Árboles de decisión

gonzalo.martinez@uam.es

Guión

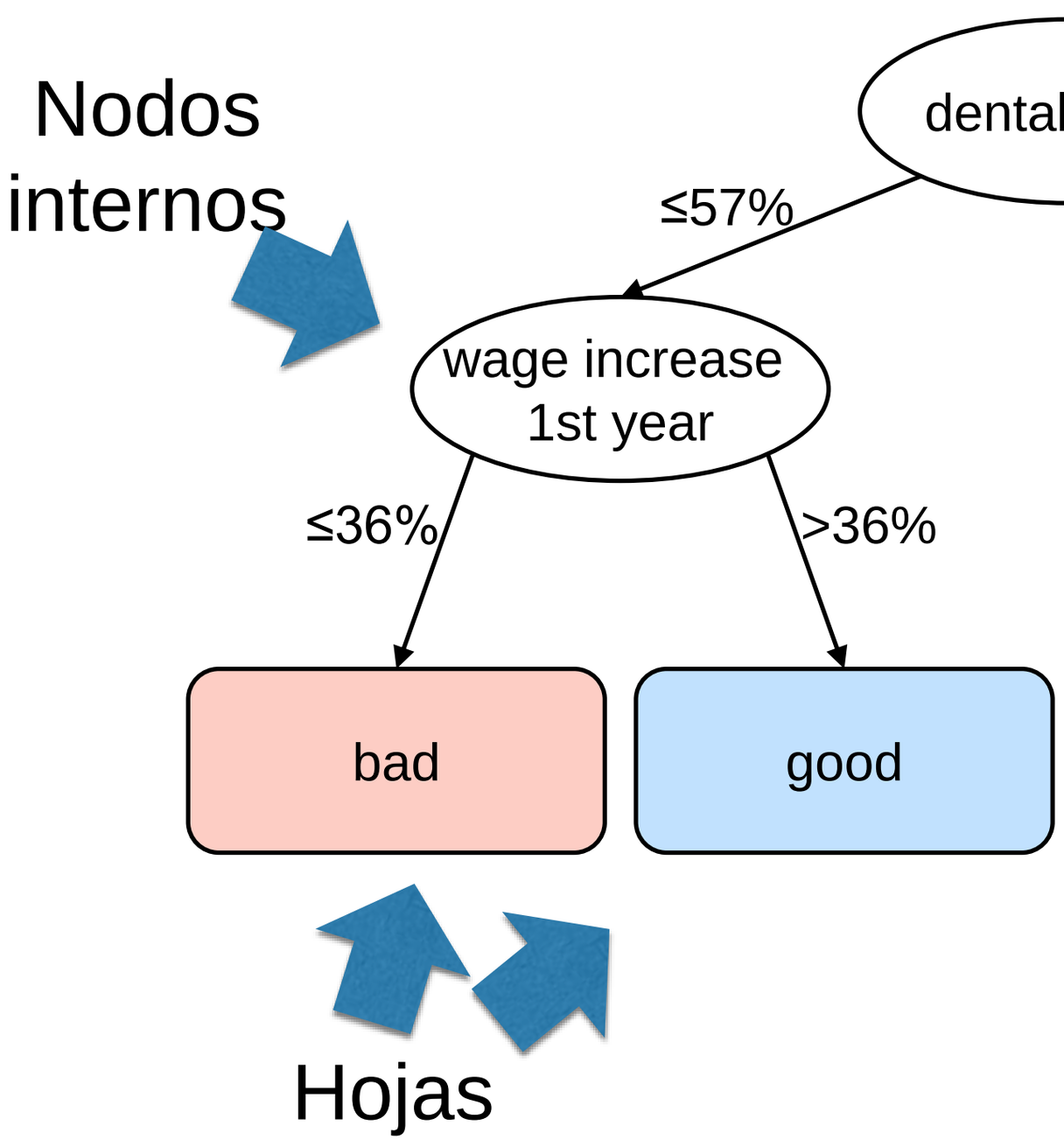
- ¿Qué es un árbol de decisión?
- Historia
- Algoritmos para creación de árboles de decisión
  - Crecer el árbol
  - Podar el árbol
- Capacidades

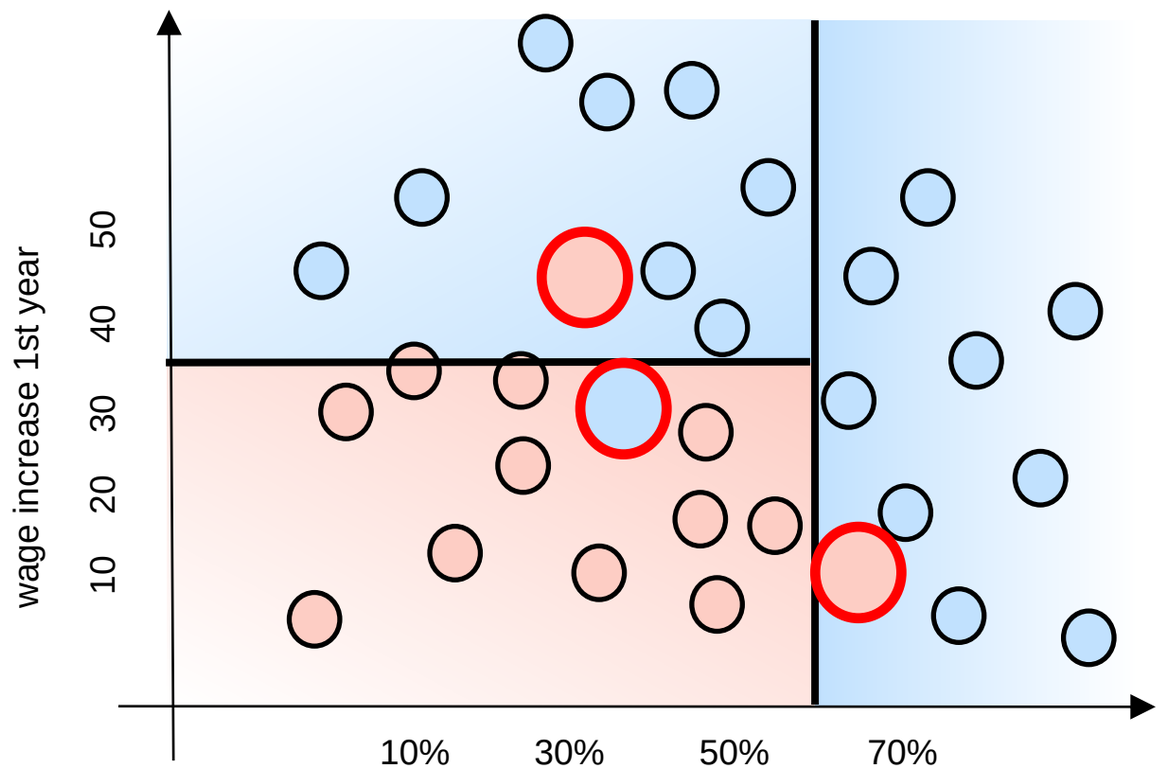
¿Qué es un árbol de decisión?

- Modelo de aprendizaje jerárquico que divide el espacio de usando reglas de decisión
- Es válido para regresión y clasificación
- Vale, pero ¿Qué es un árbol de decisión?

Ejemplo para clasificación

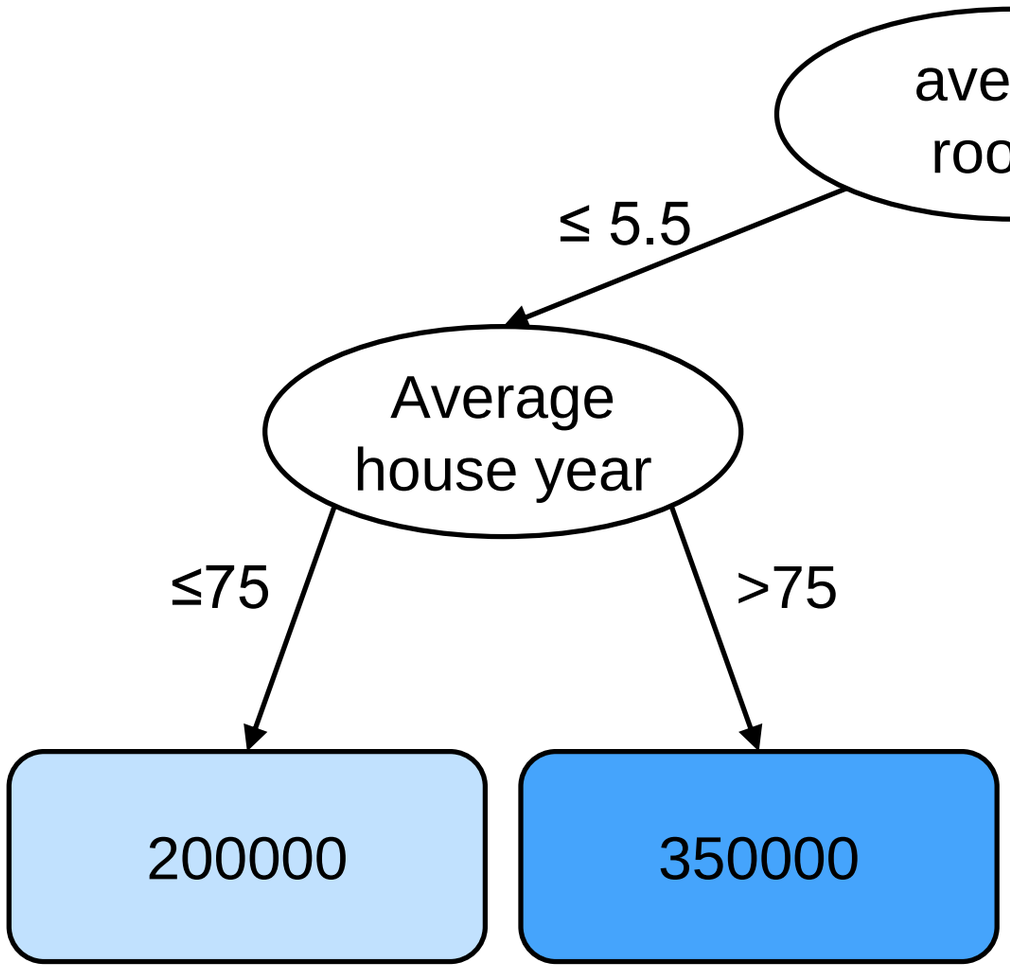
- Negociaciones laborales: predecir if un convenio colectivo es bueno o malo

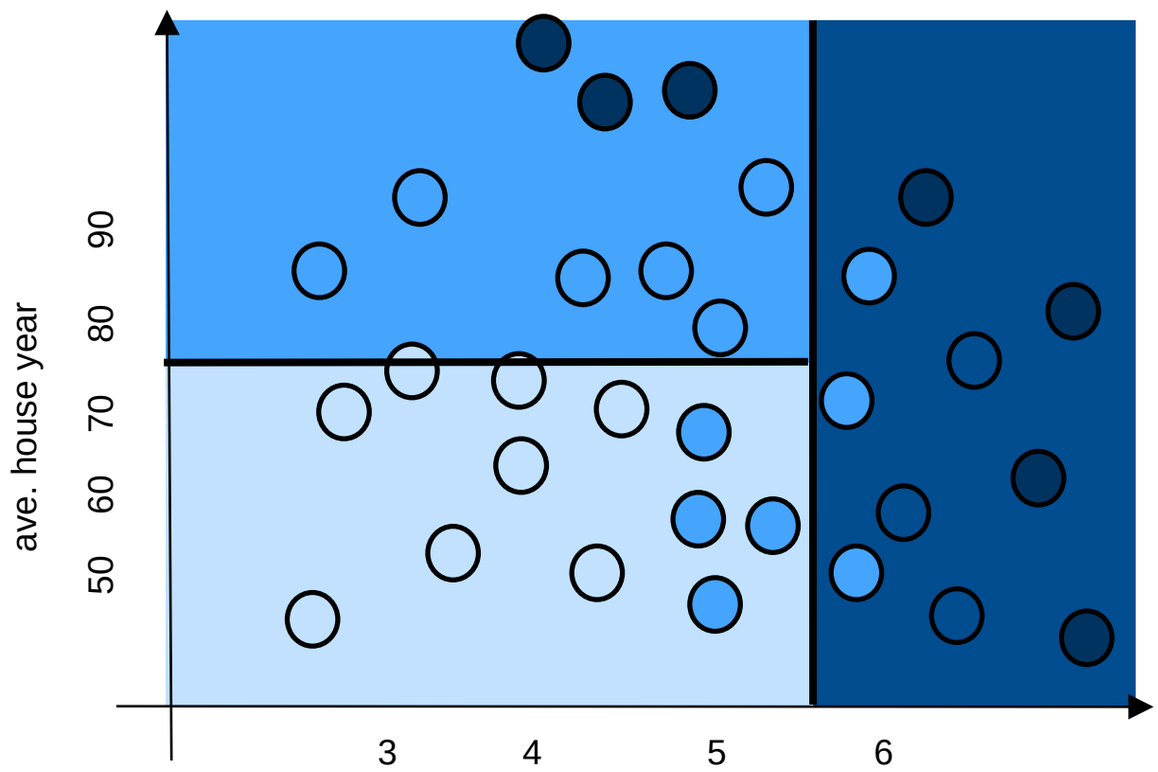




Ejemplo para regresión

- Vivienda en Boston: Predición precios de viviendas en Boston por barrio





Historia

- Precursores: Sistemas expertos (SE)

  SE = BBDD de conocimiento + Motor de inferencia

  - MYCIN: Diagnóstico medico basado en 600 reglas
  - XCON: Sistema para configurar ordenadores VAX, 2500 reglas (1982)

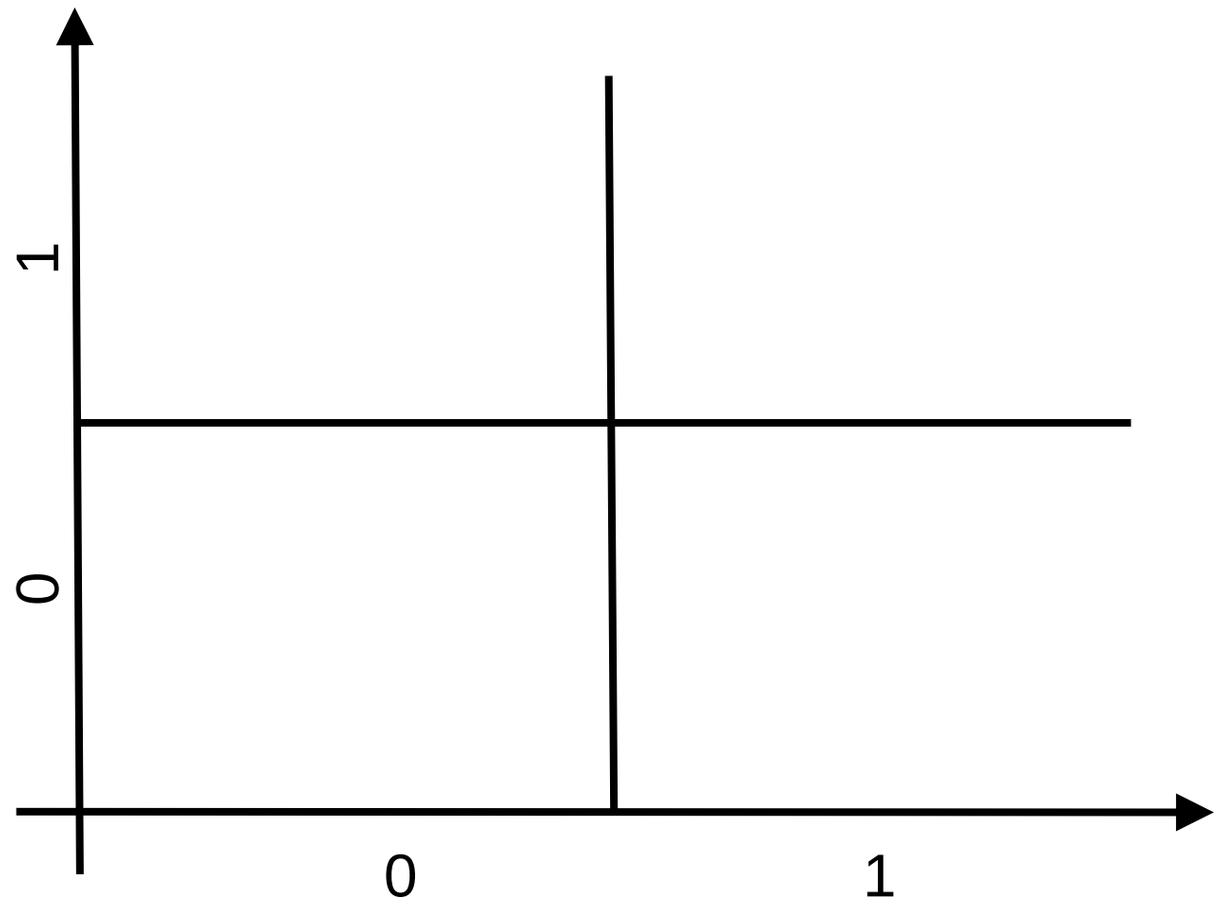
- Las reglas se creaban por expertos a mano!!
- La adquisición de conocimiento debe automatizarse
  - Sustituir al **Experto** por su **archivo de casos resueltos**

Historia

- CHAID (CHi-squared Automatic Interaction Detector) Gordon V. Kass ,1980
- CART (Classification and Regression Trees), Breiman, Friedman, Olsen y Stone, 1984
- ID3 (Iterative Dichotomiser 3), Quinlan, 1986
- C4.5, Quinlan 1993: Basado en ID3

Computacional

- Considera dos variables binarias ¿De cuántas formas posibles podemos dividir el espacio con un árbol de decisión?



- Dos posibles divisiones y dos posibles asignaciones a las hojas → Al menos 8 árboles

Computacional

- ¿Bajo qué condiciones esperamos en un restaurante para cenar?

Example	Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	*Yes*
X_2	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	*No*
X_3	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	*Yes*
X_4	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*10–30*	*Yes*
X_5	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	*No*
X_6	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	*Yes*
X_7	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	*No*
X_8	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	*Yes*
X_9	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	*No*
X_{10}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	*No*
X_{11}	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	*No*
X_{12}	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	*Yes*

Hay 2 x 2 x 2 x 2x 3 x 3 x 2 x 2 x 4 x 4 = 9216 casos

y dos clases → 2^{9216} posibles hipótesis y muchos más árboles !!!!

Computacional

- Es simplemente inviable encontrar la solución óptima
- Debemos tener un sesgo (preferencia) a la hora de construir el modelo.
- Este es un problema general en aprendizaje automático.

Computacional

Para árboles normalmente se una un sesgo codicioso (o cortoplacista):

- Construir el árbol paso a paso en vez de globalmente
- En cada paso seleccionar la mejor división con respecto a los datos de entrenamiento (siguiendo un **criterio de división**).
- El árbol se crece hasta que se cumple un **criterio de parada.**
- El árbol se poda (siguiendo un **criterio de poda**) para evitar sobre ajuste.

Algoritmo básico de construcción de árboles de decisión

```
trainTree(Dataset L)

1. T = growTree(L)
2. pruneTree(T,L)
3. return T
```

La poda: elimina subárboles de validez incierta.

```
growTree(Dataset L)

1. T.s      = findBestSplit(L)
2. if T.s == null return null
3. (L1, L2) = splitData(L,T.s)
4. T.left   = growTree(L1)
5. T.right  = growTree(L2)

6. return T
```

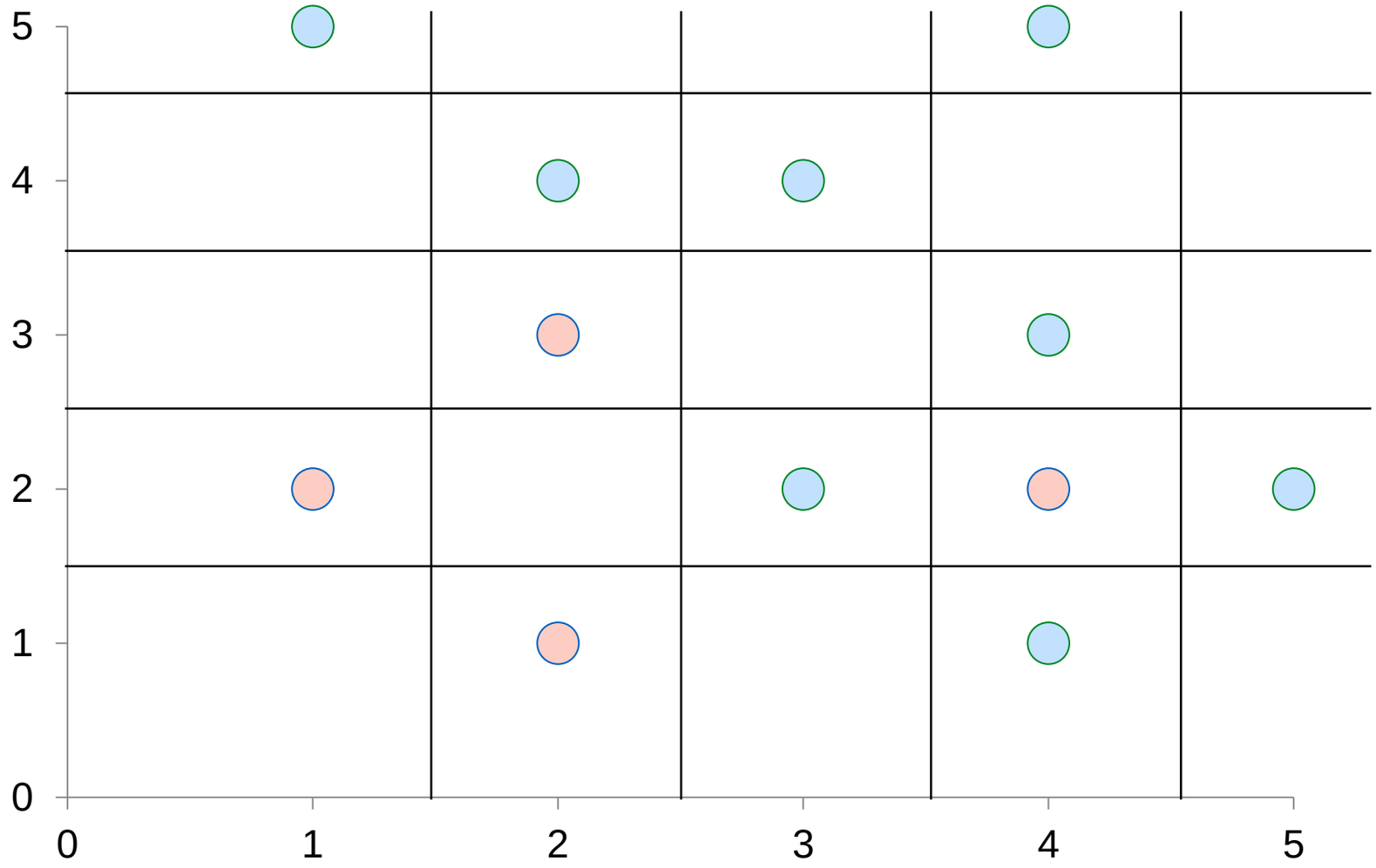
Encontrar la mejor división

```
findBestSplit(Dataset L)

1. Try all possible splits
2. return best
```

Pero ¿Cuál es la mejor división?

No hay demasiadas divisiones →Viable computacional-mente

O(no. atribs. x no. puntos)



Criterio de división

- Debe medir la impureza del nodo:

$$i(t) = \sum_{i=1}^{m} f_i(1 - f_i)$$ impureza Gini (CART)

donde f_i es la fracción de ejemplos de clase i en el nodo t y m es el número de clases

- La mejora que da una división es la variación de la impureza antes y después de la división

$$\Delta i(t, s) = i(t) - p_L i(t_L) - p_R i(t_R)$$

donde p_L (p_R) es la proporción de ejemplos que van al nodo izquierdo (derecho) tras la división

Criterio de división

$$\Delta i(t,s) = i(t) - p_L i(t_L) - p_R i(t_R)$$

$$i(t) = 2 \times \frac{8}{12} \times \frac{4}{12} = 0{,}44$$

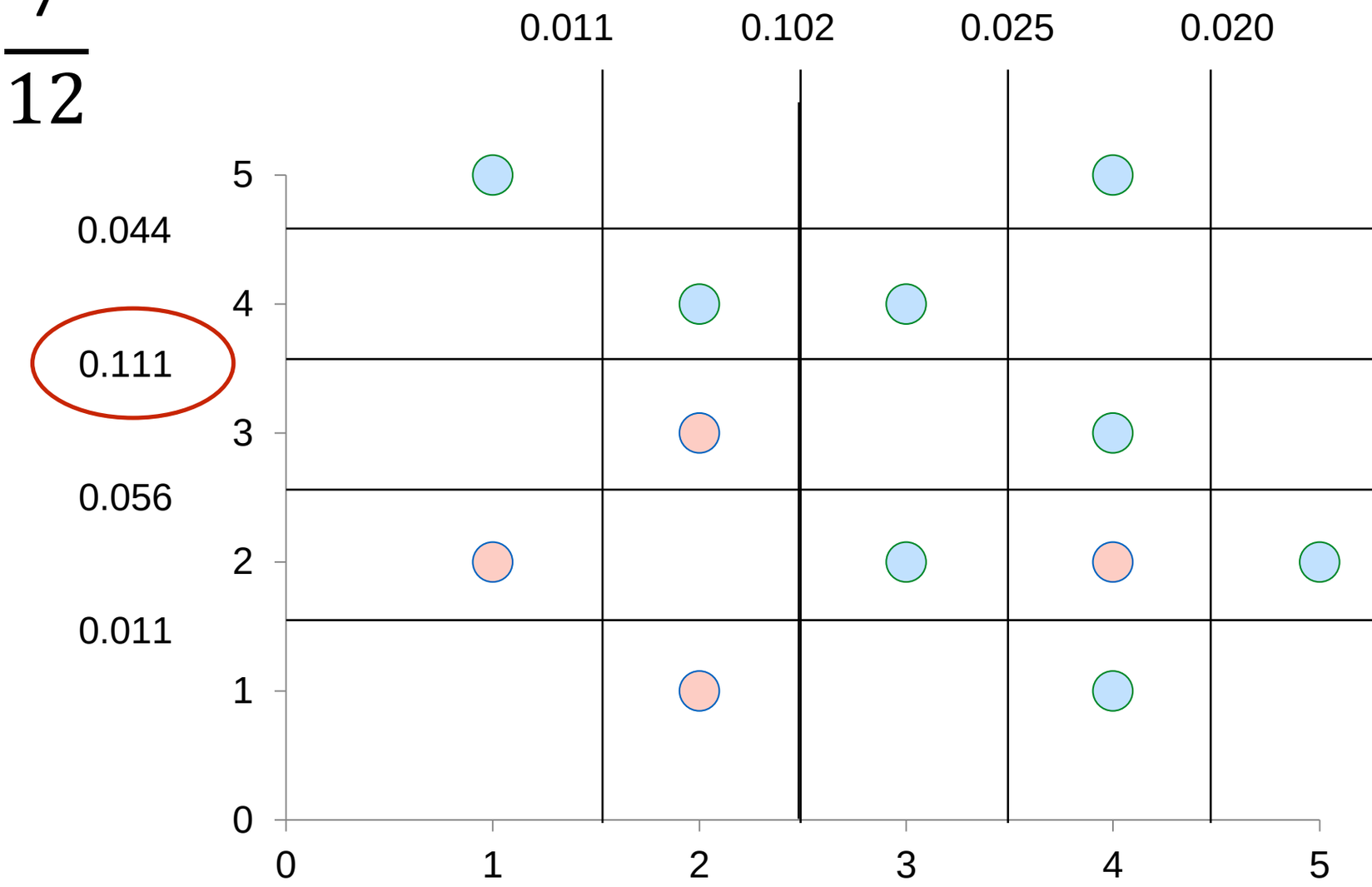
$$p_L = \frac{5}{12} \; ; \; p_R = \frac{7}{12}$$

$$i(t_L) = 2 \times \frac{2}{5} \times \frac{3}{5}$$

$$i(t_R) = 2 \times \frac{6}{7} \times \frac{1}{7}$$

$$\Delta i(t,s) = 0.102$$

¿Cuál es la mejor división?



Otros criterios

- Basados en entropía

$$H(t) = -\sum_{i=1}^{m} f_i \log_2 f_i$$

- Ganancia de información (IG), usado en ID3 y C4.5

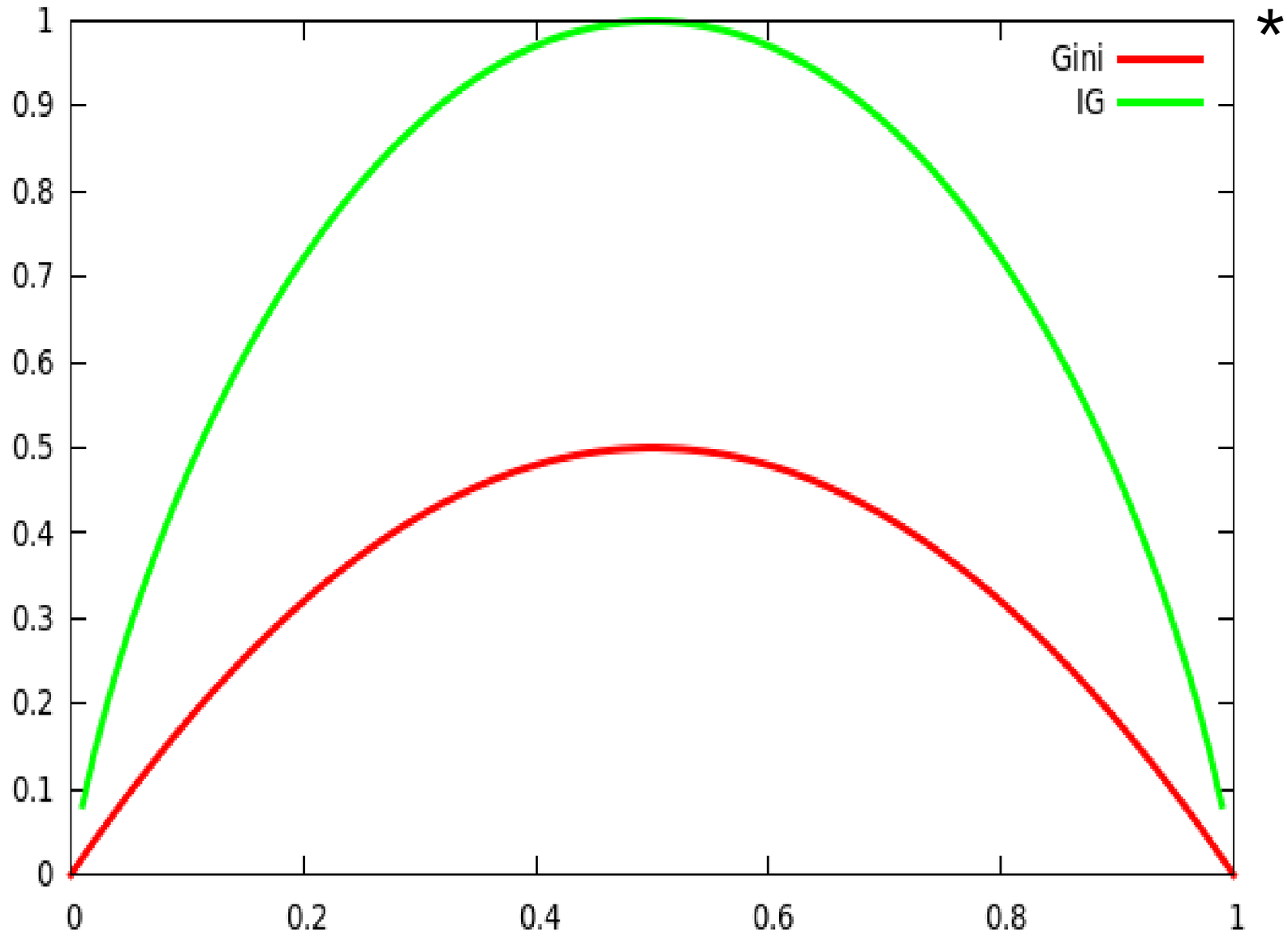
$$IG(t,s) = H(t) - p_L H(t_L) - p_R H(t_R)$$

- Ratio de ganancia de información (IGR) en C4.5

$$IGR(t,s) = IG(t,s)/H(s)$$

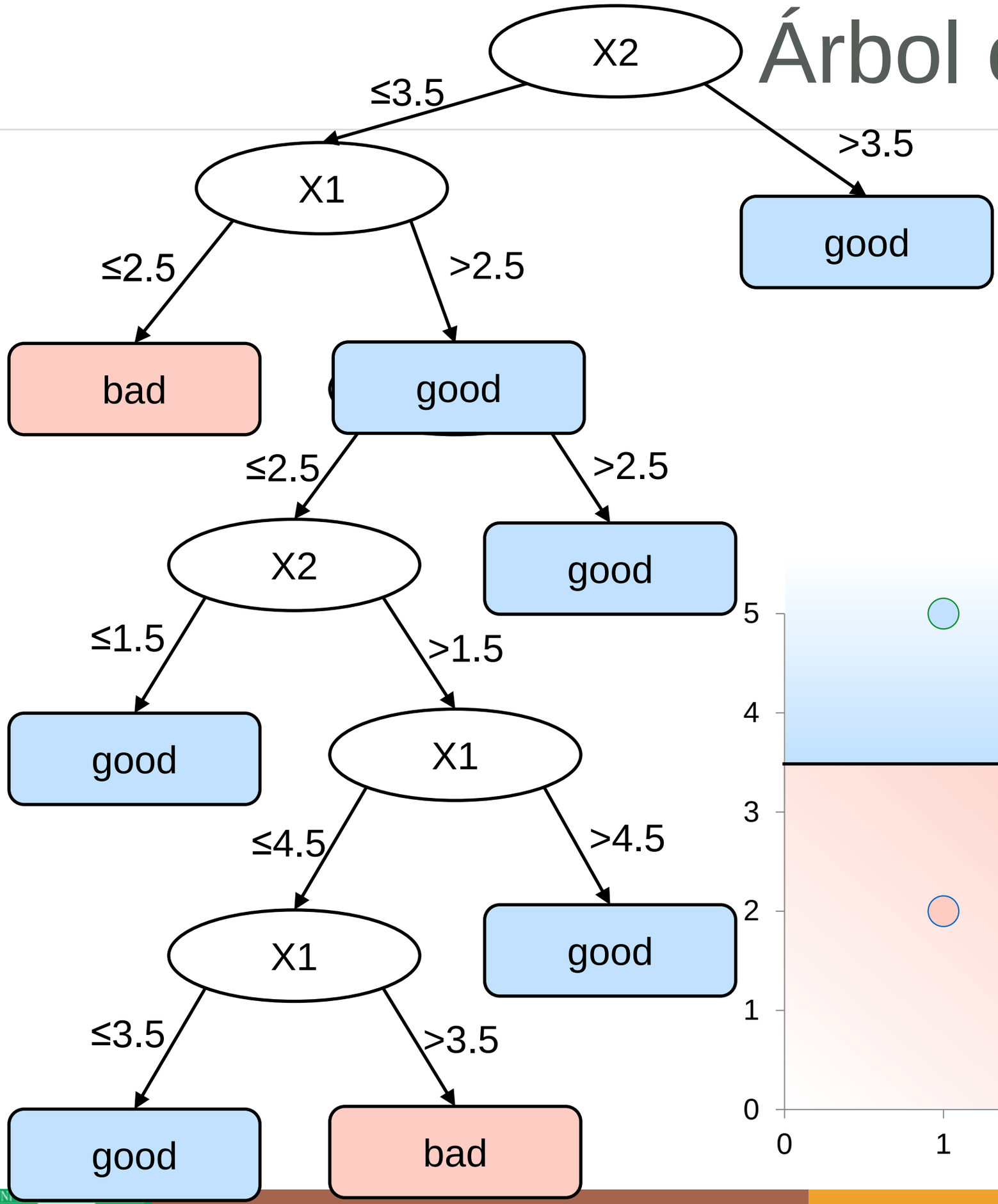
Criterio de división

- En general, cualquier criterio con esta forma es bueno

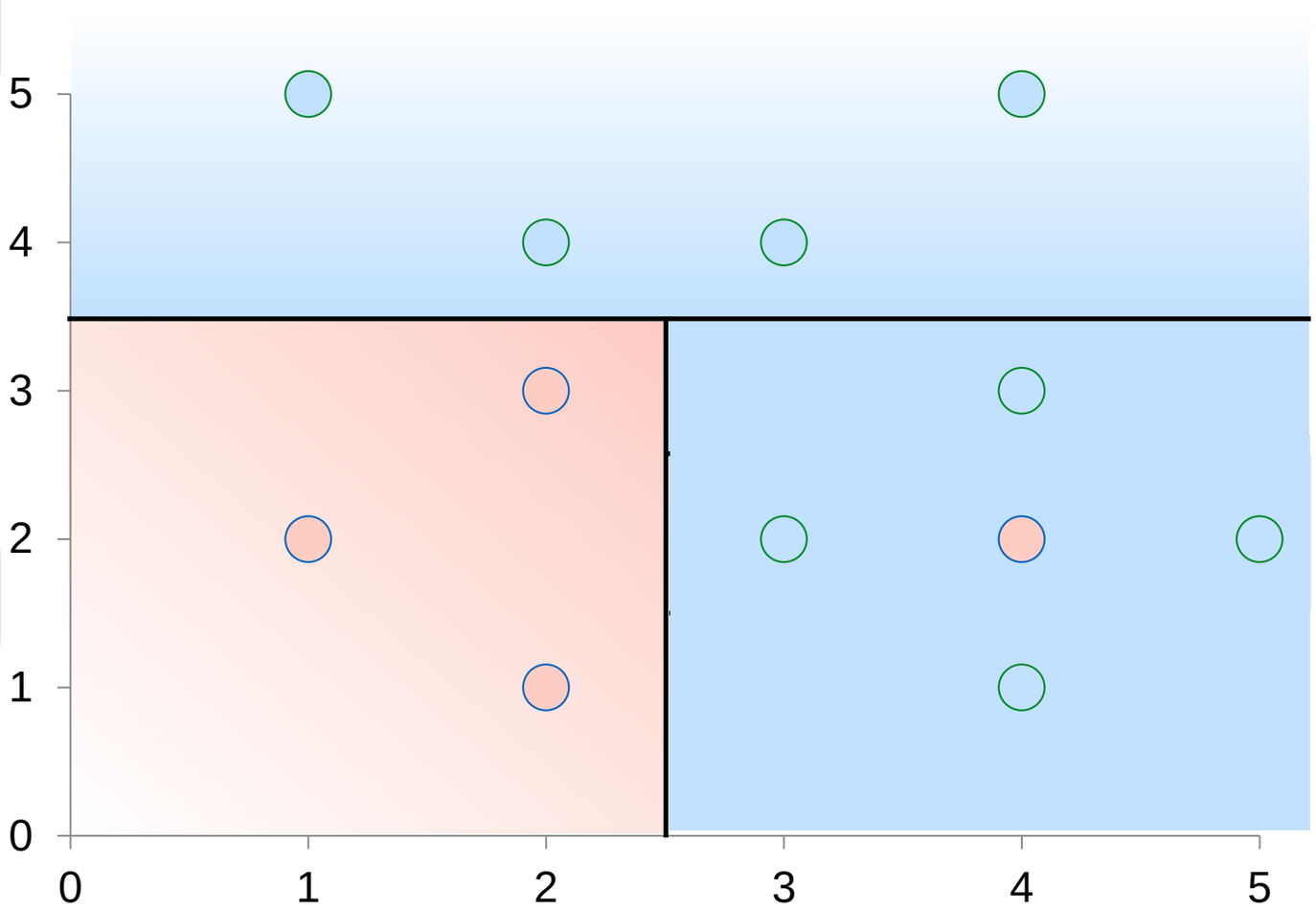


* Esto es para problemas binarios, es decir de dos clases

Árbol crecido al límite



¿Qué tal este otro?



Criterio de parada

- Todos los ejemplos asignados al nodo son de la misma clase.
- No se encuentra ninguna división para partir los datos.
- El número de ejemplos en el nodo es inferior a un número predefinido (p.e. si hay menos de 10 ejemplos no se buscan más divisiones)
- La ganancia de impureza en un nodo no está por debajo de un umbral predefinido.
- El árbol alcanza la profundidad máxima.

Estos tres últimos elementos se llaman también pre-poda

Poda

- Otra opción comúnmente utilizada es post-poda (o poda). Consiste en:
  - Crecer el árbol lo más posible.
  - Podar posteriormente sustituyendo sub-arboles por un único nodo a hoja si el error no empeora demasiado.
  - Este proceso continua hasta que no se puede podar más.
- Realmente estamos yendo hacia atrás pero por un camino distinto al usado para crecer el árbol
- La idea de la poda es evitar el sobre ajuste

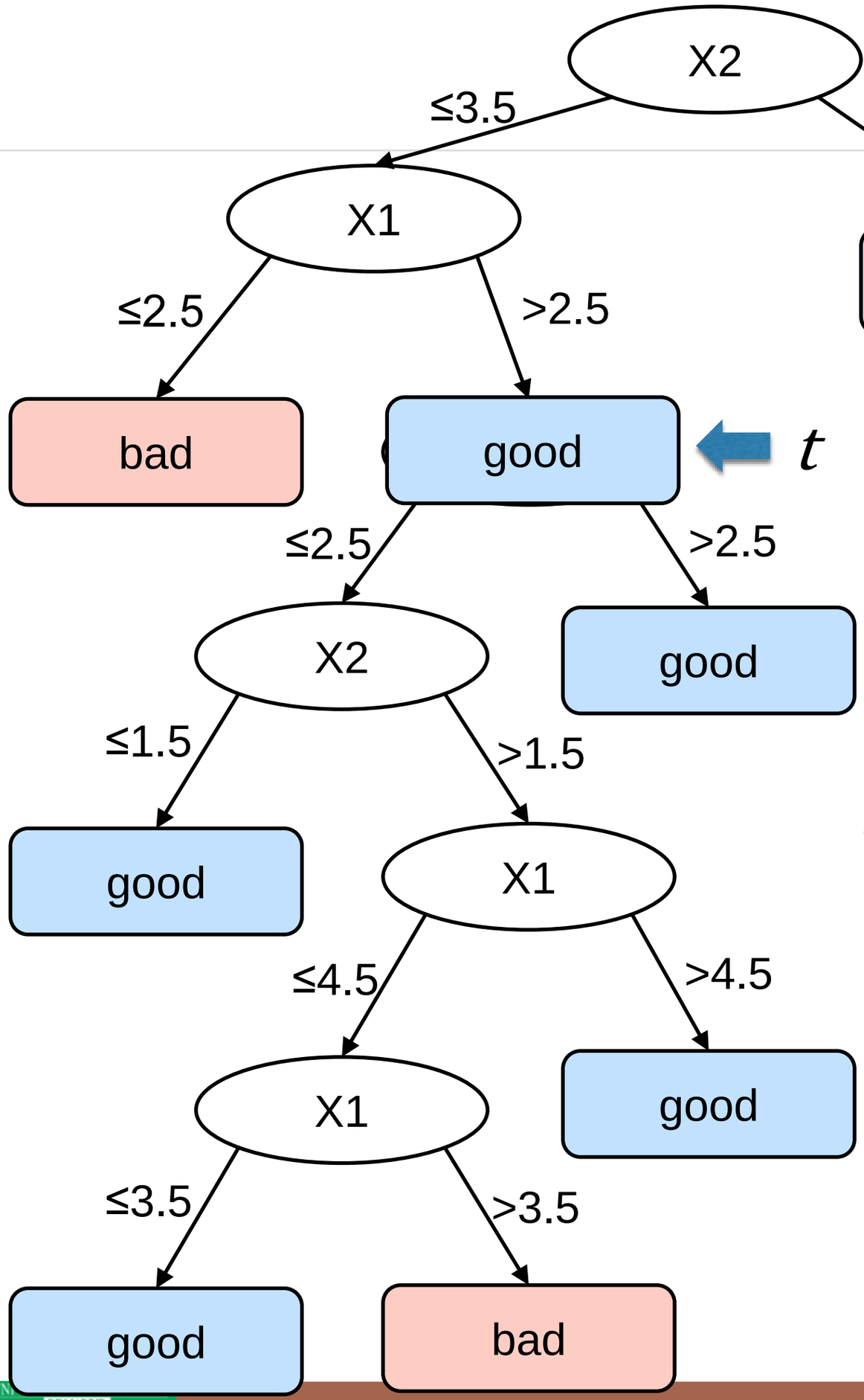
Cost-complexity pruning (CART)

- Poda de coste-complejidad:

$$R_\alpha(t) = R(t) + \alpha \cdot C(t)$$

  - R(t) es el error del árbol con raíz en el nodo t
  - C(t) es el número de hojas desde el node t
  - El parámetro α especifica el peso relativo entre la precisión y la complejidad del árbol

Poda con CART



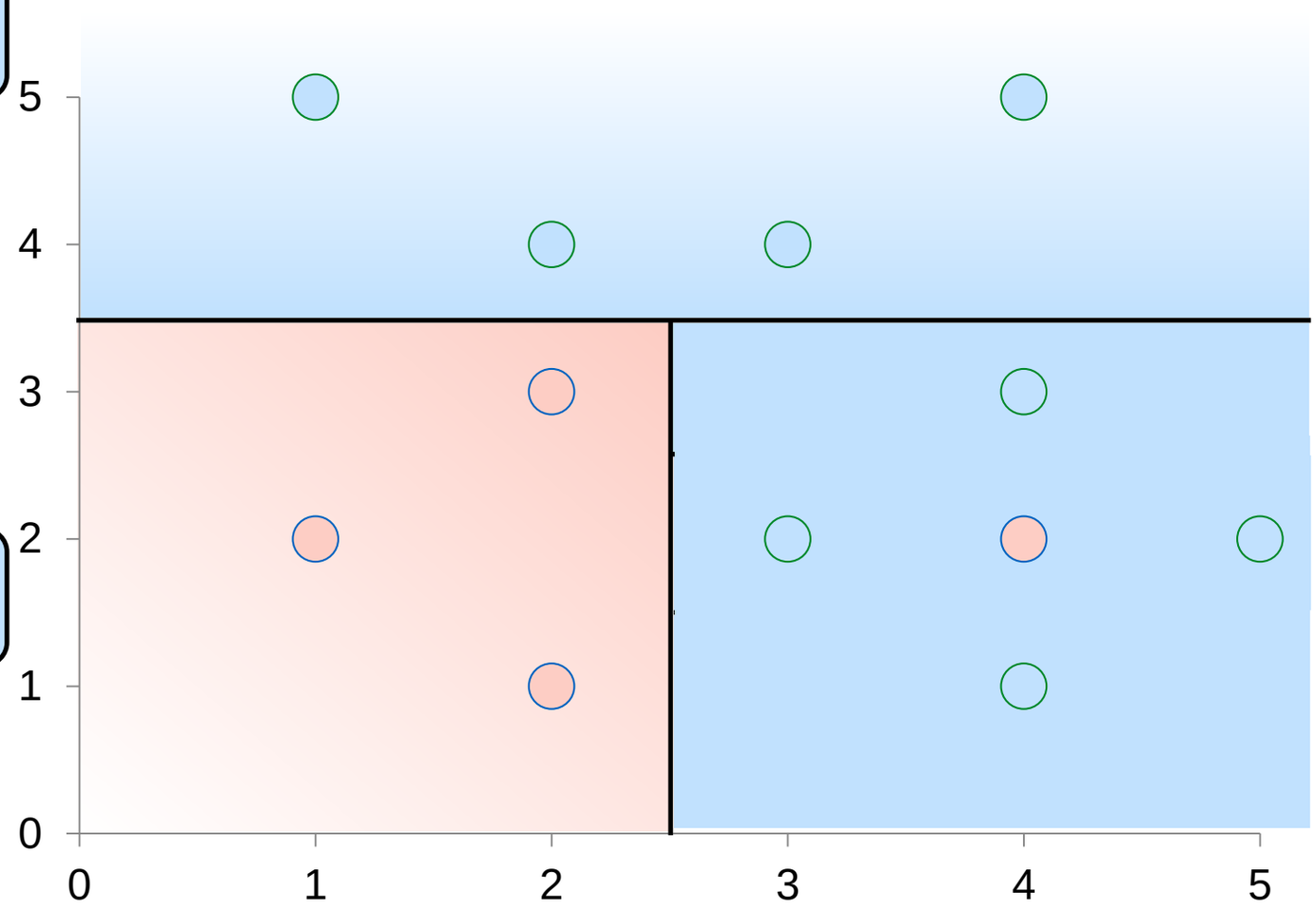
Ponemos $\alpha = 0.1$

Podado:

$$R_{\alpha=0.1}(t) = 1/5 + 0.1 \cdot 1 = 0.3$$

No podado

$$R_{\alpha=0.1}(t) = 0 + 0.1 \cdot 5 = 0.5$$



Cost-complexity pruning (CART)

- CART usa validación cruzada en 10 grupos en los datos de entrenamiento para estimar alfa. De forma iterativa 9 grupos se usan para entrenar un árbol y uno para probarlo.
- Un árbol es entrenado usando 9 grupos y se poda usando todos los posibles alfas (que son finitos).
- Cada árbol podado se prueba en el conjunto restante.
- El proceso se repite 10 veces y el valor de alfa que mejor error de generalización devuelve se utiliza para contruir el árbol final usando todos los datos

Statistical pruning (C4.5)

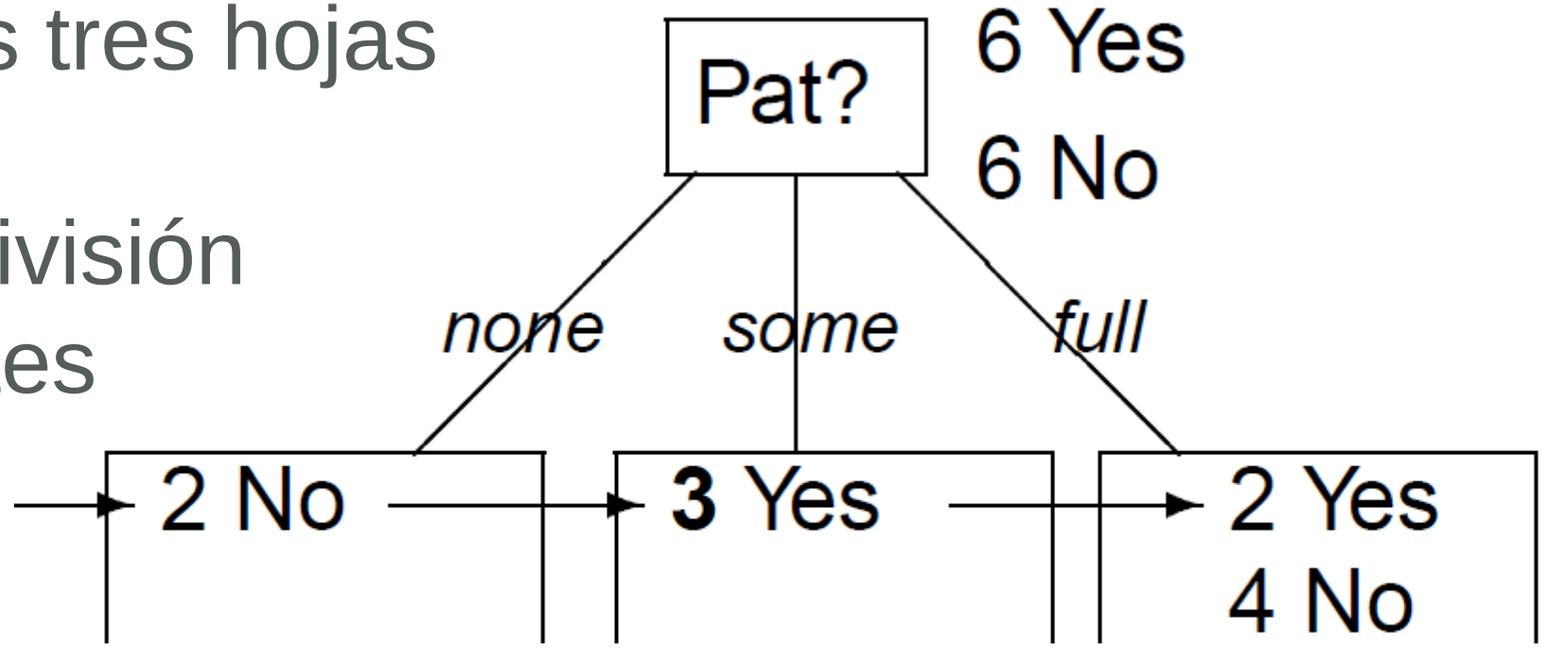
- C4.5 estima el error en las hojas usando el valor superior de confianza (es un parámetro) de una distribución normal en vez de usar el error directamente.
- El error de un sub-árbol es la suma ponderada de los errores de cada una de sus hojas
- Este error tiende a ser mayor cuanto menos ejemplos haya en una hoja.
- Por tanto, las hojas con pocas instancias tienden a ser podadas.

Poda (CART vs. C4.5)

- Poda de CART (cost-complexity) es más lenta ya que hay que construir 10 árboles extra para estimar alfa.
- La poda de C4.5 es más rápida. Sin embargo el algoritmo no propone cómo estimar el valor umbral para la confianza
- La base estadística de la poda en C4.5 es cuestionable.
- Utilizar validación cruzada es más seguro

Valores desconocidos (*Missing values*)

- ¿Cómo gestionan los árboles los *missing values*?
- Supongamos que el valor para “Pat” para una instancia es desconocido.
- Solución: el ejemplo con el valor desconocido baja ponderado por las tres hojas
- La validez de la división se calcula como antes



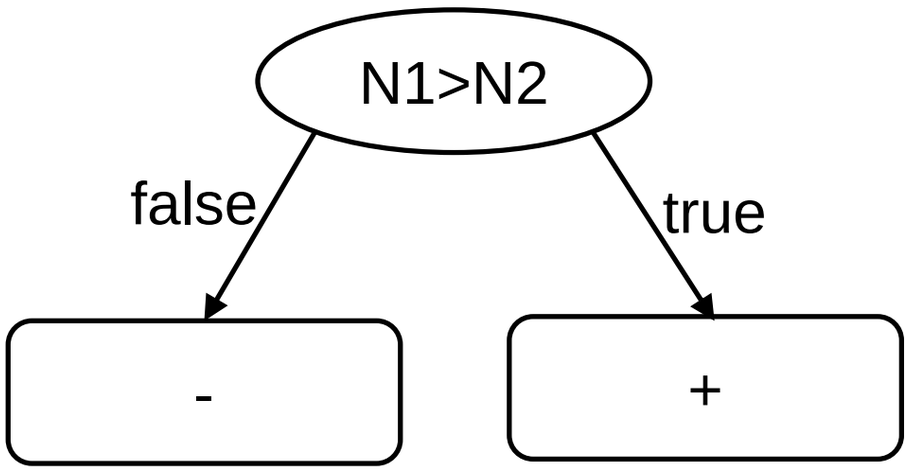
Divisiones oblicuas

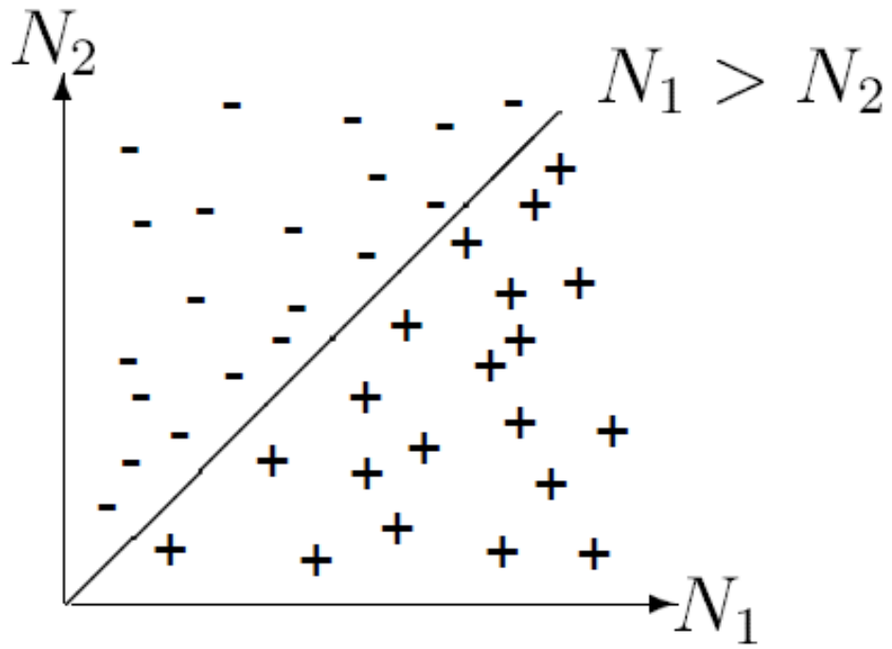
- El algoritmo CART permite hacer divisiones oblicuas, es decir, divisiones no ortogonales a los ejes de atributos.

- El algoritmo busca planos con buena reducción de impureza

- El crecimiento del árbol se ralentiza

- Pero los árboles son más expres





Parámetros

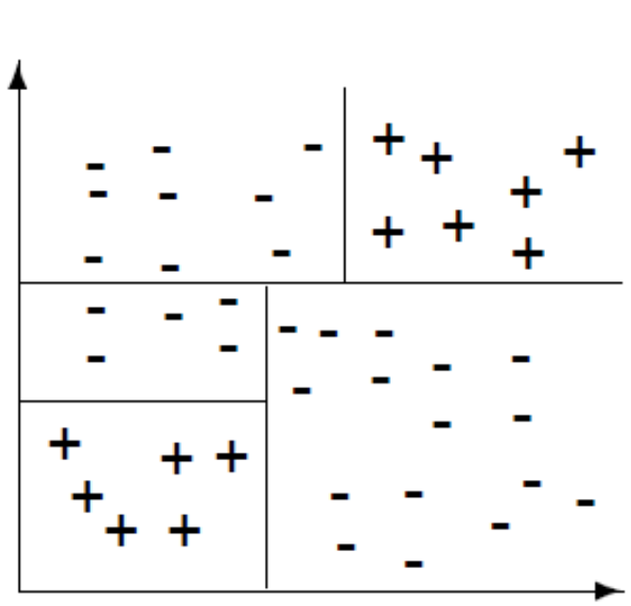
- Número mínimo de ejemplos necesarios para dividir una hoja.
- Aplicar o no poda
- Confianza en la poda ¿Cuñando podar?
- Si hay limitaciones de computacionales se puede limitar la profundidad o el número de nodos del árbol

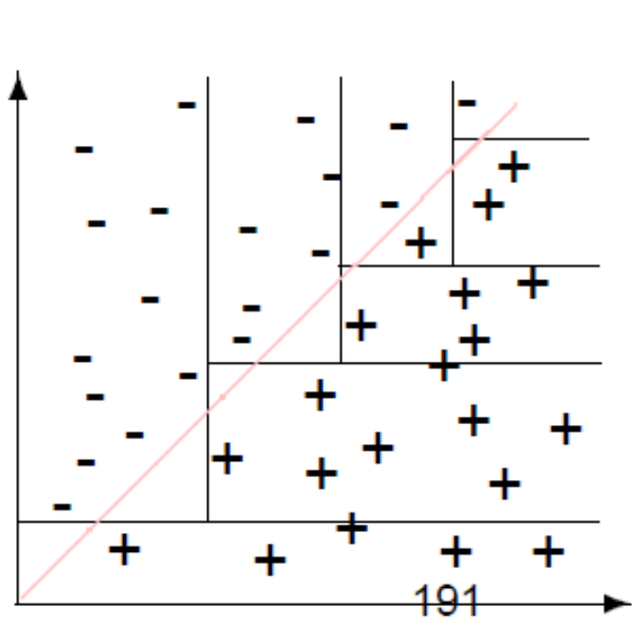
Detalles de los algoritmos

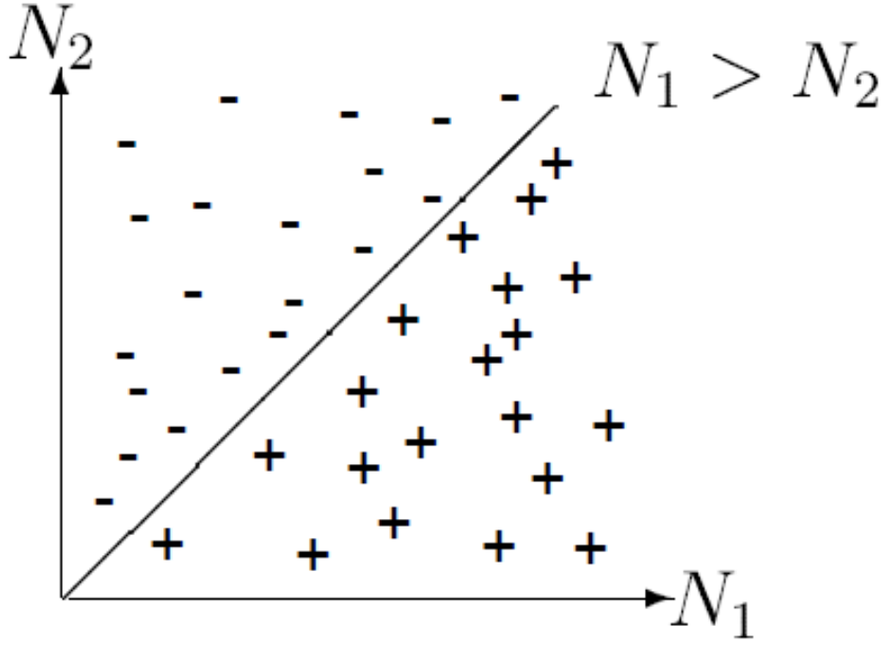
	Criterio de división	Criterio de poda	Otras características
CART	• Gini • Twoing	Cost-complexity post-poda	• Regresión/Clasif. • Atributos: categóricos /numéricos • *Missing values* • Divisiones oblicuas • Divisiones de atribs. categóricos por agrupaciones
ID3	Information Gain (IG)	Pre-poda	• Clasificación • Atributos categóricos
C4.5	• Information Gain (IG) • Information Gain Ratio (IGR)	Post-poda estadística	• Clasificación • Atributos: categóricos /numéricos • *Missing values* • Generador de reglas • Divisiones múltiples.

Desventajas de usar árboles

- ¡Ninguna! Bueno alguna sí…
- No manejan bien interacciones complejas entre atributos. Falta de poder expresivo

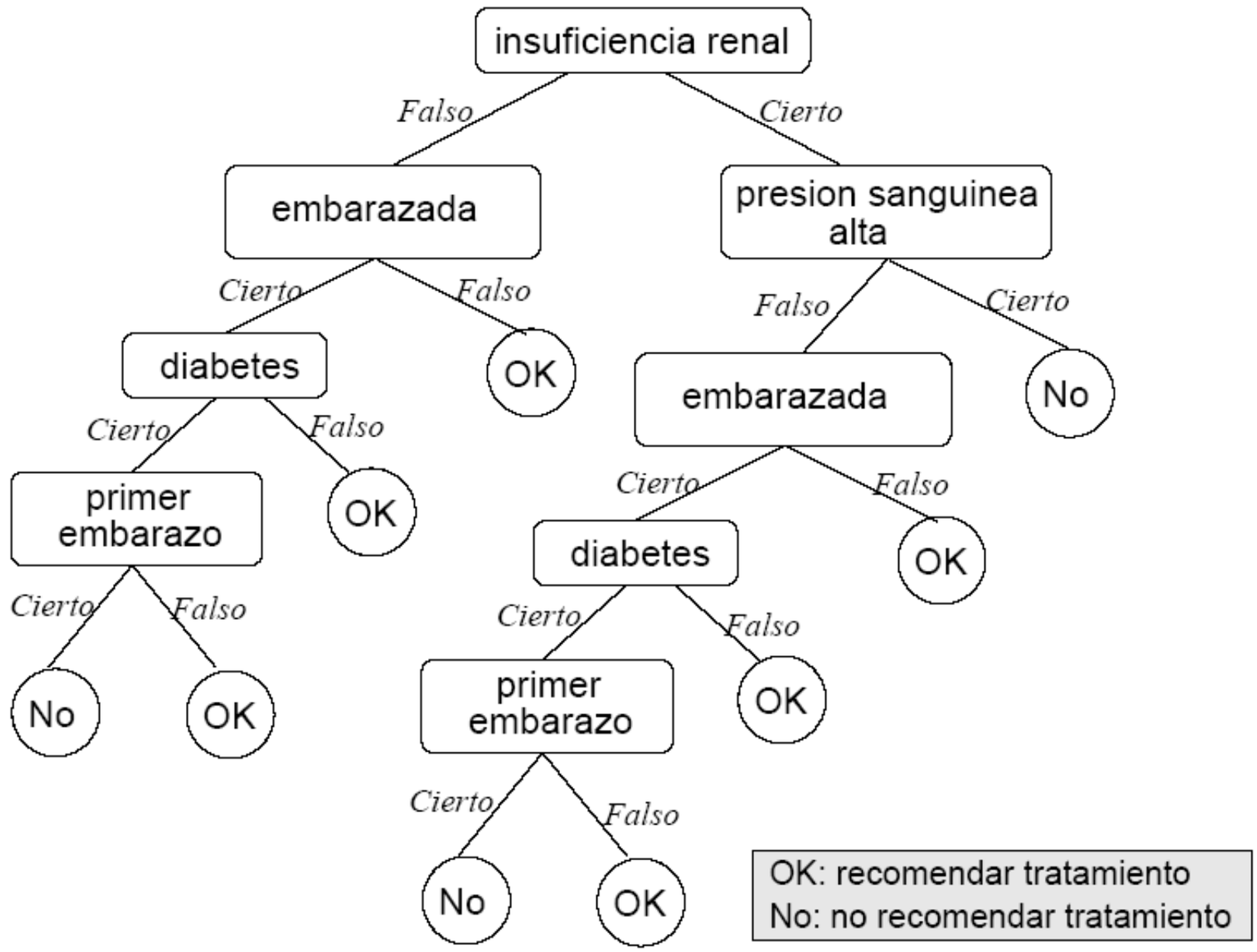






Desventajas de usar árboles

- Problema de replicación. Se puede acabar con árboles similares es regiones excluyentes



Ventajas de usar árboles

- Interpretables. Fáciles de entender para no expertos. Se pueden convertir a reglas.
- Manejan atributos nominales y numéricos.
- Gestionan bien atributos no informativos o redundantes.
- Pueden trabajar con *missing values*.
- Método no paramétrico. No hay una idea predefinida sobre el concepto a aprender
- Fácil de hacer el ajuste de parámetros. Tienen muy pocos parámetros

Árboles en sklearn I

Modelos:

```
DecisionTreeClassifier(criterion='gini', splitter='best', max_depth=None,
    min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
    max_features=None, random_state=None, max_leaf_nodes=None,
    min_impurity_split=1e-07, class_weight=None, presort=False)

DecisionTreeRegressor(criterion='mse', splitter='best', max_depth=None,
    min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
    max_features=None, random_state=None, max_leaf_nodes=None,
    min_impurity_split=1e-07, presort=False)
```

Librería:

from sklearn import tree

Ejemplo:

```
from sklearn import tree
# Crear un árbol con los parámetros por defecto
clf = tree.DecisionTreeClassifier()
```

Árboles en sklearn II

Parámetros de tree:

- **criterion**: Criterio de división.
  - Clasificación (por omisión="gini"): Puede ser “gini” o “entropy”
  - Regrsesión (por omisión=“mse"): Puede ser “mse” (error cuadrático medio) o “mae” (error absoluto medio)

Criterios de prepoda

- **min_samples_leaf** (=1): Número mínimo de ejemplos que debe haber en cada hoja creada.
- **max_depth** (=None): Límite a la profundidad máxima del árbol.
- **min_samples_split** (=2), **min_weight_fraction_leaf** (=0.0), **max_leaf_nodes** (=None), **min_impurity_split** (=1e-07): Otros criterios de prepoda

Árboles en sklearn III

Parámetros de tree:

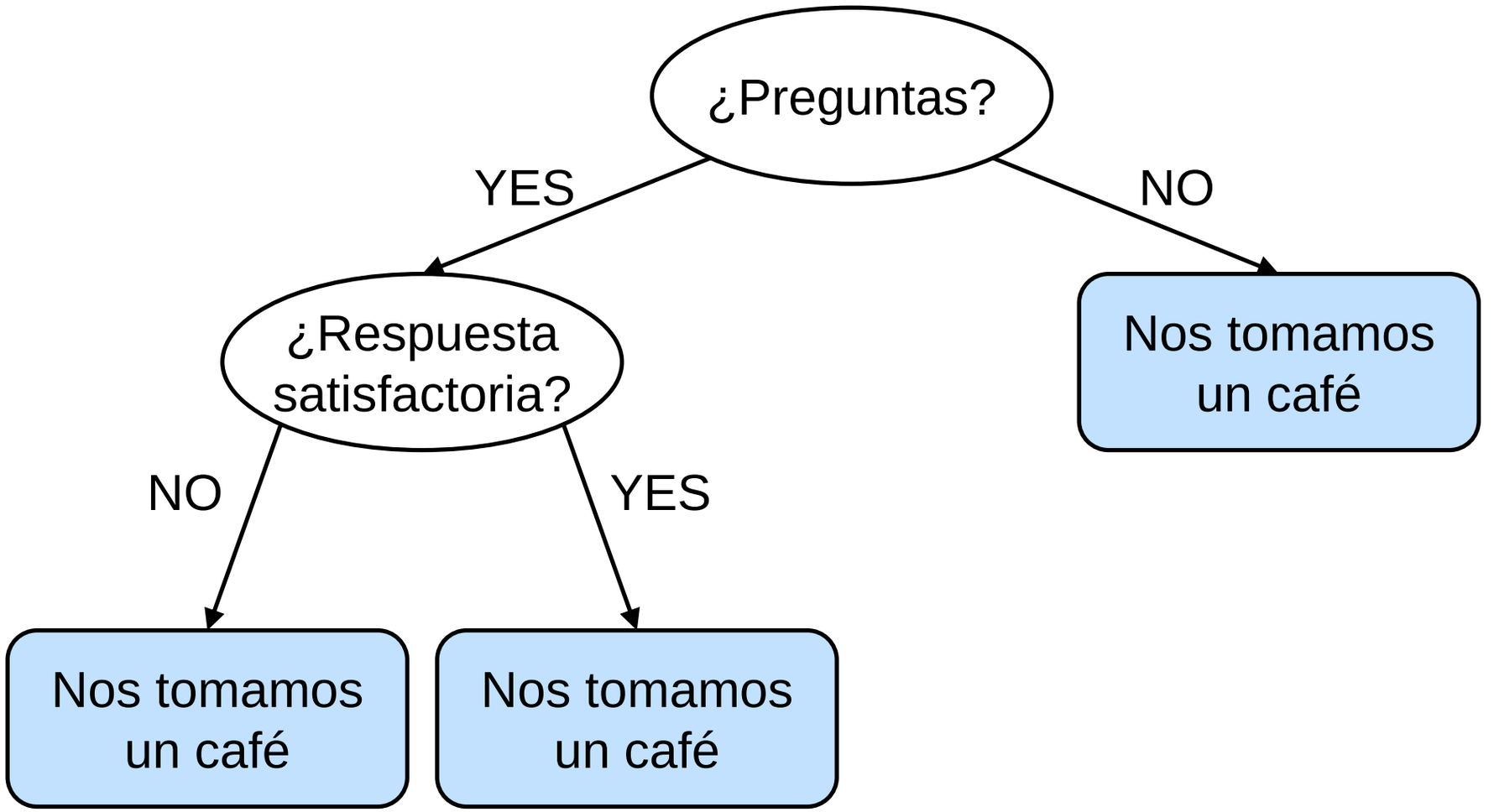
- **max_features** (=None): Número de atributos sobre el que se extrae la mejor división. Otros valores pueden ser: ‘sqrt’, ‘auto’, ‘log’. Esto se usa junto con Random Forest generalmente
- **random_state** (=None): Semilla aleatoria que determina las operaciones aleatorias. Para una misma semilla y datos sale en mismo árbol.
- **[Solo clasificación] class_weight** (=None): Modificar la importancia de cada clase. Útil cuando tenemos conjuntos desequilibrados. Se usa poniendo ‘balanced’

```
Ejemplo: Crear un modelo usando IG  y un mínimo de 10 ejemplos por hoja

 clf = tree.DecisionTreeClassifier(criterion="entropy", min_samples_leaf=10)
```

Ejemplo de clasificación en sklearn

```
# Cargar de datos
fP = 'pimaND.csv'
d  = pd.read_csv(fP, sep=',')

# Creamos las particiones en entrenamiento y test para probar el modelo
#    (pe 10-fold cross validation) (version 0.18 de sklearn)
indexFolds = StratifiedKFold(n_folds=10, shuffle=True, random_state=11)

# Creamos clasificador
clf = tree.DecisionTreeClassifier()

aciertos= []
# Bucle que recorre las particiones
for idxTr, idxTs in indexFolds.split(d.values):
    # Entrenar modelo en datos de train
    clf.fit(d.values[idxTr,:-1],d.values[idxTr,-1])
    # Validar modelo en datos de test
    acierto = clf.score(d.values[idxTs,:-1],d.values[idxTs,-1])
    aciertos.append(acierto)

#  Salida
aciertos = np.array(aciertos)
print "%0.3g%%" % (100*aciertos.mean()) + " +- %.3g" % (100*aciertos.std())
```

Fin


¿Preguntas?
YES
NO
¿Respuesta satisfactoria?
Nos tomamos un café
NO
YES
Nos tomamos un café
Nos tomamos un café