MINERÍA DE DATOS

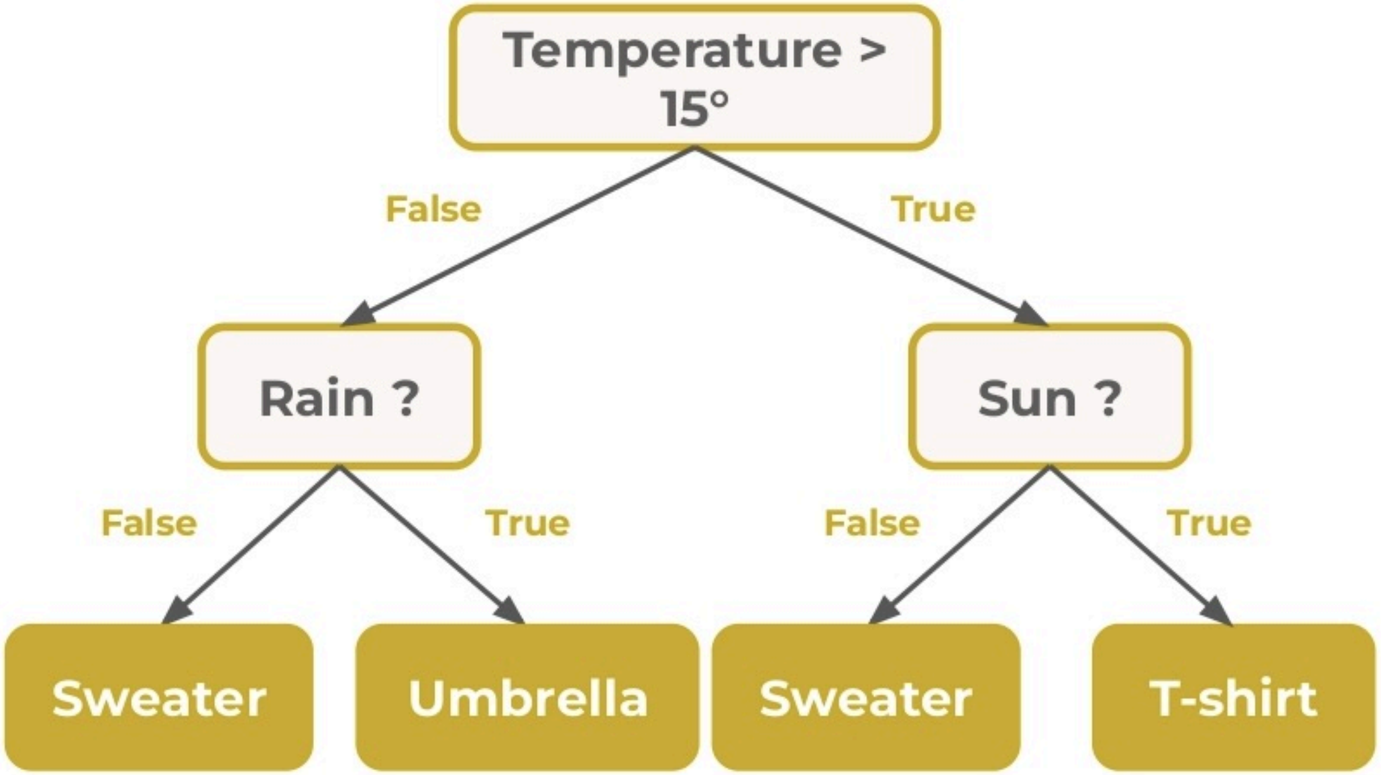
Maximiliano Ojeda

muojeda@uc.cl

IIC-2433

Árboles de Decisión

Árboles de Decisión



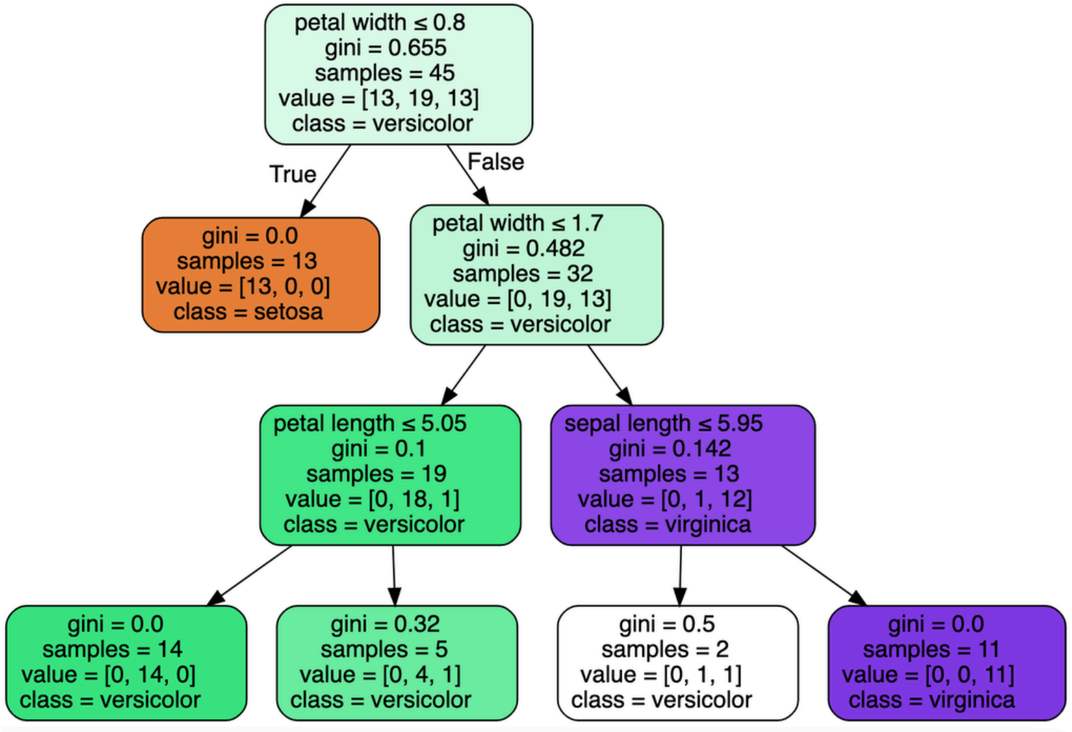
Árboles de Decisión

Los nodos del árbol representan variables, las ramas representan valores de las variables que permiten clasificar

Las hojas del árbol corresponden a la clasificación

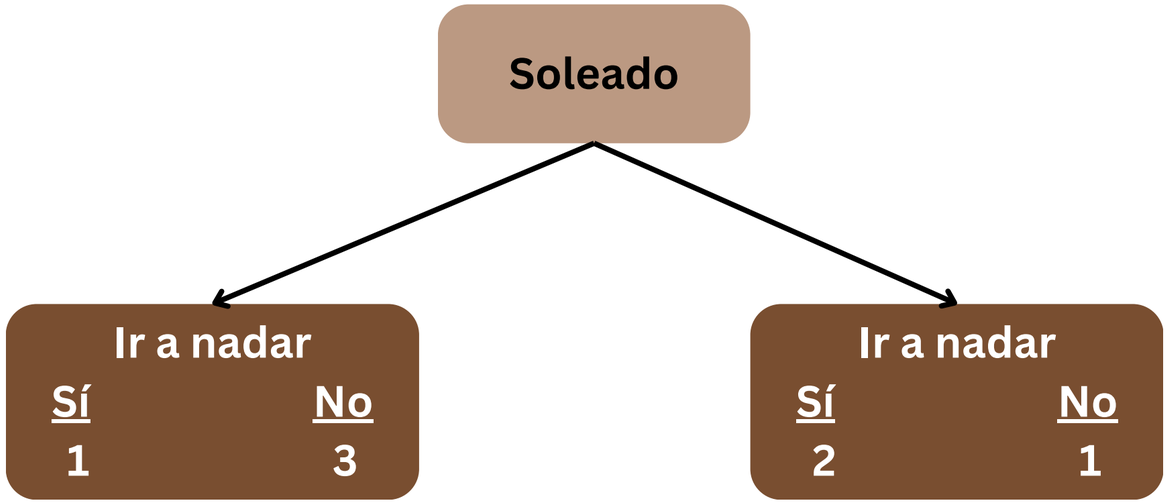
El **algoritmo selecciona la característica que mejor separa los datos**. Se usan métricas como:

- Gini (impureza Gini)
- Entropía (enfoque de la teoría de la información)



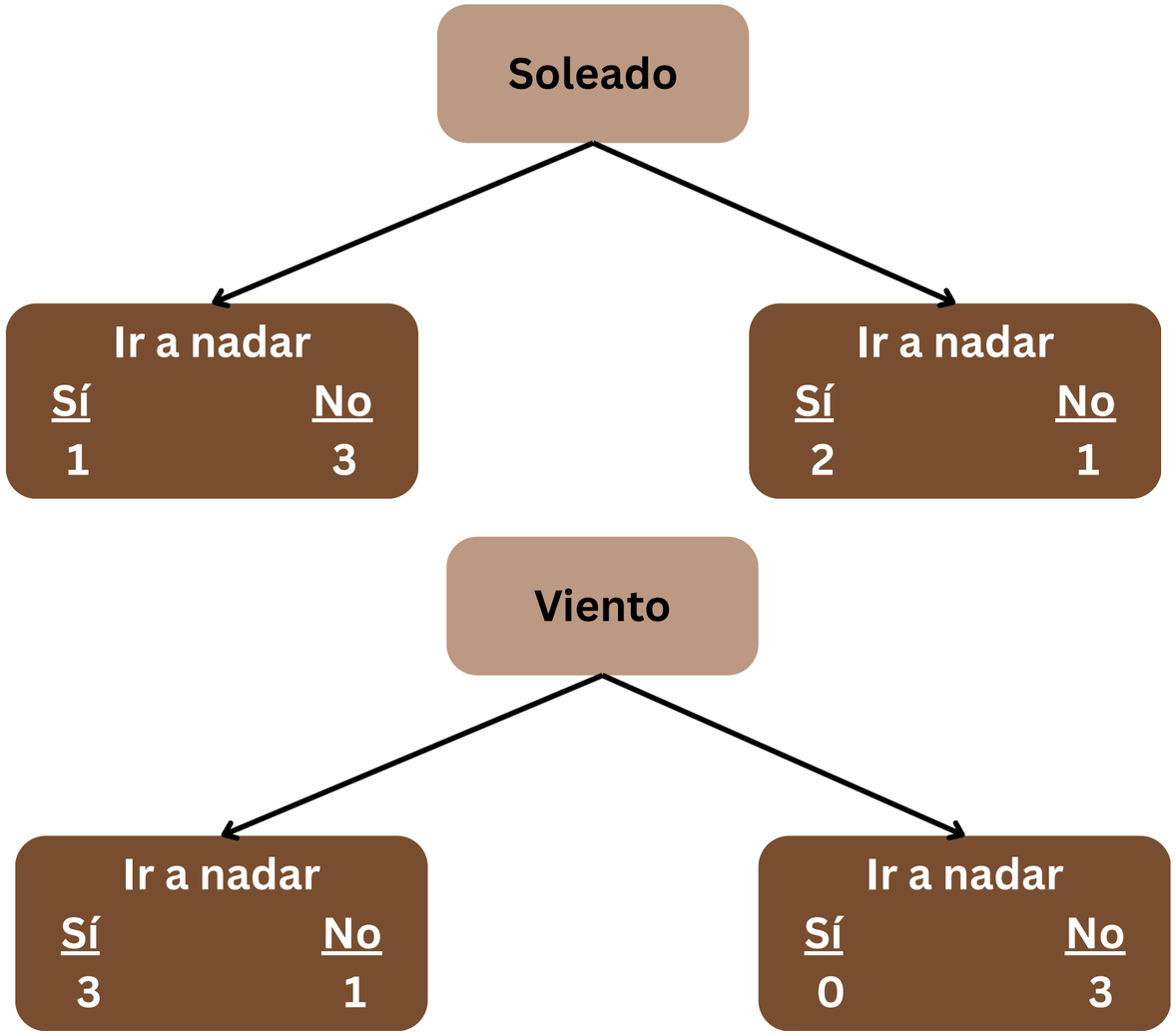
Árboles de Decisión

Soleado	Viento	Edad	Ir a nadar
Sí	Leve	23	No
Sí	Fuerte	14	No
No	Leve	30	Sí
No	Leve	16	Sí
Sí	Leve	27	Sí
Sí	Fuerte	18	No
No	Fuerte	20	No



Árboles de Decisión

Soleado	Viento	Edad	Ir a nadar
Sí	Leve	16	No
Sí	Fuerte	14	No
No	Leve	23	Sí
No	Leve	20	Sí
Sí	Leve	26	Sí
Sí	Fuerte	32	No
No	Fuerte	27	No



Métricas de “pureza”

Impureza Gini

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

- Más rápida de calcular (no usa logaritmos).
- Favorece la clase mayoritaria un poco más que la entropía.

Entropía (Information Gain)

$$H = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

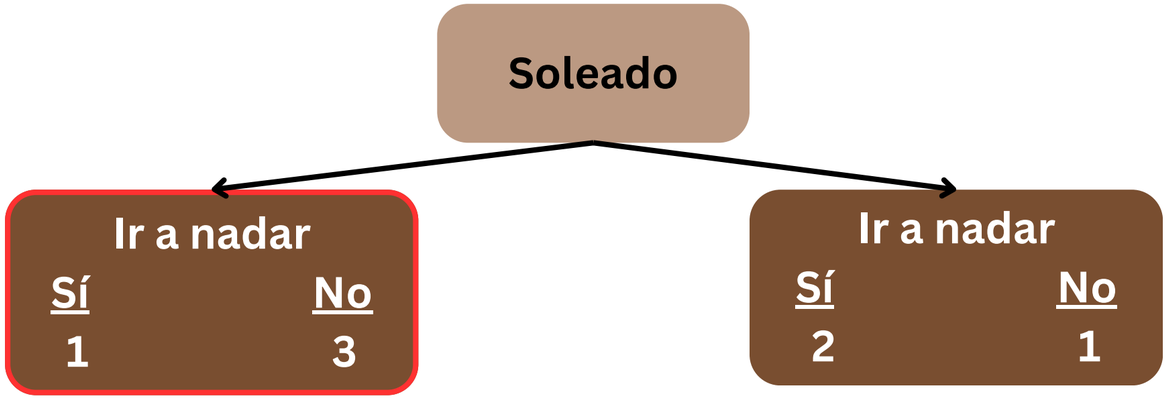
- Se basa en teoría de la información.
- Da una visión más “matemática” de la incertidumbre.

Métricas de “pureza”

Impureza Gini

$$Gini = 1 - \sum_{i=1}^{C} p_i^2$$

- Más rápida de calcular (no usa logaritmos).
- Favorece la clase mayoritaria un poco más que la entropía.



La **impureza de Gini para la hoja A** es:

$$\text{Gini}_A = 1 - (\text{prop. clase “Si”})^2 - (\text{prop. clase “No”})^2$$

$$\text{Gini}_A = 1 - (\frac{1}{4})^2 - (\frac{3}{4})^2$$

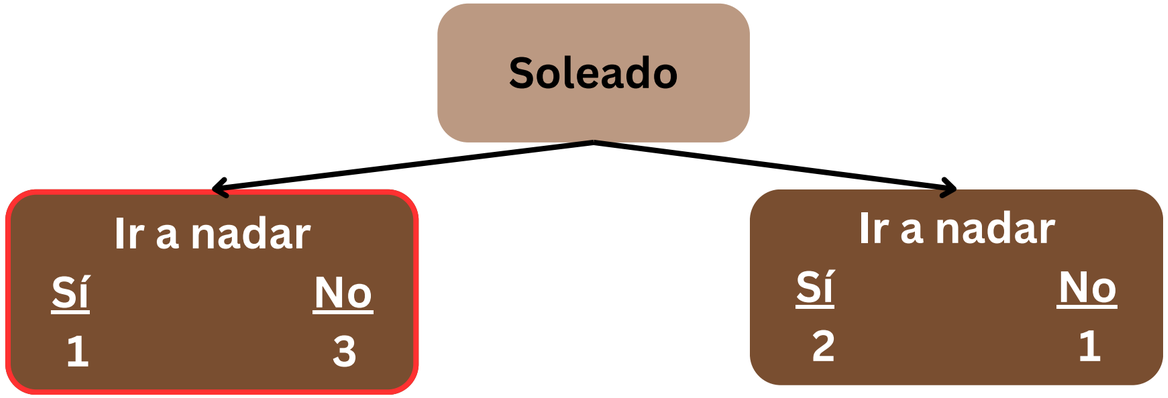
$$\text{Gini}_A = 0.375$$

De forma similar para la hoja B es: $\text{Gini}_B = 0.444$

La impureza de Gini total considera una ponderación:

$$\text{Gini} = \frac{4}{7} \times 0.375 + \frac{3}{7} \times 0.444 = 0.405$$

Métricas de “pureza”



$$H_A = -\left(0.25 \log_2 0.25 + 0.75 \log_2 0.75\right)$$

$$H_A = -\left(0.25 \times (-2) + 0.75 \times (-0.415)\right)$$

$$H_A = 0.5 + 0.311 = 0.811$$

De forma similar para la hoja B es: $H_B = 0.918$

La entropía total ponderada es:

$$H_{total} = \frac{4}{7} \times 0.811 + \frac{3}{7} \times 0.918 = 0.856$$

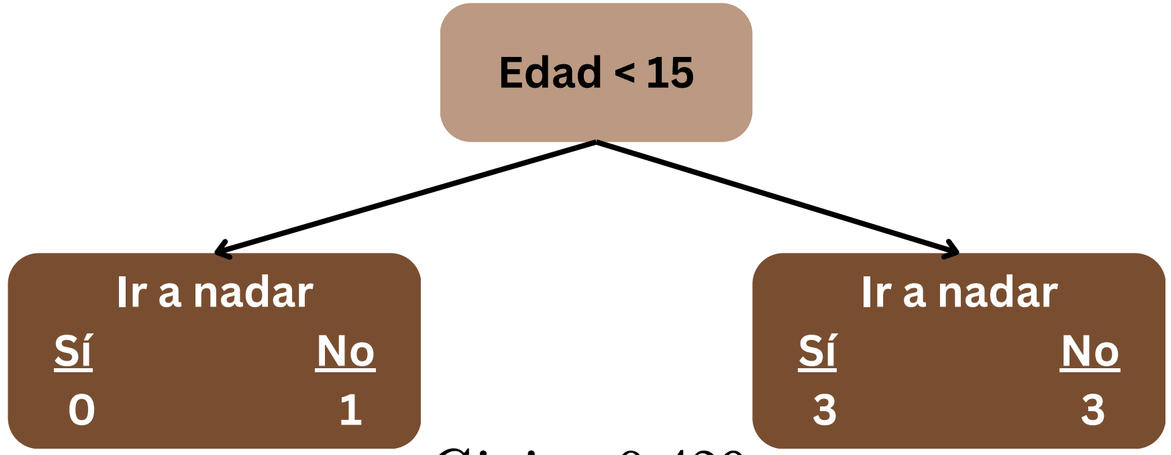
Entropía (Information Gain)

$$H = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

- Se basa en teoría de la información.
- Da una visión más “matemática” de la incertidumbre.

Datos numéricos

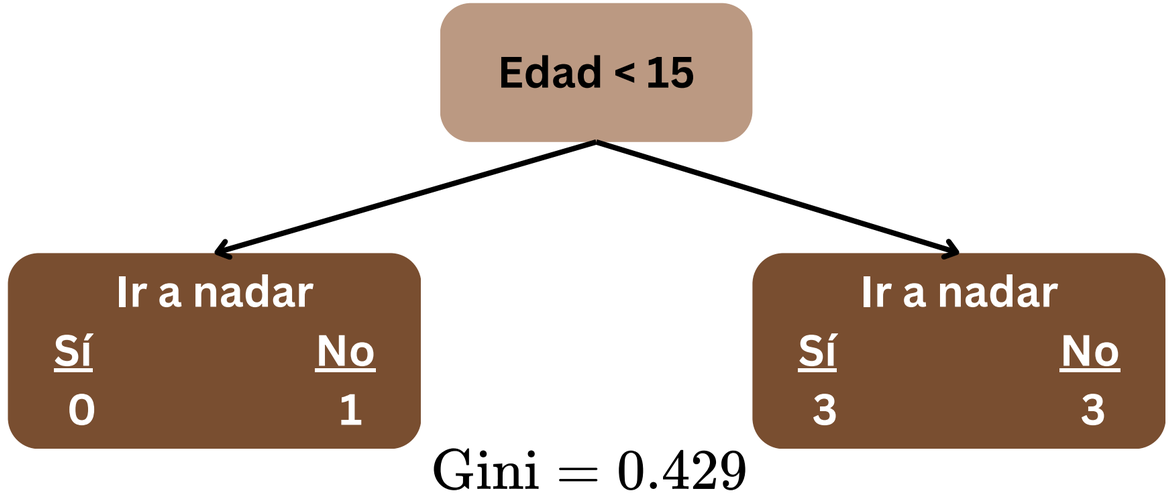
Edad	Ir a nadar
14	No
16	No
20	Sí
23	Sí
26	Sí
27	No
32	No



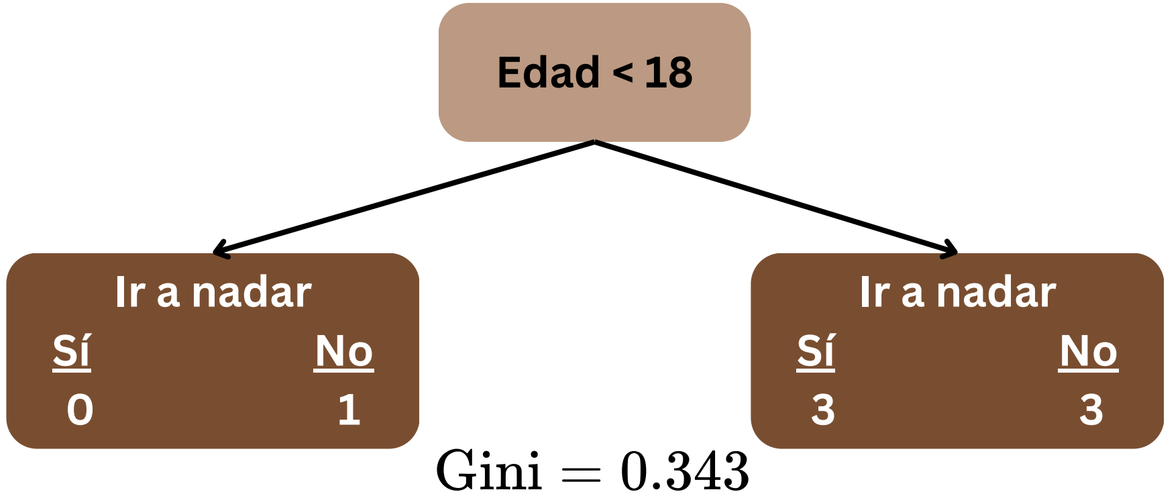
$$\text{Gini} = 0.429$$

Datos numéricos

Edad	Ir a nadar
14	No
16	No
20	Sí
23	Sí
26	Sí
27	No
32	No



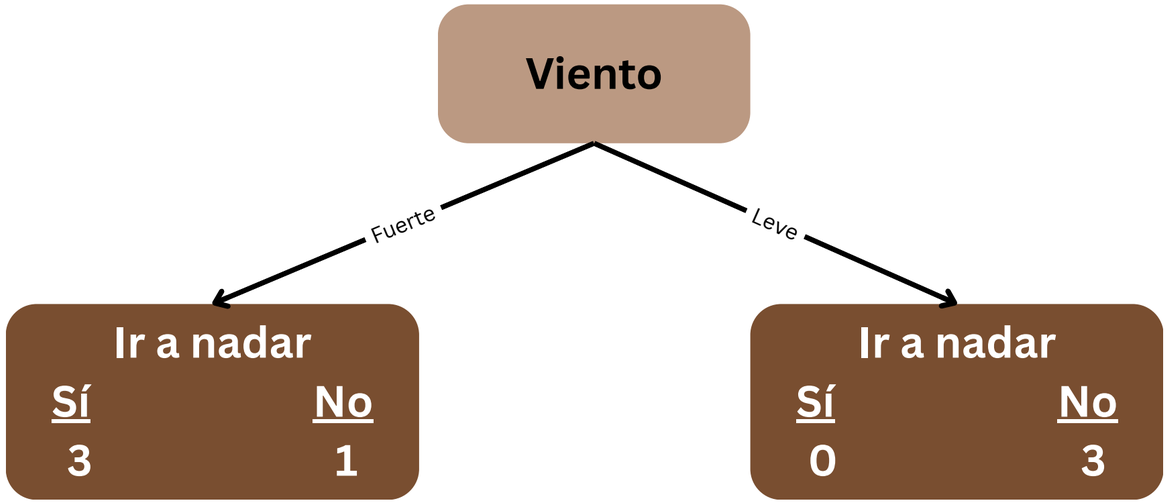
$$\text{Gini} = 0.429$$



$$\text{Gini} = 0.343$$

Construir Árbol de Decisión

Soleado	Viento	Edad	Ir a nadar
Sí	Leve	23	No
Sí	Fuerte	14	No
No	Leve	30	Sí
No	Leve	16	Sí
Sí	Leve	27	Sí
Sí	Fuerte	18	No
No	Fuerte	20	No



$$\text{Soleado: Gini} = 0.405$$

$$\text{Viento: Gini} = 0.214$$

$$\text{Edad: Gini} = 0.343$$

Árboles de Decisión

Parámetros importantes

- **criterion**: métrica de impureza
- **max_depth**: límite de profundidad
- **min_samples_split**: mínimo de muestras para dividir un nodo
- **min_samples_leaf**: mín. muestras en una hoja

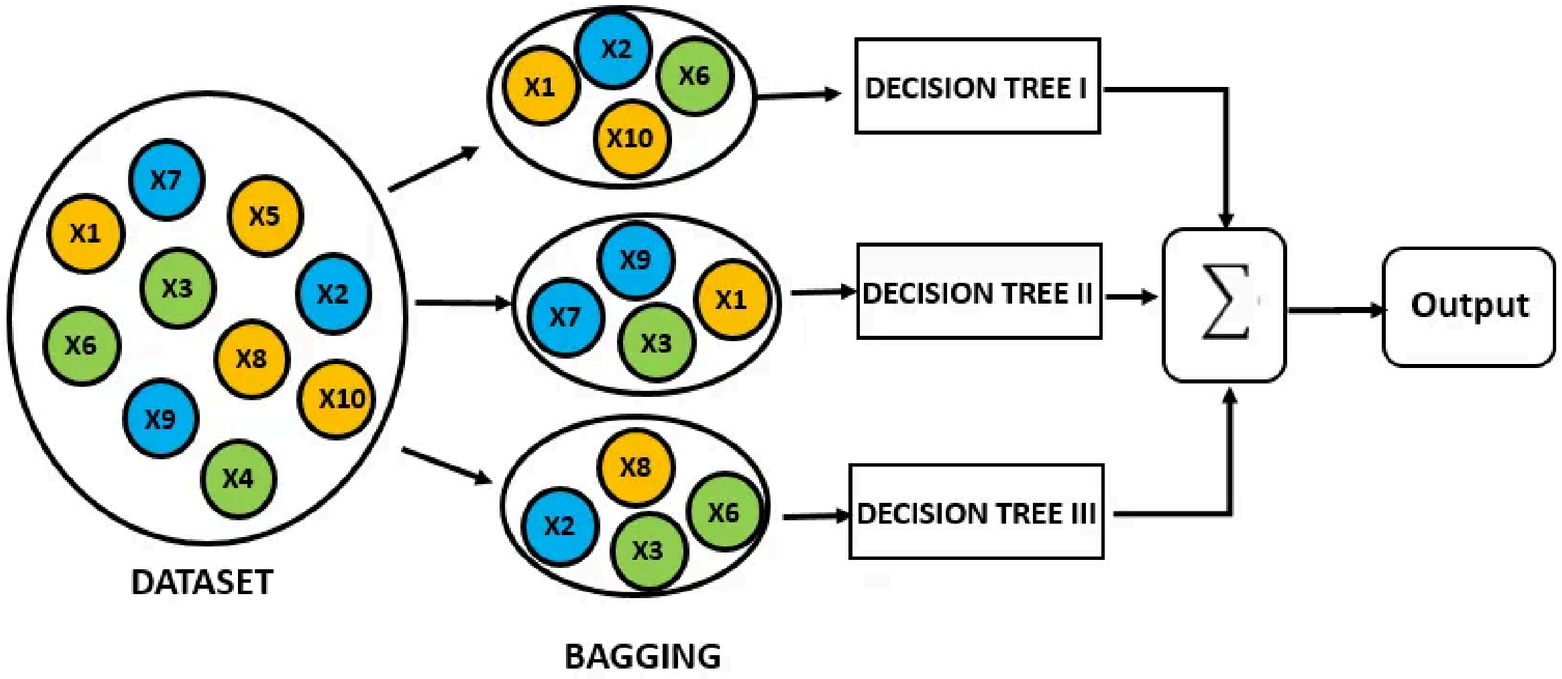
```python
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import classification_report, confusion_matrix

# Datos de ejemplo
X, y = make_classification(n_samples=800)
X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=0.20)

# Árbol de clasificación
clf = DecisionTreeClassifier(
    criterion="gini",        # o "entropy"
    max_depth=None,          # ayuda a evitar overfitting
    min_samples_split=2,
    min_samples_leaf=1,
    random_state=42
)
clf.fit(X_tr, y_tr)

print("Accuracy (train):", clf.score(X_tr, y_tr))
print("Accuracy (test): ", clf.score(X_te, y_te))
print(classification_report(y_te, clf.predict(X_te)))
```

Random Forest

Random Forest

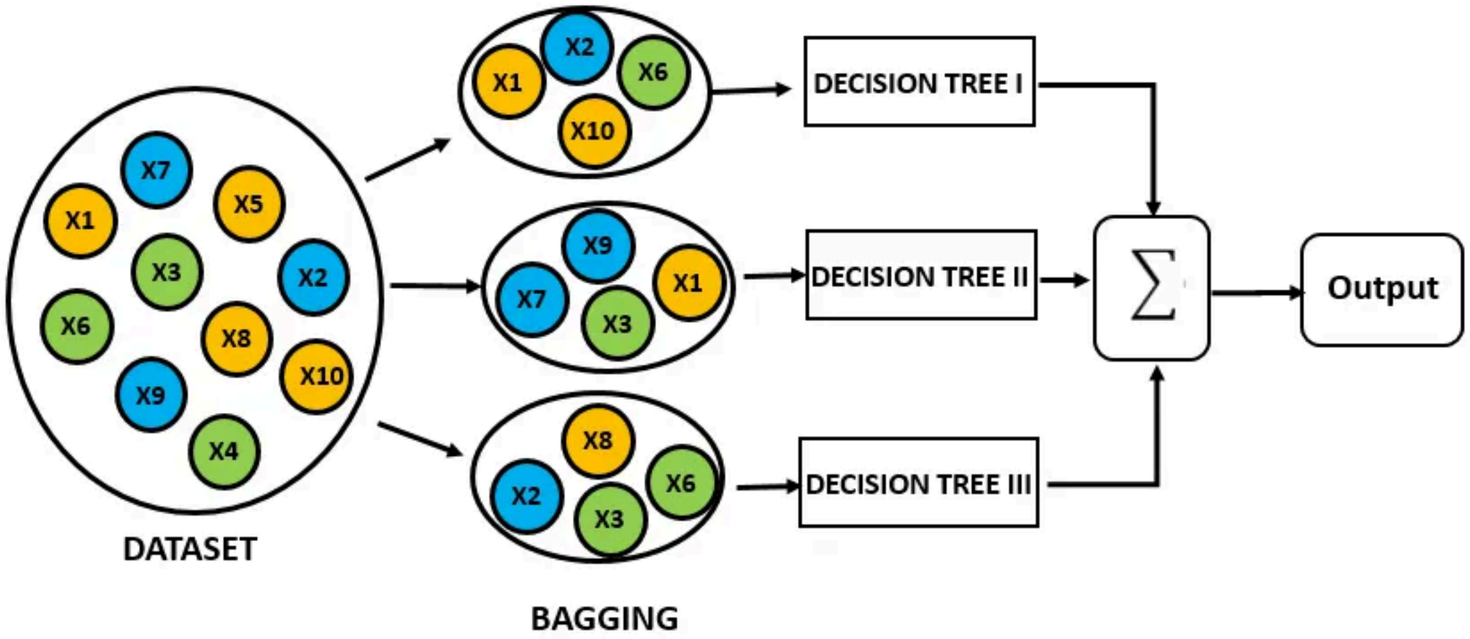
Un árbol de decisión por sí solo es un modelo simple y explicable, pero tiene un gran problema: **tiende a sobreajustar (overfitting) si no se limita bien**.

Leo Breiman, 2001, propuso una mejora al árbol decisión, el cual es un modelo con muy buenos resultados y rendimiento.



Random Forest

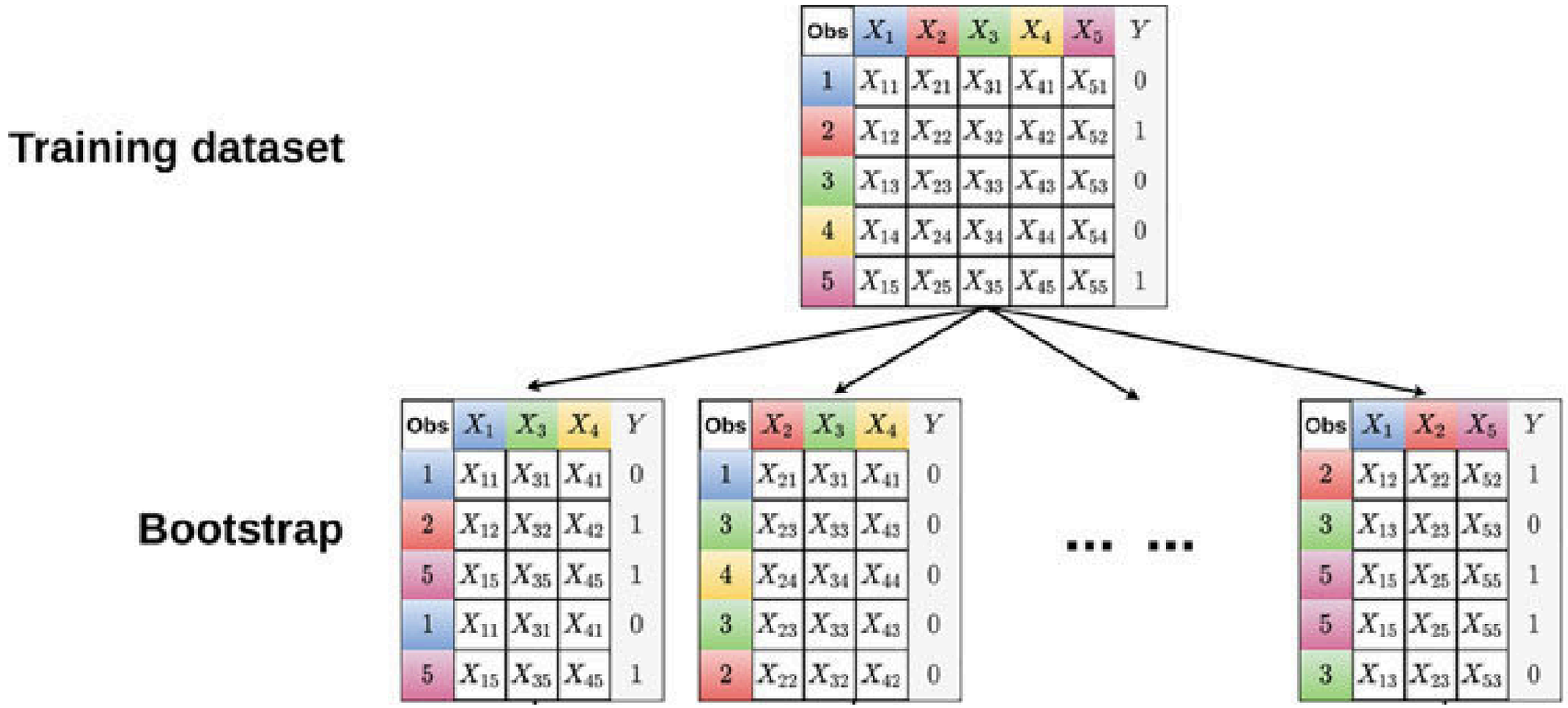
- Modelo **basado en los árboles de decisión**.
- Como su nombre lo dice, **genera un bosque** o selva de ellos para tomar una decisión.
- Cada árbol está formado por un subconjunto de los atributos totales.
- Para clasificar se genera una votación entre todos los árboles



Random Forest

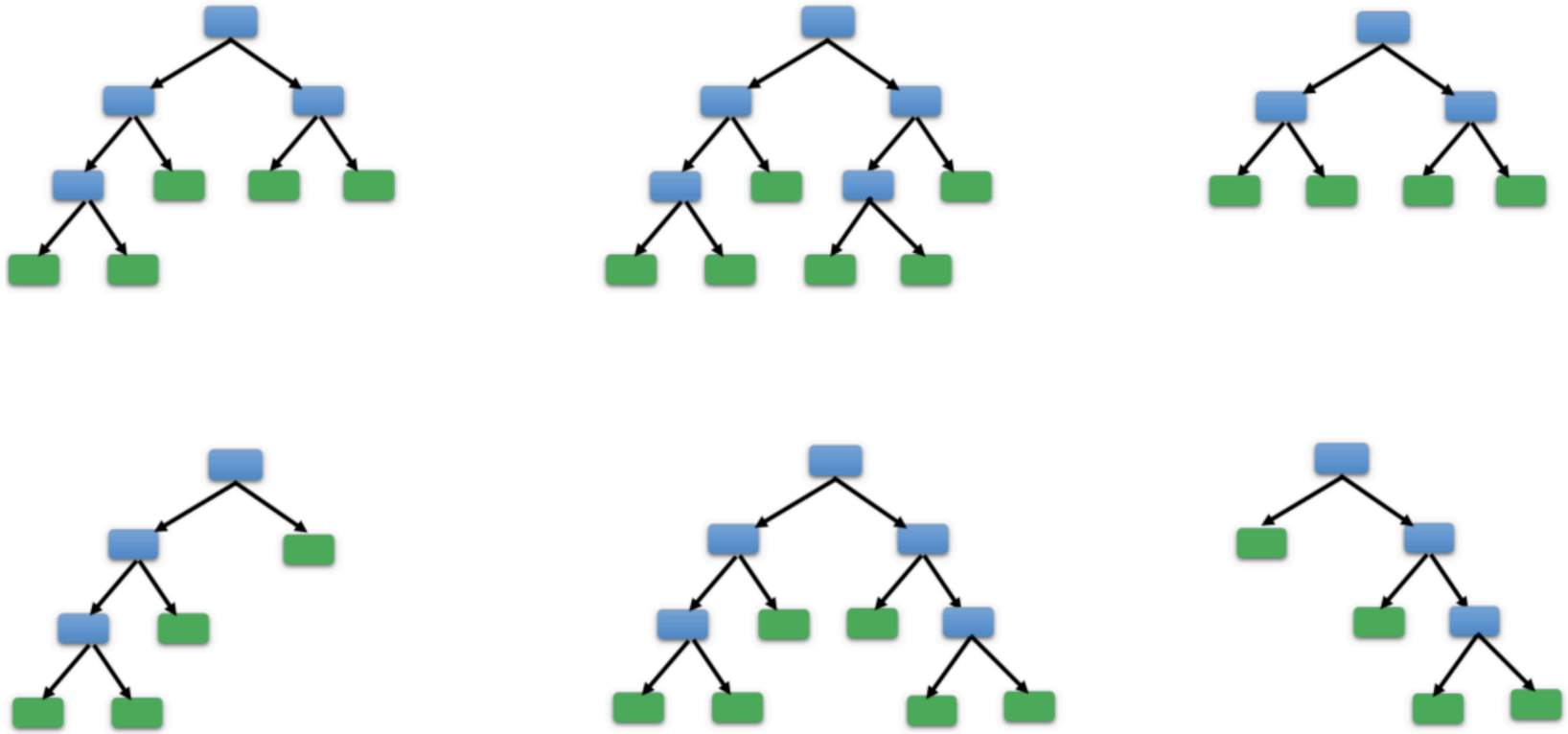
Es un ensamble de muchos árboles de decisión, donde cada árbol:

- Se entrena sobre un subconjunto aleatorio de los datos (**bootstrapping → muestreo con reemplazo**).
- En cada división, considera solo un subconjunto aleatorio de las variables.



Random Forest

Se entrenan varios árboles de decisión, más de 100, con los distintos ***bootstrapped***



Out-of-Bag (OOB)

Es una técnica de validación interna que aparece de manera natural en los Random Forests gracias al muestreo bootstrap.

Soleado	Viento	Edad	Ir a nadar
Sí	Leve	23	No
Sí	Fuerte	14	No
No	Leve	30	Sí
No	Leve	16	Sí
Sí	Leve	27	Sí
Sí	Fuerte	18	No
No	Fuerte	20	No

Bootstraped

Soleado	Viento	Edad	Ir a nadar
Sí	Leve	23	No
...	...	...	...
No	Fuerte	20	No

Out-of-Bag

Soleado	Viento	Edad	Ir a nadar
Sí	Leve	27	Sí
No	Fuerte	20	No

Out-of-Bag (OOB)

Dentro del Random Forest

- Cada árbol se entrena con una muestra bootstrap del dataset (selección aleatoria con reemplazo).
- En promedio, cada muestra bootstrap contiene alrededor del 63% de los datos originales.

¿Qué se hace con los OOB?

- Para cada árbol, se puede usar sus datos fuera de bolsa (OOB) para probar cómo clasifica.
- Al final, se combinan las predicciones de todos los árboles donde un ejemplo estuvo “fuera de bolsa”.
- Esto genera una estimación del error de generalización del bosque sin necesidad de un conjunto de validación separado

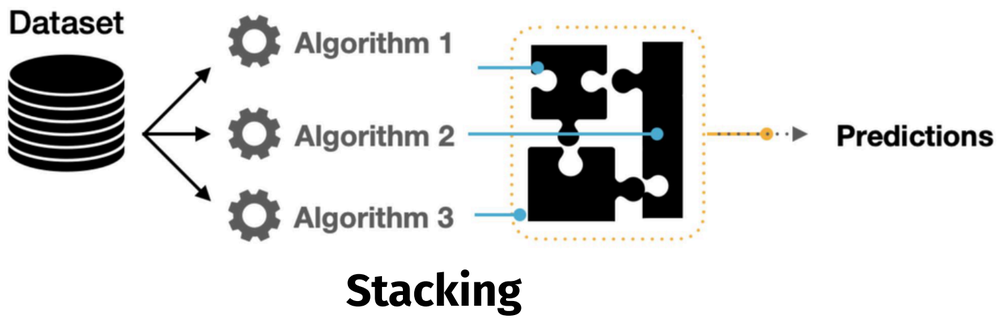
Random Forest

n_estimators: número de árboles (más árboles → más estable, pero más lento).

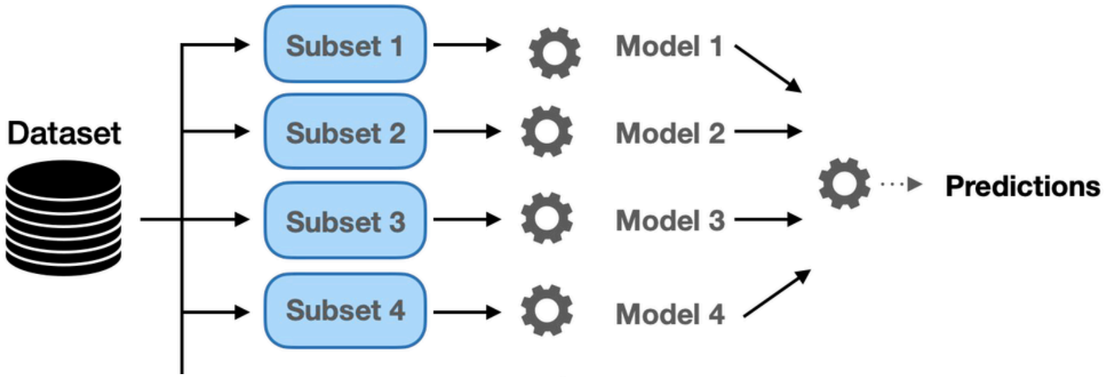
```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

X, y = load_iris(return_X_y=True)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)

# Crear y entrenar un Random Forest
rf = RandomForestClassifier(
    n_estimators=100,    # número de árboles
    max_depth=None,      # sin límite de profundidad
    random_state=42,
    oob_score=True       # usar muestras OOB para estimar el rendimiento
)
rf.fit(X_train, y_train)

# Predicción y evaluación
y_pred = rf.predict(X_test)
print("Accuracy en test:", accuracy_score(y_test, y_pred))
print("OOB score (estimado durante entrenamiento):", rf.oob_score_)
```

Ensemble Models

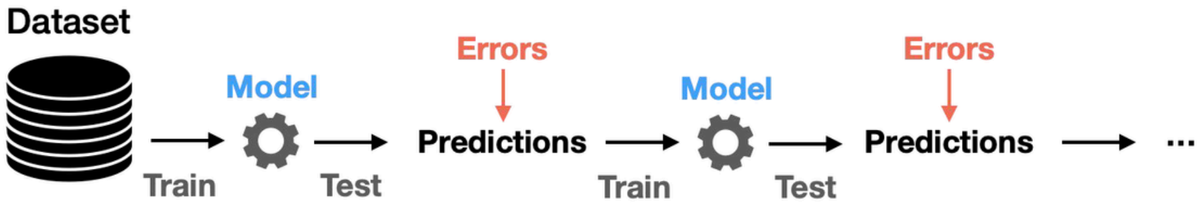
- Plantea que una mayor diversidad de modelos mejoran el performance general.
- Se pueden generar varias combinaciones, ya sea votaciones, ponderaciones o incluso, un aprendizaje continuo



Stacking



Bagging



Boosting