```python
### SCRIPT 16 - PYTHON
# OPTIMIZACION DEL UMBRAL "BAJO" PARA EL MODELO XGB DE COMPETENCIA
#
# ======================================================================
# -*- coding: utf-8 -*-

import json
import joblib
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score, precision_recall_fscore_support, accuracy_score,
 confusion_matrix

# ========================
# 1. Rutas de entrada
# ========================
PATH_TRAIN =
"/content/drive/MyDrive/_Pipeline_desarrollo_modelos_v2/validacion/03_xgb_n_ofertantes
/df_train_XGB_v8.csv"
PATH_XGB_PACK =
"/content/drive/MyDrive/_Pipeline_desarrollo_modelos_v2/modelos/model_Noferta_XGB_v8.p
kl"
PATH_XGB_META =
"/content/drive/MyDrive/_Pipeline_desarrollo_modelos_v2/modelos/xgb_meta.json"  #
ajusta si hace falta

# ========================
# 2. Cargar datos y construir y_train_bin
# ========================
df_train = pd.read_csv(PATH_TRAIN)

print("Columnas disponibles en df_train:")
print(df_train.columns.tolist())

# Usamos N_ofertantes_label = BAJO / MEDIO / ALTO
if "N_ofertantes_label" not in df_train.columns:
    raise ValueError("No encuentro 'N_ofertantes_label' en df_train_XGB_v8.csv")

print("\nValores únicos en N_ofertantes_label:")
print(df_train["N_ofertantes_label"].unique())

labels_norm = df_train["N_ofertantes_label"].astype(str).str.strip().str.upper()
y_train_bin = (labels_norm == "BAJO").astype(int).values

# Comprobación básica
unique, counts = np.unique(y_train_bin, return_counts=True)
print("\nDistribución y_train_bin (0=NO BAJO, 1=BAJO):")
print(dict(zip(unique, counts)))

if len(unique) < 2:
    raise ValueError("y_train_bin solo tiene una clase. Revisa las etiquetas (no hay
    BAJO o no BAJO en TRAIN).")

# ========================
# 3. Cargar pack XGB y metadatos
# ========================
xgb_pack = joblib.load(PATH_XGB_PACK)   # contiene {"modelo": ..., "encoders": ...}
with open(PATH_XGB_META, "r", encoding="utf-8") as f:
    xgb_meta = json.load(f)

xgb_model = xgb_pack["modelo"]
xgb_vars  = xgb_meta["variables_predictoras"]
encoders  = xgb_pack.get("encoders", {})
labels    = xgb_meta.get("labels", ["BAJO", "MEDIO", "ALTO"])
idx_bajo  = labels.index("BAJO") if "BAJO" in labels else 0

policies  = xgb_meta.get("policies", {})
fallback_classes = policies.get("fallback_class_por_col", {})
enc_cols  = xgb_meta.get("encoders_columns", [])

print("\nVariables usadas por XGB:")
print(xgb_vars)
```

```python

# ========================
# 4. Construir X como en producción
# ========================
def build_X_for_xgb(df_raw: pd.DataFrame) -> pd.DataFrame:
    X = pd.DataFrame(index=df_raw.index)

    alias_mapping = {
        "C_precio_p": "Criterio_precio_p",
    }

    for col in xgb_vars:
        if col in df_raw.columns:
            X[col] = df_raw[col]
        elif col in alias_mapping and alias_mapping[col] in df_raw.columns:
            X[col] = df_raw[alias_mapping[col]]
        else:
            if col in enc_cols:
                fallback_val = fallback_classes.get(col, "")
                X[col] = fallback_val
            else:
                X[col] = 0

    # Encoders categóricos
    for col, le in encoders.items():
        if col in X.columns:
            s = X[col].astype("string")
            fb = fallback_classes.get(col, None)
            if fb is not None:
                s = s.fillna(fb)
                s = s.replace(["nan", "NaN", "None", ""], fb)
            else:
                moda = s.mode().iloc[0] if not s.mode().empty else "UNK"
                s = s.fillna(moda)
                s = s.replace(["nan", "NaN", "None", ""], moda)

            known = set(le.classes_.astype(str))
            if fb is not None and fb in known:
                s = s.apply(lambda v: v if v in known else fb)
            else:
                default_class = le.classes_[0]
                s = s.apply(lambda v: v if v in known else default_class)

            X[col] = le.transform(s.astype(str))
        else:
            X[col] = 0

    if "Mes_lici" in X.columns and not pd.api.types.is_integer_dtype(X["Mes_lici"]):
        X["Mes_lici"] = pd.to_numeric(X["Mes_lici"], errors="coerce").fillna(0).astype
        (int)

    return X[xgb_vars]

X_train = build_X_for_xgb(df_train)

# ========================
# 5. Probabilidades de BAJO en TRAIN
# ========================
proba = xgb_model.predict_proba(X_train)
p_bajo_train = proba[:, idx_bajo]

# ========================
# 6. Buscar umbral que maximiza F1 (BAJO)
# ========================
thresholds = np.linspace(0.0, 1.0, 101)
best_t = None
best_f1 = -1.0

for thr in thresholds:
    y_pred = (p_bajo_train >= thr).astype(int)
    f1 = f1_score(y_train_bin, y_pred)
    if f1 > best_f1:
```

```
            best_f1 = f1
            best_t = thr

    print(f"\nUmbral óptimo F1 (TRAIN): {best_t:.2f}  |  F1_BAJO = {best_f1:.3f}")

    # ==========================
    # 7. Métricas detalladas en TRAIN con el umbral óptimo
    # ==========================
    y_pred_best = (p_bajo_train >= best_t).astype(int)

    prec, rec, f1, _ = precision_recall_fscore_support(
        y_train_bin, y_pred_best, average="binary", pos_label=1
    )
    acc = accuracy_score(y_train_bin, y_pred_best)
    cm = confusion_matrix(y_train_bin, y_pred_best)

    print("\nMatriz BAJO vs NO BAJO en TRAIN (umbral óptimo F1):")
    print(cm)  # [[TN, FP], [FN, TP]]

    print("\nMétricas binaria BAJO vs NO BAJO en TRAIN:")
    print(f"  Precision_BAJO = {prec:.3f}")
    print(f"  Recall_BAJO    = {rec:.3f}")
    print(f"  F1_BAJO        = {f1:.3f}")
    print(f"  Accuracy       = {acc:.3f}")
```