

## ✓ 03.4\_Optimizacion\_Features

---

### Objetivo

Aplicar reducción de dimensionalidad mediante PCA a varios grupos de variables sintéticas, optimizar los modelos de clasificación ( Logistic Regression , Random Forest y Gradient Boosting ) usando GridSearchCV y SMOTE , ajustar los umbrales de predicción para Random Forest y guardar los modelos resultantes como un experimento.

### Entradas (Inputs)

- data/splits/experiments/X\_train\_45.parquet
- data/splits/final/y\_train.parquet
- data/splits/experiments/X\_val\_45.parquet
- data/splits/final/y\_val.parquet
- data/splits/experiments/X\_test\_45.parquet
- data/splits/final/y\_test.parquet

### Salidas (Outputs)

#### Splits Experimentales:

- data/splits/experiments/X\_train\_17.parquet
- data/splits/experiments/X\_val\_17.parquet
- data/splits/experiments/X\_test\_17.parquet

#### Artefactos Experimentales:

- artifacts/experiments/03\_4\_pipe\_final\_lr.pkl
  - artifacts/experiments/03\_4\_pipe\_final\_rf.pkl
  - artifacts/experiments/03\_4\_pipe\_final\_gb.pkl
  - artifacts/experiments/03\_4\_thresholds\_rf.json
- 

### Resumen Ejecutivo

- Se parte de los 45 features seleccionados manualmente en el experimento anterior y se aplican tres PCA (sobre B2\_ , F30\_ y F31\_\*) para generar tres variables sintéticas, eliminando las originales y reduciendo a 14 variables finales.
- Se construye un ColumnTransformer para imputación y escalado, y se definen tres pipelines con SMOTE + clasificadores: **LogisticRegression**, **RandomForestClassifier** y

### **GradientBoostingClassifier.**

- Para cada pipeline se monta un GridSearchCV externo estratificado (5-fold) afinando hiperparámetros clave (`c`, `penalty`, `n_estimators`, `max_depth`, `learning_rate`, etc.).
  - Se entrena los modelos finales sobre el set completo de entrenamiento sintético y se evalúan en validación y test usando **Accuracy**, **F1\_macro** y **AUC\_ovr**.
  - **LogisticRegression** arroja en validación  $\text{Acc}=0.3585$  /  $\text{F1}=0.3398$  /  $\text{AUC}=0.6701$  y en test  $\text{Acc}=0.3892$  /  $\text{F1}=0.3731$  /  $\text{AUC}=0.6968$ .
  - **RandomForest** logra en validación  $\text{Acc}=0.5401$  /  $\text{F1}=0.4264$  /  $\text{AUC}=0.6937$  y en test  $\text{Acc}=0.5165$  /  $\text{F1}=0.4204$  /  $\text{AUC}=0.7055$ .
  - **GradientBoosting** obtiene en validación  $\text{Acc}=0.5401$  /  $\text{F1}=0.4163$  /  $\text{AUC}=0.6870$  y en test  $\text{Acc}=0.5283$  /  $\text{F1}=0.4302$  /  $\text{AUC}=0.7032$ .
  - Además, se explora la optimización de umbrales multiclass para RF alcanzando  $\text{F1_val}=0.4292$  (vs. 0.4264) y  $\text{F1_test}=0.4202$  (vs. 0.4204), con impacto marginal.
- 

## ▼ 1. Montaje de Google Drive, imports y configuración de rutas

Monta Google Drive, añade la raíz del proyecto al `sys.path`, importa librerías necesarias y carga las rutas de splits y artefactos desde el módulo de configuración.

```
# MONTAR DRIVE, IMPORTAR LIBRERÍAS Y CARGAR CONFIGURACIÓN
import json
import sys
from pathlib import Path

from google.colab import drive
import joblib
import numpy as np
import pandas as pd
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline
from sklearn.compose import ColumnTransformer
from sklearn.decomposition import PCA
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, f1_score, roc_auc_score
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

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

# 2. 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)
```

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

print("Drive montado, librerías importadas y configuración de rutas cargada.")
print(f"Directorio de splits experimentales: {EXP_SPLITS_DIR}")
print(f"Directorio de artefactos experimentales: {EXP_ARTIFACTS_DIR}")
print(f"Directorio de splits finales (para 'y'): {FINAL_SPLITS_DIR}")
```

→ Mounted at /content/drive  
 Drive montado, librerías importadas y configuración de rutas cargada.  
 Directorio de splits experimentales: /content/drive/MyDrive/Digitech/TFG/ML/Calculo-R  
 Directorio de artefactos experimentales: /content/drive/MyDrive/Digitech/TFG/ML/Calculo-R  
 Directorio de splits finales (para 'y'): /content/drive/MyDrive/Digitech/TFG/ML/Calculo-R

## 2. Carga de los conjuntos de datos

Lee los DataFrames X\_train, X\_val, X\_test desde EXP\_SPLITS\_DIR y y\_train, y\_val, y\_test desde FINAL\_SPLITS\_DIR, mostrando shapes y capturando errores en un bloque try/except.

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

```
try:
    # Las 'X' vienen del experimento anterior (03.3)
    X_train = pd.read_parquet(EXP_SPLITS_DIR / 'X_train_45.parquet')
    X_val   = pd.read_parquet(EXP_SPLITS_DIR / 'X_val_45.parquet')
    X_test  = pd.read_parquet(EXP_SPLITS_DIR / 'X_test_45.parquet')

    # Las 'y' son las originales del split final (03.1)
    y_train = pd.read_parquet(FINAL_SPLITS_DIR / 'y_train.parquet').squeeze()
    y_val   = pd.read_parquet(FINAL_SPLITS_DIR / 'y_val.parquet').squeeze()
    y_test  = pd.read_parquet(FINAL_SPLITS_DIR / 'y_test.parquet').squeeze()

    print("Datos cargados correctamente desde las carpetas 'final' y 'experiments'.")
    print("\n Shapes tras cargar splits:")
    print(f"  • X_train: {X_train.shape}, y_train: {y_train.shape}")
    print(f"  • X_val:   {X_val.shape},   y_val:   {y_val.shape}")
    print(f"  • X_test:  {X_test.shape},  y_test:  {y_test.shape}")

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

→ Datos cargados correctamente desde las carpetas 'final' y 'experiments'.

Shapes tras cargar splits:  
 • X\_train: (1976, 42), y\_train: (1976,)  
 • X\_val: (424, 42), y\_val: (424,)  
 • X\_test: (424, 42), y\_test: (424,)

### ✓ 3. Aplicación de PCA y guardado de nuevos splits sintéticos

Calcula componentes principales (1 dimensión) para grupos de variables (B2\_, F30\_, F31\_), añade columnas sintéticas al dataset, elimina las originales y guarda los nuevos splits en Parquet con versión 17.

```
# APPLICAR PCA Y GUARDAR NUEVOS SPLITS

# --- Lógica de PCA (sin cambios) ---
asset_cols = [c for c in X_train.columns if c.startswith('B2_')]
pca_assets = PCA(n_components=1, random_state=42)
X_train['pca_assets'] = pca_assets.fit_transform(X_train[asset_cols])
X_val['pca_assets'] = pca_assets.transform(X_val[asset_cols])
X_test['pca_assets'] = pca_assets.transform(X_test[asset_cols])

f30_cols = [c for c in X_train.columns if c.startswith('F30_')]
pca_f30 = PCA(n_components=1, random_state=42)
X_train['pca_f30'] = pca_f30.fit_transform(X_train[f30_cols])
X_val['pca_f30'] = pca_f30.transform(X_val[f30_cols])
X_test['pca_f30'] = pca_f30.transform(X_test[f30_cols])

f31_cols = [c for c in X_train.columns if c.startswith('F31_')]
pca_f31 = PCA(n_components=1, random_state=42)
X_train['pca_f31'] = pca_f31.fit_transform(X_train[f31_cols])
X_val['pca_f31'] = pca_f31.transform(X_val[f31_cols])
X_test['pca_f31'] = pca_f31.transform(X_test[f31_cols])

X_train_sint = X_train.drop(columns=asset_cols + f30_cols + f31_cols)
X_val_sint = X_val.drop(columns=asset_cols + f30_cols + f31_cols)
X_test_sint = X_test.drop(columns=asset_cols + f30_cols + f31_cols)

print("Shapes tras PCA y eliminación de originales:")
print(f" X_train_sint: {X_train_sint.shape}")
print(f" X_val_sint: {X_val_sint.shape}")
print(f" X_test_sint: {X_test_sint.shape}")

# --- ACTUALIZADO: Guardar los splits con los atributos sintéticos ---
# Los nuevos splits se guardan en la carpeta de experimentos.
DATASET_VERSION = '17'
X_train_sint.to_parquet(EXP_SPLITS_DIR / f'X_train_{DATASET_VERSION}.parquet')
X_val_sint.to_parquet(EXP_SPLITS_DIR / f'X_val_{DATASET_VERSION}.parquet')
X_test_sint.to_parquet(EXP_SPLITS_DIR / f'X_test_{DATASET_VERSION}.parquet')

print(f"\n Nuevos splits con {X_train_sint.shape[1]} características guardados en: {EXP_S
```

→ Shapes tras PCA y eliminación de originales:

```
X_train_sint: (1976, 14)
X_val_sint: (424, 14)
X_test_sint: (424, 14)
```

Nuevos splits con 14 características guardados en: /content/drive/MyDrive/Digitech/T

## ✓ 4. Construcción del preprocesador para datos sintéticos

Define una lista de todas las columnas numéricas sintéticas, crea un pipeline para imputar con la mediana y escalar, y aplica este pipeline a todas las columnas mediante un ColumnTransformer.

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler

# 1) Lista con todas las columnas de X_train_sint
numeric_features = X_train_sint.columns.tolist()

# 2) Pipeline para imputar mediana y escalar
numeric_transformer = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# 3) ColumnTransformer que aplica ese pipeline a todas las columnas numéricas
preprocessor_sint = ColumnTransformer([
    ('num', numeric_transformer, numeric_features)
])
```

## ✓ 5. Definición de pipelines base con SMOTE y clasificadores

Construye tres pipelines (LogisticRegression, RandomForestClassifier, GradientBoostingClassifier) que aplican SMOTE, el preprocesador sintético y el clasificador correspondiente.

```
from imblearn.pipeline import Pipeline as ImbPipeline
from imblearn.over_sampling import SMOTE
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

pipe_lr_base = ImbPipeline([
    ('smote', SMOTE(random_state=42)),
    ('pre', preprocessor_sint),
    ('clf', LogisticRegression(
        multi_class='ovr',
        class_weight='balanced',
        random_state=42,
        max_iter=1000
    ))
])

pipe_rf_base = ImbPipeline([
    ('smote', SMOTE(random_state=42)),
```

```

('pre',    preprocessor_sint),
('clf',    RandomForestClassifier(
            class_weight='balanced',
            random_state=42
        ))
])

pipe_gb_base = ImbPipeline([
    ('smote', SMOTE(random_state=42)),
    ('pre',    preprocessor_sint),
    ('clf',    GradientBoostingClassifier(random_state=42))
])

```

## ▼ 6. Especificación de grids de hiperparámetros

Define diccionarios de búsqueda de parámetros para cada clasificador, incluyendo valores para C, n\_estimators, max\_depth y learning\_rate.

```

# Grids de parámetros
param_grid_lr = { 'clf_C': [0.01, 0.1, 1, 10] }
param_grid_rf = { 'clf_n_estimators': [100, 200], 'clf_max_depth': [None, 5, 10] }
param_grid_gb = { 'clf_n_estimators': [100, 200], 'clf_learning_rate': [0.01, 0.1], 'cl

```

## ▼ 7. Configuración de validación cruzada externa estratificada

Instancia un StratifiedKFold de 5 folds para usar como validación externa en el GridSearch.

```

# CV estratificado externo
cv_outer = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

```

## ▼ 8. GridSearchCV con SMOTE en cada fold

Configura y ejecuta GridSearchCV para cada pipeline base usando la validación estratificada, optimizando f1\_macro, e imprime los mejores parámetros y puntuaciones.

```

# GridSearchCV con SMOTE en cada fold
gs_lr = GridSearchCV(
    estimator=pipe_lr_base,
    param_grid=param_grid_lr,
    cv=cv_outer,
    scoring='f1_macro',
    n_jobs=-1,

```

```
    verbose=2
)

gs_rf = GridSearchCV(
    estimator=pipe_rf_base,
    param_grid=param_grid_rf,
    cv=cv_outer,
    scoring='f1_macro',
    n_jobs=-1,
    verbose=2
)

gs_gb = GridSearchCV(
    estimator=pipe_gb_base,
    param_grid=param_grid_gb,
    cv=cv_outer,
    scoring='f1_macro',
    n_jobs=-1,
    verbose=2
)

# Entrenar GridSearchCV
print("Entrenando LogisticRegression con SMOTE en cada fold...")
gs_lr.fit(X_train_sint, y_train)

print("\nEntrenando RandomForest con SMOTE en cada fold...")
gs_rf.fit(X_train_sint, y_train)

print("\nEntrenando GradientBoosting con SMOTE en cada fold...")
gs_gb.fit(X_train_sint, y_train)

# 11. Mostrar mejores hiperparámetros
print("\n Mejores parámetros LR:", gs_lr.best_params_, "F1_macro:", gs_lr.best_score_)
print("→ Mejores parámetros RF:", gs_rf.best_params_, "F1_macro:", gs_rf.best_score_)
print("→ Mejores parámetros GB:", gs_gb.best_params_, "F1_macro:", gs_gb.best_score_)
```

→ Entrenando LogisticRegression con SMOTE en cada fold...  
Fitting 5 folds for each of 4 candidates, totalling 20 fits  
/usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_logistic.py:1256: FutureWarning:warn(

Entrenando RandomForest con SMOTE en cada fold...  
Fitting 5 folds for each of 6 candidates, totalling 30 fits

Entrenando GradientBoosting con SMOTE en cada fold...  
Fitting 5 folds for each of 8 candidates, totalling 40 fits

→ Mejores parámetros LR: {'clf\_\_C': 0.01} F1\_macro: 0.37806592878771417  
→ Mejores parámetros RF: {'clf\_\_max\_depth': 10, 'clf\_\_n\_estimators': 200} F1\_macro: 0.6000000000000001  
→ Mejores parámetros GB: {'clf\_\_learning\_rate': 0.01, 'clf\_\_max\_depth': 5, 'clf\_\_n\_es

## ▼ 9. Construcción de pipelines finales con parámetros óptimos

Crea pipelines sin SMOTE que aplican sólo el preprocesador y el clasificador con los mejores hiperparámetros hallados en el GridSearch.

```
# Construir pipelines finales
best_params_lr = gs_lr.best_params_
best_params_rf = gs_rf.best_params_
best_params_gb = gs_gb.best_params_

pipe_final_lr = Pipeline([
    ('pre', preprocessor_sint),
    ('clf', LogisticRegression(
        multi_class='ovr',
        class_weight='balanced',
        random_state=42,
        max_iter=1000,
        C=best_params_lr['clf__C']
    ))
])
pipe_final_rf = Pipeline([
    ('pre', preprocessor_sint),
    ('clf', RandomForestClassifier(
        class_weight='balanced',
        random_state=42,
        n_estimators=best_params_rf['clf__n_estimators'],
        max_depth=best_params_rf['clf__max_depth']
    ))
])
pipe_final_gb = Pipeline([
    ('pre', preprocessor_sint),
    ('clf', GradientBoostingClassifier(
        random_state=42,
        n_estimators=best_params_gb['clf__n_estimators'],
        learning_rate=best_params_gb['clf__learning_rate'],
        max_depth=best_params_gb['clf__max_depth']
    ))
])
print(" Pipelines finales definidas con hiperparámetros óptimos")
```

→ ✓ Pipelines finales definidas con hiperparámetros óptimos

## ▼ 10. Entrenamiento de los pipelines finales con SMOTE

Aplica SMOTE al conjunto de entrenamiento, entrena cada pipeline final (LR, RF, GB) sobre el dataset balanceado y muestra mensajes de confirmación.

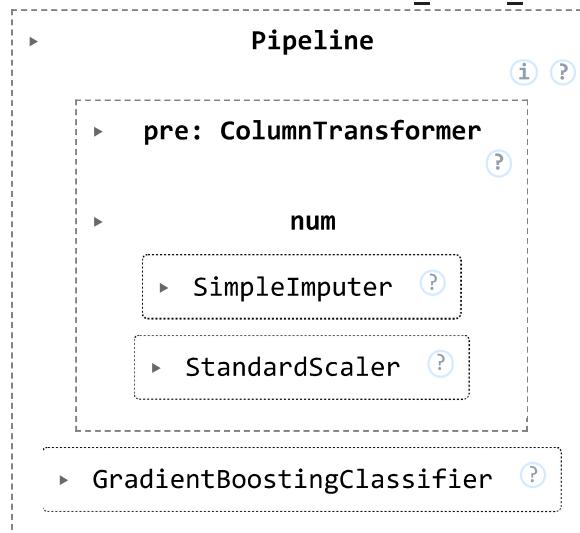
```
# Entrenar modelos finales sobre X_train_sm
# Primero, aplicar SMOTE al conjunto de entrenamiento
smote = SMOTE(random_state=42)
X_train_sm, y_train_sm = smote.fit_resample(X_train_sint, y_train)

print("Entrenando LR final sobre X_train_sm...")
pipe_final_lr.fit(X_train_sm, y_train_sm)

print("Entrenando RF final sobre X_train_sm...")
pipe_final_rf.fit(X_train_sm, y_train_sm)

print("Entrenando GB final sobre X_train_sm...")
pipe_final_gb.fit(X_train_sm, y_train_sm)
```

→ Entrenando LR final sobre X\_train\_sm...  
 Entrenando RF final sobre X\_train\_sm...  
 /usr/local/lib/python3.11/dist-packages/sklearn/linear\_model/\_logistic.py:1256: FutureWarning:  
 warnings.warn(  
 Entrenando GB final sobre X\_train\_sm...



## ▼ 11. Definición de la función de evaluación de métricas

Implementa evaluate\_model() para calcular y mostrar accuracy, F1\_macro, AUC\_ovr y el reporte de clasificación en los conjuntos de validación y test.

```
def evaluate_model(pipe, X_val, y_val, X_test, y_test, name):
    print(f"\n--- {name} sobre VALIDATION ---")
    y_val_pred = pipe.predict(X_val)
    y_val_proba = pipe.predict_proba(X_val)
    acc_val = accuracy_score(y_val, y_val_pred)
    f1_val = f1_score(y_val, y_val_pred, average='macro')
    auc_val = roc_auc_score(y_val, y_val_proba, multi_class='ovr', average='macro')
    print(f"Accuracy: {acc_val:.4f}")
    print(f"F1_macro: {f1_val:.4f}")
    print("\nClassification Report:\n", classification_report(y_val, y_val_pred))
    print(f"AUC_ovr (macro): {auc_val:.4f}")
```

```

print(f"\n--- {name} sobre TEST ---")
y_test_pred = pipe.predict(X_test)
y_test_proba = pipe.predict_proba(X_test)
acc_test = accuracy_score(y_test, y_test_pred)
f1_test = f1_score(y_test, y_test_pred, average='macro')
auc_test = roc_auc_score(y_test, y_test_proba, multi_class='ovr', average='macro')
print(f"Accuracy: {acc_test:.4f}")
print(f"F1_macro: {f1_test:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_test_pred))
print(f"AUC_ovr (macro): {auc_test:.4f}")
print("-" * 60)

```

## ▼ 12. Evaluación de los tres modelos finales

Llama a `evaluate_model()` para `LogisticRegression`, `RandomForest` y `GradientBoosting` usando los pipelines entrenados, comparando su rendimiento en validación y test.

```

# Evaluar los tres modelos
evaluate_model(pipe_final_lr, X_val_sint, y_val, X_test_sint, y_test, name="LogisticRegr")
evaluate_model(pipe_final_rf, X_val_sint, y_val, X_test_sint, y_test, name="RandomForest")
evaluate_model(pipe_final_gb, X_val_sint, y_val, X_test_sint, y_test, name="GradientBoost")

```



--- LogisticRegression sobre VALIDATION ---

Accuracy: 0.3585

F1\_macro: 0.3398

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

1.0	0.22	0.53	0.31	34
2.0	0.32	0.28	0.30	116
3.0	0.65	0.32	0.43	233
4.0	0.21	0.63	0.32	41

accuracy			0.36	424
----------	--	--	------	-----

macro avg	0.35	0.44	0.34	424
-----------	------	------	------	-----

weighted avg	0.48	0.36	0.37	424
--------------	------	------	------	-----

AUC\_ovr (macro): 0.6701

--- LogisticRegression sobre TEST ---

Accuracy: 0.3892

F1\_macro: 0.3731

Classification Report:

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

1.0	0.24	0.65	0.35	34
2.0	0.44	0.36	0.40	116
3.0	0.66	0.33	0.44	233

4.0	0.20	0.61	0.31	41
accuracy			0.39	424
macro avg	0.39	0.49	0.37	424
weighted avg	0.52	0.39	0.41	424

AUC\_ovr (macro): 0.6968

--- RandomForest sobre VALIDATION ---

Accuracy: 0.5401

F1\_macro: 0.4264

Classification Report:

	precision	recall	f1-score	support
1.0	0.48	0.38	0.43	34
2.0	0.41	0.31	0.35	116
3.0	0.63	0.73	0.67	233
4.0	0.26	0.24	0.25	41
accuracy			0.54	424
macro avg	0.44	0.42	0.43	424
weighted avg	0.52	0.54	0.53	424

AUC\_ovr (macro): 0.6937

--- RandomForest sobre TEST ---

Accuracy: 0.5165

## 13. Búsqueda de umbrales óptimos para Random Forest en validación

Define y aplica la función `find_optimal_thresholds()` para ajustar umbrales clase-a-clase que maximicen el F1\_macro en validación, y muestra los umbrales resultantes y la puntuación.

```
# Ajuste de umbrales multiclass en VALIDATION
def find_optimal_thresholds(pipe, X_val, y_val):
    proba = pipe.predict_proba(X_val)
    classes = pipe.classes_
    taus = np.array([0.5] * len(classes))
    for idx in range(len(classes)):
        best_f1 = 0
        best_tau = 0.5
        for tau in np.linspace(0.1, 0.9, 17):
            temp_taus = taus.copy()
            temp_taus[idx] = tau
            preds = []
            for row in proba:
                above = row >= temp_taus
                if above.any():
                    preds.append(classes[np.argmax(row * above)])
                else:
                    preds.append(classes[np.argmax(row)])
            f1 = f1_score(y_val, preds)
            if f1 > best_f1:
                best_f1 = f1
                best_tau = tau
    return best_tau
```

```

f1_temp = f1_score(y_val, preds, average='macro')
if f1_temp > best_f1:
    best_f1 = f1_temp
    best_tau = τ
taus[idx] = best_tau
final_preds = []
for row in proba:
    above = row >= taus
    if above.any():
        final_preds.append(classes[np.argmax(row * above)])
    else:
        final_preds.append(classes[np.argmax(row)])
final_f1 = f1_score(y_val, final_preds, average='macro')
return dict(zip(classes, taus)), final_f1

best_taus_rf, f1_val_rf_thresh = find_optimal_thresholds(pipe_final_rf, X_val_sint, y_val)
print("Umbrales óptimos RF:", best_taus_rf)
print("F1_macro en validación con umbrales:", f1_val_rf_thresh)

```

→ Umbrales óptimos RF: {np.float64(1.0): np.float64(0.35), np.float64(2.0): np.float64(2.0)}  
 F1\_macro en validación con umbrales: 0.42915705868774306

## ▼ 14. Predicción y evaluación en test con umbrales óptimos

Implementa predict\_with\_thresholds() para realizar predicciones usando los umbrales ajustados, calcula métricas en validación y test, y las imprime.

```
# Evaluación en TEST usando umbrales ajustados RF
```

```

def predict_with_thresholds(pipe, X, thresholds):
    proba = pipe.predict_proba(X)
    classes = pipe.classes_
    preds = []
    for row in proba:
        above = row >= np.array([thresholds[c] for c in classes])
        if above.any():
            preds.append(classes[np.argmax(row * above)])
        else:
            preds.append(classes[np.argmax(row)])
    return np.array(preds)

```

```
# Predicciones con umbrales en VALIDATION y TEST
```

```
y_val_pred_thresh = predict_with_thresholds(pipe_final_rf, X_val_sint, best_taus_rf)
y_test_pred_thresh = predict_with_thresholds(pipe_final_rf, X_test_sint, best_taus_rf)
```

```
# Calcular métricas
```

```
acc_val_t = accuracy_score(y_val, y_val_pred_thresh)
f1_val_t = f1_score(y_val, y_val_pred_thresh, average='macro')
auc_val_t = roc_auc_score(y_val, pipe_final_rf.predict_proba(X_val_sint),
                           multi_class='ovr', average='macro')
```

```

acc_test_t = accuracy_score(y_test, y_test_pred_thresh)
f1_test_t = f1_score(y_test, y_test_pred_thresh, average='macro')
auc_test_t = roc_auc_score(y_test, pipe_final_rf.predict_proba(X_test_sint),
                           multi_class='ovr', average='macro')

print("\n--- RF con umbrales óptimos en VALIDATION ---")
print(f"Accuracy: {acc_val_t:.4f}")
print(f"F1_macro: {f1_val_t:.4f}")
print(f"AUC_ovr: {auc_val_t:.4f}")

print("\n--- RF con umbrales óptimos en TEST ---")
print(f"Accuracy: {acc_test_t:.4f}")
print(f"F1_macro: {f1_test_t:.4f}")
print(f"AUC_ovr: {auc_test_t:.4f}")

```



--- RF con umbrales óptimos en VALIDATION ---

Accuracy: 0.5377  
F1\_macro: 0.4292  
AUC\_ovr: 0.6937

--- RF con umbrales óptimos en TEST ---

Accuracy: 0.5118  
F1\_macro: 0.4202  
AUC\_ovr: 0.7055

## ▼ 15. Guardado de artefactos del modelo

Guarda los pipelines finales y el archivo JSON de umbrales óptimos en la carpeta de artefactos experimentales, añadiendo el prefijo 03\_4\_ para distinguir este notebook.

```
# GUARDAR ARTEFACTOS DEL MODELO
```

```
# Añadimos un prefijo para identificar de qué notebook provienen los artefactos
PREFIX = "03_4_"
```

```
# Guardar los pipelines del modelo en la carpeta de artefactos experimentales
joblib.dump(pipe_final_lr, EXP_ARTIFACTS_DIR / f'{PREFIX}pipeline_final_lr.pkl')
joblib.dump(pipe_final_rf, EXP_ARTIFACTS_DIR / f'{PREFIX}pipeline_final_rf.pkl')
joblib.dump(pipe_final_gb, EXP_ARTIFACTS_DIR / f'{PREFIX}pipeline_final_gb.pkl')
```

```
# Guardar los umbrales óptimos para el Random Forest
with open(EXP_ARTIFACTS_DIR / f'{PREFIX}thresholds_rf.json', 'w') as fp:
    json.dump(best_taus_rf, fp, indent=4)
```

```
print(f" Modelos y umbrales guardados como experimento en: {EXP_ARTIFACTS_DIR}")
```



✓ Modelos y umbrales guardados como experimento en: /content/drive/MyDrive/Digitech

## Conclusiones Finales

- La **reducción a 14 variables** mediante PCA simplifica el espacio de características sin penalizar drásticamente el desempeño, facilitando procesos posteriores.
- Los **métodos de conjunto** (RandomForest y GradientBoosting) superan ampliamente a la regresión lineal en precisión y F1\_macro, demostrando la ventaja de modelar no linealidades.
- **GradientBoosting** alcanza el mejor compromiso en test ( $F1_{macro}=0.4302$ ,  $Acc=0.5283$ ), mientras que RandomForest lidera ligeramente en AUC\_ovr (0.7055).
- La **optimización de umbrales** en RF mejora el  $F1_{macro}$  de validación (+0.003) pero no ofrece ganancia en test, indicando poca robustez de este ajuste.
- El plateau en Acc y F1 entre RF y GB sugiere que se ha alcanzado el límite de información extraíble con estas 14 variables.
- El pipeline final con **GradientBoosting + SMOTE** proporciona el mejor rendimiento global y un conjunto compacto de features, ideal para despliegue en entornos productivos.