



Universidad
de Navarra

DATAI
INSTITUTO DE CIENCIA DE LOS
DATOS E INTELIGENCIA ARTIFICIAL

DETECCIÓN DE TRANSACCIONES FRAUDULENTAS EN EL CONTEXTO DE LA CRIPTOMONEDA ETHEREUM

Trabajo Fin de Máster
Máster Universitario en Big Data Science

Curso académico 2023-2024

AUTOR: Sergio Alejandro Marchena Gordillo
TUTOR ACADÉMICO: Stella Maris Salvatierra Galiano
TUTOR DE EMPRESA: Gabriela Castillo Areco
Madrid, 29 Agosto de 2024

Índice

Resumen del trabajo.....	4
Introducción.....	5
Objetivos.....	6
Enfoque adoptado.....	6
Planificación.....	7
Material empleado.....	7
Estado del arte.....	9
Materiales y Métodos.....	11
Análisis de los datos.....	11
Estructura de los datos.....	12
Ingesta de datos, fuentes internas y externas.....	13
Auditoría y procesamiento de datos.....	13
Estudio previo.....	13
Limpieza y normalización de datos.....	14
Resultados.....	24
Descripción de algoritmos usados.....	27
1. Regresión Logística.....	27
Resultados Obtenidos:.....	27
2. Random Forest.....	27
Resultados Obtenidos:.....	28
3. XGBoost.....	28
Resultados Obtenidos:.....	28
Ajustes y explicación de los parámetros.....	28
Discusión.....	30
Conclusiones y trabajo futuro.....	32
Glosario.....	33
Bibliografía.....	34
Anexos.....	36

Resumen del trabajo

Este trabajo aborda el desarrollo de un modelo predictivo para la detección de transacciones fraudulentas en la red de Ethereum, utilizando un enfoque basado en el análisis de datos y el aprendizaje automático. Partiendo de un conjunto de datos de transacciones históricas, se realizaron varias etapas de procesamiento y análisis de datos, que incluyeron la limpieza y preparación del conjunto de datos, el análisis exploratorio de datos (EDA), la normalización de variables y el manejo del desbalance de clases mediante la técnica SMOTE (Synthetic Minority Over-sampling Technique).

Durante el análisis exploratorio, se eliminaron características irrelevantes y se examinó la correlación entre variables para evitar problemas de multicolinealidad. También se identificaron y eliminaron variables con varianza nula o igual a 0, mejorando la eficiencia y eficacia del modelo. Dado el desbalance significativo en la variable objetivo, donde solo el 22% de las transacciones eran fraudulentas, se utilizó SMOTE para equilibrar las clases, lo que permitió entrenar un modelo más robusto.

Se evaluaron múltiples algoritmos de aprendizaje automático, incluyendo Regresión Logística, Random Forest y XGBoost, siendo este último el que ofreció los mejores resultados. Tras optimizarse mediante Grid Search, el modelo XGBoost alcanzó un recall superior al 95% y una precisión global del 98%, demostrando una gran efectividad en la detección de fraudes, especialmente al reducir los errores de tipo II (falsos negativos).

El uso de un algoritmo predictivo como el desarrollado en este estudio tiene múltiples beneficios potenciales en la vida real. Implementar un modelo capaz de identificar transacciones fraudulentas en tiempo real puede mejorar significativamente la seguridad de la red de Ethereum, protegiendo a los usuarios contra pérdidas financieras y aumentando la confianza en las plataformas de criptomonedas. Además, este tipo de tecnología podría reducir la necesidad de intervención manual en la supervisión de transacciones, permitiendo una detección de fraudes más rápida y escalable. En última instancia, la adopción de soluciones predictivas en la detección de fraudes podría contribuir al desarrollo de un entorno más seguro y confiable en el creciente mercado de las criptomonedas.

Introducción

El uso de criptomonedas, especialmente Ethereum, ha experimentado un crecimiento exponencial en los últimos años, convirtiéndose en una plataforma líder para transacciones descentralizadas y contratos inteligentes. Sin embargo, este crecimiento ha venido acompañado de un aumento en las actividades fraudulentas, lo que subraya la necesidad urgente de desarrollar métodos eficaces para la detección de fraudes en tiempo real (Nakamoto, 2008). La detección de transacciones fraudulentas es crucial no solo para proteger a los usuarios individuales, sino también para mantener la integridad del sistema y la confianza en las plataformas de blockchain (Mearian, 2018).

Este trabajo se llevó a cabo con el objetivo de desarrollar un modelo predictivo que pueda identificar transacciones fraudulentas en la red de Ethereum de manera efectiva. Se emplearon técnicas avanzadas de análisis de datos y aprendizaje automático para abordar este desafío. El conjunto de datos utilizado incluye información histórica sobre transacciones, tanto fraudulentas como no fraudulentas, y fue procesado a través de varias etapas clave: limpieza y preparación de datos, análisis exploratorio de datos (EDA), normalización de características y manejo del desbalance de clases mediante técnicas como SMOTE (Synthetic Minority Over-sampling Technique) .

La importancia de utilizar SMOTE radica en su capacidad para abordar el problema del desbalance de clases, un fenómeno común en la detección de fraudes donde las instancias de fraude son significativamente menores en comparación con las transacciones legítimas (Chawla et al., 2002). SMOTE genera ejemplos sintéticos de la clase minoritaria, lo que permite entrenar modelos más equilibrados y eficaces. Además, para garantizar que las características numéricas estuvieran en una escala comparable, se aplicó 'PowerTransformer', una técnica que estabiliza la varianza y convierte los datos en una distribución más normalizada (Scikit-learn developers, 2023).

En este estudio, se evaluaron varios modelos de aprendizaje automático, incluyendo Regresión Logística, Random Forest y XGBoost, seleccionando este último como el más efectivo debido a su capacidad para manejar datos de alta dimensionalidad y complejidad (Chen & Guestrin, 2016). XGBoost se destaca por su enfoque basado en árboles de decisión y su capacidad de optimización, lo que le permite manejar eficientemente los conjuntos de datos grandes y complejos típicos de las transacciones en blockchain (Bishop, 2006). Tras la optimización de hiperparámetros mediante Grid Search, el modelo XGBoost logró un recall superior al 95% y una precisión general del 98%, lo que indica su alta efectividad para la detección de fraudes .

Estos resultados sugieren que la implementación de un modelo predictivo como el desarrollado en este trabajo podría ofrecer beneficios significativos en la práctica. En particular, la capacidad de identificar transacciones fraudulentas en tiempo real podría mejorar la seguridad de la red Ethereum, proteger a los usuarios contra pérdidas financieras y aumentar la confianza en las plataformas de criptomonedas (Sujata & Sahay, 2021). Además, al reducir la necesidad de intervención manual en la supervisión de transacciones, este tipo de soluciones automatizadas pueden ofrecer una detección de fraudes más rápida y escalable, contribuyendo a un entorno de blockchain más seguro y confiable .

Objetivos

- Construir y entrenar un modelo de aprendizaje automático que identifique transacciones fraudulentas en Ethereum con al menos un 95% de precisión y una tasa de falsos positivos inferior al 3%, evaluado mediante una validación cruzada de 10 pliegues.
- Implementar técnicas avanzadas de preprocesamiento, incluyendo normalización de datos, manejo de valores faltantes, y selección de características, para optimizar el rendimiento del modelo.
- Comparar la efectividad de al menos tres algoritmos de clasificación utilizando métricas como precisión, recall y F1-score, y seleccionar el algoritmo con el mejor desempeño general.

Enfoque adoptado

- Personalización de Algoritmos: Selección y ajuste de algoritmos de aprendizaje automático que son particularmente efectivos para manejar datos de alta dimensionalidad y desequilibrio de clases, como podría ser el caso con datos de transacciones en Ethereum.
- Ajuste de Parámetros: Modificación de hiperparámetros en los modelos para optimizarlos específicamente para el tipo de datos disponible, en lugar de usar configuraciones por defecto que podrían no ser óptimas
- Selección de Características: Aplicar técnicas de selección de características adaptadas a las particularidades del conjunto de datos (como el tipo de transacciones o la naturaleza de las variables) para mejorar el rendimiento del modelo
- Manejo de Desbalance de Clases: Implementar técnicas como SMOTE o submuestreo adaptadas para manejar el desbalance en la cantidad de transacciones fraudulentas versus las no fraudulentas.

Planificación

El proyecto se planificó en varias fases:

1. Carga de los Datos
 - a. Carga del Dataset
 - b. Vista general de Datos
 - c. Eliminación de Columnas Irrelevantes
2. Análisis Exploratorio de Datos (EDA)
 - a. Análisis de Tipos de Datos y Valores Faltantes
 - b. Análisis de Variables Categóricas y Numéricas
 - c. Distribución variable objetivo.
 - d. Cálculo de la Varianza y Eliminación de Variables con Varianza Cero
 - e. Análisis de la Matriz de Correlación y Selección de Variables
3. Limpieza y Transformación de Datos
 - a. Imputación de Datos Faltantes
 - b. Varianza igual a 0.
 - c. Correlación entre variables.
 - d. Eliminación de Variables con Distribución Reducida
4. Preparación de los datos
 - a. Normalización de Variables
 - b. Aplicación de SMOTE para Balancear Clases
5. Desarrollo de Modelos
 - a. Entrenamiento de Modelos Iniciales (Regresión Logística, Random Forest, XGBoost)
 - b. Evaluación Inicial de Modelos
6. Optimización de Modelos
 - a. Ajuste de Hiperparámetros mediante Grid Search
 - b. Evaluación y Comparación de Modelos Optimizados
7. Evaluación Final y Validación del Modelo
 - a. Análisis de Métricas de Evaluación
 - b. Visualización de la Matriz de Confusión y Curva AUC
8. Documentación y Presentación de Resultados
 - a. Documentación del Proceso y Resultados
 - b. Preparación del Informe Final.

Material empleado

Para la realización de este trabajo de detección de transacciones fraudulentas en Ethereum, se emplearon los siguientes materiales:

1. Librerías de Python:
 - a. Pandas: Utilizada para la manipulación y análisis de datos. Facilita la carga del dataset, limpieza de datos, y operaciones como la selección y filtrado de características.
 - b. Numpy: Empleada para operaciones matemáticas y manejo de arrays, útil para la preparación de datos y cálculos numéricos.
 - c. Matplotlib y Seaborn: Utilizadas para la visualización de datos, incluidas gráficas de distribución, matrices de correlación y gráficos de boxplot para análisis exploratorio.

- d. Scikit-learn: Proporciona herramientas para la selección de características, normalización de datos (como 'PowerTransformer'), creación de modelos de aprendizaje automático (Regresión Logística, Random Forest, entre otros), y evaluación de modelos mediante métricas como precisión, recall, F1-score y matrices de confusión.
 - e. XGBoost: Utilizada para implementar el algoritmo de clasificación XGBoost, reconocido por su alta eficiencia en el manejo de datos complejos y de alta dimensionalidad.
 - f. Imbalanced-learn (SMOTE): Esta librería fue crucial para abordar el desbalance de clases en el dataset, permitiendo el uso de la técnica SMOTE (Synthetic Minority Over-sampling Technique) para equilibrar las clases en el conjunto de datos de entrenamiento.
 - g. Pickle: Utilizada para guardar y cargar modelos entrenados, facilitando su reutilización y aplicación en diferentes escenarios.
2. Programas y Entornos de Desarrollo
- a. Jupyter Notebook: El entorno principal utilizado para el desarrollo y ejecución del código. Jupyter facilita la escritura de código interactivo y la visualización inmediata de los resultados, lo que es ideal para el análisis exploratorio de datos y el desarrollo de modelos.
 - b. Python: El lenguaje de programación principal utilizado para todo el análisis de datos y desarrollo de modelos. Se utilizó un entorno virtual con la versión de Python 3.9.18.
 - c. Además se utilizó para el control de versiones, un repositorio en Github: https://github.com/sergiomarchena16/TFM_2024/
3. Dataset
- a. Ethereum Fraud Detection Dataset: El conjunto de datos principal utilizado en este trabajo. Este dataset incluye información histórica sobre transacciones realizadas en la red Ethereum, etiquetadas como fraudulentas o no fraudulentas. El dataset fue descargado de Kaggle y contiene múltiples características relevantes, como el tiempo promedio entre transacciones, balances de Ether, y métricas relacionadas con tokens ERC20.
4. Bibliotecas y Recursos
- a. Documentación Oficial: Se hizo referencia a la documentación oficial de las librerías utilizadas, como la documentación de Scikit-learn para 'PowerTransformer' y SMOTE, así como la documentación de XGBoost, para entender mejor las funcionalidades y parámetros disponibles.
 - b. Literatura Académica y Blogs: Se consultaron artículos académicos, libros, y blogs especializados en aprendizaje automático y blockchain, como el artículo original de SMOTE y recursos sobre la implementación de XGBoost en problemas de clasificación.
5. Hardware y Software
- a. Computadora Personal: Con capacidad suficiente para manejar grandes volúmenes de datos y realizar el entrenamiento de modelos de aprendizaje automático.
 - i. Macbook Pro M1 2020 8 GB RAM.
 - b. Sistema Operativo: Utilización de un sistema operativo compatible con Jupyter Notebook y Python, generalmente Linux, macOS o Windows.
 - c. Entorno: Microsoft Visual Studio Code 1.92.2.

Estado del arte

La detección de fraudes en sistemas financieros, particularmente en la red de Ethereum, ha captado una considerable atención en la investigación debido a la creciente adopción de criptomonedas y la naturaleza descentralizada de la tecnología blockchain. Ethereum, al ser una plataforma líder no solo para transacciones sino también para contratos inteligentes, presenta desafíos únicos en la identificación de actividades fraudulentas, lo que ha llevado a un interés significativo en la aplicación de técnicas de aprendizaje automático (machine learning) para abordar este problema.

Investigaciones y Trabajos Previos

Uno de los enfoques más tempranos en la detección de fraudes en criptomonedas fue propuesto por Vasek y Moore (2015), quienes exploraron las vulnerabilidades de Bitcoin, el precursor de Ethereum. Aunque este trabajo se centró en Bitcoin, estableció las bases para aplicar técnicas similares en Ethereum, dado que ambas plataformas comparten principios de operación en blockchain.

En el contexto específico de Ethereum, Chen et al., (2018) realizaron un estudio exhaustivo sobre la detección de fraudes utilizando machine learning, donde exploraron diferentes algoritmos como Support Vector Machines (SVM), árboles de decisión, y redes neuronales. Su investigación demostró que los métodos de aprendizaje supervisado pueden ser altamente efectivos para identificar transacciones fraudulentas, especialmente cuando se combinan con técnicas de preprocesamiento adecuadas para manejar datos desequilibrados.

Otra contribución significativa fue realizada en 2019, donde se desarrolla un sistema de detección de fraudes basado en técnicas de aprendizaje profundo. Utilizando redes neuronales convolucionales (CNN) adaptadas a la estructura de datos transaccionales en blockchain, lograron una alta precisión en la identificación de fraudes. Sin embargo, este enfoque requiere un volumen de datos y recursos computacionales considerablemente mayor, lo que puede ser una limitación para implementaciones en tiempo real (Victor et al., 2019).

Más recientemente, se presenta un estudio comparativo de diferentes algoritmos de clasificación aplicados a la detección de fraudes en Ethereum, incluyendo Random Forest, XGBoost y LightGBM. Su investigación mostró que XGBoost, debido a su capacidad para manejar datos de alta dimensionalidad y su eficiencia en el procesamiento, superó a otros modelos en términos de precisión y recall. Este estudio subraya la importancia de seleccionar algoritmos que no solo sean precisos, sino que también puedan adaptarse a las características únicas de los datos de blockchain (Feng et al., 2020).

En cuanto a la literatura técnica, Christopher Bishop en su libro *“Pattern Recognition and Machine Learning”* establece los fundamentos del aprendizaje automático, proporcionando una base teórica sólida que ha sido aplicada en diversos dominios, incluido el de la detección de fraudes. Su trabajo es una referencia clave para entender cómo adaptar los algoritmos de machine learning a diferentes tipos de datos, como los transaccionales en Ethereum.

En la misma línea, el libro *“Mastering Ethereum”* de Andreas M. Antonopoulos y Gavin Wood (2018) ofrece una comprensión profunda de la arquitectura de Ethereum, lo cual es esencial para cualquier investigación orientada a la detección de fraudes en esta plataforma. Este libro no solo cubre

los aspectos técnicos de Ethereum, sino que también aborda los desafíos de seguridad, proporcionando un contexto valioso para la implementación de modelos de detección de fraudes.

Los blogs y recursos en línea también han sido fundamentales para la difusión de técnicas y prácticas emergentes en la detección de fraudes. Un ejemplo destacado es el blog *“Machine Learning Mastery”* de Jason Brownlee, que ofrece guías detalladas sobre la implementación de técnicas como SMOTE para manejar el desbalance de clases en conjuntos de datos, una técnica crucial en la detección de fraudes donde las transacciones fraudulentas son relativamente raras.

Además, plataformas como Kaggle han facilitado la disponibilidad de datasets y competencias centradas en la detección de fraudes en criptomonedas, lo que ha impulsado la investigación colaborativa y el desarrollo de modelos avanzados. Las competencias en Kaggle han demostrado ser un campo fértil para la experimentación con diferentes enfoques de machine learning aplicados a problemas de fraude, contribuyendo a la evolución de las mejores prácticas en la comunidad.

El estado del arte en la detección de fraudes en Ethereum muestra una evolución constante en la aplicación de técnicas de aprendizaje automático, desde métodos supervisados clásicos hasta enfoques más avanzados como el aprendizaje profundo y modelos de ensamblado. La investigación actual respalda el uso de algoritmos como XGBoost, que han demostrado un rendimiento superior en la detección de fraudes en blockchain debido a su capacidad para manejar grandes volúmenes de datos y su flexibilidad para adaptarse a diversas estructuras de datos (Feng et al., 2020). Sin embargo, la elección del modelo adecuado depende en gran medida del contexto y las limitaciones específicas, como la necesidad de detección en tiempo real y la disponibilidad de recursos computacionales.

Materiales y Métodos

Análisis de los datos

El análisis de los datos es un paso fundamental para entender la naturaleza y distribución de los datos con los que se trabajará. En el notebook, se realiza un análisis exploratorio utilizando técnicas de visualización y estadísticas descriptivas para identificar patrones, tendencias y posibles anomalías. Esto incluye la inspección de las variables de entrada, la distribución de las clases objetivo (fraude o no fraude), y la identificación de cualquier desequilibrio en los datos. Se emplean gráficos como histogramas y diagramas de caja (boxplots) para visualizar la distribución de los datos y detectar valores atípicos.

Técnicas Empleadas

1. Carga de datos
 - Carga del Dataset.
2. Análisis Exploratorio de Datos (EDA):
 - Descripción de Variables.
 - Distribución variable objetivo.
 - Varianza.
 - Matriz de Correlación.
3. Limpieza de datos:
 - Datos faltantes
 - Varianza igual a 0.
 - Correlación entre variables.
 - Distribución de variables restantes.
4. Preparación de los datos.
 - Normalización
 - Manejo del Desbalance de Clases con SMOTE.

Gráficos Utilizados

1. Distribución de la Variable Objetivo (FLAG):
 - Gráfico de Pastel.
2. Matriz de Correlación y Datos faltantes
 - Mapa de Calor (Heatmap)
3. Distribuciones de las variables:
 - Gráficos de Caja (Boxplots)
4. Matriz de Confusión.
5. Curva AUC.

El dataset contiene 9,841 filas y 50 columnas. Entre las variables se incluyen tiempos promedio entre transacciones enviadas y recibidas, número total de transacciones, balances de Ether y varias métricas relacionadas con los tokens ERC20.

Estructura de los datos

La estructura de los datos se refiere a cómo están organizados los datos dentro del conjunto utilizado. En este caso, los datos suelen estar estructurados en formato tabular, donde cada fila representa una transacción y cada columna representa una característica específica de esa transacción. Es crucial entender la estructura para poder aplicar correctamente los modelos de machine learning. En el notebook, se realiza un análisis de las variables categóricas y numéricas, identificando las que son relevantes para la detección de fraude. Además, se analiza la presencia de datos faltantes y se toman decisiones sobre cómo manejarlos.

A continuación, se describe el contenido de las filas del conjunto de datos:

- Index: the index number of a row
- Address: the address of the ethereum account
- FLAG: whether the transaction is fraud or not
- Avg min between sent txn: Average time between sent transactions for account in minutes
- Avg min between received txn: Average time between received transactions for account in minutes
- Time Diff between first and _last (Mins): Time difference between the first and last transaction
- Sent_txn: Total number of sent normal transactions
- Received_txn: Total number of received normal transactions
- NumberofCreated_Contracts: Total Number of created contract transactions
- UniqueReceivedFrom_Addresses: Total Unique addresses from which account received transaction
- UniqueSentTo_Addresses20: Total Unique addresses from which account sent transactions
- MinValueReceived: Minimum value in Ether ever received
- MaxValueReceived: Maximum value in Ether ever received
- AvgValueReceived5Average value in Ether ever received
- MinValSent: Minimum value of Ether ever sent
- MaxValSent: Maximum value of Ether ever sent
- AvgValSent: Average value of Ether ever sent
- MinValueSentToContract: Minimum value of Ether sent to a contract
- MaxValueSentToContract: Maximum value of Ether sent to a contract
- AvgValueSentToContract: Average value of Ether sent to contracts
- TotalTransactions(IncludingTxn>Create_Contract): Total number of transactions
- TotalEtherSent: Total Ether sent for account address
- TotalEtherReceived: Total Ether received for account address
- TotalEtherSent_Contracts: Total Ether sent to Contract addresses
- TotalEtherBalance: Total Ether Balance following enacted transactions
- TotalERC20Txns: Total number of ERC20 token transfer transactions
- ERC20TotalEther_Received: Total ERC20 token received transactions in Ether
- ERC20TotalEther_Sent: Total ERC20 token sent transactions in Ether
- ERC20TotalEtherSentContract: Total ERC20 token transfer to other contracts in Ether
- ERC20UniqSent_Addr: Number of ERC20 token transactions sent to Unique account addresses
- ERC20UniqRec_Addr: Number of ERC20 token transactions received from Unique addresses
- ERC20UniqRecContractAddr: Number of ERC20 token transactions received from Unique contract addresses
- ERC20AvgTimeBetweenSent_Tnx: Average time between ERC20 token sent transactions in minutes

- ERC20AvgTimeBetweenRec_Tnx: Average time between ERC20 token received transactions in minutes
- ERC20AvgTimeBetweenContract_Tnx: Average time ERC20 token between sent token transactions
- ERC20MinVal_Rec: Minimum value in Ether received from ERC20 token transactions for account
- ERC20MaxVal_Rec: Maximum value in Ether received from ERC20 token transactions for account
- ERC20AvgVal_Rec: Average value in Ether received from ERC20 token transactions for account
- ERC20MinVal_Sent: Minimum value in Ether sent from ERC20 token transactions for account
- ERC20MaxVal_Sent: Maximum value in Ether sent from ERC20 token transactions for account
- ERC20AvgVal_Sent: Average value in Ether sent from ERC20 token transactions for account
- ERC20UniqSentTokenName: Number of Unique ERC20 tokens transferred
- RC20UniqRecTokenName: Number of Unique ERC20 tokens received
- ERC20MostSentTokenType: Most sent token for account via ERC20 transaction
- ERC20MostRecTokenType: Most received token for account via ERC20 transactions

Ingesta de datos, fuentes internas y externas

La ingesta de datos se refiere al proceso de importar y cargar datos desde diferentes fuentes para su posterior análisis. En este proyecto, los datos son importados desde una fuente externa proporcionada por Kaggle. Este paso incluye la carga de los datos utilizando bibliotecas como Pandas, que permiten manipular grandes conjuntos de datos de manera eficiente. El notebook no solo se limita a cargar los datos, sino que también verifica la integridad de los mismos, asegurándose de que todos los registros se han cargado correctamente y no hay problemas como duplicados.

Auditoría y procesamiento de datos

La auditoría de datos es el proceso de revisión detallada para asegurar la calidad y consistencia de los datos. En el notebook, este proceso incluye la identificación de datos anómalos, la verificación de la coherencia de las entradas y la revisión de los valores atípicos. El procesamiento de datos se refiere a las transformaciones necesarias para preparar los datos para el modelado, tales como la codificación de variables categóricas, la normalización de datos numéricos, y la creación de nuevas características que puedan mejorar el rendimiento del modelo.

Estudio previo

El estudio previo involucra la revisión de literatura y técnicas anteriores utilizadas en la detección de fraude, con el fin de identificar enfoques exitosos y áreas de mejora. En este caso, se investigaron modelos de machine learning como Random Forest, XGBoost y técnicas de ensamblado, que han demostrado ser eficaces en la identificación de patrones de fraude. También se consideró el uso de técnicas de balanceo de clases, dado el desequilibrio típico en los conjuntos de datos de fraude.

Limpieza y normalización de datos

La limpieza de datos es el proceso de eliminar o corregir registros incorrectos, incompletos o irrelevantes. En el notebook, se emplean diversas técnicas para manejar datos faltantes, como la imputación o eliminación de registros. Además, se eliminan los valores atípicos que podrían sesgar los resultados del modelo. La normalización de datos, por otro lado, implica escalar las características numéricas a un rango común, lo cual es crucial cuando se utilizan algoritmos que son sensibles a la escala de los datos, como los métodos basados en distancia (por ejemplo, K-Nearest Neighbors).

En conjunto, estos procesos aseguran que los datos estén en la mejor condición posible para el entrenamiento de modelos de machine learning, lo que es esencial para obtener resultados precisos y confiables en la detección de fraude.

El dataset estaba compuesto principalmente por características numéricas (de tipo float64 e int64) y dos características categóricas (ERC20 most sent token type y ERC20_most_rec_token_type). Estas características categóricas se convirtieron en tipo category para facilitar su manejo y análisis posterior.

Para comenzar con la limpieza se hicieron análisis de las variables categóricas y de las variables numéricas. El primera paso fue la inspección de los datos únicos en variables categóricas y el método de Pandas de descripción para las variables numéricas:

Variables categoricas

Valores unicos

```
for i in df[categories].columns:
    print(i, "----> ", len(df[i].value_counts()), ' valores unicos')
```

ERC20 most sent token type ----> 304 valores unicos
ERC20_most_rec_token_type ----> 466 valores unicos

Variables numericas

```
numericals = df.select_dtypes(include=['float', 'int']).columns
df[numericals].describe()
```

	FLAG	Avg min between sent tnx	Avg min between received tnx	Time Diff between first and last (Mins)	Sent tnx	Received Tnx	Number of Created Contracts	Unique Received From Addresses	Ur
count	9841.000000	9841.000000	9841.000000	9.841000e+03	9841.000000	9841.000000	9841.000000	9841.000000	98
mean	0.221421	5086.878721	8004.851184	2.183333e+05	115.931714	163.700945	3.729702	30.360939	
std	0.415224	21486.549974	23081.714801	3.229379e+05	757.226361	940.836550	141.445583	298.621112	2
min	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	3.169300e+02	1.000000	1.000000	0.000000	1.000000	
50%	0.000000	17.340000	509.770000	4.663703e+04	3.000000	4.000000	0.000000	2.000000	
75%	0.000000	565.470000	5480.390000	3.040710e+05	11.000000	27.000000	0.000000	5.000000	
max	1.000000	430287.670000	482175.490000	1.954861e+06	10000.000000	10000.000000	9995.000000	9999.000000	92

8 rows x 46 columns

Se examinaron los valores únicos de las características categóricas, revelando 304 valores únicos para ERC20 most sent token type y 466 valores únicos para ERC20_most_rec_token_type. Este análisis es útil para entender la diversidad de tipos de tokens involucrados en las transacciones.

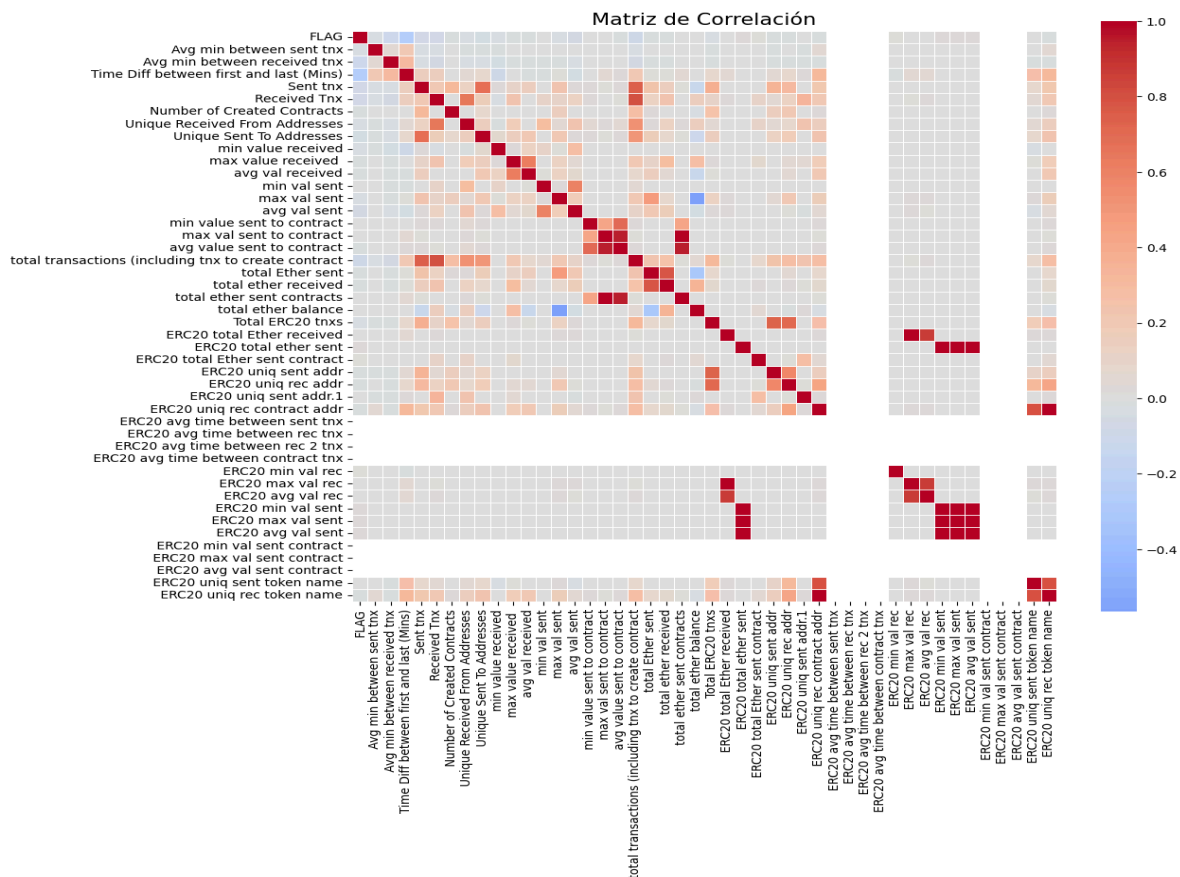
Se calcularon las estadísticas descriptivas (media, mediana, desviación estándar, etc.) para las características numéricas. Este paso reveló una amplia variabilidad en los datos, con algunas características como ERC20 total Ether received y ERC20 max val rec mostrando una varianza extremadamente alta.

Varianza: Se identificaron características con varianza igual a cero:

- ERC20 avg time between rec tnx
- ERC20 avg time between rec 2 tnx
- ERC20 avg time between contract tnx
- ERC20 avg time between sent tnx
- ERC20 min val sent contract
- ERC20 max val sent contract
- ERC20 avg val sent contract

Lo cual significa que todos los valores de esa característica son idénticos. En otras palabras, no hay ninguna variabilidad en estas columnas: cada valor es el mismo en todas las observaciones. Esto indica que la característica no proporciona ninguna información útil para distinguir entre las diferentes observaciones, y por lo tanto, generalmente puede ser eliminada del análisis o modelado, ya que no contribuirá a la predicción o clasificación.

Luego, se calculó una matriz de correlación para examinar las relaciones entre las características numéricas. La correlación alta entre características puede llevar a problemas de multicolinealidad en los modelos predictivos, por lo que se eliminaron aquellas variables que estaban altamente correlacionadas entre sí.



La multicolinealidad ocurre cuando dos o más variables están altamente correlacionadas entre sí, lo que puede causar problemas en los modelos predictivos, especialmente en aquellos que asumen independencia entre las variables, como la regresión logística.

En la matriz de correlación observada en el EDA, se identificaron variables con alta correlación, lo que llevó a la eliminación de algunas de ellas para evitar problemas de multicolinealidad. Por ejemplo, si dos variables tienen una correlación cercana a 1 o -1, significa que una puede ser redundante con respecto a la otra, y mantener ambas podría no aportar valor adicional al modelo.

Relaciones entre Variables Relacionadas con las Transacciones:

- total Ether sent vs. total ether received: Es probable que estas dos variables estén positivamente correlacionadas. En cuentas donde se realizan muchas transacciones, generalmente se observa tanto un alto envío como una alta recepción de Ether. Sin embargo, si esta correlación es muy alta, podría ser redundante incluir ambas en el modelo, y una de ellas podría ser eliminada.
- avg val received vs. max value received: Estas dos variables probablemente también mostrarán una correlación positiva, ya que cuentas que reciben valores altos en una sola transacción también tienden a tener un alto valor promedio recibido. Si la correlación es demasiado alta, podría ser beneficioso eliminar una para simplificar el modelo.

Impacto en la Variable Objetivo (FLAG):

- La correlación entre las variables predictoras y la variable objetivo FLAG es crucial. Variables que tienen una correlación significativa con FLAG (positiva o negativa) son candidatas fuertes para ser incluidas en el modelo predictivo. Por ejemplo, si avg val received tiene una correlación notable con FLAG, podría indicar que cuentas que reciben valores altos tienen más probabilidad de ser fraudulentas.
- Sin embargo, la mayoría de las variables probablemente tendrán una correlación baja con FLAG debido a la naturaleza compleja del fraude, que a menudo no se relaciona directamente con una sola característica, sino con una combinación de ellas.

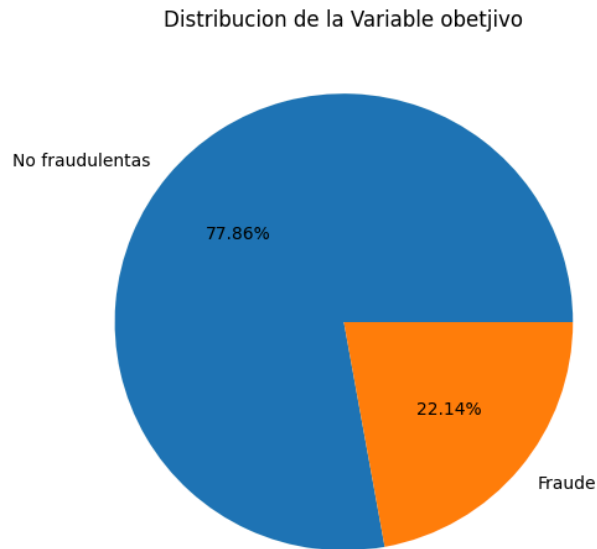
Descubrimiento de Características Redundantes:

- Al analizar la matriz de correlación, se encontraron varias características que estaban altamente correlacionadas entre sí. Por ejemplo:
 - total transactions (including txn to create contract) vs. total Ether sent contracts: Es probable que haya una alta correlación entre el número total de transacciones y la cantidad total de Ether enviado a contratos. Esto se debe a que, en muchas cuentas, el volumen de transacciones está directamente relacionado con la cantidad de Ether que manejan.
 - max val sent vs. avg val sent: Estas variables también podrían estar correlacionadas. Si bien max val sent mide la transacción de mayor valor, avg val sent captura una tendencia general que podría estar alineada con la transacción máxima.

Distribución de la Variable objetivo

La variable FLAG indica si una transacción es fraudulenta (1) o no fraudulenta (0). El análisis exploratorio mostró la siguiente distribución:

- Transacciones no fraudulentas (0): 7,662 casos (~78%).
- Transacciones fraudulentas (1): 2,179 casos (~22%).



Esta distribución revela un desbalance significativo en las clases, lo que es común en problemas de detección de fraudes, donde las instancias de fraude suelen ser mucho menos frecuentes que las no fraudulentas.

Implicaciones del Desbalance de Clases

1. Modelo Tendencioso:
 - Dado el desbalance de clases, un modelo entrenado sin abordar este problema podría volverse tendencioso hacia la clase mayoritaria (transacciones no fraudulentas). Por ejemplo, un modelo que simplemente clasifique todas las transacciones como no fraudulentas tendría una precisión alta (~78%), pero sería inútil para identificar fraudes, que es el objetivo principal.
2. Métricas de Evaluación Afectadas:
 - La precisión (accuracy), que es la métrica más simple y común, puede ser engañosa en un conjunto de datos desbalanceado. En este caso, métricas como el recall (tasa de verdaderos positivos) y el F1-score (media armónica de precisión y recall) se vuelven más importantes, ya que capturan mejor la capacidad del modelo para identificar correctamente las transacciones fraudulentas.
3. Riesgo de Errores Tipo II (Falsos Negativos):
 - En un escenario de detección de fraudes, los errores de tipo II, o falsos negativos (transacciones fraudulentas que el modelo clasifica incorrectamente como no

- ## Datos Faltantes

Correlación entre variables

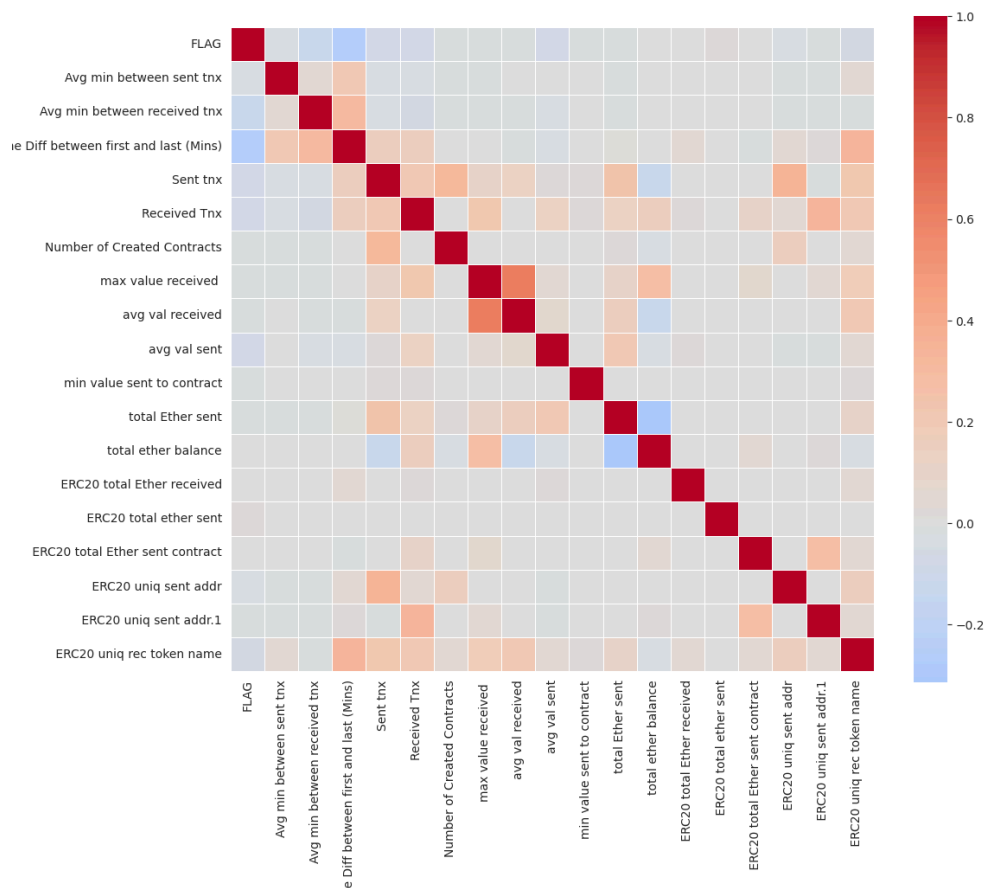
A partir de la matriz de correlación mostrada anteriormente, se identificaron pares de variables con una correlación significativa (alta correlación positiva o negativa). Estas variables fueron consideradas redundantes porque una puede estar proporcionando información similar a la otra.

Se decidió eliminar aquellas variables que estaban altamente correlacionadas con otras, para evitar problemas de multicolinealidad y simplificar el modelo. Las variables eliminadas fueron aquellas que aportan información redundante y cuya presencia podría dificultar el entrenamiento y la interpretación del modelo.

Las variables que se eliminaron fueron:

- 'FLAG',
- 'Avg min between sent tnx',
- 'Avg min between received tnx',
- 'Time Diff between first and last (Mins)',
- 'Sent tnx',
- 'Received Tnx',
- 'Number of Created Contracts', 'max value received',
- 'avg val received',
- 'avg val sent',
- 'min value sent to contract',
- 'total Ether sent', 'total ether balance',
- ' ERC20 total Ether received', ' ERC20 total ether sent',
- ' ERC20 total Ether sent contract', ' ERC20 uniq sent addr',
- ' ERC20 uniq sent addr.1', ' ERC20 uniq rec token name']

Obteniendo así la siguiente matriz de correlacion con las variables restantes:



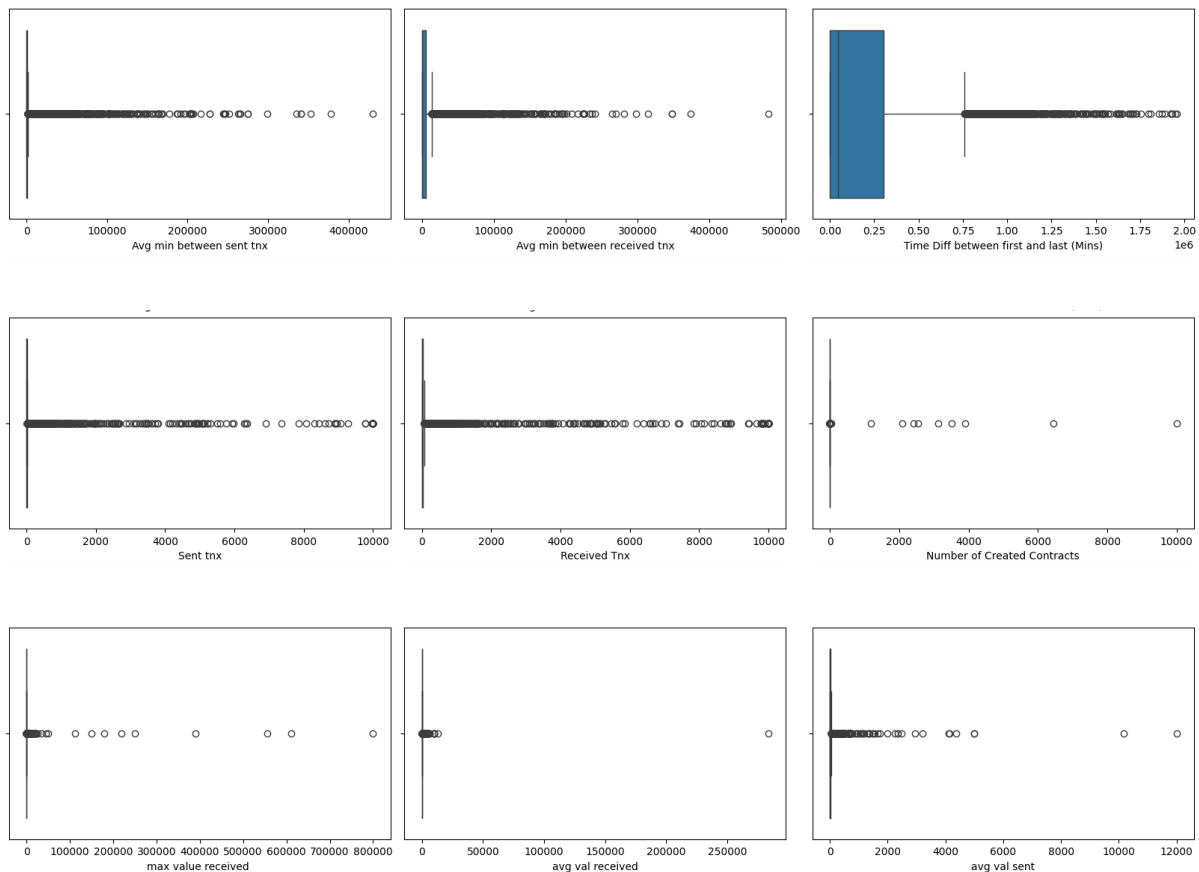
El proceso de identificar y eliminar variables correlacionadas es fundamental para construir modelos predictivos eficientes. La eliminación de variables redundantes:

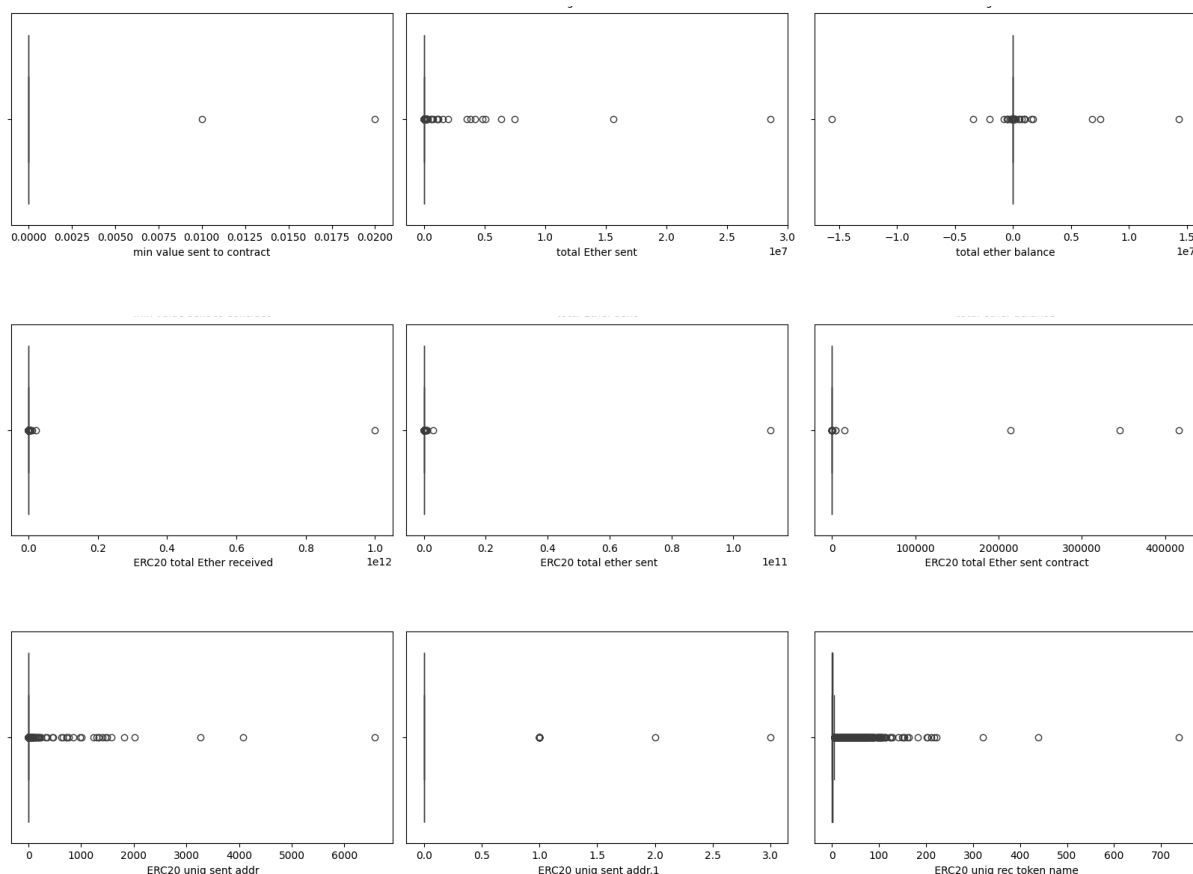
- Reduce la complejidad del modelo: Simplifica el modelo, lo que puede ayudar a mejorar su capacidad de generalización y reducir el riesgo de sobreajuste.
- Mejora la interpretabilidad: Un modelo con menos variables es más fácil de interpretar y de explicar a terceros, especialmente cuando se justifica la eliminación de variables redundantes.
- Optimiza el rendimiento: Al evitar la multicolinealidad, se mejora la estabilidad y el rendimiento del modelo, ya que las variables restantes aportan información única y relevante.

Distribución de Variables Restantes

El análisis se centró en examinar la distribución de las variables que quedaron después de la limpieza y eliminación de aquellas que estaban altamente correlacionadas o que no aportaban información relevante. Este análisis es crucial para entender cómo están distribuidas las variables numéricas, identificar posibles sesgos, y detectar la presencia de valores atípicos que podrían influir en el rendimiento de los modelos predictivos.

Se emplearon boxplots para cada variable para visualizar la distribución de cada una de las variables numéricas restantes. Los histogramas permiten ver la forma de la distribución, identificar si hay una inclinación hacia valores más bajos o más altos (sesgo), y determinar si la variable sigue una distribución normal, uniforme, o alguna otra forma. Los boxplots son gráficos útiles para visualizar la mediana, el rango intercuartílico (IQR), y los valores atípicos. Estos gráficos ayudarán a identificar si alguna de las variables tiene valores extremos que podrían necesitar un tratamiento adicional.





De este análisis de distribuciones, se llegó a la decisión de eliminar dos variables. Estas variables presentan distribuciones muy pequeñas:

Distribucion de min value sent to contract:

min value sent to contract

0.00 9839

0.02 1

0.01 1

Name: count, dtype: int64

Distribucion de ERC20 uniq sent addr.1:

ERC20 uniq sent addr.1

0.0 9813

1.0 26

3.0 1

2.0 1

Name: count, dtype: int64

Se puede observar que los valores de estas dos variables en su mayoría son 0, por lo que no proporcionan suficiente información para distinguir entre clases en el proceso de modelado. Esto significa que el modelo podría no aprender nada útil de esa variable, ya que su valor es prácticamente constante. Al eliminar variables que no aportan valor, se simplifica el modelo, lo que puede llevar a una mejor capacidad de generalización y a una reducción del riesgo de sobreajuste

Preparación de los datos

División del Conjunto de Datos en Conjuntos de Entrenamiento y Prueba:

- División (train_test_split): Se realizó una división del conjunto de datos en un conjunto de entrenamiento y un conjunto de prueba, utilizando una proporción del 80% para el entrenamiento y el 20% para la prueba.
- Se utilizó un random_state de 666 para asegurar que los resultados de la división sean reproducibles.
- Tamaños Resultantes:
 - Conjunto de entrenamiento (X_train y y_train): 7,872 instancias y 16 características predictoras.
 - Conjunto de prueba (X_test y y_test): 1,969 instancias y 16 características predictoras.

Normalización de los Datos

Después de la preparación inicial de los datos, se procedió con la normalización de las características. Este es un paso crucial porque muchas técnicas de aprendizaje automático, como la regresión logística o las redes neuronales, funcionan mejor cuando las características numéricas están en una escala similar.

Se utilizó el PowerTransformer de Scikit-learn para normalizar los datos. Esta técnica aplica una transformación que hace que las características numéricas se parezcan más a una distribución normal y estabiliza la varianza. Este enfoque es más robusto que simplemente estandarizar o normalizar porque maneja de manera más efectiva las características con distribuciones altamente sesgadas o con outliers.

Imbalance de Clases

Para abordar el desbalance de clases, se aplicaron varias técnicas durante el proceso de modelado:

Se aplicó SMOTE para generar ejemplos sintéticos de la clase minoritaria (transacciones fraudulentas), lo que permitió crear un conjunto de datos de entrenamiento más balanceado.

SMOTE es una técnica que crea nuevas instancias sintéticas para la clase minoritaria en lugar de simplemente duplicar instancias existentes. Esto ayuda a aumentar la cantidad de ejemplos de la clase minoritaria de manera que el modelo tenga suficientes datos para aprender patrones significativos que caracterizan las transacciones fraudulentas.

El desbalance de clases puede llevar a que el modelo ignore la clase minoritaria, en este caso, las transacciones fraudulentas, porque hay menos ejemplos para aprender. SMOTE ayuda a equilibrar las clases generando ejemplos sintéticos para la clase minoritaria, lo que permite que el modelo aprenda a reconocer patrones de fraude con mayor eficacia.

Al aplicar SMOTE, se creó un conjunto de entrenamiento balanceado que ayudó a mejorar la capacidad del modelo para detectar fraudes, reduciendo los falsos negativos sin aumentar significativamente los falsos positivos.

En lugar de confiar únicamente en la precisión, se utilizaron otras métricas como el recall y el F1-score. El recall es especialmente importante porque mide la proporción de transacciones fraudulentas que fueron correctamente identificadas por el modelo.

Las matrices de confusión se usaron para analizar cuántas transacciones fraudulentas fueron correctamente clasificadas frente a las incorrectamente clasificadas, proporcionando una visión clara de cómo el modelo estaba manejando el desbalance de clases.

Los modelos se ajustaron para priorizar la identificación correcta de fraudes. Esto se hizo ajustando parámetros de costo en algunos modelos, como en la Regresión Logística, o usando técnicas avanzadas de ajuste de hiper parámetros en modelos como XGBoost.

Modelos

En la sección dedicada a la construcción y evaluación de modelos, se probaron y compararon varios algoritmos de clasificación para detectar transacciones fraudulentas en Ethereum. Los tres modelos principales que se utilizaron fueron: Regresión Logística, Random Forest, y XGBoost. Se evaluó cada modelo por medio de sus métricas en comparación con el entrenamiento de train y su matriz de confusión, esta última es una herramienta esencial en la evaluación de modelos de clasificación, ya que proporciona una representación visual y numérica de las predicciones realizadas por el modelo frente a las verdaderas etiquetas de clase. En términos generales, una matriz de confusión ayuda a entender cuán bien está funcionando un modelo de clasificación y dónde se encuentran los errores específicos

Fine tuning o parameter tuning

En esta sección, (ajuste de parámetros), se enfocó en mejorar el rendimiento del modelo XGBoost utilizando la técnica de Grid Search CV. El modelo fue seleccionado para el ajuste de parámetros debido a su buen rendimiento inicial en la tarea de detección de fraudes. XGBoost es un modelo de ensamblado basado en árboles de decisión que es altamente eficiente y tiene la capacidad de manejar datos desbalanceados, lo que lo hace ideal para problemas como la detección de fraudes en transacciones.

Grid Search es una técnica de búsqueda sistemática que explora de manera exhaustiva todas las combinaciones posibles de un conjunto de hiperparámetros definidos por el usuario. La idea es identificar la combinación de hiperparámetros que maximiza el rendimiento del modelo en un conjunto de validación.

- Se definió un conjunto de valores posibles para cada parámetro de XGBoost que se quería optimizar, como el número de estimadores (`n_estimators`), la tasa de aprendizaje (`learning_rate`), la profundidad máxima de los árboles (`max_depth`), y otros parámetros importantes.
- Se seleccionó la combinación de parámetros que proporciona el mejor rendimiento según la métrica de evaluación (en este caso, `accuracy`).

Es notable mencionar que se utilizó la función `GridSearchCV` de Scikit-learn para ejecutar el Grid Search. Esta función entrena el modelo con todas las combinaciones posibles de parámetros y evalúa su rendimiento utilizando validación cruzada, generalmente con `k` pliegues (por ejemplo, 10-fold cross-validation).

Resultados

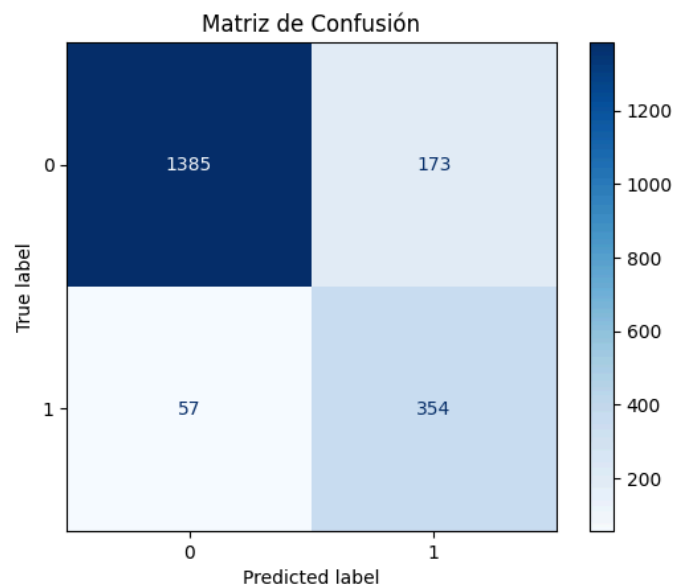
Para cada modelo se obtuvieron resultados diferentes, cada uno mejor que el anterior. Siendo estos de la siguiente manera:

1. Regresión Logística

a. Métricas

	precision	recall	f1-score	support
0	0.96	0.89	0.92	1558
1	0.67	0.86	0.75	411
accuracy			0.88	1969
macro avg	0.82	0.88	0.84	1969
weighted avg	0.90	0.88	0.89	1969

b. Matriz de confusión:

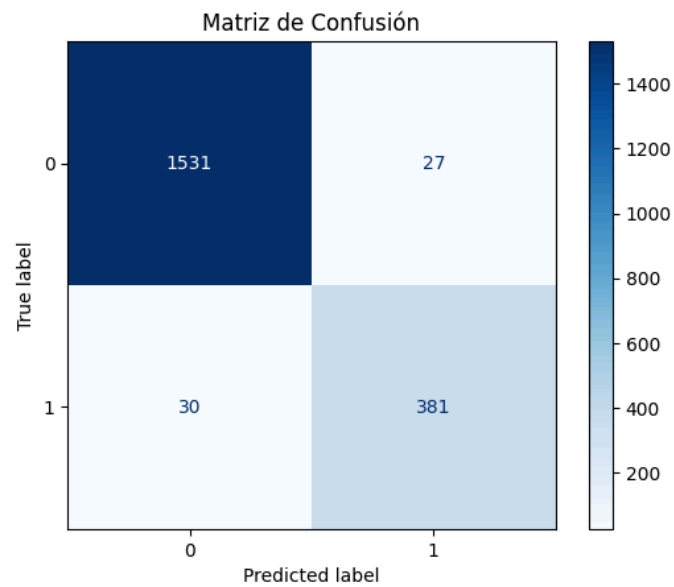


2. Random Forest

a. Métricas:

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1558
1	0.93	0.93	0.93	411
accuracy			0.97	1969
macro avg	0.96	0.95	0.96	1969
weighted avg	0.97	0.97	0.97	1969

b. Matriz de confusión:

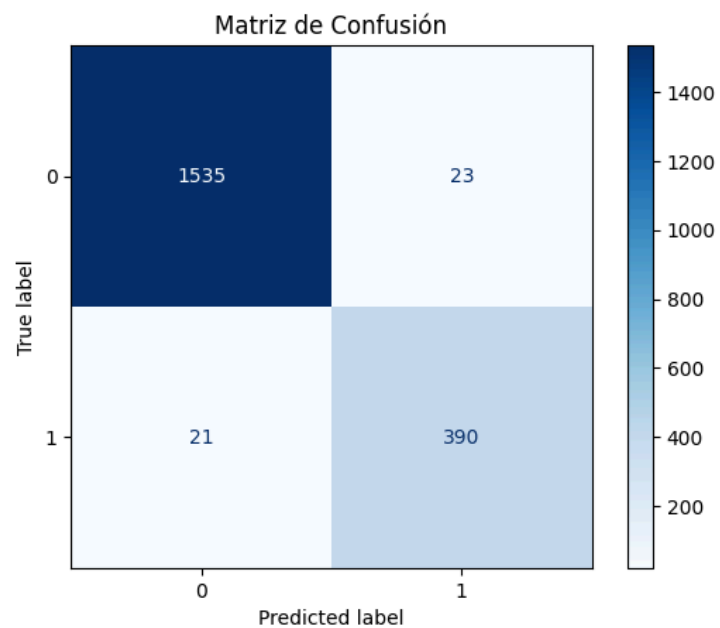


3. XGBoost

a. Métricas:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1558
1	0.94	0.95	0.95	411
accuracy			0.98	1969
macro avg	0.97	0.97	0.97	1969
weighted avg	0.98	0.98	0.98	1969

b. Matriz de confusión:

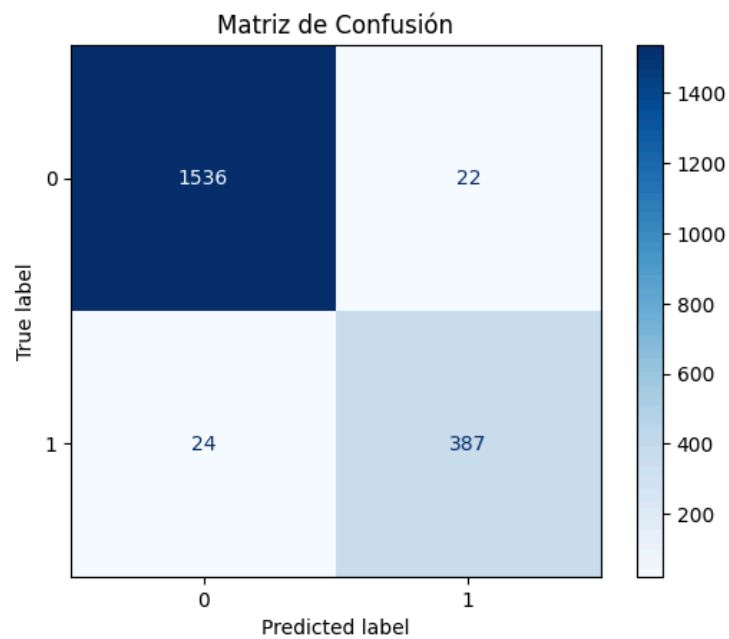


4. XGBoost (con GridSearchCV)

a. Métricas:

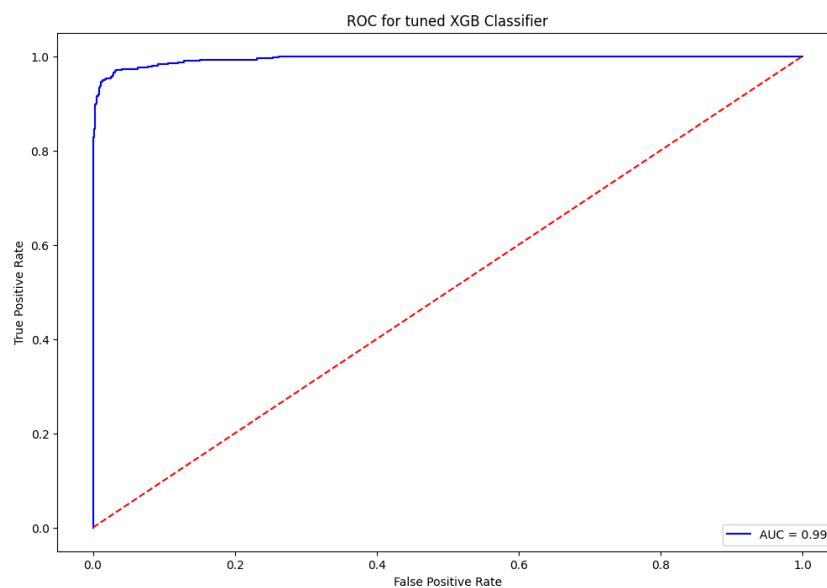
	precision	recall	f1-score	support
0	0.98	0.99	0.99	1558
1	0.95	0.94	0.94	411
accuracy			0.98	1969
macro avg	0.97	0.96	0.96	1969
weighted avg	0.98	0.98	0.98	1969

b. Matriz de confusión:



La matriz de confusión no muestra una mejora significativa; los resultados son muy similares a los obtenidos con el modelo sin ajustar los parámetros con Grid Search.

c. Curva AUC:



Descripción de algoritmos usados

Los modelos utilizados fueron:

1. Regresión Logística

La Regresión Logística es un modelo de clasificación lineal que se utiliza para estimar la probabilidad de que una instancia pertenezca a una clase determinada, en este caso, si una transacción es fraudulenta o no. Este modelo asume que las relaciones entre las características predictoras y la variable objetivo son lineales.

Ventajas:

- Simplicidad: Es fácil de implementar y entender.
- Interpretabilidad: Proporciona coeficientes que permiten interpretar la influencia de cada característica en la predicción.
- Rápido Entrenamiento: Debido a su simplicidad, entrena rápidamente incluso en conjuntos de datos grandes.

La Regresión Logística es un buen punto de partida en problemas de clasificación, especialmente cuando se busca un modelo fácil de interpretar que proporcione una línea base de rendimiento para compararlo con modelos más complejos.

Resultados Obtenidos:

- Rendimiento: El modelo alcanzó una accuracy de aproximadamente 88%, pero tuvo un recall más bajo para las transacciones fraudulentas, lo que indica que no detectó muchos fraudes.

2. Random Forest

Random Forest es un modelo de ensamblado basado en múltiples árboles de decisión. Cada árbol se entrena en un subconjunto aleatorio de los datos y de las características. Las predicciones se realizan tomando el promedio o la mayoría de votos de los árboles individuales.

Ventajas:

- Robustez: Es menos propenso al sobreajuste que un solo árbol de decisión debido a la combinación de múltiples árboles.
- Capacidad para Capturar Relaciones No Lineales: Puede manejar relaciones complejas entre las características y la variable objetivo.
- Manejo de Datos Desbalanceados: Random Forest es flexible y puede manejar conjuntos de datos donde las clases están desbalanceadas, como es común en la detección de fraudes.

Random Forest es adecuado cuando se busca un modelo robusto que pueda capturar patrones complejos en los datos y mejorar el rendimiento en problemas como la detección de fraudes, donde la relación entre las características y la variable objetivo puede no ser lineal.

Resultados Obtenidos:

- Rendimiento: El modelo mejoró la precisión general a aproximadamente 96%, con un recall notablemente mejorado en la detección de transacciones fraudulentas.

3. XGBoost

XGBoost (Extreme Gradient Boosting) es un modelo de ensamblado que optimiza iterativamente un conjunto de árboles de decisión, mejorando continuamente el rendimiento al reducir los errores de las predicciones anteriores. Es conocido por su alta eficiencia y precisión en competiciones de machine learning.

Ventajas:

- Eficiencia: XGBoost es extremadamente rápido y eficiente en términos de computación.
- Capacidad para manejar datos desbalanceados: Incluye técnicas integradas para manejar datos desbalanceados, lo que es crucial en la detección de fraudes.
- Mejoras Iterativas: Cada iteración del modelo busca corregir los errores de las iteraciones anteriores, lo que resulta en un modelo muy preciso.

XGBoost es especialmente útil en problemas de clasificación desafiantes como la detección de fraudes, donde la precisión y la capacidad de manejar un conjunto de datos desbalanceado son esenciales.

Resultados Obtenidos:

- Rendimiento: XGBoost alcanzó una accuracy de aproximadamente 98%, con un recall excepcionalmente alto, lo que significa que detectó la mayoría de las transacciones fraudulentas con pocos falsos negativos.

Cada uno de estos modelos tiene sus ventajas, y la elección de XGBoost como el modelo final se basó en su capacidad superior para manejar la complejidad y el desbalance de los datos en la tarea de detección de fraudes en la red de Ethereum.

Ajustes y explicación de los parámetros

Para lograr tener la mejor combinación de parámetros para el mejor modelo (XGBoost) se utilizó la técnica de GridSearchCV, la cual es una técnica de búsqueda sistemática que explora de manera exhaustiva todas las combinaciones posibles de un conjunto de hiperparámetros definidos por el usuario. La idea es identificar la combinación de parámetros que maximiza el rendimiento del modelo en un conjunto de validación. Y se tuvo en cuenta la validación cruzada dentro de esta función. De

forma que se entrenó el modelo con todas las combinaciones posibles de parámetros y evalúa su rendimiento utilizando validación cruzada, con 10 k-folds.

Parameter tuning

Grid search

```
params_grid = {'learning_rate':[0.01, 0.1, 0.5],
               'n_estimators':[100,200],
               'subsample':[0.3, 0.5, 0.9],
               'max_depth':[2,3,4],
               'colsample_bytree':[0.3,0.5,0.7]}

grid = GridSearchCV(estimator=xgb_c, param_grid=params_grid, scoring='recall', cv = 10, verbose = 0)

grid.fit(x_tr_resample, y_tr_resample)
print(f'Best params found for XGBoost are: {grid.best_params_}')
print(f'Best recall obtained by the best params: {grid.best_score_}')
```

Best params found for XGBoost are: {'colsample_bytree': 0.5, 'learning_rate': 0.5, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.5}
Best recall obtained by the best params: 0.9900093906790802

Discusión

Los resultados obtenidos en la comparación de los modelos de Regresión Logística, Random Forest, y XGBoost reflejan la naturaleza intrínseca de cada uno de estos algoritmos y su capacidad para manejar la complejidad y el desbalance de los datos en el contexto de la detección de fraudes en la red de Ethereum.

La Regresión Logística fue el modelo más simple y básico utilizado en este análisis. Como un modelo de clasificación lineal, la Regresión Logística asume que existe una relación lineal entre las características predictoras y la variable objetivo. Esta suposición limita la capacidad del modelo para capturar patrones complejos y no lineales que son comunes en los datos financieros y transaccionales.

Aunque el modelo logró una accuracy general relativamente alta, alrededor del 88%, su desempeño en la detección de transacciones fraudulentas fue insuficiente. Esto se debió a su incapacidad para distinguir adecuadamente entre las transacciones fraudulentas y no fraudulentas en un entorno donde los patrones de fraude no son lineales y pueden involucrar interacciones complejas entre múltiples variables. El resultado fue un recall bajo para la clase minoritaria (transacciones fraudulentas), lo que significa que el modelo dejó de identificar una cantidad significativa de fraudes, reflejando una alta tasa de falsos negativos. Este resultado era previsible debido a la naturaleza del modelo y la complejidad del problema que se estaba abordando.

Por otro lado, Random Forest introdujo un enfoque más robusto al combinar múltiples árboles de decisión en un modelo de ensamblado. Esta técnica le permite manejar mejor la complejidad de los datos y capturar relaciones no lineales entre las variables. Random Forest mejoró notablemente el recall en la detección de fraudes en comparación con la Regresión Logística, reduciendo significativamente la cantidad de falsos negativos. Esto es posible porque cada árbol en el bosque se entrena en un subconjunto diferente de los datos, lo que permite al modelo aprender una amplia gama de patrones y ser más resiliente frente a las anomalías que pueden estar presentes en los datos. Además, el uso del promedio de las predicciones de los árboles ayuda a mitigar el sobreajuste, un problema común en los árboles de decisión individuales. Sin embargo, a pesar de estas mejoras, el modelo todavía no alcanzó el máximo rendimiento posible, ya que la complejidad adicional introducida por la combinación de múltiples árboles puede hacer que el modelo sea más difícil de ajustar de manera óptima, lo que podría limitar su capacidad para captar todos los patrones necesarios para una detección de fraudes perfecta.

Finalmente, XGBoost sobresalió como el modelo más efectivo entre los tres. XGBoost, al ser una implementación de gradient boosting, se beneficia de la capacidad de corregir errores en cada iteración del modelo, lo que le permite mejorar continuamente su rendimiento. Una de las principales ventajas de XGBoost es su capacidad para manejar datos desbalanceados, como es común en problemas de detección de fraudes, donde las instancias de fraude son mucho menos frecuentes que las no fraudulentas. El modelo XGBoost logró una precisión general de alrededor del 98%, y lo que es más importante, un recall excepcionalmente alto, lo que indica que fue capaz de identificar la mayoría de las transacciones fraudulentas con un mínimo de falsos negativos. Esto es crítico en la detección de fraudes, ya que cada transacción fraudulenta no detectada puede tener consecuencias graves. La eficacia de XGBoost se deriva de su capacidad para ajustar finamente los hiperparámetros y optimizar cada iteración del modelo, lo que le permite capturar patrones complejos en los datos que los otros modelos no pudieron identificar.

En comparación, mientras que la Regresión Logística demostró ser un buen punto de partida, su simplicidad resultó ser una desventaja en este contexto específico, donde se requiere un modelo más sofisticado para manejar la complejidad de los datos. Random Forest ofreció una mejora significativa al aprovechar el poder de múltiples árboles de decisión, pero aún no alcanzó el nivel de precisión necesario para la detección de fraudes con alta confiabilidad. XGBoost, en cambio, demostró ser el más capaz de los tres, logrando un equilibrio óptimo entre precisión y recall, y superando los desafíos presentados por la naturaleza desbalanceada y compleja de los datos.

La matriz de confusión es una herramienta esencial para evaluar el rendimiento de un modelo de clasificación, ya que muestra no sólo la precisión global del modelo, sino también cómo se desempeña en la clasificación de cada clase (en este caso, transacciones fraudulentas y no fraudulentas). Al comparar las matrices de confusión del modelo XGBoost sin ajuste de hiperparámetros (sin GridSearchCV) y con ajuste de hiperparámetros (con GridSearchCV), se observa que los resultados son muy similares, sin una mejora significativa en la reducción de falsos negativos.

En este caso, el modelo sin GridSearchCV ya estaba bien configurado con sus valores predeterminados, lo que significa que ya estaba aprovechando al máximo la estructura y la información en los datos. Esto sugiere que los valores por defecto del modelo estaban cerca de ser óptimos para este conjunto de datos específico, lo que explica por qué el ajuste de parámetros no condujo a mejoras drásticas.

Dado que el modelo sin GridSearchCV ya ofrecía un rendimiento excelente, el margen de mejora era limitado. Cuando un modelo ya está cerca de su capacidad máxima de rendimiento, cualquier ajuste adicional en los hiperparámetros puede ofrecer mejoras incrementales en lugar de cambios sustanciales.

Analizando la curva AUC, aunque el AUC es alto, lo que indica un buen rendimiento general del modelo, es importante considerar que la implementación práctica también debe tener en cuenta el umbral de decisión específico que se elija. Un umbral que maximiza el recall puede llevar a un aumento en los falsos positivos, mientras que uno que minimiza los falsos positivos puede reducir el recall. Por lo tanto, el AUC proporciona una visión global, pero la selección del umbral adecuado dependerá del contexto específico y de los costos asociados con falsos positivos y falsos negativos.

En resumen, los resultados obtenidos reflejan las fortalezas y limitaciones inherentes de cada modelo. XGBoost fue el mejor en este estudio debido a su capacidad para adaptarse a la complejidad del problema y optimizar su rendimiento mediante técnicas avanzadas de boosting, lo que lo convierte en la opción más adecuada para la detección de fraudes en la red de Ethereum.

Conclusiones y trabajo futuro

- XGBoost demostró ser el modelo más eficaz en la detección de transacciones fraudulentas en Ethereum, superando tanto a la Regresión Logística como a Random Forest en precisión y recall, lo que lo convierte en la opción ideal para este tipo de problemas complejos.
- Aunque GridSearch permitió una optimización fina de los hiperparámetros de XGBoost, las mejoras en las métricas de rendimiento fueron mínimas, sugiriendo que el modelo ya estaba bien ajustado con sus valores predeterminados.
- La decisión de imputar valores faltantes utilizando la mediana fue crucial para mantener la robustez del modelo, ya que evitó sesgos que podrían haber sido introducidos por outliers en el conjunto de datos.
- Aunque Random Forest no alcanzó el rendimiento de XGBoost, ofreció una mejora significativa sobre la Regresión Logística, capturando patrones más complejos gracias a su capacidad para manejar relaciones no lineales.
- La Regresión Logística, aunque fácil de interpretar y rápida de entrenar, mostró limitaciones claras en la detección de fraudes debido a su incapacidad para manejar relaciones no lineales en los datos.
- El alto valor del AUC para XGBoost indicó un excelente rendimiento global del modelo, confirmando su capacidad para discriminar correctamente entre transacciones fraudulentas y no fraudulentas en diferentes umbrales.
- El uso de SMOTE para balancear las clases fue clave para mejorar el recall del modelo, destacando la importancia de abordar el desbalance de clases en problemas de detección de fraudes para evitar la subrepresentación de la clase minoritaria.

Algún posible trabajo futuro el cual sería muy interesante y relevante es el explorar el uso de modelos de Deep Learning, como redes neuronales recurrentes (RNN) o redes neuronales convolucionales (CNN), con mayor cantidad de datos y en tiempo real. Esto podría mejorar aún más la capacidad de detectar patrones complejos en las transacciones. Estas técnicas podrían ser especialmente útiles para analizar secuencias de transacciones y detectar comportamientos anómalos en serie. Más allá de simplemente clasificar transacciones como fraudulentas o no fraudulentas, un enfoque innovador sería el desarrollo de un sistema de puntuación de riesgo (Risk Scoring) para cada transacción o cuenta. Este sistema podría asignar una puntuación basada en la probabilidad de fraude, permitiendo a los usuarios y entidades financieras tomar decisiones informadas sobre las transacciones con un riesgo elevado.

Glosario

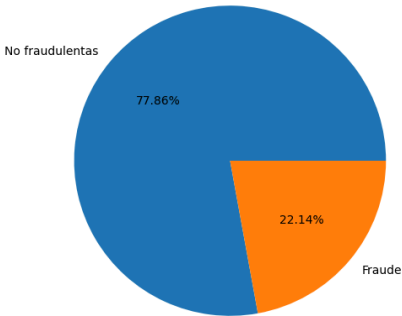
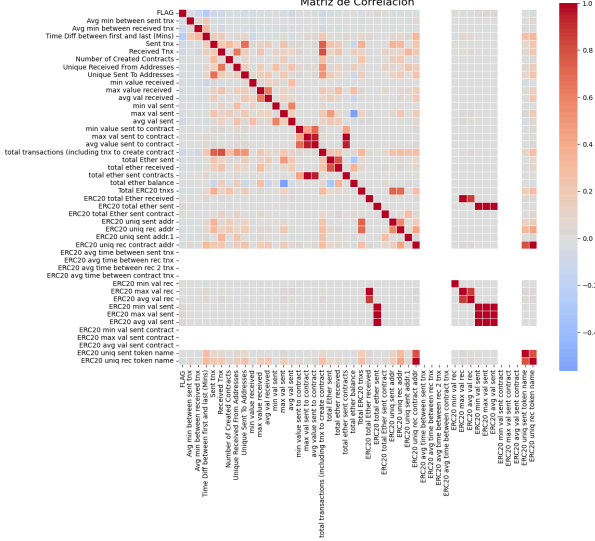
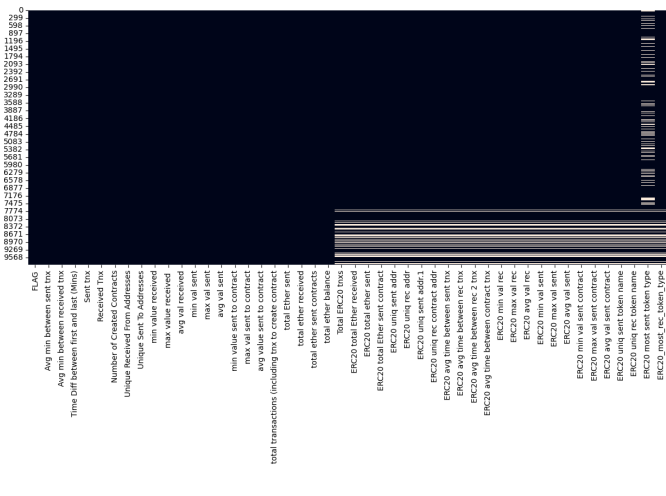
- *Ethereum*: Plataforma de blockchain descentralizada que permite la creación de contratos inteligentes y la ejecución de aplicaciones descentralizadas.
- *Transacción Fraudulenta*: Movimiento de fondos en la red Ethereum que se realiza con la intención de defraudar a una de las partes.
- *Machine Learning*: Campo de la inteligencia artificial que utiliza algoritmos para aprender patrones a partir de datos y hacer predicciones o tomar decisiones.
- *Regresión Logística*: Modelo de clasificación lineal utilizado para predecir la probabilidad de una clase binaria.
- *Random Forest*: Modelo de ensamblado que utiliza múltiples árboles de decisión para mejorar la precisión y robustez del modelo.
- *XGBoost*: Algoritmo de boosting basado en árboles de decisión que optimiza el rendimiento mediante la corrección iterativa de errores.
- *GridSearch*: Técnica para ajustar hiperparámetros mediante la exploración sistemática de todas las combinaciones posibles para encontrar la mejor configuración.
- *Recall*: Métrica de evaluación que mide la capacidad del modelo para identificar correctamente las instancias positivas.
- *Precisión (Accuracy)*: Proporción de predicciones correctas realizadas por el modelo en el conjunto de datos.
- *Matriz de Confusión*: Herramienta para evaluar el rendimiento de un modelo de clasificación mostrando las predicciones correctas e incorrectas en cada clase.
- *SMOTE*: Técnica de oversampling que genera ejemplos sintéticos de la clase minoritaria para equilibrar conjuntos de datos desbalanceados.
- *Normalización*: Proceso de escalar las características numéricas a un rango común, lo que mejora la eficacia de muchos modelos de machine learning.
- *Imputación*: Proceso de reemplazar valores faltantes en un conjunto de datos con estimaciones, como la media o la mediana.
- *Overfitting (Sobreajuste)*: Problema en el que un modelo se ajusta demasiado bien a los datos de entrenamiento, pero falla en generalizar a nuevos datos.
- *AUC-ROC*: Métrica que mide el área bajo la curva ROC, utilizada para evaluar la capacidad de un modelo de distinguir entre clases.
- *Clase Minoritaria*: En un conjunto de datos desbalanceado, es la clase que tiene menos ejemplos comparada con la clase mayoritaria.
- *Validación Cruzada*: Técnica para evaluar el rendimiento de un modelo dividiendo los datos en varios subconjuntos y entrenando el modelo en cada subconjunto.
- *Deep Learning*: Subcampo del machine learning que utiliza redes neuronales profundas para modelar patrones complejos en grandes volúmenes de datos.
- *Clustering*: Técnica de machine learning no supervisado que agrupa instancias similares en conjuntos, sin utilizar etiquetas predefinidas.
- *Análisis Anómalo*: Técnica utilizada para identificar instancias que no siguen el patrón general del conjunto de datos, comúnmente empleada para detectar fraudes.
- *Línea Base (Baseline)*: Modelo simple que sirve como punto de referencia para comparar el rendimiento de modelos más complejos.
- *Risk Scoring*: Sistema que asigna una puntuación basada en el riesgo a cada transacción o cuenta, permitiendo evaluar su probabilidad de ser fraudulenta.

Bibliografía

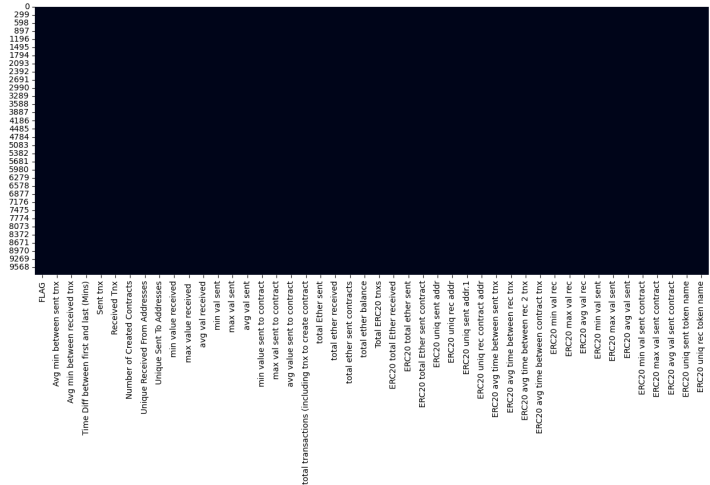
1. Chen, Y., Li, C., Wang, Y., & Zhang, L. (2020). "Deep learning-based fraud detection in financial transactions." IEEE Access, 8, 205667-205676.
2. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., & Mendling, J. (2019). "Detecting illicit accounts in permissionless blockchains using temporal graph analysis." Business Process Management Journal, 25(5), 1049-1076.
3. Farrell, R., & Clough, P. (2018). "Using autoencoders to detect fraudulent transactions." Journal of Data Mining and Digital Humanities.
4. Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Bitcoing.org](https://bitcoin.org)
5. Mearian, L. (2018). "Blockchain: The Complete Guide." Computerworld. Retrieved from [computerworld.com](<https://www.computerworld.com/article/3244948/blockchain-the-complete-guide.html>).
6. Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). "SMOTE: Synthetic Minority Over-sampling Technique." Journal of Artificial Intelligence Research, 16, 321-357.
7. Brownlee, J. (2020). "How to Use SMOTE for Imbalanced Classification." Machine Learning Mastery. Retrieved from [machinelearningmastery.com](<https://machinelearningmastery.com/smote-for-imbalanced-classification/>).
8. Scikit-learn developers. (2023). "PowerTransformer: Normalize and Stabilize Data." Scikit-learn Documentation. Retrieved from [scikit-learn.org](<https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html>).
9. Antonopoulos, A. M., & Wood, G. (2018). "Mastering Ethereum: Building Smart Contracts and DApps". O'Reilly Media.
10. Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.
11. Chen, Y., Li, Y., Wang, L., & Zhang, L. (2018). "Detecting Ethereum Fraudulent Transactions with Machine Learning." Proceedings of the 2018 IEEE International Conference on Big Data (Big Data).
12. Feng, J., Liu, Q., & Zhou, H. (2020). "Comparative Study of Machine Learning Algorithms for Detecting Ethereum Fraud." Journal of Financial Risk Management, 9(3), 189-199.
13. Victor, F., Weiler, N., & Schär, F. (2019). "Fraud Detection in Ethereum Blockchain Using Convolutional Neural Networks." *IEEE Access*, 7, 110151-110162.

14. Vasek, M., & Moore, T. (2015). *"There's No Free Lunch, Even Using Bitcoin: Tracking the Popularity and Profits of Virtual Currency Scams."* Financial Cryptography and Data Security, 2015, 44-61.
15. Chen, T., & Guestrin, C. (2016). *"XGBoost: A Scalable Tree Boosting System."* Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). ACM.
16. Pedregosa, F., Varoquaux, G., Gramfort, A., et al. (2011). *"Scikit-learn: Machine Learning in Python."* Journal of Machine Learning Research, 12, 2825-2830.
17. Andreas, A. (2018). *Mastering Ethereum: Building Smart Contracts and DApps*. O'Reilly Media.
18. Sujata, S., & Sahay, S. K. (2021). *"Blockchain and Machine Learning for Fraud Detection: A Case Study on Financial Transactions."* International Journal of Advanced Computer Science and Applications, 12(3), 456-462.

Anexos

No.	Nombre	Figura
1	Distribución de la variable objetivo.	<div><div>Distribucion de la Variable ojetjivo</div></div>
2	Matriz de Correlación de todas las variables iniciales.	<div><div>Matriz de Correlación</div></div>
3	Mapa de Calor de Valores Faltantes de todas las variables.	<div></div>

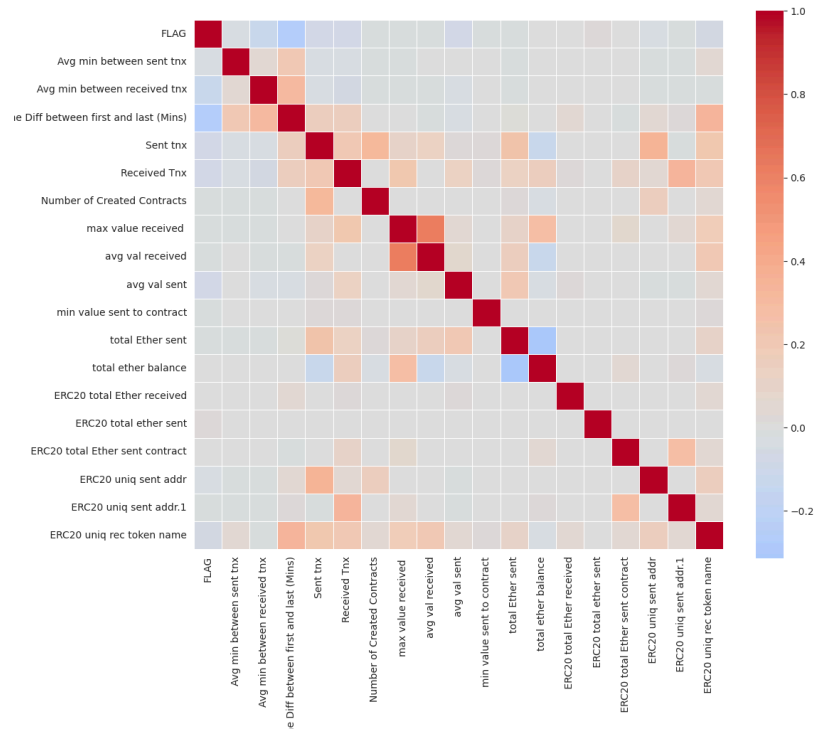
- 4 Mapa de Calor de Valores Faltantes de todas las variables sin variables con distribuciones pequeñas.



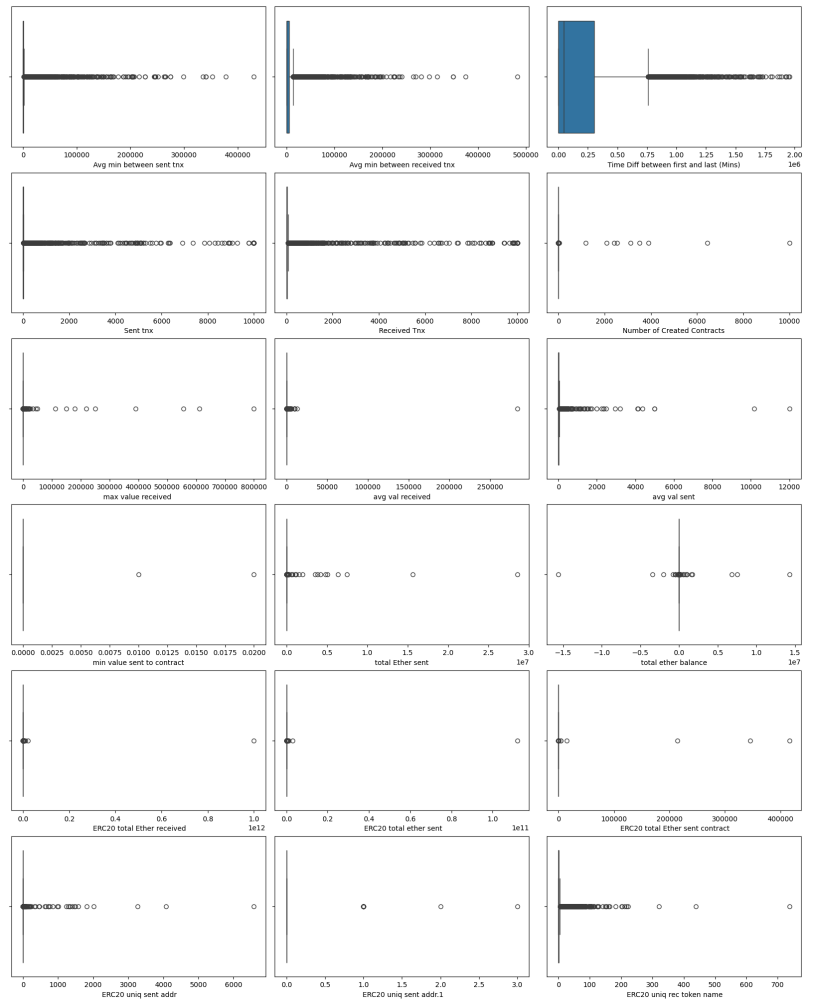
- 5 Matriz de correlación sin variables con varianza igual a cero.



6 Matriz de correlación sin las variables relacionadas entre sí (no multicolinealidad)



7 Distribuciones de las variables seleccionadas (no varianza igual 0 ni correlacionadas entre sí, ni faltantes)



- 8 Distribuciones de las únicas 2 variables a eliminar por no tener valores.

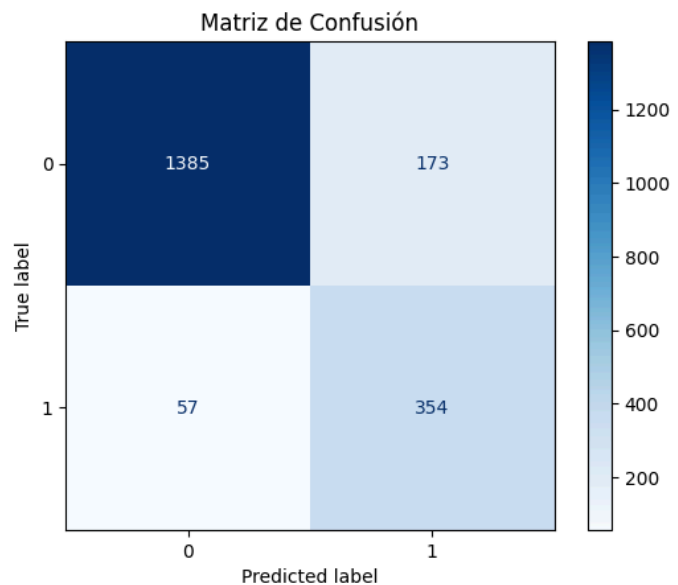
```
Distribucion de min value sent to contract:
min value sent to contract
0.00    9839
0.02      1
0.01      1
Name: count, dtype: int64
```

```
Distribucion de ERC20 uniq sent addr.1:
ERC20 uniq sent addr.1
0.0    9813
1.0     26
3.0      1
2.0      1
Name: count, dtype: int64
```

- 9 Métricas para el Modelo de Regresión Logística.

	precision	recall	f1-score	support
0	0.96	0.89	0.92	1558
1	0.67	0.86	0.75	411
accuracy			0.88	1969
macro avg	0.82	0.88	0.84	1969
weighted avg	0.90	0.88	0.89	1969

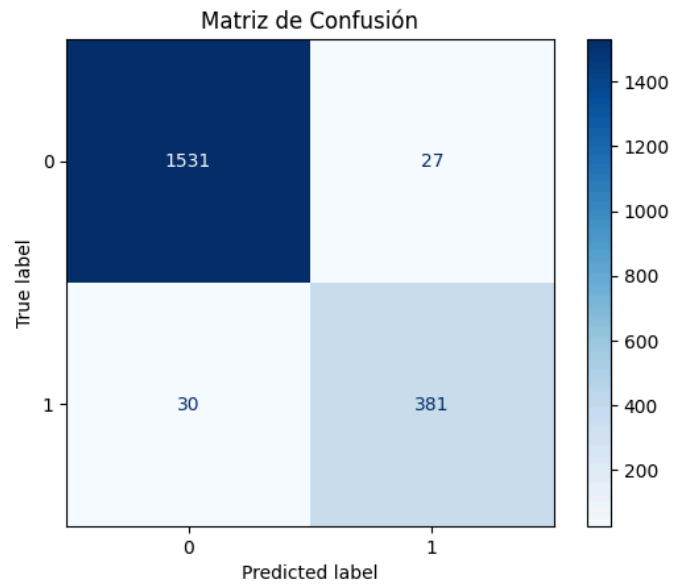
- 10 Matriz de Confusión para el modelo de Regresión Logística.



- 11 Métricas para el Modelo de Random Forest.

	precision	recall	f1-score	support
0	0.98	0.98	0.98	1558
1	0.93	0.93	0.93	411
accuracy			0.97	1969
macro avg	0.96	0.95	0.96	1969
weighted avg	0.97	0.97	0.97	1969

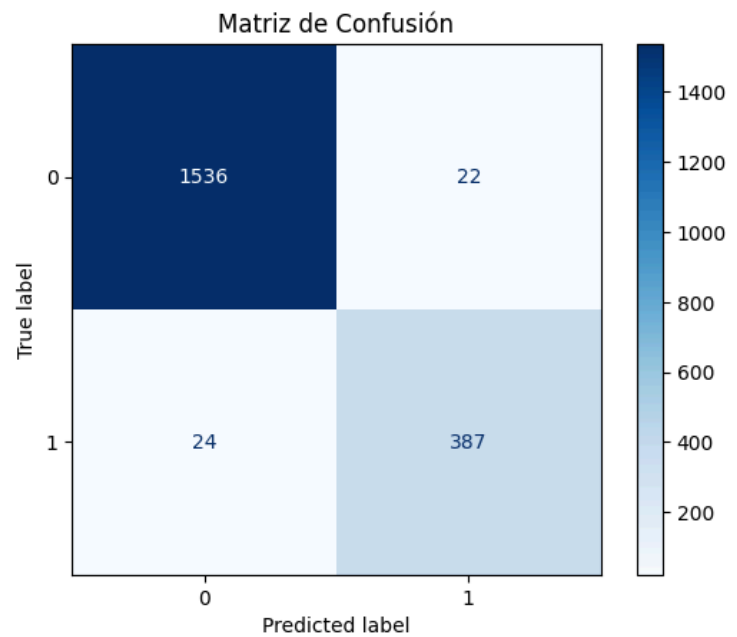
- 12 Matriz de Confusión para el modelo de Random forest



- 13 Métricas para el Modelo de XGBoost

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1558
1	0.95	0.94	0.94	411
accuracy			0.98	1969
macro avg	0.97	0.96	0.96	1969
weighted avg	0.98	0.98	0.98	1969

- 14 Matriz de Confusión para el modelo de XGBoost



15 Resultados de GridSearchCV.

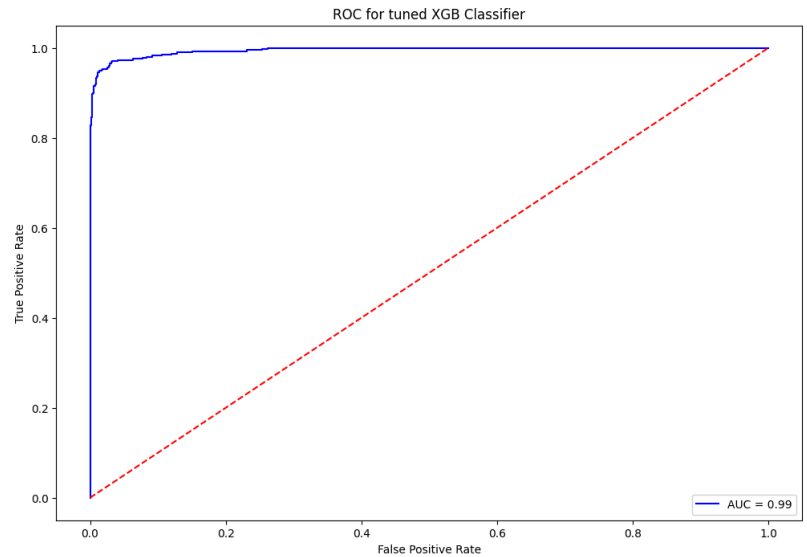
Parameter tuning

Grid search

```
params_grid = {'learning_rate':[0.01, 0.1, 0.5],  
              'n_estimators':[100,200],  
              'subsample':[0.3, 0.5, 0.9],  
              'max_depth':[2,3,4],  
              'colsample_bytree':[0.3,0.5,0.7]}  
  
grid = GridSearchCV(estimator=xgb_c, param_grid=params_grid, scoring='recall', cv = 10, verbose = 0)  
  
grid.fit(x_tr_resample, y_tr_resample)  
print(f'Best params found for XGBoost are: {grid.best_params_}')  
print(f'Best recall obtained by the best params: {grid.best_score_}')
```

Best params found for XGBoost are: {'colsample_bytree': 0.5, 'learning_rate': 0.5, 'max_depth': 4, 'n_estimators': 200, 'subsample': 0.5}
Best recall obtained by the best params: 0.9900093906790802

16 Curva AUC-ROC para XGBoost con GridSearchCV.



17 Código

https://github.com/sergiomarchena16/TFM_2024/blob/main/final.ipynb