Aprendizaje automático

K-NN y árboles de decisión

FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Agenda

- Clasificación por vecinos más cercanos (K-NN)
- Clasificación por árboles de decisión

Aprendizaje Supervisado

Aprendizaje supervisado

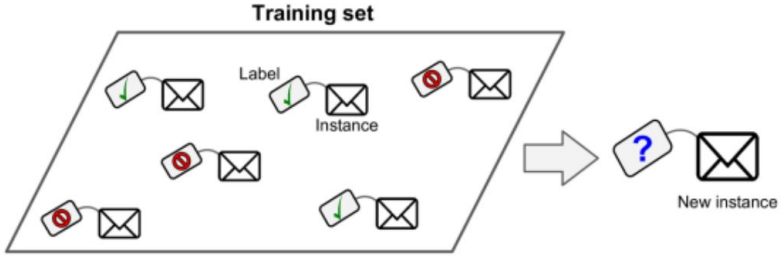
- Pares de datos de entrada-salida generados por humanos

- Según el tipo de salida dos posibles problemas:

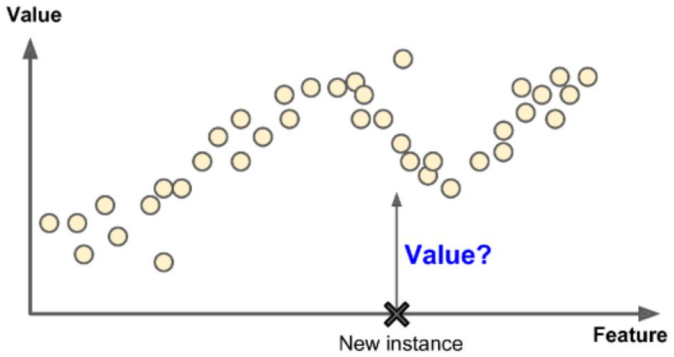
  salida categórica: clasificación

  salida numérica: regresión

Problema de Clasificación



Problema de Regresión



Components of learning

Formalization:

- Input: $\mathbf{x}$ (*customer application*)
- Output: y (*good/bad customer?*)
- Target function: $f : \mathcal{X} \rightarrow \mathcal{Y}$ (*ideal credit approval formula*)
- Data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)$ (*historical records*)

$\downarrow \quad \downarrow \quad \downarrow$

- Hypothesis: $g : \mathcal{X} \rightarrow \mathcal{Y}$ (*formula to be used*)

Visión unificada: aproximación de funciones

Objetivo del aprendizaje automático

- Construir $f(X)$ a partir de **datos conocidos**

Generalización

- $y \approx f(X)$ para x **desconocida**

Premisas

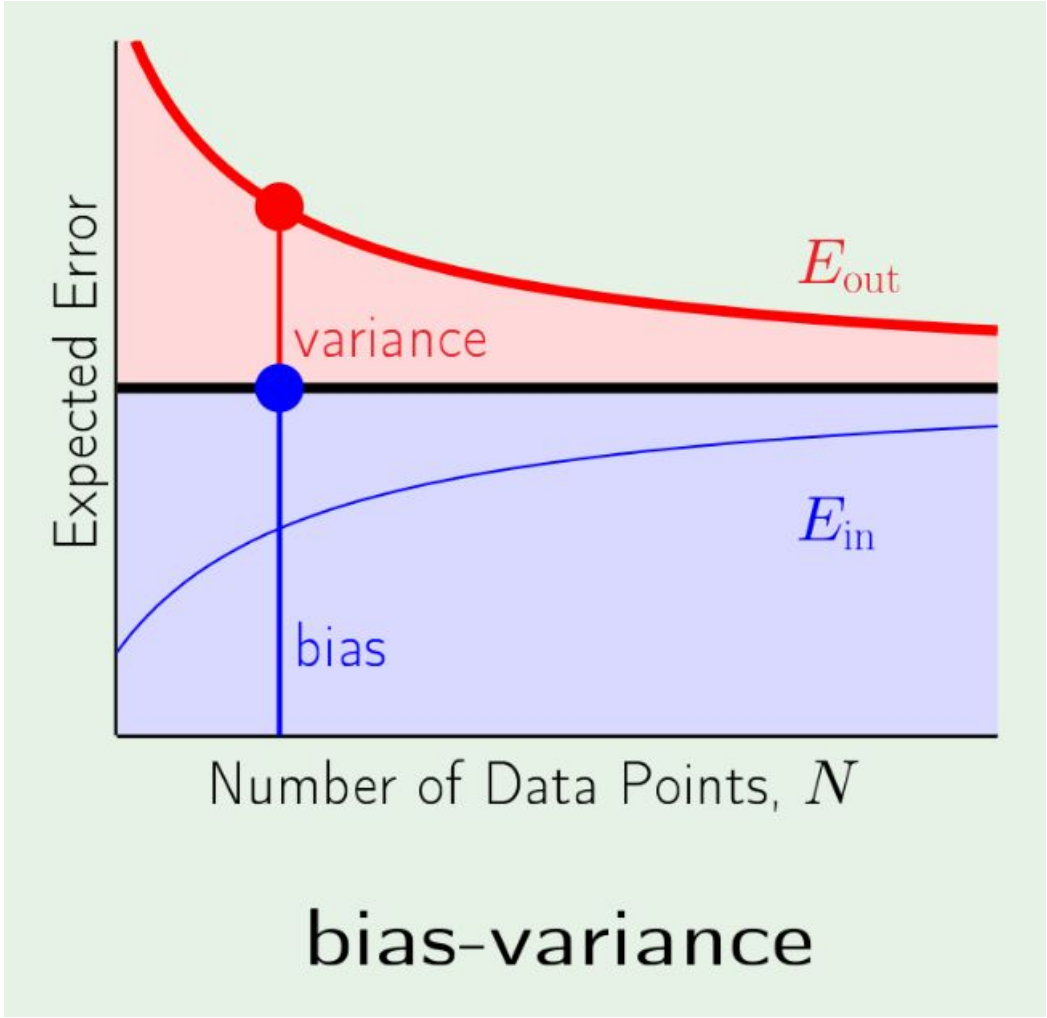
- No **existe** una $\tilde{f}(X)$ **real** o **correcta**!
- No podemos dar **garantías**
- Sí podemos dar **evidencias**
- Podemos usar **criterios** filosóficos (ej., Occam)

Ejemplo de problema de clasificación

- Objetivo: determinar clase de cada muestra
  - ¿A qué especie pertenece una flor ($\mathbf{x}$)?
  - Iris (Fisher, 1936) ($g(\mathbf{x})$)
- Se aprende con ejemplos previamente clasificados $((\mathbf{x}_1,y_1), \ldots ,(\mathbf{x}_N,y_N))$
- Se aplica conocimiento a nuevos ejemplos sin clasificar

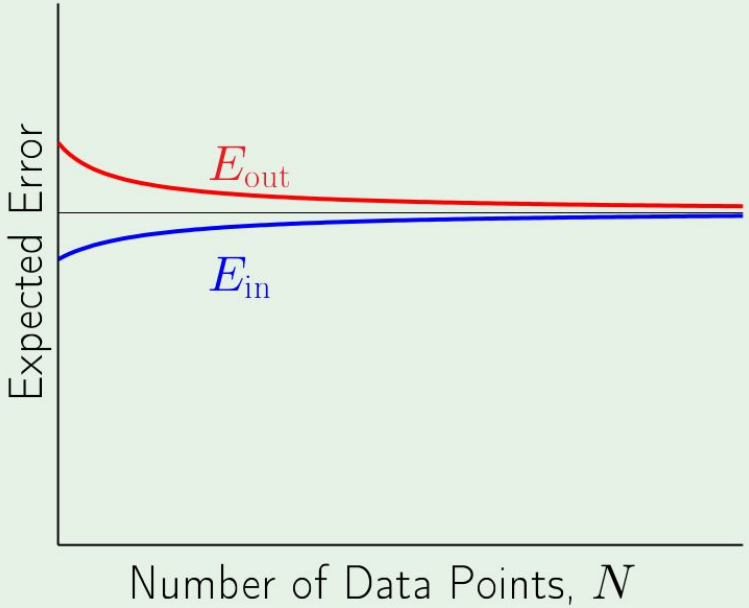
Fotografía de una flor de la familia “Iris”

Sub y sobre ajuste

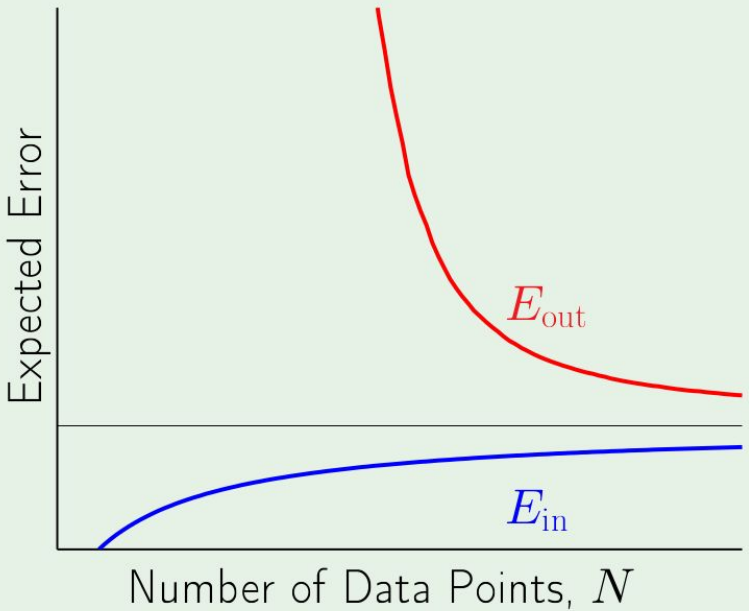


bias-variance

The curves



Simple Model



Complex Model

Subajuste (underfitting)

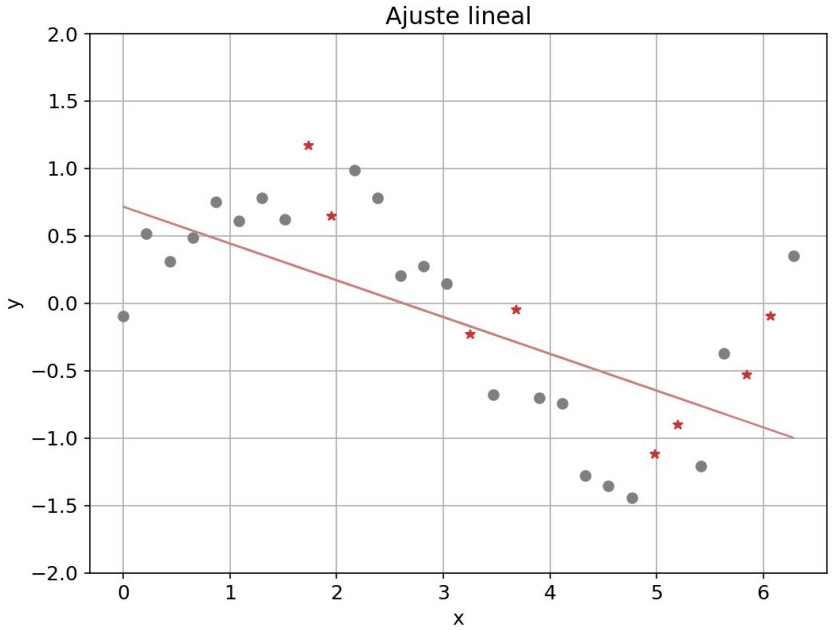
Escenario

- Modelo demasiado simple
- Muy pocos parámetros

Consecuencia

- No se ajusta bien a nada
- No captura estructura

Ocurre cuando el modelo es demasiado simple para aprender la estructura subyacente de los datos.



Posibles soluciones

- Seleccionar un modelo con más parámetros
- Introducir mejores características en el algoritmo de aprendizaje (ingeniería de características)
- Reducir las restricciones del modelo (por ejemplo, reducir los hiperparámetros de regularización)

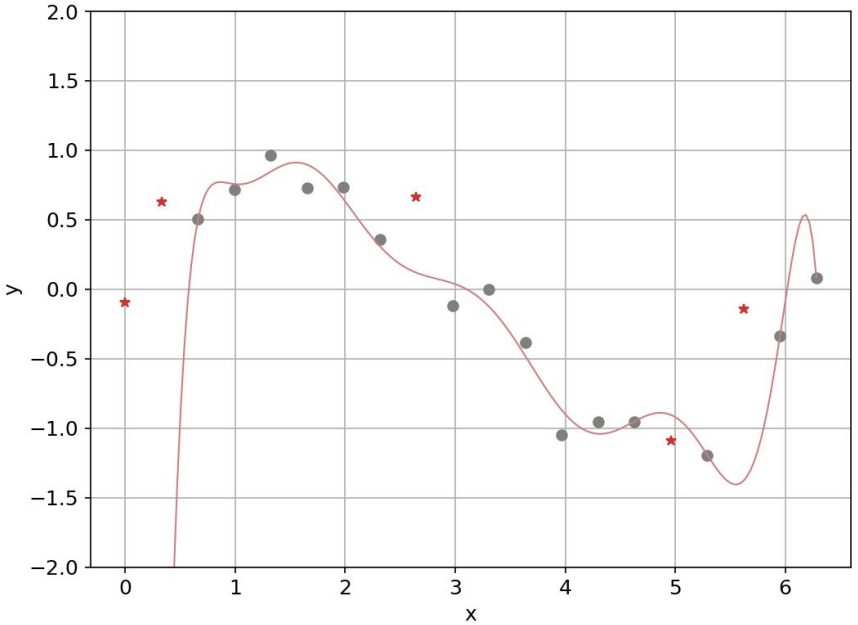
Sobreajuste (overfitting)

Escenario

- g(**x**) demasiado compleja
- Demasiados parámetros
- Pocas muestras

Consecuencia

- Captura particularidades
- No captura generalidades



El modelo funciona bien en los datos de entrenamiento, pero no generaliza bien.

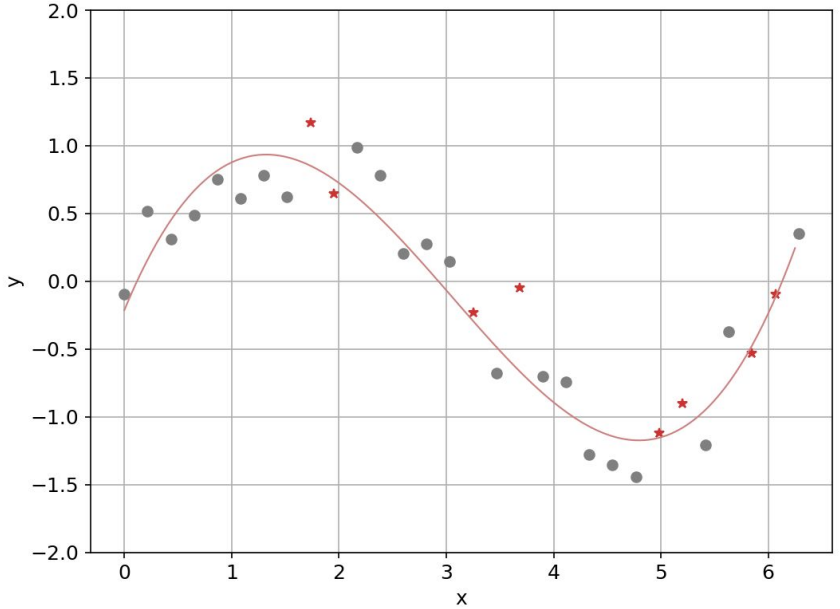
Siempre se puede construir un modelo que funcione perfecto con los datos de entrenamiento.

Posibles soluciones

- Simplificar el modelo seleccionando uno con menos parámetros, reduciendo el número de atributos en los datos de entrenamiento o restringiendo el modelo.
- Reunir más datos de entrenamiento
- Reducir el ruido en los datos de entrenamiento

Ajuste óptimo

- Complejidad óptima
- Captura toda la estructura
- No captura particularidades
- Cómo identificar modelo ideal?
  - Selección de modelos
  - Validación cruzada
- Depende de cantidad de datos de entrenamiento disponibles!
- Depende del problema



Clasificación por vecinos más cercanos (k-NN)

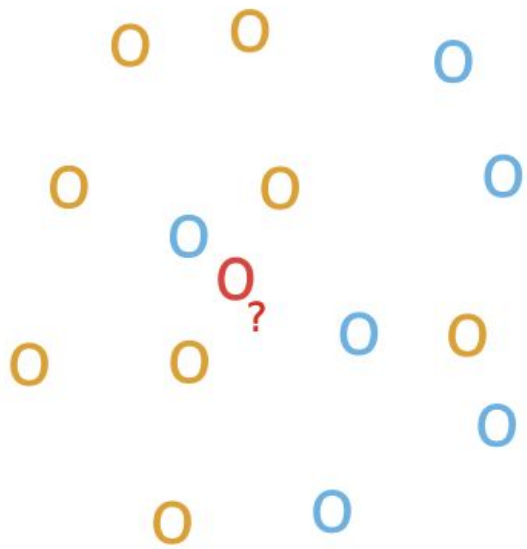
Clasificación por vecinos más cercanos (k-NN)

- Método de **aprendizaje supervisado** utilizado para clasificación y regresión
- No paramétrico: ninguna suposición sobre la forma de g(**x**)
- Basado en instancias: no hay una fase de entrenamiento explícita: memoriza los datos de entrenamiento como conocimiento para la fase de predicción.
- Idea principal: puntos de datos similares tienden a tener etiquetas similares
- Entrenamiento mínimo pero inferencia (muy) costosa!

Clasificación por vecinos más cercanos (k-NN)

Se asigna la clase dominante dentro de los k puntos más cercanos del conjunto de entrenamiento.

Dado un nuevo punto **x** (conjunto de test):

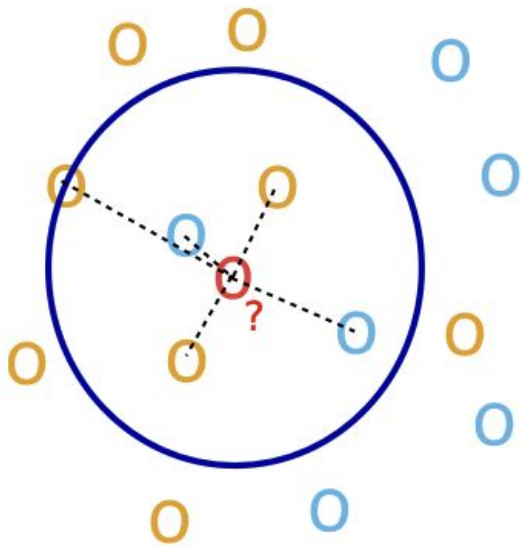


Clasificación por vecinos más cercanos (k-NN)

Se asigna la clase dominante dentro de los k puntos más cercanos del conjunto de entrenamiento.

Dado un nuevo punto **x** (conjunto de test):

1. Se buscan los k vecinos más cercanos en el conjunto de entrenamiento según cierta distancia



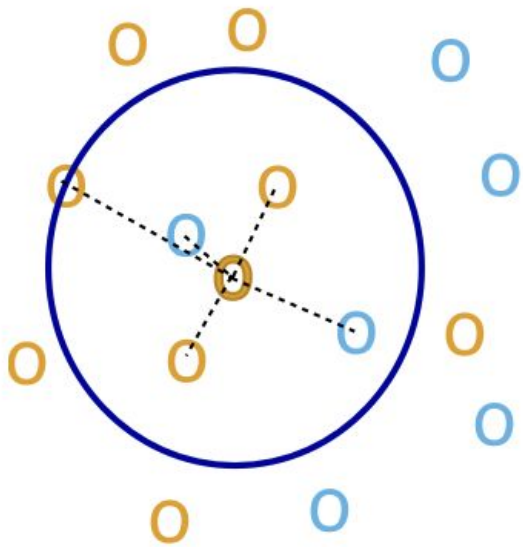
Clasificación por vecinos más cercanos (k-NN)

Se asigna la clase dominante dentro de los k puntos más cercanos del conjunto de entrenamiento.

Dado un nuevo punto **x** (conjunto de test):

1. Se buscan los k vecinos más cercanos en el conjunto de entrenamiento según cierta distancia

2. Se clasifica a **x** con la etiqueta dominante entre los k vecinos



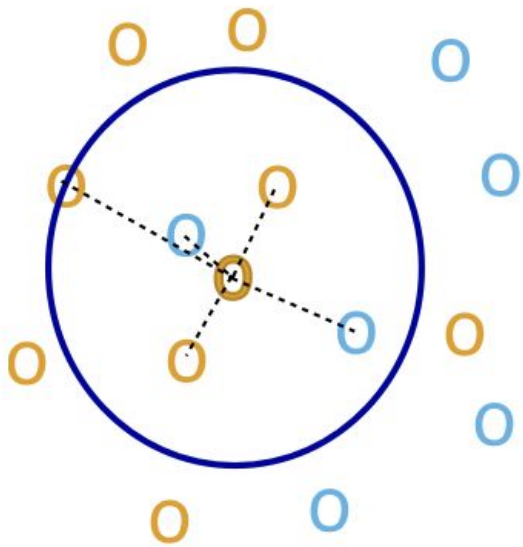
Clasificación por vecinos más cercanos (k-NN)

Se asigna la clase dominante dentro de los k puntos más cercanos del conjunto de entrenamiento.

Dado un nuevo punto **x** (conjunto de test):

1. Se buscan los k vecinos más cercanos en el conjunto de entrenamiento según cierta distancia

2. Se clasifica a **x** con la etiqueta dominante entre los k vecinos



k y la distancia son hyperparámetros del método.

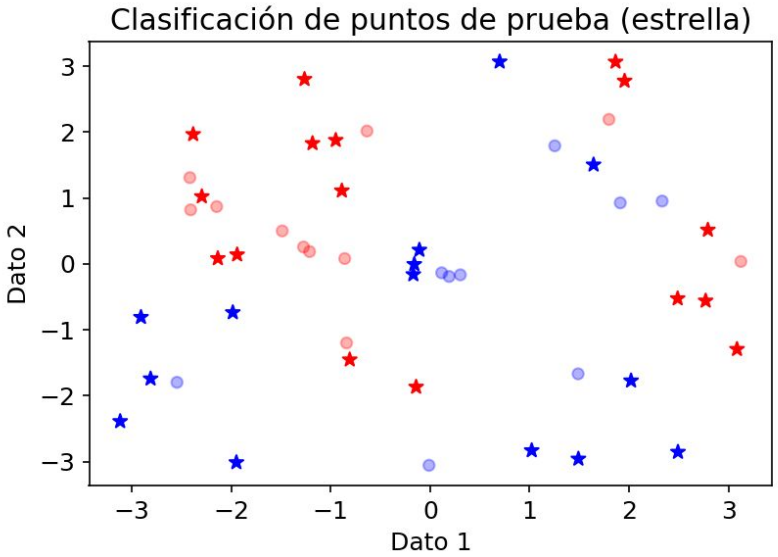
Algoritmo

Para un valor de k, una métrica d y un dato nuevo **x**:

1. Se mide la distancia **segùn d** entre **x** y todos los puntos conocidos etiquetados
2. Retenemos los **k vecinos** más cercanos según d (conjunto A)
3. Se calcula la distribución condicional de cada clase

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

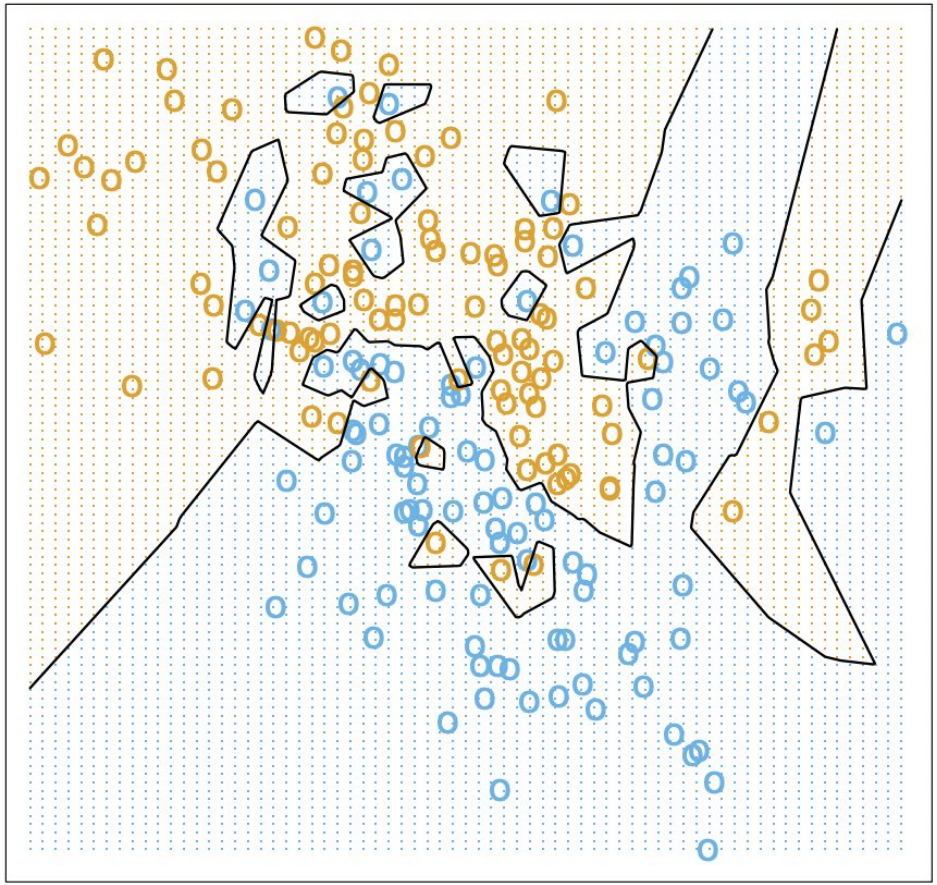
4. Se asigna la clase con mayor probabilidad a **x**



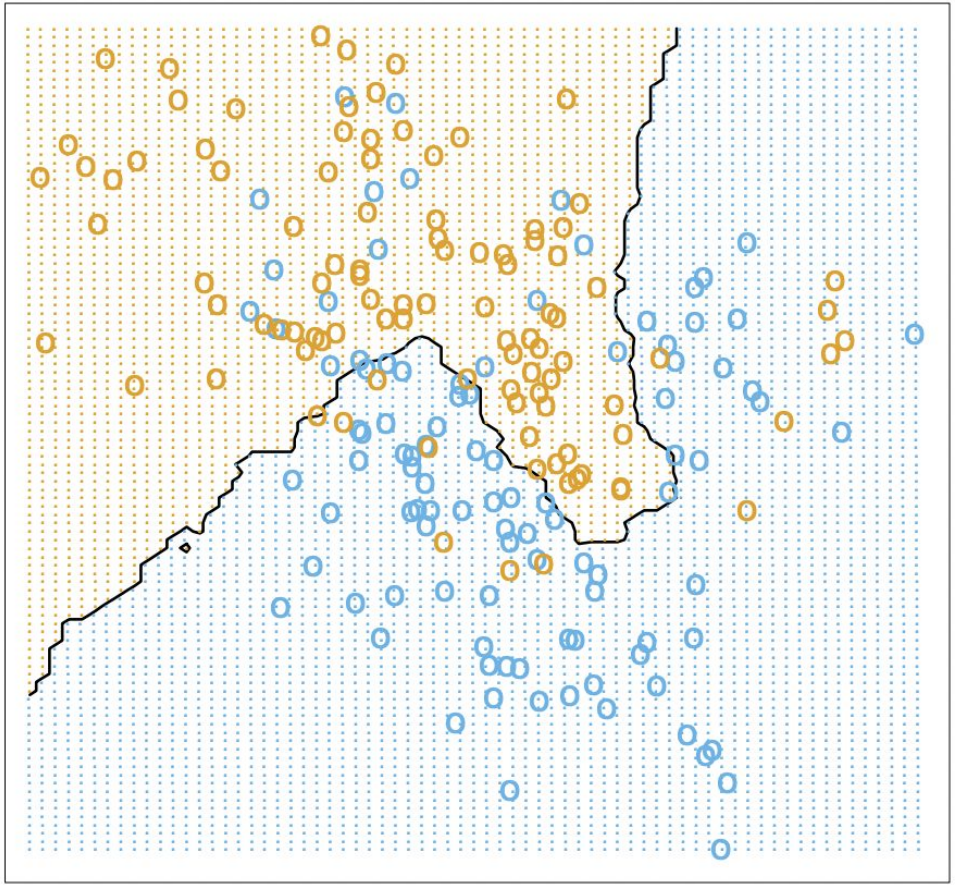
k=1: copiamos clase de punto más cercano

Hiperparámetro k (cantidad de vecinos)

1-NN



15-NN



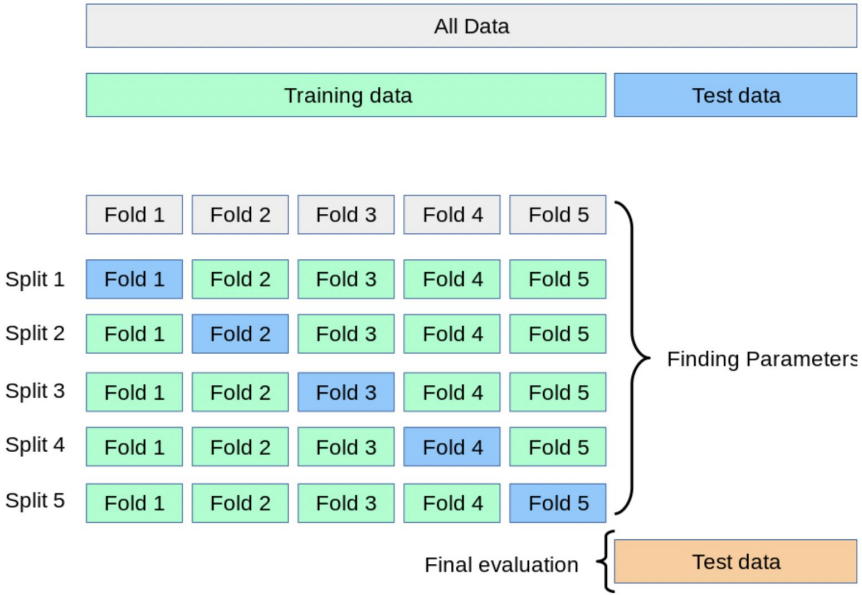
Hiperparámetro k (cantidad de vecinos)

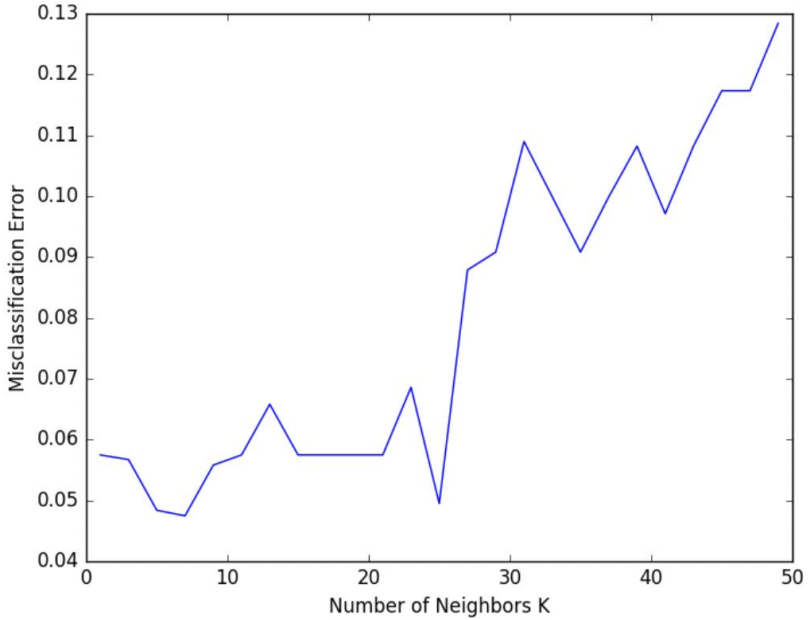
- k muy chico ($k \to 1$):
  - sobre ajuste: se ajusta muy bien a los datos de entrenamiento y no es bueno generalizando
  - las fronteras de decisión son irregulares

- k grande:
  - tiende a un mejor comportamiento
  - si k es muy grande se empiezan a considerar puntos que no son vecinos
  - subajuste
  - las fronteras de decisión son más homogéneas

Cómo elegir el k óptimo?

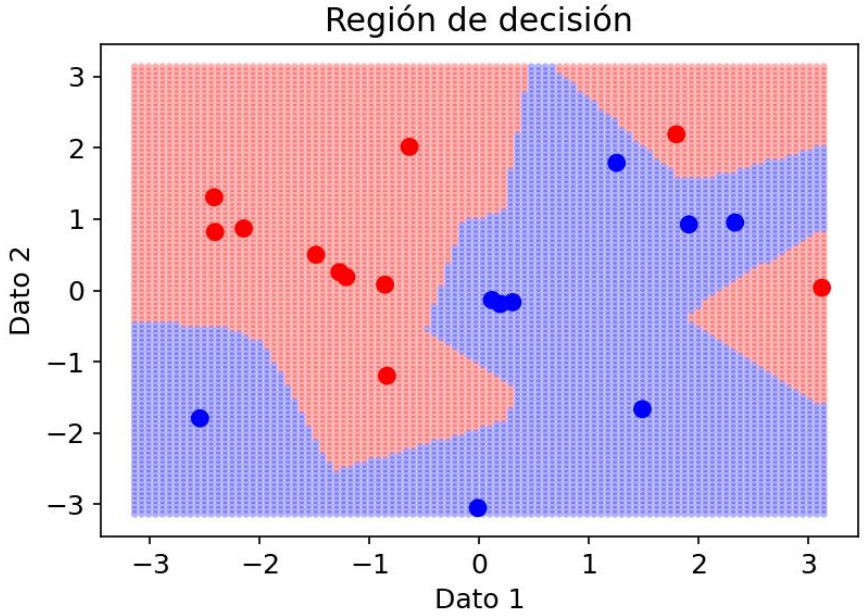
Usar alguna técnica de optimización de hiperparámetros para encontrar k óptimo (e.g. validación cruzada)



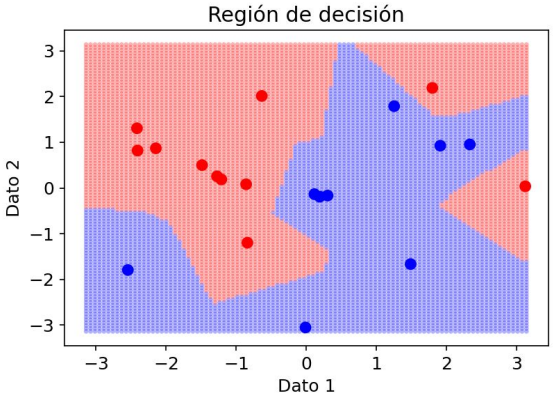


Vecinos más cercanos: función y regiones de decisión

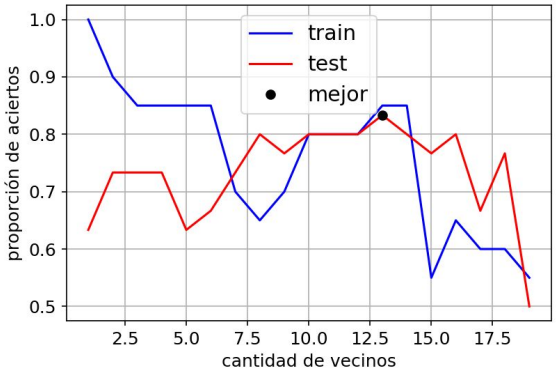
- Cómo es g(X) en el espacio?
- g(X) es constante en cada región
- El valor de g(X) indica la clase
- C clases, C regiones
- Frontera de decisión
    - frontera entre dos regiones

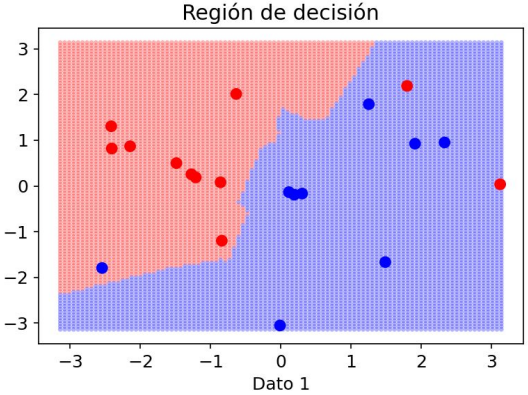


Complejidad del modelo y número K de vecinos



K=1





K=3

Intuitivamente: la complejidad del modelo se refleja en la complejidad de las fronteras de decisión

k-NN ponderado

Los ejemplos de una clase más frecuente tienden a dominar la predicción de la clasificación del nuevo punto.

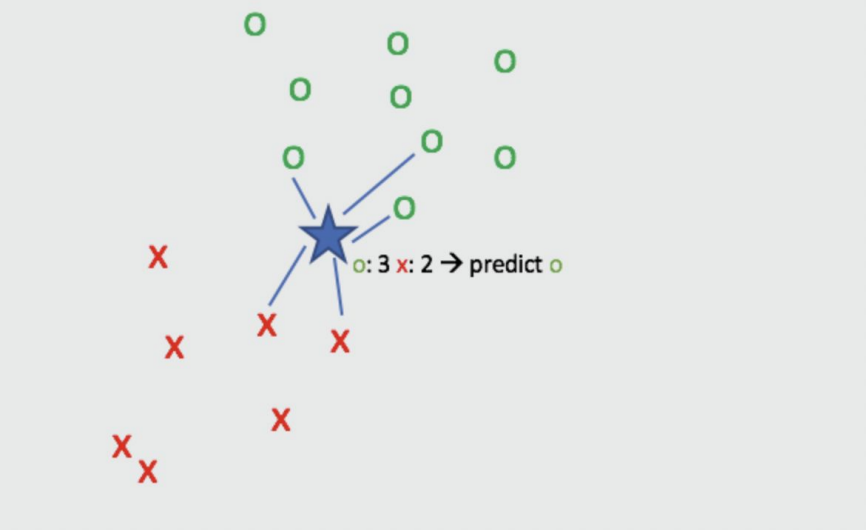
POSIBLE SOLUCION: ponderar la clasificación

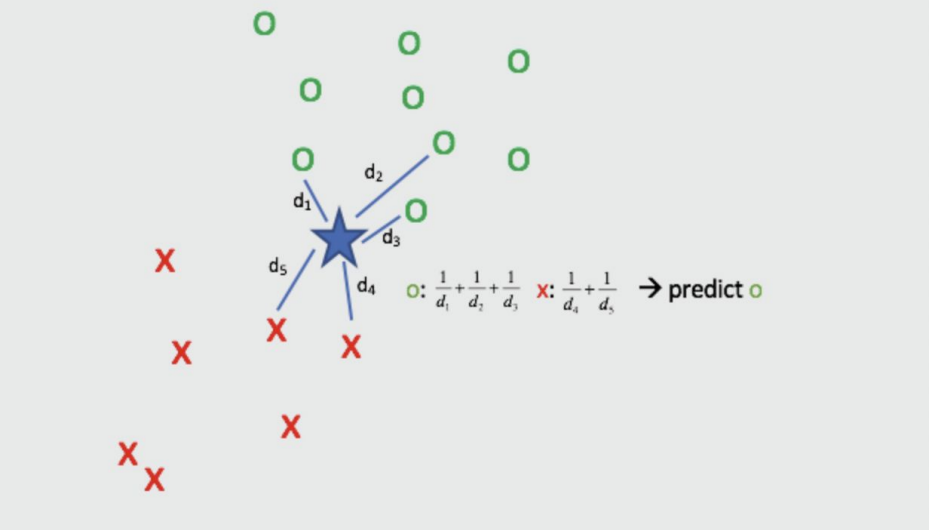
Ponderar la clasificación, teniendo en cuenta la distancia del nuevo punto con respecto a cada uno de sus k vecinos más próximos : la clase de cada uno de estos k vecinos más próximos se multiplica por un peso proporcional a la inversa de la distancia de este vecino al punto que se quiere clasificar.

k-NN ponderado

Los ejemplos de una clase más frecuente tienden a dominar la predicción de la clasificación del nuevo punto.

SOLUCIÓN: ponderar la clasificación





sklearn.neighbors.KNeighborsClassifier

sklearn.neighbors.KNeighborsClassifier

class sklearn.neighbors.**KNeighborsClassifier**(*n_neighbors=5*, ***, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*) [source]

Classifier implementing the k-nearest neighbors vote.

Read more in the User Guide.

Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

weights : *{'uniform', 'distance'}, callable or None, default='uniform'*
Weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

Refer to the example entitled Nearest Neighbors Classification showing the impact of the `weights` parameter on the decision boundary.

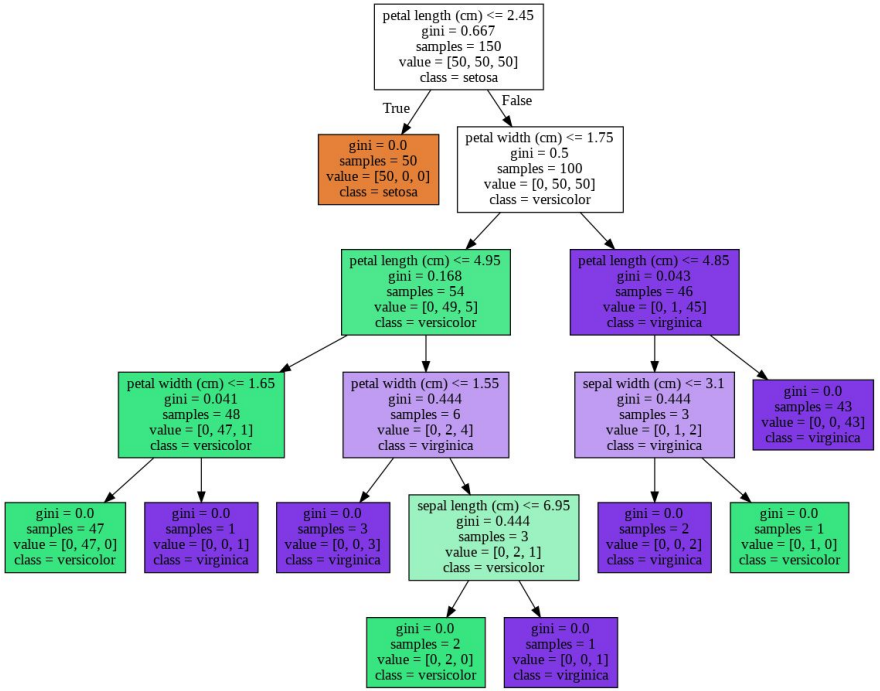
Clasificación por árboles de decisión

Árboles de decisión

- Método de **aprendizaje supervisado** utilizado para clasificación y regresión
- No paramétrico: ninguna suposición sobre la forma de g(**x**)
- Idea principal: predecir el valor de una variable objetivo mediante el aprendizaje de reglas de decisión sencillas inferidas a partir de las características de los datos de entrenamiento.

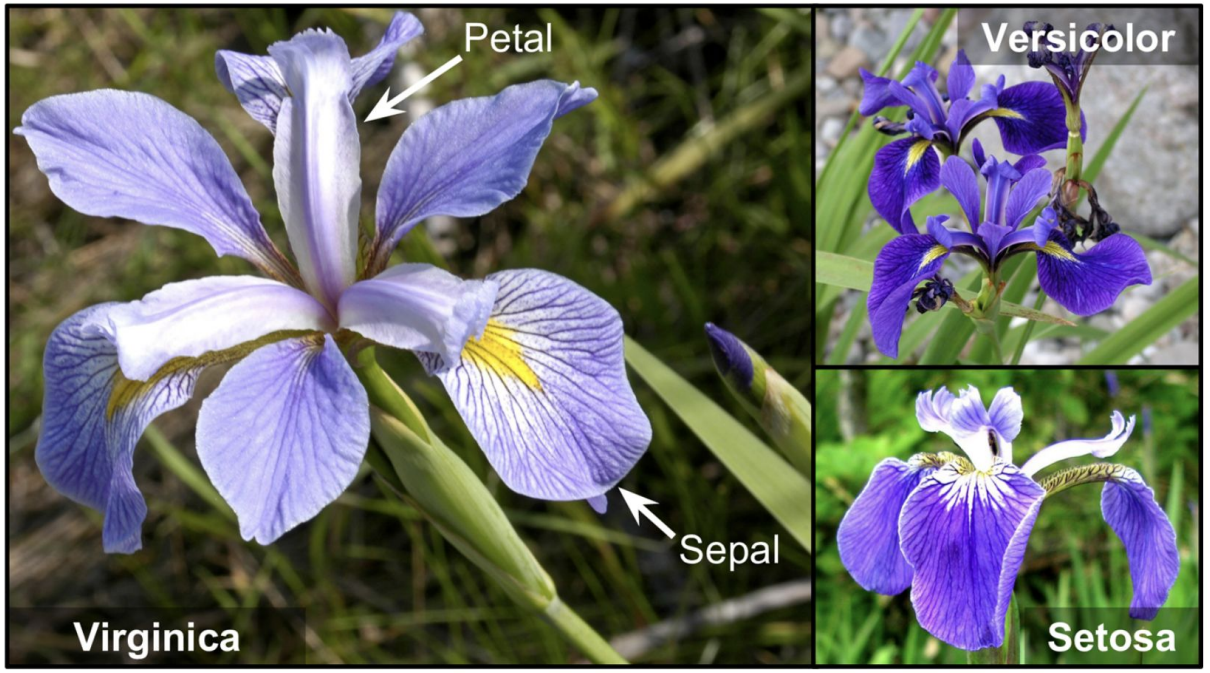
Árboles de decisión

- Clase/valor se decide por secuencia de preguntas
- Muy intuitivos: fácil de entender e interpretar
- Los árboles se pueden visualizar
- No se precisa tratar los datos previo a la clasificación/regresión
- **Muy efectivos!**



Dataset Iris

Conjunto de datos de sklearn que contiene la longitud y ancho de los sépalos y pétalos de 150 flores de iris de tres especies diferentes: Iris setosa, Iris versicolor e Iris virginica

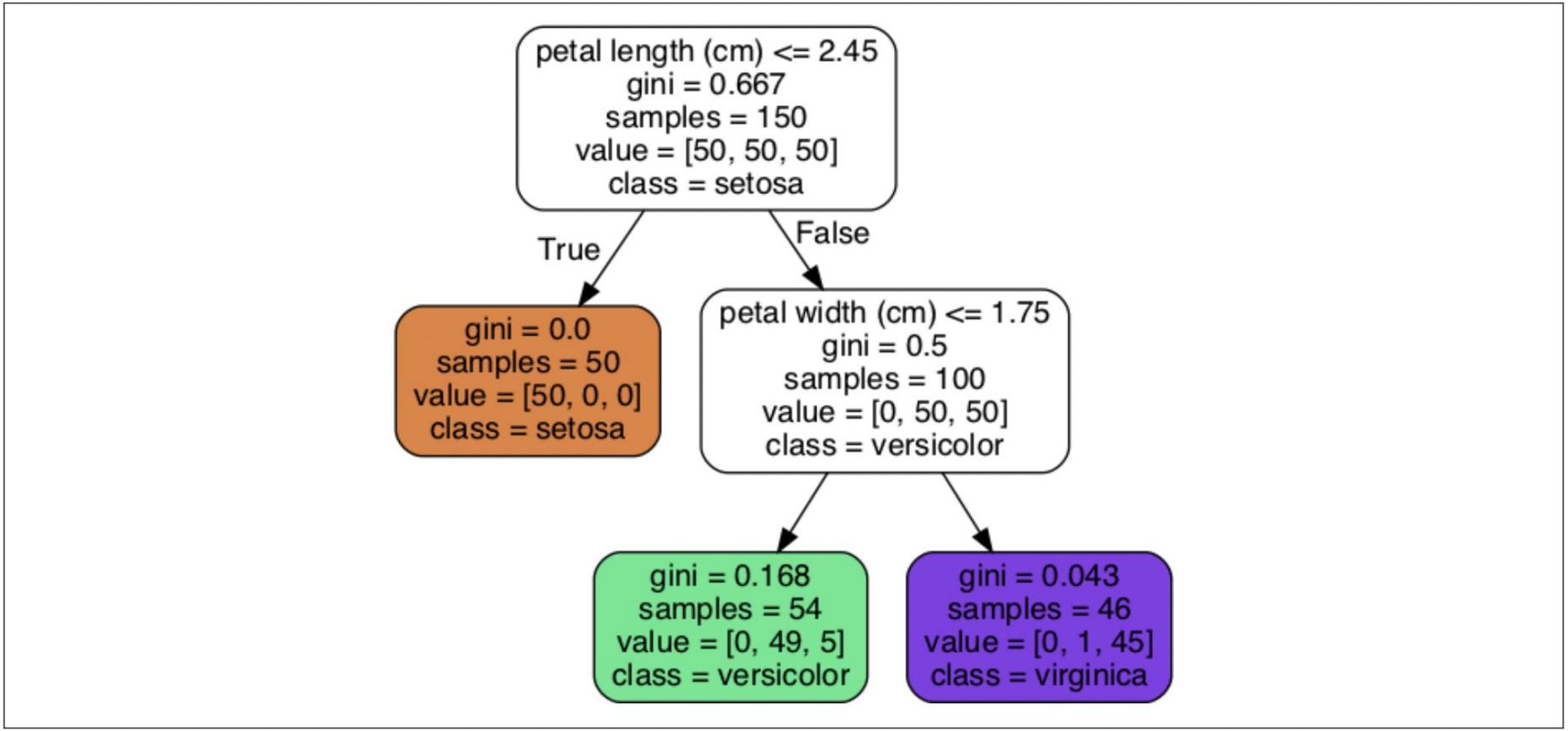


Ejemplo de clasificación para el dataset iris

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier

iris = load_iris(as_frame=True)
X_iris = iris.data[["petal length (cm)", "petal width (cm)"]].values
y_iris = iris.target

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)
tree_clf.fit(X_iris, y_iris)
```

```
from sklearn.tree import export_graphviz

export_graphviz(
        tree_clf,
        out_file=str(IMAGES_PATH / "iris_tree.dot"),  # path differs in the book
        feature_names=["petal length (cm)", "petal width (cm)"],
        class_names=iris.target_names,
        rounded=True,
        filled=True
    )
```

```
from graphviz import Source

Source.from_file(IMAGES_PATH / "iris_tree.dot")  # path differs in the book
```

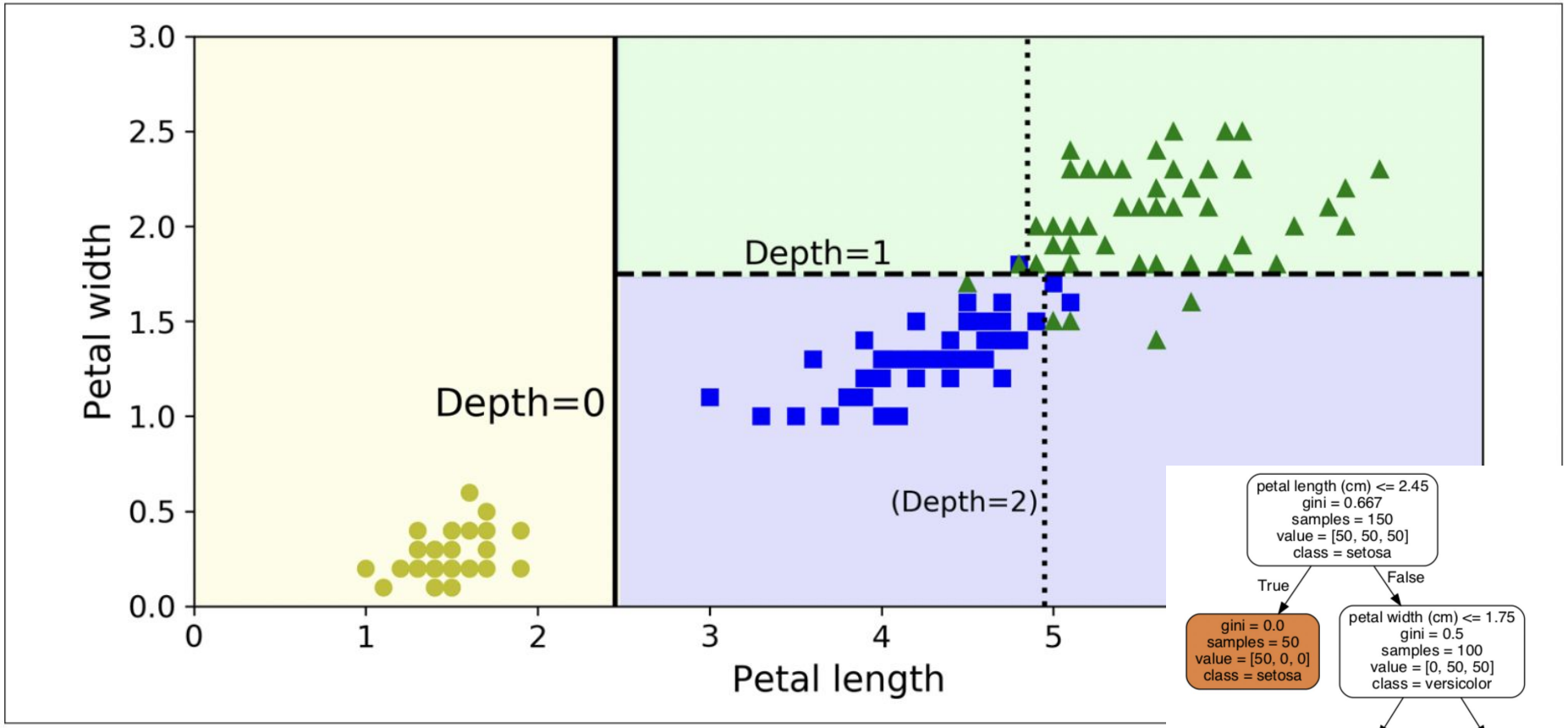


Árbol de decisión para la clasificación

- Samples: cuenta a cuántas instancias de entrenamiento se aplica el nodo. Por ejemplo: el nodo nivel 1 derecho se aplica a 100 instancias de entrenamiento que tienen una longitud de pétalo superior a 2,45 cm.
- Value: indica a cuántas instancias de entrenamiento de cada clase se aplica este nodo. Por ejemplo: el nodo inferior derecho se aplica a 0 Iris setosa, 1 Iris versicolor y 45 Iris virginica
- Gini: mide su impureza: un nodo es"puro" (gini=0) si todas las instancias de entrenamiento a las que se aplica pertenecen a la misma clase

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2$$

donde $p_{i,k}$ es la proporción de instancias de la clase k entre las instancias de entrenamiento en el nodo i-ésimo. Por ejemplo, el nodo izquierdo de profundidad 1 es puro y su valor de gini es 0 ya que sólo se aplica a las instancias de entrenamiento de Iris setosa.



Fronteras de decisión del árbol de decisión

Classification and Regression Tree (CART)

1. Si se cumplen los criterios de parada, crear un nodo hoja.
2. En caso contrario, encontrar la mejor característica y valor para dividir los datos.
3. Dividir los datos según la característica y valor elegidos.
4. Aplicar recursivamente los pasos 1-3 a los 2 subconjuntos resultantes.
5. Devolver nodos de decisión para cada división.

Funciones:

- Criterios de Parada: Comprobar si se cumplen condiciones como la profundidad máxima o el número mínimo de muestras.
- Encontrar la Mejor División: Evaluar las divisiones basadas en la reducción de la impureza.

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

- Crear Nodo Hoja: Asignar la etiqueta de clase mayoritaria o el valor medio.
- Crear Nodo de Decisión: Almacenar información sobre la característica y valor de división.

Classification and Regression Tree (CART)

- Algoritmo recursivo
- Dividir el conjunto de entrenamiento en dos subconjuntos usando una característica k y un umbral t_k
- Cómo se elige el par (k,t_k)? Buscando el par que genera los subconjuntos más puros, i.e. minimizando:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

- Una vez que se separó el conjunto en dos, se vuelve a empezar con cada subconjunto hasta llegar al criterio de parada

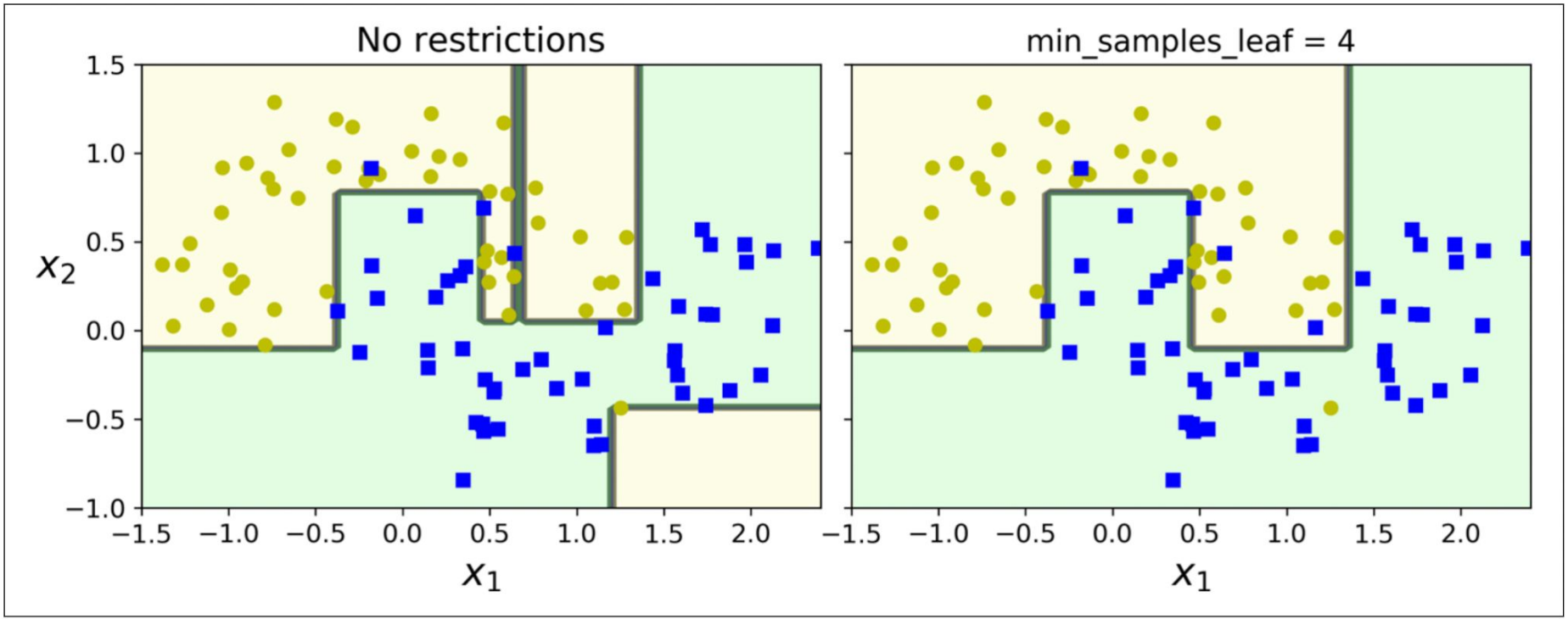
Hiperparámetros de regularización

- Si no se impone ninguna restricción, la estructura del árbol se adaptará a los datos de entrenamiento y se sobre ajustará a ellos
  - restringir fijando hiperparámetros de antemano (CART)
  - no restringir y luego podar arbol

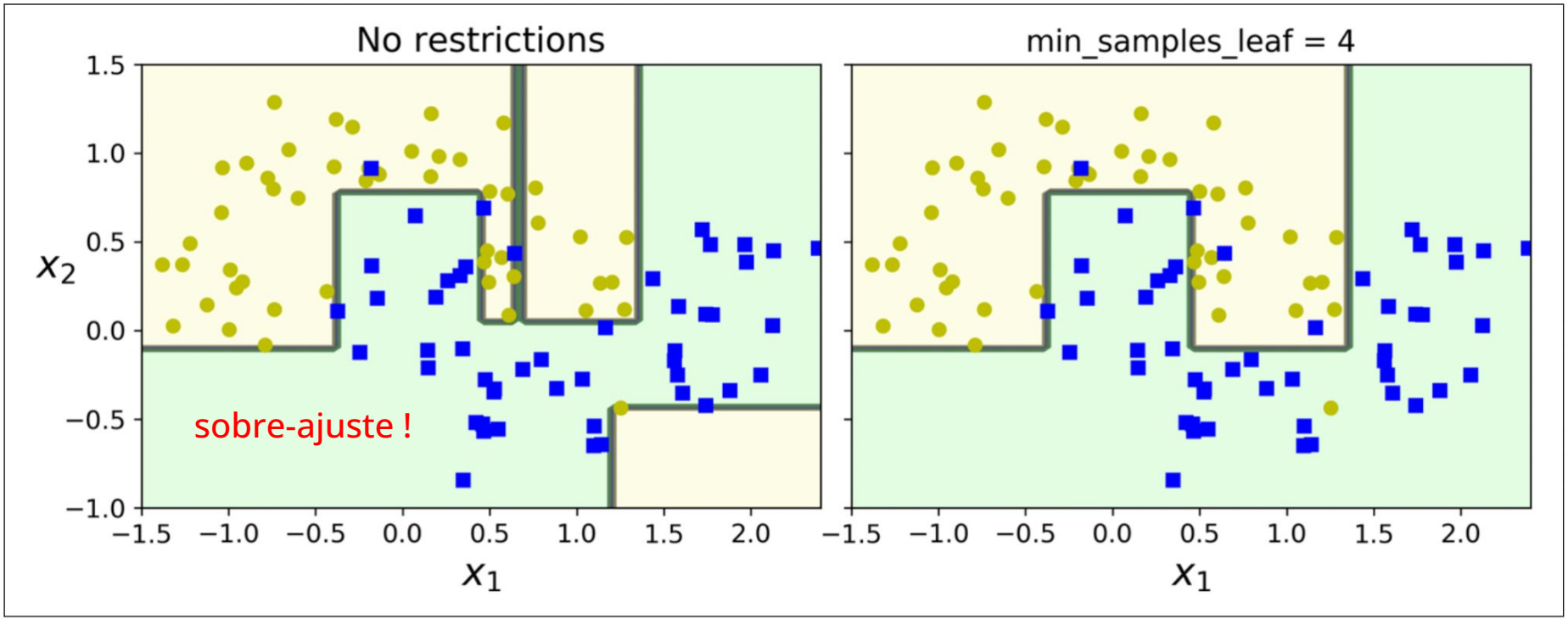
- En general se puede al menos restringir la profundidad máxima del árbol de decisión a través del hiperparámetro `max_depth`

- Reducir `max_depth` regularizará el modelo y reducirá el riesgo de sobreajuste

- La clase `DecisionTreeClassifier` tiene otros parámetros que restringen similarmente la forma del árbol de Decisión, e.g. `min_samples_split`, `min_samples_leaf`, `min_weight_fraction_leaf`, `max_leaf_nodes`, `max_features`, …

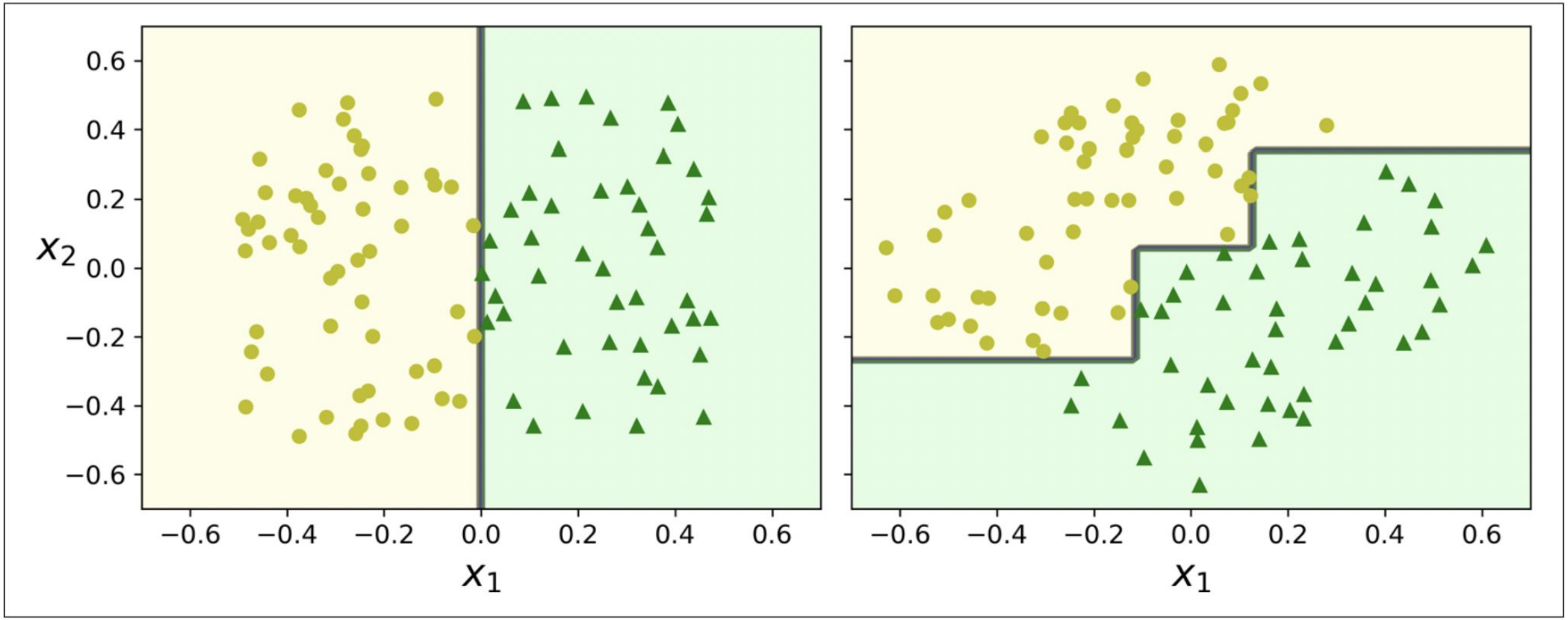


Regularización usando `min_samples_leaf` (el número mínimo de instancias que debe tener un nodo hoja)



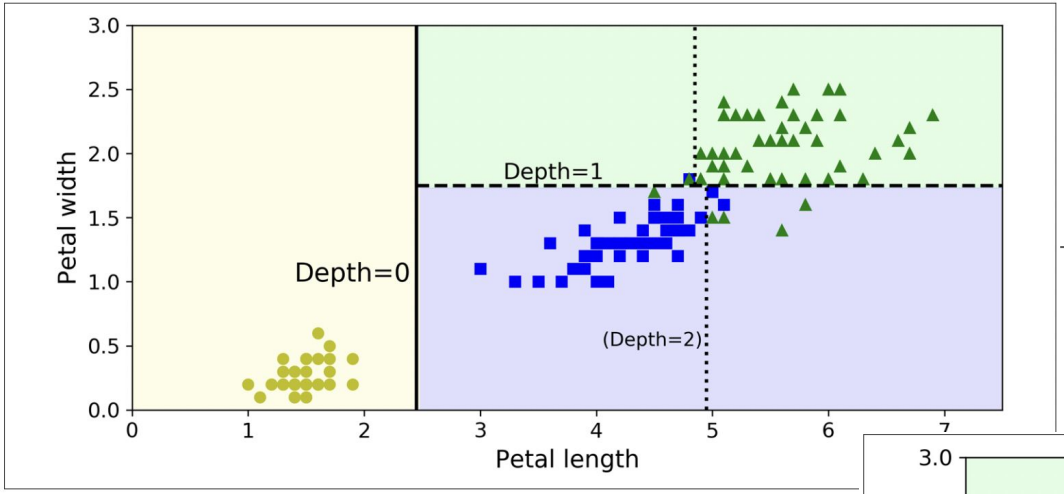
Regularización usando `min_samples_leaf` (el número mínimo de instancias que debe tener un nodo hoja)

Limitations: orthogonal decision boundaries

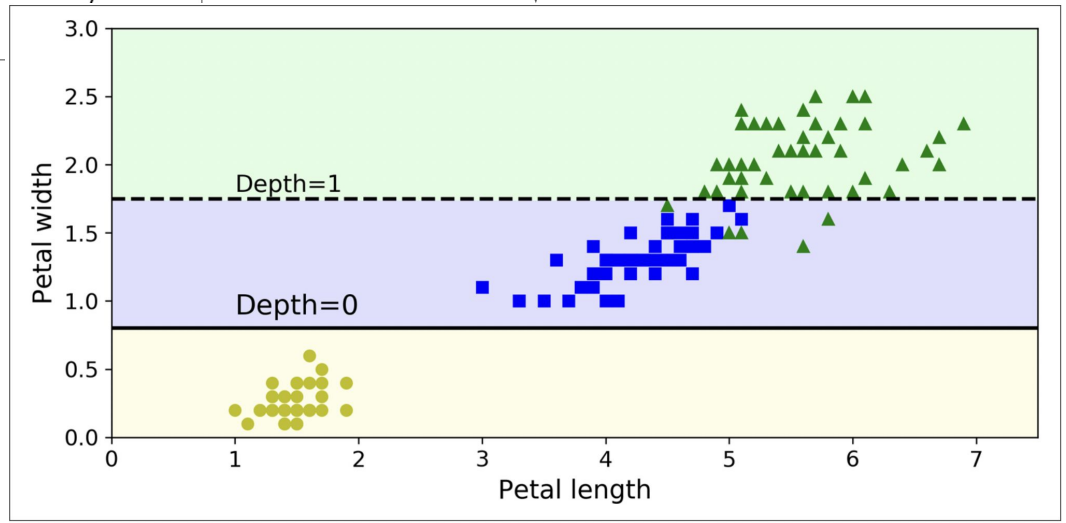


Sensibilidad a la rotación del conjunto de entrenamiento

Limitations: sensitive to small variations



eliminar el iris versicolor de mayor amplitud del conjunto de entrenamiento

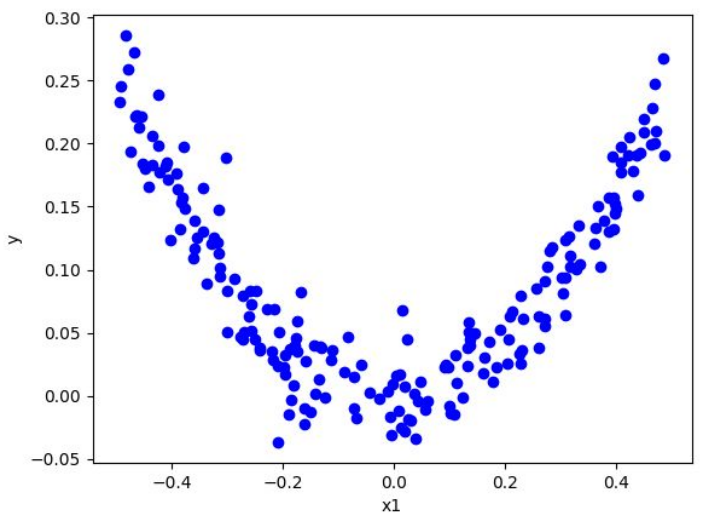


Sensibilidad a los detalles del conjunto de entrenamiento

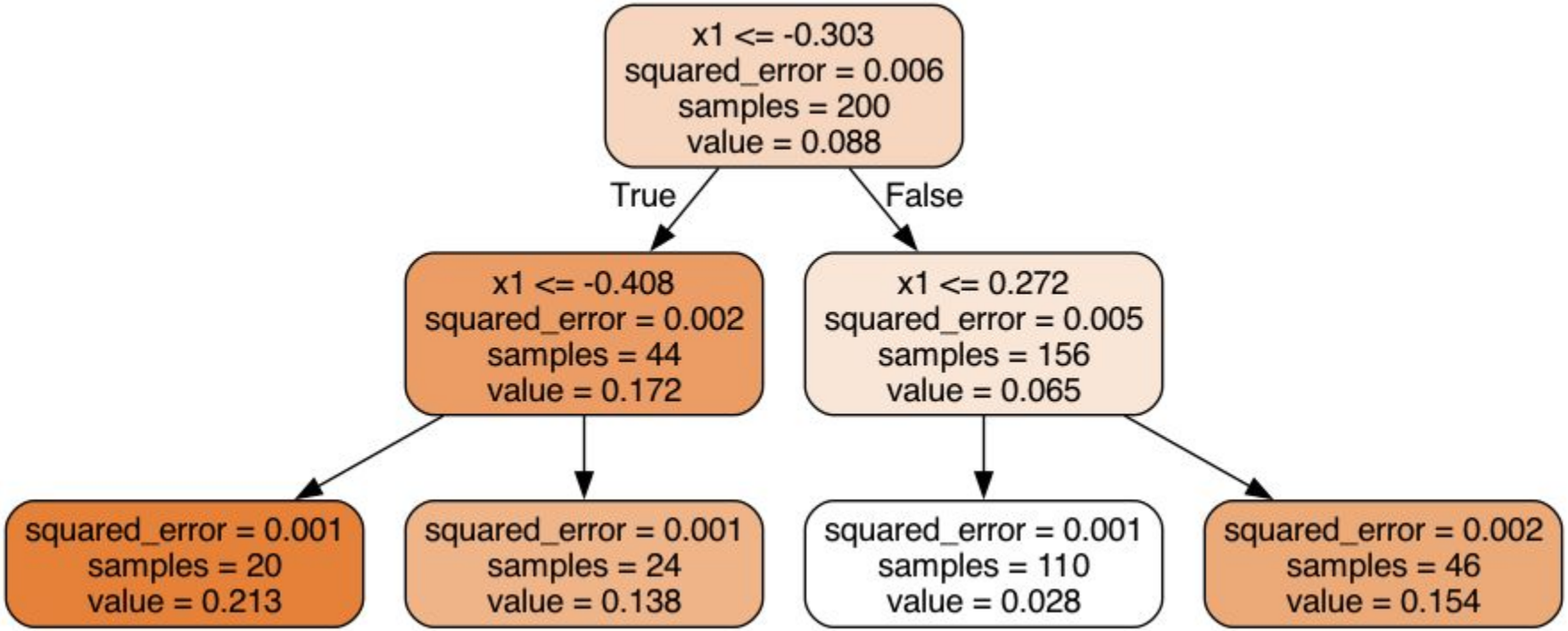
REGRESIÓN con árboles de decisión

- Los árboles de decisión también pueden aplicarse para regresión.
- Ejemplo un árbol de regresión entrenado en un conjunto de datos cuadráticos ruidosos y con `max_depth=2`

```
from sklearn.tree import DecisionTreeRegressor

np.random.seed(42)
X_quad = np.random.rand(200, 1) - 0.5  # a single random input feature
y_quad = X_quad ** 2 + 0.025 * np.random.randn(200, 1)

tree_reg = DecisionTreeRegressor(max_depth=2, random_state=42)
tree_reg.fit(X_quad, y_quad)
```



REGRESIÓN con árboles de decisión



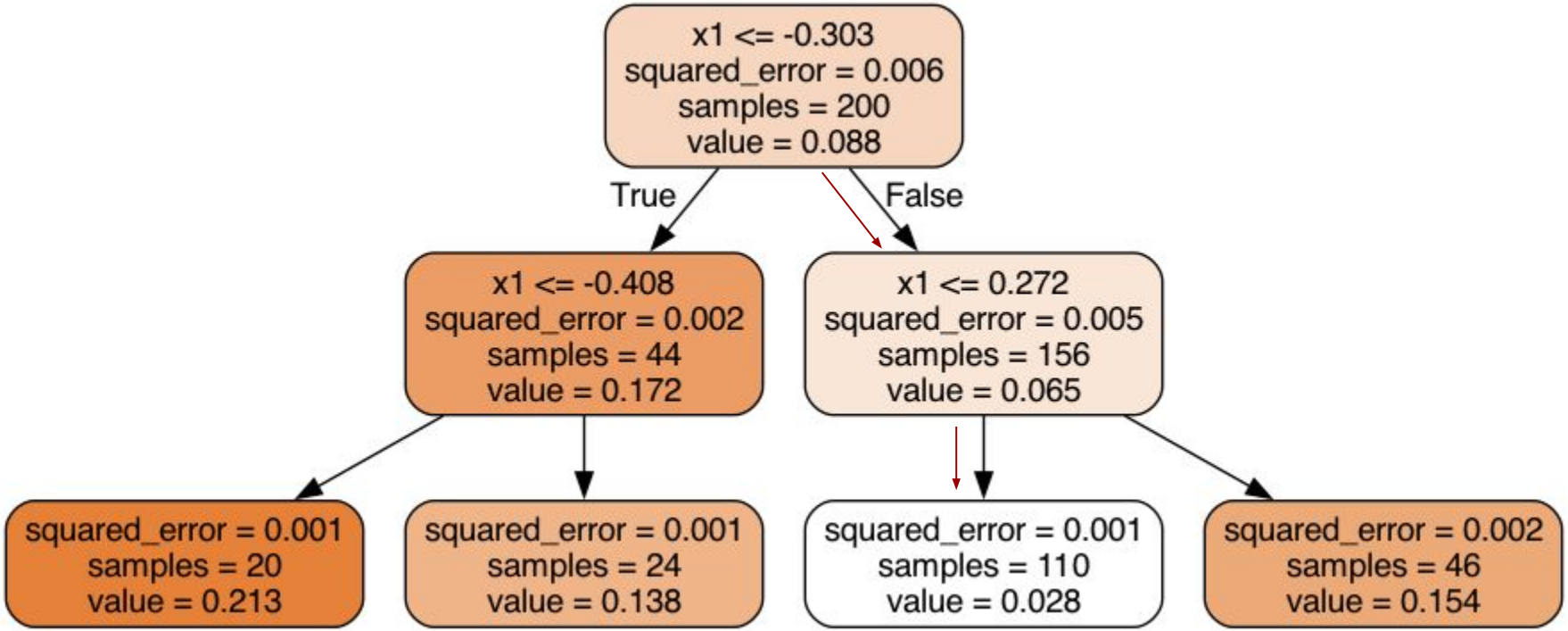
Árbol de decisión para la regresión

- En lugar de predecir una clase en cada nodo, predice un valor

- Supongamos que queremos predecir el valor para un nuevo punto x1 = 0.2

REGRESIÓN con árboles de decisión

x1=0.2, g(x1)=?



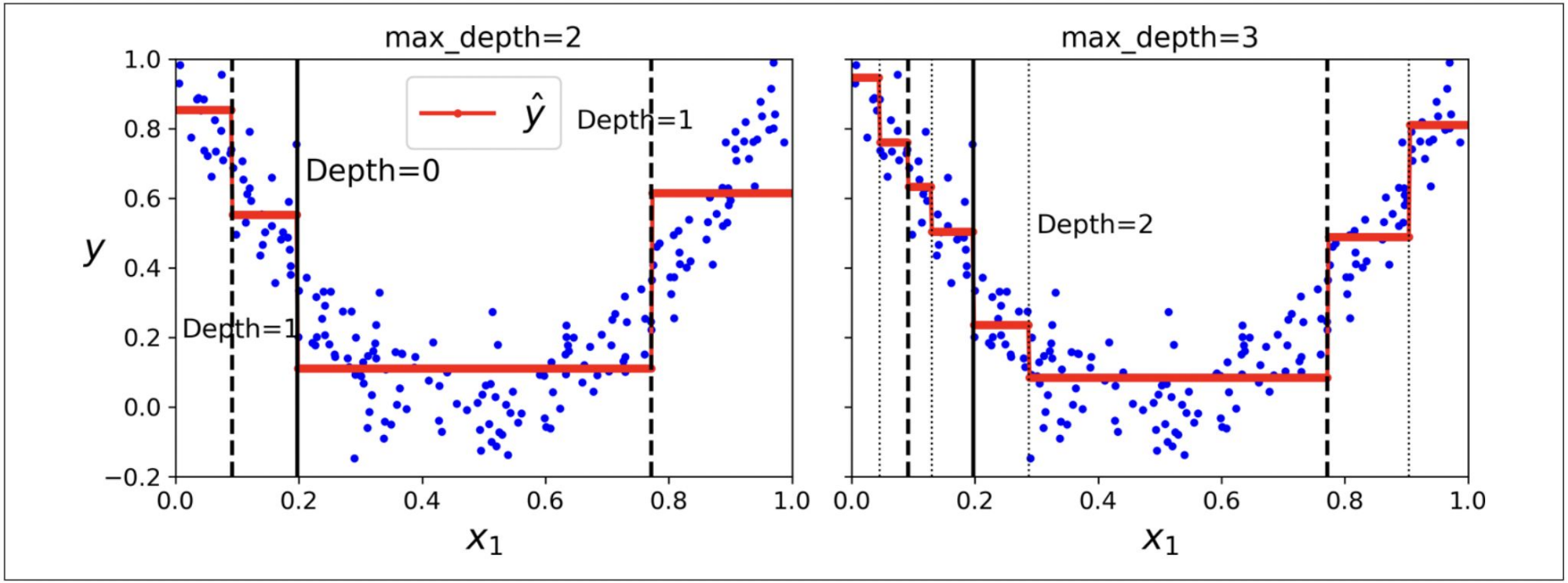
y = g(x1) = 0.028

Árbol de decisión para la regresión

- En lugar de predecir una clase en cada nodo, predice un valor

- Supongamos que queremos predecir el valor para un nuevo punto x1 = 0.2
- Recorriendo el árbol, llegamos al nodo hoja que predice el valor 0.028
- Esta prediccion, es el promedio de las 110 muestras de entrenamiento asociados a este nodo hoja
- Tiene un error cuadrático medio (MSE) de 0.001

Representación de las predicciones



Predicción de dos árboles de regresión.

La predicción en cada región es siempre el valor promedio de las instancias de cada región.

Algoritmo CART para la regresión

- Casi mismo algoritmo
- Diferencia: en vez de dividir el conjunto de entrenamiento de forma que se minimice el valor de gini (impureza) de ambos conjuntos, se intenta dividir el conjunto de entrenamiento de forma que se minimice el MSE

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}} \quad \text{where} \begin{cases} \text{MSE}_{\text{node}} = \sum_{i \in \text{node}} \left(\hat{y}_{\text{node}} - y^{(i)}\right)^2 \\ \hat{y}_{\text{node}} = \frac{1}{m_{\text{node}}} \sum_{i \in \text{node}} y^{(i)} \end{cases}$$

Hiperparámetros de regularización

Al igual que para el caso de clasificación, si no se restringe (regulariza) el modelo, habrá un sobre-ajuste a los datos de entrenamiento