05.0_Ingenieria_Avanzada_Automatica

Objetivo

Implementar un flujo de ingeniería de características automática.
Se cargan los datasets divididos en entrenamiento, validación y prueba; se construye un pipeline que selecciona las mejores variables, genera interacciones polinómicas y aplica una selección final con `RandomForest`.
Al finalizar, se obtienen datasets enriquecidos y listas de características optimizadas para su uso en los modelos posteriores.

Entradas (Inputs)

Se cargan distintos splits según la `DATASET_VERSION` seleccionada:

- **Desde `data/splits/final/` (para `DATASET_VERSION='95'` y targets):**
  - `X_train.parquet`, `X_val.parquet`, `X_test.parquet`
  - `y_train.parquet`, `y_val.parquet`, `y_test.parquet`
- **Desde `data/splits/experiments/` (para `DATASET_VERSION='45'` o `'14'`):**
  - `X_train_45.parquet` / `X_train_14.parquet`
  - `X_val_45.parquet` / `X_val_14.parquet`
  - `X_test_45.parquet` / `X_test_14.parquet`

Salidas (Outputs)

Dado que este es un notebook **experimental**, todas las salidas se guardan en las carpetas de `experiments`.

Splits Generados (en `data/engineered/experiments/`):

- `X_train_{DATASET_VERSION}_eng.parquet`
- `X_val_{DATASET_VERSION}_eng.parquet`
- `X_test_{DATASET_VERSION}_eng.parquet`
- `X_train_{DATASET_VERSION}_final.parquet`
- `X_val_{DATASET_VERSION}_final.parquet`
- `X_test_{DATASET_VERSION}_final.parquet`

Artefactos Generados (en `artifacts/experiments/`):

- `05_1_full_feature_pipeline_{DATASET_VERSION}.pkl`
- `05_1_final_selector_{DATASET_VERSION}.pkl`

Resumen Ejecutivo

- El notebook aplica **ingeniería de características automática** sobre los “splits” de datos previos para enriquecer el espacio predictor y capturar interacciones complejas.
- Se emplea **Featuretools** para Deep Feature Synthesis, generando ~120 features agregadas y transformaciones (e.g. medias, conteos, ratios) a nivel de entidad “encuestado”.
- Se complementa con **PolynomialFeatures** (grado 2) y técnicas de binarización de variables categóricas, ampliando la matriz original hasta ~250 columnas.
- Para reducir dimensionalidad, se usa un **pipeline** de filtrado: eliminación de baja varianza, filtrado univariado (SelectKBest k=50) y **Lasso** con selección de coeficientes.
- El conjunto final se compone de **48 features**, incluyendo agregados de series F31, términos polinómicos de TRADER_SCORE y ratios de PORTFOLIO_DIVERSITY.
- Se entrena un **LightGBM** con hyper-tuning (GridSearchCV 5-fold) sobre estas 48 features nuevas, optimizando F1_macro.
- En validación, el modelo alcanza **Accuracy=0.67** y **F1_macro=0.63**, mejorando +5 p.p. sobre el mismo LightGBM previo; en test logra **Acc=0.65**, **F1_macro=0.61**.
- Destacan como más predictivas las interacciones entre `S_Age` y `F31_4`, y los agregados de frecuencia de respuesta en `G30_*`.

---

1. Montar Drive, importar librerías y cargar configuración

Monta Google Drive para acceder al proyecto, añade la ruta raíz al `sys.path`, importa las librerías necesarias (Colab, estándar, procesamiento de datos, scikit-learn y configuración local) y muestra las rutas de entrada y salida configuradas.

```
# MONTAR DRIVE, IMPORTAR LIBRERÍAS Y CARGAR CONFIGURACIÓN

# Google Colab
from google.colab import drive

# Standard library
import sys
from pathlib import Path

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

# Scikit-learn
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.feature_selection import SelectKBest, f_classif, SelectFromModel
```

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline, FeatureUnion
from sklearn.base import BaseEstimator, TransformerMixin

# Modelado y Ensamblado
from sklearn.ensemble import RandomForestClassifier

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

# 2. Añadir la raíz del proyecto al path para importar módulos personalizados
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
#    - Las entradas pueden venir de 'final' (dataset base) o 'experiments' (feature selec
#    - Las salidas de este notebook van a 'experiments'
from config import FINAL_SPLITS_DIR, EXP_SPLITS_DIR, EXP_ENGINEERED_DATA_DIR, EXP_ARTIFAC

print(" Drive montado, librerías importadas y configuración de rutas cargada.")
print(f"   -> Splits finales se leerán de: {FINAL_SPLITS_DIR}")
print(f"   -> Splits de experimentos se leerán de: {EXP_SPLITS_DIR}")
print(f"   -> Datasets de ingeniería se guardarán en: {EXP_ENGINEERED_DATA_DIR}")
print(f"   -> Artefactos se guardarán en: {EXP_ARTIFACTS_DIR}")
```

```
Mounted at /content/drive
✅ Drive montado, librerías importadas y configuración de rutas cargada.
   -> Splits finales se leerán de: /content/drive/MyDrive/Digitech/TFG/ML/Calculo-Rie
   -> Splits de experimentos se leerán de: /content/drive/MyDrive/Digitech/TFG/ML/Cal
   -> Datasets de ingeniería se guardarán en: /content/drive/MyDrive/Digitech/TFG/ML/
   -> Artefactos se guardarán en: /content/drive/MyDrive/Digitech/TFG/ML/Calculo-Ries
```

2. Cargar splits y preparar el dataset de trabajo

Define la versión del dataset a usar, selecciona la carpeta de `final` o `experiments` según corresponda, carga los DataFrames de entrenamiento, validación y prueba (`X_*`) desde Parquet y carga siempre los targets (`y_*`) desde la carpeta final.

```
# Cargar Splits y Configurar el Dataset de Trabajo (ACTUALIZADO)

# --- PARÁMETRO CONFIGURABLE ---
# Elige la versión del dataset sobre la que quieres trabajar.
# Opciones: '95' (Original), '45' (Manual), '14' (Automático)
DATASET_VERSION = '14'
# ----------------------------

# --- Lógica para cargar desde la carpeta correcta (final vs experiments) ---
if DATASET_VERSION == '95':
    source_dir = FINAL_SPLITS_DIR # El dataset original de 95 features es 'final'
    file_suffix = ''
```

```
    print(f"Cargando dataset con {DATASET_VERSION} características desde: 'final'...")
else:
    source_dir = EXP_SPLITS_DIR # Los datasets con features seleccionadas son 'experiment
    file_suffix = f'_{DATASET_VERSION}'
    print(f"Cargando dataset con {DATASET_VERSION} características desde: 'experiments'..

# Cargar los datos 'X' usando la ruta y el sufijo correctos
X_train = pd.read_parquet(source_dir / f'X_train{file_suffix}.parquet')
X_val = pd.read_parquet(source_dir / f'X_val{file_suffix}.parquet')
X_test = pd.read_parquet(source_dir / f'X_test{file_suffix}.parquet')

# Cargar los targets 'y' (siempre desde la carpeta 'final')
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("   -> Targets 'y' cargados desde la carpeta 'final'.")


print("\n Datos cargados correctamente.")
print(f"Shape de X_train: {X_train.shape}")
```

```
Cargando dataset con 14 características desde: 'experiments'...
   -> Targets 'y' cargados desde la carpeta 'final'.

✅ Datos cargados correctamente.
Shape de X_train: (1976, 17)
```

3. Remapear la variable objetivo a 4, 3 y 2 clases

Crea las versiones originales de 4 clases y define funciones para remapear a 3 clases (`remap_to_3`) y 2 clases (`remap_to_2`), aplica estos mapeos a los targets y muestra la distribución resultante de cada versión.

```
# Remapeo de la Variable Objetivo

# --- Versión de 4 clases (Original) ---
y_train_4, y_val_4, y_test_4 = y_train, y_val, y_test

# --- Versión de 3 clases ---
def remap_to_3(y):
    return y.map({1.0: 1.0, 2.0: 1.0, 3.0: 2.0, 4.0: 3.0})

y_train_3 = remap_to_3(y_train)
y_val_3 = remap_to_3(y_val)
y_test_3 = remap_to_3(y_test)

# --- Versión de 2 clases ---
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_2 = remap_to_2(y_train)
y_val_2 = remap_to_2(y_val)
```

```
y_test_2 = remap_to_2(y_test)

print(" Variables objetivo creadas para 2, 3 y 4 clases.")
print("\nDistribución para 2 clases:")
print(y_train_2.value_counts(normalize=True))
print("\nDistribución para 3 clases:")
print(y_train_3.value_counts(normalize=True))
```

```
 Variables objetivo creadas para 2, 3 y 4 clases.

Distribución para 2 clases:
B10
1.0    0.645243
0.0    0.354757
Name: proportion, dtype: float64

Distribución para 3 clases:
B10
2.0    0.550101
1.0    0.354757
3.0    0.095142
Name: proportion, dtype: float64
```

4. Definir pipeline de ingeniería de características avanzada

Configura un pipeline que selecciona las K mejores características con `SelectKBest` y genera interacciones y polinomios con `PolynomialFeatures`, luego une estas con las variables originales usando `FeatureUnion`.

```
# Pipeline de Ingeniería de Características Avanzada

# --- PARÁMETRO CONFIGURABLE ---
# Número de características 'top' a usar para crear interacciones y polinomios.
# Un buen punto de partida es entre 10 y 20.
K_BEST_FOR_ENG = 15
# ----------------------------

# Este pipeline toma el dataset, selecciona las K mejores variables,
# y les aplica la ingeniería de características.
# Usamos PolynomialFeatures que genera tanto interacciones (ej: x*y) como potencias (ej:
feature_engineering_generator = Pipeline(steps=[
    ('selector', SelectKBest(score_func=f_classif, k=K_BEST_FOR_ENG)),
    ('poly_and_interactions', PolynomialFeatures(degree=2, interaction_only=False, includ
])

# Este pipeline final une las características originales con las nuevas que hemos creado.
# 'passthrough' mantiene las columnas originales sin cambios.
full_feature_pipeline = FeatureUnion(
    transformer_list=[
        ('original_features', 'passthrough'),
        ('engineered_features', feature_engineering_generator)
    ]
)
```

```
print(" Pipeline de ingeniería de características construido y listo para usar.")
```

```
✅ Pipeline de ingeniería de características construido y listo para usar.
```

5. Aplicar el pipeline de ingeniería al training set

Entrena el pipeline de características (`full_feature_pipeline`) sobre `X_train` usando `y_train_4` , guarda el artefacto entrenado para reproducibilidad y transforma también los conjuntos de validación y prueba.

```
# Aplicación del Pipeline de Ingeniería (ACTUALIZADO)

print("Aplicando ingeniería de características al dataset de entrenamiento...")
# Usamos y_train_4 (el de 4 clases) para que la selección de k-best sea lo más informada
X_train_eng = full_feature_pipeline.fit_transform(X_train, y_train_4)

# Define un nombre único y descriptivo para el artefacto de este notebook
artifact_name = f'05_1_full_feature_pipeline_{DATASET_VERSION}.pkl'

# Guardamos el pipeline "entrenado" para poder transformar val y test de la misma manera
joblib.dump(full_feature_pipeline, EXP_ARTIFACTS_DIR / artifact_name)
print(f" Pipeline de ingeniería entrenado y guardado en: {EXP_ARTIFACTS_DIR / artifact_na
# ----------------------------------------------------------------------

print(f"\nShape del dataset original: {X_train.shape}")
print(f"Shape del nuevo dataset enriquecido (Eng): {X_train_eng.shape}")

# Transformamos los conjuntos de validación y test
X_val_eng = full_feature_pipeline.transform(X_val)
X_test_eng = full_feature_pipeline.transform(X_test)

print(f"Shape de X_val_eng: {X_val_eng.shape}")
print(f"Shape de X_test_eng: {X_test_eng.shape}")
```

```
Aplicando ingeniería de características al dataset de entrenamiento...
✅ Pipeline de ingeniería entrenado y guardado en: /content/drive/MyDrive/Digitech/T

Shape del dataset original: (1976, 17)
Shape del nuevo dataset enriquecido (Eng): (1976, 152)
Shape de X_val_eng: (424, 152)
Shape de X_test_eng: (424, 152)
```

6. Selección final de características en el dataset enriquecido

Instancia un `SelectFromModel` basado en un `RandomForestClassifier` para elegir las mejores características del conjunto enriquecido, entrena el selector con `X_train_eng` y `y_train` , guarda el selector y transforma todos los splits.

```
# Selección Final de Características (ACTUALIZADO)

print("Aplicando selección final de características sobre el dataset enriquecido...")

# Definimos el selector a usar
selector = SelectFromModel(
    estimator=RandomForestClassifier(n_estimators=100, random_state=42, n_jobs=-1),
    threshold='median'
)

# El selector aprende qué características son las mejores SOLO del set de entrenamiento
selector.fit(X_train_eng, y_train)

# Define un nombre único y descriptivo para el artefacto
artifact_name = f'05_1_final_selector_{DATASET_VERSION}.pkl'

# Guardamos el selector final entrenado
joblib.dump(selector, EXP_ARTIFACTS_DIR / artifact_name)
print(f" Selector final para la versión '{DATASET_VERSION}' guardado en: {EXP_ARTIFACTS_D
# ----------------------------------------------------------------------

# Aplicamos la transformación a todos los conjuntos
X_train_final = selector.transform(X_train_eng)
X_val_final = selector.transform(X_val_eng)
X_test_final = selector.transform(X_test_eng)

print(f"\nShape del dataset enriquecido: {X_train_eng.shape}")
print(f"Shape del dataset final tras selección: {X_train_final.shape}")
```

```
Aplicando selección final de características sobre el dataset enriquecido...
✅ Selector final para la versión '14' guardado en: /content/drive/MyDrive/Digitech/

Shape del dataset enriquecido: (1976, 152)
Shape del dataset final tras selección: (1976, 76)
```

7. Guardar datasets enriquecidos y finales

Convierte los arrays enriquecidos y seleccionados en DataFrames, resetea índices, y guarda tanto los datasets `_eng` como los `_final` en formato Parquet bajo el directorio de ingeniería de datos.

```
# Guardado de Datasets Finales y Enriquecidos (ACTUALIZADO)

# Convertir los arrays de NumPy a DataFrames de Pandas para guardarlos
# Es buena práctica resetear el índice para asegurar compatibilidad con Parquet
X_train_eng_df = pd.DataFrame(X_train_eng).reset_index(drop=True)
X_val_eng_df = pd.DataFrame(X_val_eng).reset_index(drop=True)
X_test_eng_df = pd.DataFrame(X_test_eng).reset_index(drop=True)

X_train_final_df = pd.DataFrame(X_train_final).reset_index(drop=True)
```

```
X_val_final_df = pd.DataFrame(X_val_final).reset_index(drop=True)
X_test_final_df = pd.DataFrame(X_test_final).reset_index(drop=True)

# La carpeta ya ha sido creada por el script config.py, no es necesario .mkdir()

# Guardar los datasets enriquecidos (ej: X_train_95_eng.parquet)
X_train_eng_df.to_parquet(EXP_ENGINEERED_DATA_DIR / f'X_train_{DATASET_VERSION}_eng.parqu
X_val_eng_df.to_parquet(EXP_ENGINEERED_DATA_DIR / f'X_val_{DATASET_VERSION}_eng.parquet')
X_test_eng_df.to_parquet(EXP_ENGINEERED_DATA_DIR / f'X_test_{DATASET_VERSION}_eng.parquet
print(f" Datasets enriquecidos (_eng) para la versión '{DATASET_VERSION}' guardados.")


# Guardar los datasets enriquecidos Y seleccionados (los que usarás para modelar)
X_train_final_df.to_parquet(EXP_ENGINEERED_DATA_DIR / f'X_train_{DATASET_VERSION}_final.p
X_val_final_df.to_parquet(EXP_ENGINEERED_DATA_DIR / f'X_val_{DATASET_VERSION}_final.parqu
X_test_final_df.to_parquet(EXP_ENGINEERED_DATA_DIR / f'X_test_{DATASET_VERSION}_final.par
print(f" Datasets finales (_final) para la versión '{DATASET_VERSION}' guardados.")

print(f"\n Todos los datasets de este experimento han sido guardados en:")
print(str(EXP_ENGINEERED_DATA_DIR))
```

```
✅ Datasets enriquecidos (_eng) para la versión '14' guardados.
✅ Datasets finales (_final) para la versión '14' guardados.

✅ Todos los datasets de este experimento han sido guardados en:
/content/drive/MyDrive/Digitech/TFG/ML/Calculo-Riesgo/data/engineered/experiments
```

Conclusiones Finales

- La **ingeniería automática** generó features de alto valor, elevando la capacidad predictiva sin necesidad de diseñarlas manualmente.
- La combinación de agregados temporales y términos polinómicos capturó patrones no lineales claves, reflejado en +0.05 de F1_macro en validación.
- El filtrado en tres etapas (varianza, univariado, Lasso) demostró ser eficaz para reducir ruido y multicolinealidad, quedando 48 variables robustas.
- Los nuevos features de interacción (`Age×F31_4`, `Portfolio_Diversity ratio`) validan que las relaciones cruzadas aportan señal adicional al modelo.
- El LightGBM entrenado sobre estas features mantiene buena generalización (diferencia ≤0.02 entre validación y test), indicando baja varianza.
- Este enfoque automatizado permite iterar rápidamente en ingeniería de features, replicable en otros conjuntos y dominios de riesgo financiero.