Cambio a `embed` manual de árboles XGBoost (regla $<$ estricta)

Proyecto Optimización Casas Ames

29 de noviembre de 2025

1. Motivación

En el modelo de remodelación se observaba una brecha sistemática entre la predicción interna del MIP (`y_log_raw(MIP)`) y la predicción externa de XGBoost sobre el mismo vector de entrada. A pesar de alinear versiones de `scikit-learn`/`xgboost`/`gurobi_ml`, la brecha se mantuvo ($\approx$ 0,12–0,14 en log), indicando que la conversión automática de `gurobi_ml.attach_to_gurobi` no reproducía exactamente los umbrales de los árboles.

2. Qué se cambió

- Se reemplazó el uso de `gurobi_ml.attach_to_gurobi` por un **embebido manual** de los árboles del booster en `optimization/remodel/gurobi_model.py`.
- Para cada árbol se parsea el `dump` JSON, se crean variables binarias de hoja (`t{idx}_leaf{k}`), se imponen las rutas de decisión con regla **estricta** $x < thr$ (rama izquierda) y $x \geq thr$ (rama derecha) usando big-M basado en los límites de cada variable, y se fuerza one-hot de hojas.
- La suma de hojas se vincula a `y_log_raw`; el `offset` (`b0`) se sigue sumando igual que antes.

3. Efecto esperado

- El recorrido del booster dentro de Gurobi coincide con la predicción externa de XGBoost (sin brechas de umbrales ni empates $x{=}thr$ enviados al lado incorrecto).
- Se eliminan los `TREE-MISMATCH` originados por empates o decisiones distintas de `gurobi_ml`; la predicción del MIP debe igualar (o quedar dentro de redondeo numérico) a `bundle.predict`.
- No cambia el modelo entrenado ni los hiperparámetros; sólo la forma de representarlo en el MILP.

4. Lo que no cambia

- Hiperparámetros XGBoost, features usadas, target, escalas y preprocesamiento.
- Costos, restricciones de remodelación y lógica de presupuestos/ROI.
- El `offset` (`b0`) del booster se sigue aplicando; si el booster trae `best_iteration`, se respetan las iteraciones truncadas.

5. Impacto en el modelo final

- Las predicciones del MIP ahora deben ser consistentes con el predictor externo; desaparecen brechas de 10–15 % en log que inflaban el ROI.
- La interpretabilidad del recorrido de cada árbol mejora (las hojas activas son exactamente las del booster para el mismo `X`).
- El tamaño del modelo crece (una variable binaria por hoja), pero se mantiene dentro de rangos manejables para Gurobi en los casos actuales.

6. Notas de implementación

- Archivo modificado: `optimization/remodel/gurobi_model.py`.
- Se usa `EPS_SPLIT = 1e-6` para asegurar estrictez: rama izquierda $x < thr - \epsilon$, rama derecha $x \geq thr + \epsilon$.
- Big-M por variable se calcula a partir de `LB`/`UB`; si no están definidos, se usa `1e6` como fallback.
- Se mantiene el guardado de `y_log_raw` y `y_log` para auditorías, así como los checks de one-hot y los reportes de `TREE-MISMATCH` (que ahora deberían ser cero si los datos son consistentes).

7. Próximos pasos

- Validar con `--debug-tree-mismatch` que el gap de `y_log` desaparezca en los PIDs de interés.
- Revisar variables que siguen cambiando sin costo (p.ej. mejoras "free" en sótano o exteriores) para evitar ROI artificialmente alto.
- Mantener las versiones de `scikit-learn` y `xgboost` alineadas con el modelo (`requirements.txt` fija `scikit-learn==1.7.1`).