Aprendizaje máquina para clasificación

Lorena Álvarez Pérez

Universidad Carlos III de Madrid

Curso académico 2024/2025

Índice

Problemas de decisión definidos en términos de datos

- **Decisión**
  - Un conjunto de **hipótesis** generan observaciones $\Rightarrow$ cada observación proviene de una de estas hipótesis.
  - El **decisor** averigua qué hipótesis fue la responsable de generar cada observación.
  - Caracterización estadística:
    - Distribución *a priori*: $P(H = h), h = 1, \ldots, L$
    - Verosimilitud: $p_{\mathbf{X}|H}(\mathbf{x}|h), h = 1, \ldots, L$
    - Política de costes: c_{dh} siendo $d, h = 1, \ldots, L$
    - Probabilidad *a posteriori*: $P_{H|\mathbf{X}}(h|\mathbf{x}), h = 1, \ldots, L$
- **Clasificación**
  - **Colección de datos** de N pares (observación, etiqueta) $\left\{(\mathbf{x}_i, y_i)\right\}_{i=1}^{N}$.
  - Las clases (etiquetas) se pueden asimilar a las hipótesis que generan los datos; $y_i \in \{1, \ldots, L\}$ es la clase real de cada observación $\mathbf{x}_i$.
  - Estos N pares de datos no son aleatorios (han sido **observados**).
  - El **clasificador** averigua la clase de cada observación de test (éstas sí son aleatorias).

Métodos analíticos, semi-analíticos y máquina

Existen tres métodos para resolver problemas de decisión. Resolver un problema de decisión significa **encontrar una función discriminante** para clasificar (encontrar la hipótesis verdadera) una muestra de test.

- El problema se define en términos **estadísticos** (distribución *a priori* y verosimilitudes). La **Teoría de Decisión** permite encontrar la función discriminante.
- El problema se define a partir de un conjunto de datos:
  - **Métodos semi-analíticos**: se utilizan los datos para **estimar** las distribuciones *a priori* y verosimilitudes, y posteriormente, se aplica la **Teoría de Decisión** con estas funciones de densidad de probabilidad *aprendidas* para obtener la función discriminante.
  - **Aprendizaje máquina**: se utilizan los datos para diseñar una función discriminante que sea capaz de clasificar correctamente observaciones de test.

Aprendizaje máquina

Hasta ahora, hemos estudiado cómo diseñar decisores usando un modelo matemático del problema que necesitamos resolver: verosimilitudes, probabilidades *a priori*, *a posteriori*, costes, etc. La situación más común en la vida real es que estos problemas estén definidos por conjuntos de observaciones o muestras.

Etapas - Aprendizaje máquina

1. Recibir un conjunto de entrenamiento: observaciones y etiquetas (clase real de cada observación)
2. Escoger una familia de modelos
3. Seleccionar los hiperparámetros: optimización del modelo
4. Ajustar el modelo con el conjunto de entrenamiento
5. Recibir el conjunto de test: observaciones sin etiquetas
6. Predecir, con el modelo, una etiqueta para cada observación del conjunto de test
7. Recibir las etiquetas reales del conjunto de test y **evaluar la precisión del modelo**

Conjunto de entrenamiento y de test

El **conjunto de entrenamiento** está formado por N pares $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. El **conjunto de test** está formado por N_t pares $\{(\mathbf{x}_t, y_t)\}_{t=1}^{N_t}$.
Asumimos que las muestras del conjunto de entrenamiento y test son i.i.d. (misma distribución de probabilidad y mútuamente independientes).

- Las observaciones $\mathbf{x}_i$ son d-dimensionales: $\mathbf{x}_i \in \mathbb{R}^d$.
- Las clases son discretas y finitas. "Asumen" el papel de las hipótesis en decisión analítica. En problemas multiclase, normalmente usamos números enteros para denotar las diferentes clases. En clasificación binaria, utilizamos $y_i \in -1, 1$ (clases negativas y positivas).

Matrices de datos

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,d} \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Cada fila de $\mathbf{X}$ almacena una observación

Generalización y sobreajuste

El objetivo del aprendizaje máquina es aprender un modelo que consiga una **alta tasa de acierto de clasificación en el conjunto de test.**

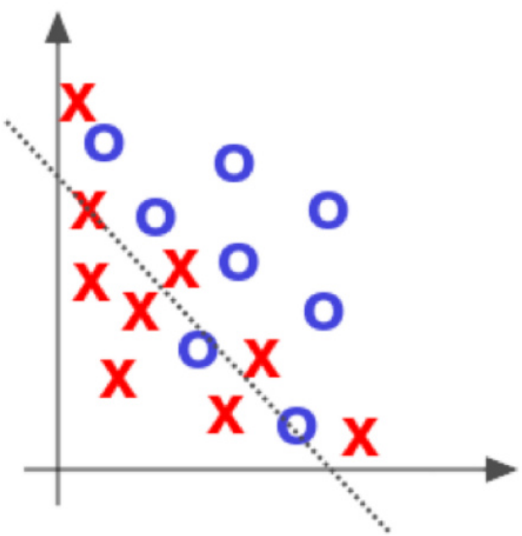
En general, aprender a clasificar correctamente el conjunto de entrenamiento, es una buena estrategia para clasificar correctamente las observaciones del conjunto de test no utilizadas en el diseño del modelo.

Sin embargo, el proceso de optimización que ajusta el modelo puede llevar al **sobreajuste**. Esto sucede cuando la precisión del modelo sobre el conjunto de entrenamiento es mucho mejor que la obtenida con el conjunto de test.

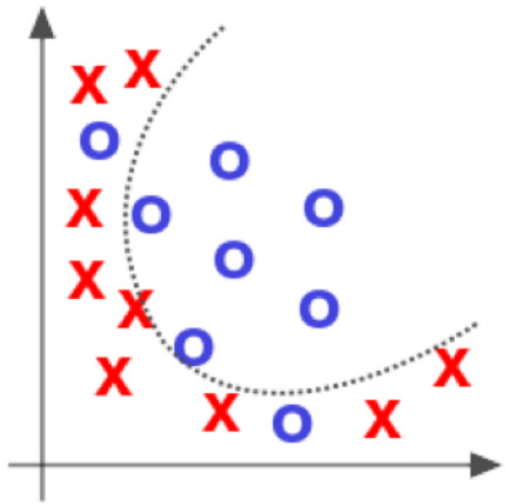
Esta generalización con el conjunto de test está relacionada con el coste (riesgo) medio de un clasificador: la tasa de error media sobre el conjunto de test es el **coste real** de un clasificador diseñado con un método máquina, mientras que la tasa de error media sobre el conjunto de entrenamiento es el **coste empírico**. El coste empírico se puede medir, mientras que el coste real, en la mayoría de los casos, sólo se puede estimar.

A lo largo de este curso, presentaremos varias formas de medir ese coste real (uso de un **conjunto de test separado** o **validación cruzada**).

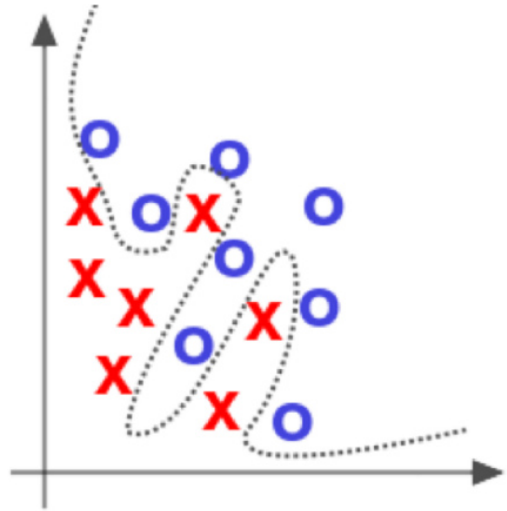
Ejemplos: subajuste y sobreajuste



Subajuste



Apropiado



Sobreajuste

Métodos paramétricos y no-paramétricos

El objetivo del aprendizaje máquina es ajustar los parámetros del modelo. Esto es equivalente a diseñar la función discriminante $g(\mathbf{x})$.

Generalmente, el modelo es una "plantilla" con parámetros libres y el aprendizaje implica una optimización que utiliza las muestras del conjunto de entrenamiento para ajustar estos parámetros a valores que dotan al modelo de una gran capacidad de generalización.

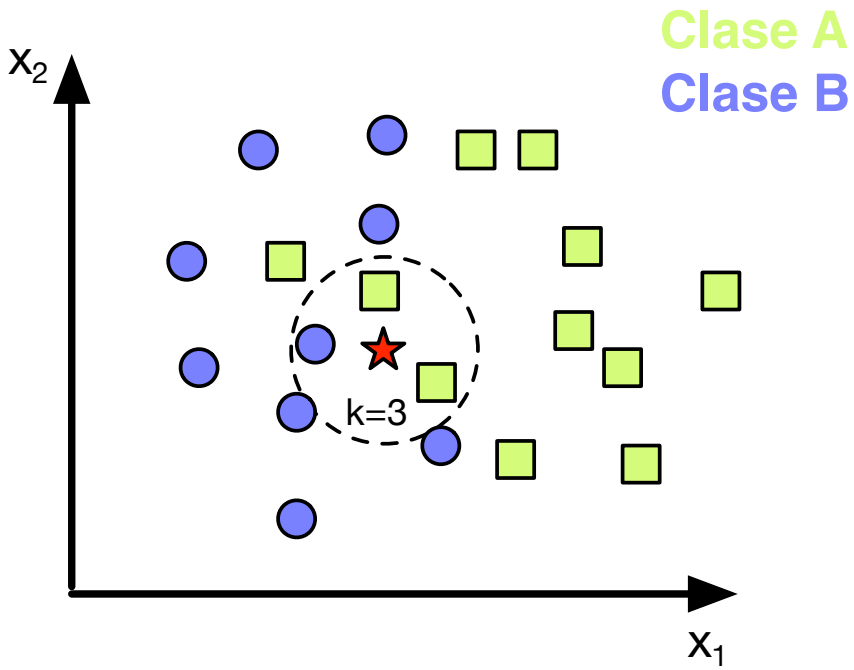
Los modelos pueden ser clasificados en dos grupos según el número de parámetros a ajustar:

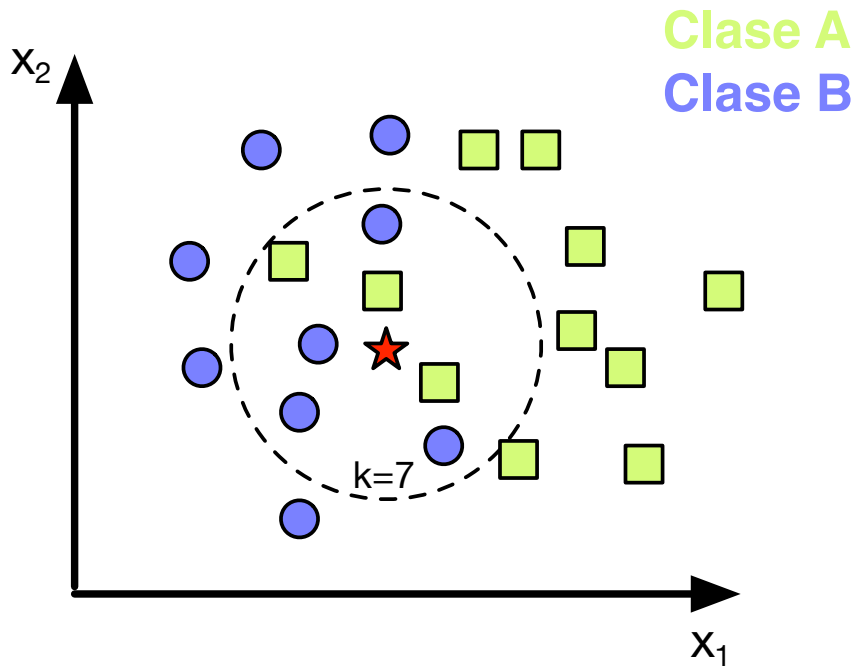
- **Modelos paramétricos**: El número de parámetros se fija antes de obtener el conjunto de datos de entrenamiento. Por ejemplo, en el caso de un clasificador lineal, donde los datos tienen 4 dimensiones, se necesitaría ajustar 5 parámetros (independientemente del tamaño del conjunto de entrenamiento).
- **Modelos no-paramétricos**: El número de parámetros es función del tamaño del conjunto de entrenamiento (algunos ejemplos están al final de este documento).

Introducción a k-NN

- Modelo no-paramétrico. k representa el número de vecinos (hiperparámetro).
- Suposición: si dos observaciones $\mathbf{x}_i$ and $\mathbf{x}_j$ están suficientemente próximas, es razonable pensar que pertenecerán a la misma clase.
- Clasificación de localidad: cada muestra del conjunto de test se clasifica exclusivamente según sus vecinos (del conjunto de entrenamiento).
- El valor de k determina el tamaño del vecindario y localidad del clasificador:
  - $k = 1$: a cada muestra del conjunto de test se le asigna la clase de la muestra más cercana en el conjunto de entrenamiento.
  - $k = N$: a todas las muestras del conjunto de test se le asigna la **clase mayoritaria** (del conjunto de entrenamiento).
  - valores intermedios de k modulan el tamaño del vecindario.
- El valor de k debe establecerse previamente. El rango de valores de k que mejor funciona depende del problema. La densidad de observaciones puede variar a través del espacio de entrada. Para una observación de test en un área densamente poblada, sus vecinos más próximos pueden caber en una esfera de radio más pequeño que en el caso de una observación en un área escasamente poblada.

Ejemplo del algoritmo k-NN para clasificación





Función discriminante en k-NN

La función discriminante para k-NN se construye de forma algorítmica. Para cada observación $\mathbf{x_t}$, se siguen los siguientes pasos:

1. Calcular la **distancia** entre $\mathbf{x}_t$ y todas las observaciones en el conjunto de entrenamiento.
2. Ordenar esas distancias en orden ascendente.
3. Encontrar las observaciones del conjunto de entrenamiento que ocupan las k primeras posiciones en el *ranking* de distancias a $\mathbf{x_t}$. Sea $\mathcal{I}_k = \{i_1, i_2, \ldots, i_k\}$ los índices de dichas observaciones en el conjunto de entrenamiento.
4. Obtener (recuperar) las clases reales de dichas k observaciones: $\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$
5. $\widehat{y}_\mathrm{t} = \mathrm{moda}\{y_{i_1}, y_{i_2}, \ldots, y_{i_k}\}$

El último paso puede modificarse por un esquema de votación, donde la clase de cada vecino $\mathbf{x}_{\mathrm{i_j}}$ es ponderada por un factor que es inversamente proporcional a la distancia entre $\mathbf{x}_{i_j}$ y $\mathbf{x}_\mathrm{t}$ (parámetro `weights` en `kNeighborsClassifier` de `sklearn`).

Modelo k-NN (desde un punto de vista estadístico)

Ponderación uniforme de votos

"Aproximación local" de la probabilidad *a posteriori* de cada clase

$$P_{H|\mathbf{x}}(H = h|\mathbf{x}_t) \approx \frac{N_h}{k}$$

Ponderación de los votos según distancia

"Aproximación local" de la probabilidad *a posteriori* de cada clase

$$P_{H|\mathbf{x}}(H = h|\mathbf{x}_t) = \frac{p_{\mathbf{x}|H}(\mathbf{x}|H = h)P_H(h)}{p_{\mathbf{x}}(\mathbf{x})}$$

siendo

$$p_{\mathbf{x}|H}(\mathbf{x}|H = h)P_H(h) \approx \sum_{\mathbf{x}_i \in n_k(\mathbf{x}_t), y_i \in h} \frac{1}{d(\mathbf{x}, \mathbf{x}_i)}$$

donde $d(\mathbf{x}, \mathbf{x}_i)$ es la distancia entre la observación de test y cada una de las observaciones del conjunto de entrenamiento en la vecindad de $\mathbf{x}$.

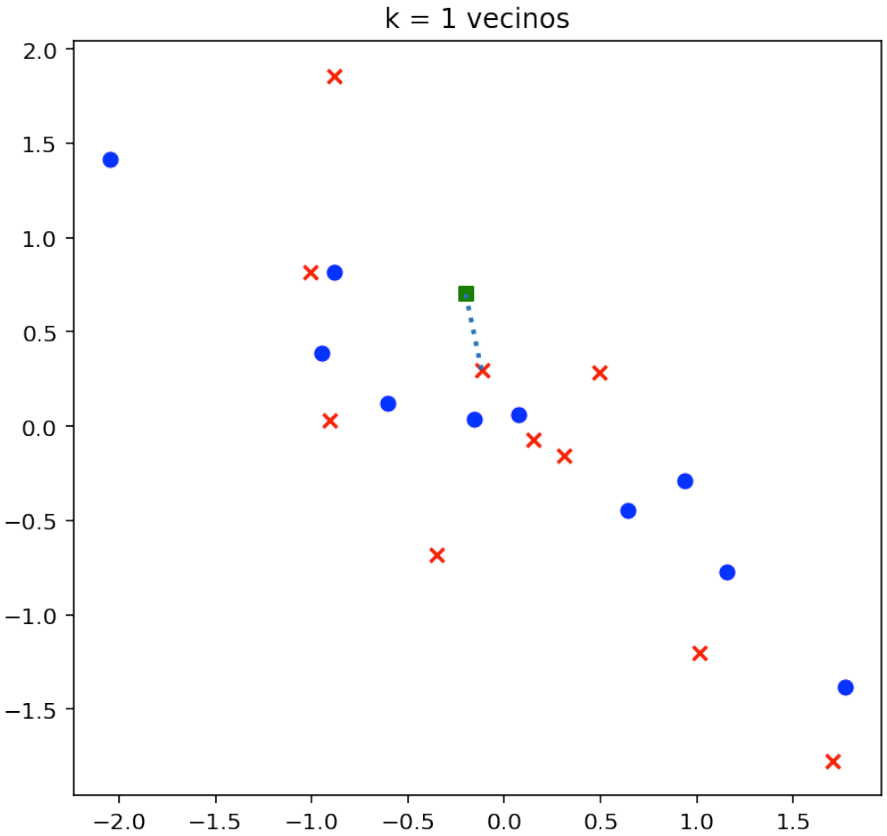
Aprendiendo el modelo k-NN

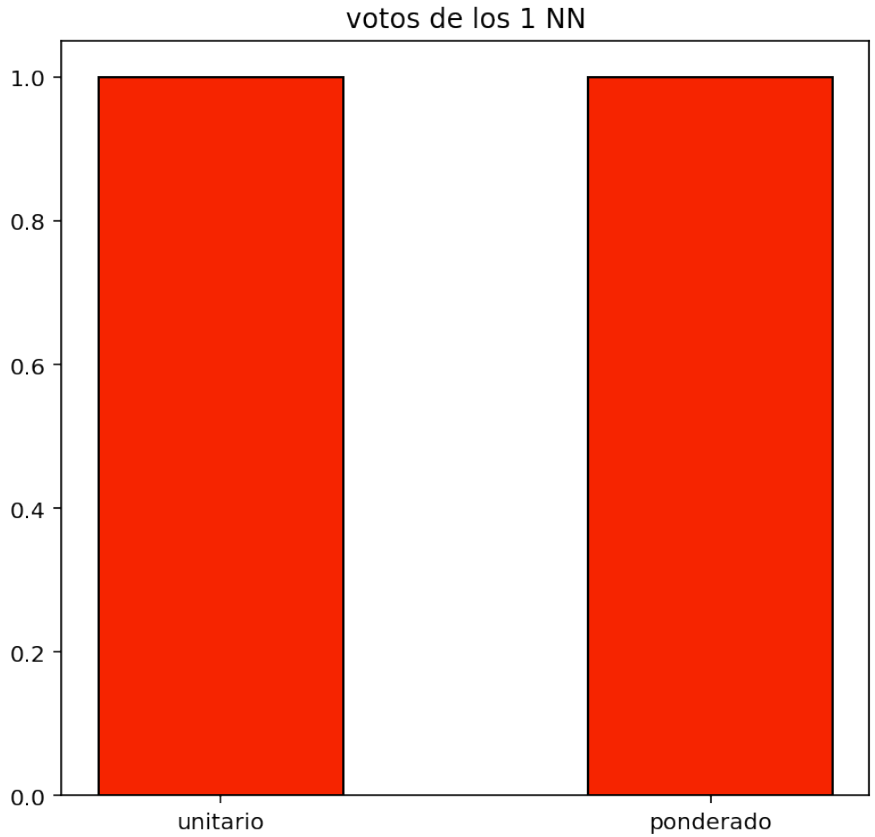
- No existe un proceso de aprendizaje como en otros modelos máquina. k-NN sólo necesita configurar una **tabla indexada que almacena las muestras de entrada** y sus **clases reales**.
- Elegir una **distancia** o **métrica** que permita comparar las observaciones, normalmente la **distancia Euclídea** (o norma L_2):

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{\sum_{j=1}^{d} (x_{1,j} - x_{2,j})^2} = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^\top (\mathbf{x}_1 - \mathbf{x}_2)}$$

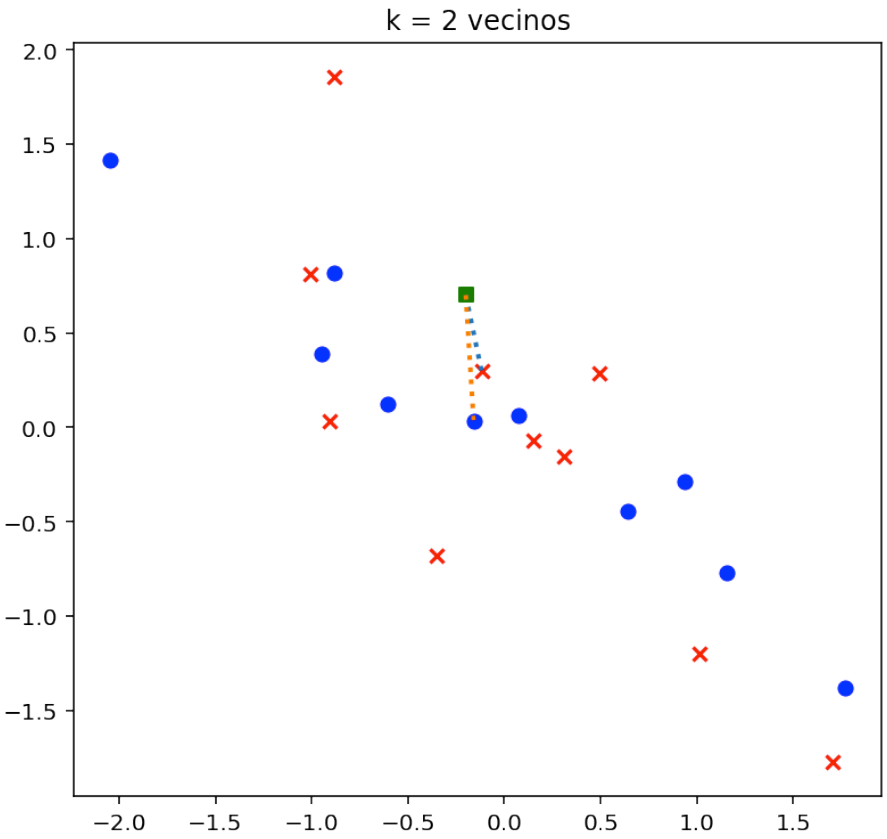
  en Python `d_x1_x2 = numpy.linalg.norm(x1-x2)`
- Fijar k
- Seleccionar esquema de votación para determinar la clase mayoritaria de los k vecinos:
  - Cada vecino aporta un voto unitario a su clase correspondiente (equivale a calcular la moda de las clases)
  - La clase de cada vecino es ponderada por un factor inversamente proporcional a su distancia a $\mathbf{x}_t$

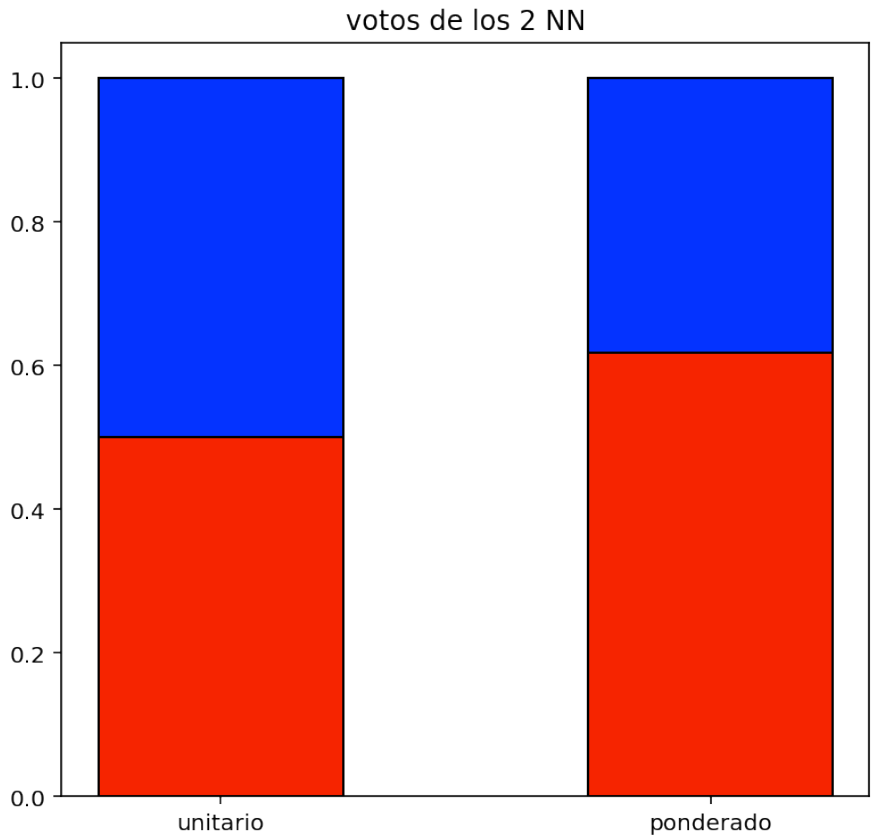
Ejemplo


k = 1 vecinos
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

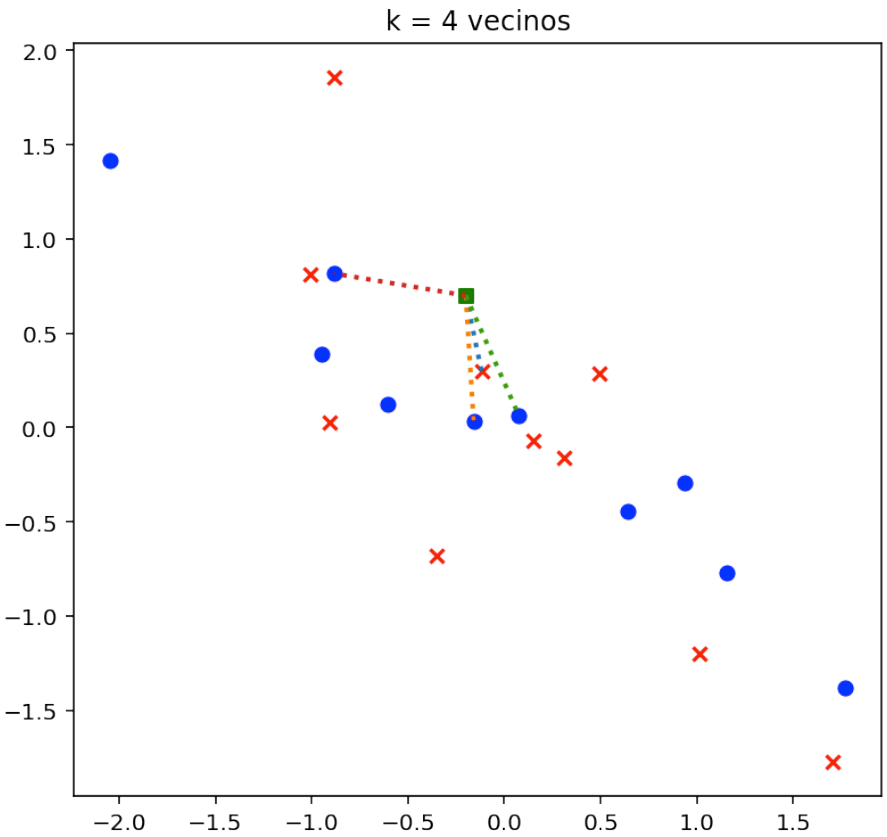

votos de los 1 NN
1.0
0.8
0.6
0.4
0.2
0.0
unitario
ponderado

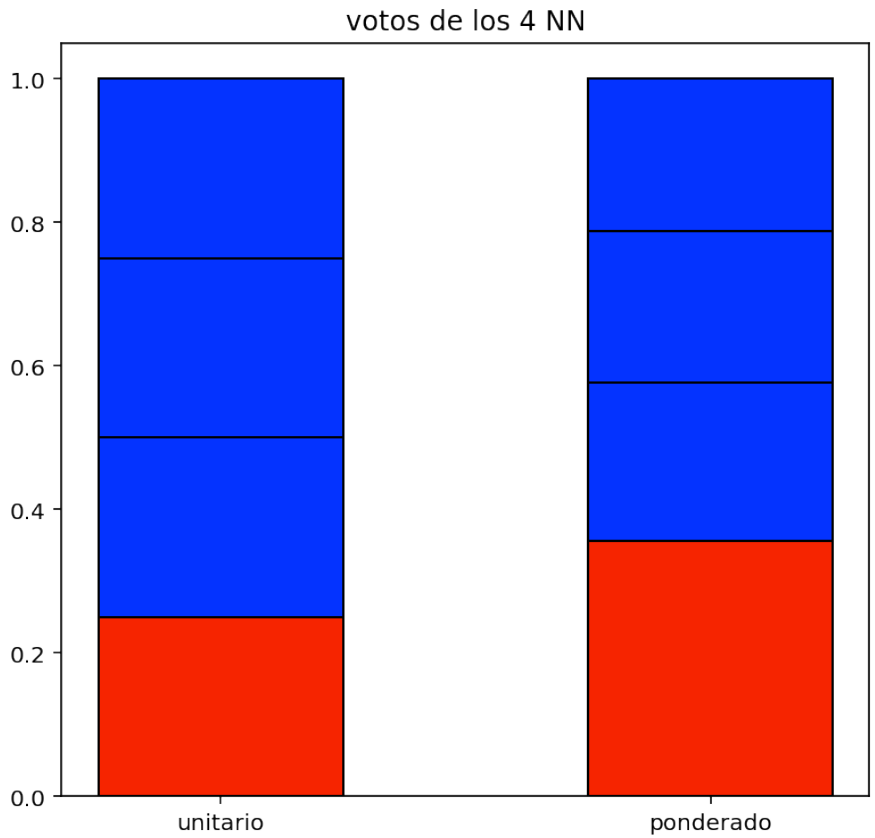
Ejemplo


k = 2 vecinos
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

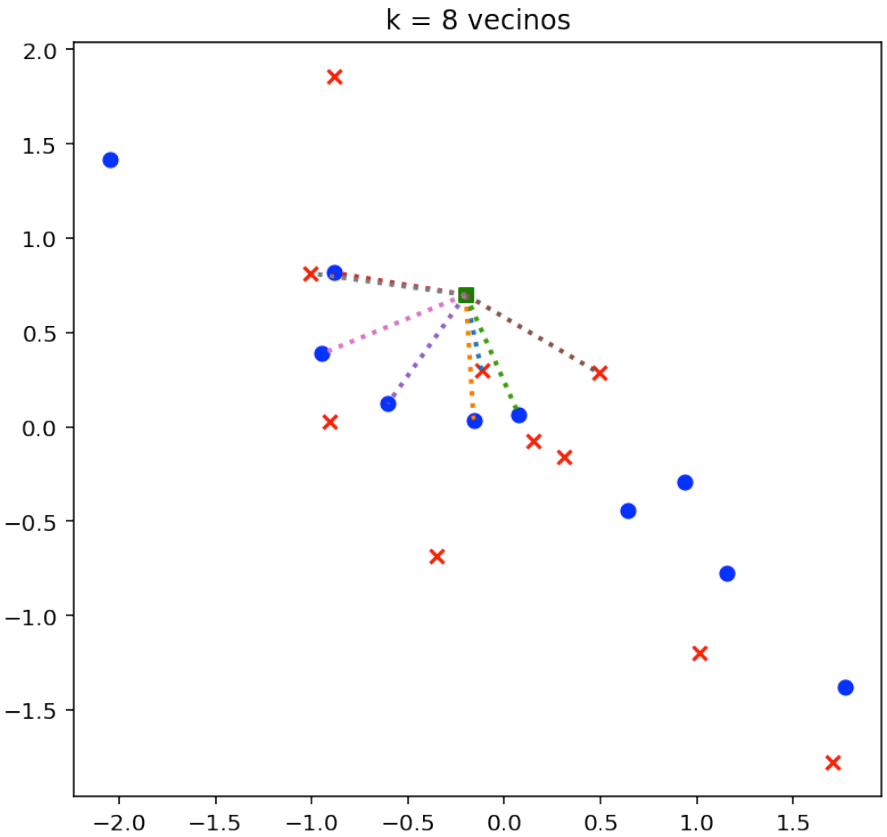

votos de los 2 NN
1.0
0.8
0.6
0.4
0.2
0.0
unitario
ponderado

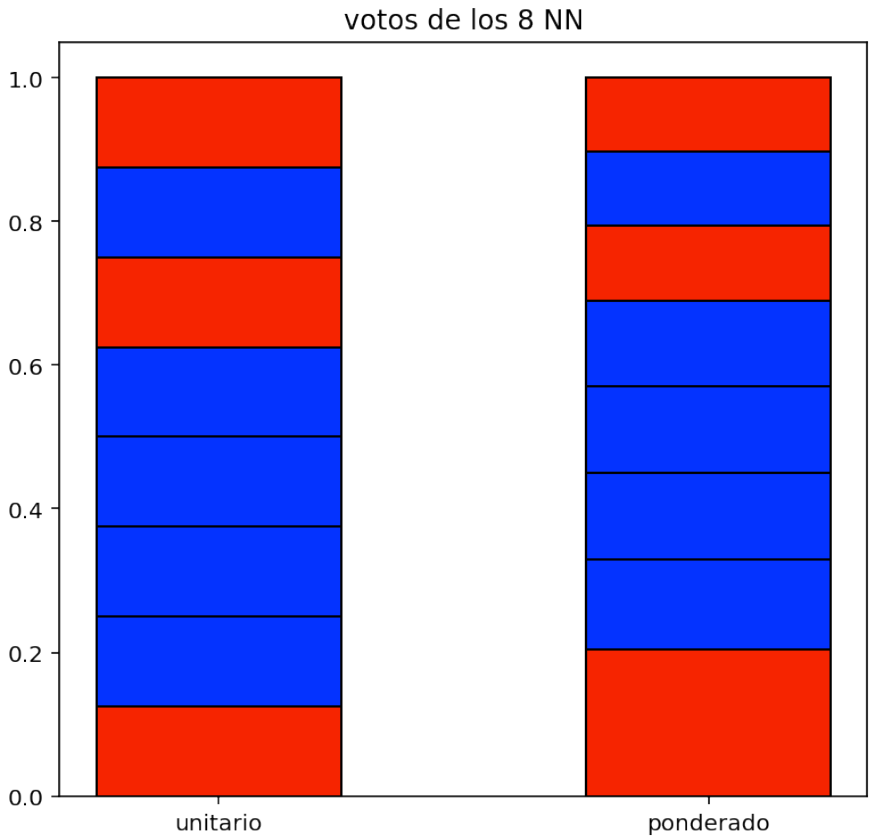
Ejemplo


k = 4 vecinos
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

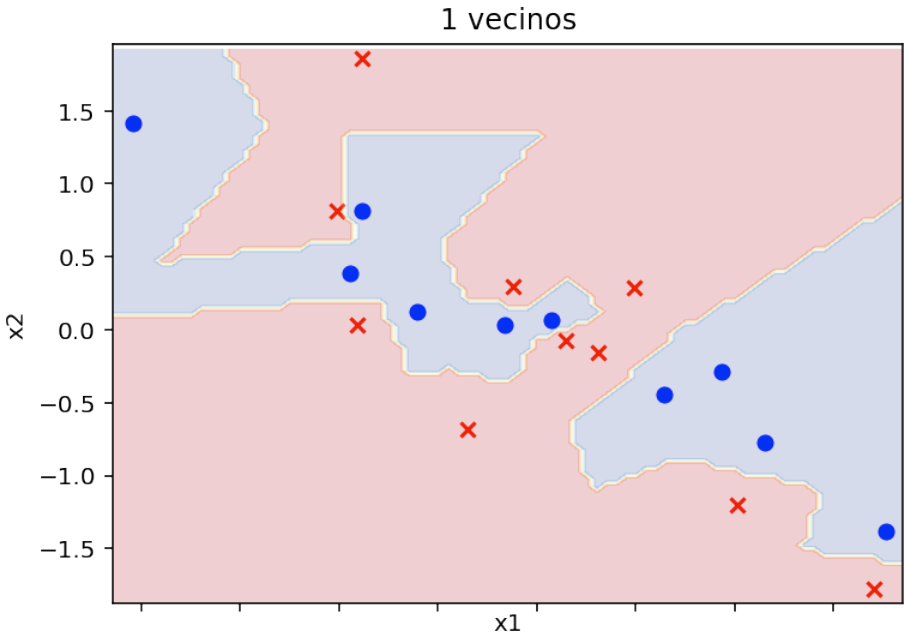

votos de los 4 NN
1.0
0.8
0.6
0.4
0.2
0.0
unitario
ponderado

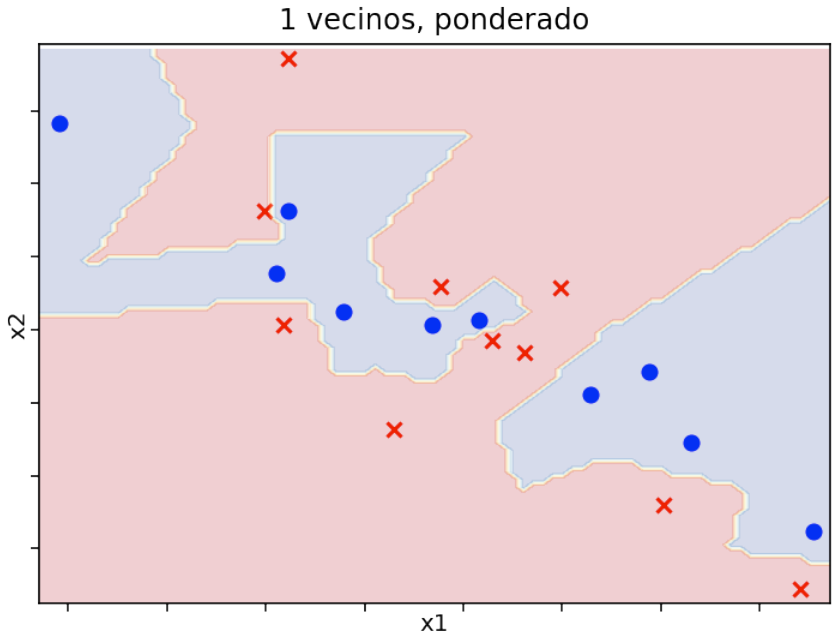
Ejemplo


k = 8 vecinos
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

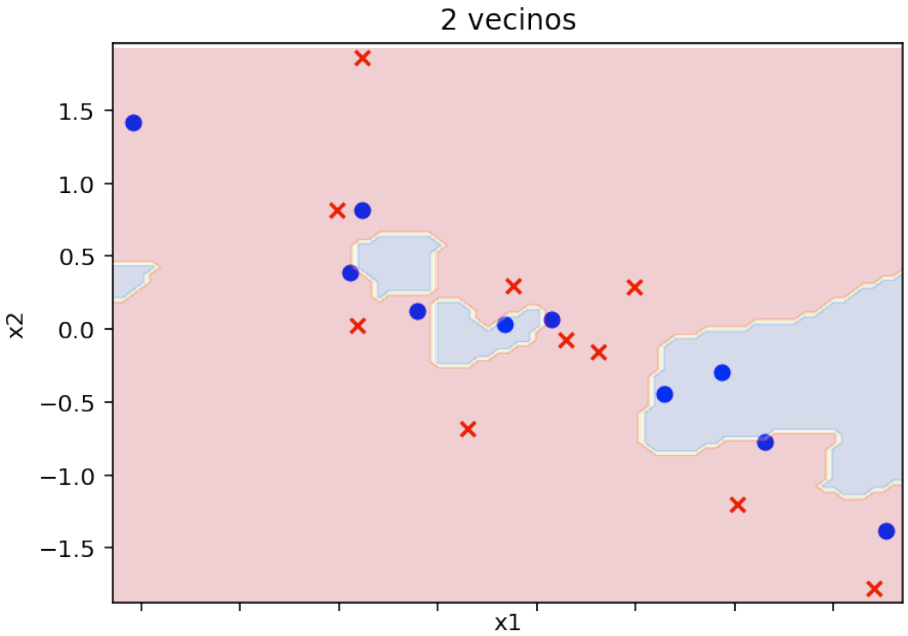

votos de los 8 NN
1.0
0.8
0.6
0.4
0.2
0.0
unitario
ponderado

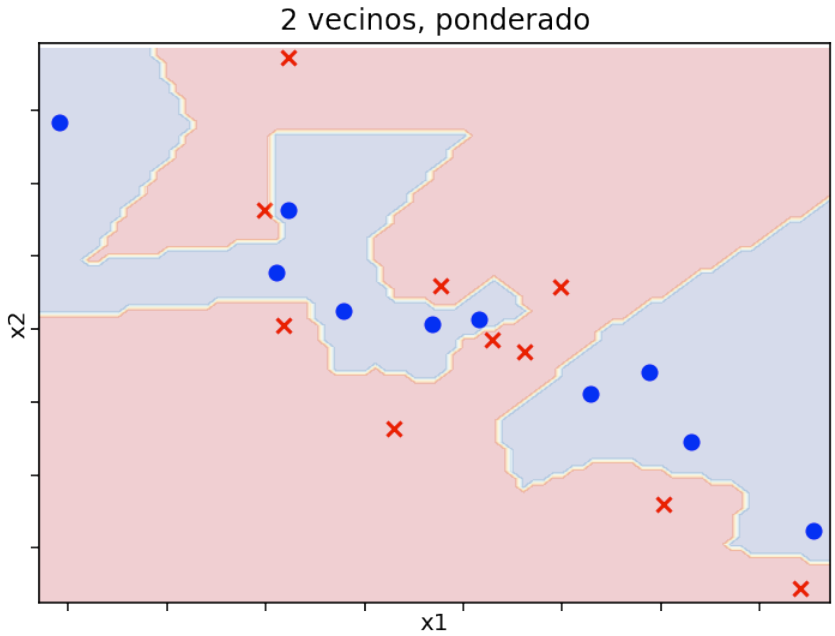
Regiones de clasificación



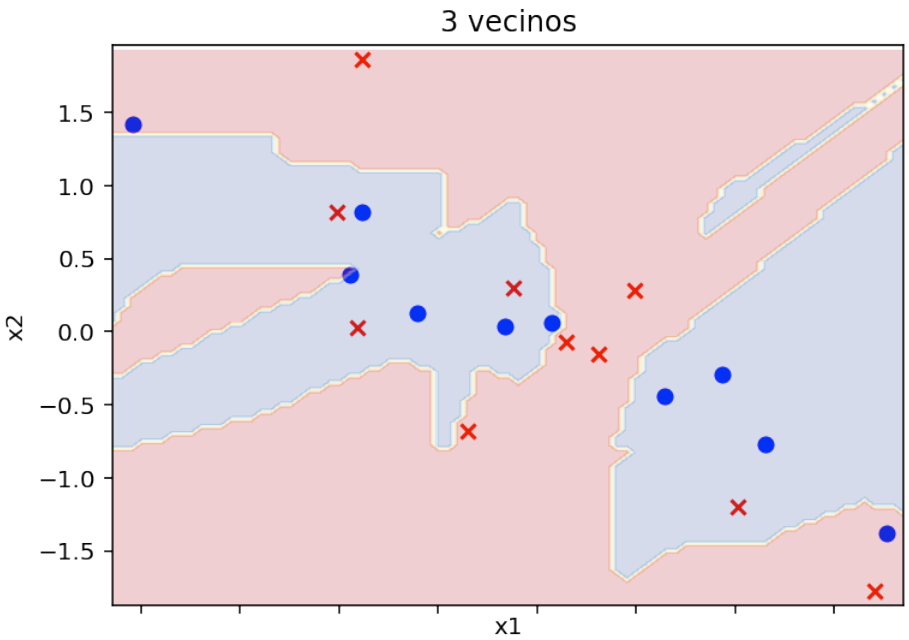


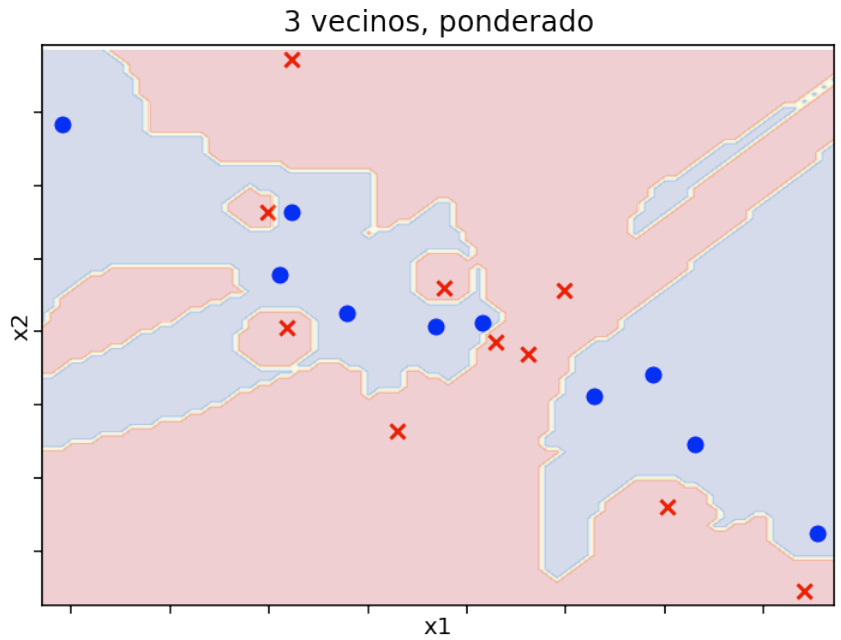
Regiones de clasificación



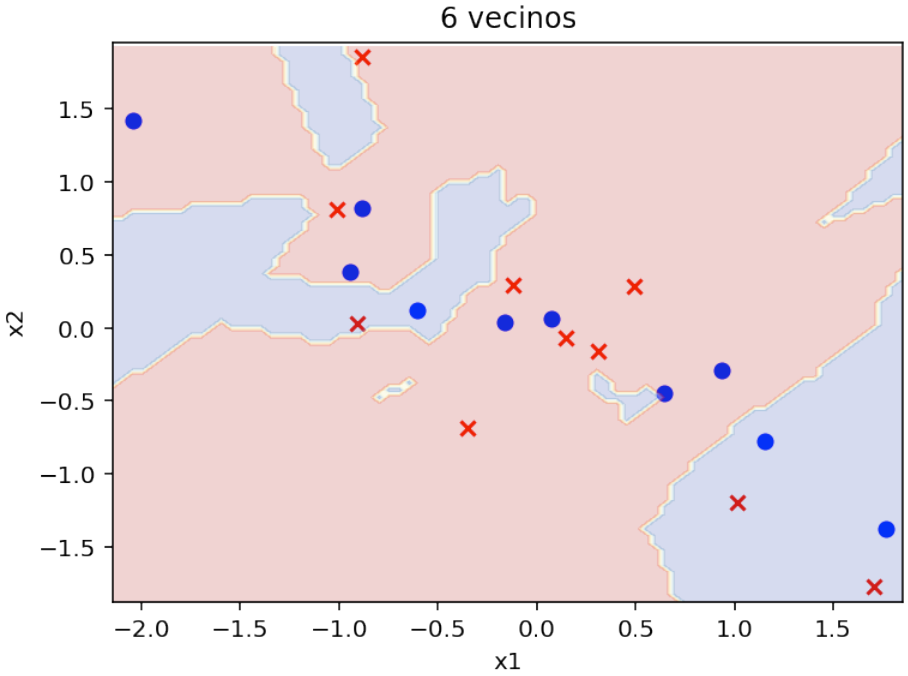


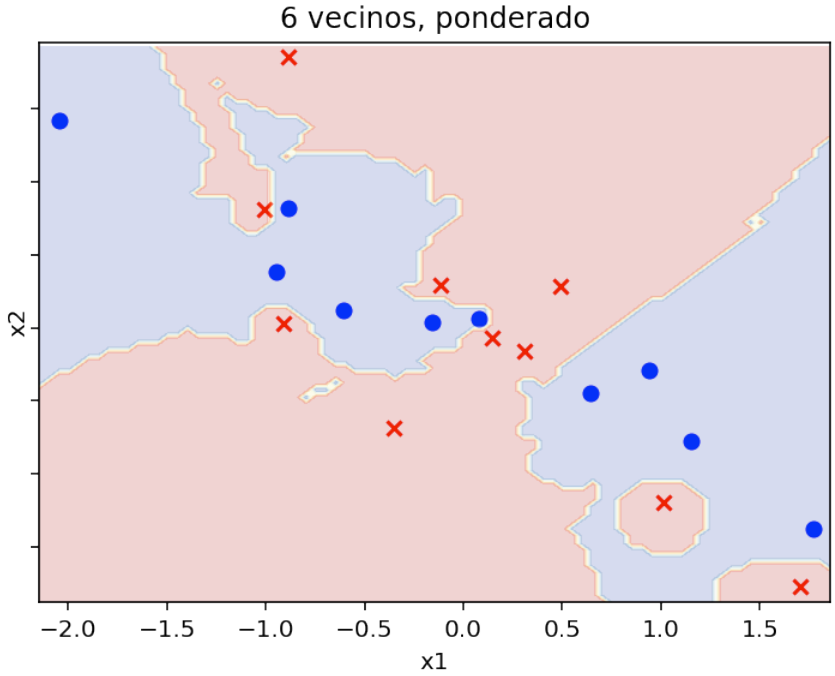
Regiones de clasificación





Regiones de clasificación





Clasificador stump en 1D

Un **árbol de decisión** puede considerarse una estructura jerárquica diseñada a partir de una aplicación recursiva de clasificadores stump.

Clasificador stump

Modelo máquina con un umbral de comparación binario para observaciones 1D

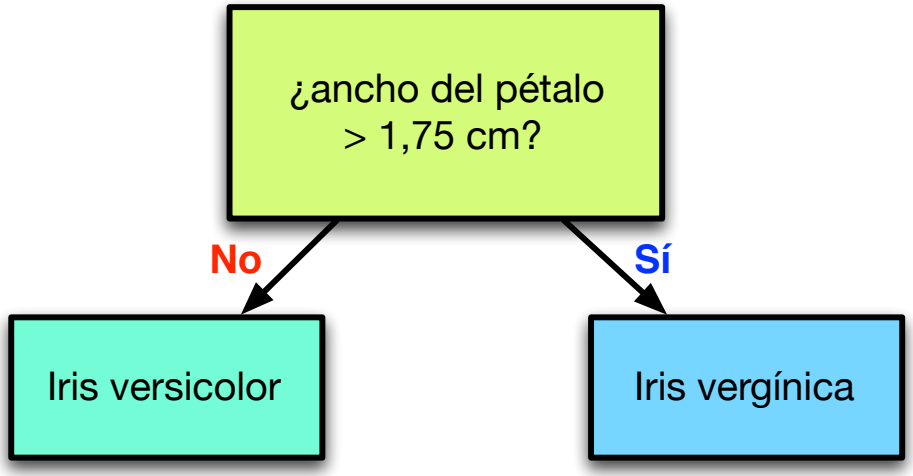
- Clasificador binario: dos clases de salida
- Se define a partir de un umbral η

$$f_s(x) = x \underset{D_0}{\overset{D_1}{\gtrless}} \eta \rightarrow \text{ equivalente a } f_s(x) = \begin{cases} \text{Cierto} & \text{si } x > \eta \\ \text{Falso} & \text{si } x \leq \eta \end{cases}$$

- Divide el espacio de entrada en dos: aquellas observaciones para las que el umbral de comparación es Cierto, y aquellas para las que el umbral es Falso.
- Para finalizar el clasificador, es necesario asignar una de las clases a cada salida del umbral de comparación (esto se hace para maximizar la tasa de acierto).

Ejemplo de un clasificador stump en 1D

Clasificador stump que discrimina entre dos de las tres clases de la base de datos de flores Iris: Iris versicolor e Iris virgínica. El ancho del pétalo se mide en centímetros. Este clasificador stump consigue una tasa de acierto del 94 % para estas dos clases.



Entrenamiento de un clasificador stump en 1D

Ajusta el valor del umbral y la clase de salida de cada rama de la siguiente forma:

η, clase_cierta, clase_falsa = stump1d($\mathbf{X}$, $\mathbf{y}$)

1. Ordena todas las observaciones de entrenamiento según el valor de observación x.
2. En un conjunto con N observaciones, hay un máximo de N posibles valores para η:
   - $(x_i + x_{i+1})/2$, $i = 1, \ldots, N-1$ y
   - bien $\eta = -\infty$ o $\eta = \infty$, es decir, todas las observaciones pertenecen a la misma clase y, por tanto, no existe un punto de división en el grupo
3. Bucle para todos los N posibles valores de η
   1. Asigna una clase en cada rama del clasificador stump
      - clase_cierta_n = moda de la clases con $x > \eta_n$
      - clase_falsa_n = moda de las clases con $x \leq \eta_n$
   2. Evalúa la calidad del clasificador (con la tasa de error u otra métrica)
4. Retorna el valor de η y la asignación de la clase de cada rama con la que mejor valor se obtiene para la métrica seleccionada.

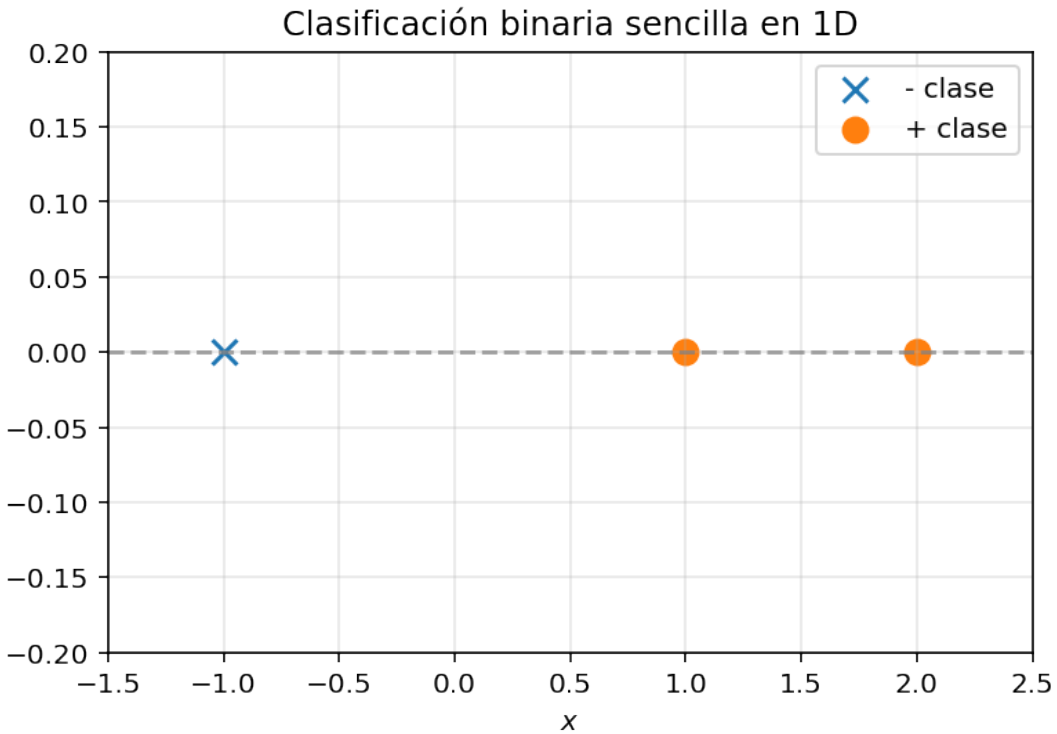
Evaluación de un clasificador stump en 1D

$\widehat{y}$ = evaluate_stump1d(x, η, clase_cierta, clase_falsa)

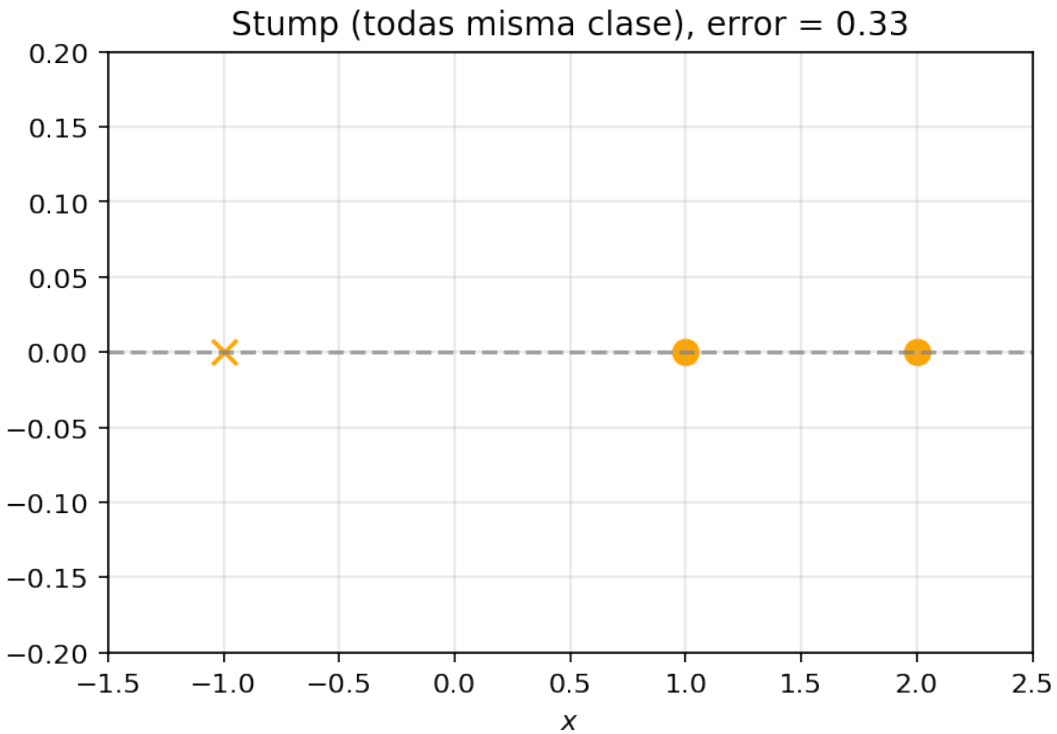
$$f_s(x) = \begin{cases} \text{clase\_cierta} & \text{si } x > \eta \\ \text{clase\_falsa} & \text{si } x \leq \eta \end{cases}$$

Devuelve $\widehat{y} = f_s(x)$

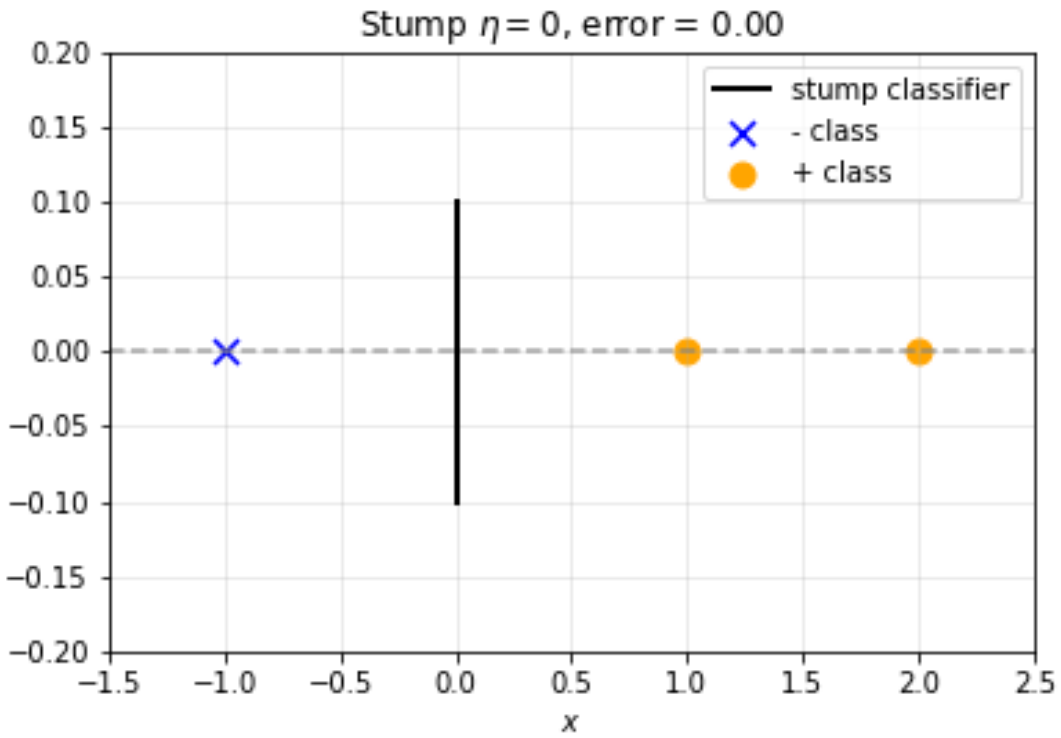
Ejemplo


Clasificación binaria sencilla en 1D
- clase
+ clase
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

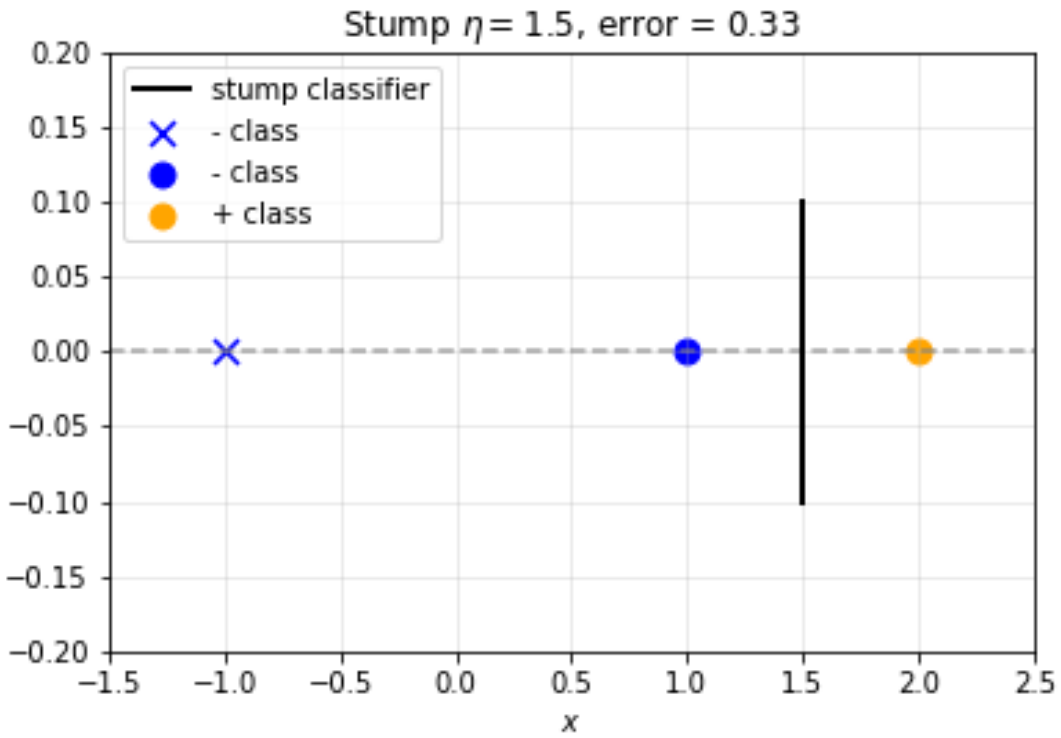
Ejemplo


Stump (todas misma clase), error = 0.33
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

Ejemplo

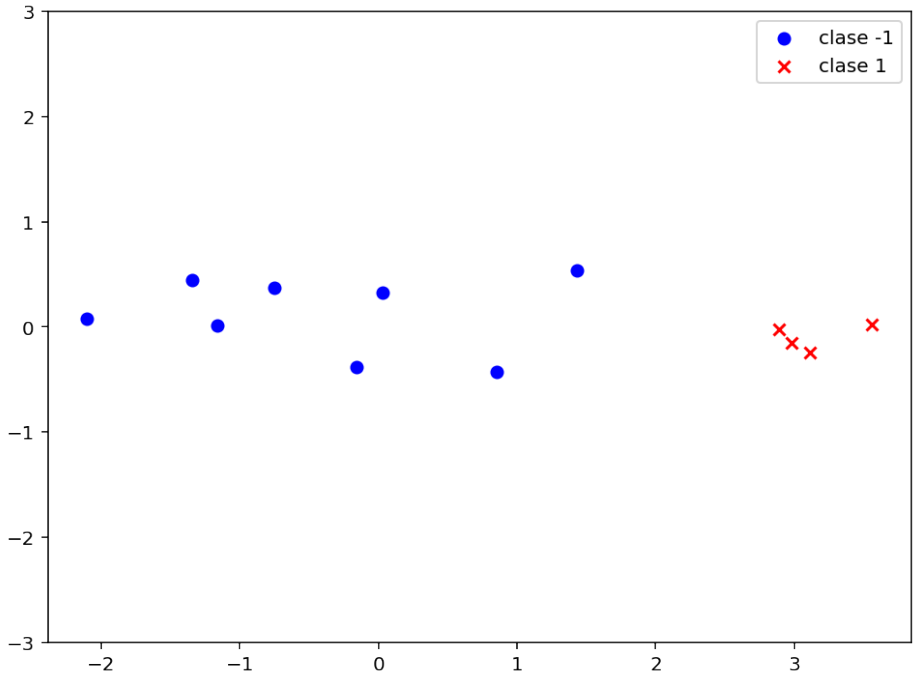

Stump $\eta = 0$, error = 0.00
stump classifier
- class
+ class
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

Ejemplo


Stump $\eta = 1.5$, error = 0.33
stump classifier
- class
- class
+ class
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
x

Clasificadores stump en problemas multidimensionales

El clasificador stump puede extenderse a observaciones con más de 1 dimensión, $\mathbf{x} \in \mathbb{R}^d$ con $d > 1$.



Entrenamiento de clasificadores stump en problemas multidimensionales

Se ejecuta el algoritmo `stump1d` dentro de un bucle anidado que itera sobre las d dimensiones.
Para cada dimensión $j = 1, \ldots, d$, el algoritmo `stump1d` devolverá el mejor stump.
Una vez exploradas las d dimensiones, el algoritmo devolverá el stump correspondiente a la dimensión con la que se obtuvo el mejor valor de la métrica evaluada.

j, η, clase_cierta, clase_falsa = stump($\mathbf{X}$, $\mathbf{y}$)

1. Bucle para todas las dimensiones de las observaciones (número de columnas en $\mathbf{X}$).
   `For` $j = 1, \ldots, d$
   - Z es un array N-dimensional con la j-th columna de $\mathbf{X}$.
   - η_j`, clase_cierta_j, clase_falsa_j = stump1d(`Z`, y)`
   - Almacenar en la variable s_j el valor de la métrica obtenida con (Z,$\mathbf{y}$) en el clasificador stump definido con η_j`, clase_cierta_j, clase_falsa_j`
2. Encontrar la mejor métrica de todas las iteraciones
3. Devolver la dimensión j_* con la que se consigue mejor métrica y el correspondiente clasificador stump definido con η_{j_*}, así como la asignación de clases en cada rama

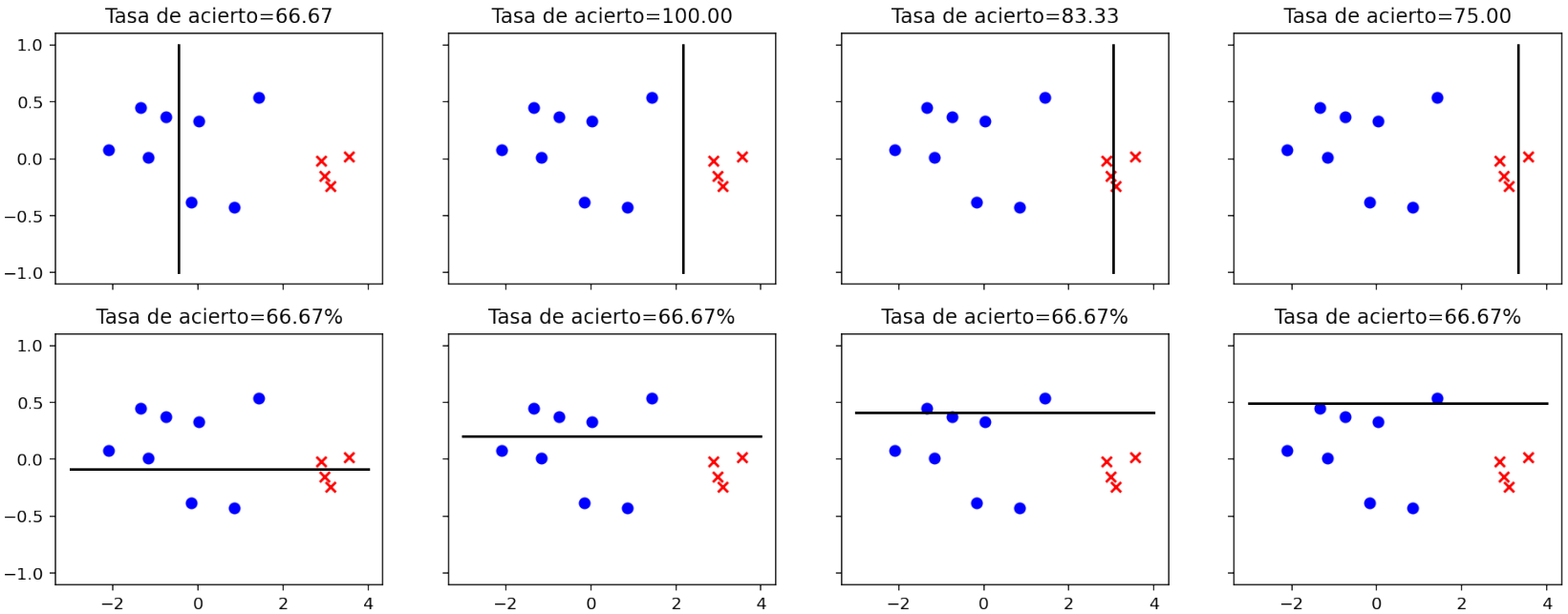
Evaluación de un clasificador stump con datos multidimensionales

$\widehat{y}$ = evaluate_stump($\mathbf{x}$, j, η, clase_cierta, clase_falsa)

$$f_s(\mathbf{x}) = \begin{cases} \text{clase\_cierta} & \text{si } x_j > \eta \\ \text{clase\_falsa} & \text{si } x_j \leq \eta \end{cases}$$

Devolver $\widehat{y} = f_s(\mathbf{x})$

Ejemplo

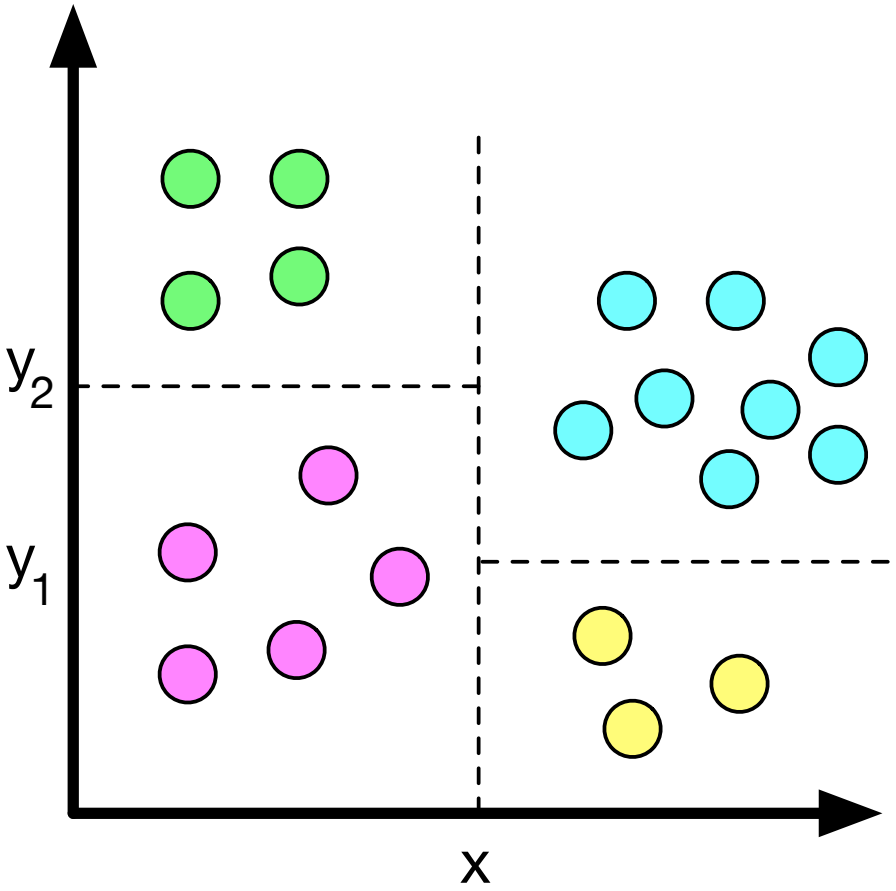

Tasa de acierto=66.67
Tasa de acierto=100.00
Tasa de acierto=83.33
Tasa de acierto=75.00
Tasa de acierto=66.67%
Tasa de acierto=66.67%
Tasa de acierto=66.67%
Tasa de acierto=66.67%

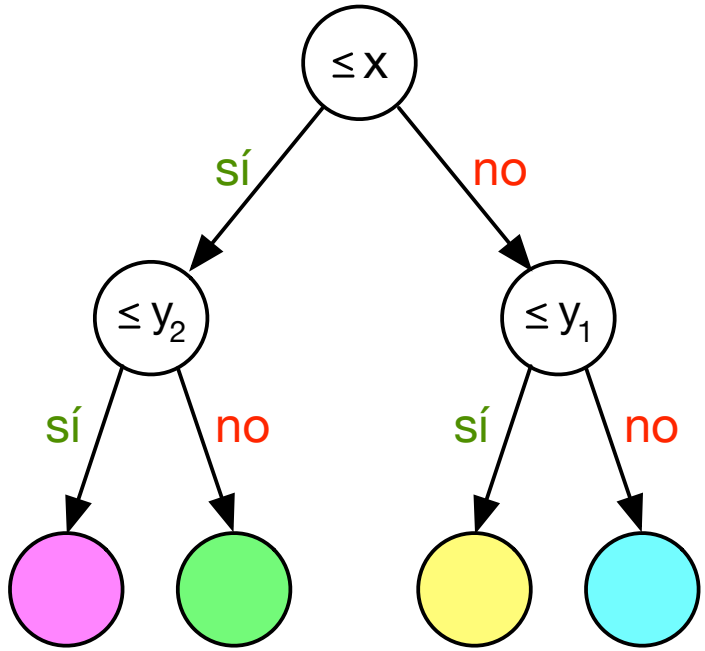
Construcción de un árbol de decisión

Construcción de un árbol

1. Comenzar con un **nodo raíz**, s_0, que incluya todo el conjunto de entrenamiento.
2. Dividir s_0 en dos **nodos ramas** (s_1 y s_2) con el algoritmo `stump`.
3. Colocar s_1 y s_2 en una lista de nodos ramas a explorar $\mathcal{B}$. Cada rama sólo recibe un subconjunto del conjunto de entrenamiento: s_1 recibe las observaciones que en el stump en s_0 son clasificadas como Ciertas y s_2 las Falsas.
4. Aplicar la siguiente **recursividad** en cada nodo s_k en $\mathcal{B}$. Explorar s_k con el algoritmo `stump` y el subconjunto de entrenamiento recibido de su nodo padre.
   - Si s_k is **homogéneo** (el stump no divide las observaciones) s_k es un **nodo hoja**. s_k consigue una etiqueta (la clase mayoritaria entre todas las observaciones de entrenamiento que le llegaron)
   - Si s_k es dividido en dos ramas (s_j y s_l) por el stump:
     - Incluir s_j y s_l en $\mathcal{B}$
     - Asignar a s_j y s_l los correspondientes subconjuntos de las observaciones de entrenamiento que llegaron a s_k
   - Eliminar s_k de $\mathcal{B}$
5. Ejecutar la recursividad hasta que $\mathcal{B}$ está vacío

Ejemplo de un árbol para clasificación





Medidas de "impureza" para clasificación

Cada nodo en el árbol es explorado con el algoritmo `stump`. Este algoritmo necesita una métrica que permita evaluar la **calidad** de la división. Las métricas más comunes para problemas de clasificación son:

- Tasa de clasificación de error
- **Coeficiente de Gini**. En un problema con K clases, se define p_k como la fracción de observaciones en el nodo que pertenecen a la clase t_k

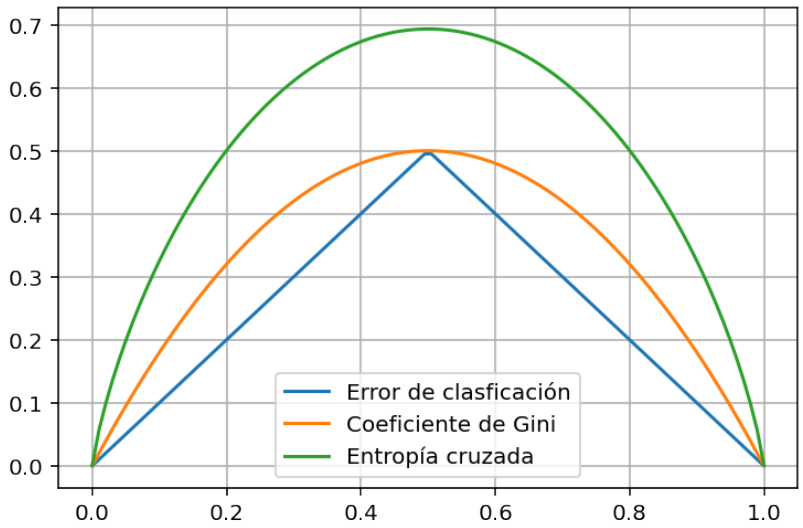
$$\text{Gini} = 1 - \sum_{k=1}^{K} p_k^2$$

- **Entropía cruzada**

$$-\sum_{k=1}^{K} p_k \log p_k$$

Intuitivamente, el índice de Gini o la entropía cruzada parecen ser una mejor opción que la tasa de error, ya que en casos en el que dos divisiones presenten una tasa de error igual, ambas métricas permiten una división descendente más "pura".

Medidas de impureza para un problema de clasificación con 2 clases



(Para 2 clases, Gini $= 2p(1-p)$, y la entropía cruzada es, $-p\log(p) - (1-p)\log(1-p)$, donde p es la proporción de observaciones en la primera clase).

Evaluación de un árbol

Evaluación de un árbol

1. La evaluación de cada observación comienza con el stump en el nodo raíz s_0. La salida de este test conduce cada observación a través de uno de los dos nodos (s_1 o s_2) que descienden de s_0.
2. La observación recursivamente fluye hacia abajo de la jerarquía del árbol, en función de lo que decidan los stumps en las ramas.
3. La observación finaliza su recorrido cuando llega a un nodo hoja. La observación es clasificada con la clase de dicho nodo hoja.

Controlar el crecimiento del árbol

- El algoritmo para hacer crecer el árbol finaliza cuando $\mathcal{B}$ está vacío. Esto significa que todas las observaciones del conjunto de entrenamiento han terminado en un nodo hoja. En el peor de los casos, el árbol terminará con muchas hojas, donde cada hoja clasifica una única observación de entrenamiento (pobre generalización).
- Nótese que la calidad de cada clasificador stump en los nodos rama depende del tamaño del subconjunto de entrenamiento que le llega. Por esta razón, a medida que el árbol crece (en profundidad), la calidad de los clasificadores stump empeora.
- Para evitar esta situación, se pueden introducir algunas "reglas" externas que permitan controlar el crecimiento del árbol. Estás reglas están codificadas en **hiperparámetros** que se evalúan a lo largo de los correspondientes pasos del algoritmo de crecimiento del árbol.

Hiperparámetros para el crecimiento del árbol

- Criterio que permita evaluar la calidad de los stumps: "giny" o "entropía"
- Profundidad máxima del árbol
- Número mínimo de observaciones en un nodo para que éste sea dividido. Si s_k recibe menos observaciones, se convierte en un nodo hoja.
- Número mínimo de observaciones en un nodo para que éste sea considerado un nodo hoja. Si una rama recibe menos muestras, el nodo padre se convierte en nodo hoja.
- Número de características exploradas en el algoritmo `stump`. El algoritmo `stump` no explora todas las columnas de $\mathbf{X}$, sólo un subconjunto aleatorio en cada división.
- Número máximo de hojas. Explorar $\mathcal{B}$ de la mejor manera posible hasta que se alcanza ese límite.
- Mínimo decremento en el parámetro de impureza utilizado para dividir un nodo. Si las impurezas (ponderadas) de dos ramas de un nodo s_k no se ven reducidas ese límite, entonces s_k es considerado un nodo hoja.

Parámetros e hiperparámetros

Durante el entrenamiento de un modelo de aprendizaje máquina como es un árbol de clasificación, se necesita encontrar valores para dos clases de parámetros:

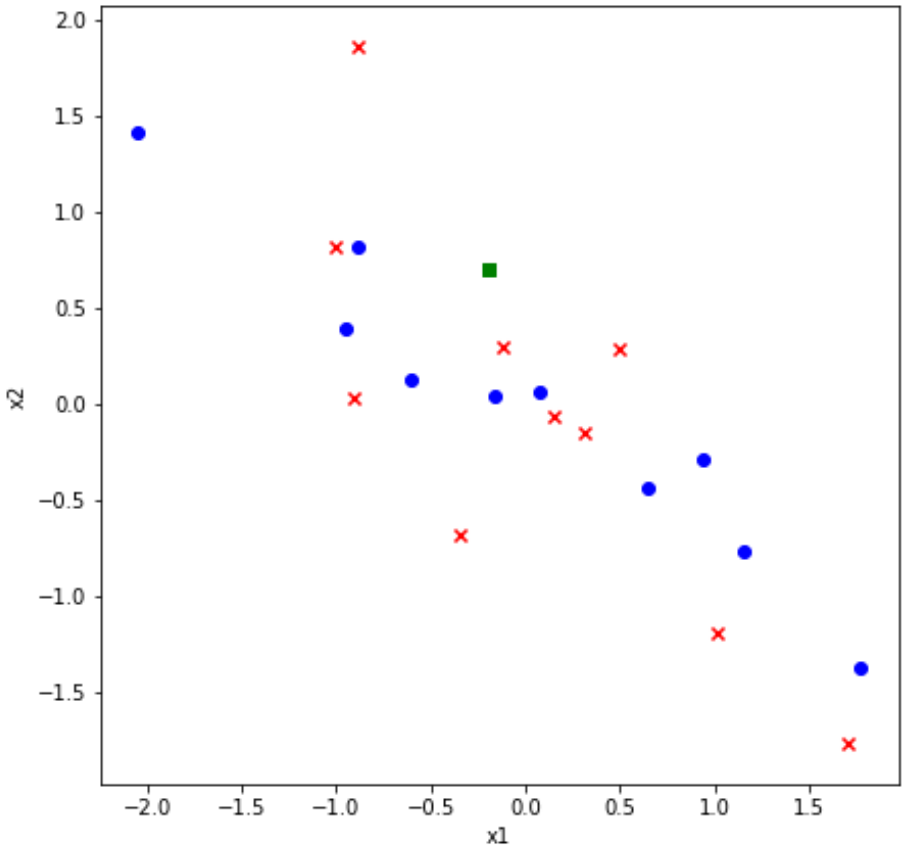
Parámetros ajustables

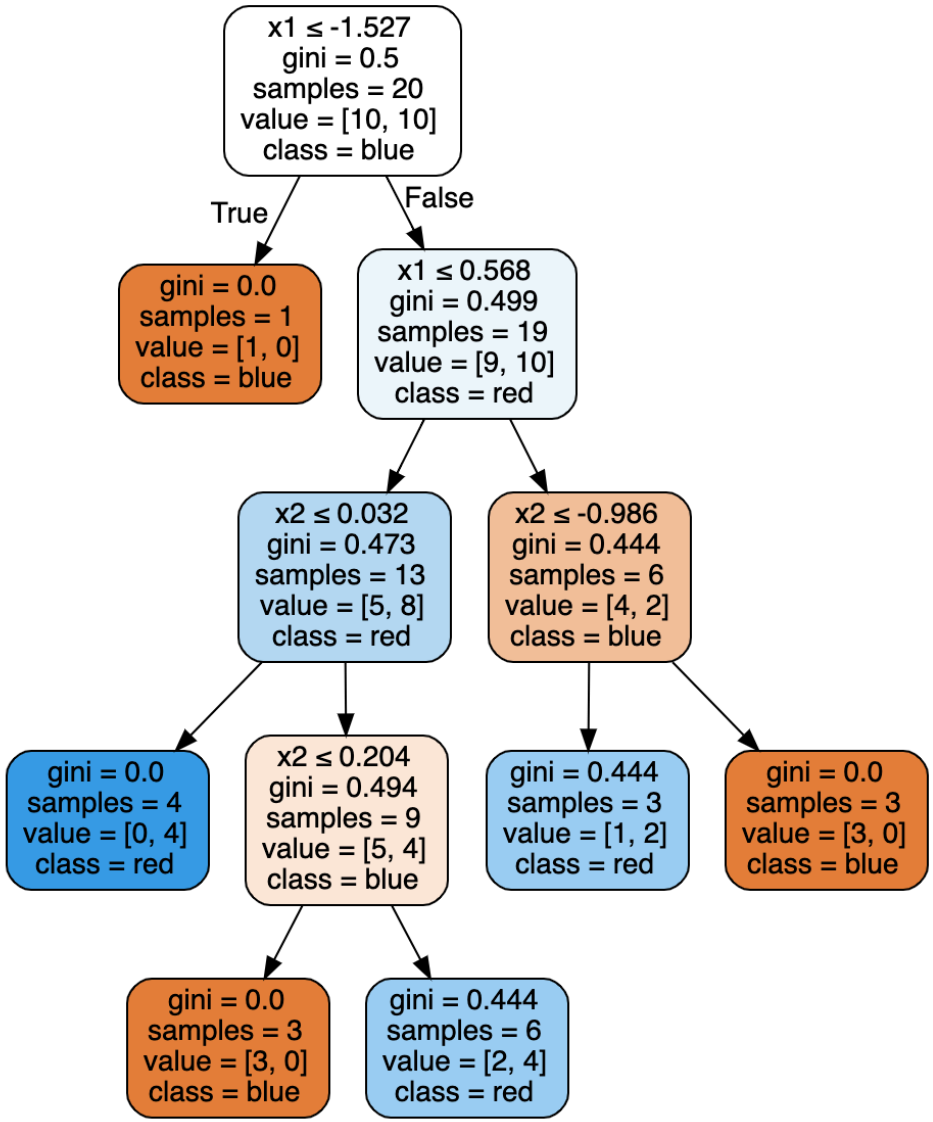
Parámetros que definen cada stump en el árbol: la característica j y umbral η. Sus valores son ajustados durante el proceso de optimización implicado en el ajuste del modelo.

Hiperparámetros

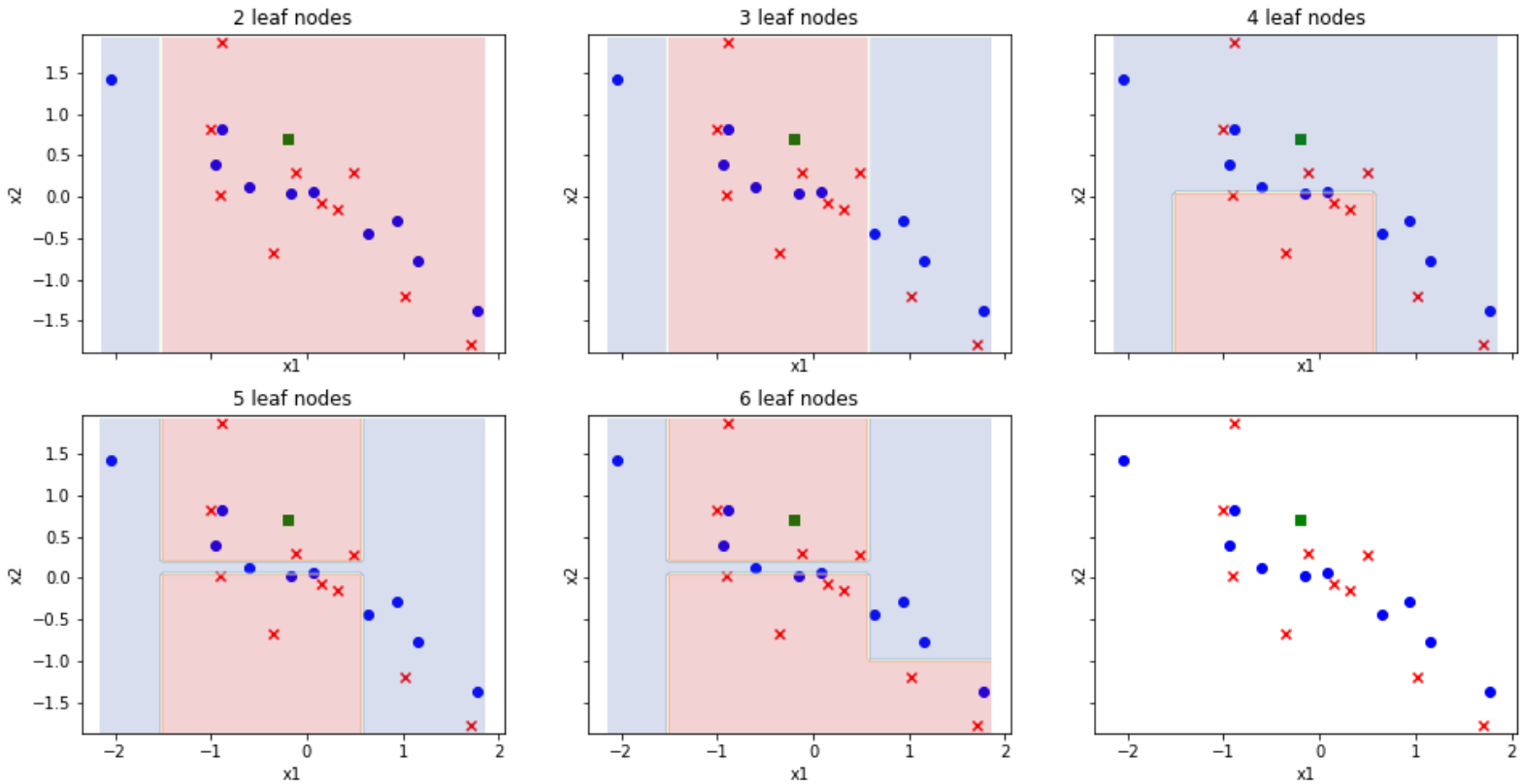
Su papel es controlar la optimización dentro del ajuste del modelo. Sus valores se ajustan usando **conocimiento previo** del problema. De esta forma controlan la capacidad de generalización del modelo (ej.: la profundidad máxima del árbol de decisión).

Clasificación con 2 clases en 2D





Regiones de clasificación en árboles de decisión



Métodos de conjunto

Una forma común de desarrollar modelos de aprendizaje máquina robustos es entrenar un **conjunto** de modelos con un mismo conjunto de entrenamiento pero introduciendo una pequeña cantidad de **diversidad** en cada miembro del conjunto:

- **Submuestreo** del conjunto de entrenamiento para conseguir un conjunto ligeramente diferente para cada aprendiz.
- Usar diferentes hiperparámetros (quizás elegidos de forma aleatoria) para cada aprendiz.

Aprendices débiles

Cada miembro del conjunto se denomina **aprendiz débil** ya que no es necesario que utilice un algoritmo de aprendizaje muy preciso. En clasificación binaria, hay garantías de que cada aprendiz débil es capaz de obtener una tasa de acierto ligeramente superior al 50 %, de manera que el conjunto de todos ellos puede converger a mejores prestaciones.

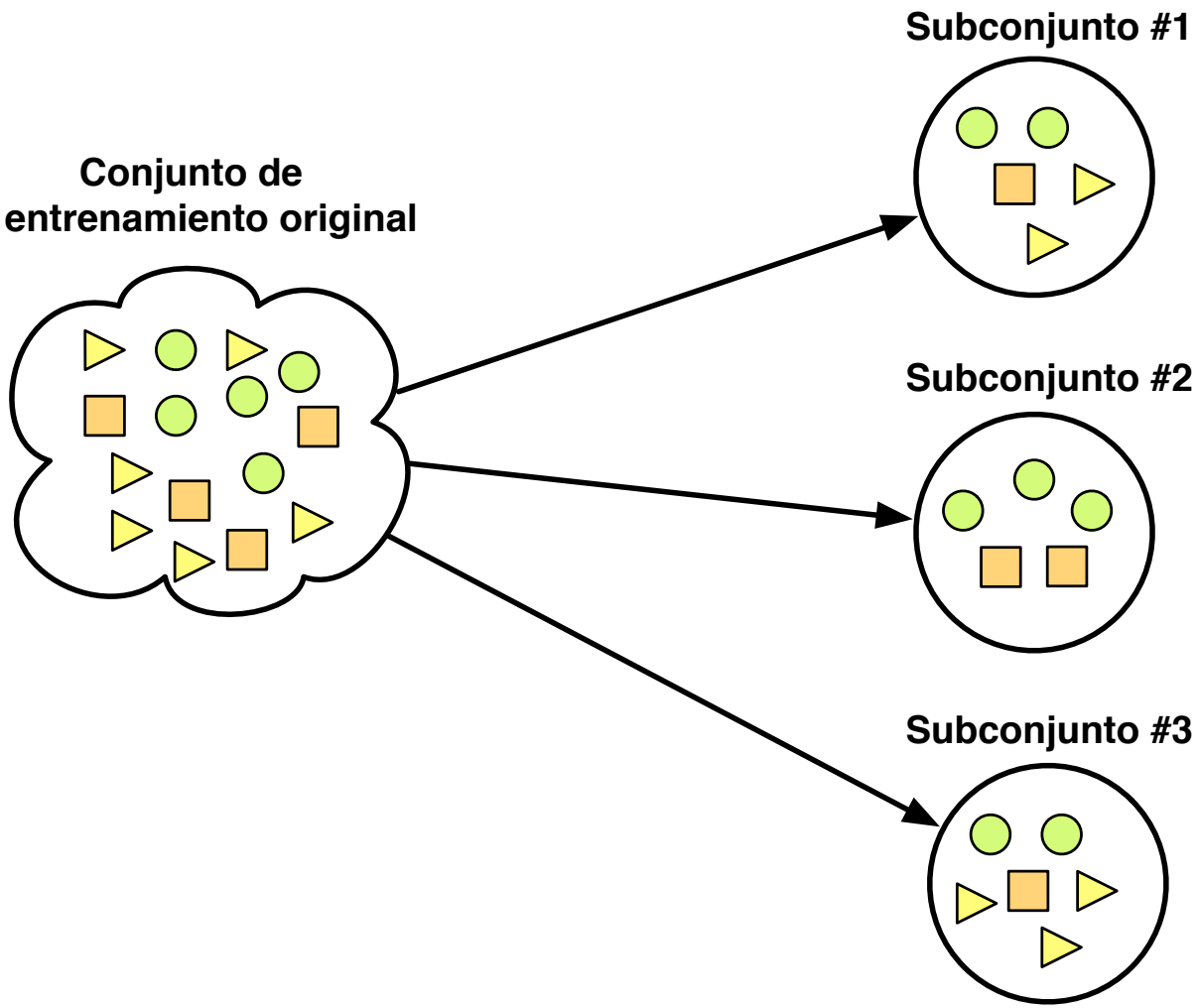
Una vez diseñado el conjunto, la predicción para cada observación del conjunto de test se realiza combinando las predicciones individuales de cada aprendiz débil.

- Calculando la media de las predicciones individuales
- Calculando la moda de las predicciones individuales
- Ponderando la predicción de cada aprendiz por un coeficiente que refleja la "confianza" en sus predicciones.

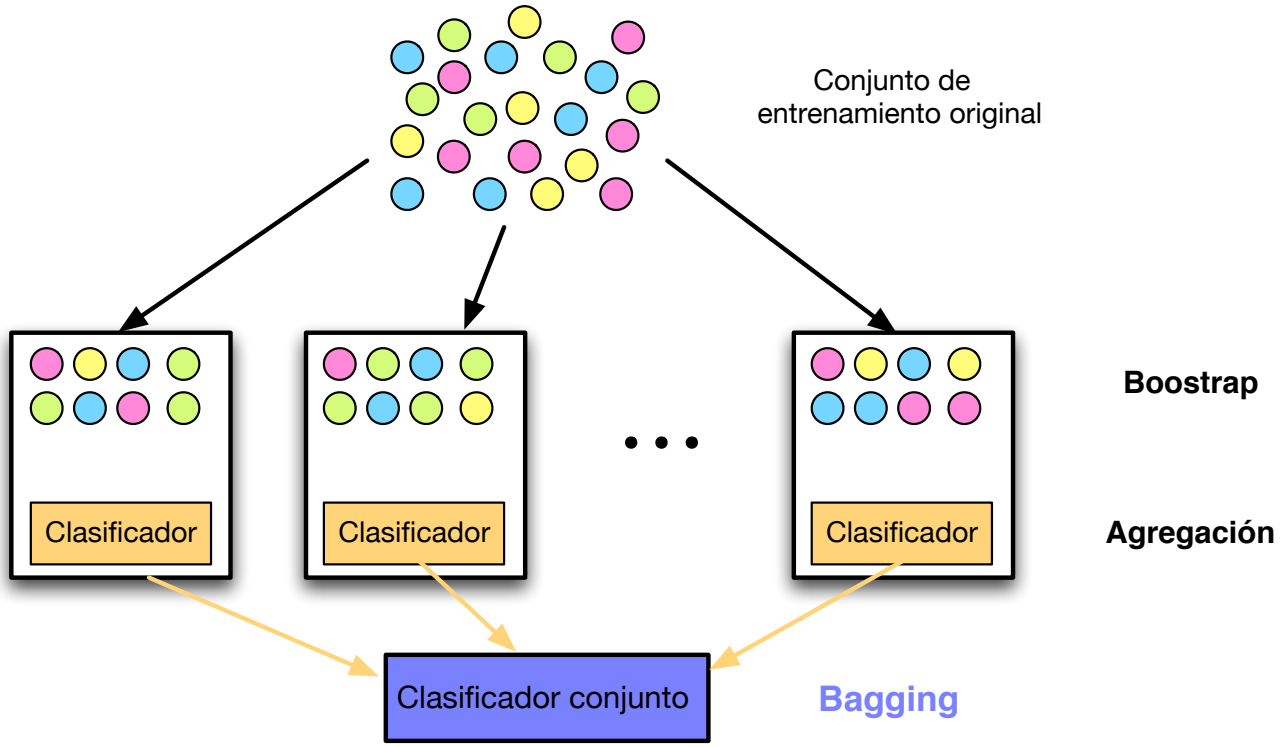
Bagging

- Cada aprendiz débil se entrena con el mismo algoritmo, pero con un subconjunto de entrenamiento diferente extraído del conjunto de entrenamiento original (**bootstrap**)
- Los aprendices débiles son entrenados en paralelo
- La salida del conjunto es la media de las salidas de los aprendices débiles
- Ejemplo: Random Forest

Ejemplo de Boostraping



Ejemplo de Bagging



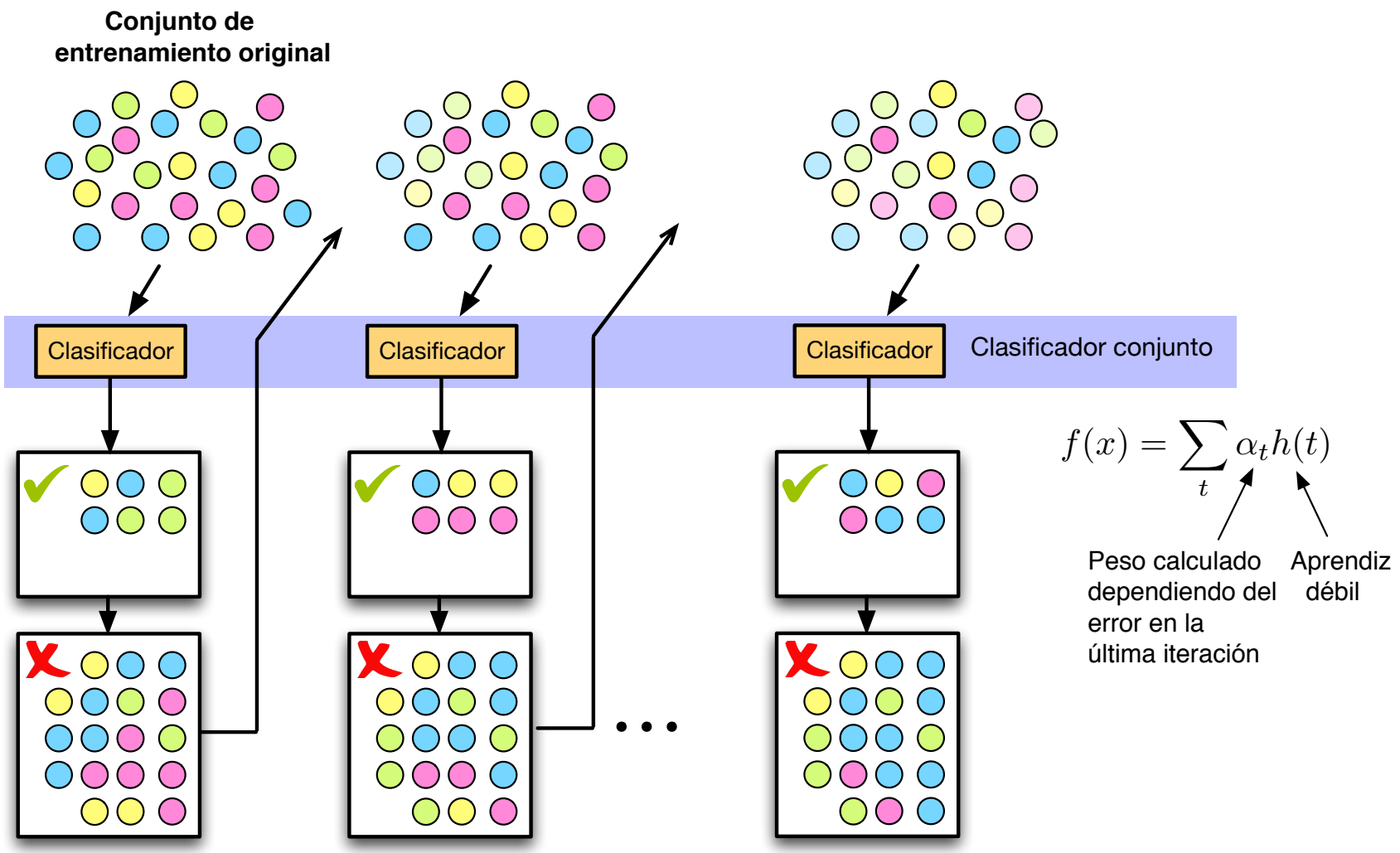
$$f(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

Aprendiz débil

Boosting

- Los aprendices débiles son entrenados secuencialmente usando el mismo algoritmo, pero con conjuntos de entrenamiento ligeramente diferentes.
- El primer aprendiz débil se entrena con el conjunto de entrenamiento original.
- El segundo aprendiz débil recibe un conjunto de entrenamiento construido remuestreando el conjunto original, de forma que se da más importancia a aquellas observaciones mal clasificadas por el primer aprendiz débil.
- Los subsiguientes aprendices débiles reciben un conjunto de entrenamiento construido remuestreando el conjunto original, de forma que se da más importancia a aquellas observaciones mal clasificadas por el conjunto formado por los aprendices débiles anteriores.
- Cada aprendiz débil se añade al conjunto con el propósito de corregir los errores de sus predecesores.
- La salida del conjunto es la media ponderada de las salidas de los aprendices débiles. Estos pesos son proporcionales a la precisión de cada aprendiz débil.

Ejemplo de Boosting



$$f(x) = \sum_t \alpha_t h(t)$$

Peso calculado dependiendo del error en la última iteración

Aprendiz débil

Comité de expertos

- El conjunto no está formado por aprendices débiles, sino por algoritmos diversos y sofisticados
- Cada algoritmo se especializa en aprender una parte del problema
- Un modelo de aprendizaje máquina, denominado **gate**, se ajusta para decidir qué experto clasificará cada observación de test, usando como información, la precisión de los diferentes expertos en el conjunto de entrenamiento

Entrenamiento de Random Forest

Random Forest es un conjunto de B árboles de decisión.
Básicamente consiste en un bucle que ejecuta una iteración para cada árbol (B iteraciones). El algoritmo recibe como entrada un conjunto de entrenamiento con N observaciones en d dimensiones, con sus correspondientes etiquetas, y el tamaño del conjunto (B) (número de árboles de decisión).

Algoritmo Random Forest

For $b = 1, \ldots, B$:

1. Elegir un subconjunto aleatorio de N_b observaciones del conjunto de entrenamiento
2. Entrenar el árbol T_b con ese subconjunto de entrenamiento, pero antes de optimizar cada stump para cada nodo rama, se eligen aleatoriamente, m_b características del conjunto disponible d. De esta forma, cada subconjunto sólo accede a un subconjunto del espacio de características de entrada para optimizar cada stump, lo que introduce mayor diversidad en el árbol
3. Inferencia. Cada árbol proporciona una salida para cada observación de test. La clase final es la clase más votada

Regiones de clasificación en Random Forest