LightGBM: 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

Lista de tablas

1. Resumen

LightGBM (Light Gradient Boosting Machine) es un framework de boosting de gradiente basado en árboles de decisión, optimizado para alto rendimiento en datos de gran escala. Se emplea tanto en regresión como en clasificación, prometiendo alta precisión y eficiencia computacional. Sus innovaciones (crecimiento leaf-wise, histogramas de valores, muestreo GOSS y agrupamiento exclusivo de características) aceleran el entrenamiento sin sacrificar exactitud. Se puede usar en datasets tabulares de gran tamaño (millones de filas o alta dimensionalidad) donde los tiempos de entrenamiento de GBDT tradicionales o XGBoost son prohibitivos. . En general, LightGBM entrega modelos muy precisos que manejan no linealidades y heterogeneidad en los datos, especialmente útiles en conjuntos tabulares grandes.

2. Formulación matemática

La base matemática de **LightGBM** se fundamenta en el marco estándar de Gradient Boosting, pero con modificaciones clave en el proceso de construcción del árbol. Esto inicia con la función objetivo fundamental de Gradient Boosting que luego será optimizada (Brenndoerfer, 2025).

LightGBM se basa en el marco de Gradient Boosting de segundo orden, donde el modelo final es una suma aditiva de árboles:

$$F^{(t)}(x) = F^{(t-1)}(x) + f_t(x)$$

donde $F^{(t)}(x)$: predicción acumulada tras t árboles, $f_t(x)$: el árbol añadido en la iteración t y $t = 1,2,\ldots,T$: número de árboles.

La función objetivo del gradient boosting combina la función de costo y un término de regularización:

$$\mathscr{L}^{(t)} = \sum_{i=1}^{n} l(y_i, F^{(t-1)}(x_i) + f_t(x_i)) + \Omega(f_t)$$

donde $l(y_i, F^{(t-1)}(x_i) + f_t(x_i))$ es la función de costo, $f_t(x_i))$ es el arból t-ésimo que se está sumando y $\Omega(f_t)$ es el término de regularización para ese árbol.

En **LightGBM**, se usa una expansión de Taylor de segundo orden para aproximar este objetivo y hacerla optimizable:

$$\mathscr{L}^{(t)} \approx \sum_{i=1}^{n} \left[l(y_i, F^{(t-1)}(x_i)) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

que al eliminar el término constante $l(y_i, F^{(t-1)}(x_i))$ queda:

$$\mathscr{L}^{(t)} = \sum_{i=1}^{n} \left(g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right) + \Omega(f_t)$$

con

$$g_i = \left. \frac{\partial l(y_i, \hat{y})}{\partial \hat{y}} \right|_{\hat{y}=F^{(t-1)}(x_i)}$$
$$h_i = \left. \frac{\partial^2 l(y_i, \hat{y})}{\partial \hat{y}^2} \right|_{\hat{y}=F^{(t-1)}(x_i)}$$

Ahora bien, en **LightGBM** la representación del árbol y el valor óptimo por hoja se ve de la siguiente manera.

Sea I_j el conjunto de muestras que caen en la hoja j del árbol f_t. **LightGBM** asigna a

todos los puntos en esa hoja un valor constante:

$$f_t(x_i) = w_j \quad \text{si } x_i \in I_j$$

Sustituyendo en la función objetivo:

$$\mathscr{L}^{(t)} = \sum_{j=1}^{J} \left[\sum_{i \in I_j} w_j g_i + \frac{1}{2} w_j^2 \sum_{i \in I_j} h_i \right] + \lambda \sum_{j=1}^{J} w_j^2$$

Derivando respecto a w_j, el valor óptimo de cada hoja es:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

En el caso de **LightGBM**, cada hoja tiene su propio valor óptimo, y la actualización se realiza hoja por hoja **(leaf-wise)**. Esta enfoque **leaf-wise** hace uso del "split gain", a partir del cual busca busca entre todas las hojas y selecciona aquella cuyo split resulta en la máxima reducción de la función de costo (la mayor ganancia)(Shi, 2007) . Esta se calcula como:

$$\text{Gain} = \frac{1}{2} \left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I} g_i)^2}{\sum_{i \in I} h_i + \lambda} \right] - \gamma$$

donde g_i: gradiente de primer orden, h_i: gradiente de segundo orden, I: conjunto de todas las instancias en la hoja actual antes de la división, I_L: conjunto de instancias que irían a la hoja hija izquierda después de la división, I_R: conjunto de instancias que irían a la hoja hija derecha después de la división, λ: parámetro de regularización L2 (que penaliza pesos de hoja grandes) y γ: umbral de ganancia mínimo requerido para hacer una división (divisiones con ganancia menor a γ no se realizan).

Finalmente, la actualización del modelo en la iteración t es:

$$F^{(t)}(x) = F^{(t-1)}(x) + \eta f_t(x)$$

donde η es la *learning rate*.

Respecto al sampling, **LightGBM** utiliza técnicas de muestreo antes y durante el entrenamiento. Por un lado, en la etapa de preprocesamiento utiliza **EFB (exclusive feature bundling)** el cual reduce drásticamente el número de features aprovechando que, en datos reales muy dispersos, muchas características son “mutuamente exclusivas”, es decir, nunca toman valores distintos de cero simultáneamente (Ke y cols., 2017). Al agrupar muchas características esparcidas en unos pocos "paquetes"(bundles) densos, el costo de construir histogramas se reduce significativamente, pasando de ser $O(\#data \times \#feature)$ a $O(\#data \times \#bundle)$.

Por otro lado, durante el entrenamiento utiliza **GOSS (Muestreo Unilateral Basado en Gradientes)** que es una técnica de optimización diseñada para acelerar drásticamente el entrenamiento sin perder precisión.

Se basa en que las muestras con **gradientes grandes** (las que el modelo predice mal) son mucho más informativas que las muestras con **gradientes pequeños** (las que el modelo ya predice bien).

De esta manera, conserva todas las muestras con los gradientes más grandes (ejm 20 % superior) y muestrea aquellas que tienen los gradientes más pequeños (ejm 20 % del grupo restante). Para evitar sesgar el modelo, **GOSS** amplifica el peso de las muestras "fáciles"que seleccionó para que su contribución total a la hora de calcular la ganancia (Gain) sea estadísticamente similar a si se hubieran usado todas (Brenndoerfer, 2025). Así el árbol se construye usando un subconjunto de datos mucho más pequeño, lo que reduce enormemente el costo computacional.

3. Algoritmo

Entrada:

Datos de entrenamiento: $D = \{(\chi_1, y_1), (\chi_2, y_2), \ldots, (\chi_N, y_N)\}, \chi_i \in \mathscr{X}, y_i \in \mathbb{R}, y_i \in \{-1, +1\}$; función de pérdida: $L(y, \theta(\chi))$;

Interaciones:

M; Tasa de muestreo de datos con gradiente grande: a; Tasa de muestreo de datos con gradiente pequeño: b;

1. Combinar características que son mutuamente exclusivas (es decir, características que nunca aceptan valores distintos de cero simultáneamente) de $\chi_i, i = \{1, \ldots, N\}$ mediante la técnica de exclusive feature bundling (EFB);

2. Establecer $\theta_0(\chi) = \arg\min_{\mathscr{C}} \sum_{i=1}^{N} L(y_i, \mathscr{C})$;

3. Para $m = 1$ hasta M:

4. Calcular los valores absolutos del gradiente:

$$r_i = \left| \frac{\partial L(y_i, \theta(\chi_i))}{\partial \theta(\chi_i)} \right|_{\theta(\chi) = \theta_{m-1}(\chi)}, i = \{1, \ldots, N\}$$

5. Remuestrear el conjunto de datos usando el proceso de gradient-based one-side sampling (GOSS):

$\text{topN} = a \times \text{len}(D)$; $\text{randN} = b \times \text{len}(D)$;
$sorted = \text{GetSortedIndices}(\text{abs}(r))$;
$A = sorted[1 : \text{topN}]$; $B = \text{RandomPick}(sorted[\text{topN} : \text{len}(D)], \text{randN})$;
$D' = A + B$;

6. Calcular las ganancias de información:

$$V_j(d) = \frac{1}{n} \left(\frac{(\sum_{\chi_i \in A_L} r_i + \frac{1-a}{b} \sum_{\chi_i \in B_L} r_i)^2}{n_L^j(d)} + \frac{(\sum_{\chi_i \in A_R} r_i + \frac{1-a}{b} \sum_{\chi_i \in B_R} r_i)^2}{n_R^j(d)} \right)$$

7. Desarrollar un nuevo árbol de decisión $\theta_m(\chi)'$ sobre el conjunto D'

8. Actualizar $\theta_m(\chi) = \theta_{m-1}(\chi) + \theta_m(\chi)'$

9. End for.

10. Retornar $\hat{\theta}(\chi) = \theta_M(\chi)$

4. Hiperparámetros clave y efectos

LightGBM es una implementación de GBDT que introduce optimizaciones clave: GOSS (Gradient-based One-Side Sampling) y EFB (Exclusive Feature Bundling). Sus hiperparámetros están diseñados para controlar estas optimizaciones y la complejidad del modelo, enfocándose en el crecimiento por hoja (leaf-wise) .

Para **optimizar el uso de *leaf-wise***, algunos de los parámetros más importantes son: `num_leaves`, `max_depth` y `min_data_in_leaf`. El parámetro `num_leaves` controla la complejidad. Este valor debe ser menor que $2^{\texttt{max\_depth}}$ para evitar el sobreajuste, dado que los árboles *leaf-wise* tienden a ser más profundos (Microsoft, 2025) . Adicionalmente, `min_data_in_leaf`, que es el cantidad mínima de información en una hoja, es un parámetro crucial para prevenir el *overfitting*, ya que detiene el crecimiento en hojas con pocas muestras. Finalmente, `max_depth` puede usarse para limitar explícitamente la profundidad del árbol.

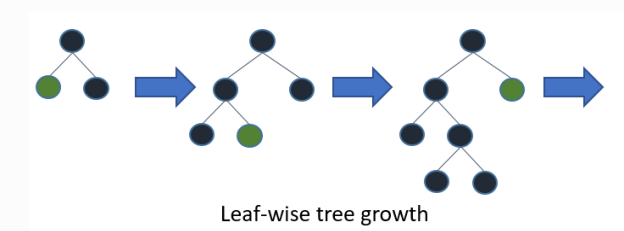

Figura 4.1

Crecimiento por hoja (leaf-wise)

Para **acelerar el entrenamiento** de LightGBM, se pueden aplicar varias estrategias. Es posible reducir la complejidad del modelo aumentando `min_gain_to_split` para ignorar divisiones con ganancias mínimas, o aumentando `min_data_in_leaf` y `min_sum_hessian_in_leaf` para evitar hojas con pocas muestras. Otra táctica es construir menos árboles, ya sea reduciendo `num_iterations` (ajustando `learning_rate` en consecuencia) o usando `early_stopping_round` con un set de validación. Se puede acelerar la evaluación de divisiones (splits) reduciendo `max_bin` (menos bins para features continuas), disminuyendo `feature_fraction` (muestreo de columnas) o `max_cat_threshold` (simplificar splits categóricos) (Microsoft, 2025). También, usar menos datos mediante *bagging* (activando `bagging_fraction` y `bagging_freq`) puede mejorar la velocidad . Para **mejorar la precisión**, se puede usar un `max_bin` grande, un `learning_rate` pequeño con muchas `num_iterations`, y un `num_leaves` elevado, además de más datos de entrenamiento o probar *dart*. Para **controlar el overfitting**, se recomienda usar `max_bin` y `num_leaves` pequeños, ajustar `min_data_in_leaf` y `min_sum_hessian_in_leaf`, emplear *bagging* y muestreo de columnas (`feature_fraction`) (Microsoft, 2025). También ayuda usar más datos, regularización (con `lambda_l1`, `lambda_l2`, `min_gain_to_split`), limitar la profundidad (`max_depth`), o probar *extra_trees*.

5. Ventajas y limitaciones

LightGBM presenta diversas ventajas, destacando su alta velocidad de entrenamiento gracias a su estrategia de crecimiento *leaf-wise* y al uso de algoritmos basados en histogramas. La discretización de características continuas en *bins* reduce el costo de búsqueda de umbrales óptimos y disminuye las operaciones por iteración (Brenndoerfer, 2025). Esto permite expandir siempre la hoja con mayor reducción de pérdida y alcanzar altos niveles de precisión con menos iteraciones. A ello se suma el mecanismo *Exclusive Feature Bundling* (EFB), que agrupa características mutuamente excluyentes y reduce la dimensionalidad efectiva, acelerando aún más el entrenamiento (Ke y cols., 2017). Estas técnicas posicionan a LightGBM como uno de los métodos de boosting más rápidos, especialmente en *datasets* grandes (Royal Research, 2025).

Además de su rapidez, LightGBM es altamente eficiente en el uso de memoria. La construcción de histogramas reduce el tamaño de las estructuras internas y permite procesar grandes volúmenes de datos sin hardware costoso (Shi, 2007). EFB potencia aún más esta eficiencia al disminuir el número de *bins* necesarios, lo cual facilita trabajar con conjuntos de alta dimensionalidad que serían inviables para otros algoritmos de boosting (TutorialsPoint, 2025).

La escalabilidad es otra fortaleza clave. LightGBM mantiene un rendimiento robusto incluso con millones de observaciones, ofreciendo tiempos de entrenamiento significativamente menores que frameworks como XGBoost (Ke y cols., 2017). Esto se debe a su crecimiento hoja por hoja, al aprendizaje basado en histogramas y a su compatibilidad con paralelización y GPU, que lo hacen adecuado para sistemas con recursos computacionales variados.

En términos de precisión, LightGBM suele superar a otros modelos gracias a su estrategia *leaf-wise*, que prioriza divisiones con mayor reducción de pérdida y genera árboles asimétricos capaces de corregir errores relevantes de manera eficiente (GeeksforGeeks, 2025b). La discretización mediante histogramas actúa además como una forma de regularización que favorece la generalización (Surana, 2020). LightGBM también ofrece soporte nativo para variables categóricas, evitando la necesidad de *one-hot encoding* (Budiawan, 2025).

No obstante, LightGBM presenta limitaciones. En *datasets* pequeños puede sobreajustarse debido al crecimiento profundo generado por su estrategia *leaf-wise*, por lo que es necesario ajustar cuidadosamente hiperparámetros como el número de hojas, la profundidad máxima y el mínimo de datos por hoja. Su interpretabilidad también es limitada, ya que el modelo consiste en numerosos árboles complejos; aunque herramientas como SHAP o las importancias de características ayudan, no ofrecen la transparencia de modelos más simples (Budiawan, 2025). Finalmente, LightGBM es sensible al *tuning*: parámetros como *learning rate*, *num_leaves*, profundidad, *subsampling* y regularización influyen directamente en su estabilidad y desempeño. Un ajuste incorrecto puede degradar la precisión o elevar el tiempo de entrenamiento.

6. Buenas prácticas

El uso adecuado de LightGBM requiere no solo ajustar hiperparámetros, sino también aplicar buenas prácticas que garanticen estabilidad, generalización y una correcta interpretación del modelo. Uno de los aspectos más importantes es la relación entre el *learning rate* y *n_estimators*. Como LightGBM es sensible a este parámetro, se recomienda emplear valores pequeños (0.01–0.1) junto con un número elevado de árboles, lo que produce un aprendizaje más estable y reduce el riesgo de converger en mínimos locales. Esto se puede complementar con *early stopping*, una técnica que detiene el entrenamiento cuando la métrica de validación deja de mejorar y que suele configurarse con 50 a 100 rondas de tolerancia. Su uso es esencial para prevenir el sobreajuste y optimizar el tiempo de cómputo (?, ?).

Otro punto clave es el manejo de variables categóricas. LightGBM ofrece soporte nativo para este tipo de características, por lo que no es necesario recurrir a *one-hot encoding*. Sin embargo, es fundamental identificarlas explícitamente y evitar cardinalidades excesivas en conjuntos pequeños, donde pueden introducir ruido y causar sobreajuste. En casos de alta cardinalidad, se recomienda agrupar categorías infrecuentes o emplear técnicas como *target encoding*, siempre aplicadas con validación cruzada para evitar filtraciones de información (Merckel, 2019).

Respecto a los valores atípicos, aunque LightGBM es relativamente robusto frente a *outliers* en las variables predictoras, estos pueden afectar las primeras divisiones del árbol o distorsionar tareas de regresión cuando aparecen en la variable objetivo. Por ello, es útil detectar y tratar *outliers* mediante winsorización, transformaciones logarítmicas o eliminando valores claramente erróneos. El uso de funciones de pérdida robustas o mayor regularización L1/L2 también ayuda a limitar su impacto.

En escenarios donde existe una relación monotónica conocida entre una variable y la salida —como en riesgo crediticio o decisiones regulatorias— pueden aplicarse *monotonic constraints*. Estas restricciones obligan al modelo a mantener relaciones crecientes o decrecientes específicas, aumentando la coherencia del modelo y su aceptabilidad en aplicaciones sensibles, aunque puedan reducir ligeramente la precisión (Nikhil, 2021).

Finalmente, dado que LightGBM genera modelos complejos y poco interpretables, es fundamental emplear herramientas como las importancias de características y SHAP para entender tanto el comportamiento global del modelo como el impacto individual de cada variable. Estas técnicas permiten verificar consistencia con el dominio, identificar dependencias críticas y detectar posibles problemas como sobreajuste o excesiva sensibilidad a variables ruidosas, contribuyendo a un uso responsable y transparente del algoritmo.

7. Casos de uso y pitfalls frecuentes

LightGBM se emplea ampliamente en aplicaciones que requieren alta velocidad, escalabilidad y capacidad para modelar relaciones no lineales. Es especialmente útil en modelos de riesgo y crédito, donde bancos y fintech lo usan para predecir impago, fraude y puntajes crediticios gracias a su eficiencia con datos tabulares (Royal Research, 2025). También destaca en marketing y *churn prediction*, donde se necesitan modelos actualizados para segmentación y recomendación. En plataformas como Kaggle es uno de los métodos más usados por su rapidez y precisión. Además, funciona bien en series temporales tabulares mediante ingeniería de características y en salud para predicción de riesgo clínico, apoyado por técnicas como SHAP (TutorialsPoint, 2025).

Entre los riesgos más relevantes está el *data leakage*, que ocurre cuando información del futuro o del conjunto de validación se filtra en el entrenamiento, generando métricas artificialmente altas. Debido a su crecimiento *leaf-wise*, LightGBM amplifica fácilmente señales filtradas, causando fallos en producción. Esto sucede al usar estadísticas globales, variables con información futura o mezclas incorrectas de ventanas temporales. Para prevenirlo, se debe separar datos por tiempo, calcular estadísticas solo sobre entrenamiento y revisar correlaciones anómalas.

Otro riesgo crítico es el *target leakage* en *target encoding*. Si las medias del objetivo se calculan con todo el dataset, el modelo accede indirectamente al target de validación, causando sobreajuste. La solución es aplicar el encoding dentro de cada fold, evitar medias globales y usar suavizado o ruido (Merckel, 2019). Indicadores típicos incluyen métricas casi perfectas en entrenamiento y caídas pronunciadas en test.

LightGBM también es sensible al *data drift*, es decir, cambios en la distribución entre entrenamiento y producción. Como sus divisiones dependen de histogramas, si las distribuciones cambian los *bins* dejan de ser representativos. Esto puede manifestarse como *covariate drift*, *prior drift* o *concept drift*. Para mitigarlo se deben monitorear distribuciones (KS, PSI), usar SHAP para detectar variaciones en patrones y reentrenar el modelo cuando sea necesario.

Otro problema frecuente es el sobreajuste silencioso causado por valores altos de `num_leaves`, que generan árboles demasiado profundos bajo la estrategia *leaf-wise*. Sin *early stopping*, este sobreajuste puede pasar inadvertido. De igual manera, aunque LightGBM maneja categóricas de forma nativa, las de alta cardinalidad pueden producir divisiones ruidosas, por lo que conviene agrupar categorías poco frecuentes o aplicar *target encoding* con validación cruzada.

Finalmente, LightGBM se utiliza en numerosos casos de uso como predicción de precios, análisis crediticio, comportamiento del cliente, ranking en motores de búsqueda, salud, finanzas y sistemas de recomendación. Su velocidad, precisión y capacidad para manejar grandes volúmenes explican su amplia adopción en industrias donde la escalabilidad es esencial.

8. Checklist de tuning

El ajuste de hiperparámetros en LightGBM requiere un enfoque sistemático, dado que el framework incluye parámetros que regulan la estructura del modelo, la regularización, el muestreo y la dinámica del crecimiento *leaf-wise*. Su alto rendimiento y bajo consumo de memoria se explican por técnicas como GOSS, que prioriza muestras con gradientes altos, y EFB, que agrupa características mutuamente excluyentes para reducir dimensionalidad. Asimismo, su estrategia *best-first leaf-wise* permite una rápida reducción de la pérdida, aunque también aumenta el riesgo de sobreajuste si no se controla adecuadamente.

El tuning inicia fijando parámetros básicos que aseguren un entrenamiento estable. Se recomienda `boosting = gbdt`, definir un objetivo adecuado (regresión, binario, multiclase) y establecer métricas como `auc`, `binary_logloss` o `rmse`. Para esta fase se usa un `learning_rate` provisional entre 0.01 y 0.05, junto con un número alto de iteraciones y *early stopping* (50–100 rondas), que permite estimar el número óptimo de árboles. Cuando no se define una métrica, LightGBM selecciona automáticamente la asociada al objetivo (Microsoft, 2025).

El siguiente paso es ajustar parámetros estructurales. El más importante es `num_leaves`, que determina la complejidad del modelo y debe mantenerse por debajo de $2^{\texttt{max\_depth}}$. En la práctica se usan valores entre 32 y 256 según el tamaño del dataset. La profundidad `max_depth`, comúnmente entre 4 y 12, sirve como límite cuando se detecta sobreajuste. Para evitar hojas con muy pocos datos, `min_data_in_leaf` suele ubicarse entre 20 y 200, o valores mayores en grandes volúmenes de datos. Otros parámetros como `min_gain_to_split`, `min_sum_hessian_in_lea` y `max_bin` controlan la granularidad del modelo, siendo este último sensible al equilibrio entre precisión y riesgo de sobreajuste (GeeksforGeeks, 2025a).

Tras definir la estructura del árbol, se ajusta la regularización. Los parámetros `lambda_l1` y `lambda_l2` penalizan modelos complejos y ayudan frente a características ruidosas; valores iniciales entre 0 y 5 son adecuados. La regularización puede complementarse con muestreo mediante `feature_fraction` y `bagging_fraction`, junto con `bagging_freq`. Reducir estas fracciones (0.6–0.9) ayuda a controlar la varianza y el sobreajuste. LightGBM también ofrece GOSS, cuyos parámetros `top_rate` y `bottom_rate` pueden acelerar el entrenamiento en datasets muy grandes.

Una vez estabilizados los parámetros estructurales y de regularización, se ajusta el `learning_rate` definitivo junto con `n_estimators`. Tasas pequeñas (0.005–0.03) mejoran la precisión pero requieren más árboles, por lo que el *early stopping* sigue siendo clave para determinar la iteración óptima. En escenarios donde la velocidad es crítica, reducir `num_leaves` y `max_depth` o aumentar `min_gain_to_split` acelera el entrenamiento. Además, el uso de GPU o entrenamiento distribuido puede disminuir significativamente los tiempos (Microsoft, 2025).

Finalmente, LightGBM ofrece opciones avanzadas para casos específicos. El soporte nativo de `categorical_feature` evita codificación *one-hot*, aunque en variables de alta cardinalidad pueden requerirse técnicas como agrupación o *target encoding* con validación cruzada para evitar fugas de información. Las `monotone_constraints` permiten imponer relaciones crecientes o decrecientes entre características y objetivo, útiles en dominios regulados. También existen variantes como `dart` y `rf`, apropiadas cuando se necesita mayor robustez o diversificación del modelo.

En síntesis, el tuning óptimo sigue un flujo en etapas: definir parámetros básicos y métricas; ajustar la estructura mediante `num_leaves`, `max_depth` y `min_data_in_leaf`; aplicar regularización; introducir muestreo; ajustar tasa de aprendizaje e iteraciones; y usar parámetros avanzados según el dominio. Este enfoque escalonado favorece modelos estables, eficientes y con alta capacidad de generalización.

Referencias

Brenndoerfer, M. (2025). *Lightgbm: Fast gradient boosting & leaf-wise tree growth. a complete guide to the mathematical foundations and Python implementation.* https://mbrenndoerfer.com/writing/lightgbm-fast-gradient-boosting-leaf-wise-tree-growth-complete-guide-mathematical-foundations-python-implementation. (Accessed: 2025-11-17)

Budiawan, J. C. (2025). *Understanding lightgbm: A performance-boosting algorithm for machine learning.* https://medium.com/@jeremychrist23/understanding-lightgbm-a-performance-boosting-algorithm-for-machine-learning-756a1edb38c4. (Accessed: 2025-11-17)

GeeksforGeeks. (2025a). *Lightgbm feature parameters.* https://www.geeksforgeeks.org/machine-learning/lightgbm-feature-parameters/. (Accessed: 2025-11-17)

GeeksforGeeks. (2025b). *Lightgbm (light gradient boosting machine).* https://www.geeksforgeeks.org/machine-learning/lightgbm-light-gradient-boosting-machine/. (Accessed: 2025-11-17)

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. En *31st conference on neural information processing systems (nips 2017).* Long Beach, CA, USA.

Merckel, L. (2019). *Target encoding and lightgbm.* https://www.kaggle.com/code/merckel/target-encoding-and-lightgbm. (Accessed: 2025-11-17)

Microsoft. (2025). *Parameters tuning.* https://lightgbm.readthedocs.io/en/stable/Parameters-Tuning.html. (Accessed: 2025-11-17)

Nikhil. (2021). *Monotonic constraints in xgboost, lgbm and catboost.* https://www.kaggle.com/discussions/getting-started/273527. (Accessed: 2025-11-17)

Royal Research. (2025). *Lightgbm: Lightning fast gradient boosting for large datasets.* https://royalresearch.in/lightgbm-lightning-fast-gradient-boosting-for-large-datasets. (Accessed: 2025-11-17)

Shi, H. (2007). *Best-first decision tree learning* (Tesis de Master no publicada). The University of Waikato, Hamilton, New Zealand. (Thesis for the degree of Master of Science)

Surana, S. (2020). *What is lightgbm? advantages & disadvantages? lightgbm vs xgboost?* https://www.kaggle.com/discussions/general/264327. (Accessed: 2025-11-17)

TutorialsPoint. (2025). *Lightgbm overview.* https://www.tutorialspoint.com/lightgbm/lightgbm-overview.htm. (Accessed: 2025-11-16)