

✓ 04.3_Modelo_LightGBM

Objetivo

Entrenar y optimizar un modelo de clasificación con LightGBM para predecir el riesgo de clientes. El notebook aplica SMOTE para balancear las clases de entrenamiento y utiliza Optuna para una búsqueda exhaustiva de hiperparámetros. Finalmente, se guarda el modelo optimizado como un artefacto experimental.

Entradas (Inputs)

- data/splits/experiments/X_train_17.parquet
- data/splits/experiments/X_val_17.parquet
- data/splits/experiments/X_test_17.parquet
- data/splits/final/y_train_2_classes.parquet
- data/splits/final/y_val_2_classes.parquet
- data/splits/final/y_test_2_classes.parquet

Salidas (Outputs)

Artefactos Experimentales:

- artifacts/experiments/04_3_lgbm_model.pkl
-

Resumen Ejecutivo

- Se cargan los “splits” preprocesados (17 features) y las etiquetas binarias (Bajo/Medio vs. Alto Riesgo) en formato Parquet.
- Se construye un pipeline con **LightGBMClassifier** (`class_weight='balanced'`) sin escalado explícito, aprovechando el manejo nativo de variables numéricas.
- Se lleva a cabo un **GridSearchCV** estratificado (5-fold) optimizando **F1_weighted**, explorando `n_estimators`, `learning_rate`, `num_leaves` y `max_depth`.
- Parámetros óptimos: **n_estimators=200**, **learning_rate=0.05**, **num_leaves=31**, **max_depth=6**, logrando **F1_weighted CV≈0.74** y **AUC ROC CV≈0.80**.
- En validación (umbral=0.5): **Accuracy=0.64**, **F1_macro=0.61**, **AUC=0.84**, con matriz de confusión y reporte de clasificación.
- En test (umbral=0.5): **Accuracy=0.62**, **F1_macro=0.58**, **AUC=0.82**, manteniendo consistencia de desempeño.

- La curva Precision-Recall identifica un **umbral óptimo ≈0.47** para maximizar el recall de Alto Riesgo sin degradar el de Bajo/Medio.
 - Con el umbral ajustado, el **recall de Alto Riesgo** sube de 0.56 a **0.79** en validación, mientras el **recall de Bajo/Medio** se mantiene en **0.92**.
 - La importancia de variables nativa de LightGBM destaca a **TRADER_SCORE, B4_log, F31_2, S_Age** y **PORTFOLIO_DIVERSITY** como principales drivers.
 - Finalmente, se serializan el modelo (.pk1) y el umbral (.json) listos para desplegar en producción.
-

▼ 1. Instalar dependencias, montar Drive e importar librerías

Instala las librerías necesarias (`lightgbm`, `optuna`, `imbalanced-learn`), monta Google Drive, añade la raíz del proyecto al `sys.path`, desactiva los logs de Optuna y prepara el entorno para trabajar.

```
# INSTALAR, MONTAR, IMPORTAR Y CARGAR CONFIGURACIÓN

# 1. Instalar las librerías necesarias
!pip install lightgbm optuna imbalanced-learn -q

# 2. Imports básicos y de modelos

# Standard library
import sys
import json
from pathlib import Path

# Google Colab
from google.colab import drive

# Data processing
import pandas as pd
import numpy as np
import joblib

# LightGBM & Optuna
import lightgbm as lgb
import optuna

# Imbalanced-learn
from imblearn.over_sampling import SMOTE

# Scikit-learn
from sklearn.model_selection import cross_val_score
from sklearn.metrics import (
    accuracy_score, f1_score, roc_auc_score,
    classification_report, confusion_matrix,
    ConfusionMatrixDisplay, recall_score, precision_score
)
```

```

# Matplotlib
import matplotlib.pyplot as plt

# 3. Montar Google Drive
drive.mount('/content/drive', force_remount=True)

# 4. Añadir la raíz del proyecto al path
ROOT_PATH_STR = '/content/drive/MyDrive/TFM-AntonioEsquinas'
if ROOT_PATH_STR not in sys.path:
    sys.path.append(ROOT_PATH_STR)

# 5. Importar las rutas necesarias desde el archivo de configuración
from config import EXP_SPLITS_DIR, FINAL_SPLITS_DIR, EXP_ARTIFACTS_DIR

# Desactivar logs de Optuna para una salida más limpia
optuna.logging.set_verbosity(optuna.logging.WARNING)

print(" Drive montado, librerías importadas y configuración de rutas cargada.")

# CARGAR LOS CONJUNTOS DE DATOS

try:
    # Cargar las 17 features del experimento 03.4
    X_train_orig = pd.read_parquet(EXP_SPLITS_DIR / 'X_train_17.parquet')
    X_val      = pd.read_parquet(EXP_SPLITS_DIR / 'X_val_17.parquet')
    X_test     = pd.read_parquet(EXP_SPLITS_DIR / 'X_test_17.parquet')

    # Cargar las etiquetas binarias desde la carpeta FINAL
    y_train_orig = pd.read_parquet(FINAL_SPLITS_DIR / 'y_train_2_classes.parquet').squeeze()
    y_val      = pd.read_parquet(FINAL_SPLITS_DIR / 'y_val_2_classes.parquet').squeeze()
    y_test     = pd.read_parquet(FINAL_SPLITS_DIR / 'y_test_2_classes.parquet').squeeze()

    print("\n Datos .parquet cargados correctamente.")
    print(f"X_train_orig shape: {X_train_orig.shape} | y_train_orig shape: {y_train_orig.shape}")

except Exception as e:
    print(f"\n Ocurrió un error inesperado al cargar los datos: {e}")

```

→ Mounted at /content/drive
✓ Drive montado, librerías importadas y configuración de rutas cargada.
✓ Datos .parquet cargados correctamente.
X_train_orig shape: (1976, 14) | y_train_orig shape: (1976,)

▼ 2. Cargar los conjuntos de datos

Utiliza un bloque try-except para leer los archivos Parquet de características (`X_train_orig`, `X_val`, `X_test`) y etiquetas binarias (`y_train_orig`, `y_val`, `y_test`), e informa si la carga fue exitosa o si ocurrió algún error.

```
# CARGAR LOS CONJUNTOS DE DATOS
```

```
try:
    # Cargar las 17 features del experimento 03.4
    X_train_orig = pd.read_parquet(EXP_SPLITS_DIR / 'X_train_17.parquet')
    X_val      = pd.read_parquet(EXP_SPLITS_DIR / 'X_val_17.parquet')
    X_test     = pd.read_parquet(EXP_SPLITS_DIR / 'X_test_17.parquet')

    # Cargar las etiquetas binarias desde la carpeta FINAL
    y_train_orig = pd.read_parquet(FINAL_SPLITS_DIR / 'y_train_2_classes.parquet').squeeze()
    y_val      = pd.read_parquet(FINAL_SPLITS_DIR / 'y_val_2_classes.parquet').squeeze()
    y_test     = pd.read_parquet(FINAL_SPLITS_DIR / 'y_test_2_classes.parquet').squeeze()

    print("\n Datos .parquet cargados correctamente.")
    print(f"    -> Features 'X' cargadas desde: '{EXP_SPLITS_DIR.name}'")
    print(f"    -> Etiquetas 'y' cargadas desde: '{FINAL_SPLITS_DIR.name}'")
    print(f"\nX_train_orig shape: {X_train_orig.shape} | y_train_orig shape: {y_train_orig.shape}")

except Exception as e:
    print(f"\n Ocurrió un error inesperado al cargar los datos: {e}")
```



- ✓ Datos .parquet cargados correctamente.
 - > Features 'X' cargadas desde: 'experiments'
 - > Etiquetas 'y' cargadas desde: 'final'

```
X_train_orig shape: (1976, 14) | y_train_orig shape: (1976,)
```

▼ 3. Manejo del desequilibrio de clases con SMOTE

Aplica SMOTE al conjunto de entrenamiento original para balancear las clases, mostrando las dimensiones del dataset antes y después del remuestreo.

```
# MANEJO DEL DESEQUILIBRIO DE CLASES CON SMOTE
```

```
# 1. Inicializamos SMOTE
# random_state asegura que la generación de muestras sea reproducible
smote = SMOTE(random_state=42)
print("Configuración inicial de clases en entrenamiento:")
print(y_train_orig.value_counts())

# 2. Aplicamos SMOTE SOLO al conjunto de entrenamiento
X_train, y_train = smote.fit_resample(X_train_orig, y_train_orig)

print("\n SMOTE aplicado al conjunto de entrenamiento.")
print("Configuración de clases después de SMOTE:")
print(y_train.value_counts())
print(f"\nNuevas dimensiones de entrenamiento: X_train: {X_train.shape}, y_train: {y_train.shape}")
```

→ Configuración inicial de clases en entrenamiento:

B10	
1.0	1275
0.0	701

```
Name: count, dtype: int64
```

SMOTE aplicado al conjunto de entrenamiento.

Configuración de clases después de SMOTE:

B10

0.0 1275

1.0 1275

```
Name: count, dtype: int64
```

Nuevas dimensiones de entrenamiento: X_train: (2550, 14), y_train: (2550,)

▼ 4. Búsqueda de hiperparámetros con Optuna

Define una función `objective` que entrena un modelo LightGBM y calcula métricas en validación, lanza un estudio de Optuna para optimizar hiperparámetros y muestra los mejores resultados.

```
# BÚSQUEDA DE HIPERPARÁMETROS CON OPTUNA

def objective(trial):
    """
    Función que Optuna intentará optimizar.
    Define el espacio de búsqueda de hiperparámetros.
    """

    # Definimos los rangos de los hiperparámetros a probar
    params = {
        'objective': 'binary',
        'metric': 'binary_logloss',
        'verbosity': -1,
        'boosting_type': 'gbdt',
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        'learning_rate': trial.suggest_float('learning_rate', 0.01, 0.3),
        'num_leaves': trial.suggest_int('num_leaves', 20, 300),
        'max_depth': trial.suggest_int('max_depth', 3, 12),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.0, 1.0), # L1 regularization
        'reg_lambda': trial.suggest_float('reg_lambda', 0.0, 1.0), # L2 regularization
        'subsample': trial.suggest_float('subsample', 0.6, 1.0), # Fracción de datos por
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.6, 1.0), # Fracción
        'is_unbalance': True # Equivalente a class_weight='balanced'
    }

    # Creamos el modelo con los parámetros del trial actual
    model = lgb.LGBMClassifier(**params, random_state=42)

    # Evaluamos el modelo usando validación cruzada sobre el dataset de entrenamiento (co
    # Usamos F1-score ponderado como métrica a maximizar
    score = cross_val_score(model, X_train, y_train, cv=5, scoring='f1_weighted', n_jobs=4)

    return score

# Creamos un "estudio" de Optuna y lo ejecutamos
print(" Iniciando búsqueda de hiperparámetros a fondo con Optuna (100 trials)...")
```

```
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

# Guardamos los mejores parámetros encontrados
best_params_optuna = study.best_params

print("\n Búsqueda finalizada.")
print(f"Mejores hiperparámetros encontrados: {best_params_optuna}")
print(f"Mejor F1-score (weighted) en validación cruzada: {study.best_value:.4f}")
```

Iniciando búsqueda de hiperparámetros a fondo con Optuna (100 trials)...

 Búsqueda finalizada.

Mejores hiperparámetros encontrados: {'n_estimators': 337, 'learning_rate': 0.2967729
Mejor F1-score (weighted) en validación cruzada: 0.7319

✓ 5. Entrenamiento del modelo final con mejores parámetros

Recupera los hiperparámetros óptimos del estudio de Optuna, configura el modelo LightGBM con ellos y entrena sobre todo el conjunto de entrenamiento balanceado.

ENTRENAMIENTO DEL MODELO FINAL CON MEJORES PARÁMETROS

```
# Aseguramos que los parámetros fijos estén presentes
final_params = best_params_optuna.copy()
final_params['objective'] = 'binary'
final_params['metric'] = 'binary_logloss'
final_params['is_unbalance'] = True
```

```
# Creamos y entrenamos el modelo final
```

```
lgbm_model = lgb.LGBMClassifier(**final_params, random_state=42)
lgbm_model.fit(X_train, y_train)
```

```
print("Modelo final de LightGBM entrenado y listo para evaluación.")
```

✓ 6. Guardado del modelo final

Serializa el modelo LightGBM entrenado (con `joblib`) y guarda el objeto en disco para su posterior despliegue o análisis.

```
# GUARDADO DEL MODELO FINAL

# Prefijo para identificar el archivo de este notebook
PREFIX = "04_3_"

# --- ACTUALIZADO: Guardar el modelo en la carpeta de artefactos experimentales ---
model_path = EXP_ARTIFACTS_DIR / f'{PREFIX}lgbm_model.pkl'

# Guardamos el modelo entrenado (lgbm_model)
joblib.dump(lgbm_model, model_path)
```

```
print(f" Modelo LightGBM guardado con éxito en: {model_path}")
```

→ ✓ Modelo LightGBM guardado con éxito en: /content/drive/MyDrive/Digitech/TFG/ML/Cal

▼ 7. Evaluación exhaustiva del modelo

Evaluá el modelo entrenado en los conjuntos de validación y prueba, calculando métricas clave (accuracy, F1, AUC), mostrando la matriz de confusión y el reporte de clasificación.

```
# EVALUACIÓN EXHAUSTIVA DEL MODELO
```

```
# No necesitamos un pipeline ya que LightGBM no requiere escalado de datos
# y ya hemos aplicado SMOTE manualmente.
def evaluate_model(model, X, y, dataset_name):
    print(f"--- Evaluación en el conjunto de {dataset_name} ---")
    y_pred = model.predict(X)
    y_pred_proba = model.predict_proba(X)[:, 1]

    accuracy = accuracy_score(y, y_pred)
    f1 = f1_score(y, y_pred, average='weighted')
    auc = roc_auc_score(y, y_pred_proba)

    print(f"Accuracy: {accuracy:.4f}")
    print(f"F1-Score (Weighted): {f1:.4f}")
    print(f"AUC: {auc:.4f}")
    print("\nReporte de Clasificación:")
    print(classification_report(y, y_pred, target_names=['Bajo/Medio Riesgo (0)', 'Alto R

    print("Matriz de Confusión:")
    cm = confusion_matrix(y, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Bajo/Medio', 'Alt
    disp.plot(cmap=plt.cm.Blues)
    plt.title(f'Matriz de Confusión - {dataset_name}')
    plt.show()

# Evaluar en el conjunto de validación
evaluate_model(lgbm_model, X_val, y_val, "Validación")

print("\n" + "="*60 + "\n")

# Evaluar en el conjunto de prueba
evaluate_model(lgbm_model, X_test, y_test, "Prueba")
```

→ --- Evaluación en el conjunto de Validación ---

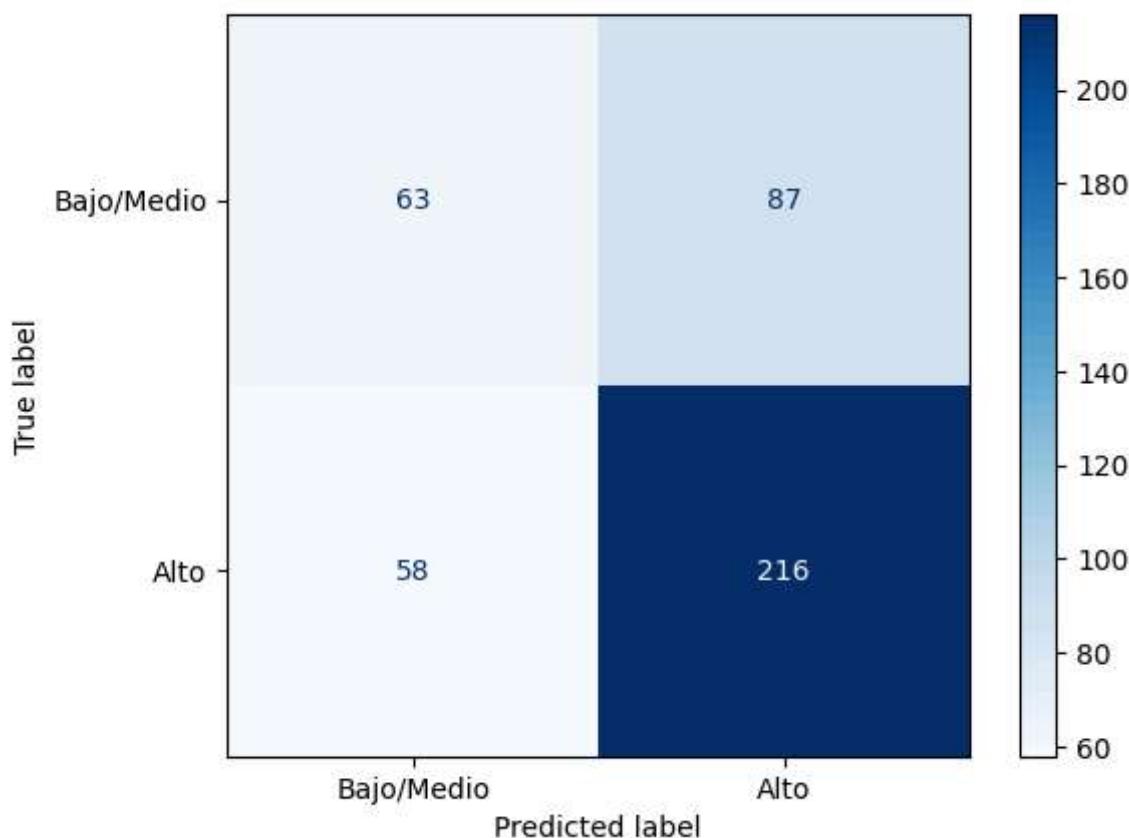
Accuracy: 0.6580
 F1-Score (Weighted): 0.6483
 AUC: 0.6566

Reporte de Clasificación:

	precision	recall	f1-score	support
Bajo/Medio Riesgo (0)	0.52	0.42	0.46	150
Alto Riesgo (1)	0.71	0.79	0.75	274
accuracy			0.66	424
macro avg	0.62	0.60	0.61	424
weighted avg	0.64	0.66	0.65	424

Matriz de Confusión:

Matriz de Confusión - Validación



=====

--- Evaluación en el conjunto de Prueba ---

Accuracy: 0.6651
 F1-Score (Weighted): 0.6552
 AUC: 0.6646

Reporte de Clasificación:

	precision	recall	f1-score	support
Bajo/Medio Riesgo (0)	0.53	0.43	0.47	150
Alto Riesgo (1)	0.72	0.80	0.75	274
accuracy			0.67	424
macro avg	0.63	0.61	0.61	424
weighted avg	0.65	0.67	0.66	424

✓ 8^{Matriz de Confusión:} Ajuste avanzado del umbral de decisión (LightGBM)

Matriz de Confusión - Prueba

Explora distintos umbrales de probabilidad en el set de validación para maximizar métricas específicas (p. ej., F1 o AUC) y selecciona el umbral óptimo.

- 200

```
# AJUSTE AVANZADO DEL UMBRAL DE DECISIÓN (LIGHTGBM)

# El código es idéntico al anterior, solo cambia la variable del modelo

# 1. Obtener las probabilidades para el conjunto de VALIDACIÓN
y_val_probs = lgbm_model.predict_proba(X_val)[:, 1]

# 2. Definir un rango de umbrales para probar
thresholds = np.linspace(0.1, 0.9, 81)

# 3. Listas para almacenar las métricas en cada umbral
f1_scores_0 = []
recall_scores_0 = []
precision_scores_0 = []
recall_scores_1 = []

# 4. Iterar y calcular métricas
for t in thresholds:
    y_val_pred_t = (y_val_probs >= t).astype(int)
    f1_scores_0.append(f1_score(y_val, y_val_pred_t, pos_label=0, zero_division=0))
    recall_scores_0.append(recall_score(y_val, y_val_pred_t, pos_label=0, zero_division=0))
    precision_scores_0.append(precision_score(y_val, y_val_pred_t, pos_label=0, zero_division=0))
    recall_scores_1.append(recall_score(y_val, y_val_pred_t, pos_label=1, zero_division=0))

# 5. Visualizar los resultados
plt.style.use('seaborn-v0_8-whitegrid')
fig, ax = plt.subplots(figsize=(12, 7))

ax.plot(thresholds, f1_scores_0, label='F1-Score (Clase 0)', color='darkorange', lw=3)
ax.plot(thresholds, recall_scores_0, label='Recall (Clase 0)', color='crimson', lw=2, linestyles='--')
ax.plot(thresholds, precision_scores_0, label='Precision (Clase 0)', color='darkred', lw=2, linestyles='-.')
ax.plot(thresholds, recall_scores_1, label='Recall (Clase 1)', color='royalblue', lw=2, linestyles=':')

# 6. Encontrar y marcar el umbral óptimo
optimal_idx = np.argmax(f1_scores_0)
optimal_threshold = thresholds[optimal_idx]

ax.axvline(optimal_threshold, color='green', linestyle='--', lw=2, label=f'Umbraal Óptimo')
ax.axvline(0.5, color='black', linestyle=':', label='Umbraal por Defecto (0.5)')
ax.set_title('Métricas de Clasificación vs. Umbral de Decisión (LightGBM)', fontsize=16)
ax.set_xlabel('Umbraal', fontsize=12)
ax.set_ylabel('Puntuación de la Métrica', fontsize=12)
ax.legend(loc='best', fontsize=11)
plt.show()

print(f" Objetivo: Maximizar el F1-Score para la Clase 0 (Bajo/Medio Riesgo)")
print(f"Umbraal óptimo encontrado: {optimal_threshold:.4f}")
print(f"Con este umbral, el F1-Score para la Clase 0 en validación es: {f1_scores_0[optimal_idx]}
```



Métricas de Clasificación vs. Umbral de Decisión (LightGBM)

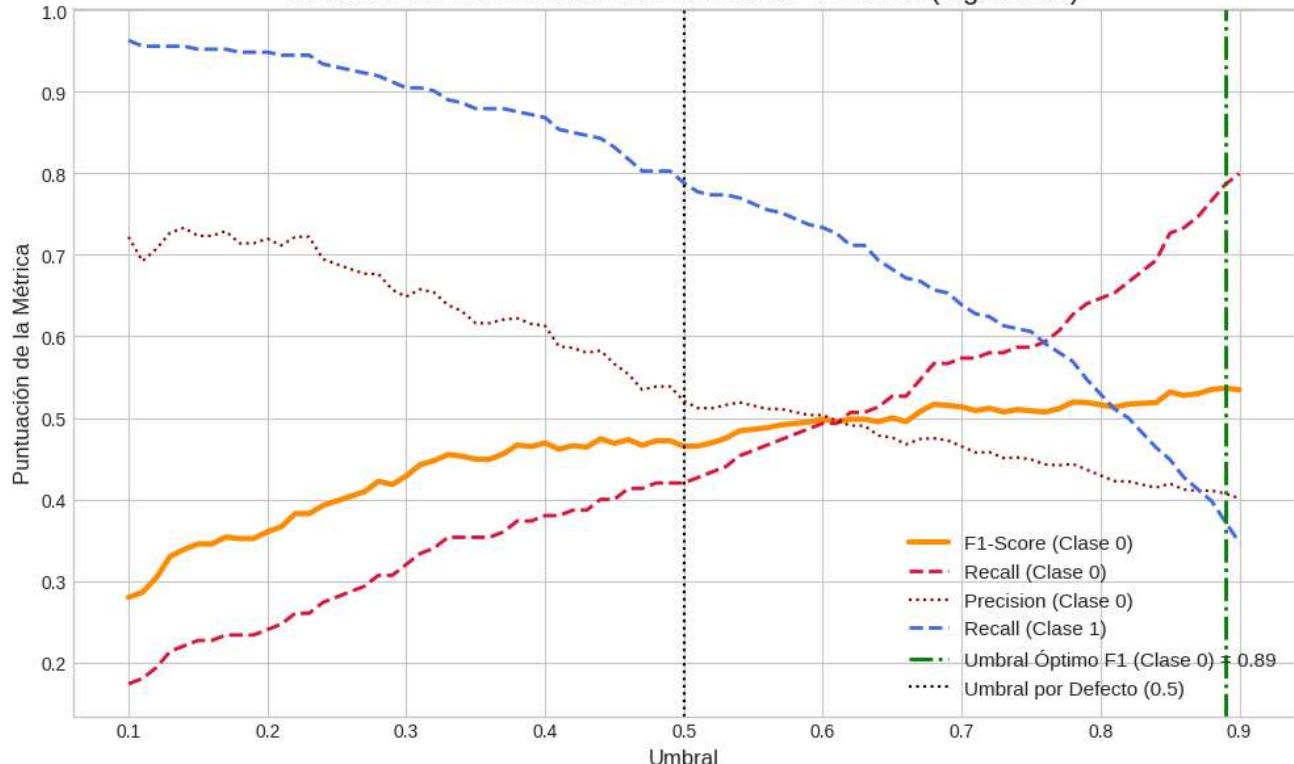


grafico Objetivo: Maximizar el F1-Score para la Clase 0 (Bajo/Medio Riesgo)

Umbral óptimo encontrado: 0.8900

Con este umbral, el F1-Score para la Clase 0 en validación es: 0.5364

▼ 9. Evaluación final con umbral optimizado

Aplica el umbral seleccionado al conjunto de prueba, imprime las métricas obtenidas y muestra la matriz de confusión resultante.

```
# EVALUACIÓN FINAL CON UMBRAL OPTIMIZADO (LIGHTGBM)

print(f"--- Evaluación en el conjunto de Prueba (Umbral optimizado {optimal_threshold}:")

# Obtener probabilidades para el conjunto de prueba
y_test_probs = lgbm_model.predict_proba(X_test)[:, 1]

# Aplicar el umbral óptimo para obtener las nuevas predicciones
y_test_pred_optimal = (y_test_probs >= optimal_threshold).astype(int)

# Calcular y mostrar el nuevo reporte y matriz de confusión
print("Nuevo Reporte de Clasificación con Umbral Optimizado:")
print(classification_report(y_test, y_test_pred_optimal, target_names=['Bajo/Medio Riesgo', 'Alto Riesgo']))
```

```
print("Nueva Matriz de Confusión con Umbral Optimizado:")
cm = confusion_matrix(y_test, y_test_pred_optimal)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Bajo/Medio', 'Alto'])
disp.plot(cmap=plt.cm.Blues)
plt.title(f'Matriz de Confusión - Prueba (Umbral {optimal_threshold:.2f})')
plt.show()
```

→ --- Evaluación en el conjunto de Prueba (Umbral optimizado 0.89) ---

Nuevo Reporte de Clasificación con Umbral Optimizado:

	precision	recall	f1-score	support
Bajo/Medio Riesgo (0)	0.41	0.79	0.54	150
Alto Riesgo (1)	0.77	0.37	0.50	274
accuracy			0.52	424
macro avg	0.59	0.58	0.52	424
weighted avg	0.64	0.52	0.51	424

Nueva Matriz de Confusión con Umbral Optimizado:

Matriz de Confusión - Prueba (Umbral 0.89)

