

# Modelos de Ensamble: De Random Forest a XGBoost

## HE2: Consultoría Económica con IA Responsable

Santiago Neira & Catalina Bernal

Universidad de los Andes  
Departamento de Economía

Febrero 2026

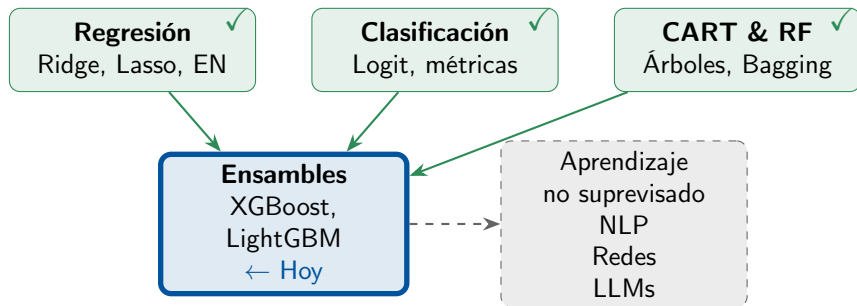
# Agenda de hoy

- 1 Roadmap y Recap
- 2 Extra-Trees: Aleatorización Extrema
- 3 AdaBoost: El Origen del Boosting
- 4 Gradient Boosting y XGBoost
- 5 LightGBM: Velocidad y Escala
- 6 Panorama Comparativo

# Agenda

- 1 Roadmap y Recap
- 2 Extra-Trees: Aleatorización Extrema
- 3 AdaBoost: El Origen del Boosting
- 4 Gradient Boosting y XGBoost
- 5 LightGBM: Velocidad y Escala
- 6 Panorama Comparativo

# ¿Dónde Estamos?



La clase pasada construimos **un** árbol y después **muchos árboles independientes** (Random Forest). Hoy la pregunta cambia: **¿qué pasa si los árboles aprenden de los errores de los anteriores?**

# Las Dos Filosofías de los Ensamblados

## Bagging (paralelo)

Muchos modelos **independientes** entrenados en datos distintos. Se **promedian** las respuestas.

Random Forest, Extra-Trees  
*Reduce varianza*

vs.

## Boosting (secuencial)

Cada modelo **corrige** los errores del anterior. Se **acumulan** las correcciones.

AdaBoost, XGBoost, LightGBM  
*Reduce sesgo y varianza*

## Analogía para el cliente

**Bagging:** Pedirle opinión a 500 analistas independientes y tomar el consenso. Cada uno ve datos ligeramente distintos. **Boosting:** Un analista que revisa su trabajo iterativamente, enfocándose cada vez más en los casos que falló.

# Recap Rápido: Random Forest

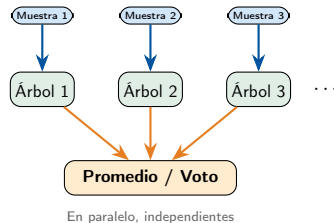
## ¿Qué hace RF?

- $B$  árboles, cada uno con una **muestra bootstrap**
- En cada split: subconjunto aleatorio de  $m$  features
- Predicción = promedio (regresión) o voto (clasificación)

## ¿Por qué funciona?

- Árboles **decorrelacionados**  $\rightarrow$  promediar reduce varianza
- Cada árbol tiene alto sesgo bajo pero alta varianza; el promedio cancela el ruido

**Limitación clave:** RF no puede *aprender de sus errores*. Si todos los árboles fallan en ciertos casos, el promedio también falla.



# Agenda

- 1 Roadmap y Recap
- 2 **Extra-Trees: Aleatorización Extrema**
- 3 AdaBoost: El Origen del Boosting
- 4 Gradient Boosting y XGBoost
- 5 LightGBM: Velocidad y Escala
- 6 Panorama Comparativo

# Extra-Trees (*Extremely Randomized Trees*)

**Geurts, Ernst & Wehenkel (2006).** Una variante de RF que lleva la aleatorización al extremo.

## Random Forest

- Muestra bootstrap de datos
- Subconjunto aleatorio de features
- **Busca el mejor punto de corte** para cada feature seleccionada

## Extra-Trees

- **Usa todos los datos** (sin bootstrap)
- Subconjunto aleatorio de features
- **Punto de corte aleatorio** para cada feature seleccionada

¿Qué cambia en la práctica?

- **Más rápido:** No busca el split óptimo, lo elige al azar → menos cómputo por nodo
- **Más varianza por árbol, menos correlación entre árboles:** el promedio la compensa
- **Mejor en datos con mucho ruido:** los splits aleatorios son más robustos a outliers

En `scikit-learn`: `ExtraTreesClassifier` / `ExtraTreesRegressor`. Misma API que RF.



# RF vs. Extra-Trees: ¿Cuándo UsarCuál?

## Random Forest

Split óptimo

Bootstrap (con reemplazo)

Más lento por split

Menor sesgo por árbol

Árboles más correlacionados

## Extra-Trees

Split aleatorio

Datos completos (sin reemplazo)

Más rápido por split

Mayor sesgo por árbol

Árboles menos correlacionados

## Regla práctica para consultoría

En la mayoría de problemas tabulares, RF y Extra-Trees dan resultados *muy similares*.

Extra-Trees es útil cuando: (1) el dataset es grande y se necesita velocidad, o (2) los datos tienen mucho ruido y queremos splits menos “obsesivos” con patrones espurios.

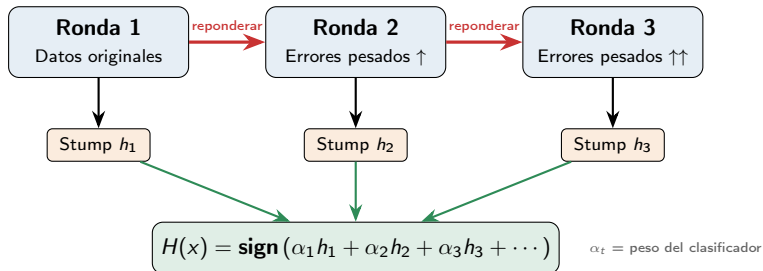
# Agenda

- 1 Roadmap y Recap
- 2 Extra-Trees: Aleatorización Extrema
- 3 AdaBoost: El Origen del Boosting**
- 4 Gradient Boosting y XGBoost
- 5 LightGBM: Velocidad y Escala
- 6 Panorama Comparativo

# AdaBoost: La Idea Fundacional

**Freund & Schapire (1997)** — uno de los papers más influyentes en ML.

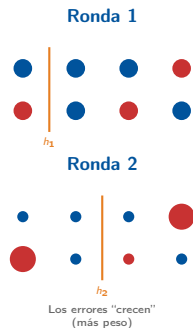
**Pregunta clave:** ¿Puedo combinar muchos clasificadores *apenas mejores que el azar* para obtener uno excelente?



# AdaBoost: ¿Cómo Funciona?

## Algoritmo intuitivo (3 pasos por ronda):

1. **Entrenar** un clasificador débil  $h_t$  (típicamente un *stump*: árbol de profundidad 1) con los datos ponderados.
2. **Calcular su peso**  $\alpha_t$ : los clasificadores que aciertan más reciben más voto. Si  $h_t$  es apenas mejor que el azar,  $\alpha_t$  es pequeño. Si es muy bueno,  $\alpha_t$  es grande.
3. **Reponderar** las observaciones: las que  $h_t$  clasificó mal reciben *más peso* en la siguiente ronda. Las bien clasificadas, menos.



Después de  $T$  rondas, la predicción es el voto ponderado:

$$H(x) = \text{sign} \left( \sum_{t=1}^T \alpha_t h_t(x) \right)$$

# AdaBoost: Intuición y Limitaciones

## Fortalezas

- Elegante y teóricamente fundamentado
- Funciona bien con clasificadores muy simples (*stumps*)
- No requiere tunear muchos hiperparámetros
- Resistente al overfitting *en ciertos contextos*

## Limitaciones

- **Sensible al ruido:** si una observación es outlier, AdaBoost la repondera cada vez más
- Solo optimiza error exponencial (no se adapta fácilmente a otras loss)
- En la práctica, XGBoost y LightGBM lo superan consistentemente

## Analogía para el cliente

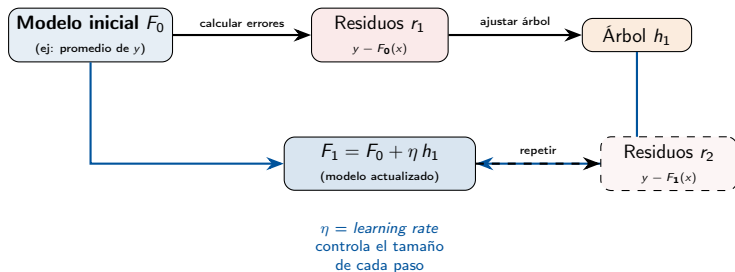
Piense en un equipo que revisa solicitudes de crédito. Después de cada ronda de revisión, los casos que *se clasificaron mal* reciben un **sticker rojo** para que el siguiente revisor les ponga más atención. Al final, las decisiones difíciles recibieron la mayor cantidad de atención del equipo.

# Agenda

- 1 Roadmap y Recap
- 2 Extra-Trees: Aleatorización Extrema
- 3 AdaBoost: El Origen del Boosting
- 4 Gradient Boosting y XGBoost**
- 5 LightGBM: Velocidad y Escala
- 6 Panorama Comparativo

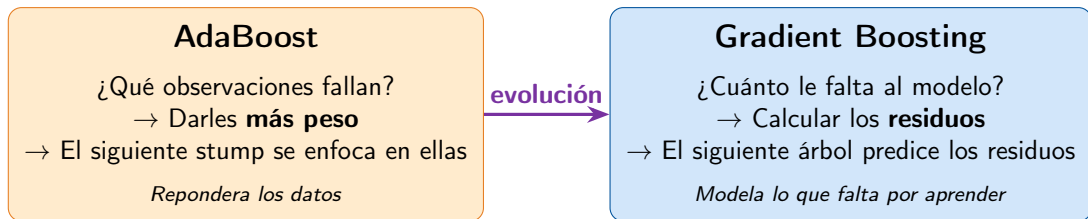
# El Cambio de Paradigma: Aprender de los Residuos

Friedman (2001) generalizó la idea de boosting con un marco poderoso.



**Idea central:** En cada ronda, no reponderamos observaciones (como AdaBoost), sino que entrenamos un **nuevo árbol sobre los errores residuales** del modelo acumulado.

# AdaBoost vs. Gradient Boosting: El Cambio Conceptual



¿Por qué “Gradient”? Porque los residuos son exactamente el **gradiente negativo** de la función de pérdida MSE. Friedman mostró que esto se generaliza: puedes usar *cualquier* función de pérdida diferenciable, y en cada paso te mueves en la dirección del gradiente.

Esto es descenso de gradiente, pero en el *espacio de funciones*, no de parámetros.



# XGBoost: El Gradient Boosting Industrializado

**Chen & Guestrin (2016)** — ganó virtualmente todas las competencias de Kaggle entre 2015–2020.

¿Qué añade XGBoost sobre Gradient Boosting clásico?

## Regularización integrada:

- Penaliza la complejidad del árbol directamente en la función objetivo
- Parámetros  $\lambda$  (L2 en pesos de hojas) y  $\alpha$  (L1, como Lasso)
- Equivalente a la intuición de Ridge/Lasso pero para árboles

## Column subsampling:

- Como RF: en cada árbol o split, usa un subconjunto de features
- Reduce correlación entre árboles y overfitting

## Eficiencia computacional:

- Paralelización de la búsqueda de splits
- Manejo nativo de missing values
- Compresión y caching de datos

## Row subsampling:

- Cada árbol usa solo una fracción de las filas
- Introduce aleatorización tipo bagging dentro del boosting

**XGBoost = Gradient Boosting + regularización + ingeniería computacional**

# XGBoost: Los Hiperparámetros Clave

Hiperparámetro	Default	Efecto e intuición
— Del Boosting —		
n_estimators	100	Nº de rondas (árboles secuenciales). Más → más capacidad, riesgo de overfit.
learning_rate ( $\eta$ )	0.3	Tamaño del paso. Más bajo = aprende más lento, necesita más árboles.
subsample	1.0	Fracción de filas por árbol. $< 1$ introduce aleatorización.
colsample_bytree	1.0	Fracción de columnas por árbol (estilo RF).
— De cada árbol —		
max_depth	6	Profundidad máxima. En boosting se usan árboles <b>pequeños</b> (3–8).
min_child_weight	1	Mín. suma de pesos en una hoja. Regulariza.
gamma ( $\gamma$ )	0	Mín. reducción de loss para hacer un split.
— Regularización —		
reg_lambda ( $\lambda$ )	1	Penalización L2 sobre pesos de hojas.
reg_alpha ( $\alpha$ )	0	Penalización L1 sobre pesos de hojas.

# XGBoost: Guía Práctica de Tuning

## Receta básica (80% de los casos):

- 1 Fijar `learning_rate` = 0.1
- 2 Tunear `n_estimators` con *early stopping* (el modelo para cuando el error de validación deja de bajar)
- 3 Tunear `max_depth`: probar 3, 5, 7
- 4 Tunear `subsample` y `colsample_bytree`: probar 0.7, 0.8, 0.9
- 5 Ajustar regularización si hay overfit

**Early stopping:** dejar de entrenar cuando el error en validación no mejora por  $N$  rondas.

## Trade-off fundamental

`learning_rate` y `n_estimators` están inversamente relacionados:

- $\eta$  bajo + muchos árboles = mejor generalización
- $\eta$  alto + pocos árboles = rápido pero menos preciso

**Regla:** Reducir  $\eta$  y usar `early_stopping` para encontrar el  $T$  óptimo.

## Error común

Tunear `n_estimators` sin `early_stopping` es como elegir la penalización de Lasso sin CV: no sabes cuándo parar.

# ¿Por Qué Árboles Pequeños en Boosting?

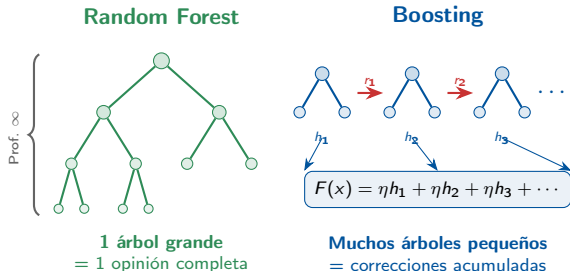
## En Random Forest:

- Cada árbol debe ser lo más expresivo posible
- Profundidad grande (ilimitada por default)
- La varianza se reduce por promedio

## En Boosting:

- Cada árbol solo captura una *pequeña corrección*
- Profundidad típica: 3–8
- La expresividad viene de **acumular** muchos árboles simples

**Analogía:** RF = comité de expertos con opiniones completas. Boosting = un analista que revisa su trabajo y en cada paso corrige lo que falló.



# Agenda

- 1 Roadmap y Recap
- 2 Extra-Trees: Aleatorización Extrema
- 3 AdaBoost: El Origen del Boosting
- 4 Gradient Boosting y XGBoost
- 5 LightGBM: Velocidad y Escala**
- 6 Panorama Comparativo

# LightGBM: ¿Qué Problema Resuelve?

Ke et al. (2017), Microsoft Research.

XGBoost funciona excelente, pero con datasets muy grandes ( $>100K$  filas,  $>100$  features) se vuelve lento. LightGBM introduce **dos innovaciones** que lo hacen mucho más rápido sin sacrificar (y a veces mejorando) el desempeño.

## 1. Histogram-based splits

En lugar de evaluar *cada* valor posible como punto de corte, agrupa los valores continuos en **bins** (histogramas, típicamente 255).

**Efecto:** Reduce la complejidad por nodo de  $O(N \cdot p)$  a  $O(\text{bins} \cdot p)$ .

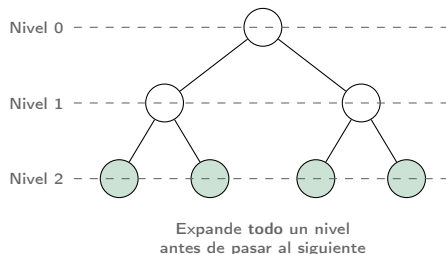
## 2. Leaf-wise growth

En lugar de crecer el árbol **nivel por nivel** (como XGBoost), LightGBM crece la hoja con **mayor reducción de pérdida**.

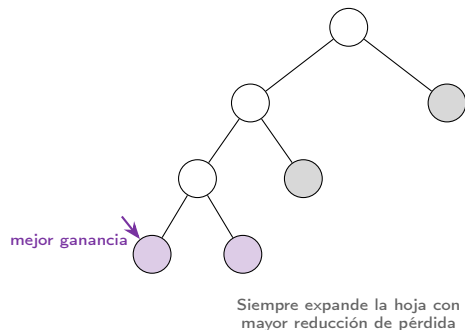
**Efecto:** Árboles asimétricos que alcanzan menor error con menos hojas.

# Level-wise vs. Leaf-wise: La Diferencia Visual

## XGBoost: Level-wise



## LightGBM: Leaf-wise



## Cuidado con leaf-wise

Leaf-wise puede hacer overfit más fácilmente en datasets pequeños (<10K obs) porque genera árboles muy profundos en una dirección. Por eso LightGBM necesita `num_leaves` como control.

# LightGBM: Hiperparámetros Distintivos

Además de los parámetros de boosting (similares a XGBoost), LightGBM introduce:

Hiperparámetro	Default	Efecto e intuición
num_leaves	31	Máximo de hojas por árbol. <b>Reemplaza a max_depth</b> como control principal. Más hojas = más capacidad.
max_bin	255	N° de bins para histogramas. Más = más preciso pero más lento.
min_data_in_leaf	20	Mín. observaciones por hoja. <b>Regularizador clave</b> (más alto = menos overfit).
feature_fraction	1.0	Equivalente a colsample_bytree en XGBoost.
bagging_fraction	1.0	Equivalente a subsample en XGBoost.
lambda_l1 / lambda_l2	0.0	Regularización L1/L2 (como en XGBoost).

**Regla práctica:**  $\text{num\_leaves} \leq 2^{\text{max\_depth}}$ . Si usas  $\text{max\_depth}=7$ , entonces  $\text{num\_leaves} \leq 128$ .



# XGBoost vs. LightGBM: ¿Cuándo UsarCuál?

## XGBoost

Level-wise (balanceado)

Exact split search

Más lento, más preciso por árbol

Robusto en datasets pequeños

Comunidad más grande

## LightGBM

Leaf-wise (asimétrico)

Histogram-based splits

Más rápido, escala mejor

Puede hacer overfit si  $N$  pequeño

Nativo para categóricas

## En la práctica de consultoría

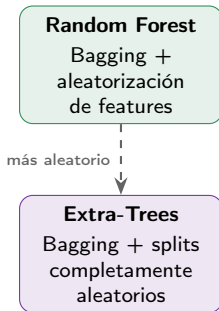
- Ambos dan resultados *muy similares* en la mayoría de problemas tabulares
- LightGBM es preferido cuando hay muchas filas ( $>100K$ ) o muchas categóricas
- XGBoost es el “safe default” con más documentación y comunidad
- **Lo que importa más:** buen tuning de hiperparámetros y feature engineering

# Agenda

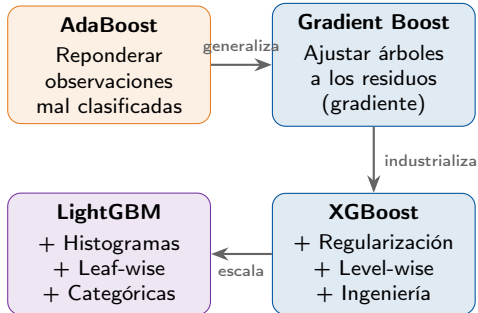
- 1 Roadmap y Recap
- 2 Extra-Trees: Aleatorización Extrema
- 3 AdaBoost: El Origen del Boosting
- 4 Gradient Boosting y XGBoost
- 5 LightGBM: Velocidad y Escala
- 6 Panorama Comparativo**

# El Mapa Completo de los Ensamblados

## Familia Bagging



## Familia Boosting



## Tabla Resumen: ¿Cuándo Usar Qué?

	Random Forest	Extra-Trees	XGBoost	LightGBM
<b>Paradigma</b>	Bagging	Bagging extremo	Boosting	Boosting
<b>Árboles</b>	Grandes, indep.	Grandes, indep.	Pequeños, se- cuenc.	Pequeños, se- cuenc.
<b>Velocidad</b>	Media	Rápido	Medio-lento	Rápido
<b>Overfit</b>	Difícil	Difícil	Posible	Posible (leaf- wise)
<b>Tuning</b>	Fácil	Fácil	Medio	Medio
<b>Mejor para</b>	Default robusto	Datos con ruido	Máx. precisión	Datasets grandes
<b>Catóricas</b>	Requiere encod- ing	Requiere encod- ing	Requiere encod- ing	Nativo

### El consenso en la industria (Grinsztajn et al., 2022)

Para **datos tabulares**, los modelos basados en árboles (especialmente XGBoost y LightGBM) siguen superando a las redes neuronales en la mayoría de benchmarks. Random Forest es un excelente baseline. La elección entre XGBoost y LightGBM rara vez cambia el resultado final.