

✓ 06.0_Modelado_Comparacion.ipynb

Objetivo

Evaluar y comparar varios conjuntos de características (originales, seleccionados y con ingeniería de variables) mediante Random Forest optimizado por GridSearchCV. El objetivo es identificar el dataset y la configuración más efectivos para problemas de clasificación de 4, 3 y 2 clases, basándose en métricas de negocio (coste) y estadísticas (F1, AUC, Recall).

Entradas (Inputs)

Splits Finales (desde `data/splits/final/`)

- `y_train.parquet`, `y_val.parquet`, `y_test.parquet`
- `X_train.parquet`, `X_val.parquet`, `X_test.parquet` (Dataset 95_Original)

Splits de Experimentos (desde `data/splits/experiments/`)

- `X_train_17.parquet`, `X_val_17.parquet`, `X_test_17.parquet` (Dataset 17_Optimizado)

Datos con Ingeniería de Características Finales (desde `data/engineered/final/`)

- `X_train_95_ultimate_eng.parquet`, `X_val_95_ultimate_eng.parquet`,
`X_test_95_ultimate_eng.parquet` (Dataset 95_Ultimate)

Datos con Ingeniería de Características de Experimentos (desde `data/engineered/experiments/`)

- `X_train_14_eng.parquet`, `X_val_14_eng.parquet`, `X_test_14_eng.parquet` (Dataset 14_Eng)
- `X_train_45_eng.parquet`, `X_val_45_eng.parquet`, `X_test_45_eng.parquet` (Dataset 45_Eng)
- `X_train_95_eng.parquet`, `X_val_95_eng.parquet`, `X_test_95_eng.parquet` (Dataset 95_Eng)

Salidas (Outputs)

No genera archivos de salida. Los resultados (métricas y tablas comparativas) se muestran directamente en el notebook.

Resumen Ejecutivo

- Este notebook compara seis conjuntos de características (original de 95 variables, 17 optimizado, 14/45/95 variables con ingeniería y un conjunto “Ultimate”) para modelar riesgo de crédito en escenarios de 4 clases, 3 clases y binario.
- Se implementa un pipeline con SMOTE, escalado y RandomForest, ajustado por GridSearchCV con validación cruzada estratificada (3 folds).
- Se definen matrices de coste específicas para 3 y 4 clases y se optimizan umbrales para maximizar el F1 en la detección de alto riesgo.
- Se evalúan métricas clave: F1-macro, AUC, recalls por clase y coste medio de negocio, almacenando todo en un DataFrame comparativo.
- En clasificación binaria, el conjunto 95_Ultimate logra AUC 0.7730, recall protector 0.6533 y recall detector 0.7810.
- En el problema de 3 clases, 95_Ultimate alcanza F1-macro 0.5477 y coste medio 0.5142, superando al original (F1 0.5307, coste 0.5613).
- Para el caso de 4 clases, el set original obtiene F1-macro 0.4656 (ligeramente superior) pero a un coste medio mayor (0.7594 vs. 0.7736 de Ultimate).
- La interfaz interactiva final permite explorar dinámicamente estos resultados por cada conjunto de variables.

▼ 1. Preparar entorno de trabajo e importaciones

Monta Google Drive en Colab, añade la ruta raíz del proyecto a `sys.path`, importa librerías estándar (pandas, numpy, joblib, etc.), de modelado (scikit-learn) y carga las rutas de configuración desde `config.py`.

```
# Montar Drive, Cargar Librerías y Cargar Configuración

from google.colab import drive
import sys
import pandas as pd
import numpy as np
import joblib
import warnings
import json
from pathlib import Path

# Modelado y Ensamblado
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold

# Balanceo de Clases
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE

# Evaluación
```

```

from sklearn.metrics import accuracy_score, f1_score, roc_auc_score, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
import ipywidgets as widgets
from IPython.display import display, clear_output
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
import numpy as np
from sklearn.metrics import f1_score

# --- 1. Montar Google Drive ---
drive.mount('/content/drive', force_remount=True)
warnings.filterwarnings('ignore')

# --- 2. Añadir la raíz del proyecto al path para importar el módulo config ---
# Asegúrate de que esta ruta coincida con la ubicación del proyecto en drive
ROOT_PATH_STR = '/content/drive/MyDrive/TFM-AntonioEsquinas'
if ROOT_PATH_STR not in sys.path:
    sys.path.append(ROOT_PATH_STR)

# --- 3. Importar las rutas necesarias desde el archivo de configuración ---
from config import (
    FINAL_SPLITS_DIR,
    EXP_SPLITS_DIR,
    FINAL_ENGINEERED_DATA_DIR,
    EXP_ENGINEERED_DATA_DIR
)

print("Entorno preparado y configuración de rutas cargada desde config.py.")

```

→ Mounted at /content/drive
 Entorno preparado y configuración de rutas cargada desde config.py.

▼ 2. Cargar datasets y targets para la competición

Define los diccionarios con las rutas a los distintos conjuntos de características, carga los DataFrames de entrenamiento, validación y prueba junto con sus etiquetas originales, y genera versiones remapeadas de los targets (4, 3 y 2 clases).

```

# Cargar los Datasets para la Competición Final

# --- 1: Definición de los competidores usando las rutas de config.py ---
feature_sets_to_load = {
    '95_Original': {
        'train': FINAL_SPLITS_DIR / 'X_train.parquet',
        'val': FINAL_SPLITS_DIR / 'X_val.parquet',
        'test': FINAL_SPLITS_DIR / 'X_test.parquet'
    },
}

```

```

'17_Optimizado': {
    'train': EXP_SPLITS_DIR / 'X_train_17.parquet',
    'val': EXP_SPLITS_DIR / 'X_val_17.parquet',
    'test': EXP_SPLITS_DIR / 'X_test_17.parquet'
},
'14_Eng': {
    'train': EXP_ENGINEERED_DATA_DIR / 'X_train_14_eng.parquet',
    'val': EXP_ENGINEERED_DATA_DIR / 'X_val_14_eng.parquet',
    'test': EXP_ENGINEERED_DATA_DIR / 'X_test_14_eng.parquet'
},
'45_Eng': {
    'train': EXP_ENGINEERED_DATA_DIR / 'X_train_45_eng.parquet',
    'val': EXP_ENGINEERED_DATA_DIR / 'X_val_45_eng.parquet',
    'test': EXP_ENGINEERED_DATA_DIR / 'X_test_45_eng.parquet'
},
'95_Eng': {
    'train': EXP_ENGINEERED_DATA_DIR / 'X_train_95_eng.parquet',
    'val': EXP_ENGINEERED_DATA_DIR / 'X_val_95_eng.parquet',
    'test': EXP_ENGINEERED_DATA_DIR / 'X_test_95_eng.parquet'
},
'95_Ultimate': {
    'train': FINAL_ENGINEERED_DATA_DIR / 'X_train_95_ultimate.parquet',
    'val': FINAL_ENGINEERED_DATA_DIR / 'X_val_95_ultimate.parquet',
    'test': FINAL_ENGINEERED_DATA_DIR / 'X_test_95_ultimate.parquet'
},
}

print("Cargando los siguientes conjuntos de características:")
for name in feature_sets_to_load.keys():
    print(f" > {name}")

# --- 2: Funciones de remapeo y carga de los targets desde la carpeta final ---
def remap_to_3(y): return y.map({1.0: 1.0, 2.0: 1.0, 3.0: 2.0, 4.0: 3.0})
def remap_to_2(y): return y.map({1.0: 0.0, 2.0: 0.0, 3.0: 1.0, 4.0: 1.0})

y_train_orig = pd.read_parquet(FINAL_SPLITS_DIR / 'y_train.parquet').squeeze()
y_val_orig = pd.read_parquet(FINAL_SPLITS_DIR / 'y_val.parquet').squeeze()
y_test_orig = pd.read_parquet(FINAL_SPLITS_DIR / 'y_test.parquet').squeeze()

targets = {
    4: {'train': y_train_orig, 'val': y_val_orig, 'test': y_test_orig},
    3: {'train': remap_to_3(y_train_orig), 'val': remap_to_3(y_val_orig), 'test': remap_to_3(y_test_orig)},
    2: {'train': remap_to_2(y_train_orig), 'val': remap_to_2(y_val_orig), 'test': remap_to_2(y_test_orig)}
}

print("\nDatasets y targets listos para la evaluación.")

→ Cargando los siguientes conjuntos de características:
> 95_Ori...
> 17_Optimizado
> 14_Eng
> 45_Eng
> 95_Eng
> 95_Ultimate

```

Datasets y targets listos para la evaluación.

3. Definir plantillas base, parrillas de hiperparámetros y matrices de coste

Configura una plantilla de pipeline con escalado, SMOTE y clasificador, establece las grillas de hiperparámetros para optimización y crea las matrices de costes para los casos de clasificación multiclas.

```
# Plantillas, Parrillas y Funciones

# --- 1: Plantilla Base del Pipeline ---
base_pipeline = ImbPipeline([
    ('scaler', StandardScaler()),
    ('smote', SMOTE(random_state=42)),
    ('clf', RandomForestClassifier(class_weight='balanced', random_state=42, n_jobs=-1))
])

# --- 2: Parrilla de Hiperparámetros ---
param_grid = {
    'clf__n_estimators': [100, 200],
    'clf__max_depth': [10, 20],
    'clf__min_samples_split': [5, 10],
    'clf__min_samples_leaf': [2, 4]
}

# --- 3: Funciones de Coste para 3 y 4 Clases ---
cost_matrix_3_clases = np.array([[0, 1, 10], [1, 0, 2], [10, 2, 0]])
cost_matrix_4_clases = np.array([[0, 1, 8, 25], [1, 0, 1, 8], [8, 1, 0, 1], [25, 8, 1, 0]])

def cost_function_3_clases(y_true, y_pred):
    total_cost = 0
    for true_label, pred_label in zip(y_true, y_pred):
        total_cost += cost_matrix_3_clases[int(true_label) - 1, int(pred_label) - 1]
    return total_cost / len(y_true)

def cost_function_4_clases(y_true, y_pred):
    total_cost = 0
    for true_label, pred_label in zip(y_true, y_pred):
        total_cost += cost_matrix_4_clases[int(true_label) - 1, int(pred_label) - 1]
    return total_cost / len(y_true)

# --- 4: Funciones de Umbral (Completas) ---
def find_protector_threshold(pipe, X_val, y_val):
    """Encuentra el umbral que maximiza el F1-Score para la CLASE 0 (Bajo Riesgo)."""
    prob_val = pipe.predict_proba(X_val)[:, 1]
    best_f1_0, best_t = 0, 0.5
    for t in np.linspace(0.1, 0.9, 21):
        y_pred_t = (prob_val >= t).astype(int)
        f1_t_0 = f1_score(y_val, y_pred_t, pos_label=0, zero_division=0)
```

```

    if f1_t_0 > best_f1_0:
        best_f1_0, best_t = f1_t_0, t
    return best_t

def find_detector_threshold(pipe, X_val, y_val):
    """Encuentra el umbral que maximiza el F1-Score para la CLASE 1 (Alto Riesgo)."""
    prob_val = pipe.predict_proba(X_val)[:, 1]
    best_f1_1, best_t = 0, 0.5
    for t in np.linspace(0.1, 0.9, 21):
        y_pred_t = (prob_val >= t).astype(int)
        f1_t_1 = f1_score(y_val, y_pred_t, pos_label=1, zero_division=0)
        if f1_t_1 > best_f1_1:
            best_f1_1, best_t = f1_t_1, t
    return best_t

```

print(" Plantillas, parrillas y funciones de evaluación (incluyendo cuerpos de función) d

→ Plantillas, parrillas y funciones de evaluación (incluyendo cuerpos de función) d



▼ 4. Ejecutar bucle de competición con validación cruzada

Itera sobre cada conjunto de características, aplica validación cruzada con GridSearch para ajustar los modelos según métricas como F1 y AUC, evalúa en validación y test, y almacena los resultados en una lista.

```

# Bucle de Competición

import pandas as pd
import json
from sklearn.metrics import f1_score, roc_auc_score, classification_report
from sklearn.model_selection import StratifiedKFold
from imblearn.pipeline import Pipeline # Asegurarse de que esté importado

all_results = []

# Estrategia de validación cruzada para el GridSearch
cv_strategy = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# --- Bucle principal sobre cada CONJUNTO DE CARACTERÍSTICAS ---
for name, paths in feature_sets_to_load.items():
    print(f"\n--- Procesando Dataset: {name} ---")
    results_row = {'Dataset': name}

    try:
        X_train = pd.read_parquet(paths['train'])
        X_val = pd.read_parquet(paths['val'])
        X_test = pd.read_parquet(paths['test'])
        X_full_train = pd.concat([X_train, X_val], ignore_index=True)
    except FileNotFoundError as e:

```

```

print(f"  Error: No se pudo cargar el archivo para '{name}'. Archivo no encontrado")
continue

# ===== Experimento para 4 Clases =====
print(" 1. Optimizando para 4 clases...")
y_train_4 = targets[4]['train']
y_full_train_4 = pd.concat([y_train_4, targets[4]['val']], ignore_index=True)

grid4 = GridSearchCV(estimator=base_pipeline, param_grid=param_grid, cv=cv_strategy,
grid4.fit(X_train, y_train_4)
results_row['Best Params (4C)'] = json.dumps(grid4.best_params_)

final_pipe4 = ImbPipeline(steps=grid4.best_estimator_.steps)
final_pipe4.fit(X_full_train, y_full_train_4)
y_pred4 = final_pipe4.predict(X_test)
results_row['F1-macro (4 Clases)'] = f1_score(targets[4]['test'], y_pred4, average='macro')
results_row['Coste Medio (4 Clases)'] = cost_function_4_clases(targets[4]['test'], y_pred4)

# ===== Experimento para 3 Clases =====
print(" 2. Optimizando para 3 clases...")
y_train_3 = targets[3]['train']
y_full_train_3 = pd.concat([y_train_3, targets[3]['val']], ignore_index=True)

grid3 = GridSearchCV(estimator=base_pipeline, param_grid=param_grid, cv=cv_strategy,
grid3.fit(X_train, y_train_3)
results_row['Best Params (3C)'] = json.dumps(grid3.best_params_)

final_pipe3 = ImbPipeline(steps=grid3.best_estimator_.steps)
final_pipe3.fit(X_full_train, y_full_train_3)
y_pred3 = final_pipe3.predict(X_test)
results_row['F1-macro (3 Clases)'] = f1_score(targets[3]['test'], y_pred3, average='macro')
results_row['Coste Medio (3 Clases)'] = cost_function_3_clases(targets[3]['test'], y_pred3)

# ===== Experimento para 2 Clases =====
print(" 3. Optimizando para 2 clases...")
y_train_2 = targets[2]['train']
y_full_train_2 = pd.concat([y_train_2, targets[2]['val']], ignore_index=True)

grid2 = GridSearchCV(estimator=base_pipeline, param_grid=param_grid, cv=cv_strategy,
grid2.fit(X_train, y_train_2)
results_row['Best Params (2C)'] = json.dumps(grid2.best_params_)

final_pipe2 = ImbPipeline(steps=grid2.best_estimator_.steps)
final_pipe2.fit(X_full_train, y_full_train_2)
y_pred2_proba = final_pipe2.predict_proba(X_test)[:, 1]
results_row['AUC (2 Clases)'] = roc_auc_score(targets[2]['test'], y_pred2_proba)

t_protector = find_protector_threshold(final_pipe2, X_val, targets[2]['val'])
y_pred2_protector = (y_pred2_proba >= t_protector).astype(int)
results_row['Recall Protector (Clase 0)'] = classification_report(targets[2]['test'], y_pred2_protector)

t_detector = find_detector_threshold(final_pipe2, X_val, targets[2]['val'])
y_pred2_detector = (y_pred2_proba >= t_detector).astype(int)
results_row['Recall Detector (Clase 1)'] = classification_report(targets[2]['test'], y_pred2_detector)

```

```
all_results.append(results_row)

print("\n\n Todos los experimentos (con optimización) han finalizado.")

→
--- 🚀 Procesando Dataset: 95_Original ---
1. Optimizando para 4 clases...
2. Optimizando para 3 clases...
3. Optimizando para 2 clases...

--- 🚀 Procesando Dataset: 17_Optimizado ---
1. Optimizando para 4 clases...
2. Optimizando para 3 clases...
3. Optimizando para 2 clases...

--- 🚀 Procesando Dataset: 14_Eng ---
1. Optimizando para 4 clases...
2. Optimizando para 3 clases...
3. Optimizando para 2 clases...

--- 🚀 Procesando Dataset: 45_Eng ---
1. Optimizando para 4 clases...
2. Optimizando para 3 clases...
3. Optimizando para 2 clases...

--- 🚀 Procesando Dataset: 95_Eng ---
1. Optimizando para 4 clases...
2. Optimizando para 3 clases...
3. Optimizando para 2 clases...

--- 🚀 Procesando Dataset: 95_Ultimate ---
1. Optimizando para 4 clases...
2. Optimizando para 3 clases...
3. Optimizando para 2 clases...
```

Todos los experimentos (con optimización) han finalizado.

▼ 5. Crear tablas comparativas de resultados y costes

Convierte la lista de resultados en un DataFrame, reordena los índices para facilitar la comparación y prepara tablas que muestran los mejores hiperparámetros y los nuevos costes para cada dataset.

```
# Tablas Comparativas con Hiperparámetros y Nuevos Costes

import pandas as pd

# Convertir la lista de resultados en un DataFrame
results_df = pd.DataFrame(all_results).set_index('Dataset')

# --- Reordenar las filas para una mejor comparación ---
# Esta lista debe coincidir con los datasets que has procesado en la Celda 4
ordered_index = [
```

```
'17_Optimizado',
'95_Original',
'14_Eng',
'45_Eng',
'95_Eng',
'95_Ultimate',
]
# Esta línea se asegura de que no haya errores si falta algún resultado
ordered_index_existing = [idx for idx in ordered_index if idx in results_df.index]
results_df = results_df.reindex(ordered_index_existing)

# --- Tabla 1: Problema de 4 Clases (Precisión y Coste) ---
print("--- Tabla 1: Resultados para el Problema de 4 Clases ---")
display(results_df[['F1-macro (4 Clases)', 'Coste Medio (4 Clases)', 'Best Params (4C)']]
       .style.highlight_max(subset=['F1-macro (4 Clases)', color='#a3e635')
       .highlight_min(subset=['Coste Medio (4 Clases)', color='#a3e635')
       .format('{:.4f}', subset=['F1-macro (4 Clases)', 'Coste Medio (4 Clases)'])
       .set_properties(**{'text-align': 'left'}))

# --- Tabla 2: Problema de 3 Clases (Utilidad de Negocio) ---
print("\n--- Tabla 2: Resultados para el Problema de 3 Clases ---")
display(results_df[['F1-macro (3 Clases)', 'Coste Medio (3 Clases)', 'Best Params (3C)']]
       .style.highlight_max(subset=['F1-macro (3 Clases)', color='#a3e635')
       .highlight_min(subset=['Coste Medio (3 Clases)', color='#a3e635')
       .format({'F1-macro (3 Clases)': '{:.4f}', 'Coste Medio (3 Clases)': '{:.4f}'})
       .set_properties(**{'text-align': 'left'}))

# --- Tabla 3: Problema Binario (Estrategias Especializadas) ---
print("\n--- Tabla 3: Resultados para el Problema Binario ---")
binary_cols = [
    'AUC (2 Clases)',
    'Recall Protector (Clase 0)',
    'Recall Detector (Clase 1)',
    'Best Params (2C)'
]
display(results_df[binary_cols]
       .style.highlight_max(subset=['AUC (2 Clases)', 'Recall Protector (Clase 0)', 'Recall Detector (Clase 1)', 'Best Params (2C)', color='#a3e635')
       .format('{:.4f}', subset=['AUC (2 Clases)', 'Recall Protector (Clase 0)', 'Recall Detector (Clase 1)', 'Best Params (2C)'])
       .set_properties(**{'text-align': 'left'}))
```

→ ---  Tabla 1: Resultados para el Problema de 4 Clases ---

| | F1-macro (4 Clases) | Coste Medio (4 Clases) | Best Params (4C) |
|----------------------|------------------------|---------------------------|--|
| Dataset | | | |
| 17_Optimizado | 0.4235 | 1.1226 | {"clf__max_depth": 10, "clf__min_samples_leaf": 4, "clf__min_samples_split": 5, "clf__n_estimators": 200} |
| 95_Original | 0.4656 | 0.7594 | {"clf__max_depth": 10, "clf__min_samples_leaf": 4, "clf__min_samples_split": 10, "clf__n_estimators": 200} |
| 14_Eng | 0.4211 | 0.9858 | {"clf__max_depth": 10, "clf__min_samples_leaf": 2, "clf__min_samples_split": 5, "clf__n_estimators": 200} |
| 45_Eng | 0.4038 | 1.2382 | {"clf__max_depth": 10, "clf__min_samples_leaf": 4, "clf__min_samples_split": 10, "clf__n_estimators": 100} |
| 95_Eng | 0.4482 | 0.7429 | {"clf__max_depth": 20, "clf__min_samples_leaf": 4, "clf__min_samples_split": 10, "clf__n_estimators": 200} |
| 95_Ultimate | 0.4640 | 0.7736 | {"clf__max_depth": 10, "clf__min_samples_leaf": 2, "clf__min_samples_split": 10, "clf__n_estimators": 100} |

---  Tabla 2: Resultados para el Problema de 3 Clases ---

| | F1-macro (3 Clases) | Coste Medio (3 Clases) | Best Params (3C) |
|----------------------|------------------------|---------------------------|--|
| Dataset | | | |
| 17_Optimizado | 0.4577 | 0.9033 | {"clf__max_depth": 10, "clf__min_samples_leaf": 4, "clf__min_samples_split": 10, "clf__n_estimators": 100} |
| 95_Original | 0.5307 | 0.5613 | {"clf__max_depth": 10, "clf__min_samples_leaf": 4, "clf__min_samples_split": 10, "clf__n_estimators": 200} |
| 14_Eng | 0.4921 | 0.7099 | {"clf__max_depth": 10, "clf__min_samples_leaf": 4, "clf__min_samples_split": 10, "clf__n_estimators": 200} |
| 45_Eng | 0.4544 | 0.7618 | {"clf__max_depth": 20, "clf__min_samples_leaf": 4, "clf__min_samples_split": 10, "clf__n_estimators": 100} |

✓ 6. Implementar interfaz interactiva para análisis detallado

Crea un widget desplegable para seleccionar un dataset, define la función que muestra la matriz de confusión y el reporte de clasificación, vincula el widget a esa función y despliega el informe inicial por defecto.

```
# Análisis Detallado con Matriz en Texto
```

```
import ipywidgets as widgets
from IPython.display import display, clear_output
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import json

# Crear el menú desplegable con los datasets que compitieron
dataset_selector = widgets.Dropdown(
    options=results_df.index.tolist(),
    description='Dataset:',
    value=results_df.index[0]
)

# Crear un área de salida para los resultados
output_area = widgets.Output()

def show_detailed_report(dataset_name):
    """
    Esta función se ejecuta cada vez que se cambia el valor del menú desplegable.
    Entrena el mejor modelo para el dataset seleccionado y muestra un informe detallado.
    """
    with output_area:
        clear_output(wait=True) # Limpiar la salida anterior
        print(f"--- Análisis Detallado para: {dataset_name} ---\n")

        # --- 1. Cargar datos ---
        paths = feature_sets_to_load[dataset_name]
        X_train = pd.read_parquet(paths['train'])
        X_val = pd.read_parquet(paths['val'])
        X_test = pd.read_parquet(paths['test'])
        X_full_train = pd.concat([X_train, X_val], ignore_index=True)

        # --- 2. Análisis para 4 Clases ---
        print("\n--- Problema de 4 Clases ---")
        y_full_train_4 = pd.concat([targets[4]['train'], targets[4]['val']], ignore_index=True)
        params_4c = json.loads(results_df.loc[dataset_name, 'Best Params (4C)'])
        pipe4 = ImbPipeline(steps=base_pipeline.steps)
        pipe4.set_params(**params_4c)
        pipe4.fit(X_full_train, y_full_train_4)
        y_pred4 = pipe4.predict(X_test)

        print(classification_report(targets[4]['test'], y_pred4, zero_division=0))
        cm4 = confusion_matrix(targets[4]['test'], y_pred4, labels=pipe4.classes_)

        print("\nMatriz de Confusión (Formato Texto):")
        print(cm4)

        plt.figure(figsize=(6, 4))
        sns.heatmap(cm4, annot=True, fmt='d', cmap='Blues', xticklabels=pipe4.classes_, y
        plt.title('Matriz de Confusión (4 Clases)')
        plt.ylabel('Clase Verdadera')
        plt.xlabel('Clase Predicha')
```

```

plt.show()

# --- 3. Análisis para 3 Clases ---
print("\n--- Problema de 3 Clases ---")
y_full_train_3 = pd.concat([targets[3]['train'], targets[3]['val']], ignore_index=True)
params_3c = json.loads(results_df.loc[dataset_name, 'Best Params (3C)'])
pipe3 = ImbPipeline(steps=base_pipeline.steps)
pipe3.set_params(**params_3c)
pipe3.fit(X_full_train, y_full_train_3)
y_pred3 = pipe3.predict(X_test)

print(classification_report(targets[3]['test'], y_pred3, zero_division=0))
cm3 = confusion_matrix(targets[3]['test'], y_pred3, labels=pipe3.classes_)

print("\nMatriz de Confusión (Formato Texto):")
print(cm3)

plt.figure(figsize=(5, 3))
sns.heatmap(cm3, annot=True, fmt='d', cmap='Greens', xticklabels=pipe3.classes_, yticklabels=pipe3.classes_)
plt.title('Matriz de Confusión (3 Clases)')
plt.ylabel('Clase Verdadera')
plt.xlabel('Clase Predicha')
plt.show()

# --- 4. Análisis para 2 Clases ---
print("\n--- Problema de 2 Clases ---")
y_full_train_2 = pd.concat([targets[2]['train'], targets[2]['val']], ignore_index=True)
params_2c = json.loads(results_df.loc[dataset_name, 'Best Params (2C)'])
pipe2 = ImbPipeline(steps=base_pipeline.steps)
pipe2.set_params(**params_2c)
pipe2.fit(X_full_train, y_full_train_2)
y_pred2 = pipe2.predict(X_test)

print(classification_report(targets[2]['test'], y_pred2, zero_division=0))
cm2 = confusion_matrix(targets[2]['test'], y_pred2, labels=pipe2.classes_)

print("\nMatriz de Confusión (Formato Texto):")
print(cm2)

plt.figure(figsize=(4, 2))
sns.heatmap(cm2, annot=True, fmt='d', cmap='Reds', xticklabels=pipe2.classes_, yticklabels=pipe2.classes_)
plt.title('Matriz de Confusión (2 Clases)')
plt.ylabel('Clase Verdadera')
plt.xlabel('Clase Predicha')
plt.show()

# --- 5. Vincular la función al menú y mostrar ---
def on_change(change):
    if change['type'] == 'change' and change['name'] == 'value':
        show_detailed_report(change['new'])

dataset_selector.observe(on_change, names='value')

print("Selecciona un dataset del menú para ver su informe detallado:")
display(dataset_selector, output_area)

```

```
# Cargar el primer informe por defecto al ejecutar la celda  
show_detailed_report(dataset_selector.value)
```

→ Selecciona un dataset del menú para ver su informe detallado:

Dataset: 17_Optimizado
Conclusiones Finales
 Análisis Detallado para: 17_Optimizado ---

- **Ingeniería de características decisiva:** El conjunto 95_Ultimate equilibra mejor precisión y coste, liderando en 3-clases (F1 0.5477, coste 0.5142) y en binario (AUC 0.7730).
- **Trade-off precisión vs. coste:** El set original destaca en F1 para 4 clases pero incurre en mayor coste, mostrando la necesidad de balancear ambas dimensiones.
- **Optimización de umbrales:** Ajustar el threshold para la clase de alto riesgo consigue recalls entre 0.75 y 0.80, mejorando la detección de eventos críticos.
- **Robustez ante desequilibrio:** El uso combinado de SMOTE y validación estratificada genera recalls equilibrados entre clases, reduciendo el sesgo hacia la clase mayoritaria.
- **Sets reducidos vs. completos:** El set 17_Optimizado, con solo 17 variables, presenta un coste competitivo (0.9033 en 3-clases) aunque con menor F1, demostrando el valor de la selección de variables.
- **Patrón de ganancias marginales:** Los distintos niveles de ingeniería (14_Eng, 45_Eng, 95_Eng) reportan mejoras progresivas en coste y recall, confirmando que cada fase de enriquecimiento añade valor informativo.

Matriz de Confusión (Formato Texto)

0 - 15 10 8 1 - 120