CatBoost (Categorical Boosting)

Data Mining P-Set 5

Autores: Brandon Mosquera (00331115), Maria Emilia Granda (00330318),
Víctor Bastidas (00215865) y Juan Chicaiza (00328175)

23 de noviembre de 2025

1. Resumen

CatBoost (Categorical Boosting) es un algoritmo de *gradient boosting* sobre árboles de decisión (GBDT) de código abierto desarrollado por Yandex [1, 2]. Su nombre se deriva de su capacidad fundamental para manejar características **cat**egóricas de forma nativa y eficiente [4]. La idea central de CatBoost es doble: primero, elimina la necesidad de preprocesamiento manual de características categóricas (como *one-hot encoding* o *target encoding*), integrando un manejo superior directamente en el algoritmo [5]. Segundo, resuelve un problema estadístico fundamental de sesgo conocido como *prediction shift* (fuga de predicción), presente en implementaciones GBDT tradicionales [1]. Esto se logra mediante dos innovaciones algorítmicas: *Ordered Target Statistics* (OTS) para la codificación de categóricas y *Ordered Boosting* para el cálculo de gradientes [1,2]. CatBoost es la herramienta de elección para datos tabulares con alta cardinalidad categórica, demostrando una robustez y precisión excepcionales [6].

2. Formulación Matemática

CatBoost construye un modelo aditivo fuerte optimizando una función de pérdida diferenciable mediante descenso de gradiente en el espacio de funciones [3].

2.1. Mecanismo Aditivo (Ensamblado)

El objetivo es aprender una función de predicción $F(\mathbf{x})$ que minimice una pérdida $L(y, F(\mathbf{x}))$. El modelo final es la suma de T árboles de regresión h_t:

$$F_T(\mathbf{x}) = F_0(\mathbf{x}) + \sum_{t=1}^{T} \alpha \cdot h_t(\mathbf{x})$$

2.2. Descenso de Gradiente

En cada paso t, se entrena un árbol h_t para predecir el negativo del gradiente de la función de pérdida (pseudo-residuales) con respecto a la predicción anterior $F_{t-1}(\mathbf{x})$:

$$h_t = \arg\min_h \sum_{i=1}^{N} (r_{i,t} - h(\mathbf{x}_i))^2$$

donde $r_{i,t}$ es el pseudo-residual [3].

2.3. Modelo de Predicción para P-Set 5

Para este problema específico, definimos el modelo matemático de regresión para predecir el monto total y (`total_amount`) en función de un vector de características $\mathbf{x}$ restringido para evitar *data leakage*.

Sea $\mathbf{x}_i$ el vector de entrada para el viaje i, definido por la concatenación de las siguientes variables observadas *a priori*:

$$\mathbf{x}_i = [\mathbf{t}_i, \mathbf{l}_i, \mathbf{s}_i, \mathbf{p}_i, \mathbf{c}_i] \tag{1}$$

Donde los componentes son:

- $\mathbf{t}_i$ (Tiempo): {hora, día, mes, año}. Variables cíclicas y ordinales que capturan estacionalidad y demanda.
- $\mathbf{l}_i$ (Ubicación): {PU_LocationID}. Variable categórica de alta cardinalidad que indica el origen.
- $\mathbf{s}_i$ (Servicio): {trip_type, store_and_fwd}. Variables categóricas del tipo de despacho.
- $\mathbf{p}_i$ (Proveedor): {VendorID}. Variable categórica.
- $\mathbf{c}_i$ (Características): {passenger_count, RatecodeID}.

El modelo busca aproximar la función de precio real Φ:

$$\hat{y}_i = F(\mathbf{x}_i) \approx \Phi(\mathbf{t}_i, \mathbf{l}_i, \mathbf{s}_i, \mathbf{p}_i, \mathbf{c}_i)$$

Nótese que variables como `trip_distance` o `DOLocationID` están explícitamente excluidas de $\mathbf{x}_i$ dado que representan información que no se conoce con certeza al momento de solicitar el servicio (no-leakage).

2.3.1. RMSE (Root Mean Square Error)

Es la función de pérdida por defecto para regresión en CatBoost. Penaliza cuadráticamente los errores (siendo muy sensible a *outliers*). La fórmula implementada (que incluye pesos w_i) es:

$$L_{RMSE} = \sqrt{\frac{\sum_{i=1}^{N}(a_i - t_i)^2 w_i}{\sum_{i=1}^{N} w_i}}$$

donde a_i es la predicción (actual) y t_i es el valor real (target).

2.3.2. MAE (Mean Absolute Error)

Es una alternativa más robusta a los *outliers*, ya que penaliza los errores de forma lineal [4]. La fórmula es:

$$L_{MAE} = \frac{\sum_{i=1}^{N} w_i |a_i - t_i|}{\sum_{i=1}^{N} w_i}$$

Dado que los datos de tarifas de taxi (como `total_amount`) a menudo presentan valores atípicos (viajes inusualmente caros), entrenar con `loss_function='MAE'` puede producir un modelo más robusto [7].

2.4. Regularización (L2)

Para combatir el sobreajuste, CatBoost implementa regularización. El hiperparámetro clave requerido por el P-Set es `l2_leaf_reg` [12]. Este parámetro añade una penalización L2 (similar a la regresión Ridge) al cálculo de los valores finales en las hojas de cada árbol [1]. En esencia, penaliza valores grandes en las hojas (contribuciones del árbol), forzando al modelo a ser más conservador y mejorando su capacidad de generalización.

3. Algoritmo y Mecanismos Internos

CatBoost no es simplemente otra implementación de *GBDT*. Incorpora mecanismos diseñados para reducir el sesgo por ordenamiento y el sobreajuste, especialmente en la manipulación de variables categóricas.

3.1. Algoritmo (alto nivel)

1. **Inicialización**: Se comienza con un modelo constante (por ejemplo, la media del objetivo).
2. **Generación de permutaciones**: Se generan varias permutaciones aleatorias de los datos para estimar estadísticas y gradientes de forma estable.
3. **Iteración** ($t = 1 \dots M$):
   I. **Cálculo de gradientes/residuos**: Se obtienen los gradientes negativos usando el esquema de *Ordered Boosting*, donde cada punto solo utiliza información disponible "hasta su posición" en la permutación.
   II. **Construcción del árbol**: Se entrena un árbol simétrico (*oblivious tree*).
   - Se evalúan divisiones tanto en variables numéricas como en combinaciones de categóricas.
   - Se selecciona la división con mayor ganancia (por ejemplo, usando la similitud coseno con el gradiente).

   III. **Cálculo de hojas**: Se asignan los valores óptimos a cada hoja para minimizar la pérdida en los ejemplos que llegan a ella.
   IV. **Actualización**: El ensamble se actualiza con el nuevo árbol,

$$F_t = F_{t-1} + \eta \cdot h_t.$$

3.2. El Problema Fundamental: *Prediction Shift*

El problema central que CatBoost aborda es el *prediction shift* (desplazamiento de la predicción) [1]. En los GBDT estándar, la distribución de las predicciones (y de los gradientes) en los datos de entrenamiento difiere sistemáticamente de la distribución en los datos de prueba. Este sesgo es causado por una forma sutil de *target leakage*: el pseudo-residual $r_{i,t}$ para una muestra x_i se calcula usando la predicción $F_{t-1}(x_i)$. Sin embargo, $F_{t-1}(x_i)$ es un modelo entrenado (en pasos anteriores) usando la misma muestra x_i y, por lo tanto, tiene conocimiento de y_i. Esto significa que y_i influye en el cálculo de $r_{i,t}$, sesgando los gradientes [1].

3.3. Innovación 1: Manejo de Categóricas (*Ordered Target Statistics*)

Esta es la solución de CatBoost al *target leakage* en la codificación de características categóricas. El método ingenuo (por ejemplo, reemplazar 'Midtown' por el `avg(total_amount)` de todos los viajes con origen 'Midtown') filtra directamente el valor objetivo y_i en la característica x_i de esa misma muestra [5].

CatBoost resuelve esto con *Ordered Target Statistics* (OTS) [2]:

1. **Permutaciones Aleatorias:** CatBoost genera múltiples permutaciones aleatorias (σ) del conjunto de entrenamiento.
2. **Codificación Ordenada:** Para codificar la característica categórica de una muestra x_k, el algoritmo utiliza únicamente los valores objetivo y_j de aquellas muestras x_j que aparecen *antes* que x_k en esa permutación σ.
3. **Prevención de *Leakage*:** Esto simula un orden temporal artificial, asegurando que la información de y_k nunca se use para calcular las características de x_k [1].

3.4. Innovación 2: Árboles Simétricos (*Oblivious Trees*)

CatBoost utiliza una estructura de árbol diferente a la de sus competidores (como XGBoost [8]). En lugar de árboles asimétricos, CatBoost construye *oblivious trees*

(árboles simétricos). En un árbol simétrico, todos los nodos en un mismo nivel de profundidad utilizan *exactamente la misma* característica y el mismo umbral de división [2].

Esto tiene varias ventajas significativas:

1. **Regularización Estructural:** La estructura está balanceada, actuando como un regularizador que previene el sobreajuste.

2. **Inferencia Rápida:** La estructura simétrica permite una evaluación de predicciones extremadamente rápida [4].

3.5. Innovación 3: *Ordered Boosting*

Esta innovación aplica el mismo principio de .ºrdenamiento"de OTS al proceso de *boosting*. En *Ordered Boosting*, CatBoost entrena modelos de soporte en subconjuntos prefijos de los datos, según distintas permutaciones aleatorias, para que el gradiente de una muestra se calcule con un modelo que no ha "visto.ªesa muestra durante el entrenamiento [1].

4. Hiperparámetros Clave y Efectos

La Tabla 1 resume los efectos de los parámetros clave solicitados en el P-Set 5 [12].

5. Ventajas y Limitaciones

5.1. Ventajas

- **Manejo nativo de categóricas:** Su capacidad de procesar características categóricas con OTS elimina la necesidad de codificación manual [6].

- **Alta precisión y robustez:** Gracias a *Ordered Boosting*, CatBoost tiende a evitar el sobreajuste de forma inherente [1].

- **Inferencia rápida:** La estructura simétrica de los árboles permite predicciones muy rápidas, ideal para producción [4].

5.2. Limitaciones

- **Velocidad de entrenamiento en CPU:** Debido a las permutaciones de OTS, puede ser más lento que LightGBM [9] en CPU. Se recomienda GPU para datasets grandes.

- **Datos puramente numéricos:** Si no hay variables categóricas, su ventaja competitiva disminuye frente a XGBoost o LightGBM [6].

6. Buenas Prácticas de Implementación

6.1. Optimización con `early_stopping`

Un error común es sintonizar manualmente `n_estimators`. La mejor práctica es utilizar el detector de sobreajuste incorporado. Se fija `iterations` en un valor alto y se usa `early_stopping_rounds` con un conjunto de validación [4].

6.2. Interpretabilidad con SHAP

CatBoost se integra nativamente con la librería SHAP para explicar predicciones [10].

- **Global:** `shap.summary_plot` para ver qué variables (ej. `trip_distance`) importan más.

- **Local:** `shap.force_plot` para explicar una predicción individual.

6.3. Conocimiento de Dominio (`monotonic_constraints`)

CatBoost permite imponer restricciones monotónicas. Para el P-Set 5, sabemos que `total_amount` debe aumentar con `trip_distance`. Fijar `monotonic_constraints` fuerza esta lógica, reduciendo el sobreajuste al ruido [4].

7. Casos de Uso y *Pitfalls*

7.1. Casos de Uso

Ideal para datos con alta cardinalidad categórica, detección de fraude y sistemas de recomendación [1].

7.2. Pitfall: Deriva de Datos (*Data Drift*)

La deriva de datos ocurre cuando la distribución en producción difiere del entrenamiento [11]. En el P-Set 5 (2015-2025), eventos como la pandemia crean deriva. Se debe monitorear y reentrenar periódicamente, o usar pesos temporales.

8. Checklist de *tuning*: orden recomendado

El ajuste de hiperparámetros en CatBoost debe seguir un orden racional. Esto evita perder tiempo explorando combinaciones innecesarias y permite estabilizar primero los parámetros que más influyen en el desempeño. La siguiente secuencia ofrece un camino claro para construir un modelo sólido sin desviarse de lo esencial.

Cuadro 1: Hiperparámetros clave de CatBoost para regresión

Hiperparámetro	Descripción	Efecto en el modelo (sesgo vs. varianza)
`depth`	Profundidad máxima de los árboles simétricos (Default: 6).	**Controla la complejidad:** mayor profundidad reduce el sesgo, pero incrementa la varianza y el tiempo de entrenamiento [4].
`learning_rate`	Tasa de aprendizaje que escala la contribución de cada árbol (Default: auto) [3].	**Convergencia:** tasas más pequeñas reducen la varianza pero requieren más iteraciones para lograr buen desempeño.
`l2_leaf_reg`	Coeficiente de regularización L2 aplicado en las hojas (Default: 3).	**Regularización explícita:** valores mayores penalizan la complejidad, reducen la varianza y previenen el sobreajuste [4].
`bagging_temperature`	Controla la intensidad del *Bayesian bootstrap*, asignando pesos aleatorios a las muestras.	**Aleatorización:** valores altos incrementan la diversidad del ensamble y mitigan el sobreajuste [2].

1. **Configuración inicial**. Comenzar con los valores por defecto, que suelen ser estables. Definir correctamente las `cat_features` es crítico para que el algoritmo trate las variables categóricas como corresponde. Si se dispone de GPU, especificar `task_type="GPU"` reduce drásticamente los tiempos de entrenamiento, lo que facilita el resto del proceso de *tuning*.

2. **Número de árboles e *learning rate***. Elegir un *learning rate* inicial razonable (por ejemplo, 0.05 o 0.1) y usar *early stopping* para determinar cuántas iteraciones son necesarias. Esto fija la "escala" del modelo. Un *learning rate* grande converge rápido pero puede ser inestable; uno pequeño requiere más árboles, pero produce modelos más suaves y generales.

3. **Profundidad del árbol (`depth`)**. Probar valores típicos como {4, 6, 8, 10}. Este es uno de los parámetros más sensibles tanto en rendimiento como en tiempo de entrenamiento. Profundidades bajas producen modelos más simples y rápidos; profundidades mayores permiten capturar interacciones más complejas, pero aumentan el riesgo de sobreajuste y el costo computacional.

4. **Regularización (`l2_leaf_reg`)**. Ajustar valores como {1, 3, 5, 10}. Este término controla la magnitud de los valores asignados a las hojas del árbol y actúa como freno para la varianza del modelo. Si el modelo se vuelve muy volátil o sobreajusta, conviene aumentar esta regularización; si el modelo queda subentrenado, puede ser útil reducirla.

5. **Estocasticidad (`rsm`, `subsample`)**. Introducir aleatoriedad en la selección de columnas (`rsm`) y filas (`subsample`) ayuda a mejorar la robustez y a evitar que el modelo dependa demasiado de patrones espurios. Es útil explorar ligeras reducciones respecto al valor 1.0, por ejemplo {0.7, 0.8, 0.9}, dependiendo del tamaño del dataset.

6. **Refinamiento final**. Una vez estabilizados los parámetros principales, se puede ajustar el modelo para producción. Esto consiste en reducir el *learning rate* y aumentar proporcionalmente el número de iteraciones. Así se obtiene un modelo más suave y preciso, sin sacrificar estabilidad. Este paso debe aplicarse sobre la mejor configuración hallada, no desde cero.

9. Aplicación al P-Set 5: Estrategia y Validación

9.1. Definición de la Tarea y Datos

El objetivo del P-Set 5 es predecir `total_amount` bajo un escenario de Pre-Dispatch o cotización previa. Para garantizar una evaluación realista y libre de fugas (*leakage*), se ha estructurado el particionamiento temporal de la siguiente manera:

- **Entrenamiento (Train):** Datos de 2022 y 2023. El modelo aprende los patrones históricos base de tarifas y estacionalidad.
- **Validación (Val):** Datos de 2024. Se utiliza para la sintonización de hiperparámetros y *early stopping*. Esto simula el despliegue del modelo en el futuro inmediato.
- **Prueba (Test):** Datos de 2025. Evaluación final no vista por el modelo.

9.2. Prevención de Data Leakage

Un desafío crítico abordado en esta implementación es el *leakage*. Variables como `trip_distance` fueron removidas. El modelo $F(\mathbf{x})$ depende exclusivamente de variables instanciadas al momento de la solicitud: ¿Dónde estoy? (`pu_*`), ¿A dónde voy? (`do_*`) ¿Qué hora es? (tiempo), ¿Quién me lleva? (`vendor`) y ¿Qué servicio pido? (`rate_code`).

Referencias

[1] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, y A. Gulin, "CatBoost: unbiased boosting with categorical features," en *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6638–6648, 2018.

[2] A. V. Dorogush, V. Ershov, y A. Gulin, "CatBoost: gradient boosting with categorical features support," *arXiv preprint arXiv:1810.11363*, 2018.

[3] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[4] Yandex, "CatBoost Documentation," [En línea]. Disponible: https://catboost.ai/docs/. Accedido: 2025.

[5] J. T. Hancock y T. M. Khoshgoftaar, "Survey on categorical data for neural networks," *Journal of Big Data*, vol. 7, no. 1, p. 1, 2020.

[6] A. Verma y S. Roul, "Comparison of Data Mining Classification Algorithms," *International Journal of Computer Applications*, vol. 177, no. 25, 2019.

[7] T. Hastie, R. Tibshirani, y J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, 2nd ed., Springer, 2009.

[8] T. Chen y C. Guestrin, "XGBoost: A Scalable Tree Boosting System," en *Proceedings of the 22nd ACM SIGKDD*, 2016.

[9] G. Ke et al., "LightGBM: A Highly Efficient Gradient Boosting Decision Tree," en *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[10] S. M. Lundberg y S.-I. Lee, "A Unified Approach to Interpreting Model Predictions," en *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[11] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, y G. Zhang, "Learning under Concept Drift: A Review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 12, 2018.

[12] Cátedra de Minería de Datos, *Data Mining P-Set 5: Predicción de Tarifa de Taxi de NYC (2015–2025)*, Universidad San Francisco de Quito, 2025.