

Inteligencia Artificial

Machine Learning - (Aprendizaje Automático)

Aprendizaje Supervisado - Clasificación

Técnicas NO Paramétricas

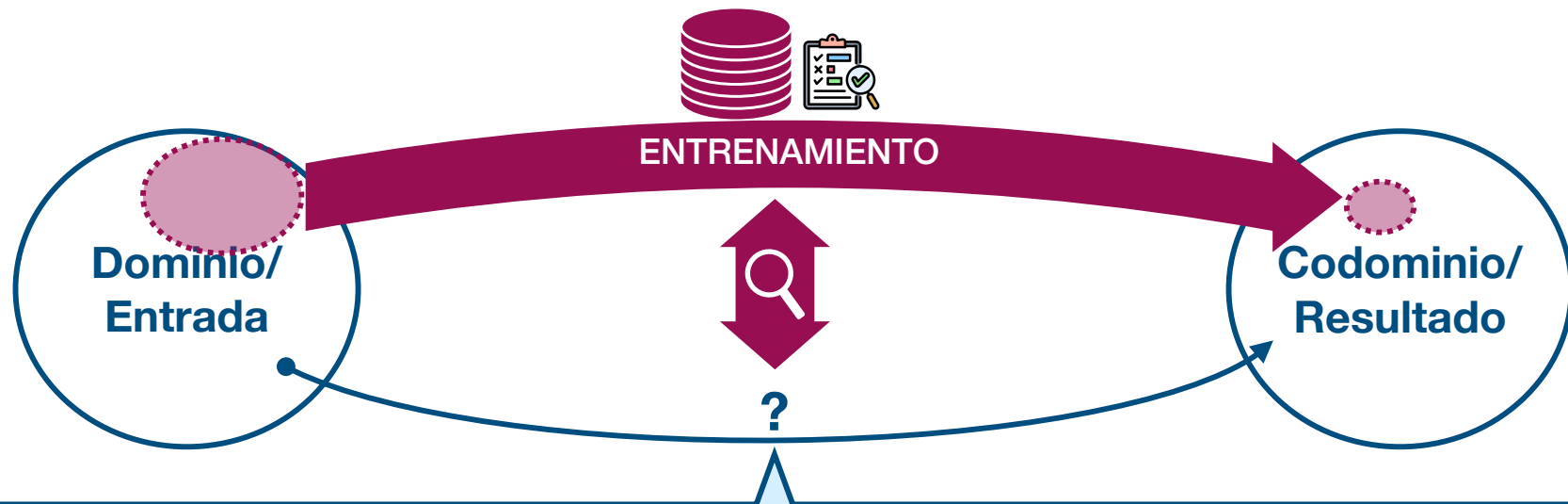


Machine Learning - Qué esperamos como resultado

Tipo de Aprendizaje	Tipo de Resultado Esperado	Ejemplo
Aprendizaje Supervisado	Clasificación	Predecir categorías. Ej: detectar si un email es spam o no.
	Regresión	Predecir valores numéricos. Ej: precio de una casa.
Aprendizaje No Supervisado	Agrupamiento (Clustering)	Descubrir grupos. Ej: segmentación de clientes por comportamiento.
	Detección de anomalías	Identificar valores atípicos. Ej: fraude bancario.
	Reducción de dimensionalidad	Simplificar datos. Ej: visualización de datos de alta dimensión.
	Reglas de asociación	Encontrar relaciones frecuentes. Ej: productos comprados juntos.
Aprendizaje por Refuerzo	Secuencias óptimas de acciones (reglas)	Tomar decisiones en ambientes dinámicos. Ej: robots o videojuegos.
Aprendizaje Auto-supervisado	Representaciones de datos	Aprender estructuras internas sin etiquetas explícitas. Ej: embeddings.
Aprendizaje Semi-supervisado	Clasificación / Regresión	Mezcla de datos etiquetados y no etiquetados. Mejora generalización.

Machine Learning - Aprendizaje Supervisado

El algoritmo recibe un conjunto de entrenamiento **con las respuestas correctas** y aprende a generalizar para responder correctamente a nuevas entradas.

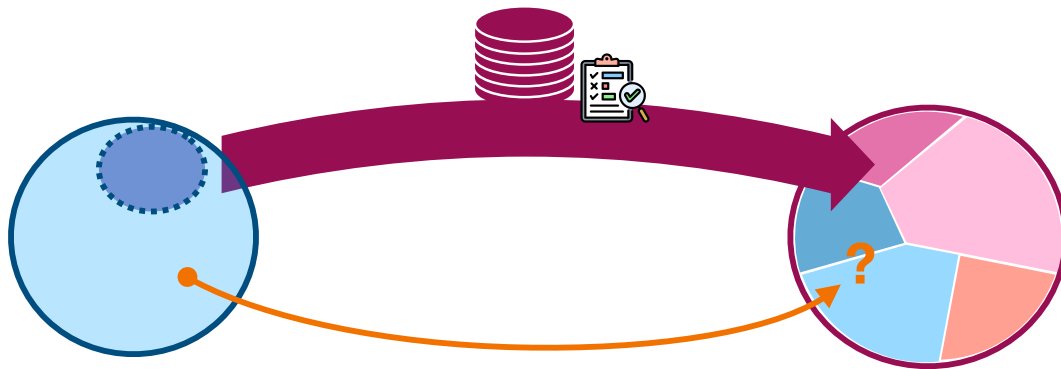


De manera general, los problemas se dividen en dos categorías, de acuerdo a la salida esperada del algoritmo:

- Problemas de **clasificación**: La salida del algoritmo es un valor tomado de un conjunto de finito de valores
- Problemas de **predicción (regresión)**: La salida es un número entero o real

ML Supervisado - Clasificación

La clasificación es una tarea de aprendizaje supervisado en la que **un modelo aprende a asignar una etiqueta o categoría a una entrada**, basándose en ejemplos previos con etiquetas conocidas. Estas etiquetas o categorías deben ser discretas y acotadas, sino estaríamos en un problema de regresión.



Existen diferentes tipos de clasificación, algunas de ellas son:

- **Clasificación Binaria:** cuando la cantidad de características o etiquetas son dos (por ejemplo, detección de spam en el correo)
- **Clasificación Multi-clase:** cuando la cantidad de características o etiquetas son más de dos (por ejemplo, detección de frutas)
- **Clasificación Multi-etiqueta:** cuando los individuos pueden tener más de una etiqueta o característica (por ejemplo, características de una foto: (sepia/color, día/noche, ...))

ML Supervisado - Clasificación - No Paramétrica

En machine learning, un modelo **no paramétrico** es aquel que no asume una forma funcional fija para los datos (como una recta o un polinomio). Además, en general, la complejidad del modelo crece con la cantidad de datos.

Algunas de las técnicas de clasificación NO paramétricas más conocidas son:

k-Nearest Neighbors (k-NN): Predice según los vecinos más cercanos en el espacio de características.

Árboles de decisión (Decision Trees): No suponen ninguna forma de la función; dividen el espacio según los datos.

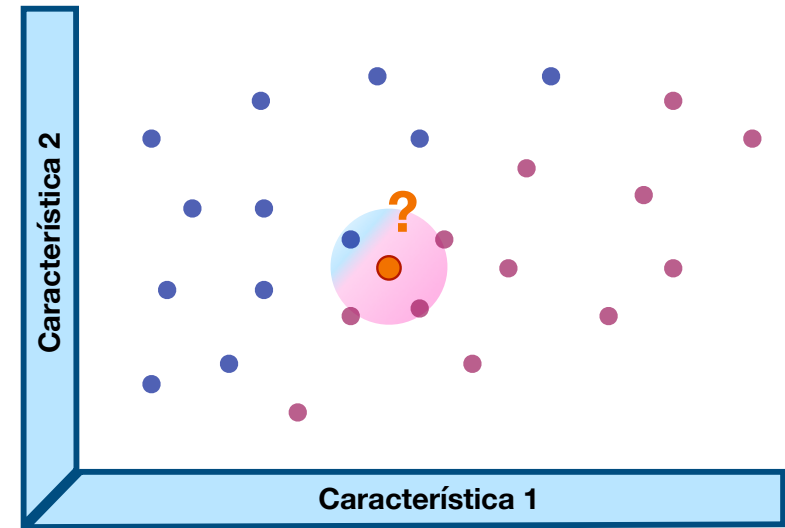
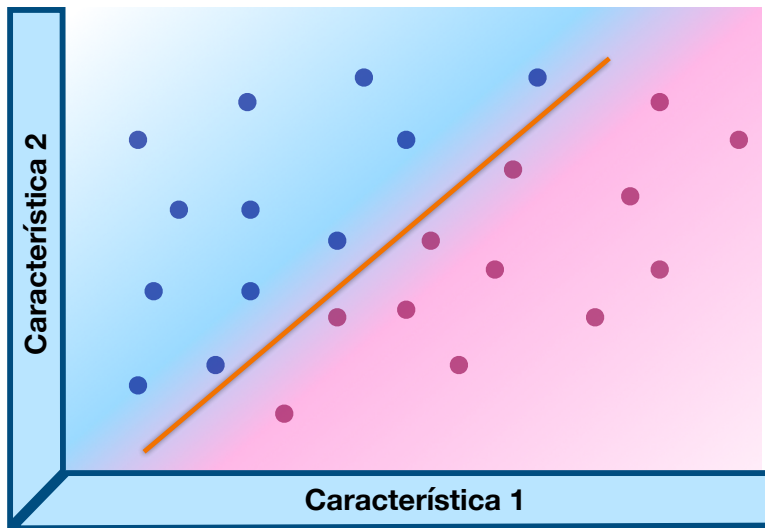
Support Vector Machines (SVM, con kernels no lineales): Expanden la dimensionalidad con kernels, sin forma paramétrica fija.

MLS Clasificación - k-Nearest Neighbors (k-NN)

k-Nearest Neighbors (k-NN) es un algoritmo de aprendizaje supervisado para clasificación y regresión. La idea es simple:

“Un ejemplo se parece más a aquellos que están más cerca en el espacio de características.”

Esta técnica no construye un modelo explícito, sino que almacena los datos de entrenamiento para buscar los **k** vecinos más cercanos al nuevo punto de entrada para predecir su valor.

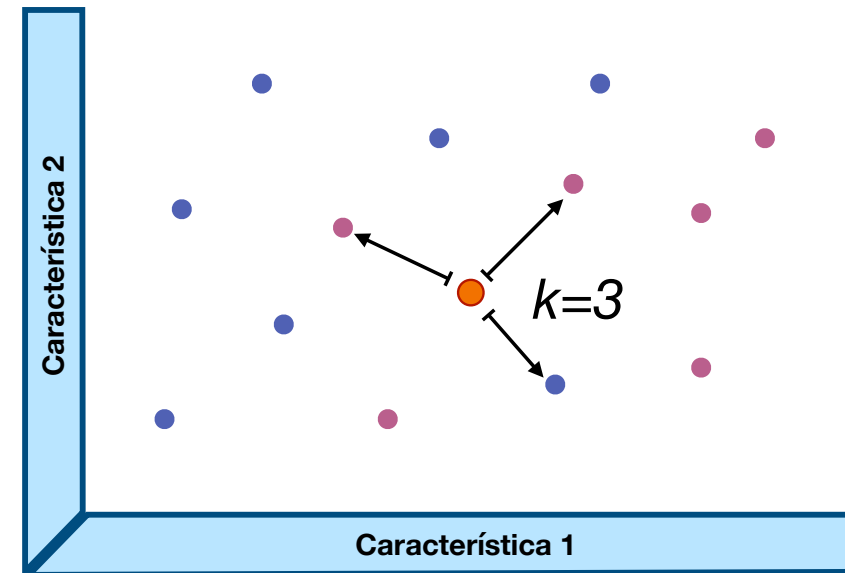


MLS Clasificación - k-Nearest Neighbors (k-NN)

La técnica consiste en buscar cuáles (k) casos del conjunto de entrenamiento (casos conocidos) son los más cercanos y en base a sus valores, predecir cuál es el valor para el actual.

Como podemos observar, no es necesario entrenamiento alguno. Si se deben tener en cuenta varios hiperparámetros:

- **Cuántos vecinos se van a tener en cuenta, usualmente una cantidad impar.**
- **Cómo medimos la distancia de proximidad**
- **Cómo determinamos el valor a predecir**



MLS Clasificación - k-NN - ¿cuál K?

Si elegimos ***k* pequeños**, el modelo se torna más flexible pero es muy sensible al ruido y valores atípicos (***overfitting***)

Si el ***k es demasiado grande*** (predecimos basándonos en muchos vecinos), mejora la robustez en cuanto a ruido y valores atípicos pero podemos perder demasiada precisión y caer en ***underfitting***.

Algunas ***heurísticas*** indican que el *k* puede calcularse en base a al tamaño del conjunto de entrenamiento, $k \approx \sqrt{N}$.

La mejor opción es utilizar **cross-validation sobre el conjunto de entrenamiento** para encontrar el mejor *k* (aquel que **maximiza la precisión**).

MLS Clasificación - k-NN - Distancias

La distancia de **Minkowski** es una generalización de varias métricas de distancia conocidas, incluidas la distancia Euclídea y la distancia Manhattan.

Dados dos vectores (casos) en un espacio n-dimensional:

$$x = (x_1, x_2, \dots, x_n), \quad y = (y_1, y_2, \dots, y_n)$$

La distancia de Minkowski de orden p se define como:

$$d(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

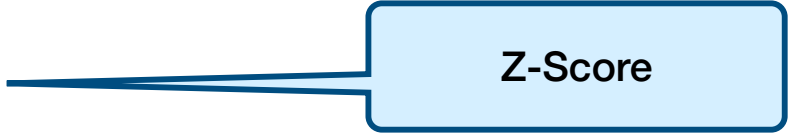
- Cuando **p = 1**: se denomina distancia **Manhattan** (o de la ciudad), funciona mejor si las diferencias individuales importan más (ej. datos dispersos).
- Cuando **p = 2**: es la clásica distancia **Euclídeana**
- Cuando **p > 2**: se denomina distancia **Chebyshev** (o máximo absoluto), penaliza más las diferencias grandes

MLS Clasificación - k-NN - Escalado

Si usamos directamente los valores “crudos” de cada dimensión (altura, peso, edad, etc.), la distancia depende de la escala de las variables (kg, tn, mts, millas, etc.). Esto puede afectar nuestra técnica, por ejemplo, medir altura en metros o en millas cambia totalmente las distancias. También es difícil comparar magnitudes diferentes: ¿qué significa que alguien difiera en 5 años de edad vs. 5 kg de peso?.

Para reducir el impacto podemos aplicar **normalización** o **estandarización**:

Se calcula la media (μ_i) y la desviación estándar (σ_i) de cada dimensión i .

Cada valor $x_{j,i}$ se transforma en: $z_{j,i} = \frac{x_{j,i} - \mu_i}{\sigma_i}$ 

Este proceso produce que **todas las variables terminan con media 0 y desviación estándar 1**. Además, ninguna dimensión domina las distancias solo por estar en otra escala.

MLS Clasificación - k-NN - Escalado

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from scipy.spatial.distance import euclidean

# Dos puntos (personas) con características: [altura en cm, peso en kg, edad en años]
p1 = np.array([180, 80, 25]) # persona 1
p2 = np.array([170, 60, 40]) # persona 2

# Distancia Euclidiana sin normalización
dist_raw = euclidean(p1, p2)

# Normalización (estandarización Z-score)
scaler = StandardScaler()
data = np.array([p1, p2])
data_scaled = scaler.fit_transform(data)

# Distancia Euclidiana con normalización
dist_scaled = euclidean(data_scaled[0], data_scaled[1])

dist_raw, dist_scaled
```

(26.92, 3.46)

MLS Clasificación - k-NN - Búsqueda eficiente

Cuando la cantidad de casos de entrenamiento es demasiado grande podemos utilizar estructuras de datos para guardar la información que permitan realizar búsquedas de manera más eficiente.

Un ejemplo es la utilización de **k-d trees (árboles k dimensionales)**. La idea es similar a un árbol binario de búsqueda (BST), pero extendido a múltiples dimensiones. Cada nivel del árbol divide el espacio de datos según una de las dimensiones. La división se hace usando un hiperplano perpendicular al eje de esa dimensión (por ejemplo la mediana).

Ejemplo en 2D, en el primer nivel, se divide según el eje x, en el segundo nivel, según el eje y. En el tercer nivel, otra vez según x, y así sucesivamente.

El costo de **construirlo** es $O(N \log N)$, la **búsqueda en el mejor caso es $O(\log N)$** y en el peor $O(N)$.

MLS Clasificación - k-NN - Valor a predecir

Para determinar el valor a predecir (clase) el método más utilizado es el de **votación mayoritaria** entre las clases de los vecinos.

En caso de estar utilizando k-NN como técnica de regresión, se pueden aplicar cálculos como el **promedio o la media ponderada** de los valores de los k vecinos.

MLS Clasificación - k-Nearest Neighbors

Ventajas:

- Sencillo de implementar.
- No requiere entrenamiento explícito.
- Flexible, ya que puede usarse para clasificación o regresión.
- Se adapta bien a problemas con fronteras de decisión complejas (no necesita que los casos sean linealmente separables).

Desventajas:

- Costoso en predicción: requiere calcular distancias con todos los datos, ineficiente con datasets grandes (usar k-d trees).
- Sensible a la escala de los datos (se recomienda normalizar o estandarizar)
- Sensible a datos atípicos o ruido (elegir correctamente k).
- Dificultad en alta dimensión (problema de la “maldición de la dimensionalidad”).

MLS Clasificación - k-NN - ScikitLearn

```
from sklearn.neighbors import KNeighborsClassifier
# Datos de ejemplo: [largo_pétalo, ancho_pétalo]
X = np.array([
    [1.0, 0.5],
    [1.2, 0.7],
    [1.1, 0.6],
    [3.0, 2.5],
    [3.2, 2.8],
    [2.9, 2.4]
])
# Etiquetas: 0 = Flor tipo A, 1 = Flor tipo B
y = np.array([0, 0, 0, 1, 1, 1])
# Creamos el clasificador KNN usando KD-Tree
knn = KNeighborsClassifier(n_neighbors=3, algorithm='kd_tree')
# Entrenamos el modelo ( construye el espacio para consultas)
knn.fit(X, y)
# Hacemos una predicción para una nueva flor
nueva_flor = np.array([[1.1, 0.65]])
prediccion = knn.predict(nueva_flor)

print("La clase predicha para la nueva flor es:", prediccion[0])
```

Cantidad de vecinos a tener en cuenta

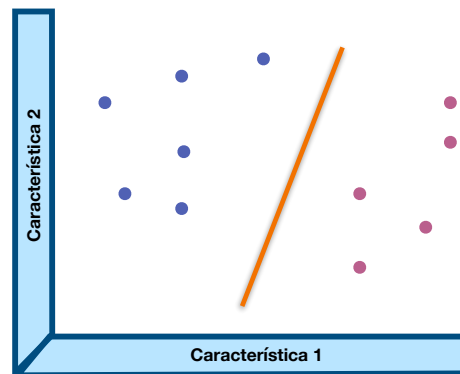
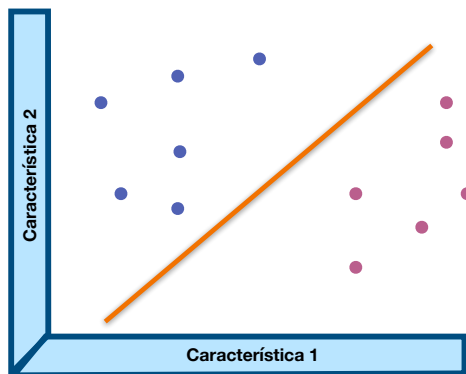
Para generar mantener los datos de manera eficiente en k dimensional tree

MLS Clasificación - Support Vector Machines (SVM)

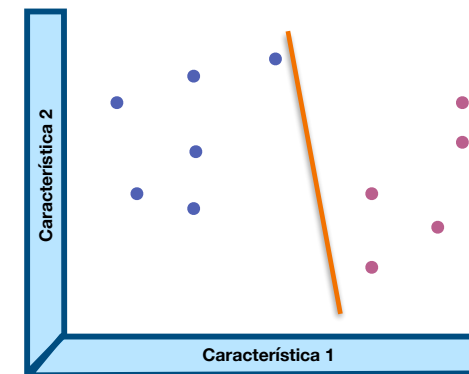
Las **Máquinas de Sopore Vectorial** (SVM) son un poderoso método para clasificación (o regresión) propuesto por **Vladimir Vapnik** durante la década del '60, quien junto **Alexey Chervonenkis** desarrollaron la Teoría de Aprendizaje Estadístico (Statistical Learning Theory) y el Principio de Minimización del Riesgo Estructural (SRM), que es la base teórica de las SVM.



La idea de SVM es tratar de encontrar cuál es la mejor forma de partir (separar) clases de valores para poder clasificarlos



...

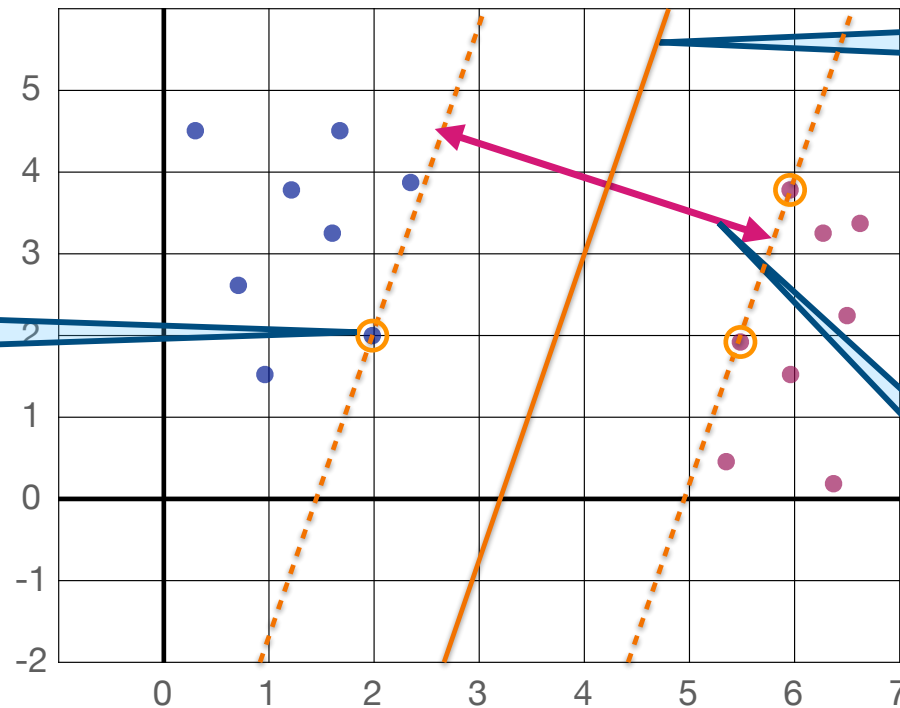


?

MLS Clasificación - Support Vector Machines (SVM)

El método SVM propone que la mejor separación entre clases es la que maximiza el **margen**, es decir, la distancia entre la recta (hiperplano en general) y los puntos más cercanos a la misma.

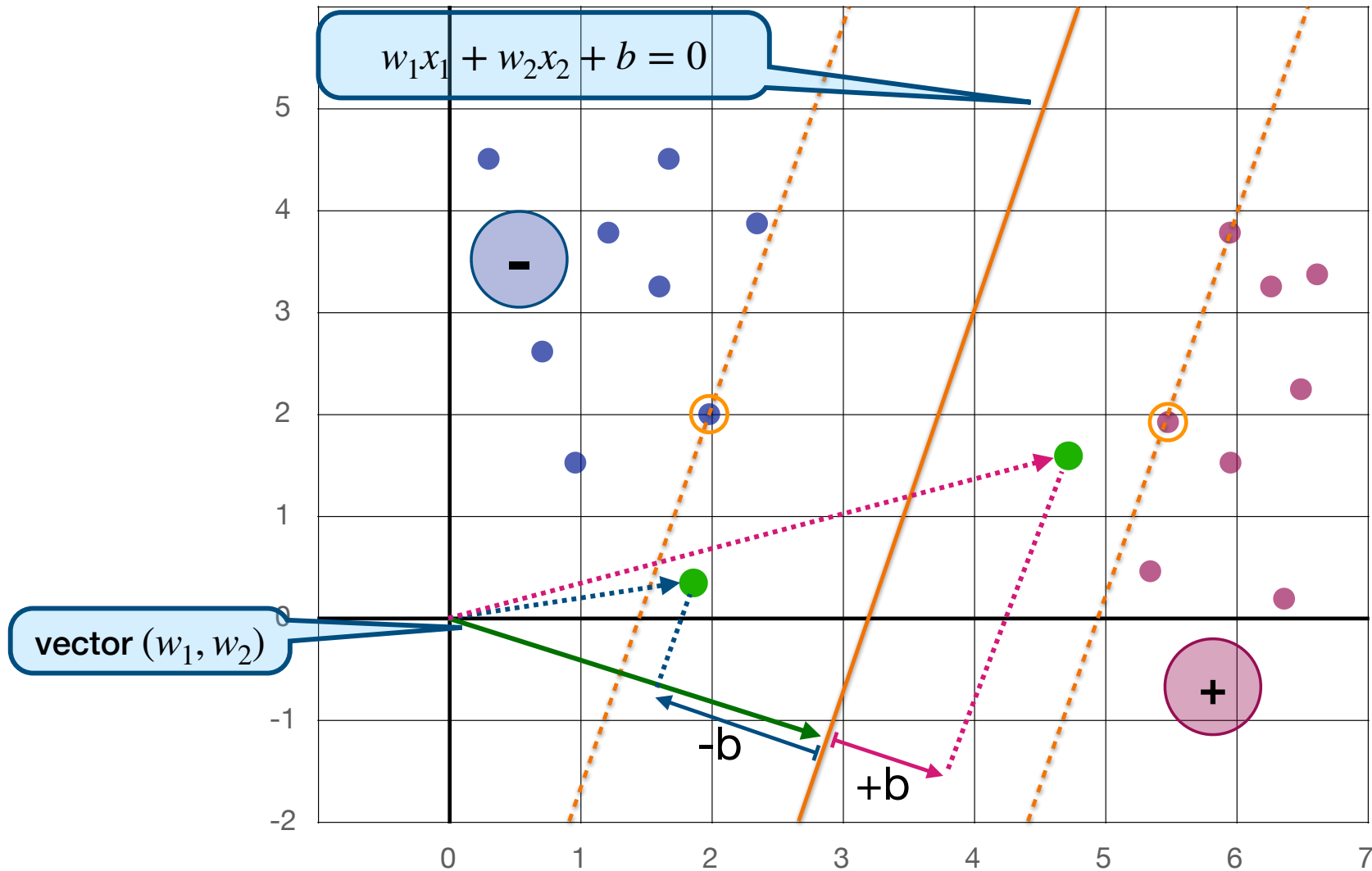
○ Vectores soporte: son los puntos de entrenamiento más cercanos al hiperplano. Definen la posición y orientación del hiperplano. Sólo estos puntos son relevantes para la decisión final.



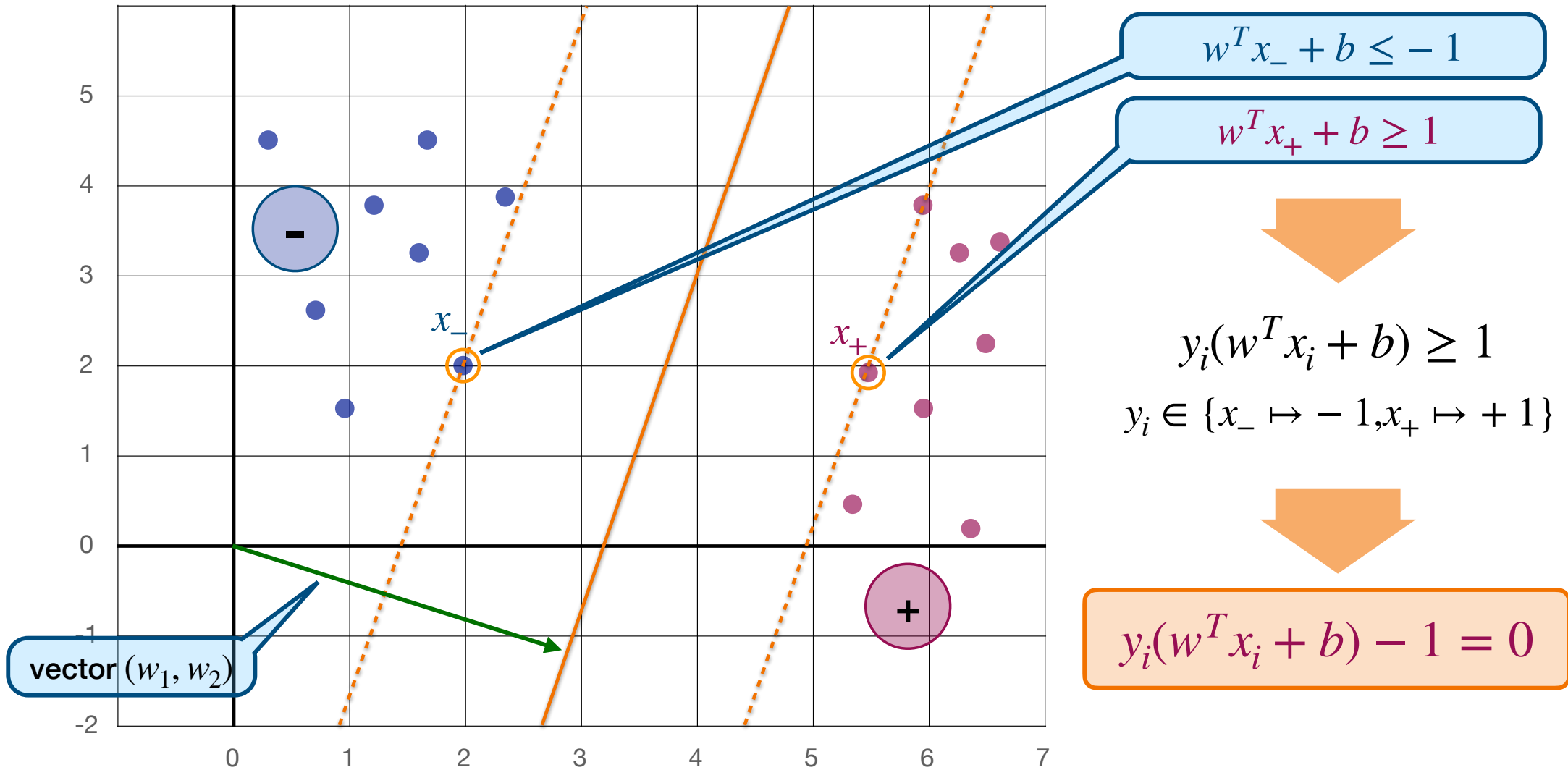
La fórmula que define el hiperplano es $w^T x + b = 0$. Donde T es la cantidad de características. En este caso como es el plano:
 $w_1 x_1 + w_2 x_2 + b = 0$

Margen a maximizar

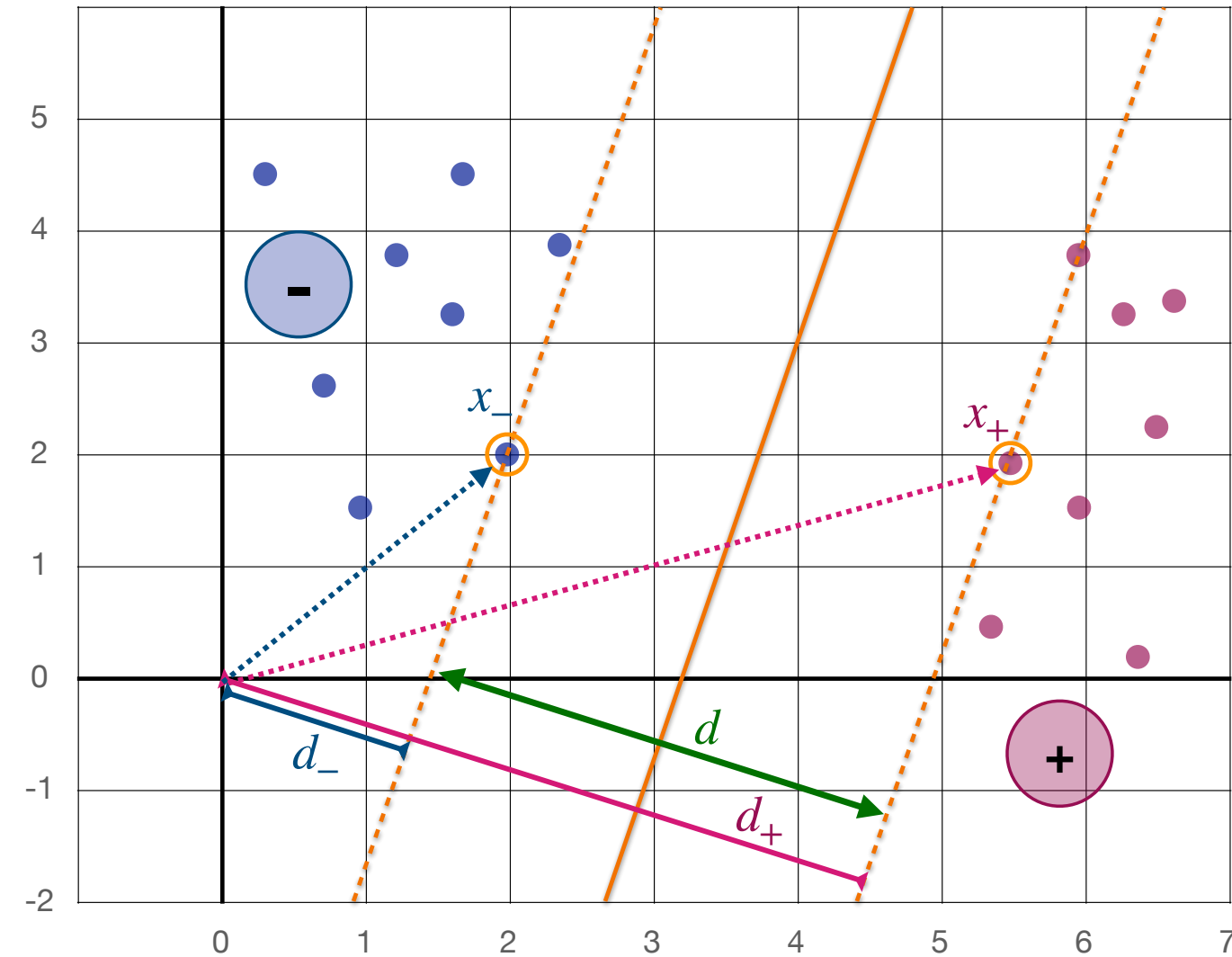
MLS Clasificación - Support Vector Machines (SVM)



MLS Clasificación - Support Vector Machines (SVM)



MLS Clasificación - Support Vector Machines (SVM)



$$y_i(w^T x_i + b) - 1 = 0$$

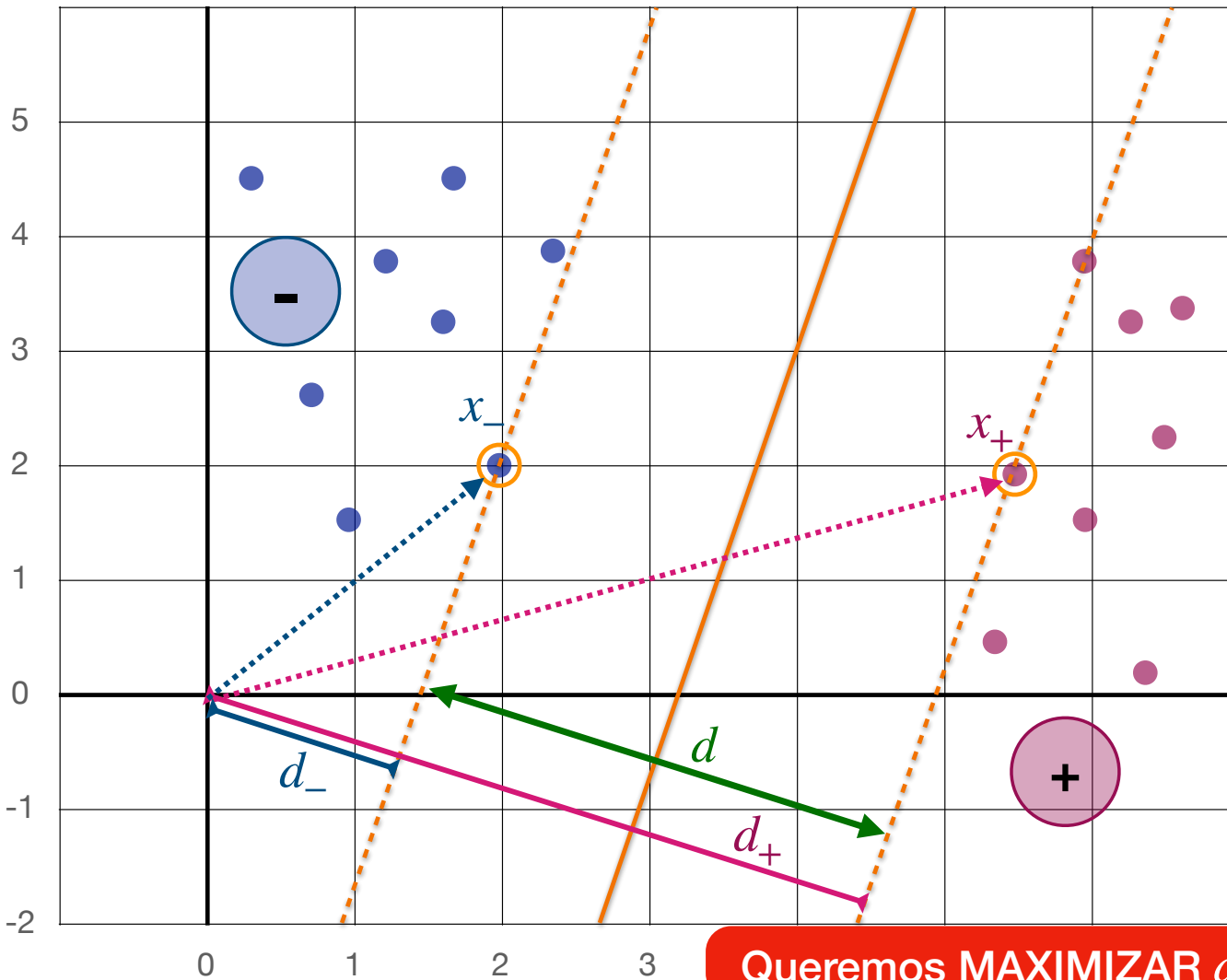
$$d_- = \frac{w^T x_-}{\|w\|} \quad d_+ = \frac{w^T x_+}{\|w\|}$$

$$d = \frac{1}{\|w\|} (w^T x_+ - w^T x_-)$$

$$d = \frac{1}{\|w\|} (1 - b + 1 + b)$$

$$d = \frac{2}{\|w\|}$$

MLS Clasificación - Support Vector Machines (SVM)



$$y_i(w^T x_i + b) - 1 = 0$$

$$d_- = \frac{w^T x_-}{\|w\|}$$

$$d_+ = \frac{w^T x_+}{\|w\|}$$

$$d = \frac{1}{\|w\|} (w^T x_+ - w^T x_-)$$

$$d = \frac{1}{\|w\|} (1 - b + 1 + b)$$

$$d = \frac{2}{\|w\|}$$

MLS Clasificación - Support Vector Machines (SVM)

$$d = \frac{2}{\|w\|}$$

$$y_i(w^T x_i + b) - 1 = 0$$

$$d = \frac{1}{2} \|w\|^2$$

Utilizando Lagrange

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Derivando

$$\frac{\partial L}{\partial w} : w = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} : \sum_{i=1}^N \alpha_i y_i = 0$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

MLS Clasificación - Support Vector Machines (SVM)

$$d = \frac{2}{\|w\|}$$

Queremos MAXIMIZAR d

$$y_i(w^T x_i + b) - 1 = 0$$

$$d = \frac{1}{2} \|w\|^2$$

Utilizando Lagrange

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Derivando

$$\frac{\partial L}{\partial w} : w = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} : \sum_{i=1}^N \alpha_i y_i = 0$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

MLS Clasificación - Support Vector Machines (SVM)

$$d = \frac{2}{\|w\|}$$

Queremos MAXIMIZAR d

$$y_i(w^T x_i + b) - 1 = 0$$

$$d = \frac{1}{2} \|w\|^2$$

equivalente a MINIMIZAR

Utilizando Lagrange

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Derivando

$$\frac{\partial L}{\partial w} : w = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} : \sum_{i=1}^N \alpha_i y_i = 0$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

MLS Clasificación - Support Vector Machines (SVM)

$$d = \frac{2}{\|w\|}$$

Queremos MAXIMIZAR d

equivalente a MINIMIZAR

$$d = \frac{1}{2} \|w\|^2$$

$$y_i(w^T x_i + b) - 1 = 0$$

Restricciones

Utilizando Lagrange

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Derivando

$$\frac{\partial L}{\partial w} : w = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} : \sum_{i=1}^N \alpha_i y_i = 0$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

MLS Clasificación - Support Vector Machines (SVM)

$$d = \frac{2}{\|w\|}$$

Queremos MAXIMIZAR d

equivalente a MINIMIZAR

$$d = \frac{1}{2} \|w\|^2$$

$$y_i(w^T x_i + b) - 1 = 0$$

Restricciones

Utilizando Lagrange

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i [y_i(w \cdot x_i + b) - 1]$$

Derivando

Depende de los valores de aprendizaje

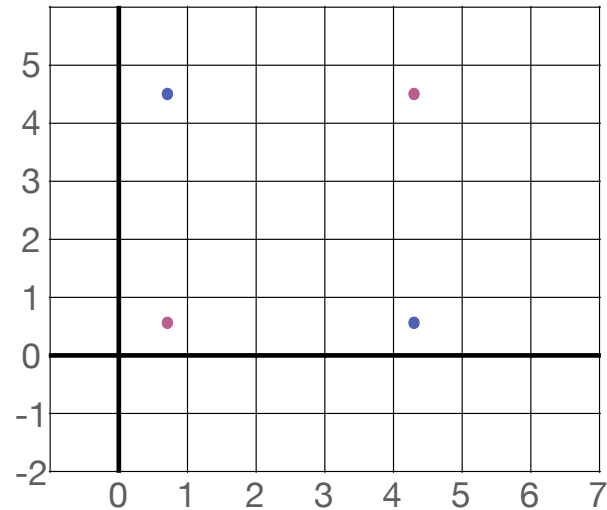
$$\frac{\partial L}{\partial w} : w = \sum_{i=1}^N \alpha_i y_i x_i$$

$$\frac{\partial L}{\partial b} : \sum_{i=1}^N \alpha_i y_i = 0$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

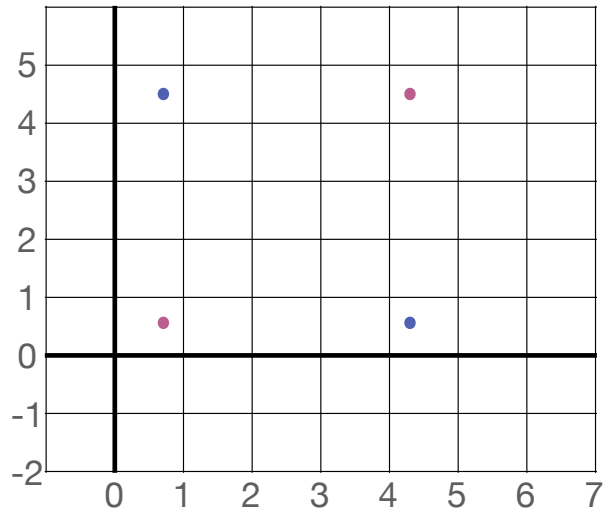
MLS Clasificación - SVM - Kernel Trick (1995)

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$



MLS Clasificación - SVM - Kernel Trick (1995)

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

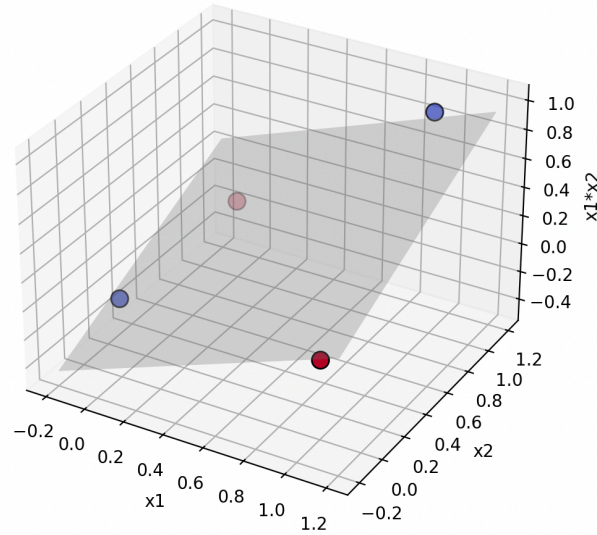
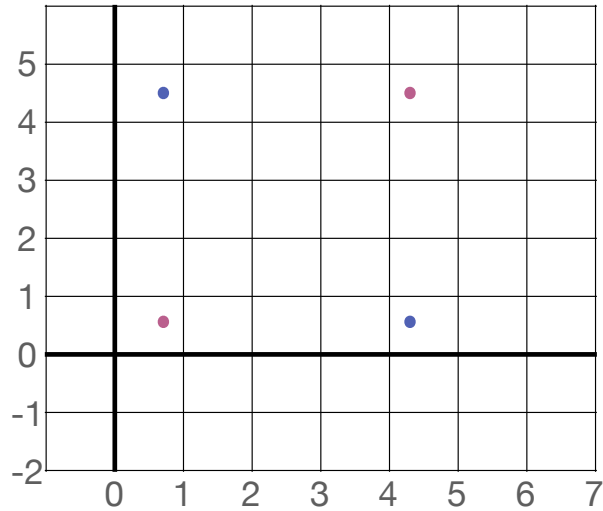


¿ Qué sucede si los
datos no son linealmente
separables ?

MLS Clasificación - SVM - Kernel Trick (1995)

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$



¿ Qué sucede si los datos no son linealmente separables ?

MLS Clasificación - SVM - Kernel Trick (1995)

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Lineal:

$$\langle x, y \rangle$$

Polinómico:

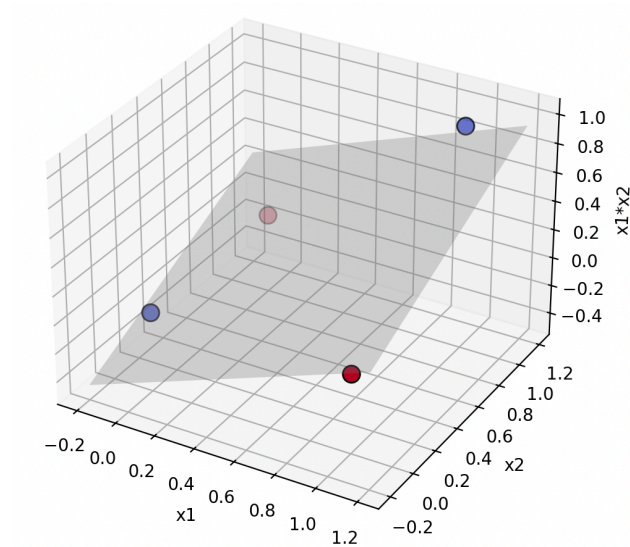
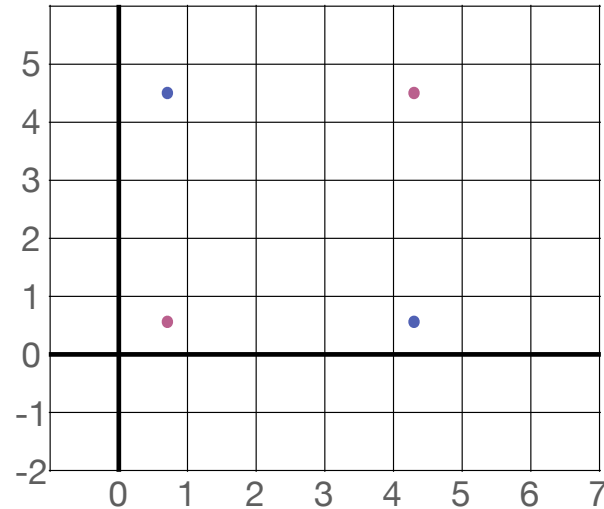
$$(\langle x, y \rangle + c)^d$$

Radial Basis Function
(RBF/Gaussiano):

$$\exp(-\gamma ||x - y||^2)$$

Sigmoide:

$$\tanh(\alpha \langle x, y \rangle + c)$$



¿ Qué sucede si los
datos no son linealmente
separables ?

MLS Clasificación - SVM - Kernel Trick (1995)

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j (x_i \cdot x_j)$$

$$L(\alpha_i) = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

Lineal:

$$\langle x, y \rangle$$

Polinómico:

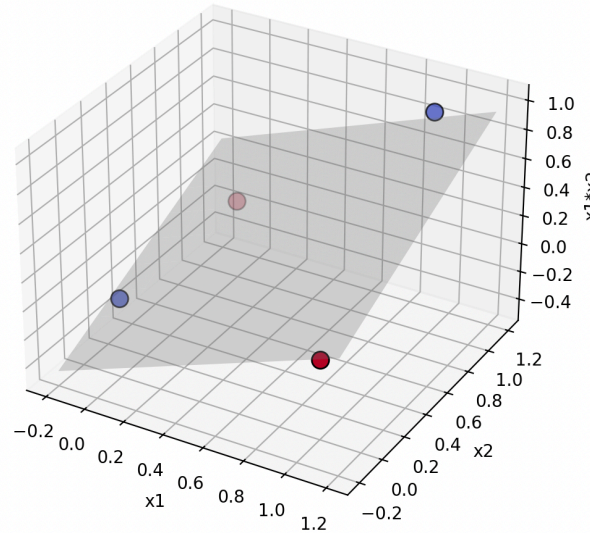
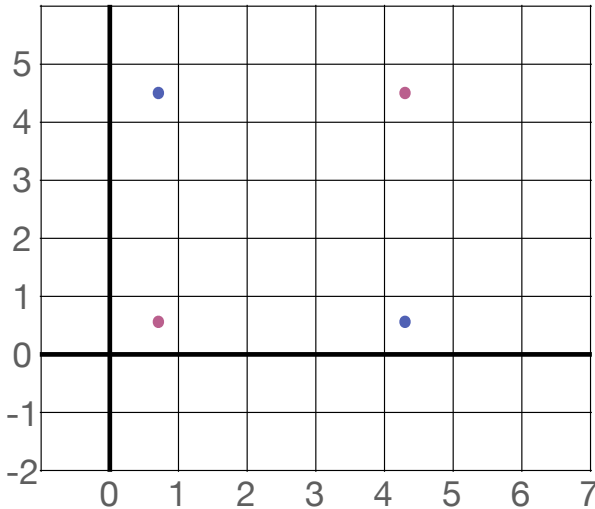
$$(\langle x, y \rangle + c)^d$$

Radial Basis Function
(RBF/Gaussiano):

$$\exp(-\gamma ||x - y||^2)$$

Sigmoide:

$$\tanh(\alpha \langle x, y \rangle + c)$$

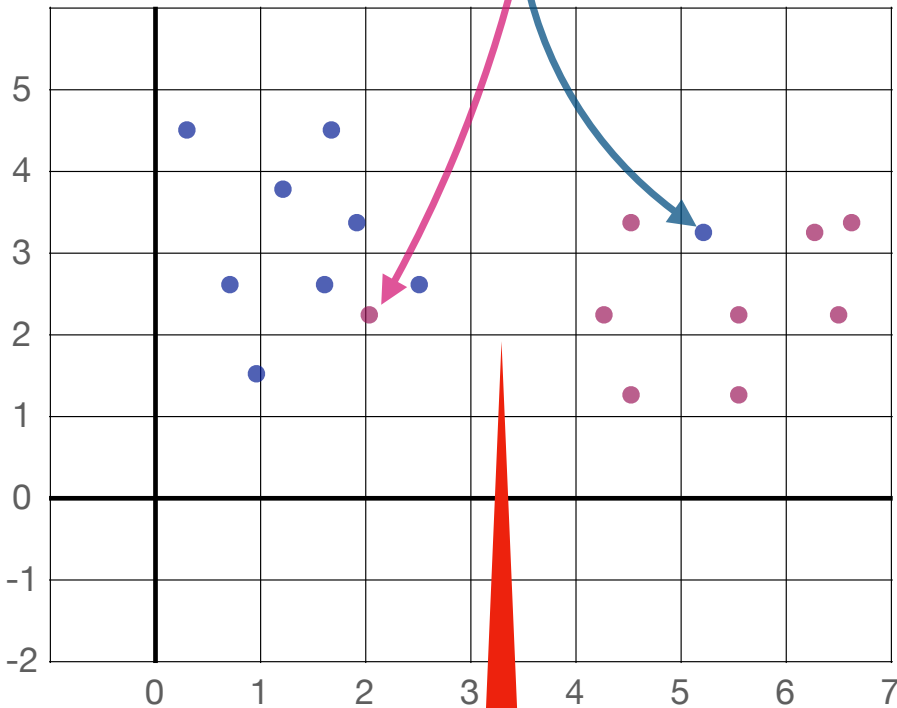


¿ Qué sucede si los
datos no son linealmente
separables ?

Se denomina truco kernel, porque no se
necesitar agregar nuevas dimensiones a los
datos, sino que se calculan directamente con
la transformación (Kernel)

MLS Clasificación - SVM - Soft Margin

Ruido o Outlier



¿ Podemos encontrar SMV ?

En un SVM se exige que todas las instancias de entrenamiento estén perfectamente separadas por el hiperplano.

El **soft margin** permite que algunos puntos estén mal clasificados o dentro del margen, con el **objetivo de obtener un modelo que generalice mejor**.

Esto se logra introduciendo **slack variables**, que miden cuántos puntos “violan” la separación correcta.

$$\min \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$$

$$y_i(w \cdot x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

C: Hiperparámetro para controlar el grado de penalidad

MLS Clasificación - SVM - SciKitLearn

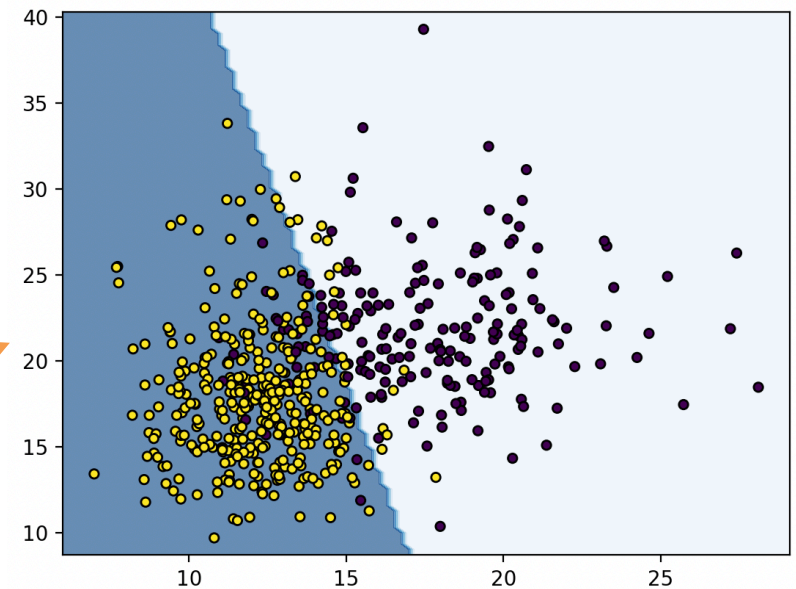
```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

svm = SVC(kernel="linear", C=2)
svm.fit(X, y)

DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    alpha=0.6, # transparencia
    cmap="Blues", # Gama de colores
)

plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```



MLS Clasificación - SVM - SciKitLearn

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

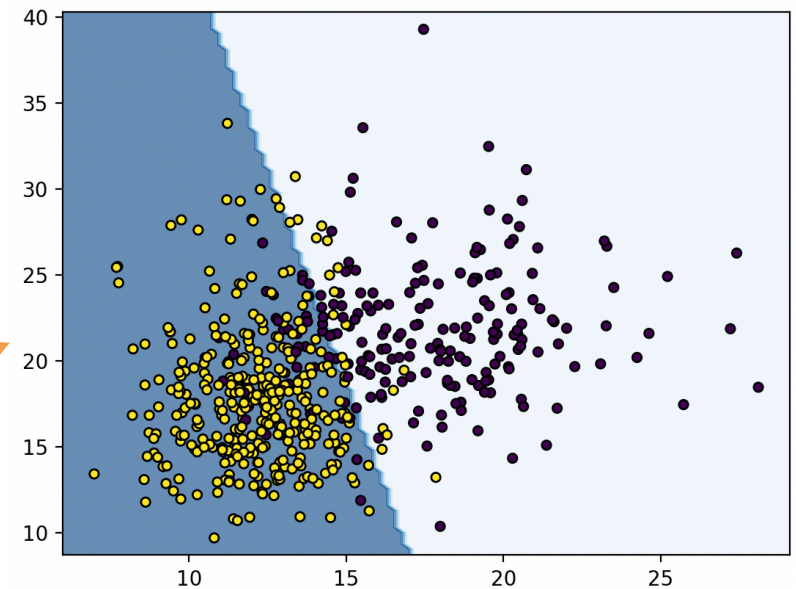
cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

svm = SVC(kernel="linear", C=2)
svm.fit(X, y)

DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    alpha=0.6, # transparencia
    cmap="Blues", # Gama de colores
)

plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```

Resultado de areas



MLS Clasificación - SVM - SciKitLearn

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

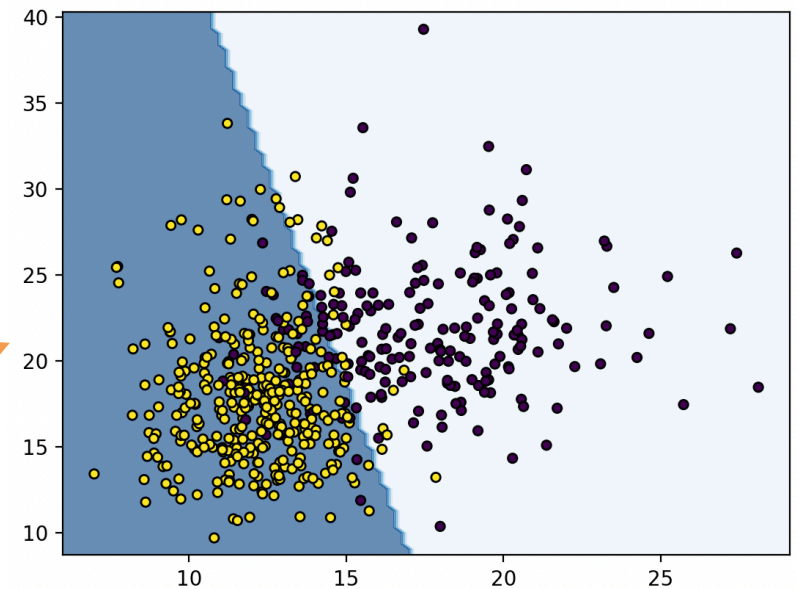
svm = SVC(kernel="linear", C=2)
svm.fit(X, y)

DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    alpha=0.6, # transparencia
    cmap="Blues", # Gama de colores
)

plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```

SVM lineal con softmargin 2

Resultado de areas



MLS Clasificación - SVM - SciKitLearn

```
from sklearn.datasets import load_breast_cancer
import matplotlib.pyplot as plt
from sklearn.inspection import DecisionBoundaryDisplay
from sklearn.svm import SVC

cancer = load_breast_cancer()
X = cancer.data[:, :2]
y = cancer.target

svm = SVC(kernel="linear", C=2)
svm.fit(X, y)

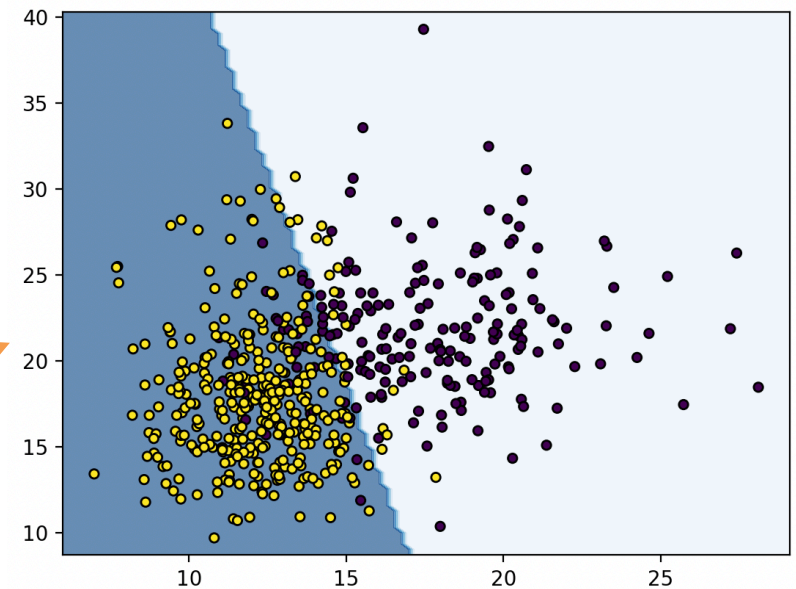
DecisionBoundaryDisplay.from_estimator(
    svm,
    X,
    response_method="predict",
    alpha=0.6, # transparencia
    cmap="Blues", # Gama de colores
)

plt.scatter(X[:, 0], X[:, 1],
            c=y,
            s=20, edgecolors="k")
plt.show()
```

Dataset ejemplos de la clase

SVM lineal con softmargin 2

Resaltado de areas



MLS Clasificación - SVM - Resumen

Ventajas

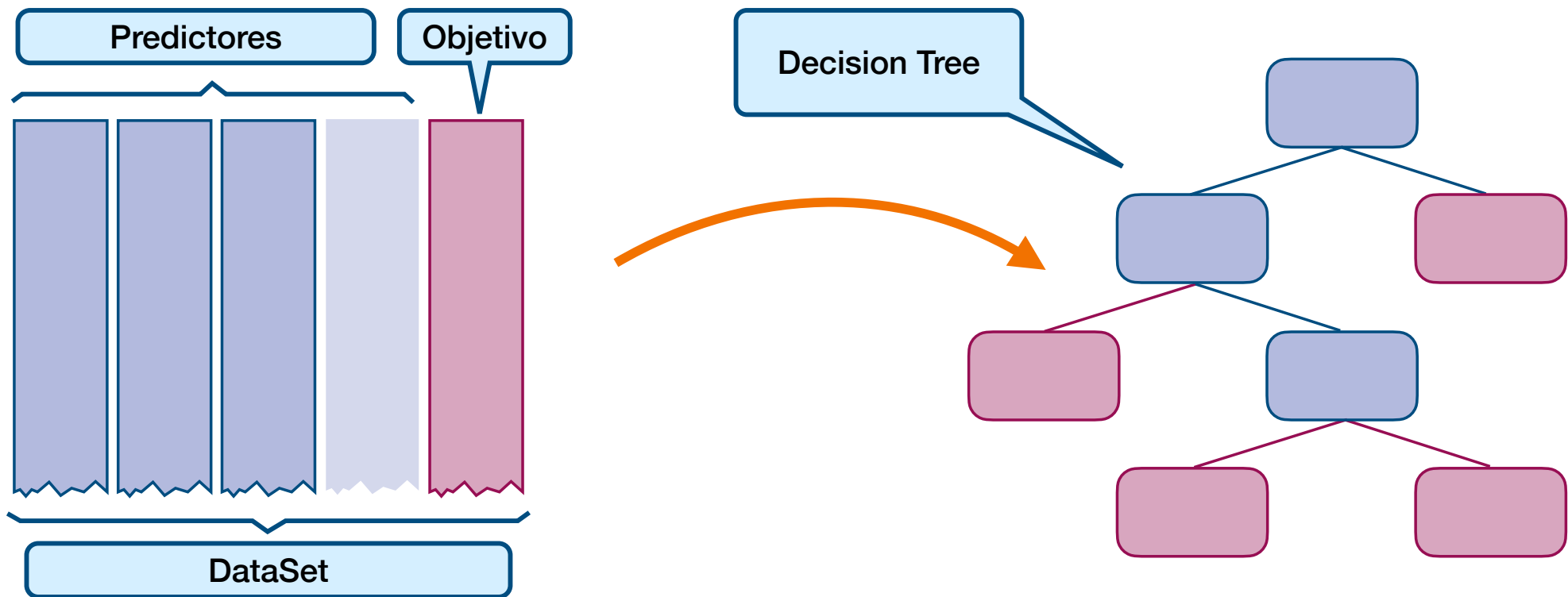
- Buen rendimiento en alta dimensión: Funciona muy bien con datos de muchas variables, por ejemplo imágenes.
- Manejo de relaciones no lineales: Gracias a los kernels (RBF, polinómico).
- Robustez frente a outliers: El soft margin permite ignorar ciertos puntos atípicos.
- Clasificación binaria y multiclase: Útil tanto para dos clases como para múltiples clases.

Desventajas

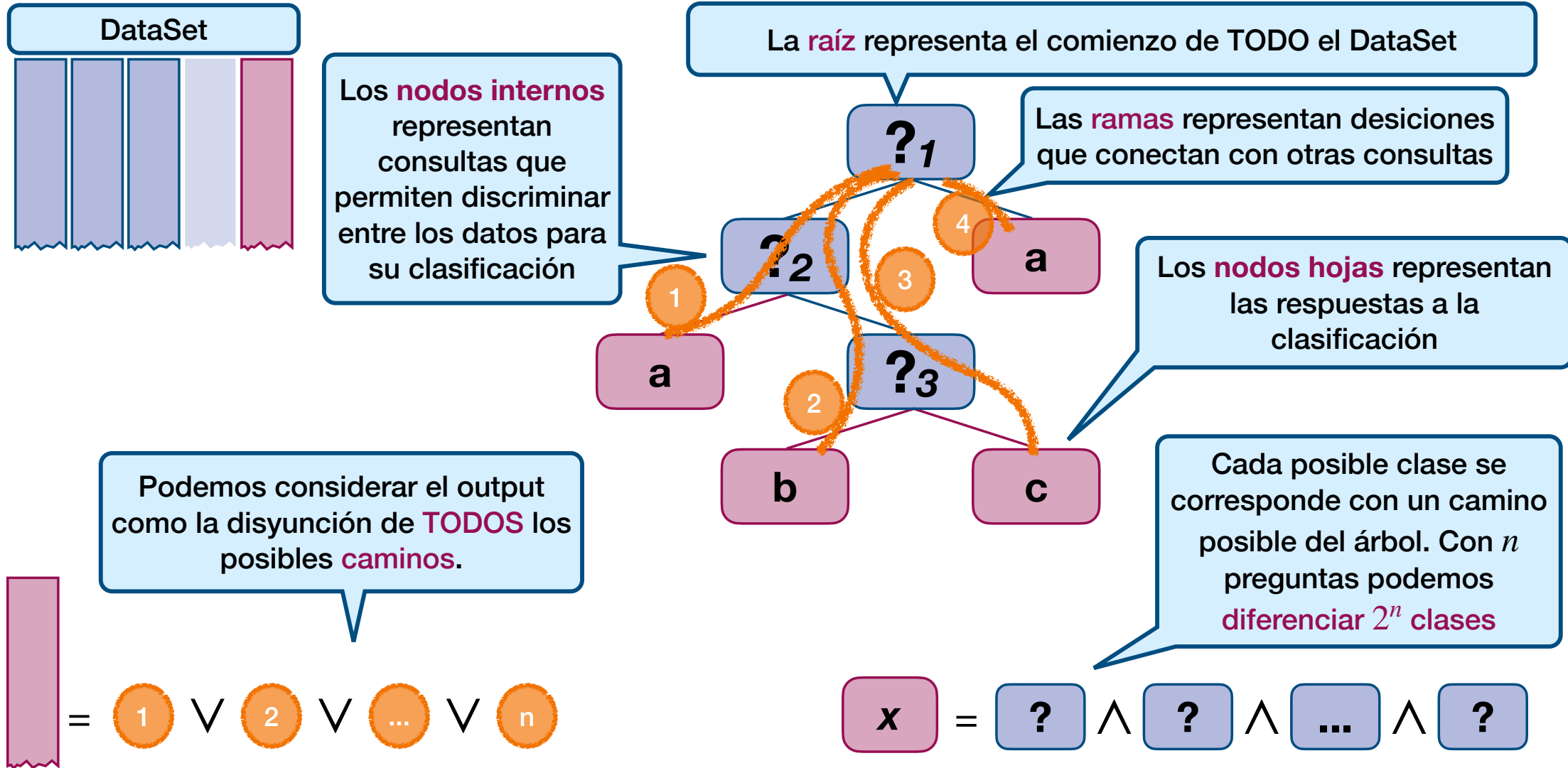
- Entrenamiento lento: Muy costoso en datasets grandes.
- Dificultad en la selección de parámetros: Elegir kernel y ajustar hiperparámetros suele ser complejo.
- Sensibilidad al ruido: Pierde rendimiento con datos ruidosos o clases solapadas.
- Poca interpretabilidad: Difícil de entender en espacios de alta dimensión.
- Necesidad de escalado: Requiere normalizar/estandarizar los datos para un buen desempeño.

Machine Learning - Árboles de decisión (DT)

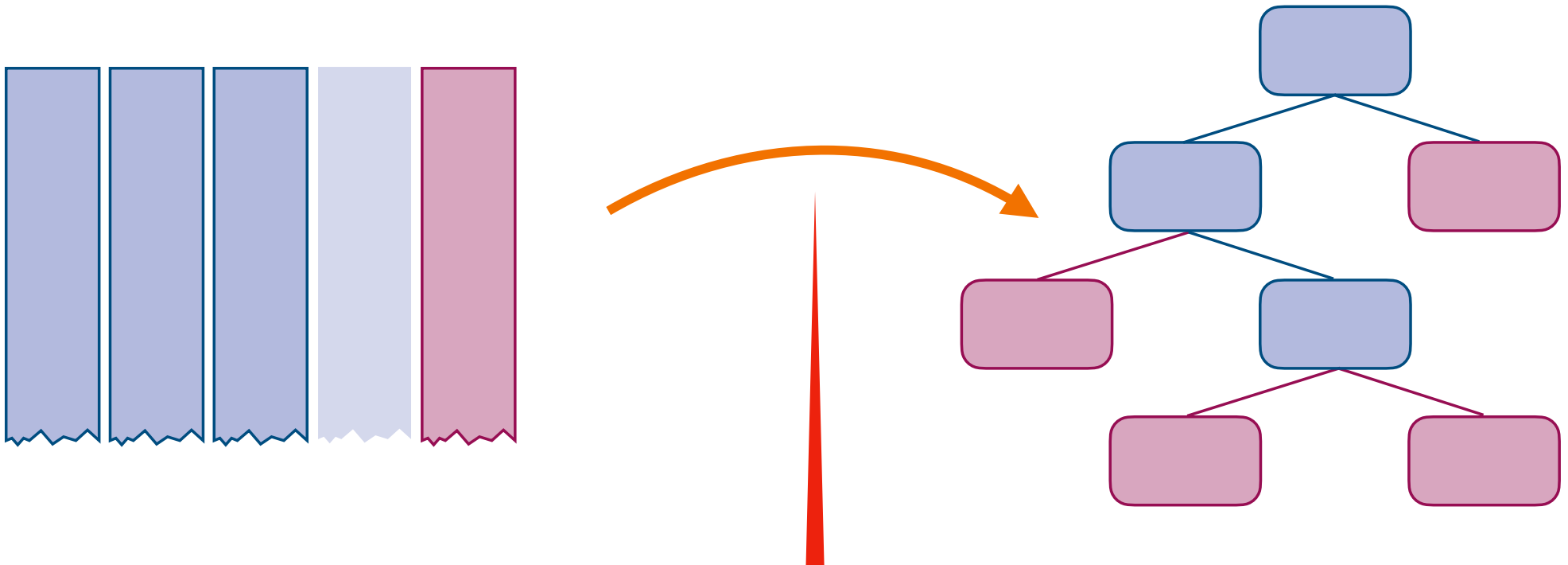
Los árboles de decisión (Decision Trees, DTs) son un método de aprendizaje supervisado no paramétrico utilizado para clasificación y regresión. Su objetivo es construir un modelo que prediga el valor de una variable objetivo mediante reglas de decisión simples derivadas de las características de los datos.



Machine Learning - Árboles de decisión (DT)

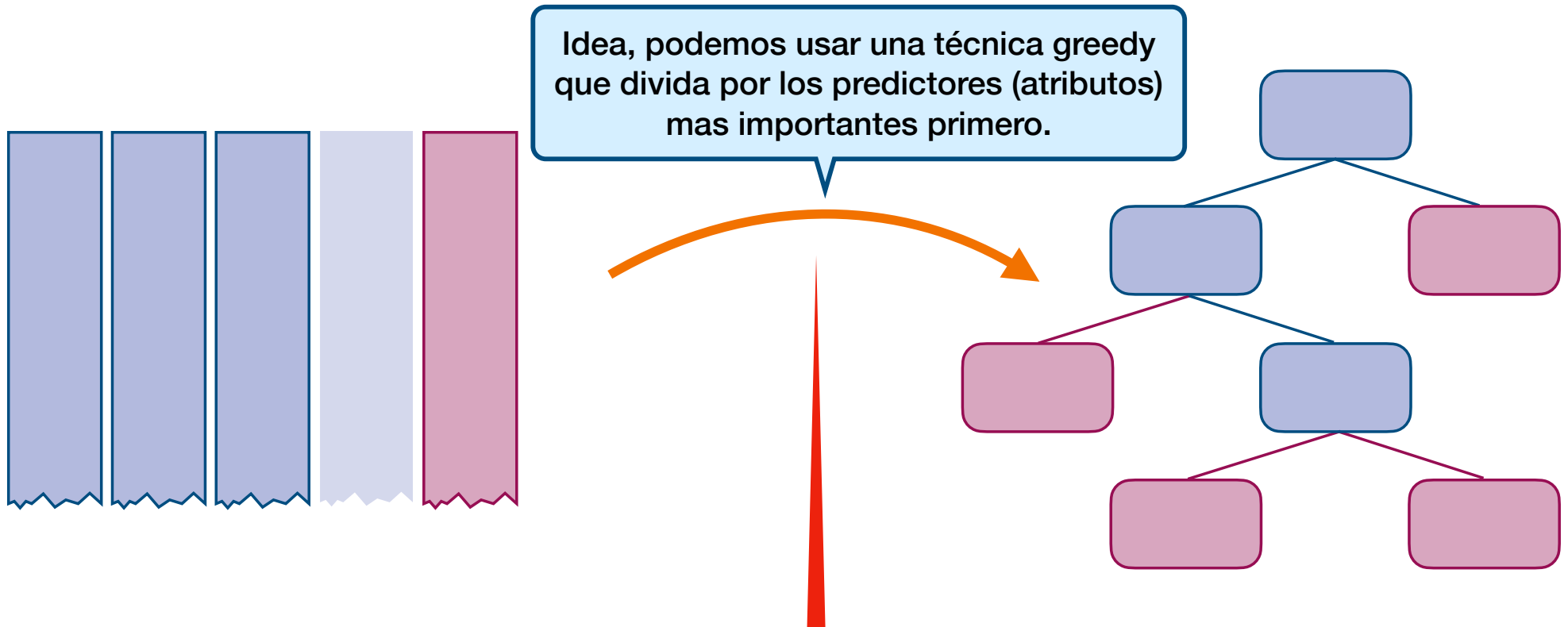


MLS Clasificación - Desicion Trees (DT)



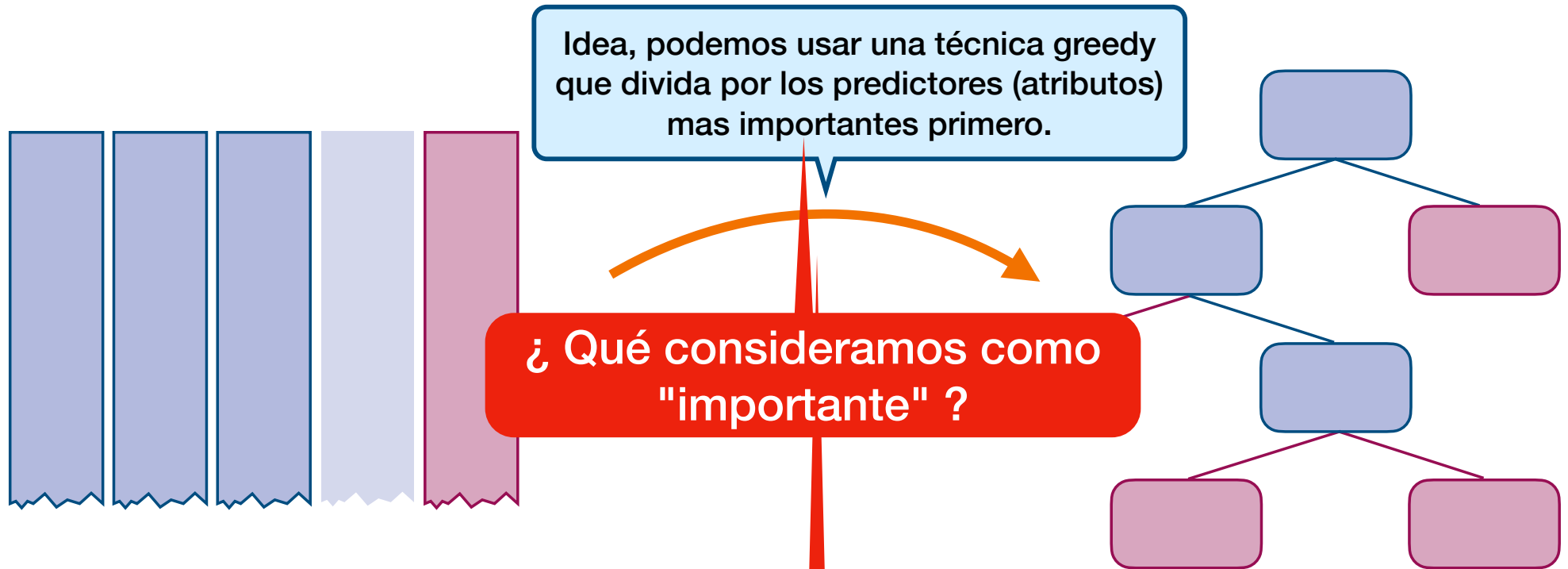
¿ Cómo podemos construir (aprender) un DT ?... de una forma optimal. La construcción del mejor DT (más eficiente) es un problema NP completo.

MLS Clasificación - Desicion Trees (DT)



¿ Cómo podemos construir (aprender) un DT ?... de una forma optimal. La construcción del mejor DT (más eficiente) es un problema NP completo.

MLS Clasificación - Desicion Trees (DT)



¿ Cómo podemos construir (aprender) un DT ?... de una forma optimal. La construcción del mejor DT (más eficiente) es un problema NP completo.

MLS Clasificación - Algoritmo CART (clasification and regresion tree)

```
function LEARN-DECISION-TREE(examples, attributes, parent_examples) returns a tree
  if examples is empty then return PLURALITY-VALUE(parent_examples)
  else if all examples have the same classification then return the classification
  else if attributes is empty then return PLURALITY-VALUE(examples)
  else
     $A \leftarrow \operatorname{argmax}_{a \in \text{attributes}} \text{IMPORTANCE}(a, \text{examples})$ 
    tree  $\leftarrow$  a new decision tree with root test A
    for each value v of A do
      exs  $\leftarrow \{e : e \in \text{examples} \text{ and } e.A = v\}$ 
      subtree  $\leftarrow$  LEARN-DECISION-TREE(exs, attributes – A, examples)
      add a branch to tree with label (A= v) and subtree subtree
  return tree
```

MLS Clasificación - DT - Función de Importancia

Índice **Gini** (para clasificación): El coeficiente gini mide la “pureza” de un nodo. Un nodo se dice “puro” (gini=0) si contiene todas muestras del mismo tipo:

$$Gini(t) = 1 - \sum_{i=1}^C \left(\frac{values[i][k]}{samples[i]} \right)^2$$

Cantidad de muestras de tipo k

Cantidad de muestras (en el nodo)

También se puede utilizar **entropía** como métrica para la división

MSE (para regresión): Error mínimo cuadrado

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y})^2$$

Cantidad de muestras (en el nodo)

Machine Learning - DT - Ejemplo Iris Dataset

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

#Cargamos el dataset Iris
iris = load_iris()
X = iris.data # características
y = iris.target # etiquetas (setosa, versicolor, virginica)

# Entrenamos el modelo de DT
clf = DecisionTreeClassifier(criterion="gini", max_depth=4, random_state=42)
clf.fit(X, y)

# Visualizamos
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=False, feature_names=iris.feature_names,
class_names=iris.target_names)
plt.show()

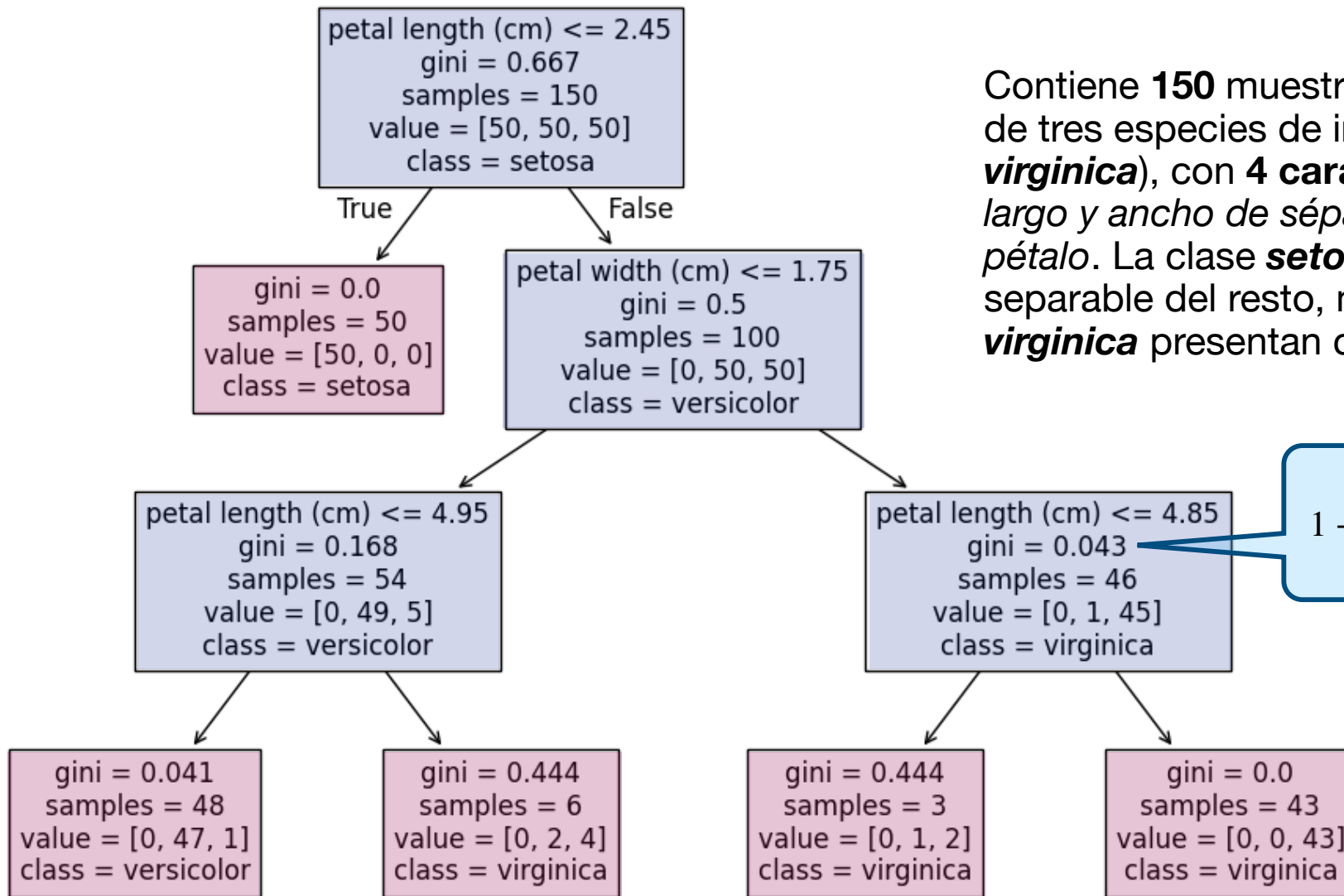
#Preguntamos por una instancia nueva
nueva_flor = [[5.0, 3.2, 1.2, 0.2]]
prediccion = clf.predict(nueva_flor)
print("La flor predicha es:", iris.target_names[prediccion])
```

Utilizamos **Gini** como
métrica para las
divisiones

Profundidad máxima
del árbol
(hiperparámetro)

Machine Learning - DT - Ejemplo Iris Dataset

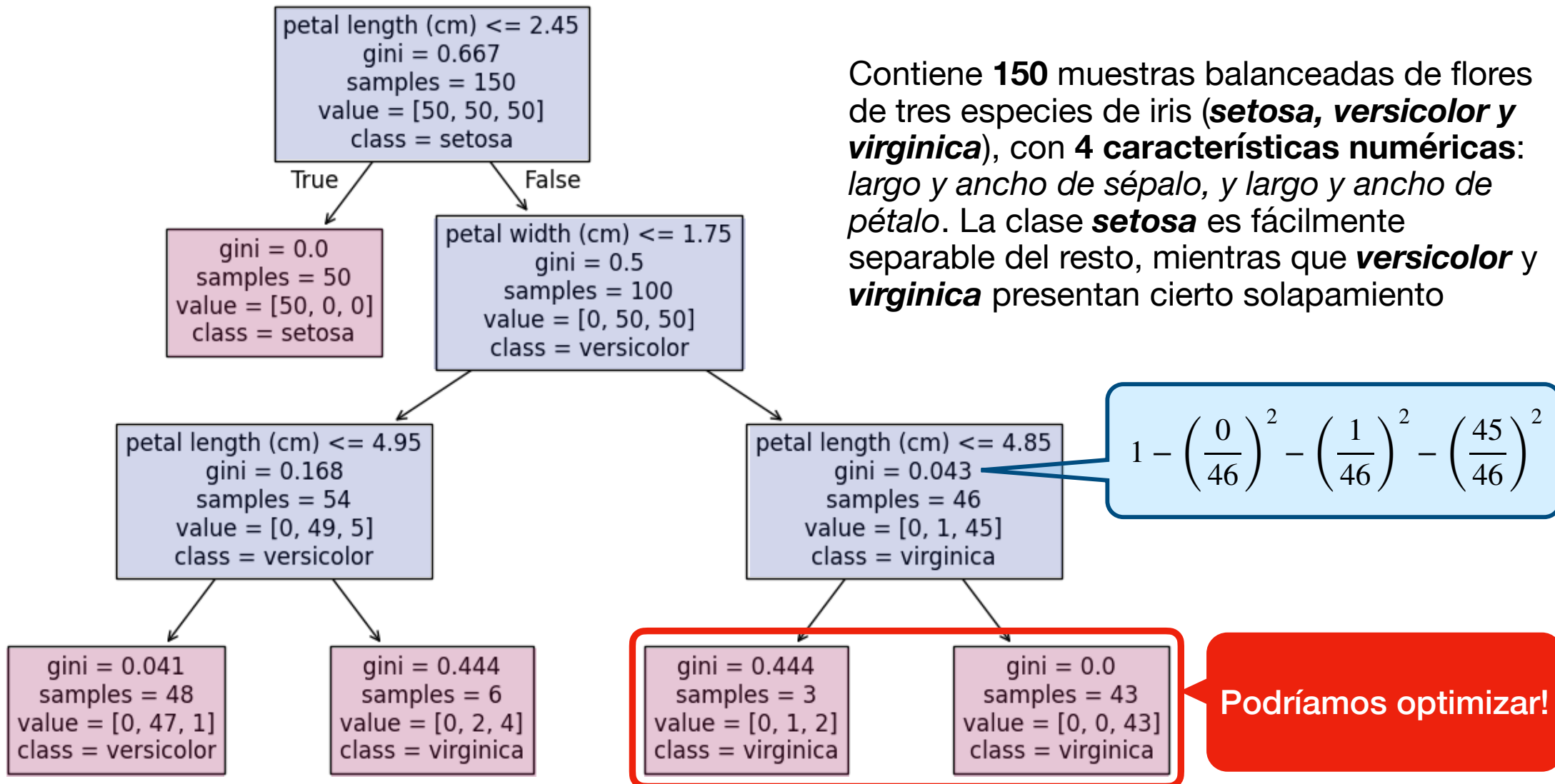
Contiene **150** muestras balanceadas de flores de tres especies de iris (**setosa**, **versicolor** y **virginica**), con **4 características numéricas**: largo y ancho de sépalo, y largo y ancho de pétalo. La clase **setosa** es fácilmente separable del resto, mientras que **versicolor** y **virginica** presentan cierto solapamiento



$$1 - \left(\frac{0}{46}\right)^2 - \left(\frac{1}{46}\right)^2 - \left(\frac{45}{46}\right)^2$$

Machine Learning - DT - Ejemplo Iris Dataset

Contiene **150** muestras balanceadas de flores de tres especies de iris (**setosa**, **versicolor** y **virginica**), con **4 características numéricas**: largo y ancho de sépalo, y largo y ancho de pétalo. La clase **setosa** es fácilmente separable del resto, mientras que **versicolor** y **virginica** presentan cierto solapamiento



Machine Learning - DT - Pruning

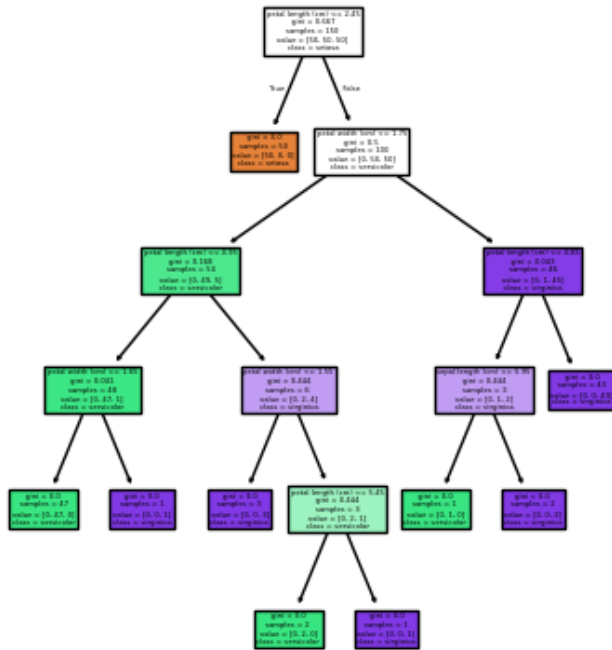
La poda es una técnica utilizada en DT para reducir su complejidad y evitar el *overfitting*. Un árbol muy profundo puede ajustarse demasiado a los datos de entrenamiento capturando ruido en lugar de patrones generales. **Pruning** consiste en **eliminar** ramas o nodos que aportan poca mejora en la predicción, logrando así un modelo más simple, generalizable y eficiente.

Existen dos enfoques principales:

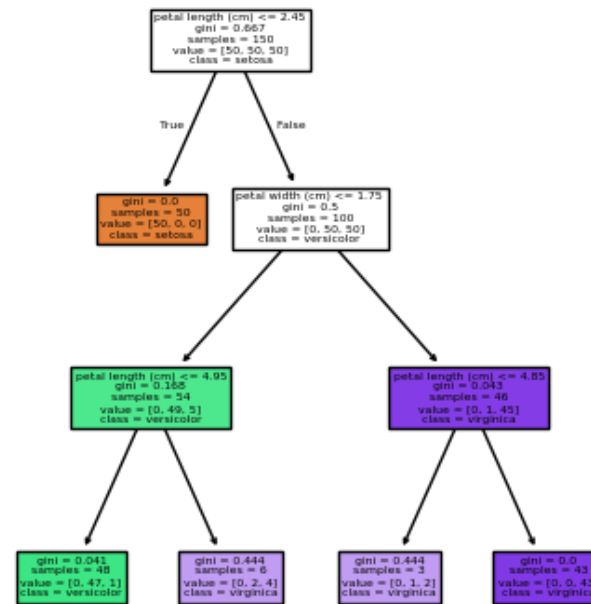
- **Pre-pruning (poda temprana)**: se detiene el crecimiento del árbol antes de que alcance su máxima profundidad, aplicando criterios como un número mínimo de muestras por nodo o una ganancia mínima de información.
- **Post-pruning (poda posterior)**: primero se entrena un árbol grande y luego se recortan ramas evaluando el impacto en un conjunto de validación o mediante métricas como error o complejidad.

Machine Learning - DT - Pruning - Ejemplo

Árbol sin poda



Árbol con pre-pruning (max_depth=3)



Árbol con post-pruning (ccp_alpha=0.0297)

