Inteligencia artificial explicable (xai) con lime y shap: interpretación del modelo del TITANIC

Alejandro Lara Caballero Erik Reyes Reyes

[1]Universidad Autónoma Metropolitana
Unidad Cuajimalpa Departamento de Matemáticas Aplicadas y Sistemas

alarac@cua.uam.mx ereyes@cua.uam.mx

Referencia de este artículo [1].

RESUMEN

La adopción y uso de modelos de aprendizaje automático para resolver diversos problemas en ciencia e ingeniería se han incrementado en años recientes. Aunque muchos de estos modelos han alcanzado una gran precisión, se puede considerar que funcionan como una "caja negra", lo que dificulta una transparencia en el proceso de toma de decisiones. Ante la creciente necesidad de explicar el porqué de los resultados obtenidos por estos métodos, surge la Inteligencia Artificial Explicable (XAI, por sus siglas en inglés). Mediante bibliotecas en Python y usando el conjunto de datos del Titanic, en este trabajo se aplican dos técnicas populares: LIME y SHAP. Estos métodos permiten a los ingenieros validar la lógica detrás de variables críticas como el género, la edad y la clase socioeconómica. Estas técnicas pueden trasladarse fácilmente a otros ámbitos de estudio.

1. Introducción

El aprendizaje automático ha permitido desarrollar modelos de gran capacidad predictiva, como las redes neuronales o los ensambles, sin embargo, estas estrategias se pueden considerar opacas, ya que no es fácil interpretar su funcionamiento interno. En campos críticos como la medicina, las finanzas o la ingeniería forense, no solo es crucial que un modelo acierte, sino también conocer bajo qué criterios lo hace para generar confianza.

En este contexto surge la Inteligencia Artificial Explicable (XAI), que engloba técnicas y métodos que busca transformar los modelos de 'caja negra' en sistemas más transparentes. Dos herramientas de uso extendido son LIME y SHAP. Estas técnicas se aplicaron en un caso de estudio clásico: los datos de supervivencia del Titanic. El objetivo no es solo predecir quién sobrevivió al naufragio, sino entender qué factores influyeron en esa probabilidad y verificar si el modelo está aprendiendo patrones reales de la tragedia o simplemente sesgos injustificados.

Cuando hablamos de XAI, existen dos categorías principales de métodos. Por un lado, están los

modelos transparentes por construcción, como la regresión lineal, cuya lógica es clara desde el inicio. Por otro, encontramos los métodos denominados post-hoc, que actúan como traductores externos para explicar modelos complejos de 'caja negra'. En esta última clasificación destacan herramientas como LIME y [1].

2. Metodología

2.1 Datos, preprocesamiento y modelo

Para aplicar ambos métodos se utilizó el conjunto de datos del Titanic, el cual reúne información descriptiva de algunos pasajeros del famoso trasatlántico y un valor binario que indica si sobrevivieron o no al hundimiento. Este conjunto incluye un identificador único para cada persona (PassengerId), el resultado de supervivencia (Survived), la clase socioeconómica asociada al boleto (Pclass), así como datos personales como nombre, género y edad. También incorpora el número de boleto (Ticket), la cuota pagada (Fare), el número de camarote (Cabin) y el puerto donde cada pasajero embarcó, entre otros. Los datos se obtuvieron del repositorio público de Kaggle en [2].

Para cargar el conjunto de datos y comenzar el análisis, fue necesario preparar primero el entorno de trabajo en Python. Para ello se emplearon bibliotecas ampliamente utilizadas: pandas y numpy para manipular la información, matplotlib y seaborn para crear visualizaciones de los datos, y scikit-learn para aplicar clasificar los datos. Este proceso inicial se ilustra en la Figura 1.

```python
# Suprimir Advertencias: Evita que aparezcan mensajes de advertencia que distraen.
import warnings
warnings.filterwarnings("ignore")

# Matplotlib Inline: Asegura que los gráficos generados por matplotlib se muestren dentro del notebook.
%matplotlib inline

# Importar Librerías de Visualización:
import matplotlib.pyplot as plt # Para gráficos básicos.
import seaborn as sns # Para visualizaciones mejoradas y estadísticas.
sns.set() # Establece un estilo predeterminado atractivo para los gráficos de seaborn.

# Importar la Librería Principal de Aprendizaje Automático (scikit-learn):
import sklearn
from sklearn import preprocessing
from sklearn import metrics
from sklearn import model_selection
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder, OneHotEncoder,
LabelBinarizer
from sklearn.metrics import roc_auc_score, roc_curve, accuracy_score, recall_score, precision_score,
f1_score, precision_recall_curve
from sklearn.model_selection import KFold, StratifiedKFold, GridSearchCV, train_test_split
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression

# Importar Utilidades de Fecha y Hora:
from datetime import datetime, date, timezone, timedelta

# Importar Librerías Numéricas y de Manejo de Datos:
import numpy as np
import pandas as pd

# Importar Gestión de Sistema y Memoria:
import os, gc
# Cargar el dataset del Titanic
df_train = pd.read_csv('https://raw.githubusercontent.com/alarac04/ColabRepo/refs/heads/main/train.csv')
df_test = pd.read_csv('https://raw.githubusercontent.com/alarac04/ColabRepo/refs/heads/main/test.csv')


# Mostrar las primeras filas del dataframe
print(df_train.head())
# Proporciona un resumen del dataframe, incluyendo tipos de datos y valores faltantes.
display(df_train.info())
```

Figura 1: Configuración del entorno y carga de datos

2.2 Construcción y Entrenamiento del Modelo

Una vez completada la configuración del entorno y la carga de los datos, el siguiente paso consiste en construir y entrenar el modelo de aprendizaje automático. En términos simples, un modelo es una herramienta matemática que aprende patrones a partir de los datos para poder hacer predicciones. En esta etapa utilizamos configuraciones predeterminadas que suelen ofrecer un desempeño adecuado. Para la demostración se emplea un clasificador basado en LightGBM [3], que funciona de forma apropiada en el conjunto de datos del Titanic. Este proceso se observa en la Figura 2.

```
!pip uninstall -y lightgbm
!pip cache purge
!pip install --no-cache-dir lightgbm==4.5.0  # Force install 4.5.0
import lightgbm as lgb

exclude_columns = [
    'Name',
    'Ticket',
    'PassengerId',
    'Survived'
] # Columnas a excluir

features = [c for c in df_train_fe.columns if c not in exclude_columns] # Columnas de características
target = df_train_fe['Survived'] # Variable objetivo
print(len(target)) # Imprimir la longitud del objetivo

gc.collect() # Liberación de memoria

X_train, X_test, y_train, y_test = train_test_split(df_train_fe[features], target, test_size=0.2,
random_state=440) # Dividir los datos

param = { # Parametros de LightGBM
    'boost': 'gbdt',
    'learning_rate': 0.008,
    'feature_fraction': 0.20,
    'bagging_freq': 1,
    'bagging_fraction': 1,
    'max_depth': -1,
    'num_leaves': 17,
    'lambda_l2': 0.9,
    'lambda_l1': 0.9,
    'max_bin': 200,
    'metric': ['auc', 'binary_logloss'],
    'tree_learner': 'serial',
    'objective': 'binary',
    'verbosity': 1,
    'seed': 42
}

num_round = 10000 # Numero de rondas de boosting
callbacks = [lgb.early_stopping(stopping_rounds=1000, verbose=1)] # Parada temprana
evals_result = {}

lgb_train = lgb.Dataset(X_train, y_train) # Datos de entrenamiento
lgb_valid = lgb.Dataset(X_test, y_test, reference=lgb_train) # Datos de validación

clf = lgb.train(param, lgb_train, num_round,
                valid_sets=[lgb_train, lgb_valid],
                callbacks=callbacks)

oof = clf.predict(X_test, num_iteration=clf.best_iteration) # Hacer predicciones OOF
print(f'OOF AUC: {roc_auc_score(y_test, oof):.4f}') # Imprimir OOF AUC

# Convertir probabilidades a predicciones binarias (0 o 1)
oof_binary = [1 if p >= 0.5 else 0 for p in oof]

# Calcular e imprimir la precisión
oof_accuracy = accuracy_score(y_test, oof_binary)
print(f'OOF Accuracy: {oof_accuracy:.4f}')

predictions = clf.predict(df_test_fe[features], num_iteration=clf.best_iteration) # Hacer predicciones en
los datos de prueba
print("Predictions on test data (first 5):", predictions[:5]) # Imprimir las primeras 5 predicciones
```

Figura 2: Entrenamiento del modelo.

2.3 LIME

LIME [4] es el acrónimo de Local Interpretable Model-agnostic Explanations. Es una técnica que permite comprender cómo funciona un modelo de aprendizaje a partir de una predicción o ejemplo concreto. LIME construye una versión simplificada del modelo alrededor del ejemplo que se desea analizar.

Para generar esta interpretación, LIME crea variaciones del dato de entrada original y observa cómo cambia la salida del modelo. Posteriormente, simplifica el modelo aproximándolo a una regresión lineal. Este proceso puede expresarse matemáticamente como la búsqueda del modelo interpretable *g* que mejor se aproxima al modelo original f en torno al punto *x:*

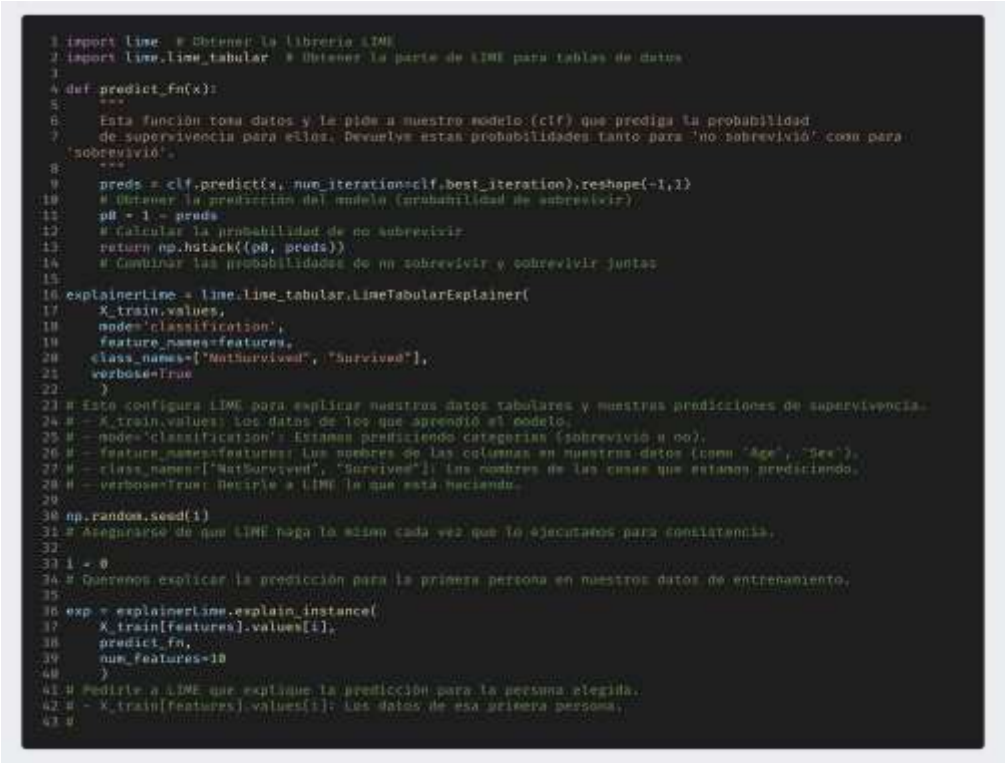
$$\xi(x) = \underset{g \in G}{\operatorname{argmín}} \mathcal{L}(f, g, \pi_x) + \Omega(g)$$

donde:

- *L(f,g,πx)* mide qué tan bien el modelo sencillo *g* imita al modelo real *f* cerca del punto de interés *x*, *πx* es una función que asigna más peso a las muestras que están más cerca de *x*,
- *Ω(g)* penaliza la complejidad del modelo *g* para asegurar que sea fácil de interpretar.

Es decir, LIME construye un acercamiento o ”zoom” sobre un caso particular, creando un modelo simple y más fácil de entender que el modelo más complejo que tomó esa decisión.

En el bloque de código de la Figura 3 se muestra cómo utilizar la biblioteca LIME en Python para generar explicaciones sobre predicciones individuales en el conjunto de datos del Titanic. Para ello se emplea el módulo lime_tabular, que se emplea para interpretar modelos que reciben datos tabulares como entrada. Los comentarios incluidos en el propio código guían paso a paso cada parte del proceso.



```python
import lime  # Obtener la librería LIME
import lime.lime_tabular  # Obtener la parte de LIME para tablas de datos

def predict_fn(x):
    """
    Esta función toma datos y le pide a nuestro modelo (clf) que prediga la probabilidad
    de supervivencia para ellos. Devuelve estas probabilidades tanto para 'no sobrevivió' como para
'sobrevivió'.
    """
    preds = clf.predict(x, num_iteration=clf.best_iteration).reshape(-1,1)
    # Obtener la predicción del modelo (probabilidad de sobrevivir)
    p0 = 1 - preds
    # Calcular la probabilidad de no sobrevivir
    return np.hstack((p0, preds))
    # Combinar las probabilidades de no sobrevivir y sobrevivir juntas

explainerLime = lime.lime_tabular.LimeTabularExplainer(
    X_train.values,
    mode='classification',
    feature_names=features,
   class_names=["NotSurvived", "Survived"],
   verbose=True
    )
# Esto configura LIME para explicar nuestros datos tabulares y nuestras predicciones de supervivencia.
# - X_train.values: Los datos de los que aprendió el modelo.
# - mode='classification': Estamos prediciendo categorías (sobrevivió o no).
# - feature_names=features: Los nombres de las columnas en nuestros datos (como 'Age', 'Sex').
# - class_names=["NotSurvived", "Survived"]: Los nombres de las cosas que estamos prediciendo.
# - verbose=True: Decirle a LIME lo que está haciendo.

np.random.seed(1)
# Asegurarse de que LIME haga lo mismo cada vez que lo ejecutamos para consistencia.

i = 0
# Queremos explicar la predicción para la primera persona en nuestros datos de entrenamiento.

exp = explainerLime.explain_instance(
    X_train[features].values[i],
    predict_fn,
    num_features=10
    )
# Pedirle a LIME que explique la predicción para la persona elegida.
# - X_train[features].values[i]: Los datos de esa primera persona.
#
```

Figura 3: Código de Uso de LIME en Python.

2.4 SHAP

Por su parte, la técnica SHAP [5] es el acrónimo de SHapley Additive exPlanations. Es una técnica que permite interpretar las predicciones de un modelo calculando cuánto aporta cada característica o atributo al resultado final. Está inspirado en la teoría de juegos cooperativos, la cual utiliza los valores de Shapley para definir de manera justa cuánto aporta cada jugador al resultado final del grupo. En el contexto de un modelo de aprendizaje, los "jugadores" representan las variables, y SHAP calcula cuánto suma o resta cada una para obtener la predicción o resultado.
La cantidad SHAP para una característica i se calcula a través de:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!\,(|F| - |S| - 1)!}{|F|!} \left[f(S \cup \{i\}) - f(S) \right]$$

donde:

- *F* representa el conjunto total de características,
- *S* es un subconjunto de características que no incluye a *i, f(S)* simboliza la predicción del modelo usando solo las características en *S*,
- ϕ_i mide la contribución promedio de la característica i en todas las combinaciones posibles.

El valor de Shapley descompone cualquier predicción *f(x)* como una suma aditiva:

$$f(x) = \phi_0 + \sum_{i=1}^{M} \phi_i$$

donde:

- *ϕ0* es el valor base (la predicción promedio del modelo), *ϕi* indica cuánto aporta
- cada característica para alejarse de ese valor base.

De esta forma, SHAP muestra qué variables influyeron más en una predicción específica y en qué dirección.

En la Figura 4 se muestra cómo emplear la biblioteca shap para generar explicaciones basadas en valores de Shapley. En particular, la función shap.force_plot permite visualizar de manera intuitiva cómo cada característica influye en la predicción de un caso específico, mostrando cómo cada característica contribuye a empujar la predicción desde el valor base (predicción promedio) hasta la predicción real.

```
import shap  # Obtener SHAP para explicar las predicciones del modelo
shap.initjs()  # Configurar SHAP para mostrar gráficos en el notebook

# Crear un explicador SHAP para nuestro modelo LightGBM ('clf')
# Le dice a SHAP cómo nuestro modelo basado en árboles toma decisiones
# 'raw' output significa que queremos la puntuación directa del modelo
explainer = shap.TreeExplainer(clf, model_output='raw')

# Calcular cuánto contribuyó cada característica a cada predicción en nuestros datos de entrenamiento
shap_values = explainer.shap_values(X_train)
instance_ind=0
# Mostrar un gráfico explicando la predicción para la primera persona en nuestros datos de entrenamiento
# Muestra como el valor de cada característica empujó la predicción hacia arriba o hacia abajo desde la
  predicción promedio
shap.force_plot(explainer.expected_value, shap_values[0], X_train.iloc[instance_ind,:])
```

Figura 4: Código para generar gráfica de fuerza SHAP en Python.

A diferencia de LIME, SHAP permite obtener una visión global sobre qué características son más importantes para el modelo en todo el conjunto de datos y, además, muestra cómo se distribuye su impacto en las predicciones. Para ello puede utilizarse la función shap.summary_plot, que resume visualmente la influencia de cada variable de manera clara. El código que genera esta gráfica se presenta en la Figura 5.

```
import shap  # Para explicar las predicciones del modelo

# Mostrar un resumen de qué características fueron más importantes y cómo afectaron las predicciones
shap.summary_plot(shap_values, X_train)

# Mostrar un gráfico de barras más simple de la importancia de las características
shap.summary_plot(shap_values, X_train, plot_type='bar')
```

Figura 5: Código para generar Gráfica de resumen de características SHAP en Python.

3. Resultados y discusión

En este apartado se presentan algunas gráficas obtenidas al aplicar las técnicas LIME y SHAP al conjunto de datos del Titanic, utilizando los códigos en Python descritos previamente.

La Figura 6 presenta una explicación local generada con la técnica LIME. En este caso, el modelo estima una probabilidad del 71 % de que el pasajero sobreviviera, y LIME identifica que factores importantes como ser mujer, pertenecer a una clase alta y tener camarote en cierta zona contribuyen positivamente a esta conclusión.

Figura 6: Gráfica LIME tabular.

La Figura 7 muestra una gráfica de fuerza SHAP que explica las razones por las que un pasajero al azar del Titanic tuvo altas posibilidades de sobrevivir. La gráfica parte de un valor promedio del modelo y muestra cómo cada característica del pasajero empuja la predicción hacia arriba (en rojo) o hacia abajo (en azul). En este caso, variables como *Cabin = 5, Title = 3, Sex = 0 y Pclass = 1* contribuyen claramente a aumentar la probabilidad estimada de supervivencia, hasta llegar a un valor final de *f(x) = 0,88.*

Figura 7: Gráfica de fuerza SHAP.

Por último, la Figura 8 muestra un SHAP summary plot que explica cómo cada característica del conjunto influye en la predicción de supervivencia. Las barras indican la importancia global de las variables, destacando el nivel social, el género, la ubicación del camarote y el tipo del boleto como las más relevantes. El diagrama superior muestra, para cada pasajero, si el valor de una característica aumenta o disminuye su probabilidad de sobrevivir: los puntos rojos representan valores altos y los azules valores bajos. Así, ser mujer o viajar en primera clase suele desplazar los puntos hacia la derecha (mayor probabilidad de supervivencia), mientras que ser hombre o pertenecer a clases bajas los desplaza hacia la izquierda (menor probabilidad).

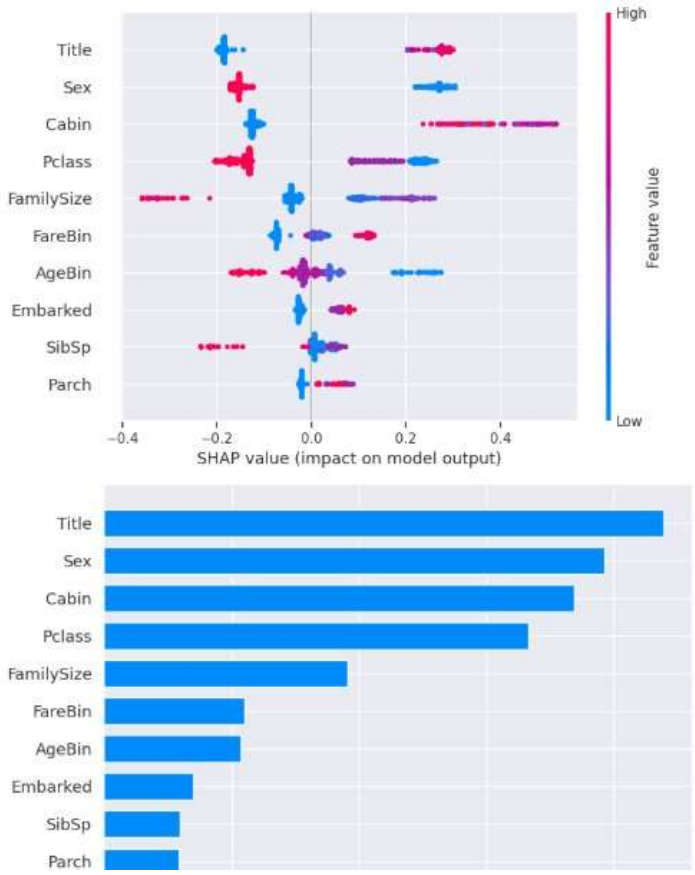


Figura 8: Gráfica de resumen de características SHAP.

4. Conclusiones

En este trabajo se aplicaron las técnicas LIME y SHAP para interpretar el conjunto de datos del Titanic. Ambas herramientas permiten aportar mayor transparencia a los modelos de aprendizaje automático considerados como "cajas negras". Las dos coinciden en señalar que el sexo, la clase social y la edad fueron factores determinantes en la supervivencia. LIME destaca por su claridad y rapidez para ofrecer explicaciones locales, mientras que SHAP proporciona una base matemática más sólida y permite obtener también una visión global del modelo. Es importante señalar que estas técnicas pueden trasladarse a otros contextos, pero sus explicaciones no deben interpretarse como relaciones causales.

Referencias

[1] Adadi, A., & Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial

intelligence (XAI). IEEE Access, 6, 52138–52160. https://doi.org/10.1109/ACCESS.2018.2870052

[2] Yasser, H. (s.f.). Titanic Dataset [Conjunto de datos]. Kaggle. Recuperado de https://www.kaggle.com/datasets/yasserh/titanic-dataset

[3] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. In Advances in Neural Information Processing Systems 30 (pp. 3146– 3154). Curran Associates, Inc.

[4] LIME: Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why should I trust you?": Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 1135–1144). ACM. https://doi.org/10.1145/2939672.2939778

[5] SHAP: Lundberg, S. M., & Lee, S. I. (2017). A unified approach to interpreting model predictions. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), Advances in Neural Information Processing Systems 30 (pp. 4765–4774). Curran Associates, Inc.

Referencia

Lara, A. & Reyes, E. *(marzo - abril, 2026)* Inteligencia artificial explicable (xai) con lime y shap: interpretación del modelo del TITANIC. *Boletín UPIITA. año 20, (11.) 2026* https://www.boletin.upiita.ipn.mx/index.php/ciencia/1105-cyt-numero-113/2484-inteligencia-artificial-explicable-xai-con-lime-y-shap-interpretacion-del-modelo-del-titanic