

Amortized Inference

Instead of evaluating each posterior distribution separately, VAEs train an encoder network to approximate all posterior distributions $q(z|x,\phi)$ where ϕ represents the encoder network parameters.

A typical encoder uses a Gaussian distribution with diagonal covariance: $q(z|x,\phi) = \prod_{j=1}^M \mathcal{N}(z_j|\mu_j(x,\phi), \sigma_j^2(x,\phi))$

The means $\mu_j(x,\phi)$ and variances $\sigma_j^2(x,\phi)$ are outputs of the encoder. z_i is sampled from q.

The Reparameterization Trick

To optimize the ELBO, we rewrite:

$L_n(q,w) = \int q \ln p(x_n|z_n,w)dz_n - \mathrm{KL}(q\|p(z_n))$

The KL divergence between Gaussian distributions can be evaluated analytically:

$\mathrm{KL}(q(z_n|x_n,\phi)\|p(z_n)) = \frac{1}{2}\sum_{j=1}^M (1+\ln\sigma_j^2(x_n) - \mu_j^2(x_n) - \sigma_j^2(x_n))$

For the first term, we use Monte Carlo estimation but need to ensure backpropagation works. The reparameterization trick reformulates sampling to allow gradient calculation:

$z_{ij}^{(l)} = \sigma_j(x_n,\phi)\epsilon_{ij}^{(l)} + \mu_j(x_n,\phi)$

where $\epsilon_{ij}^{(l)}$ is a standard normal variable sample. This makes the dependence on ϕ explicit for backpropagation.

The full error function becomes:

$L = \sum_n \left[\frac{1}{2}\sum_{j=1}^M (1+\ln\sigma_{nj}^2 - \mu_{nj}^2 - \sigma_{nj}^2) + \frac{1}{L}\sum_{l=1}^L \ln p(x_n|z_n^{(l)}, w)\right]$

where L is the number of samples from the encoder.

Training Algorithm

```
Input:
- Training data set D = {x_1, ... , x_N}
- Encoder {mu_j(x_n, phi), s2_j(x_n, phi)}_{j=1}^{M}
- Decoder g(z, w)
- Initial weight vectors w, phi
- Learning rate eta
Output:
- Final weight vectors w, phi
--------------------------------
repeat
L ← 0
for j in {1, ... , M} do
  eps_nj from N(0, 1)
  z_nj <- mu_j(x_n, phi) * eps_nj + s2_j(x_n, phi)
  L <- L + 1/2 * (1 + ln s2_nj - mu_nj**2 - s2_nj)
end for
L ← L + ln p(x_n | z_n , w)
w     ← w     + eta d_w   L
phi   ← phi   + eta d_phi   L
until converged
return w, phi
```

1. Forward propagate through encoder to get $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$
2. Sample from distribution using reparameterization trick
3. Propagate samples through decoder to evaluate ELBO
4. Calculate gradients and update parameters

After training, generate new data by sampling from prior $p(z)$ and forward propagating through decoder.

Common Issues

1. Posterior collapse: $q(z|x,\phi)$ converges to prior $p(z)$, resulting in blurry reconstructions 2. Insufficient compression: Accurate reconstructions but poor generation quality

Solution: Introduce coefficient β to control KL divergence term's regularization effect (it makes the model try to be more similar to $p(z)$):

$L_i = \beta \int q(z_i|x_i,\phi)\ln p(x_i|z_i,w)dz - \mathrm{KL}(q(z_i|x_i,\phi)\|p(z_i))$

$\beta = 1$ is the original VAE, $\beta \to$ inf is a deterministic autoencoder, $\beta \to 0$ makes the model ignore the reconstruction term.

HAC

Agrupa puntos similares, formando un árbol binario jerárquico.

Preprocess: $x_i \leftarrow (x_i - \mu)/\sigma$, $D_{ij} = \|x_i - x_j\|^2, D \in \mathbb{R}^{N\times N}$

Single Link: $d_{\mathrm{SL}}(G,H) = \min_{i\in G,\, j\in H} D_{ij}$

Complete Link: $d_{\mathrm{CL}}(G,H) = \max_{i\in G,\, j\in H} D_{ij}$

Average Link: $d_{\mathrm{AL}}(G,H) = \frac{\sum_{i\in G}\sum_{j\in H} D_{ij}}{|G|\,|H|}$

Single Link es bueno para clusters largos pero es muy sensible al ruido porque produce un efecto cadena. Complete Link permite clusters compactos pero no es bueno para clusters largos. Average Link es un compromiso entre ambos.

Pseudocódigo

```
C      = [[i] for i in range(N)]
C_idx  = list(range(N))
while len(C) > C_cut:
    min_dist = +inf
    for a in range(len(C)):
        for b in range(a+1, len(C)):
            A, B = C[a], C[b]
            # np.ix_ creates meshgrid for indexing
            # D[np.ix_(A, B)] dists all pairs
            # min for SL, max por CL, mean for AL
            dist = D[np.ix_(A, B)].min() # SL
            if dist < min_dist:
                min_dist, pair = dist, (a, b)
    i, j    = pair
    C.append(C[i] + C[j])    # merge
    del C[j], C[i]

labels = np.zeros(N, dtype=int)
for idx, G in enumerate(C):
    for p in G:
        labels[p] = idx
```

DBSCAN

ε (radio de vecindad), m (min_pts)

Regiones densas $\longrightarrow$ clusters. No requiere K.

Puntos con vecinos < min_pts $\longrightarrow$ RUIDO.

Pseudocódigo

```
labels ← NOISE      # -1
cid    ← 0
e ← epsilon
for x in X:
    if labels[x] != NOISE: continue
    Neighbours ← {q : dist(x,q) <= e}
    if |Neighbours| < m: continue # x no es núcleo
    cid ← cid + 1
    labels[x] ← cid
    Q ← list(Neighbours)           # expansión
    while Q:
        q ← Q.pop()
        if labels[q] = NOISE: labels[q] ← cid
        if labels[q] != 0: continue # ya etiquetado
        labels[q] ← cid
        Neig_q ← {s : dist(q,s) <= e}
        if |Neig_q| >= m: Q.extend(Neig_q)
```

$\mathrm{region\_query}(p, \varepsilon) = \{\, q \in D : \|p - q\| \le \varepsilon\}$
Un punto $\boldsymbol{p}$ con $|\mathrm{region\_query}(p, \varepsilon)| \ge m$ es *núcleo*; un vecino de núcleo etiquetado pero sin ser núcleo es *borde*; los restantes se mantienen como *ruido*.