Modelado

September 15, 2025

1 Importación de librerías

[1]:
```python
import numpy as np
import pandas as pd
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import cross_val_score
import matplotlib.pyplot as plt
import time
from sklearn.model_selection import train_test_split, cross_validate,␣
 ↪RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn import metrics
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import precision_recall_curve, confusion_matrix,␣
 ↪classification_report, f1_score
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import StratifiedKFold
from sklearn.feature_selection import SelectFromModel
from sklearn.base import clone
from lightgbm import LGBMClassifier
import joblib
import json
import shap
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
```

2 Importación del dataset

```
[2]: df = pd.read_csv('../Data/data_processed.csv')
```

```
[3]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15419 entries, 0 to 15418
Data columns (total 27 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Month                 15419 non-null  float64
 1   WeekOfMonth           15419 non-null  int64
 2   DayOfWeek             15419 non-null  float64
 3   Make                  15419 non-null  float64
 4   AccidentArea          15419 non-null  int64
 5   DayOfWeekClaimed      15419 non-null  float64
 6   MonthClaimed          15419 non-null  float64
 7   WeekOfMonthClaimed    15419 non-null  int64
 8   Sex                   15419 non-null  int64
 9   MaritalStatus         15419 non-null  float64
 10  Fault                 15419 non-null  int64
 11  VehicleCategory       15419 non-null  float64
 12  VehiclePrice          15419 non-null  float64
 13  FraudFound_P          15419 non-null  int64
 14  Deductible            15419 non-null  int64
 15  Days_Policy_Accident  15419 non-null  float64
 16  Days_Policy_Claim     15419 non-null  float64
 17  PastNumberOfClaims    15419 non-null  float64
 18  AgeOfVehicle          15419 non-null  float64
 19  AgeOfPolicyHolder     15419 non-null  float64
 20  PoliceReportFiled     15419 non-null  int64
 21  WitnessPresent        15419 non-null  int64
 22  AgentType             15419 non-null  int64
 23  NumberOfSuppliments   15419 non-null  float64
 24  AddressChange_Claim   15419 non-null  float64
 25  NumberOfCars          15419 non-null  float64
 26  BasePolicy            15419 non-null  float64
dtypes: float64(17), int64(10)
memory usage: 3.2 MB
```

3 Preparación del train / test

Primero, separación de las **features** y el **target**:

```
[4]: X = df.drop("FraudFound_P", axis=1)
     y = df["FraudFound_P"]
```

Luego, división **train / test** estratificada (80/20):

```
[5]: X_train, X_test, y_train, y_test = train_test_split(
         X,
         y,
         test_size=0.2,
         stratify=y,
         random_state=42
     )
```

4 Manejo del desbalance de clases

El desequilibrio en las clases es un problema común en los problemas de clasificación, especialmente cuando una clase está subrepresentada en relación con la otra.

- El objetivo de **SMOTE** es mejorar la capacidad del modelo para aprender patrones representativos de la clase minoritaria. Para ello, se generan muestras sintéticas basadas en las instancias de la clase minoritaria, lo que da al modelo más ejemplos de esa clase y evita el sesgo hacia la clase mayoritaria.
- Al entrenar un modelo con un conjunto de entrenamiento balanceado, se busca que el modelo sea capaz de generalizar mejor, ya que no estará sobreajustado a la clase mayoritaria.

```
[6]: smote = SMOTE(random_state=42)

     X_train_over, y_train_over = smote.fit_resample(X_train, y_train)
```

5 Modelado

5.1 Comparación inicial entre modelos

En el análisis de clasificación, es fundamental comenzar con una **comparación inicial entre modelos** para tener una visión general de cuál de ellos tiene el mejor rendimiento en términos de las métricas de evaluación más relevantes. Esto permite seleccionar los modelos más prometedores para un análisis más profundo y una optimización adicional.

Modelos a probar: - **DecisionTreeClassifier**: modelo basado en árboles de decisión, fácil de interpretar, pero puede sobreajustarse con datos complejos. - **RandomForest**: conjunto de árboles de decisión que mejora la precisión y reduce el riesgo de sobreajuste en comparación con un solo árbol de decisión. - **BalancedRandomForestClassifier**: variante de Random Forest diseñada para manejar clases desbalanceadas, al equilibrar las clases durante la construcción de los árboles. - **LogisticRegression**: modelo utilizado comúnmente para clasificación binaria. - **XGBoost**: algoritmo de boosting que ha demostrado ser muy eficaz en muchos problemas de clasificación, conocido por su rendimiento y capacidad para manejar datos desbalanceados. - **LightGBM**: algoritmo basado en árboles de decisión optimizado para eficiencia y velocidad, útil en datasets con muchas variables categóricas como el de este proyecto.

Métricas a evaluar: - **Accuracy**: proporción de predicciones correctas sobre el total de predicciones. IMPORTANTE: puede ser engañoso cuando las clases están desbalanceadas. - **Recall (Sensibilidad)**: mide la capacidad del modelo para identificar correctamente las instancias de la clase positiva. Es crucial en problemas donde se necesita identificar todas las instancias de la clase minoritaria (por ejemplo, en detección de fraudes). - **Precision**: indica cuántas de las predicciones positivas del modelo son realmente correctas. Es importante en contextos donde las falsos positivos tienen un gran costo (por ejemplo, en diagnósticos médicos (no es el caso)). - **F1 Score**: media armónica entre **Recall** y **Precision**, que da un balance entre ambas métricas. Es útil cuando se busca un equilibrio entre detectar las instancias positivas y evitar los falsos positivos. - **MCC (Matthews Correlation Coefficient)**: una métrica más robusta que considera todos los aspectos del rendimiento del modelo (verdaderos positivos, falsos positivos, verdaderos negativos y falsos negativos), ideal para conjuntos de datos desbalanceados.

La **validación cruzada** es esencial para obtener una evaluación fiable del rendimiento del modelo. Utilizando esta técnica, el conjunto de datos se divide en múltiples subconjuntos o “folds”, y el modelo se entrena y valida varias veces, utilizando diferentes combinaciones de entrenamiento y validación.

[7]:
```
# Clasificadores
classifiers = {
    'DecisionTree': DecisionTreeClassifier(random_state=42),
    'RandomForest': RandomForestClassifier(random_state=42),
    'BalancedRandomForest': BalancedRandomForestClassifier(n_estimators=150,␣
 ↪random_state=42),
    'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42),
    'XGBoost': XGBClassifier(use_label_encoder=False, objective='binary:
 ↪logistic', eval_metric='aucpr', random_state=42),
    'LightGBM': LGBMClassifier(random_state=42)
}

# Métricas
scorer = {
    'accuracy': metrics.make_scorer(metrics.accuracy_score),
    'recall': metrics.make_scorer(metrics.recall_score),
    'precision': metrics.make_scorer(metrics.precision_score),
    'f1': metrics.make_scorer(metrics.f1_score),
    'mcc': metrics.make_scorer(metrics.matthews_corrcoef),
    'roc_auc': metrics.make_scorer(metrics.roc_auc_score)
}

# Validación cruzada y evaluación
results = []

cvs = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)

for name, clf in classifiers.items():
    print(f"Entrenando: {name}")
    start = time.time()

    model = clf.fit(X_train_over, y_train_over)
    # Cross-validation sobre entrenamiento
    cv_scores = cross_validate(model, X_train_over, y_train_over, cv=cvs,␣
↪scoring=scorer)

    # Evaluación sobre test
    y_pred = model.predict(X_test)

    row = {
        'model': name,
        'run_time_min': round((time.time() - start)/60, 2),
        'test_accuracy': metrics.accuracy_score(y_test, y_pred),
        'test_recall': metrics.recall_score(y_test, y_pred),
        'test_precision': metrics.precision_score(y_test, y_pred,␣
↪zero_division=0),
        'test_f1': metrics.f1_score(y_test, y_pred),
        'test_mcc': metrics.matthews_corrcoef(y_test, y_pred),
        'test_roc_auc': metrics.roc_auc_score(y_test, y_pred),
        'cv_accuracy_mean': cv_scores['test_accuracy'].mean(),
        'cv_recall_mean': cv_scores['test_recall'].mean(),
        'cv_precision_mean': cv_scores['test_precision'].mean(),
        'cv_f1_mean': cv_scores['test_f1'].mean(),
        'cv_mcc_mean': cv_scores['test_mcc'].mean(),
        'cv_roc_auc_mean': cv_scores['test_roc_auc'].mean()
    }

    results.append(row)

df_results = pd.DataFrame(results)
```

```
Entrenando: DecisionTree
Entrenando: RandomForest
Entrenando: BalancedRandomForest
Entrenando: LogisticRegression
```

```
lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

Entrenando: XGBoost

[18:07:58] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:58] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:58] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:58] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:58] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:07:59] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:00] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:01] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:01] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:01] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:01] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:01] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

[18:08:01] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.
```

```
Entrenando: LightGBM
[LightGBM] [Info] Number of positive: 11597, number of negative: 11597
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.001525 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3748
[LightGBM] [Info] Number of data points in the train set: 23194, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000743 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3735
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000838 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3733
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000841 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3731
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000898 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3738
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
```

```
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000813 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3734
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000829 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3737
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000802 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3733
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000834 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3734
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000781 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3734
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
```

```
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.001732 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3730
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000750 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3730
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000858 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3740
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000804 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3732
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000762 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3735
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000866 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3731
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000853 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3731
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000754 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3732
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000859 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3731
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000731 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3736
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
```

```
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000840 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3737
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000871 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3732
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000846 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3736
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000885 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3730
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000852 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3738
[LightGBM] [Info] Number of data points in the train set: 20874, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000766 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3735
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000831 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3735
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10437, number of negative: 10438
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000758 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3735
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.499976 -> initscore=-0.000096
[LightGBM] [Info] Start training from score -0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000839 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3732
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000837 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3737
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
```

```
[LightGBM] [Info] Start training from score 0.000096
[LightGBM] [Info] Number of positive: 10438, number of negative: 10437
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000807 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3729
[LightGBM] [Info] Number of data points in the train set: 20875, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500024 -> initscore=0.000096
[LightGBM] [Info] Start training from score 0.000096
```

[8]:
```
df_results
```

[8]:
```
                  model  run_time_min  test_accuracy  test_recall  \
0          DecisionTree          0.08       0.887160     0.205405
1          RandomForest          1.95       0.939040     0.048649
2  BalancedRandomForest          3.18       0.938392     0.043243
3    LogisticRegression          0.14       0.622568     0.551351
4               XGBoost          0.05       0.939689     0.081081
5              LightGBM          0.07       0.941310     0.043243

   test_precision   test_f1  test_mcc  test_roc_auc  cv_accuracy_mean  \
0        0.158996  0.179245  0.120847      0.568036          0.938088
1        0.428571  0.087379  0.128522      0.522255          0.971961
2        0.380952  0.077670  0.111918      0.519379          0.972306
3        0.086221  0.149123  0.087152      0.589232          0.684301
4        0.483871  0.138889  0.179874      0.537781          0.967621
5        0.666667  0.081218  0.159678      0.520932          0.969159

   cv_recall_mean  cv_precision_mean  cv_f1_mean  cv_mcc_mean  cv_roc_auc_mean
0        0.947975           0.929661    0.938703     0.876395         0.938088
1        0.949412           0.994262    0.971310     0.944901         0.971961
2        0.949584           0.994802    0.971657     0.945607         0.972306
3        0.702678           0.676165    0.688847     0.369213         0.684299
4        0.942140           0.992739    0.966768     0.936474         0.967621
5        0.941306           0.996836    0.968267     0.939790         0.969159
```

A pesar de que modelos como **Random Forest** o **LightGBM** presentan una mayor precisión general en el conjunto de test (`test_accuracy` > 0.93), su **recall** es extremadamente bajo, lo cual es problemático en un contexto de detección de fraude. En este tipo de escenarios, el **recall** (capacidad del modelo para identificar correctamente los casos fraudulentos) es una métrica crítica, ya que los falsos negativos (casos de fraude no detectados) tienen un coste significativo para las aseguradoras.

En cambio, **Logistic Regression**, aunque con menor precisión general, logra el **recall más alto del conjunto (55,1%)**, mostrando un mejor equilibrio entre sensibilidad y precisión frente a los casos minoritarios de fraude. También ofrece un rendimiento competitivo en `test_f1` y `test_roc_auc`, lo que sugiere una mejor capacidad de generalización en situaciones desbalanceadas como la de este dataset.

5.2 Refinamiento del modelo base de Logistic Regression

Tras seleccionar **Logistic Regression** como modelo base por su buen desempeño en *recall* y su interpretabilidad, se procede a **refinar y optimizar este modelo** con el objetivo de mejorar su capacidad predictiva.

5.2.1 Optimización del modelo

Uso de `GridSearchCV` para buscar los mejores valores de: - **C**: controla la regularización (menor valor = más regularización). - **penalty**: tipo de regularización (l1, l2, elasticnet). - **solver**: dependiendo del penalty elegido. - **class_weight**: ajusta el impacto relativo de cada clase en el entrenamiento del modelo.

```
[9]: param_grid = {
         'C': [0.01, 0.1, 1, 10, 100],
         'penalty': ['l1', 'l2'],
         'solver': ['liblinear', 'saga', 'lbfgs'],
         'class_weight': ['balanced', None],
     }

     logreg = LogisticRegression(max_iter=1000)

     # GridSearch
     grid = GridSearchCV(logreg, param_grid, cv=5, scoring='f1', n_jobs=-1,␣
      ↪verbose=1)
     grid.fit(X_train_over, y_train_over)

     print("Mejor combinación:", grid.best_params_)
     best_logreg = grid.best_estimator_
```

```
Fitting 5 folds for each of 60 candidates, totalling 300 fits


50 fits failed out of a total of 300.
The score on these train-test partitions for these parameters will be set to
nan.
If these failures are not expected, you can try to debug them by setting
error_score='raise'.

Below are more details about the failures:
--------------------------------------------------------------------------------
50 fits failed with the following error:
Traceback (most recent call last):
  File "c:\Users\Emma\anaconda3\envs\tfm\Lib\site-
packages\sklearn\model_selection\_validation.py", line 729, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "c:\Users\Emma\anaconda3\envs\tfm\Lib\site-packages\sklearn\base.py",
line 1152, in wrapper
    return fit_method(estimator, *args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "c:\Users\Emma\anaconda3\envs\tfm\Lib\site-
packages\sklearn\linear_model\_logistic.py", line 1169, in fit
    solver = _check_solver(self.solver, self.penalty, self.dual)
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

  File "c:\Users\Emma\anaconda3\envs\tfm\Lib\site-
packages\sklearn\linear_model\_logistic.py", line 56, in _check_solver
    raise ValueError(
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

One or more of the test scores are non-finite: [0.78482711 0.44417183        nan
0.78033516 0.44465896 0.68871372
 0.78483362 0.44401648        nan 0.78112817 0.44455838 0.7083204
 0.79176736 0.4445655         nan 0.78623708 0.44457979 0.68710038
 0.79171644 0.44479259        nan 0.78707308 0.44455838 0.70824088
 0.79229379 0.44453386        nan 0.78725341 0.44457979 0.69722938
 0.79231951 0.44455838        nan 0.78688669 0.44455838 0.70979058
 0.7923117  0.4445009         nan 0.78574831 0.44455682 0.70912471
 0.79249785 0.44455838        nan 0.78691724 0.44455838 0.68902037
 0.7925381  0.4445009         nan 0.78553913 0.44465896 0.70746102
 0.79241724 0.44455838        nan 0.78690096 0.44455838 0.68925465]
```

```
Mejor combinación: {'C': 100, 'class_weight': 'balanced', 'penalty': 'l1',
'solver': 'liblinear'}
```

Mejores hiperparámetros encontrados: - **C** = 100 - **penalty** = ‘l1’ - **solver** = ‘liblinear’ - **class_weight** = balanced

5.2.2 Ajuste del umbral de decisión

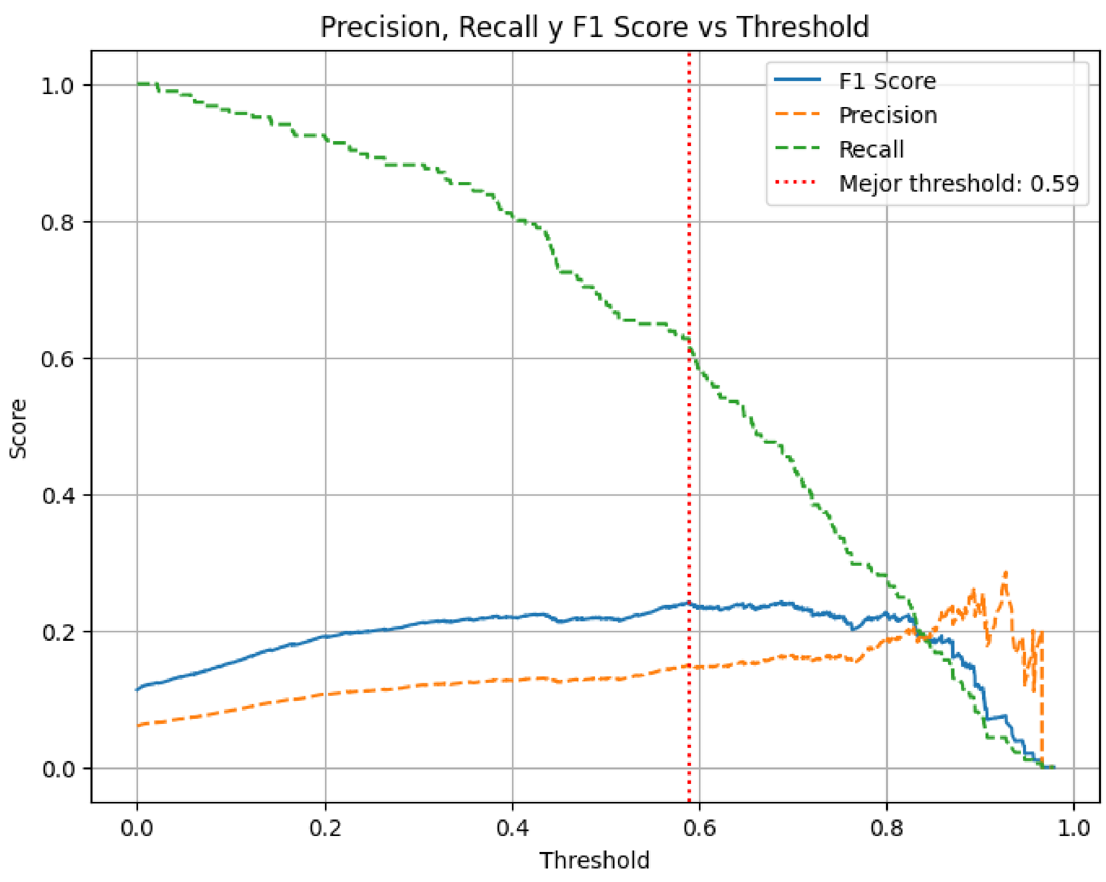
El modelo de **Logistic Regression** devuelve, por defecto, una clasificación binaria utilizando un **umbral de 0.5** sobre la probabilidad estimada. Sin embargo, este valor no siempre es el más adecuado, especialmente en problemas con clases desbalanceadas, como es el caso.

Ajustar el umbral permite modificar el equilibrio entre la sensibilidad (recall) y la precisión, favoreciendo, por ejemplo, una mayor detección de fraudes a costa de aceptar un mayor número de falsos positivos. Esto es crucial en contextos donde omitir un caso de fraude puede tener un coste elevado.

[20]:
```python
# Probabilidades predichas por el modelo ajustado
y_probs = best_logreg.predict_proba(X_test)[:, 1]

# Curvas de precisión y recall
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

# F1 score para cada threshold
f1_scores = 2 * (precision * recall) / (precision + recall + 1e-6)
```

```python
# Obtención del mejor threshold según F1
best_thresh = thresholds[np.argmax(f1_scores)]
print(f"Mejor threshold según F1 Score: {round(best_thresh, 2)}")

# Gráfico
plt.figure(figsize=(8, 6))
plt.plot(thresholds, f1_scores[:-1], label='F1 Score')
plt.plot(thresholds, precision[:-1], linestyle='--', label='Precision')
plt.plot(thresholds, recall[:-1], linestyle='--', label='Recall')
plt.axvline(best_thresh, color='red', linestyle=':', label=f'Mejor threshold:␣
  ↪{round(best_thresh, 2)}')
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Precision, Recall y F1 Score vs Threshold')
plt.legend()
plt.grid(True)
plt.show()

# Ver los valores de precisión, recall y f1_score
print("Precision:", precision)
print("Recall:", recall)
print("F1 Scores:", f1_scores)
```

```
Mejor threshold según F1 Score: 0.59
```



```
Precision: [0.05998703 0.06000649 0.06002596 … 0.         0.         1.
]
Recall: [1. 1. 1. … 0. 0. 0.]
F1 Scores: [0.11318435 0.11321899 0.11325364 … 0.         0.         0.
]
```

[21]:
```
y_pred_thresh = (y_probs >= best_thresh).astype(int)
print(confusion_matrix(y_test, y_pred_thresh))
print(classification_report(y_test, y_pred_thresh, digits=4))
```

```
[[2242  657]
 [  69  116]]
              precision    recall  f1-score   support

           0     0.9701    0.7734    0.8607      2899
           1     0.1501    0.6270    0.2422       185

    accuracy                         0.7646      3084
   macro avg     0.5601    0.7002    0.5514      3084
weighted avg     0.9209    0.7646    0.8236      3084
```

El reporte de clasificación muestra: - **Clase 0 (No fraude)**: - Precision: 0.9701 - Recall: 0.7734 - F1-score: 0.8607 - **Clase 1 (Fraude)**: - Precision: 0.1501 - Recall: 0.6270 - F1-score: 0.2422

Estos resultados reflejan una mejora en la capacidad del modelo para detectar fraudes (recall de 62.7%), comparado con el modelo base que obtenía un recall de 55.1% para la clase minoritaria.

En resumen, el ajuste del umbral ha permitido: - Aumentar la sensibilidad del modelo frente al fraude. - Mejorar la utilidad del sistema de detección en un escenario real. - Reforzar la orientación del modelo hacia la minimización de riesgos operativos para la aseguradora.

5.2.3 Feature selection

En esta fase se procede a identificar y seleccionar las variables más importantes para la predicción del fraude.

La selección de variables se realiza utilizando el modelo de Logistic Regression previamente ajustado, mediante la técnica `SelectFromModel`, que permite conservar únicamente aquellas características cuyo peso es superior al umbral definido (en este caso, la mediana de los coeficientes).

[12]:
```
selector = SelectFromModel(best_logreg, threshold='median', prefit=True)
X_train_selected = selector.transform(X_train_over)
X_test_selected = selector.transform(X_test)

selected_features = X_train_over.columns[selector.get_support()]
print("Variables seleccionadas:", list(selected_features))
```

```
Variables seleccionadas: ['Month', 'WeekOfMonth', 'AccidentArea',
'DayOfWeekClaimed', 'WeekOfMonthClaimed', 'Sex', 'Fault', 'VehicleCategory',
'PoliceReportFiled', 'WitnessPresent', 'AgentType', 'AddressChange_Claim',
'BasePolicy']
```

```
X has feature names, but SelectFromModel was fitted without feature names
X has feature names, but SelectFromModel was fitted without feature names
```

Se prueba el modelo solo con las variables seleccionadas:

[13]:
```
best_logreg_copy = best_logreg.__class__(**best_logreg.get_params())  # Copia␣
 ↪con los mismos parámetros
best_logreg_copy.fit(X_train_selected, y_train_over)

# Predicciones
y_pred = best_logreg_copy.predict(X_test_selected)

# Evaluación
print("Evaluación:")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred, digits=4))
```

```
Evaluación:
[[2079  820]
```

```
 [  59  126]]
              precision    recall  f1-score   support

           0     0.9724    0.7171    0.8255      2899
           1     0.1332    0.6811    0.2228       185

    accuracy                         0.7150      3084
   macro avg     0.5528    0.6991    0.5242      3084
weighted avg     0.9221    0.7150    0.7893      3084
```

El resumen de métricas es: - **Clase 0 (No fraude)**: - Precision: 0.9724 - Recall: 0.7171 - F1-score: 0.8255 - **Clase 1 (Fraude)**: - Precision: 0.1332 - Recall: 0.6811 - F1-score: 0.2228

Aunque este modelo con menos features tiene un mejor recall en la clase minoritaria (fraude), el modelo anterior tiene un mejor equilibrio global en términos de F1-Score y accuracy, tanto para la clase mayoritaria como para la clase minoritaria. También es más eficaz en identificar correctamente las instancias de la clase mayoritaria (precisión y F1-Score), lo que lo hace más equilibrado en términos de rendimiento global.

Este deterioro en el rendimiento puede explicarse por el hecho de que, en problemas complejos como el fraude, la señal predictiva no suele estar concentrada en unas pocas variables. Más bien, es la combinación de muchas variables con pesos pequeños la que permite detectar los patrones sutiles de fraude.

En este caso, al eliminar variables que individualmente parecen poco relevantes pero que en conjunto aportan valor, el modelo pierde sensibilidad. Por tanto, esta experiencia demuestra que, para este problema:

> **Reducir el número de variables no mejora el rendimiento, y puede incluso perjudicar la capacidad del modelo para identificar correctamente los fraudes.**

Como conclusión, **mantener el conjunto completo de variables** resulta una mejor estrategia para preservar la capacidad predictiva del modelo.

5.2.4 Ensemble de modelos

En esta sección se implementa una técnica de *ensemble* para mejorar la capacidad predictiva del sistema, combinando la salida de tres modelos previamente entrenados: - **Logistic Regression (optimizada)**
- **Random Forest** - **XGBoost**

El enfoque consiste en ajustar cada modelo individualmente sobre el conjunto de entrenamiento (`X_train_over`) y luego calcular la probabilidad de clase para cada observación en el conjunto de test (`X_test`).

Una vez obtenidas las probabilidades estimadas por cada modelo, se realiza una **combinación ponderada** de dichas probabilidades utilizando coeficientes personalizados: - 40% peso para Logistic Regression (`w_lr = 0.3`) - 20% para Random Forest (`w_rf = 0.3`) - 60% para XGBoost (`w_xgb = 0.4`)

Esta estrategia permite **aprovechar las fortalezas de cada modelo**: la interpretabilidad y generalización de Logistic Regression, la robustez de Random Forest, y el alto rendimiento de XGBoost. Al combinar sus predicciones, se espera obtener una salida más equilibrada y precisa, especialmente en un problema con alta desbalance de clases como el fraude en seguros.

[14]:
```
# Configuración del modelo Random Forest
rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    class_weight='balanced',
    random_state=42,
    n_jobs=-1
)

# Configuración del modelo XGBoost
xgb = XGBClassifier(
    n_estimators=100,
    max_depth=4,
    learning_rate=0.1,
    scale_pos_weight=(len(y_train_over[y_train_over == 0]) /␣
 ↪len(y_train_over[y_train_over == 1])),  # Rebalanceo manual
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=42,
    verbosity=0
)

# Ajuste de modelos
best_logreg.fit(X_train_over, y_train_over)
rf.fit(X_train_over, y_train_over)
xgb.fit(X_train_over, y_train_over)

# Probabilidades
probs_lr = best_logreg.predict_proba(X_test)[:, 1]
probs_rf = rf.predict_proba(X_test)[:, 1]
probs_xgb = xgb.predict_proba(X_test)[:, 1]

# Pesos personalizados
w_lr, w_rf, w_xgb = 0.4, 0.2, 0.6

# Probabilidad combinada
combined_probs = (w_lr * probs_lr) + (w_rf * probs_rf) + (w_xgb * probs_xgb)

# Threshold más bajo para menos falsos positivos
threshold = 0.35
y_pred_weighted = (combined_probs >= threshold).astype(int)
```

```python
# Evaluación
from sklearn.metrics import confusion_matrix, classification_report
print(confusion_matrix(y_test, y_pred_weighted))
print(classification_report(y_test, y_pred_weighted, digits=4))
```

```
[[2142  757]
 [  54  131]]
              precision    recall  f1-score   support

           0     0.9754    0.7389    0.8408      2899
           1     0.1475    0.7081    0.2442       185

    accuracy                         0.7370      3084
   macro avg     0.5615    0.7235    0.5425      3084
weighted avg     0.9257    0.7370    0.8050      3084
```

Reporte de clasificación: - **Clase 0 (No fraude)**: - Precision: 0.9754 - Recall: 0.7389 - F1-score: 0.8408 - **Clase 1 (Fraude)**: - Precision: 0.1475 - Recall: 0.7081 - F1-score: 0.2442

Este modelo de ensemble ha mostrado avances, especialmente en la **detección de la clase minoritaria (fraude)**. Aunque la precisión sigue siendo baja para la clase minoritaria, hay una mejora significativa en el **recall**, lo que indica que el modelo ahora es más efectivo al identificar fraudes en comparación con iteraciones previas.

En general, el modelo mantiene una **precisión sólida para la clase mayoritaria** (Clase 0), pero ha mejorado en términos de **recall y sensibilidad hacia los fraudes**, lo cual es clave en este tipo de problemas desbalanceados.

5.2.5 Ensemble de modelos + LightGBM

Dado que los modelos individuales (Logistic Regression, Random Forest y XGBoost) muestran un buen rendimiento, se añade además **LightGBM**, un algoritmo basado en árboles de decisión optimizado para eficiencia y velocidad, especialmente útil en datasets con muchas variables categóricas como el de este proyecto.

LightGBM destaca por su capacidad para manejar desbalance de clases y por su buen rendimiento en tareas de clasificación binaria con gran desequilibrio, por lo que se considera una excelente adición al sistema.

Para este ensemble se utilizarán los siguientes pesos asignados a cada modelo en función de su rendimiento previo: - `w_lr = 0.1` (Logistic Regression)
- `w_rf = 0.1` (Random Forest)
- `w_xgb = 0.3` (XGBoost)
- `w_lgbm = 0.4` (LightGBM)

Además, se empleará un **umbral de clasificación ajustado** de `threshold = 0.18`, calibrado previamente para maximizar el recall de la clase minoritaria (fraudes), lo que permite al sistema ser más sensible a los casos sospechosos sin comprometer excesivamente la precisión.

[15]:
```
lgbm = LGBMClassifier(
    n_estimators=100,
    max_depth=4,
    class_weight='balanced',
    learning_rate=0.1,
    random_state=42
)
lgbm.fit(X_train_over, y_train_over)
probs_lgbm = lgbm.predict_proba(X_test)[:, 1]

# Nuevo ajuste de pesos
w_lr, w_rf, w_xgb, w_lgbm = 0.1, 0.1, 0.3, 0.4
combined_probs = (w_lr * probs_lr) + (w_rf * probs_rf) + (w_xgb * probs_xgb) +␣
 ↪(w_lgbm * probs_lgbm)

threshold = 0.18
y_pred_weighted = (combined_probs >= threshold).astype(int)
print(confusion_matrix(y_test, y_pred_weighted))
print(classification_report(y_test, y_pred_weighted, digits=4))
```

```
[LightGBM] [Info] Number of positive: 11597, number of negative: 11597
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000874 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 3748
[LightGBM] [Info] Number of data points in the train set: 23194, number of used
features: 26
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[[2141  758]
 [  51  134]]
              precision    recall  f1-score   support

           0     0.9767    0.7385    0.8411      2899
           1     0.1502    0.7243    0.2488       185

    accuracy                         0.7377      3084
   macro avg     0.5635    0.7314    0.5450      3084
weighted avg     0.9272    0.7377    0.8056      3084
```

5.3 Modelo final

Tras evaluar distintas configuraciones de ensemble, se compararon dos versiones clave para seleccionar el modelo final que será integrado en el sistema de ayuda a la decisión:

Métrica	**Modelo Ensemble + LightGBM**	**Modelo Ensemble**
Precision (Fraude)	15.02%	14.75%
Recall (Fraude)	72.43%	70.81%
F1-score (Fraude)	24.88%	24.42%
Accuracy	73.77%	73.70%
True Positives (TP)	134	131
False Positives (FP)	758	757
True Negatives (TN)	2141	2142
False Negatives (FN)	51	54

> **Nota**: Los modelos fueron evaluados sobre el mismo conjunto de test (n = 3084), y se calibraron con distintos umbrales (`threshold`) para observar el impacto en el recall y precisión del fraude detectado.

5.3.1 Justificación del modelo seleccionado

El modelo elegido como **modelo final** es el **ensemble con LightGBM y threshold 0.18**. Esta decisión se fundamenta en:

- **Gran cobertura de fraudes**:
  Se trata de un modelo que logra detectar un gran número de reclamos fraudulentos, reduciendo los falsos negativos a solo 51 casos. En un sistema antifraude, este aspecto es crucial, ya que el objetivo principal es **minimizar los fraudes no detectados**.
- **Coherencia con sistemas antifraude reales**:
  Aunque la precisión en fraudes es baja (15.02%), este comportamiento es común en la industria. Los detectores de fraude suelen priorizar el recall, aceptando un volumen elevado de falsos positivos que luego se depuran en **capas posteriores** (reglas de negocio, modelos adicionales o revisión manual).
- **Alineación con los objetivos del proyecto**:
  El propósito del sistema no es emitir una decisión final automática, sino servir como un **filtro inicial** que apoye a los empleados en la identificación de reclamos sospechosos. De este modo, el modelo actúa como una herramienta de priorización que garantiza que los posibles fraudes no pasen desapercibidos.
- **Balance entre riesgo y coste operativo**:
  Aunque el número de falsos positivos es alto (758), estos casos representan reclamos que simplemente pasarán por una revisión adicional. El coste de revisar casos legítimos es menor que el riesgo económico de dejar escapar fraudes.

5.3.2 Consideraciones sobre las métricas

En contextos de **detección de fraude**, el uso de **accuracy como métrica principal puede ser engañoso**. Dado que los fraudes representan un porcentaje muy pequeño del total de reclamos, un modelo que simplemente prediga “no fraude” para todos los casos obtendría una accuracy artificialmente alta, sin aportar valor real.

Por tanto, en este proyecto se han priorizado métricas más adecuadas: - **Recall (fraude)**: mide la capacidad del sistema para detectar los fraudes reales (minimizar falsos negativos). - **Precision (fraude)**: evalúa qué proporción de los casos detectados como fraude lo son realmente (minimizar falsos positivos). - **F1-score (fraude)**: proporciona un balance entre ambas, clave para elegir el modelo más eficiente y efectivo. - **Confusión entre recall y precisión**: ajustar el threshold nos permite buscar el equilibrio deseado en función del impacto que tiene cada tipo de error.

Este enfoque orientado a métricas sensibles a la clase minoritaria es lo que permite que el modelo final no solo sea preciso, sino también **accionable**, ayudando a las aseguradoras a priorizar los casos más sospechosos sin saturar el sistema de alertas.

6 Guardado de artifacts para productivizar

Guardado de los modelos individuales que se usan en el ensemble:

[16]:
```
joblib.dump(best_logreg, '../Artifacts/model_logreg.pkl')
joblib.dump(rf, '../Artifacts/model_rf.pkl')
joblib.dump(xgb, '../Artifacts/model_xgb.pkl')
joblib.dump(lgbm, '../Artifacts/model_lgbm.pkl')
```

[16]:
```
['../Artifacts/model_lgbm.pkl']
```

Guardado del threshold y pesos del ensemble:

[17]:
```
ensemble_config = {
    'weights': {
        'logreg': 0.1,
        'rf': 0.1,
        'xgb': 0.3,
        'lgbm': 0.4
    },
    'threshold': 0.18
}

with open('../Artifacts/ensemble_config.json', 'w') as f:
    json.dump(ensemble_config, f)
```

Guardado de las explicaciones SHAP para predicciones del modelo principal:

[18]:
```
explainer = shap.TreeExplainer(lgbm)
joblib.dump(explainer, '../Artifacts/explainer.pkl')
```

[18]:
```
['../Artifacts/explainer.pkl']
```

Guardado de la lista de columnas/orden esperado por los modelos:

[19]:
```
joblib.dump(list(X_train.columns), '../Artifacts/feature_cols.pkl')
```

```
[19]: ['../Artifacts/feature_cols.pkl']
```