



Ayudantía 9

SVM, Gradient Boosting y presentación de la tarea 4

Por Daniel Alegría e Ignacio Garrido

24 de Octubre 2025



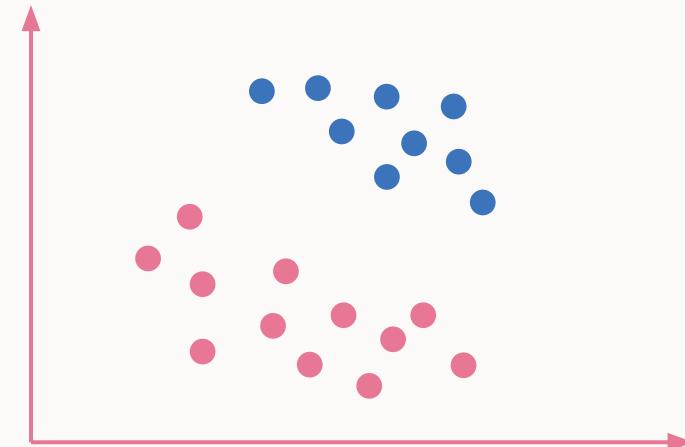
Support Vector Machine (SVM)



¿Qué es una SVM?

Es un conjunto de métodos **supervisados** de machine learning diseñados para realizar tareas de **clasificación**.

El objetivo es encontrar una **frontera de decisión óptima** que separe correctamente los datos. Para luego poder **clasificar** nuevos datos.



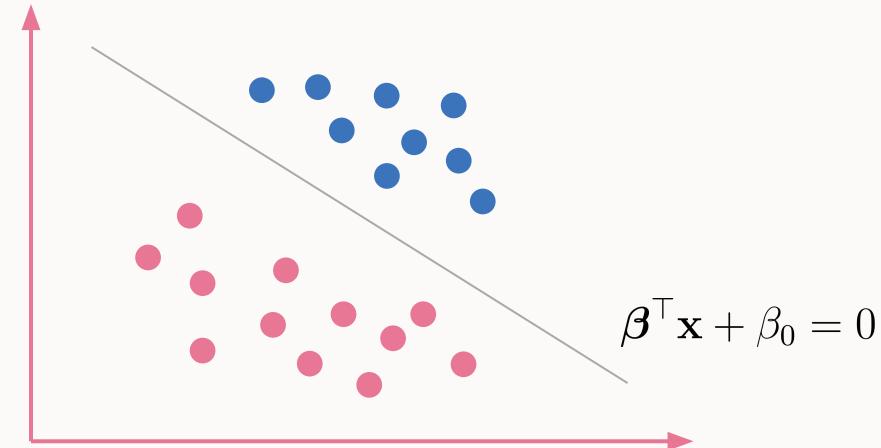


Conceptos importantes

Hiperplano: generalización del concepto de plano a espacios de dimensión arbitraria. Es un subconjunto de un espacio \mathbb{R}^n que tiene dimensión $n-1$.

β : vector normal que define la orientación del hiperplano

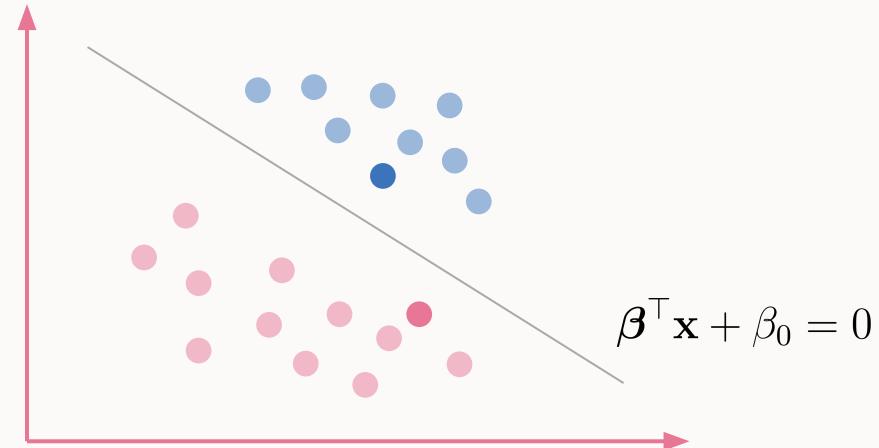
β_0 : define la posición relativa al origen.





Conceptos importantes

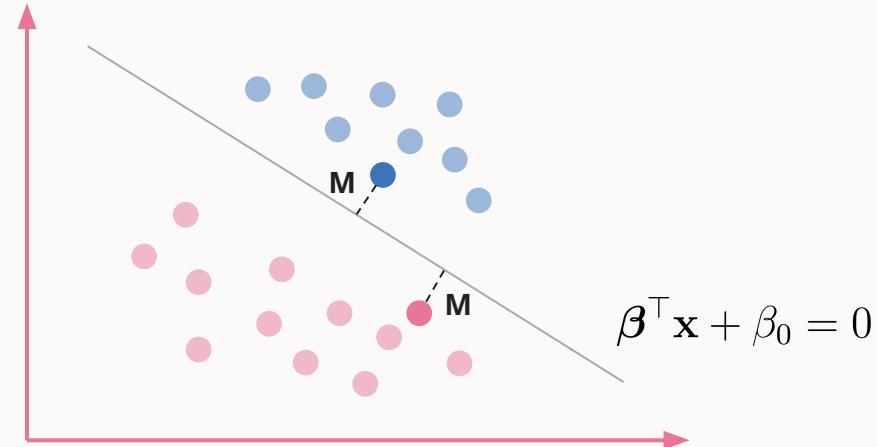
Vectores de soporte: puntos que están más cerca del hiperplano separador.





Conceptos importantes

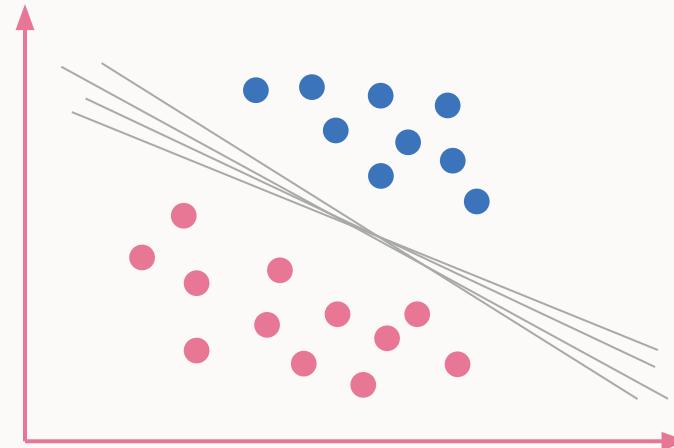
Margen: distancia mínima entre los puntos más cercanos y el hiperplano.





Funcionamiento

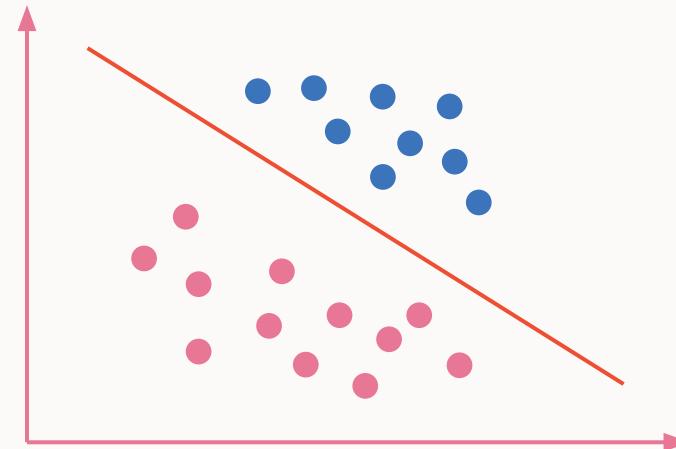
Existen **infinitas** rectas que logran separar los datos.





Funcionamiento

Existen **infinitas** rectas que logran separar los datos.
¿Cómo elegimos la mejor?





Funcionamiento

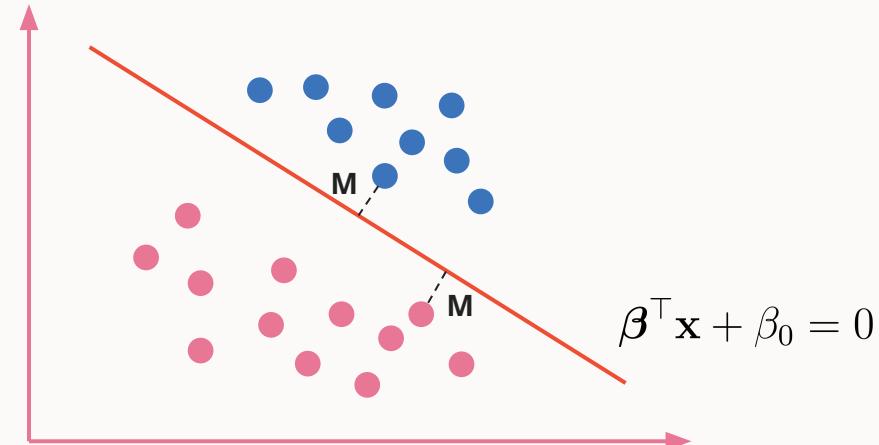
SVM busca encontrar aquel hiperplano que **maximiza** el margen:

$$\max_{\beta, \beta_0} M$$

$$\text{s.a. } \frac{1}{\|\beta\|} y_i (\mathbf{x}_i^\top \beta + \beta_0) \geq M, \quad i = 1, \dots, N$$

$$\min_{\beta, \beta_0} \frac{1}{2} \|\beta\|^2$$

$$\text{s.a. } y_i (\mathbf{x}_i^\top \beta + \beta_0) \geq 1, \quad i = 1, \dots, N$$





Soft-margin SVM

La restricción impone que el hiperplano clasifique correctamente **todos** los puntos.

Muy sensible al **ruido** o **outliers**: un solo punto mal etiquetado puede hacer que **no exista solución**.

Soft-margin SVM permite que algunos puntos estén **dentro** del margen o sean **mal clasificados**.



Soft-margin SVM

$$\min_{\boldsymbol{\beta}, \beta_0} \frac{1}{2} \|\boldsymbol{\beta}\|^2$$

$$\text{s.a. } y_i (\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1, \quad i = 1, \dots, N$$

Hard-margin

$$\begin{aligned} & \min_{\boldsymbol{\beta}, \beta_0} \quad \frac{1}{2} \|\boldsymbol{\beta}\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.a. } & y_i (\mathbf{x}_i^\top \boldsymbol{\beta} + \beta_0) \geq 1 - \xi_i, \quad i = 1, \dots, N \\ & \xi_i \geq 0, \quad i = 1, \dots, N \end{aligned}$$

Soft-margin

$\xi=0$: El punto está bien clasificado y fuera del margen.

$0 < \xi < 1$: El punto está bien clasificado, pero dentro del margen.

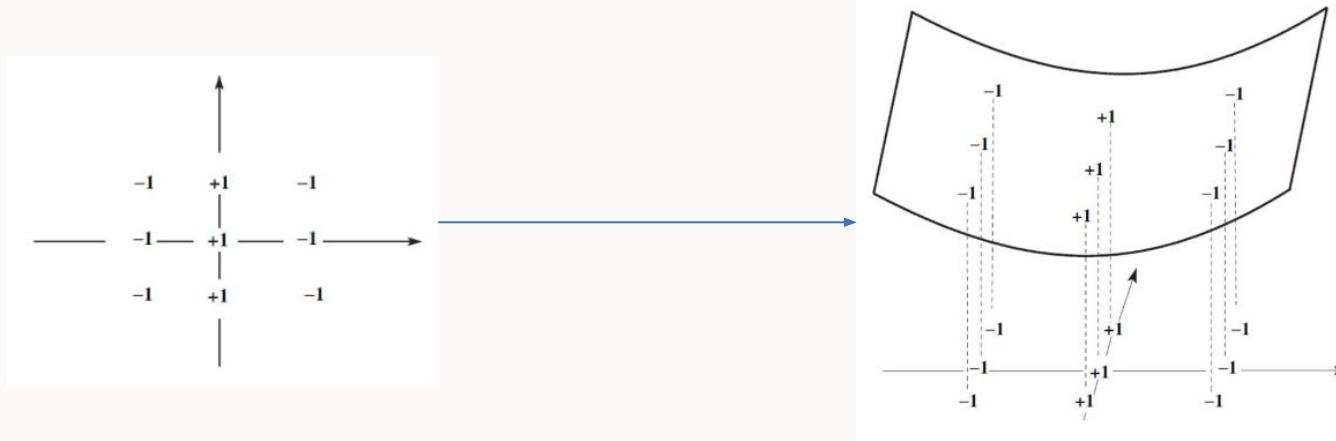
$\xi \geq 1$: El punto está mal clasificado.



Kernel

SVM sigue limitado a ser un **clasificador lineal**

→ **transformación** a las variables para poder representar **más dimensiones** y así poder **clasificar** los datos



$$\text{Espacio de características } f(x_1, x_2) = x_1^2$$

¿Cómo puedo transformar los datos a otro espacio?

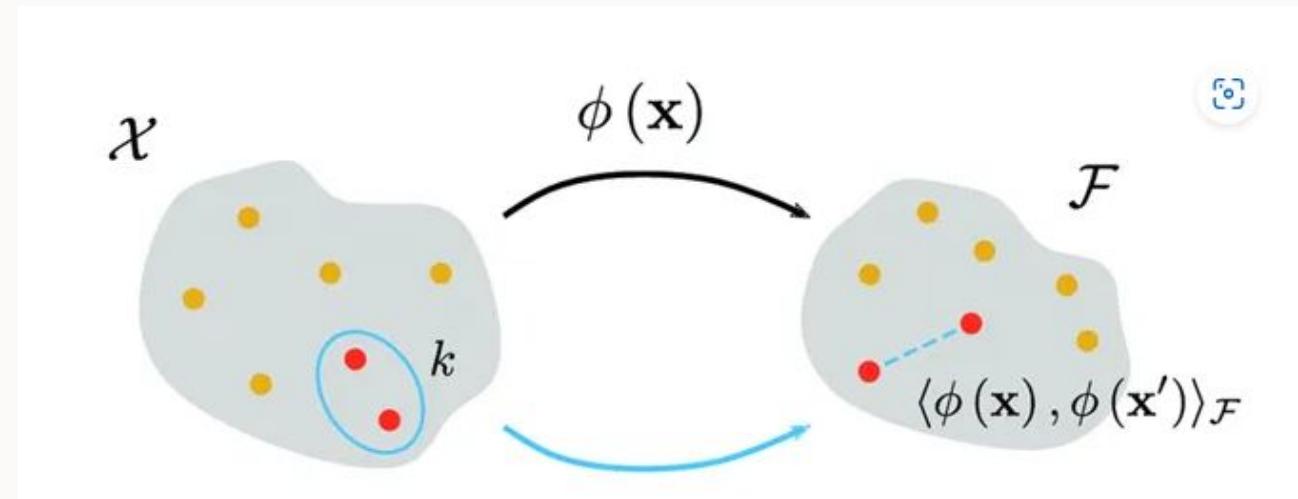


La idea nace de ver SVM dual

$$\max_{\alpha} \sum_{i=1}^M \alpha_i - \frac{1}{2} \sum_{i,j=1}^M y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)} \cdot x^{(j)}$$

subject to $0 \leq \alpha_i \leq C$ for all i , and $\sum_{i=1}^M \alpha_i y^{(i)} = 0$.

¿Cómo puedo transformar los datos a otro espacio?



¿Cómo puedo transformar los datos a otro espacio?



$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

Lineal

$$K(x, x') = x^T x'$$

Polinomial

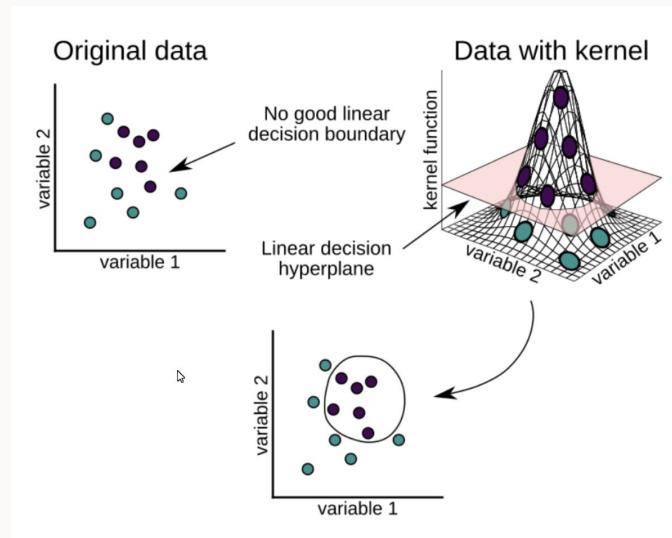
$$K(x, x') = (x^T x' + c)^d$$

RBF

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$



Función de Kernel

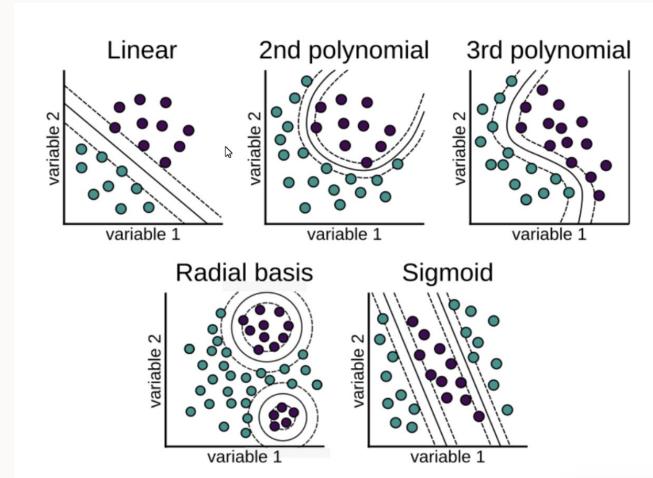


Conjunto de funciones que permite **transformar** el espacio de características con el que trabajamos

Entonces... ¿Qué es una función de Kernel?



Función que cambian el “punto de vista” de los datos para encontrar similitudes



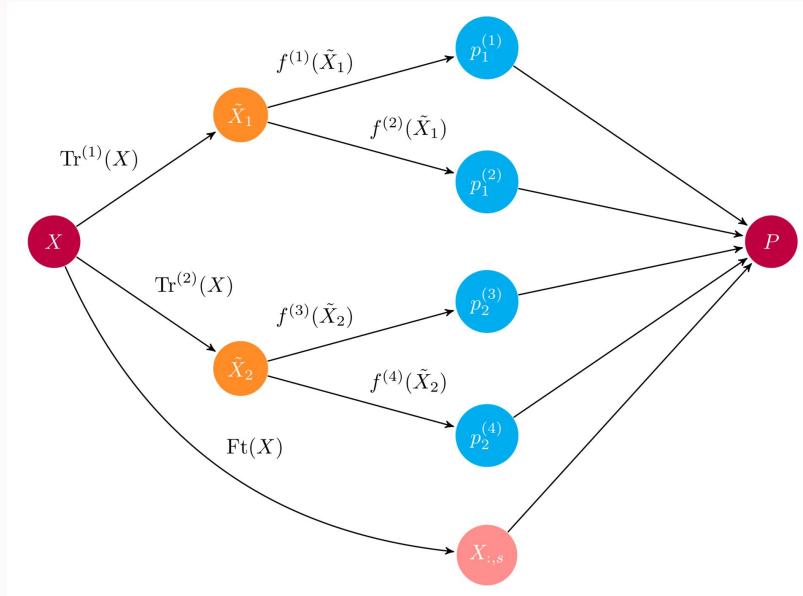
Existen varios tipos de Kernel para **distintas aplicaciones**



Gradient Boosting



Pero antes debemos hablar sobre qué es un ensamble (ensemble method)



Fuente: dataquest.io



¿Qué son los ensambles?

- es una técnica que **combina múltiples modelos base** (denominados *learners* o *weak learners*) para producir un modelo final que **tiene mejor desempeño** que cualquiera de ellos por separado.



¿Qué son los ensambles?

- La idea central es que “**varios modelos débiles pueden formar un modelo fuerte**”, si se combinan de manera adecuada.



Fuente: National Geographic



¿Por qué funciona esto?

- Esto se basa en un principio estadístico: al **promediar** o **ponderar** predicciones de modelos diversos, se reducen los errores individuales (sesgo o varianza).





RESULTADO REAL

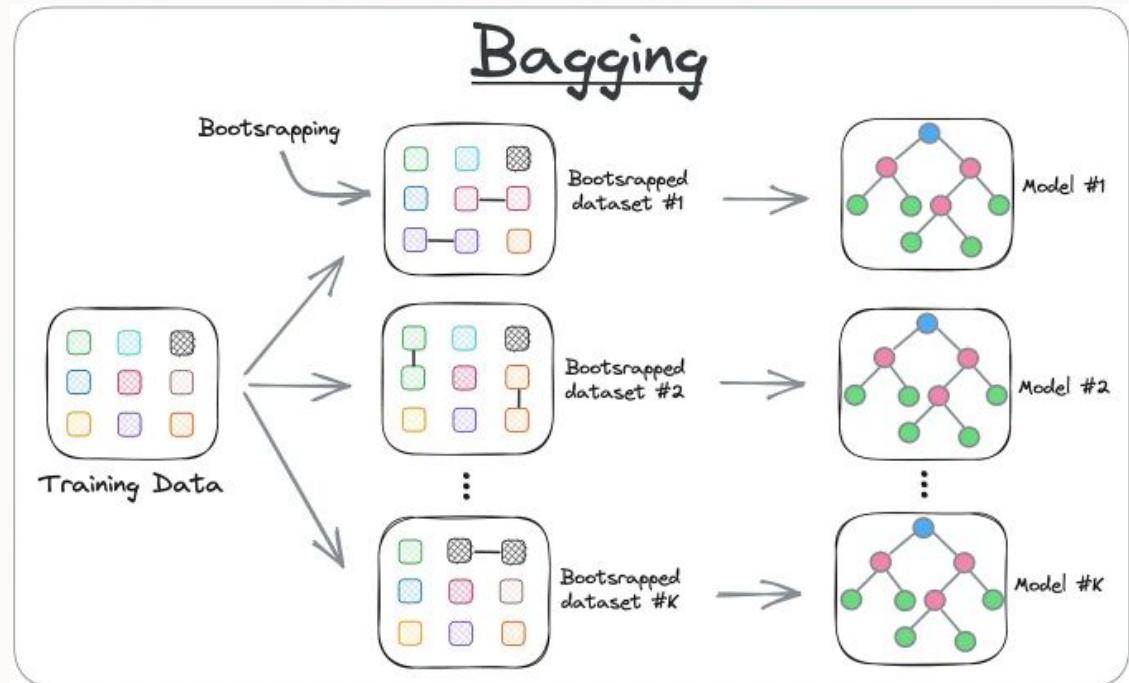
¡LA SABIDURÍA DE LA MULTITUD!

A brown sack tied with a brown strap sits on a digital scale. The scale's display shows "48 kg". To the right of the scale is a large green checkmark. Below the scale is a white speech bubble containing the mathematical formula: $45+58+50+42/4 = 48.75 \text{ kg}$.



Tipos de ensambles : *Bagging*

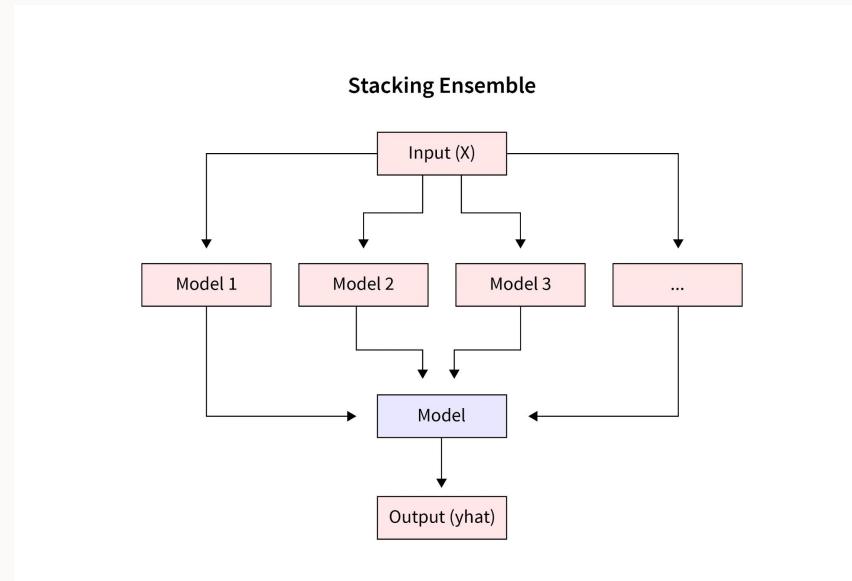
- Crear **múltiples datasets** usando muestreo **con reemplazo** y entrenar distintos modelos.
- Luego por alguna **decisión conjunta**, por ejemplo mayoría de votación se clasifica/predice.
- Random Forest es el clásico de estos modelos.





Tipos de ensambles : *Stacking*

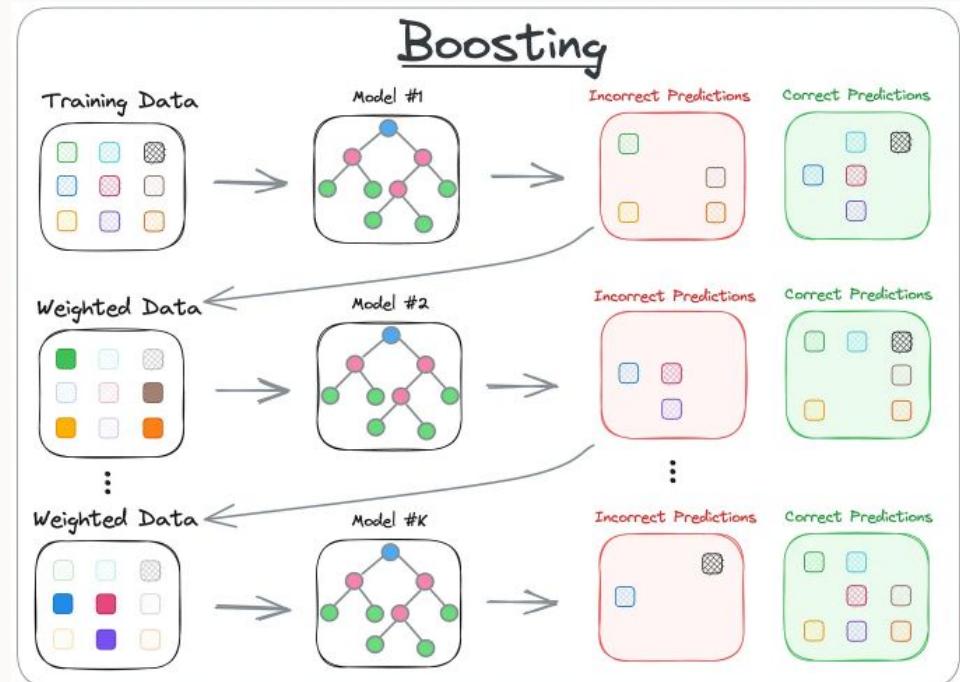
- Entrenar varios modelos con el mismo dataset. Se recomienda que sean ***weak learners***.
- Luego ***stackear*** (concatenar) los *outputs* de los modelos y usarlos como *inputs* para entrenar un ***metalearner***.
- Finalmente este *metalearner* genera el *output* esperado.





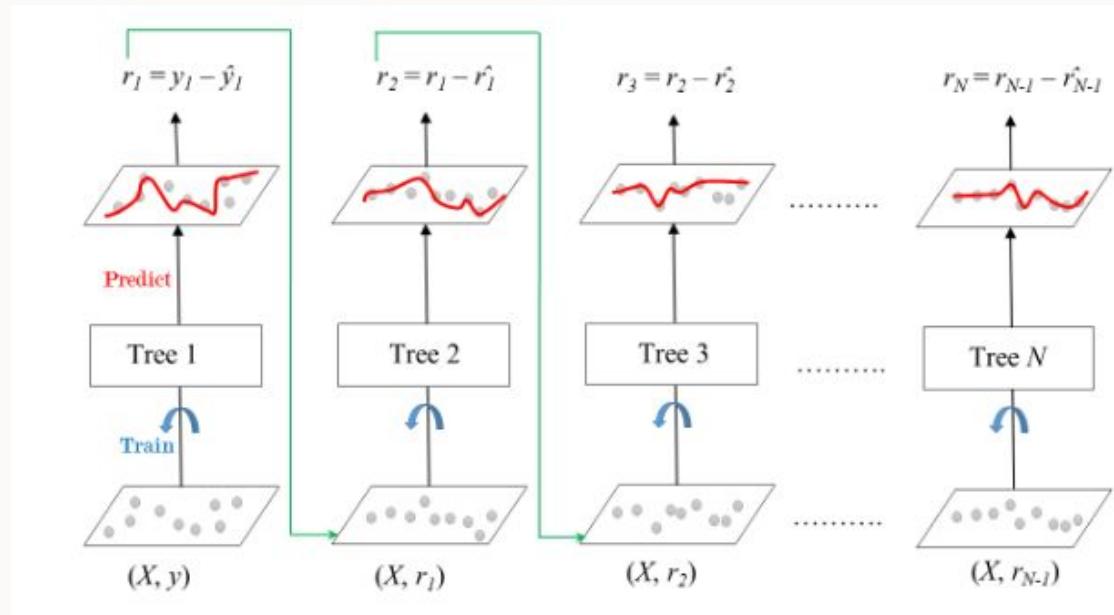
Tipos de ensambles : *Boosting*

- Aprovechar la **información errónea** de una jerarquía de modelos.
- De forma secuencial se entrena modelo cada vez más débiles, para que estos **aprendan de los errores de los anteriores**.
- Luego se **combinan** para generar el output.





Gradient Boosting





Gradient Boosting

A diferencia de otros métodos como *Random Forest* (que construye árboles en paralelo), Gradient Boosting lo hace de forma **secuencial**: cada nuevo árbol se entrena para corregir los errores del árbol anterior.



1. Primer árbol (Tree 1)

- **Entrada:** los datos originales, compuestos por las características X y la variable objetivo real y .
- **Entrenamiento:** se entrena el primer árbol para predecir y a partir de X .
- **Predicción:** el árbol genera su primera predicción \hat{y}_1 .
- **Cálculo del error (residual):**

$$r_1 = y - \hat{y}_1$$

Este residual r_1 representa aquello que el primer árbol no logró explicar de los datos.



2. Segundo árbol (Tree 2)

Aquí ocurre la esencia del método.

- **Entrada:** el segundo árbol se entrena nuevamente con las mismas características X , pero ahora su objetivo es predecir los **residuales** r_1 del árbol anterior (no y directamente).
- **Entrenamiento:** el Árbol 2 aprende a modelar los errores del Árbol 1.
- **Predicción:** genera una estimación de los residuales, \hat{r}_1 .
- **Nuevo residual:**

$$r_2 = r_1 - \hat{r}_1$$

Este nuevo residual r_2 es el error que aún queda sin corregir tras los dos primeros árboles.



3. Árboles siguientes (Tree 3 hasta Tree N)

Cada nuevo árbol intenta **mejorar** la predicción global aprendiendo los errores residuales de los árboles anteriores.

El resultado final del modelo no corresponde a la salida del último árbol, sino a la **suma acumulada** de todas las correcciones:

$$\hat{y}_{final} = \hat{y}_1 + \hat{r}_1 + \hat{r}_2 + \cdots + \hat{r}_{N-1}$$



Algorithm 2 Gradient Boosting

Input Data: $x_i \in R^n, y_i \in [-1, +1]$

1: **Initialize** $F_0(x) = argmin_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$

2: **for** $m = 1$ to M **do**

3: Compute the negative gradient

$$r_{i,m} = - \left[\frac{\delta L(y_i, F(x_i))}{\delta F(x_i)} \right]$$

4: Fit tree to targets $r_{i,m}$, with leafs $R_{j,m}$

5: For $j = 1, 2, \dots J_m$

$$\gamma_{jm} = argmin_{\gamma} \sum_{x_i \in R_{ij}} L(y_i, F_{m-1}(x) + \gamma)$$

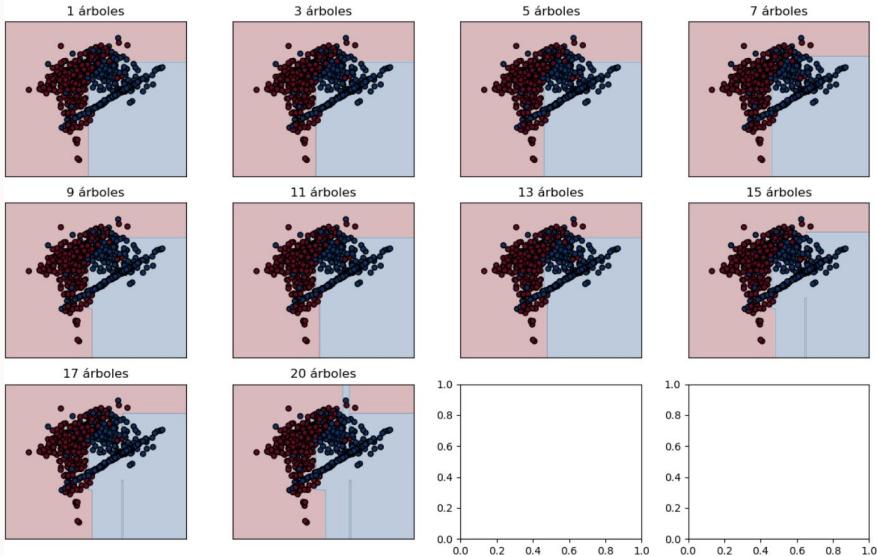
6: Update the estimation of $F(x)$

$$F_m(x) = F_{m-1}(x) + \nu(\gamma_{jm})$$

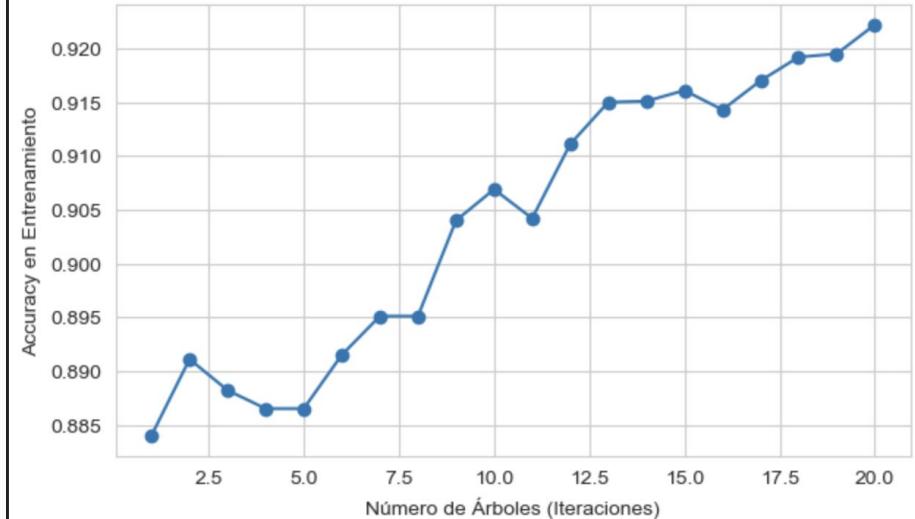
7: Output $\hat{f}(x) = f_M(x)$



Evolución de la frontera de decisión (Gradient Boosting)



Evolución del Accuracy con Gradient Boosting





Gradient Boosting : Clasificación

Si los modelos son de **regresión**, el error se calcula simplemente como la **diferencia entre el valor real y el valor predicho**.

Sin embargo, en los **problemas de clasificación** la situación es distinta: no podemos calcular un “error” como una resta directa entre clases, ya que las etiquetas **no son valores numéricos continuos** ni poseen una distancia significativa entre sí.

En cambio, los modelos de clasificación producen **probabilidades** de pertenecer a cada clase.

Por lo tanto, el error en este contexto se mide en función de **cuánto difieren las probabilidades predichas respecto a las verdaderas etiquetas**, generalmente mediante una **función de pérdida** apropiada, como la *log-loss* (entropía cruzada).



XGBoost : eXtreme Gradient Boosting

- Probablemente el algoritmo de boosting más utilizado.
- *Gradient Boosting no escala bien para muchos datos.*
- **XGboost:** Es una versión más eficiente, puesto que:
 - Agrega regularizadores L1 y L2 (quitan el overfitting)
 - Paralelizar las tareas
 - Tiene implementado una forma de **parar** basado en Cross-Validation
 - Corta las zonas menos interesantes de árboles.