Extreme Gradient Boosting

Ahmed Puco (00320082)
Martina Vásconez (00324077)
Ma. Eulalia Moncayo (00326226)
Joel Cuascota (00327494)

1. Resumen ejecutivo

XGBoost es un algoritmo de alto rendimiento basado en Gradient Boosting que construye modelos aditivos mediante la suma secuencial de árboles de decisión. Se recomienda para problemas de regresión y clasificación con datos tabulares donde se requiere alta precisión y manejo de interacciones no lineales. XGBoost combina descenso del gradiente (primer y segundo orden), regularización estructural y optimizaciones computacionales para obtener alto desempeño en benchmarks y producción. Promete mayor exactitud frente a modelos lineales o árboles aislados, a costa de mayor costo computacional y necesidad de ajuste de hiperparámetros; su interpretabilidad suele requerir herramientas como SHAP.

2. Formulación matemática

2.1 Función objetivo

La función objetivo general combina la pérdida predictiva con un término de regularización:

$L = \Sigma_i \ell(y_i, \hat{y}_i) + \Sigma\_t\ \Omega(f\_t)$

Según la tarea se usan distintas pérdidas: MSE para regresión cuadrática, MAE/Huber cuando hay outliers, y pérdida logística para clasificación binaria. Ω(f_t) penaliza la complejidad del árbol (número de hojas, magnitud de pesos).

2.2 Aditividad explícita en modelos de boosting

Los modelos boosting construyen una expansión aditiva de la forma:

$$\hat{y}^{(t)}(x) = \hat{y}^{(t-1)}(x) + v\, f\_t(x)$$

donde v es el learning rate y f_t(x) es el árbol ajustado en la iteración t. Esta representación hace explícito cómo cada árbol corrige los errores del ensamble previo.

2.3 Aproximación de segundo orden en XGBoost

XGBoost optimiza una expansión de Taylor de segundo orden de la pérdida:

$$\ell(y_i, \hat{y}_i + f(x_i)) \approx g_i\, f(x_i) + \tfrac{1}{2}\, h_i\, f(x_i)^2$$

donde g_i es el gradiente y h_i el hessiano de la pérdida respecto a $\hat{y}_i$.

Esto permite que cada árbol se ajuste utilizando información de curvatura, logrando convergencia más estable y mejor ajuste.

2.4 Mecanismo de aditividad y comparación entre AdaBoost y GBDT/XGBoost/LGBM/CatBoost

El principio de boosting agrega iterativamente modelos débiles para mejorar la predicción acumulada. Existen enfoques fundamentales diferentes según el algoritmo:

AdaBoost

• Peso a ejemplos, ya que AdaBoost mantiene un conjunto de pesos sobre las observaciones que refleja su dificultad de clasificación. Tras cada iteración, carga mayor peso sobre los ejemplos mal clasificados para forzar al siguiente clasificador a enfocarse en ellos.

• Actualización exponencial: La regla de actualización incrementa multiplicativamente el peso de ejemplos erróneos y reduce el de los correctos, esto equivale a optimizar una pérdida exponencial sobre las predicciones. En la práctica, AdaBoost construye un clasificador fuerte a partir de varios débiles priorizando los

errores difíciles. Debido a la pérdida exponencial, AdaBoost es muy sensible a outliers y ruido.

GBDT / XGBoost / LightGBM / CatBoost

• Descenso del gradiente en espacio de funciones: estos métodos no reponderan ejemplos; en su lugar, cada nuevo base-learner se ajusta para aproximar el gradiente negativo (pseudo-residuales) de la función de pérdida respecto a las predicciones actuales. Así se realiza un paso de descenso en el espacio de funciones.

• Árboles como base learners: Los modelos usan árboles de decisión que particionan el espacio y devuelven valores en hojas; cada árbol intenta corregir los errores locales detectados por el gradiente.

• Shrinkage (learning rate): Se escala la contribución de cada árbol con una tasa de aprendizaje (v), lo que reduce el riesgo de que un único árbol domine la solución y mejora la generalización al requerir más iteraciones para converger.

• Regularización (L1 / L2 y estructural): Se incorporan términos que penalizan la complejidad del árbol (por ejemplo número de hojas) y la magnitud de los valores en hojas (penalización L2 λ y opcional L1 α). Esto actúa como poda suave (elimina ramas u hojas) y evita pesos de hoja extremos.

• Submuestreo fila/columna: Técnicas de submuestreo (subsample, colsample_bytree/level/node) reducen la correlación entre árboles y la varianza del ensamble, mejorando robustez y velocidad.

• Parámetros de control de complejidad: min_child_weight / min_data_in_leaf / num_leaves: Estos parámetros previenen divisiones con pocos datos y controlan la granularidad del árbol. En XGBoost, min_child_weight evita hojas con suma de hessianos pequeña; en LightGBM, num_leaves limita el tamaño en crecimiento leaf-wise.

• Diferencias de diseño entre implementaciones: LightGBM usa crecimiento leaf-wise (más eficiente pero puede sobreajustar si no se limita num_leaves); CatBoost implementa ordered boosting y manejo nativo de categóricas para evitar sesgos en target encoding; XGBoost enfatiza estabilidad y regularización explícita

mediante términos de penalización y uso de hessianos para aproximación de segundo orden.

3. Algoritmo (alto nivel)

Pseudoflujo de entrenamiento (alto nivel):

1. Inicializar predicción base.
2. Para t = 1..T:
   a) Calcular gradientes (y hessianos si aplica) de la pérdida para cada observación.
   b) Ajustar un árbol para predecir esos pseudo-residuales.
   c) Escalar la salida del árbol por la tasa de aprendizaje (shrinkage).
   d) Actualizar la predicción acumulada sumando el árbol escalado.
3. Devolver el modelo final como la suma de todos los árboles.

Cada árbol se entrena sobre los residuos no explicados por el ensamble previo; el uso de shrinkage y regularización evita sobreajuste y permite control fino del proceso.

4. Hiperparámetros clave y sus efectos

Parámetros que controlan comportamiento y riesgo del modelo:

- **n_estimators (número de árboles):** Controla la capacidad del ensamble. Mucho bajo → subajuste; muy alto → riesgo de overfitting si no hay early stopping o regularización.
- **learning_rate (shrinkage):** Reduce la contribución de cada árbol. Inversamente proporcional a n_estimators; valores típicos 0.01–0.3.
- **max_depth / num_leaves:** Limita la complejidad de cada árbol. Profundidades mayores capturan interacciones complejas pero aumentan varianza. En LightGBM num_leaves controla estructura leaf-wise.
- **min_child_weight / min_data_in_leaf:** Evitan divisiones con pocas observaciones; aumentarlos reduce overfitting en datasets pequeños.
- **subsample:** Fracción de filas muestreadas por árbol; reduce correlación entre árboles y varianza (valores típicos 0.6–0.9).
- **colsample_bytree / colsample_bylevel:** Fracción de columnas muestreadas; reduce dependencia de features dominantes y acelera entrenamiento.

- **regulación L2 (λ) y L1 (α):** Penalizan magnitud de pesos en hojas. L2 suaviza valores, L1 promueve sparsity; ayudan a controlar predicciones extremas.

- **parámetros específicos:** Leaf-wise (LightGBM): crecimiento por hoja para eficiencia; Ordered boosting (CatBoost): evita sesgos en encoding de categóricas.

5. Ventajas y limitaciones

5.1 Ventajas

• Alta precisión en datos tabulares y capacidad para modelar interacciones no lineales.
• Manejo implícito de interacciones y robustez frente a feature scaling.
• Implementaciones maduras (XGBoost, LightGBM, CatBoost) con optimizaciones para producción.

5.2 Limitaciones

• Alto costo computacional y consumo de memoria, especialmente con muchos árboles y datos grandes.
• Sensible a la tasa de aprendizaje; requiere ajuste cuidadoso de hiperparámetros.
• Riesgo de overfitting en datasets pequeños o ruidosos si no se regula adecuadamente.
• Manejo de variables categóricas requiere cuidado; errores en encoding pueden provocar data/target leakage.

6. Buenas prácticas

Recomendaciones prácticas:
• Balance learning_rate vs n_estimators: usar learning_rate bajo y más árboles para mejor generalización.
• Early stopping con conjunto de validación para evitar sobreentrenamiento.
• Evitar target encoding sin validación temporal/stratificada; usar ordered encoding o CatBoost cuando haya muchas categóricas.
• Tratar outliers según la pérdida: MAE/Huber si hay valores extremos; MSE si se quiere penalizar grandes errores.
• Monitoreo de data drift y performance en producción; re-entrenar según deriva.
• Interpretabilidad: usar importancias de features y SHAP para explicar predicciones y detectar sesgos.

7. Casos de uso y pitfalls frecuentes

Casos de uso habituales:
• Scoring crediticio, detección de fraude, sistemas de recomendación, predicción de demanda, pricing dinámico.

Pitfalls frecuentes:
• Data leakage: incluir variables calculadas con información futura o del conjunto de test.
• Target leakage en target encoding: calcular estadísticas del target usando datos que incluyen al ejemplo en cuestión.
• Falta de validación temporal en series: usar validación adecuada (walk-forward) para problemas temporales.
• No monitorear drift o cambios en la distribución de features/target tras despliegue.

8. Checklist de tuning

Orden práctico para ajuste y rangos razonables:

1. **Fijar learning_rate bajo:** 0.01–0.1; luego usar early stopping para determinar n_estimators.

2. **Ajustar max_depth / num_leaves:** Profundidad 4–8 o num_leaves 31–127 según algoritmo y tamaño de datos.

3. **Determinar n_estimators con early stopping:** Partir de un límite alto (500–5000) y reducir según validación.

4. **Probar subsample y colsample_bytree:** Valores típicos 0.6–0.9 para reducir varianza.

5. **Ajustar min_child_weight / min_data_in_leaf:** Aumentar si hay sobreajuste; típicos 1–20 según tamaño de dataset.

6. **Añadir penalizaciones L2/L1 si persiste overfitting:** λ (L2) suele ayudar; α (L1) para sparsity.

7. **Validación final y pruebas de robustez:** Backtesting temporal, pruebas de estabilidad y análisis SHAP.

Bibliografías:

Friedman, J. H. (2001). *Greedy function approximation: A gradient boosting machine*. Annals of Statistics, 29(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

Friedman, J. H. (1999). *Stochastic gradient boosting*. Technical Report, Stanford University. https://statweb.stanford.edu/~jhf/ftp/stobst.pdf

Freund, Y., & Schapire, R. (1997). *A decision-theoretic generalization of on-line learning and an application to boosting*. Journal of Computer and System Sciences, 55(1), 119–139. https://doi.org/10.1006/jcss.1997.1504

Schapire, R. E., & Freund, Y. (2012). *Boosting: Foundations and algorithms*. MIT Press.

Chen, T., & Guestrin, C. (2016). *XGBoost: A scalable tree boosting system*. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794. https://doi.org/10.1145/2939672.2939785

XGBoost Developers. (2024). *XGBoost documentation*. https://xgboost.readthedocs.io/

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). *CatBoost: Unbiased boosting with categorical features*. Advances in Neural Information Processing Systems, 31. https://papers.nips.cc/paper/2018/hash/14491b756b3a51c3c0c07e8a5ba9e3b0-Abstract.html.