Gradient Boosting: Informe Técnico

Andrés Bohórquez (00320727)
Paulo Cantos (00326682)
John Ochoa (00345743)
Daniela Salazar (00329368)
Gian Tituaña (00325991)

Universidad San Francisco de Quito
Curso: Data Mining 3714
Profesor: Erick Ñauñay
Facultad de Ciencias e Ingenierías
Ciencias de la Computación
Quito
2025

Tabla de Contenido

Lista de figuras

1. Resumen

Gradient Boosting se ha consolidado como uno de los enfoques más efectivos para problemas de regresión y clasificación. Su fortaleza radica en la combinación aditiva de modelos débiles mediante el descenso de gradiente en el espacio de funciones. En este informe se describen su formulación matemática, el algoritmo general, los hiperparámetros más relevantes, sus ventajas y limitaciones, y un conjunto de buenas prácticas para una implementación rigurosa.

2. Formulación Matemática

Sea un conjunto de datos $\{(x_i, y_i)\}_{i=1}^{N}$ y una función de pérdida $L(y, F(x))$. El objetivo del boosting es encontrar la función

$$F^*(x) = \arg\min_F \sum_{i=1}^{N} L(y_i, F(x_i)). \tag{2.1}$$

El modelo se construye de manera aditiva:

$$F_M(x) = F_0(x) + \sum_{m=1}^{M} \nu\, h_m(x), \tag{2.2}$$

donde $h_m(x)$ es un árbol de decisión y ν es el *learning rate*. El modelo inicial $F_0(x)$ suele ser la predicción constante que minimiza la pérdida.

En cada iteración, el algoritmo calcula los pseudo–residuales:

$$r_{im} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}, \tag{2.3}$$

siguiendo la formulación de Friedman (2001). Un nuevo árbol h_m se ajusta para aproximar estos valores, y sus hojas son optimizadas mediante un ajuste lineal que minimiza la pérdida en cada región terminal.

En contraste con AdaBoost, que actualiza de forma explícita pesos w_i sobre cada observación en función de si fue bien o mal predicha y puede interpretarse como la minimización de una *exponential loss*, el Gradient Boosting clásico realiza descenso de gradiente en el espacio de funciones. Cada árbol h_m se ajusta sobre los pseudo–residuales y su contribución se escala mediante el *learning rate* ν (*shrinkage*), mientras que la complejidad del ensamble se controla mediante regularización L1/L2, profundidad máxima y submuestreo de filas y columnas.

2.1. Funciones de Pérdida en Regresión

2.1.1. Pérdida Cuadrática Media (MSE)

$$L_{\text{MSE}}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2. \tag{2.4}$$

Su derivada produce residuos tradicionales $y - \hat{y}$, lo que convierte esta variante en una extensión natural de mínimos cuadrados.

2.1.2. Pérdida Absoluta Media (MAE)

$$L_{\text{MAE}}(y, \hat{y}) = |y - \hat{y}|. \tag{2.5}$$

Es más robusta ante valores atípicos, pero es menos suave y presenta gradientes menos informativos cuando el error es pequeño.

2.1.3. Pérdida de Huber

La pérdida de Huber (1964) combina MSE y MAE:

$$L_{\text{Huber}}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| \leq \delta, \\ \delta |y - \hat{y}| - \frac{1}{2}\delta^2, & |y - \hat{y}| > \delta. \end{cases}$$

Es útil cuando se desea suavidad en errores pequeños y robustez ante outliers.

3. Algoritmo

El procedimiento de Gradient Boosting se basa en la idea de ajustar secuencialmente modelos débiles para aproximar el gradiente negativo de la función de pérdida. Como presentan Friedman (2001), el algoritmo implementa un descenso de gradiente en el espacio de funciones, y cada árbol funciona como una corrección parcial del modelo anterior. Cuando se incorpora submuestreo, como proponen Friedman (2002), se obtiene una variante estocástica que mejora la robustez y reduce la varianza.

A continuación se presenta el pseudocódigo detallado del algoritmo, siguiendo el flujo general: inicializar modelo, calcular pseudo–residuales, ajustar árbol, optimizar valores de hojas, actualizar el ensamble y repetir hasta completar las iteraciones especificadas.

```
Input: (x_i, y_i)_{i=1}^N, función de pérdida L,
       número de árboles M, learning_rate nu,
       parámetros del árbol tree_params,
       subsample  (0,1] para Stochastic GB.

1. Inicializar F_0(x):
      F_0(x) = argmin_c _{i=1}^N L(y_i, c)
   (Para MSE: F_0 = promedio de y).

2. Para m = 1 .. M:

   2.1. Seleccionar datos de la iteración:
        si subsample < 1:
            S_m ← muestra aleatoria de tamaño N * subsample
        si no:
            S_m ← todos los índices {1..N}

   2.2. Calcular pseudo-residuales (gradiente negativo):
        r_{im} = - L(y_i, F_{m-1}(x_i)) / F(x_i)
        # Para MSE: r_{im} = y_i - F_{m-1}(x_i)
        # Para MAE: r_{im} = sign(y_i - F)
        # Para Huber: usar fórmula piecewise según Huber (1964).

   2.3. Ajustar árbol base:
        h_m(x) = TreeFit({(x_i, r_{im}) : i  S_m}, tree_params)
```

```
        # tree_params controla max_depth, max_leaf_nodes,
        # min_samples_leaf, column subsampling, etc.

    2.4. Para cada hoja j del árbol:
         Definir región R_{jm}.
         Optimizar constante de la hoja:
             _{jm} = argmin_ _{i: x_i  R_{jm}}
                           L(y_i, F_{m-1}(x_i) + )
         # Para MSE: _{jm} = promedio de residuos r_{im}
         # Para pérdidas no cuadráticas: búsqueda por línea.

    2.5. Definir función del árbol con valores optimizados:
         h_m^{leaf}(x) = _j _{jm} * 1_{x  R_{jm}}

    2.6. Actualizar el modelo (shrinkage):
         F_m(x) = F_{m-1}(x) + nu * h_m^{leaf}(x)

    2.7. (Opcional) Early Stopping:
         Evaluar pérdida en validación;
         si no mejora durante T iteraciones → detener.

3. Devolver modelo final F_M(x),
   o F_* en caso de early stopping.
```

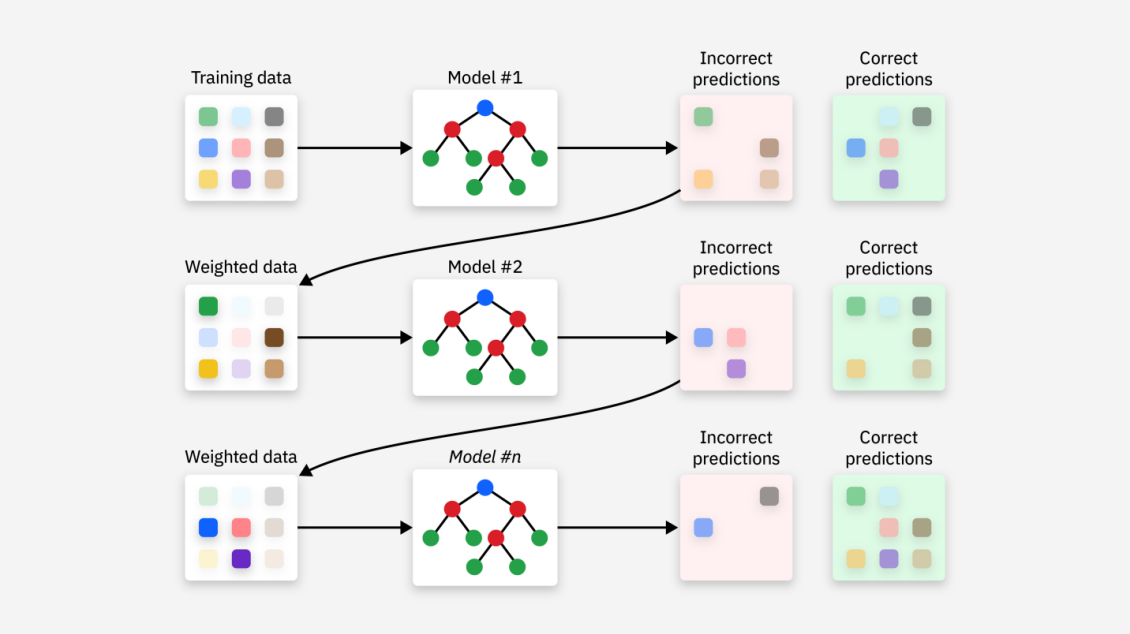

Figura 3.1

Flujo general del algoritmo Gradient Boosting. Adaptado de IBM (2024).

4. Hiperparámetros Clave y Efectos

Los hiperparámetros determinan la capacidad del ensamble:

4.0.1. Número de Árboles (M)

Un mayor número incrementa la capacidad, pero también el riesgo de sobreajuste. Un aprendizaje lento (ν pequeño) suele requerir más iteraciones.

4.0.2. Learning Rate (ν)

Su efecto regularizador fue analizado en Friedman (2001). Valores pequeños mejoran la generalización pero incrementan el costo computacional.

4.0.3. Profundidad del Árbol y Hojas

Los árboles suelen tener profundidades entre 2 y 5, lo que permite capturar interacciones sin caer en sobreajuste extremo. En implementaciones modernas se controla no solo la profundidad máxima (`max_depth`), sino también el número de hojas (`max_leaf_nodes` o `num_leaves` en LightGBM), lo que limita directamente la complejidad estructural de cada árbol.

4.0.4. Submuestreo de Filas

Friedman (2002) demuestra que usar un porcentaje del dataset reduce varianza y acelera el entrenamiento.

4.0.5. Submuestreo de Columnas

Limita el número de características en cada división, actuando como regularización adicional.

4.0.6. Regularización L1/L2

Penaliza magnitudes extremas en hojas, estabilizando el modelo y reduciendo sensibilidad al ruido.

En variantes como XGBoost, parámetros adicionales como `min_child_weight` exigen una masa mínima de datos efectiva en cada hoja, evitando particiones excesivamente específicas. LightGBM utiliza crecimiento *leaf-wise* y controla la complejidad mediante `num_leaves` y

`min_data_in_leaf`, mientras que CatBoost incorpora técnicas como *ordered boosting* para reducir el sesgo por el ordenamiento de los datos al manejar variables categóricas y mitigar el riesgo de *target leakage*.

5. Ventajas y Limitaciones

5.0.1. Ventajas

Según *Data Science Wizards* (s.f.), Gradient Boosting presenta:

- Alta precisión predictiva.
- Flexibilidad en clasificación y regresión.
- Capacidad de manejar valores faltantes mediante *surrogate splits*.
- Robustez relativa ante valores atípicos gracias al ensamble.

5.0.2. Limitaciones

- Entrenamiento costoso computacionalmente.
- Riesgo de sobreajuste con árboles profundos o ν elevado.
- Menor interpretabilidad por su estructura de ensamble.
- Sensible al tratamiento de variables categóricas; una codificación inadecuada puede inducir *leakage* o sobreajuste.

6. Buenas Prácticas

- Usar *learning rate* pequeño y mayor número de árboles.
- Aplicar *early stopping* cuando sea posible.
- Manejar outliers antes del entrenamiento.
- Controlar complejidad mediante profundidad máxima y regularización.
- Revisar derivaciones de importancia (SHAP, ganancia, frecuencia).
- Tratar cuidadosamente las variables categóricas, aprovechando manejo nativo (CatBoost) o codificaciones como *one-hot* o *target encoding* con control de *leakage*.
- Cuando el dominio lo exige, considerar *monotonic constraints* para imponer relaciones monotónicas entre ciertas variables explicativas y la predicción.

7. Casos de Uso y Pitfalls Frecuentes

Gradient Boosting resulta especialmente útil en problemas tabulares con relaciones no lineales complejas, como predicción de riesgo crediticio, estimación de demanda o fijación dinámica de precios. En estos contextos, su capacidad para combinar muchas variables heterogéneas y capturar interacciones lo hace competitivo frente a modelos lineales clásicos.

No obstante, existen varios *pitfalls* recurrentes que deben evitarse:

- **Data leakage**: utilizar información futura o variables derivadas del *target* en el entrenamiento (por ejemplo, estadísticas calculadas con toda la serie temporal) produce evaluaciones irreales del error y modelos no trasladables a producción.

- **Target leakage en *target encoding***: al codificar variables categóricas con el promedio del *target*, es crucial usar esquemas de validación cruzada o codificación ordenada para evitar que una observación "vea" su propio valor objetivo durante el entrenamiento.

- **Data drift**: cambios en la distribución de las variables o del *target* entre entrenamiento y despliegue degradan el desempeño; se recomienda monitorear métricas de drift y recalibrar los modelos de forma periódica.

8. Checklist de Tuning

En la práctica, una estrategia ordenada de ajuste de hiperparámetros puede seguir los siguientes pasos:

1. Fijar una función de pérdida adecuada (MSE, MAE o Huber) y un esquema de validación robusto (por ejemplo, validación temporal o *TimeSeriesSplit*).

2. Comenzar con un número moderado de árboles (M) y explorar una escala de *learning rate* ν (por ejemplo, 0,1, 0,05, 0,01), manteniendo árboles poco profundos.

3. Ajustar la profundidad máxima y/o número de hojas (`max_depth`, `max_leaf_nodes`, `num_leaves`) para equilibrar sesgo y varianza.

4. Introducir submuestreo de filas y columnas (`subsample`, `colsample_bytree` o `feature_fraction`) para reducir sobreajuste y acelerar el entrenamiento.

5. Refinar parámetros de regularización L1/L2 (`alpha`, `lambda`) y restricciones estructurales (`min_child_weight`, `min_data_in_leaf`), observando su impacto en el desempeño y en la estabilidad del modelo.

6. Finalmente, realizar una búsqueda más fina (*grid search* o *randomized search*) alrededor de las combinaciones prometedoras, monitorizando siempre el desempeño en validación y los tiempos de entrenamiento e inferencia.

Referencias

Data Science Wizards. (s.f.). *Understanding the gradient boosting algorithm.* https://medium.com/@datasciencewizards/understanding-the-gradient-boosting-algorithm-9fe698a352ad. (Recuperado de Medium)

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*(5), 1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*(4), 367–378.

GeeksforGeeks. (2024). *Gradient boosting algorithm in machine learning.* https://www.geeksforgeeks.org/. (Recuperado de GeeksforGeeks)

Huber, P. J. (1964). Robust estimation of a location parameter. *Annals of Mathematical Statistics*, *35*(1), 73–101.

IBM. (2024). *Gradient boosting.* https://www.ibm.com/think/topics/gradient-boosting. (Recuperado de IBM Think)