Escuela de Negocios
Tipo de documento: Tesis de maestría

Master in Management + Analytics

Predicción de Contratos Inteligentes Maliciosos en Blockchains EVM mediante Análisis de Opcodes y Machine Learning

Autoría: Landaburu, Rodrigo
Año: 2025

UNIVERSIDAD
TORCUATO DI TELLA

Master in Management + Analytics

Predicción de Contratos Inteligentes Maliciosos en Blockchains EVM mediante Análisis de Opcodes y Machine Learning

---

Autor: Rodrigo Landaburu
Tutor: Carlos Salort
Junio 2025

Predicción de Contratos Inteligentes Maliciosos en Blockchains EVM mediante Análisis de Opcodes y Machine Learning

Tesis de Maestría en Gestión y Análisis de datos

Rodrigo Landaburu

Resumen

La tecnología *blockchain* ha transformado significativamente la forma en que se intercambia información, ofreciendo transacciones seguras y transparentes. Sin embargo, esta misma percepción de seguridad ha facilitado el surgimiento de contratos inteligentes maliciosos en redes basadas en *Ethereum Virtual Machine.* La falta de verificación del código fuente en muchos contratos expone únicamente sus *opcodes* o *bytecodes* públicamente, dificultando la detección temprana de comportamientos fraudulentos.

Esta tesis propone una solución mediante la aplicación de técnicas de *Machine Learning* para predecir la maliciosidad de contratos inteligentes analizando sus *opcodes*. Utilizando un *dataset* compuesto por más de cien mil contratos inteligentes etiquetados como maliciosos y no maliciosos, se desarrollan modelos predictivos que permiten clasificar contratos en tiempo real, antes de que los usuarios interactúen con ellos.

El proceso metodológico incluye la transformación de *opcodes* en vectores numéricos mediante técnicas como TF-IDF, la selección y configuración de modelos específicos para datos desbalanceados (*Easy Ensemble Classifier* y *Balanced Random Forest Classifier*), y la optimización del desempeño mediante validación cruzada. Los resultados obtenidos se evalúan con métricas como Precision, *Recall* y *F1-score*, garantizando una alta confiabilidad en las predicciones.

Finalmente, se propone una implementación práctica del modelo a través de una API, facilitando que plataformas *blockchain*, billeteras virtuales y *exchanges* puedan consultar rápidamente el riesgo asociado a un contrato inteligente, contribuyendo así a la seguridad del ecosistema *blockchain*.

Malicious Smart Contracts prediction in EVM Blockchains through Opcode Analysis and Machine Learning

Master's Thesis in Management and Data Analytics

Rodrigo Landaburu

Abstract

Blockchain technology has significantly transformed the way information is exchanged, offering secure and transparent transactions. However, this very perception of security has facilitated the proliferation of malicious smart contracts on networks based on the Ethereum Virtual Machine (EVM). The lack of source code verification in many contracts exposes only their opcodes or bytecodes publicly, making it difficult to detect fraudulent behavior early.

This thesis proposes a solution through the application of Machine Learning techniques to predict the maliciousness of smart contracts by analyzing their opcodes. Using a dataset composed of more than hundred thousands smart contracts labeled as malicious and non-malicious, predictive models are developed to classify contracts in real time before users interact with them.

The methodological process includes transforming opcodes into numerical vectors using techniques such as TF-IDF, selecting and configuring specific models for imbalanced data (Easy Ensemble Classifier and Balanced Random Forest Classifier), and optimizing performance through cross-validation. The results obtained are evaluated with metrics such as Precision, Recall, and F1-score, ensuring high reliability in predictions.

Finally, a practical implementation of the model is proposed through an API, enabling blockchain platforms, virtual wallets, and exchanges to quickly assess the risk associated with a smart contract, thereby contributing to the security of the blockchain ecosystem.

Índice de Tablas

Índice de Figuras

1. Introducción

1.1 Contexto y motivación

En los últimos años, la adopción de la tecnología *blockchain* creció exponencialmente, impulsada principalmente por la necesidad de contar con sistemas descentralizados, transparentes y seguros. Esta tecnología, originalmente utilizada para criptomonedas como Bitcoin, se extendió rápidamente hacia otras áreas, destacándose Ethereum como una plataforma clave para la ejecución de contratos inteligentes (*smart contracts*). Estos contratos son programas autoejecutables que se activan automáticamente cuando se cumplen ciertas condiciones predefinidas, permitiendo eliminar intermediarios, reducir costos y agilizar procesos.

Sin embargo, como ocurre generalmente con cualquier avance tecnológico significativo, esta expansión también generó nuevos desafíos y vulnerabilidades. La facilidad que ofrece Ethereum para desplegar contratos inteligentes sin restricciones significativas ni controles previos generó un entorno favorable para actores malintencionados. En consecuencia, comenzaron a aparecer contratos inteligentes fraudulentos diseñados específicamente para estafar usuarios, robar fondos o comprometer información sensible. Estos contratos maliciosos no solo causan pérdidas económicas importantes, sino que también afectan negativamente la confianza de los usuarios en la tecnología *blockchain* en general.

En este contexto, la motivación principal de esta tesis se basa en la necesidad urgente de contar con mecanismos efectivos que permitan detectar de manera temprana y precisa este tipo de contratos fraudulentos. Específicamente, se busca desarrollar una herramienta capaz de analizar automáticamente contratos inteligentes antes de que los usuarios interactúen con ellos, utilizando técnicas avanzadas de *Machine Learning* aplicadas directamente sobre los *opcodes*, que son la representación compilada y pública de estos contratos. Con una herramienta de estas características, los usuarios y empresas podrán tener mayor información para la toma de decisiones a la hora de interactuar con un contrato inteligente y así evitar ser engañados y perder todos sus fondos, ya que en el mundo de *blockchain*, una sola firma en un contrato malicioso puede terminar con la pérdida de todos los activos.

Es muy difícil imaginar que nuevos usuarios puedan sumarse al ecosistema si el riesgo de perder todos los activos es tan alto. Es por esta razón que no solo es necesario para evitar futuras pérdidas financieras sino que es fundamental para que el ecosistema pueda seguir creciendo.

1.2 Problema de investigación

Actualmente, uno de los principales desafíos para detectar contratos inteligentes maliciosos en redes *blockchain* basadas en *Ethereum Virtual Machine* (EVM) es la falta de disponibilidad pública del código fuente original de la mayoría de los contratos desplegados. En general, los desarrolladores no suelen compartir el código fuente completo del contrato inteligente, dejando únicamente disponible su versión compilada en forma de *bytecode* u *opcodes*. Esta representación compilada es considerablemente más difícil de interpretar y analizar que el código fuente original, limitando significativamente la eficacia de métodos tradicionales de auditoría manual o análisis estático.

Además, el volumen creciente y continuo de contratos inteligentes desplegados diariamente hace imposible realizar auditorías manuales exhaustivas en tiempo real. Esto implica que los usuarios suelen interactuar con contratos sin contar con información clara sobre su legitimidad o potencial riesgo, lo que incrementa notablemente la posibilidad de caer en estafas o ataques.

Frente a este problema, surge la siguiente pregunta central que buscamos responder: ¿es posible desarrollar un modelo predictivo basado en técnicas de *Machine Learning* que, analizando únicamente los *opcodes* de un contrato inteligente, permita detectar con precisión si el contrato es malicioso o legítimo antes de que los usuarios interactúen con él?

Responder esta pregunta es clave para mejorar la seguridad y la confianza en el ecosistema, permitiendo tomar decisiones antes de interactuar con ellos y reduciendo significativamente el riesgo para los usuarios finales.

1.3 Objetivos generales y específicos

El objetivo general de esta tesis consiste en desarrollar un modelo predictivo basado en técnicas de *Machine Learning* que permita identificar de manera precisa y automática contratos inteligentes maliciosos en redes *Ethereum Virtual Machine* (EVM), usando solamente la información contenida en los *opcodes* del contrato.

Para alcanzar este objetivo general, se buscará abordar varios objetivos específicos. Primero, se hará al análisis de un *dataset* amplio y representativo, compuesto por contratos inteligentes previamente etiquetados como maliciosos y no maliciosos; esto incluirá la extracción y procesamiento de la información contenida en los *opcodes* de cada contrato para asegurar la suficiencia del conjunto de datos en el entrenamiento de modelos robustos y

generalizables. Seguidamente, se aplicarán técnicas avanzadas de ingeniería de características (*feature-engineering*), con énfasis particular en métodos de vectorización como TF-IDF (*Term Frequency-Inverse Document Frequency*), para transformar las secuencias de *opcodes* en vectores numéricos aptos para el entrenamiento y la evaluación de los modelos de *Machine Learning*. A continuación, se realizará la evaluación y selección de algoritmos de clasificación y se compararán los resultados contra algoritmos especialmente diseñados para manejar *datasets* desbalanceados, tales como *Easy Ensemble Classifier* y *Balanced Random Forest Classifier*, llevando a cabo una optimización de sus hiper parámetros mediante procesos de validación cruzada para maximizar su capacidad predictiva y minimizar errores de clasificación. Posteriormente, se medirá rigurosamente el desempeño del modelo final seleccionado utilizando métricas específicas y reconocidas en el ámbito de *Machine Learning*, como *Precision*, *Recall* y *F1-score*, complementado por un análisis detallado que identifique claramente las fortalezas y limitaciones del enfoque propuesto. Finalmente, se propondrá una implementación práctica del modelo resultante mediante el desarrollo conceptual de una API sencilla, eficiente y escalable, diseñada para permitir que diversas plataformas *blockchain*, billeteras virtuales, *exchanges* y usuarios finales puedan consultar en tiempo real el riesgo asociado a cualquier contrato inteligente antes de interactuar con él, contribuyendo así a mejorar significativamente la seguridad del ecosistema *blockchain*.

2. Marco Teórico

2.1 *Blockchain* y contratos inteligentes

La tecnología *blockchain* representa un cambio paradigmático en la forma en que se registran y comparten datos. Surgida inicialmente como el pilar tecnológico de Bitcoin (Nakamoto, 2008), su propuesta fundamental es la de un registro digital distribuido, inmutable y transparente. A diferencia de las bases de datos centralizadas tradicionales, susceptibles a puntos únicos de fallo y manipulación, una *blockchain* opera sobre una red de nodos donde cada participante mantiene una copia del registro. Las transacciones se agrupan en bloques que se enlazan criptográficamente de forma secuencial, creando una cadena cronológica e inalterable. Esta arquitectura distribuida fomenta la confianza sin necesidad de intermediarios, ya que la validación de las transacciones se logra mediante mecanismos de consenso acordados por la red, como *Proof-of-Work* (PoW), donde los participantes (mineros) compiten resolviendo complejos problemas criptográficos para validar transacciones y añadir bloques (Nakamoto, 2008), o *Proof-of-Stake* (PoS), donde los validadores son elegidos para crear nuevos bloques basándose en la cantidad de criptomoneda que han "apostado" o bloqueado como garantía (Ethereum Foundation, n.d.-b).

Si bien Bitcoin demostró el potencial de la *blockchain* para las transferencias de valor descentralizadas, fue la llegada de Ethereum (Buterin, 2014) la que expandió radicalmente el horizonte de aplicaciones (Antonopoulos & Wood, 2018). Ethereum introdujo el concepto de contratos inteligentes (*smart contracts*), popularizado por Nick Szabo años antes (Szabo, 1997). Estos no son contratos en el sentido legal tradicional, sino programas informáticos autoejecutables que viven en la *blockchain*. Su código define un conjunto de reglas y condiciones; y cuando estas condiciones se cumplen (por ejemplo, a través de una transacción entrante o el paso del tiempo), el contrato ejecuta automáticamente las acciones programadas.

Esta capacidad de ejecutar lógica programable directamente en la *blockchain* fue revolucionaria, habilitando un ecosistema de Aplicaciones Descentralizadas (dApps). Actualmente, este ecosistema abarca sectores tan diversos como las Finanzas Descentralizadas (DeFi)[1], que reinventan servicios financieros tradicionales, y los Tokens No Fungibles (NFTs)[2], que impulsan mercados de activos digitales únicos También le dan lugar a las Organizaciones Autónomas Descentralizadas (DAOs)[1] y permiten gestionar

[1] Entidades gobernadas por código y por las decisiones de sus miembros, que son los tienen sus los *tokens* de gobernanza.

cadenas de suministro, proveer identidad digital, gestionar juegos, y le dan un marco de trazabilidad a muchas otras áreas donde la transparencia, la automatización y la eliminación de intermediarios aportan valor.
El atractivo de los contratos inteligentes se basa en su potencial para aumentar la eficiencia, reducir costos de transacción, eliminar la necesidad de confiar en contrapartes y crear modelos de negocio completamente nuevos. Sin embargo, esta misma potencia y autonomía conllevan riesgos significativos, especialmente cuando el código que ejecutan estos contratos contiene errores o intenciones maliciosas, un tema central de esta tesis.

2.2 *Ethereum Virtual Machine* (EVM)

Para que los contratos inteligentes puedan ejecutarse de manera uniforme y predecible en una red descentralizada como Ethereum, se necesita un entorno de ejecución estandarizado. Este entorno es la *Ethereum Virtual Machine* (EVM) (Antonopoulos & Wood, 2018; Wood, 2014). La EVM puede concebirse como una computadora global, abstracta y distribuida, cuyo estado es acordado por todos los nodos participantes de la red Ethereum. Cada nodo completo de la red ejecuta la EVM para validar las transacciones y mantener la coherencia del estado global de la *blockchain*.
La EVM posee características clave que definen su funcionamiento como su aislamiento. El código de los contratos inteligentes se ejecuta en un entorno virtual aislado del sistema operativo del nodo anfitrión. Esto previene que un contrato (potencialmente malicioso) pueda afectar otros procesos o el propio nodo.
Otra característica importante es su determinismo, ya que para una misma entrada y un mismo estado inicial, la ejecución de un contrato inteligente en la EVM siempre va a producir la misma salida. Esto es fundamental para lograr el consenso en una red distribuida. Cualquier nodo puede verificar la validez de una transición de estado ejecutando el mismo código. Por ejemplo, el estado muestra una cuenta sin fondos y otra con 1 ETH, y se produce una transacción transfiriendo ese ETH, todos los nodos aplicarán la misma lógica. Como resultado, todos coincidirán en el nuevo estado donde los saldos se han actualizado.
La EVM opera como una máquina de estados basada en una pila (stack-based), lo que le permite procesar operaciones de manera secuencial y determinista. Aunque teóricamente se podría ejecutar cualquier algoritmo, su capacidad computacional está limitada en la práctica por el concepto de "Gas": una unidad que mide el costo de cada operación (cómputo, almacenamiento). Cada

transacción debe incluir una tarifa de gas para compensar a los validadores y, fundamentalmente, para prevenir bucles infinitos y ataques de denegación de servicio, incentivando la eficiencia del código.
Un aspecto crucial para esta tesis es la compatibilidad EVM. El diseño de la EVM ha sido tan influyente que muchas otras *blockchains* (como BNB Smart Chain (BSC), Polygon, Avalanche C-Chain, Arbitrum, Optimism) han adoptado la EVM o implementado versiones compatibles (Antonopoulos & Wood, 2018). Esto crea un efecto de red, permitiendo a los desarrolladores desplegar sus dApps y contratos inteligentes en múltiples cadenas con mínimas modificaciones. El *dataset* utilizado incluye contratos de varias de estas cadenas EVM compatibles, lo que muestra la relevancia de desarrollar herramientas de detección que funcionen en todo este ecosistema interconectado. Sin embargo, también cabe mencionar que las limitaciones de la propia EVM en Ethereum (principalmente escalabilidad y altos costos de gas en momentos de congestión) han impulsado el desarrollo de soluciones de Capa 2 (*Layer 2* o L2), que procesan transacciones fuera de la cadena principal de Ethereum (Capa 1) para mejorar la velocidad y reducir costos, manteniendo la seguridad derivada de Ethereum (p.ej., Arbitrum, Optimism) (Lyu et al., 2020; Ethereum Foundation, n.d.-a), así como el surgimiento de nuevas arquitecturas *blockchain* (p.ej., Solana, Polkadot) que proponen enfoques diferentes para el consenso, la ejecución de contratos y la interoperabilidad (Xu et al., 2019). Esta tesis está centrada en cadenas de L1 y L2 de EVM que son compatibles unas con otras.

2.3 *Opcodes* y *bytecodes*: Definiciones y características

Los desarrolladores escriben contratos inteligentes en lenguajes de alto nivel como Solidity o Vyper. Sin embargo, la EVM no puede ejecutar este código directamente. Antes de su despliegue, el código fuente se compila a bytecode EVM, una secuencia de bytes de bajo nivel que representa las instrucciones y los datos del contrato. Es esta secuencia de bytecode la que se almacena permanentemente en la *blockchain* y es interpretada por la EVM.
Para el análisis, generalmente se trabaja con la representación mnemotécnica del bytecode, conocida como opcodes (Códigos de Operación). Cada opcode instruye a la EVM para realizar una operación atómica. El conjunto de aproximadamente 140 opcodes definidos cubre un amplio espectro de funcionalidades, incluyendo operaciones aritméticas y lógicas, manipulación de la pila, acceso a memoria y almacenamiento, control de flujo (como JUMP y

JUMPI), interacción con el contexto de la *blockchain* (como CALLER y TIMESTAMP) y la terminación de contratos (SELFDESTRUCT).

Diagrama Conceptual:

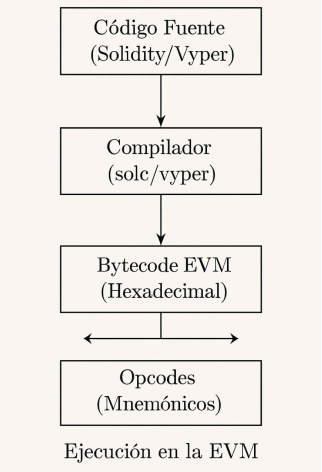

Figura 1 : Diagrama conceptual de la evolución del código fuente para ser ejecutado en la Ethereum Virtual Machine

Ejemplo de *opcodes*:

Opcode	Hex	Función principal	Costo gas (Aprox.)
STOP	00	Detiene la ejecución satisfactoriamente	0
ADD	01	Suma dos elementos de la pila	3
PUSH1	60	Empuja 1 byte a la pila	3
SLOAD	54	Lee una palabra de 32 bytes del almacenamiento	100 / 2100[1]
SSTORE	55	Escribe una palabra de 32 bytes al almacenamiento	100 / 2900 / 20000[1]
JUMP	56	Altera el contador de programa (salto)	8
REVERT	FD	Detiene ejecución, revierte estado, devuelve datos	0 (más gas restante)

CALL	F1	Llama a otro contrato/dirección	Variable (alto)
SELFDESTRUCT	FF	Destruye el contrato y transfiere el saldo restante	5000 (o negativo)[2]

[1] Los costos de SLOAD/SSTORE varían mucho según si el slot de memoria está "frío" o "caliente".
[2] SELFDESTRUCT otorga un reembolso de gas negativo.

Tabla 1 : Ejemplos de Opcodes disponibles y su representación en código hexadecimal con su función principal y costo en gas de Ethereum.

La razón fundamental para enfocar esta tesis en el análisis de *opcodes* es que estos están disponibles públicamente mientras que el código fuente de Solidity/Vyper a veces no se verifica ni se publica en exploradores de bloques como Etherscan[2], el *bytecode* (y por ende, su representación en *opcodes*) siempre es público y accesible para cualquier contrato desplegado. Esto supera una barrera crítica para el análisis a gran escala, aunque se deba decodificar los *bytecodes* a *opcodes*.
También son una representación fiel del contrato ya que reflejan la lógica real que será ejecutada por la EVM, sin la abstracción del lenguaje de alto nivel como la versión de las librerías que se usan para Solidity u otros textos presentes en el código fuente que podría ser muy ruidoso. Los patrones maliciosos o vulnerabilidades generalmente se manifiestan en secuencias específicas o en el uso particular de ciertos *opcodes* de bajo nivel (p.ej., un uso inadecuado de DELEGATECALL, patrones de bucles sospechosos antes de un REVERT, o la presencia de SELFDESTRUCT en contextos extraños).
Sin embargo, el análisis directo de *opcodes* puede ser muy complejo porque la abstracción es muy baja, interpretar la semántica de largas secuencias de *opcodes* es complejo y requiere conocimiento experto, además el volúmen de código suele ser muy grande haciendo que el análisis manual sea inviable. Existe también la posibilidad de que los atacantes puedan intentar ocultar la lógica maliciosa mediante técnicas de ofuscación a nivel de *bytecode*, como la inserción de código muerto (*opcodes* irrelevantes), o la generación dinámica de código.
Estos desafíos motivan el uso de técnicas de *Machine Learning*, como las exploradas en esta tesis, que pueden aprender patrones complejos a partir de grandes volúmenes de secuencias de *opcodes*, superando las limitaciones del

[2] Etherscan es el explorador de bloques de Ethereum, existe un explorador de bloques por cadena

análisis manual o basado en reglas heurísticas fijas. De todas formas, esto no garantiza que estos contratos puedan ser detectados en su totalidad.

2.4 Contratos inteligentes maliciosos: Riesgos y desafíos actuales

La misma flexibilidad y autonomía que hacen poderosos a los contratos inteligentes también los convierten en un medio atractivo para actividades maliciosas. Un contrato inteligente malicioso es aquel diseñado intencionadamente para defraudar, robar fondos, interrumpir operaciones o explotar a los usuarios que interactúan con él. El aumento en la cantidad de estos contratos representa una amenaza significativa para la seguridad y la confianza en *blockchain*.

Las tácticas empleadas por los actores maliciosos son diversas y evolucionan constantemente (Atzei et al., 2017). Se pueden agrupar en dos grandes categorías. La primera son los contratos diseñados intencionalmente para el fraude, como los esquemas Ponzi, los honeypots que simulan una vulnerabilidad para atraer fondos que luego no pueden ser retirados, los rug pulls comunes en el espacio DeFi y NFT, y los contratos de phishing.

La segunda categoría corresponde a contratos con vulnerabilidades no intencionales que son explotadas por atacantes. Entre las más conocidas se encuentran la Reentrada (Re-entrancy), famosa por el hack de The DAO en 2016 (Luu et al., 2016); el desbordamiento o subdesbordamiento de enteros (Integer Overflow/Underflow); las llamadas externas inseguras; y la dependencia de variables predecibles como block.timestamp para lógica crítica.

Estas últimas y el *Private Key Compromise*, que se refiere a compartir las llaves privadas de la o las wallets autorizadas para minar transacciones, no podrían ser detectadas mediante estas técnicas ya que son casos completamente diferentes y muy particulares, y para estos casos no existen grandes patrones que se puedan modelar mediante técnicas de *Machine Learning*.

Las consecuencias de los contratos inteligentes maliciosos o vulnerables son graves y una realidad constante en el ecosistema cripto. La consecuencias más directa son las enormes pérdidas financieras, con miles de millones de dólares robados, bloqueados o perdidos anualmente debido a hacks, fallos de diseño y estafas planificadas. Por un lado, tenemos las vulnerabilidades en contratos que pretendían ser legítimos: el hack de The DAO en 2016 explotó un fallo de "llamada recursiva" en su código para drenar unos 50 millones de dólares en

ETH, lo que llevó a un polémico *hard fork*[3] de Ethereum para revertirlo. Más recientemente, el hack del *bridge* Ronin en 2022 no fue un fallo directo del contrato del *bridge*[4] en sí, sino el compromiso de las claves privadas de los validadores que lo controlaban mediante un esquema multi-firma (gestionado por contratos), permitiendo a los atacantes retirar más de 600 millones de dólares. Por otro lado, existen contratos diseñados específicamente para estafar y estos son lo que más nos interesa encontrar con estos métodos, como fue el caso del token Squid Game (SQUID) en 2021, su contrato inteligente contenía código malicioso que impedía activamente que la mayoría de los compradores pudieran vender sus tokens. Esto creó un alza artificial del precio, permitiendo a los creadores anónimos ejecutar un *rug pull*, vendiendo sus propios tokens y drenando la liquidez (unos 3.3 millones de dólares en ese momento), dejando a los inversores con activos sin valor.
Según el portal de información crypto Defillama[5], los *hacks* en cripto representan pérdidas por once mil millones de dólares desde el 2018 a la actualidad. Esta métrica incluye todas las técnicas mencionadas e indica que esta problemática es uno de los principales problemas a resolver en la industria. Otra consecuencia de estos ataques es la pérdida de la confianza ya que cada incidente de alto perfil daña la reputación del espacio *blockchain* y DeFi, disuadiendo a nuevos usuarios e inversores institucionales de ingresar al ecosistema.

La detección temprana de estos contratos maliciosos presenta un desafío complejo por varias razones, en su mayoría relacionadas al diseño de la *blockchain* y a la naturaleza misma de los contratos. Una dificultad primordial es la opacidad del código fuente; como se mencionó previamente, la falta de disponibilidad generalizada del código fuente legible por humanos limita severamente las auditorías manuales y obstaculiza la eficacia de muchas herramientas de análisis estático tradicionales. Además, el volumen y la velocidad con que se despliegan nuevos contratos, corriendo de a miles diariamente en las principales *blockchains* EVM, hacen imposible revisarlos todos manualmente. A esto se suma la complejidad y sutileza de los ataques, que pueden estar ocultos en interacciones entre múltiples contratos o explotar vulnerabilidades sutiles no obvias ni siquiera en el código fuente, con

[3] Un hard fork es una actualización del protocolo de una *blockchain* que no es compatible con versiones anteriores. Obliga a los nodos de la red a adoptar las nuevas reglas, pudiendo dividir la cadena si no hay consenso.
[4] Un bridge es un contrato que permite interactuar *blockchains* diferentes entre sí.
[5] Defillama es una plataforma que brinda información sobre protocolos y cadenas de *blockchain*, en su sección de hacks recopilan datos sobre ataques en crypto. https://defillama.com/hacks

atacantes que a veces emplean técnicas de ofuscación para dificultar el análisis del *bytecode*. Finalmente, la propia evolución del ecosistema implica que los atacantes desarrollan constantemente nuevas tácticas y explotan vulnerabilidades emergentes a medida que surgen nuevos *tokens*, protocolos DeFi complejos o puentes entre cadenas.

Estos desafíos demuestran las limitaciones de los métodos de detección actuales. Las auditorías manuales, aunque valiosas, son costosas, lentas, requieren expertos escasos y no escalan para cubrir la masa de contratos ya desplegados, aplicándose principalmente a proyectos de alto perfil antes de su lanzamiento. Las herramientas de *análisis estático* (SAST) constituyen la primera línea de defensa en la auditoría de contratos. Herramientas líderes en el sector como Slither, Mythril y Securify operan analizando el código fuente (si está disponible) o el bytecode para detectar patrones de vulnerabilidad conocidos, como problemas de reentrada, desbordamientos de enteros o dependencias de timestamp. Slither, por ejemplo, traduce el código a un lenguaje intermedio llamado SlithIR para realizar un análisis de flujo de datos y detectar más de 40 tipos de vulnerabilidades comunes. Mythril, por su parte, utiliza ejecución simbólica sobre el bytecode para explorar diferentes caminos de ejecución y encontrar estados que violen las propiedades de seguridad.

Sin embargo, estos enfoques presentan limitaciones fundamentales que motivan la presente investigación. Primero, su eficacia es máxima sobre el código fuente en Solidity, pero se reduce considerablemente cuando solo se dispone del bytecode, que es el caso más común para los contratos ya desplegados. Segundo, están diseñados para encontrar vulnerabilidades de seguridad predefinidas, no necesariamente lógica maliciosa intencional. Un contrato puede ser técnicamente seguro según estas herramientas (sin reentradas ni overflows) y aun así ser una estafa, como un honeypot que implementa una lógica de negocio fraudulenta. Finalmente, aunque potentes, herramientas como Mythril pueden enfrentar problemas de "explosión de caminos" en contratos complejos, resultando en tiempos de análisis prolongados e inviables para una evaluación a gran escala y en tiempo real.

Por su parte, el análisis dinámico (DAST) y la ejecución simbólica intentan ejecutar el contrato, real o simbólicamente, para hallar estados vulnerables, pero son computacionalmente costosos, sufren también del problema de explosión de caminos[6] y pueden no cubrir todos los escenarios de ejecución posibles. Por último, la verificación formal, si bien proporciona las mayores

[6] El término explosión de caminos se refiere a un problema fundamental en el análisis de programas, especialmente en la ejecución simbólica. Ocurre porque cada bifurcación condicional (ej., if/else, bucles con condiciones variables) en el código crea múltiples rutas de ejecución posibles haciendo que sea muy complejo analizar todos los posibles caminos.

garantías de corrección, es extremadamente compleja, requiere especificaciones formales detalladas y una alta experiencia, resultando inviable para la gran mayoría de los contratos.

Ante este panorama, enfoques basados en *Machine Learning* aplicados directamente a los *opcodes* (o características derivadas de ellos), como el propuesto en esta tesis, surgen como una alternativa o complemento prometedor (p.ej., Chen et al., 2020; Ng et al., 2021). Estos métodos ofrecen el potencial de poder escalar la solución, procesando grandes volúmenes de contratos rápidamente. Tampoco se necesitaría el código fuente ya que se opera sobre información publicada obligatoriamente y además pueden ser re-entrenados con frecuencia para ir adaptándose a las nuevas tendencias.
Por supuesto, los enfoques de *Machine Learning* también tienen sus propias limitaciones como la necesidad de datos etiquetados de calidad, lo cual es un desafío muy grande. No hay mucha gente en la industria que se dedique a catalogar este tipo de contratos. Por otro lado, tiene una interpretabilidad muy baja, pero representan una dirección de investigación crucial para mejorar la seguridad del ecosistema de contratos inteligentes.

3. Datos

3.1 Descripción del *dataset* utilizado

Para llevar adelante esta investigación, se utilizó un *dataset* compuesto por aproximadamente 122.128 contratos inteligentes desplegados en redes *blockchain* compatibles con *Ethereum Virtual Machine* (EVM). Este conjunto de datos fue provisto por una empresa de seguridad en *blockchain*, que decidió ser anónima. Para poder conformar el *dataset* esta empresa dedicó personal para analizar denuncias de estafas en redes sociales, portales de seguridad como Rekt[7], Defillama o cuentas de empresas de seguridad como Chainpatrol o CyberAlert. También se utilizaron herramientas abiertas como exploradores de *blockchain* o *Block Scout*, para hacer una investigación de cada contrato, y un experto en seguridad validó cada una de las etiquetas utilizadas, analizando cada interacción con el contrato. Se utilizaron recursos de la comunidad como etiquetas de Etherscan (Las víctimas ponen etiquetas a billeteras que son estafadores) como complemento y ayuda para su validación.
Cada contrato inteligente incluido en el *dataset* cuenta con una etiqueta binaria que indica claramente si se trata de un contrato legítimo (no malicioso) o si es un contrato malicioso (fraudulento). La distribución original del *dataset* presenta un notable desbalance entre ambas clases, siendo la mayoría contratos legítimos y una proporción significativamente menor de contratos clasificados como maliciosos. Este desbalance refleja la situación real observada en el ecosistema *blockchain*, donde la mayoría de los contratos desplegados son legítimos y solo una minoría corresponde a actividades fraudulentas.
La información principal extraída de cada contrato inteligente fue su dirección de contrato y se pre-procesó cada contrato haciendo una llamada a la *blockchain* mediante un JSON-RPC. Los JSON-RPC (*Remote Procedure Call*), son interfaces estandarizadas que permiten consultar el estado de la cadena, incluyendo el *bytecode* de los contratos desplegados, las transacciones realizadas por cada *wallet* y toda la información relevante que esa *blockchain* guarda. Se presentan como una URL y se les puede hacer peticiones mediante métodos *GET* pasándole los parámetros que necesita según el método que se quiera filtrar. En este caso se le envió la dirección del contrato a consultar. Existen varios proveedores de esta información, algunos más sofisticados que otros pero hasta los más básicos contienen la información de *bytecode* de un contrato.

[7] Rekt es una página de internet que reúne una gran cantidad de ataques en *blockchain* con un análisis de sus causas y consecuencias https://rekt.news/

Los *bytecodes* obtenidos se decodificaron en *opcodes*, que como se mencionó previamente, constituyen una representación compilada del contrato original y son accesibles públicamente en la *blockchain*. Además de los *opcodes*, el *dataset* incluye metadatos adicionales relevantes para un análisis exploratorio más profundo, tales como la dirección del contrato, la fecha de despliegue, la red *blockchain* específica en la que fue desplegado (por ejemplo, Ethereum Mainnet, Binance Smart Chain, Polygon, entre otras), una variable de si fue *Recall* o no y por supuesto la etiqueta de malicioso o no malicioso. Estos metadatos permiten realizar un análisis exploratorio más completo y entender mejor las características contextuales que podrían influir en la detección de contratos maliciosos.
Finalmente, el *dataset* fue sometido a un proceso riguroso de limpieza y preprocesamiento para asegurar la calidad y coherencia de los datos utilizados. Esto incluyó la eliminación de contratos duplicados, contratos con información incompleta o inconsistencias evidentes. Este proceso garantiza que los datos utilizados para entrenar y evaluar los modelos predictivos sean confiables y representativos del problema real que se busca resolver.

En la *Tabla 2* se resumen algunas observaciones de la estructura básica del *dataset* utilizado, mostrando ejemplos representativos de cada variable.

contract_address	experiment_2_opcodes	malicious	chain	recall_data	datemonth
0x0000000000000045166c45af0fc6e4cf31d9e14b9a	PUSH1 PUSH1 MSTORE CALLVALUE DUP1 ISZERO PUSH3...	False	eth	False	2401
0x000000000000004946c0e9f43f4dee607b0ef1fa1c	PUSH1 PUSH1 MSTORE CALLVALUE DUP1 ISZERO PUSH2...	False	eth	False	2401
0x000000000000000541e251335090ac5b47176af4f7e	PUSH1 PUSH1 MSTORE PUSH1 DUP1 SLOAD PUSH1 PUSH...	False	eth	False	2401
0x000000000000000c1cb11d5c062901f32d06248ce48	PUSH1 PUSH1 MSTORE CALLVALUE DUP1 ISZERO PUSH3...	False	eth	False	2401
0x000000000000000cb2d80a37898be43579c7b616844	PUSH1 PUSH1 MSTORE CALLVALUE DUP1 ISZERO PUSH2...	False	eth	False	2401
0xaee945a2b8e7159d0344cc3a61ed9c9d7b6e979f	PUSH1 PUSH1 MSTORE UNKNOWN PUSH1 SSTORE CALLVA...	True	arbitrum	False	2404
0x53cc88448b81457e9732f1edde60bac765f53150	PUSH1 PUSH1 MSTORE UNKNOWN PUSH1 SSTORE CALLVA...	True	arbitrum	False	2404
0x6a7f4f3b97c44e4e053bf896cf986dd33965cde0	PUSH1 PUSH1 MSTORE CALLVALUE DUP1 ISZERO PUSH1...	True	polygon	False	2404
0x4f4ef46047824oee5b398cf2994a42cb3d061922	PUSH1 PUSH1 MSTORE CALLVALUE DUP1 ISZERO PUSH2...	False	eth	False	2404

0xab4549f042a27ab278ecee09 8fe9e89a8d6594e9	PUSH1 PUSH1 MSTORE PUSH2 PUSH1 SSTORE PUSH2 PU...	False	eth	False	2404

Tabla 2. Estructura del dataset mostrando las columnas disponibles al inicio de la investigación

3.2 Análisis exploratorio y descriptivo preliminar

En esta sección se presenta un análisis exploratorio inicial del *dataset*, con el objetivo de identificar patrones, distribuciones y características relevantes que permitan comprender mejor los datos antes de aplicar los modelos de *Machine Learning*.

3.2.1 Distribución de categorías

El análisis inicial del dataset reveló que, como era de esperar, cada contrato posee una dirección única (Contract_Address), mientras que se observaron opcodes idénticos para diferentes direcciones, indicando la reutilización de código. Las variables contextuales como la cadena (chain) y el mes de despliegue (datemonth) presentan un número limitado de categorías.

3.2.2 Distribución temporal

Adicionalmente, se analizó la distribución temporal de los contratos, observándose una alta concentración en fechas específicas. Esto se debe a que, durante la conformación del dataset, a un gran volumen de datos sin fecha explícita se le asignó un valor por defecto. Dado que esta variable podría introducir un sesgo y la fecha de despliegue no se consideró relevante para el análisis de los opcodes, se decidió no utilizarla como atributo en el modelo.

3.2.3 Distribución por cadena

Como se muestra en la *Figura 2*, se exploró la distribución y la cantidad de contratos entre las diferentes cadenas a fin de poder entrenar modelos específicos para cada cadena, dado el caso de uso, los usuarios generalmente operan en una cadena al interactuar con los contratos y podría ser de gran importancia que cada modelo esté optimizado para cada una de las cadenas, ya

que pueden existir algunos contratos específicos. Se observa una gran cantidad de contratos en la cadena Etherium y cantidades menores en las demás.

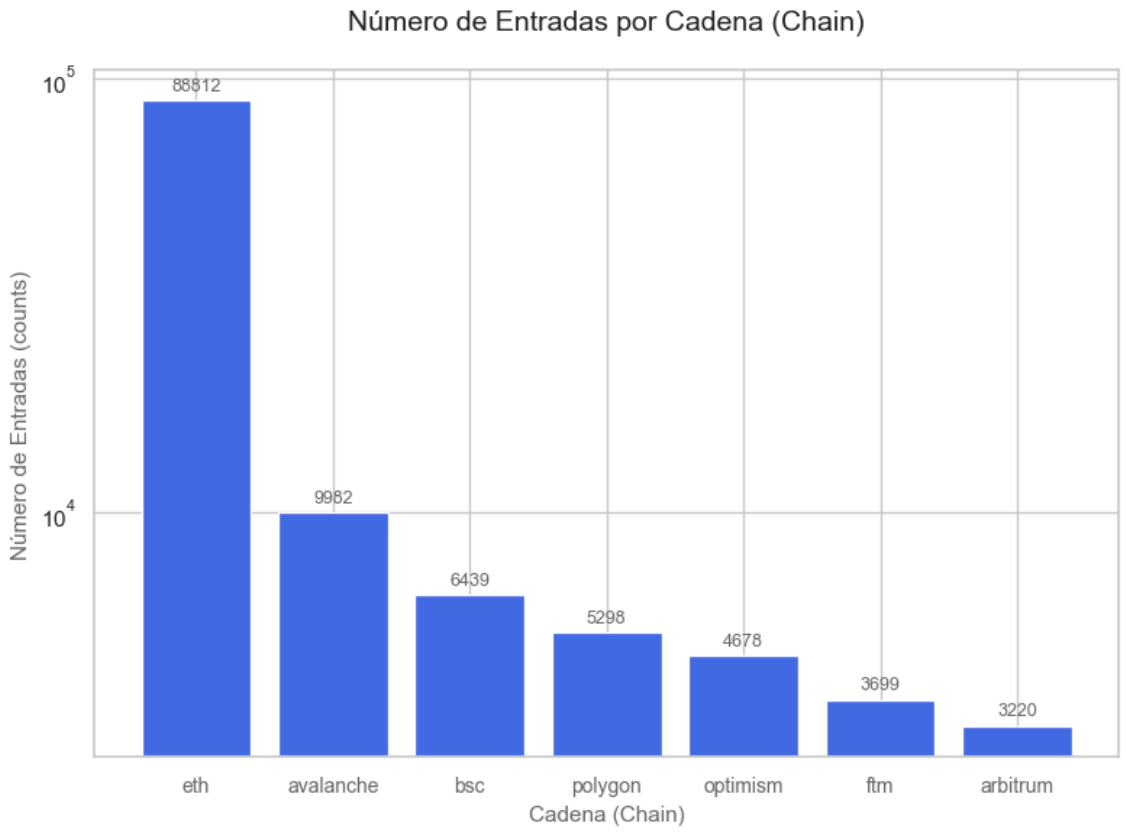

Figura 2. Cantidad de contratos por cadena en escala logarítmica

3.2.4 Proporción de contratos maliciosos

El análisis de la distribución de clases reveló un fuerte desbalance, un comportamiento esperable en un entorno real. Del total de 122.128 contratos en el dataset, solo 3.188 (un 2.6%) están etiquetados como maliciosos, mientras que el 97.4% restante corresponde a contratos legítimos. Este desbalance es un punto central para la selección y evaluación de los algoritmos de clasificación, ya que un modelo ingenuo podría alcanzar una alta *Precision* simplemente prediciendo siempre la clase mayoritaria.

3.2.5 Análisis de longitud de textos

Se realizó también un análisis sobre la longitud de los *opcodes*, medida en cantidad de palabras por contrato. La *Figura 3* muestra la distribución de

frecuencias de la longitud de los textos en la variable *OPCODES*, que será la principal predictora.

Resumen estadístico: Cantidad de *opcodes*

- Recuento: 122,128 contratos
- Promedio: 5,561.80 *opcodes*
- Desv. Estándar: 3,735.98 *opcodes*
- Valor Mínimo: 1 *opcode*
- Primer Cuartil (25%): 2,407 *opcodes*
- Mediana (50%): 4,767 *opcodes*
- Tercer Cuartil (75%): 7,833 *opcodes*
- Valor Máximo: 98,709 *opcodes*

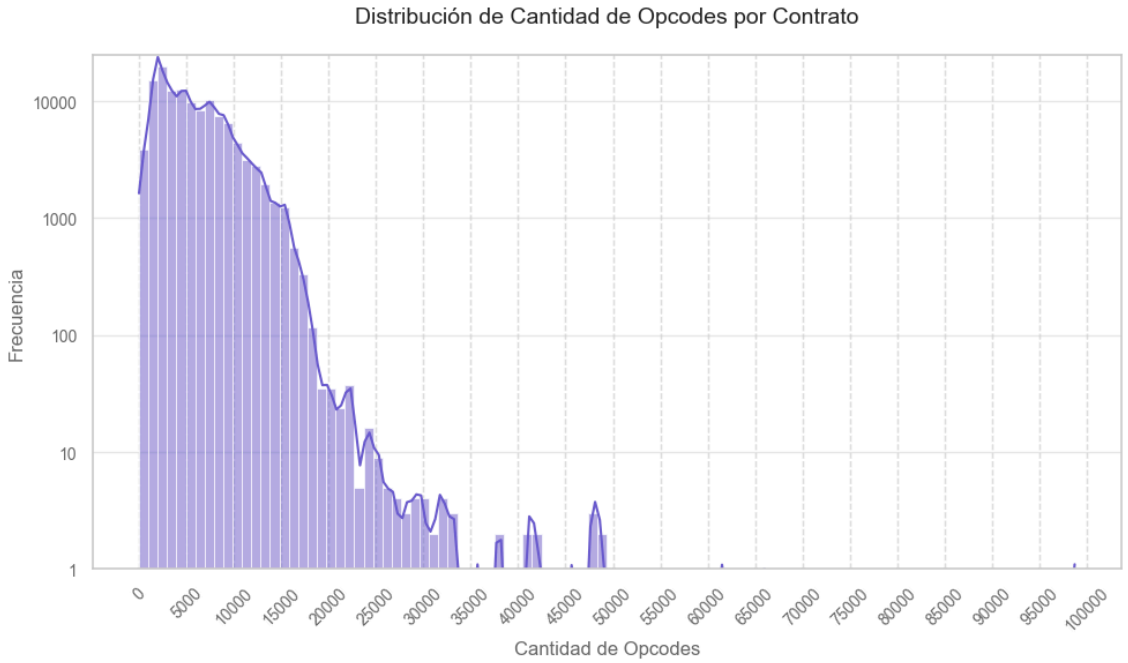

Figura 3. Distribución de la longitud de textos (cantidad de palabras)

Se observa que la mayoría de los textos se concentran entre 2.400 y 7.800 palabras, con una mediana de 4.767 y una desviación estándar de 3.733. También se observa que existe un contrato con 98.709 *opcodes*. Se decidió mantenerlo dentro del *dataset*.

3.3 *Feature Engineering*: TF-IDF (*Term Frequency-Inverse Document Frequency)*

Para poder utilizar el texto, en este caso, las secuencias de *opcodes* de los contratos inteligentes como entrada en modelos predictivos de *Machine Learning*, es necesario transformar estos datos secuenciales y categóricos en representaciones numéricas. Se adoptó la técnica TF-IDF (*Term Frequency-Inverse Document Frequency*), una metodología robusta y ampliamente reconocida en el campo del procesamiento de lenguaje natural y la recuperación de información para la clasificación textual y la ponderación de la importancia de términos (Salton & McGill, 1983; Manning et al., 2008).
TF-IDF funciona bajo la premisa de que la importancia de un término, como un *opcode* o una secuencia corta de *opcodes* en este contexto, es directamente proporcional a su frecuencia dentro de un documento específico (un contrato inteligente), pero inversamente proporcional a su frecuencia en el conjunto general de documentos (el corpus completo de contratos). Este enfoque permite asignar un mayor peso numérico a términos que son distintivos o característicos de un contrato particular. Simultáneamente, reduce el peso de términos muy comunes que aparecen en casi todos los contratos, como podrían ser *opcodes* estructurales básicos (STOP, POP), los cuales aportan poca información discriminativa para diferenciar entre contratos maliciosos y legítimos.
El cálculo de la puntuación TF-IDF para un término (t) en un documento (d) combina dos componentes clave. Primero, la Frecuencia del Término (TF) mide cuán frecuentemente aparece el término (t) dentro del documento (d). Aunque existen variantes, una definición básica podría ser contar la cantidad de veces que aparece el término en el documento. Segundo, la Frecuencia Inversa del Documento (IDF) evalúa cuán raro es el término (t) en todo el corpus. Se calcula típicamente como el logaritmo del número total de documentos (N) dividido por el número de documentos que contienen el término (t) (su frecuencia de documento, (df(t))), generalmente con ajustes de suavizado como sumar 1 al numerador y denominador para manejar términos presentes en todos los documentos o evitar divisiones por cero.

$$IDF(t) = \log\left(\frac{N+1}{df(t)+1}\right) + 1$$

La fórmula utilizada en la primera etapa asigna valores IDF bajos a términos frecuentes y valores altos a términos raros. Donde t: es el término cuya IDF se está calculando. N es el número total de documentos en la colección. df(t) es la

frecuencia de documento del término t (es decir, el número de documentos en la colección que contienen el término t).
Por otro lado, el +1 en el denominador (df(t)+1) evita la división por cero si un término no aparece en ningún documento (df(t)=0). El +1 en el numerador (N+1) y denominador juntos aseguran que el argumento del logaritmo sea siempre > 0. El +1 fuera del logaritmo asegura que la IDF nunca sea cero o negativa, incluso para términos que aparecen en todos los documentos (donde df(t) = N, el logaritmo daría log(1) = 0). Esto evita que términos muy comunes sean completamente eliminados del análisis.

$$TF - IDF(t,d) = TF(t,d) \times IDF(t)$$

El valor final se obtiene multiplicando estos dos componentes, logrando así una medida ponderada de la importancia del término en el documento dentro del contexto del corpus. Donde $TF(t,d)$ representa la puntuación o peso final del término t en el documento d y $IDF(t)$ mide la frecuencia con la que el término t aparece en el documento específico d.
Así sabemos que TF-IDF va a ser alto si el término t aparece muchas veces en el documento d (TF alto) y el término t es relativamente raro en la colección general (IDF alto) haciéndonos pensar que el término es importante y específico para ese documento.
Por otro lado TF-IDF va a ser bajo si el término t aparece pocas veces en d (TF bajo) o
el término t es muy común en toda la colección (IDF bajo, cercano a cero para palabras muy comunes) haciéndonos pensar que el término no es un buen descriptor del contenido específico de ese documento.
Para capturar no solo *opcodes* individuales sino también secuencias cortas con significado relevante para detectar patrones maliciosos o secuencias extrañas que se repiten, se configuró el procesamiento para considerar n-gramas, que son conjuntos de palabra. Específicamente, se utilizó un rango de 1 a 4 (ngram_range=(1, 4)), lo que significa que el vocabulario de "términos" no se limitó a *opcodes* individuales (unigramas), sino que también incluyó todas las secuencias contiguas de 2 *opcodes* (bigramas), 3 *opcodes* (trigramas) y 4 *opcodes* (cuatrigramas) presentes en los datos. El proceso para una secuencia hipotética como PUSH1 60 PUSH1 40 MSTORE ADD JUMP implicaría primero tratarla como una lista de *tokens* individuales. Luego, se extraerían todas las subsecuencias de longitud 1 (PUSH1, 60, etc.), longitud 2 (PUSH1 60, 60 PUSH1, etc.), longitud 3 (PUSH1 60 PUSH1, etc.) y longitud 4 (PUSH1 60 PUSH1 40, etc.). Estos n-gramas únicos, generados a partir de todos los contratos considerados

de entrenamiento, formarían el vocabulario final de características sobre el cual se calcularían las puntuaciones TF-IDF para cada contrato.

La implementación práctica de este proceso se realizó utilizando la clase Tfidf Vectorizer de la librería Scikit-learn en Python (Pedregosa et al., 2011), ya que es una herramienta estándar y eficiente para la vectorización de texto. Además del rango de n-gramas (ngram_range=(1, 4)), se estableció un número máximo de características (max_features=30000). Esta configuración instruye al vectorizador a construir el vocabulario considerando todos los unigramas, bigramas, trigramas y cuatrigramas, pero luego a conservar únicamente los 30,000 n-gramas con la frecuencia de documento más alta en todo el corpus de entrenamiento. Esta limitación es necesaria para manejar la potencial alta dimensionalidad que se da al n-gramas extensos, y además ayuda a filtrar términos extremadamente raros que podrían no generalizar bien o introducir ruido.

El resultado de este proceso de vectorización TF-IDF es una matriz numérica dispersa (*sparse matrix*). En esta matriz, cada fila corresponde a un contrato inteligente del dataset, y cada columna representa uno de los 30,000 n-gramas seleccionados como características. El valor en cada celda (i, j) es la puntuación TF-IDF del n-grama j en el contrato i, siendo cero si ese n-grama no aparece en dicho contrato. La naturaleza inherentemente dispersa de esta matriz es manejada eficientemente por librerías como Scikit-learn.

Este procedimiento concluyó con la generación de una matriz final de características numéricas, lista para ser utilizada como entrada (*input*) en los modelos predictivos que fueron desarrollados y evaluados en los siguientes capítulos de la tesis.

4. El Modelo

Después de haber preparado el *dataset* y haber hecho el análisis exploratorio de los datos, que terminó con la vectorización de las secuencias de *opcodes* mediante TF-IDF, se procedió al entrenamiento y validación de los modelos de *Machine Learning*. A continuación se explica la selección de algoritmos de clasificación considerados, el enfoque que se usó para abordar el desafío del desbalance de clase y la metodología de validación utilizada para evaluar el rendimiento de los modelos.

4.1. Explorando Diferentes Enfoques de Clasificación

Como el objetivo final es hacer una clasificación binaria y estamos partiendo de una matriz de alta dimensionalidad con una gran diferencia en la distribución de clases (tenemos pocos maliciosos y muchos no maliciosos), fue necesario explorar un conjunto de algoritmos de clasificación para identificar el más adecuado, incluyendo tanto enfoques estándar como técnicas específicamente desarrolladas para escenarios con datos desbalanceados.
Se comenzó implementando una Regresión Logística, un método estadístico conocido y testeado para predecir resultados binarios. La regresión modela la probabilidad de pertenencia a una clase (ej. "malicioso"). Primero, calcula una puntuación lineal (también llamada *logit*) a partir de las características de entrada x usando los pesos del modelo w y un término de sesgo b. Esta puntuación se define como:

$$z = w^T x + b$$

Donde z representa la puntuación lineal, w es el vector de pesos aprendidos por el modelo y x es el vector de características de la instancia de entrada. El término $w^T x$ representa el producto escalar entre los pesos y las características (el superíndice T indica la transposición del vector de pesos). Finalmente, b es el término de sesgo, también conocido como intercepto.

Luego, esta puntuación lineal z se transforma mediante la función sigmoide (o logística), denotada como σ(z), para obtener una probabilidad:

$$\sigma(z) = \frac{1}{1 + e\text{^}-z}$$

El resultado σ(z) es la probabilidad estimada de que la instancia pertenezca a la clase positiva (por ejemplo, P(y=1 | x)), y siempre va a tener un valor entre 0 y 1. Finalmente, esta probabilidad se compara con un umbral de decisión, θ (usualmente θ = 0.5), para asignar la clase final predicha ŷ. La predicción ŷ se asigna como 1 si $\sigma(z) > \theta$, y como 0 si $\sigma(z) < \theta$

Posteriormente, se implementó un *Random Forest* (RF) estándar. Este algoritmo de aprendizaje conjunto es conocido por su robustez y eficacia en tareas de clasificación con datos complejos y muchas dimensiones. Funciona construyendo un gran número de árboles de decisión individuales, digamos K árboles. Su fórmula se representa de la siguiente manera:

$$\hat{y}_{\mathrm{RF}} = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \sum_{k=1}^{K} \mathbb{I}(h_k(\mathbf{x}) = c)$$

Donde:

- $y\text{^}RF$ *es la clase final predicha por el Random Forest para la instancia* x
- $hk(x)$ *es la predicción* $(clase)$ *del árbol individual número* k *para la instancia* x.
- C *es el conjunto de todas las posibles clases* $(ej.\ \{malicioso,\ no\ malicioso\})$.
- $I(\cdot)$ *es la función indicadora:* $vale\ 1$ *si la condición dentro del paréntesis es verdadera osea que el árbol* k *predice la clase* c
- $\sum_{k=1}^{K} \mathbb{I}(h_k(\mathbf{x}) = c)$ *cuenta cuántos árboles votaron por la clase* c
- $\underset{c \in \mathcal{C}}{\operatorname{argmax}}$ *selecciona la clase* c *que recibió el mayor número de votos.*

La diversidad entre los árboles se logra mediante dos mecanismos principales. El primero es *'Bagging'* que significa que cada árbol se entrena sobre una muestra *bootstrap*, que es una muestra obtenida con reemplazo del conjunto de datos original y el segundo es 'Aleatoriedad de Características' que consiste en que cada nodo durante la construcción de un árbol, solo se considera un subconjunto aleatorio de las características (columnas del *dataset*) disponibles para determinar la mejor decisión posible.

Para determinar la predicción final de una nueva instancia, el *Random Forest* recoge las predicciones individuales de cada uno de los K árboles que lo componen. Luego, cuenta cuántos árboles votaron por cada posible clase (por ejemplo, cuántos votaron "malicioso" y cuántos votaron "no malicioso"). La clase que recibe el mayor número de votos, es decir, la predicción más frecuente entre todos los árboles, se asigna como la predicción final del

Random Forest para esa instancia. Este proceso se conoce comúnmente como voto mayoritario.
Se incluyó este algoritmo porque puede modelar relaciones no lineales y suele tener un buen rendimiento general en diversas tareas. Sin embargo, existe la preocupación de que el fuerte desbalance de clases en los datos pueda sesgar tanto el aprendizaje de los árboles individuales (favoreciendo divisiones que separan bien a la clase mayoritaria) como el resultado del voto final (donde la mayoría domina fácilmente).
Para poder atacar esta problemática particular de tener un *dataset* desbalanceado se probaron dos algoritmos que incorporan mecanismos específicos para manejar esta situación.
El *Balanced Random Forest Classifier* (BRFC) adapta directamente el algoritmo *Random Forest*. Su modificación clave ocurre durante la preparación de los datos para cada árbol individual. Antes de entrenar un árbol sobre su muestra *bootstrap*, se realiza un submuestreo interno: se conservan todas las instancias de la clase minoritaria presentes en la muestra y se selecciona aleatoriamente un número igual de instancias de la clase mayoritaria de esa misma muestra. De este modo, cada árbol aprende a partir de un conjunto perfectamente balanceado. Se espera que este enfoque mejorara significativamente la sensibilidad del modelo hacia la clase minoritaria, al forzar a cada componente del ensamble a prestar igual atención a ambas clases.
El *Easy Ensemble Classifier* (EEC) propone una arquitectura de ensamble diferente. Construye múltiples clasificadores base (en este caso, se utilizó AdaBoost para cada uno), y cada uno de ellos se entrena sobre un subconjunto de datos distinto, pero siempre balanceado. Para crear cada subconjunto balanceado, se toman todas las instancias de la clase minoritaria del conjunto de entrenamiento original y se combinan con una muestra aleatoria (de igual tamaño) extraída de la clase mayoritaria. AdaBoost, a su vez, es un algoritmo de *boosting* que pondera más las instancias mal clasificadas en iteraciones sucesivas, lo que puede ayudar a enfocarse en patrones difíciles. Al repetir este proceso de crear subconjuntos balanceados y entrenar clasificadores base sobre ellos varias veces, se genera un comité de clasificadores. La predicción final agrega la información de todos los miembros del comité. Se eligió esta técnica por su potencial para explorar diversas interacciones entre la clase minoritaria y diferentes partes de la clase mayoritaria, aprovechando la fortaleza de AdaBoost en cada subproblema balanceado.
Por lo tanto, la selección de algoritmos abarcó desde un modelo simple (Regresión Logística) y un ensamble potente estándar (*Random Forest*) hasta dos técnicas avanzadas diseñadas para el desbalance (*Balanced Random Forest* y *Easy Ensemble*). Esta variedad permitió una evaluación comparativa para

determinar la estrategia más efectiva en el contexto específico de la detección de contratos inteligentes maliciosos.

4.2. Cómo se Abordó el Desbalance de Clases

El desbalance en el *dataset* utilizado representa un desafío significativo que puede llevar a los algoritmos a ignorar la clase minoritaria. Para contrarrestar esta tendencia, se recurrió a los mecanismos integrados en BRFC y EEC.
En el *Balanced Random Forest Classifier,* el ajuste clave es el submuestreo interno por árbol. Al construir cada árbol, se asegura que la cantidad de ejemplos legítimos y maliciosos sea idéntica en los datos de entrenamiento de ese árbol específico. Esto se logra mediante submuestreo aleatorio de la clase mayoritaria dentro de la muestra *bootstrap* de cada árbol. Forzar este equilibrio a nivel de cada componente del ensamble busca asegurar que el modelo global aprenda a distinguir eficazmente ambas clases, prestando la debida atención a los patrones minoritarios.
Con el *Easy Ensemble Classifier*, la estrategia se basa en la creación de múltiples subproblemas balanceados. El algoritmo genera varios *datasets*, cada uno conteniendo la totalidad de los contratos maliciosos y una muestra aleatoria (de igual tamaño) de los contratos legítimos. Un clasificador base (AdaBoost) se entrena independientemente en cada uno de estos *datasets*. El ensamble final combina las predicciones de estos clasificadores base. Esta aproximación permite que el modelo explore la relación entre la clase minoritaria y diferentes segmentos de la clase mayoritaria a través de los distintos miembros del ensamble, buscando una comprensión más robusta y diversa.
Ambas técnicas, BRFC y EEC, implementan formas estratégicas de submuestreo dentro de un marco de ensamble para asegurar que la clase minoritaria tenga una representación efectiva durante el aprendizaje, lo cual es esencial para el objetivo de detectar contratos maliciosos.

4.3. Proceso de Entrenamiento y Validación Cruzada

Se implementó una metodología rigurosa para entrenar los modelos seleccionados y evaluar su rendimiento de manera objetiva y generalizable.
En primer lugar, se realizó una división inicial del conjunto de datos completo. Un subconjunto de 30.596 registros (aproximadamente el 25%) se reservó como conjunto de prueba final (hold-out set). Este conjunto se mantuvo completamente al margen durante las fases de entrenamiento y ajuste del modelo, destinándose exclusivamente a una evaluación final y única del

modelo seleccionado en condiciones lo más cercanas posible a un escenario real de uso con datos no vistos. Reservar un conjunto de prueba (*hold-out*) final es una práctica recomendada para obtener una estimación imparcial del rendimiento del modelo en datos no vistos (Hastie et al., 2009).

El conjunto restante, compuesto por 91.532 registros (el 75%), se utilizó para el desarrollo y la evaluación de los modelos mediante validación cruzada (VC) estratificada de 5 partes (*5-Fold Stratified Cross-Validation*). Este procedimiento consiste en dividir dicho conjunto en 5 partes (*folds*) de tamaño similar. Seguidamente, se realizan 5 iteraciones de entrenamiento y evaluación. En cada iteración, 4 *folds* o carpetas se usan para entrenar el modelo y el *fold* restante se utiliza para evaluarlo. La estratificación es clave en este caso, porque asegura que la proporción original de contratos maliciosos y legítimos se mantenga constante en cada *fold*, lo cual es fundamental dado el desbalance. Al promediar los resultados de las 5 iteraciones, se obtiene una estimación más robusta y menos dependiente de una partición específica del rendimiento de cada algoritmo. La validación cruzada estratificada es una técnica esencial cuando se trabaja con conjuntos de datos desequilibrados, ya que ayuda a evitar estimaciones sesgadas del rendimiento del modelo (Kohavi, 1995).

Durante este proceso de validación cruzada, se calcularon y registraron diversas métricas de rendimiento para cada modelo, con el fin de obtener una visión completa de su comportamiento. Se prestó especial atención a las métricas dependientes del umbral. Se evaluaron la *Precision* (proporción de predicciones positivas correctas), el *Recall* o Sensibilidad (proporción de positivos reales detectados, considerada de alta prioridad), y el F1-Score (media armónica de *Precision* y *Recall*) a través de un rango de posibles umbrales de decisión. Esto permitió analizar el compromiso (*trade-off*) entre detectar más contratos maliciosos y minimizar las falsas alarmas. Para problemas de detección de anomalías o clasificación con datos desbalanceados, es crucial evaluar el rendimiento utilizando métricas como *Precision*, *Recall* y *F1-Score*, dado que la exactitud (*accuracy*) puede ser engañosa (Tharwat, 2021).

- *Precision*: Mide qué proporción de las predicciones de "malicioso" fueron correctas (TP / (TP + FP)). Importante para evaluar las falsas alarmas.
- *Recall* (Sensibilidad): Mide qué proporción de los contratos realmente maliciosos se lograron detectar (TP / (TP + FN)). Considerada una métrica de alta prioridad para este problema.
- *F1-Score*: La media armónica de *Precision* y *Recall* (2 * P * R / (P + R)), que busca un equilibrio entre ambas.

Este enfoque metodológico, combinando la reserva de un conjunto de prueba con la validación cruzada estratificada y el cálculo de un conjunto completo de

métricas relevantes (dependientes del umbral), proporciona un marco sólido para entrenar, comparar y evaluar los algoritmos explorados, sentando las bases para la discusión de resultados del capítulo siguiente.

4.4. Optimización de Hiper Parámetros y Umbral de Decisión

Un aspecto que fue objeto de optimización explícita fue el umbral de decisión. Los modelos de clasificación probabilística generan una puntuación o probabilidad (entre 0 y 1). Convertir esta probabilidad en una clasificación final (malicioso/legítimo) requiere un umbral. Si bien 0.5 es el valor por defecto, no necesariamente es el óptimo, sobre todo en problemas desbalanceados donde el coste de los errores (Falsos Positivos vs. Falsos Negativos) no es simétrico.
Aprovechando las probabilidades predichas durante la validación cruzada, se analizó el efecto de variar este umbral sobre las métricas clave. Se examinó cómo cambiaban la *Precision*, el *Recall* y el *F1-Score* al modificar el umbral en un rango de valores (p.ej., de 0.1 a 0.9 en pequeños pasos). Se visualizaron las curvas *Precision-Recall* para entender mejor este compromiso. El objetivo fue identificar el umbral que ofreciera el mejor balance práctico, maximizando el *F1-Score* o maximizando el *Recall* sin sacrificar excesivamente la *Precision*. Este análisis permitió seleccionar un umbral operativo más adecuado que el estándar de 0.5 para los modelos finales.

5. Resultados

Este capítulo se adentra en el corazón empírico de la investigación, presentando y, fundamentalmente, interpretando los resultados derivados de la evaluación de los modelos de *Machine Learning* diseñados para la detección de contratos inteligentes maliciosos. Habiendo establecido el contexto, la problemática de la opacidad del código y el desafío del desbalance de clases, y detallado la metodología de preparación de datos y modelado, ahora se examina críticamente el desempeño de *Random Forest* (RF), *Easy Ensemble Classifier* (EEC), *Balanced Random Forest* (BRF) y Regresión Logística (LR). El análisis se desarrolla en dos fases secuenciales y complementarias: la exploración del rendimiento en validación cruzada y la evaluación definitiva sobre un conjunto de testeo independiente, culminando en una discusión razonada sobre la elección del modelo más adecuado y sus implicaciones.

5.1 Desempeño y Patrones Observados en Validación Cruzada (VC)

La validación cruzada estratificada de 5 folds, aplicada sobre el conjunto de entrenamiento, proporcionó la primera oportunidad para comparar el potencial intrínseco de los algoritmos y optimizar parámetros clave como el umbral de decisión. La optimización se enfocó en maximizar el F1-Score, una métrica que busca equilibrar la Precision y el Recall, especialmente relevante en escenarios de clasificación con clases desbalanceadas como el que teníamos.
Al examinar el rendimiento promedio agregado de los modelos a través de todas las cadenas EVM incluidas, los resultados iniciales parecieron confirmar la hipótesis de que las técnicas especializadas en el manejo del desbalance podrían ofrecer una ventaja. La Tabla 3 resume las métricas clave en el punto óptimo F1 para cada modelo durante esta fase de VC.

Modelo	F1-Score	Precision	Recall	Umbral
Balanced RF (BRF)	0.699	61.1%	81.8%	0.78
Easy Ensemble (EEC)	0.664	54.3%	85.4%	0.52
Random Forest (RF)	0.599	64.5%	56.0%	0.85

Regresión Logística (LR)	0.539	53.6%	54.2%	0.57

Tabla 3: Resumen del Rendimiento General Óptimo en Validación Cruzada (Agregado). Nota: Agregado indica que se entrenó con todas las chains juntas

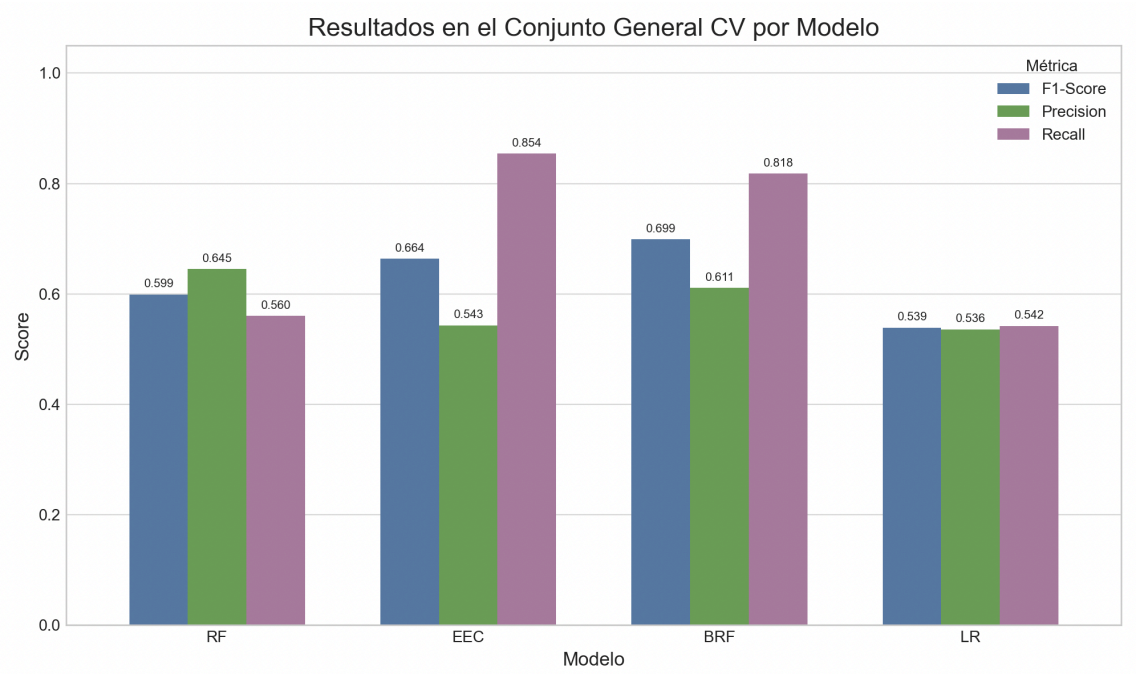

Figura 4 : Visualización de Rendimiento General Óptimo en Validación Cruzada (Agregado)

De acuerdo con la *Tabla 3*, *Balanced Random Forest* (BRF) se posicionó como el líder en *F1-Score* (0.699), seguido por *Easy Ensemble Classifier* (EEC) (0.664). Ambos modelos alcanzaron estos valores gracias a tasas de *Recall* notablemente altas (81.8% y 85.4%, respectivamente). Este alto Recall sugiere que las estrategias de submuestreo internas, diseñadas para dar mayor peso a la clase minoritaria, están funcionando eficazmente dentro de los datos de entrenamiento y validación, permitiendo a los modelos identificar una gran proporción de los contratos maliciosos presentes en esas particiones. Sin embargo, esta alta sensibilidad hacia la clase minoritaria a menudo se logra a expensas de la *Precision*. De hecho, fue el *Random Forest* (RF) estándar el que obtuvo la *Precision* más alta (64.5%) durante la VC, aunque su *F1-Score* general (0.599) fue menor debido a un Recall más conservador (56.0%). La Regresión Logística (LR), como era previsible por su naturaleza lineal y su falta de

mecanismos específicos para el desbalance, mostró el rendimiento más limitado (F1=0.539). Este panorama inicial nos muestra un escenario donde las técnicas de balanceo parecen prometedoras, pero indican un *trade-off* fundamental entre detectar muchos contratos maliciosos (alto *Recall*) y asegurar que las detecciones fueran correctas (alta *Precision*) aunque el *Recall* en este caso es mucho más valorado que la *Precision* ya que se presume que los usuarios estarían dispuesto a soportar más falsos positivos siempre y cuando se puedan encontrar más contratos maliciosos. El costo de errarle en la predicción puede ser muy alto si se cataloga un contrato malicioso como seguro.

Más allá del rendimiento agregado, el análisis desagregado por cadena reveló una complejidad adicional y subrayó la heterogeneidad del problema. La *Tabla 4* ofrece una visión detallada del mejor *F1-Score* alcanzado por cada modelo en cada cadena durante la validación cruzada, complementada visualmente por las *Figuras 5, 6 y 7*, que ilustran las comparaciones de *F1-Score, Precision* y *Recall*, respectivamente. Sin embargo, para una interpretación precisa de estos resultados, es crucial considerar el número de contratos analizados en cada cadena, como se detalla en la misma tabla.

Cadenas como Arbitrum, BSC y Polygon, que cuentan con un número considerable de muestras maliciosas (922, 657 y 125, respectivamente), parecen presentar patrones de opcodes más fácilmente discernibles por los modelos, llevando a *F1-Scores* excepcionalmente altos. En el otro extremo, el rendimiento en cadenas como FTM (con solo 3 muestras maliciosas) o Avalanche (18) debe ser interpretado con cautela, ya que sus métricas, especialmente los valores de Recall "redondos", son estadísticamente menos robustas debido al bajo volumen de datos positivos. En estos casos, más que un entorno "desafiante", la principal limitación es la falta de datos etiquetados para una evaluación concluyente.

El caso de Ethereum es particularmente interesante: a pesar de ser la cadena con el mayor número de contratos y muestras maliciosas en términos absolutos (961), la proporción de estos es la más baja, presentando el desbalance de clases más severo. Esto, combinado con una posible mayor diversidad y sofisticación en los contratos desplegados, podría explicar por qué los modelos obtienen un rendimiento más moderado en esta red.

La tendencia general en validación cruzada fue que los modelos de balanceo (BRF y EEC) dominaron en Recall (Figura 7), a menudo alcanzando valores cercanos al 100%, lo que les permitió obtener frecuentemente los *F1-Scores* más altos por cadena *(Figura 4)*. No obstante, *Random Forest* mantuvo una ventaja en *Precision* (*Figura 6*), como se vio claramente en Arbitrum y Optimism. Esta fase de validación cruzada, por lo tanto, estableció un

escenario complejo: las técnicas de balanceo parecen excelentes para maximizar la detección inicial, pero su *Precision* y, especificamente, su capacidad para generalizar este rendimiento a datos completamente nuevos, aún están por determinarse.

Cadena (Total / Contratos Maliciosos)	Modelo	Mejor F1	Precision	Recall
ETH (88812 / 961)	RF	0.414	55.7%	32.9%
	LR	0.199	22.1%	18.0%
	EEC	0.477	41.8%	55.6%
	BRF	0.522	46.7%	59.1%
BSC (1037 / 657)	RF	0.728	61.6%	89.0%
	LR	0.621	61.2%	63.0%
	EEC	0.775	63.3%	99.8%
	BRF	0.782	64.5%	99.1%
Avalanche (96 / 18)	RF	0.188	11.5%	50.0%
	LR	0.188	11.5%	50.0%
	EEC	0.319	18.9%	100.0%
	BRF	0.536	39.5%	83.3%
Arbitrum (1143 / 922)	RF	0.905	93.9%	87.3%
	LR	0.881	86.3%	89.9%
	EEC	0.892	80.6%	99.9%
	BRF	0.914	87.1%	96.1%
Polygon (327 / 125)	RF	0.474	33.7%	80.0%
	LR	0.440	34.2%	61.6%
	EEC	0.553	38.2%	100.0%
	BRF	0.579	44.4%	83.2%

Cadena (Total / Contratos Maliciosos)	Modelo	Mejor F1	Precision	Recall
Optimism (110 / 23)	RF	0.522	52.2%	52.2%
	LR	0.315	21.2%	60.9%
	EEC	0.346	20.9%	100.0%
	BRF	0.491	43.3%	56.5%
FTM (7 / 3)	RF	0.400	50.0%	33.3%
	LR	0.400	50.0%	33.3%
	EEC	0.600	42.9%	100.0%
	BRF	0.600	42.9%	100.0%

Tabla 4: Rendimiento Óptimo (Mejor F1-Score) por Modelo y Cadena en Validación Cruzada.

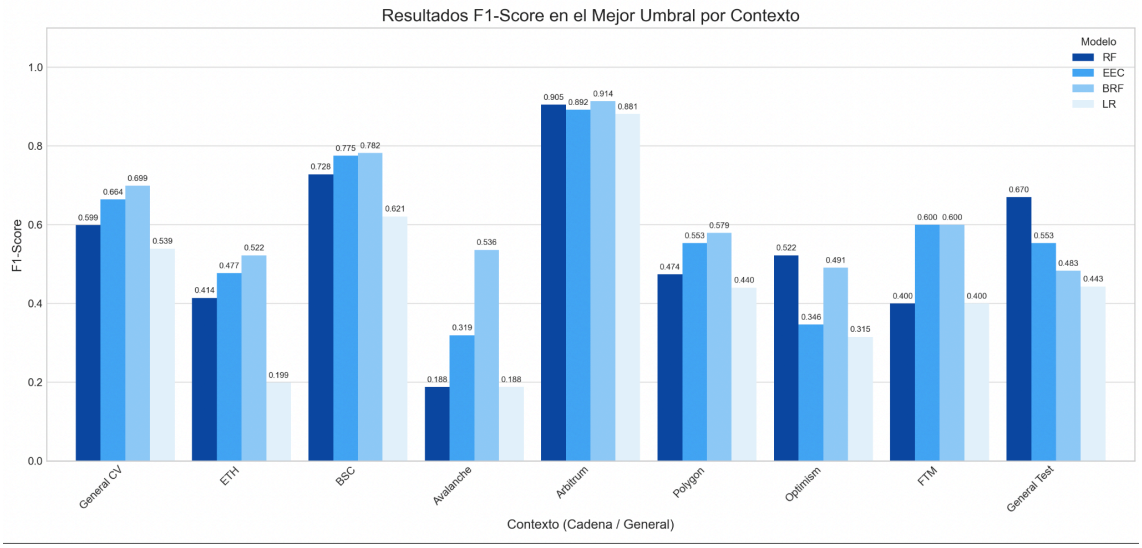

Figura 5 : F1-Score en el Mejor Umbral por Cadena, General Test y General VC.

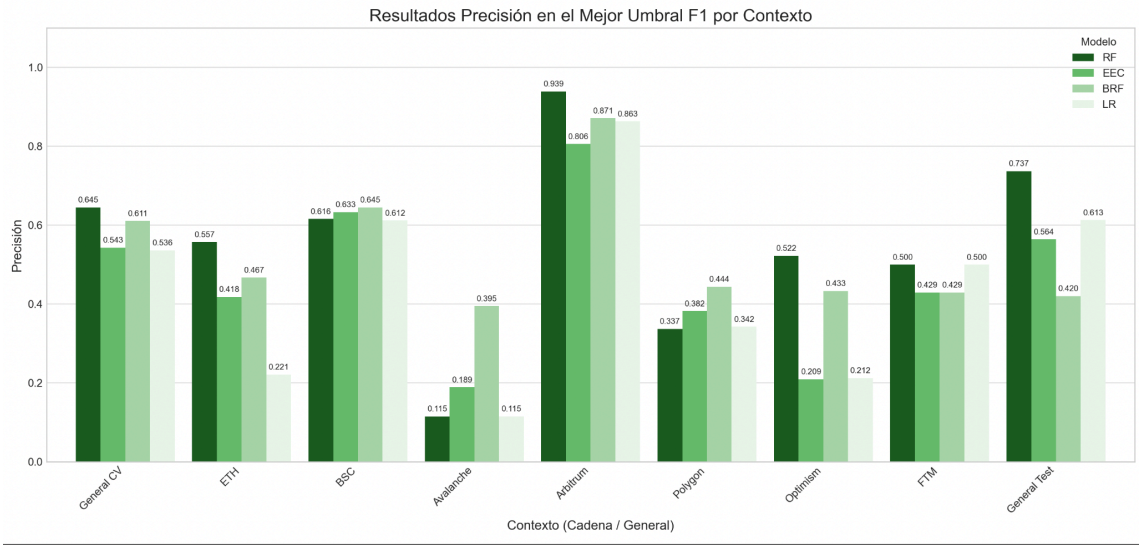

Figura 6 : Precision en el Mejor Umbral por Cadena, General Test y General VC

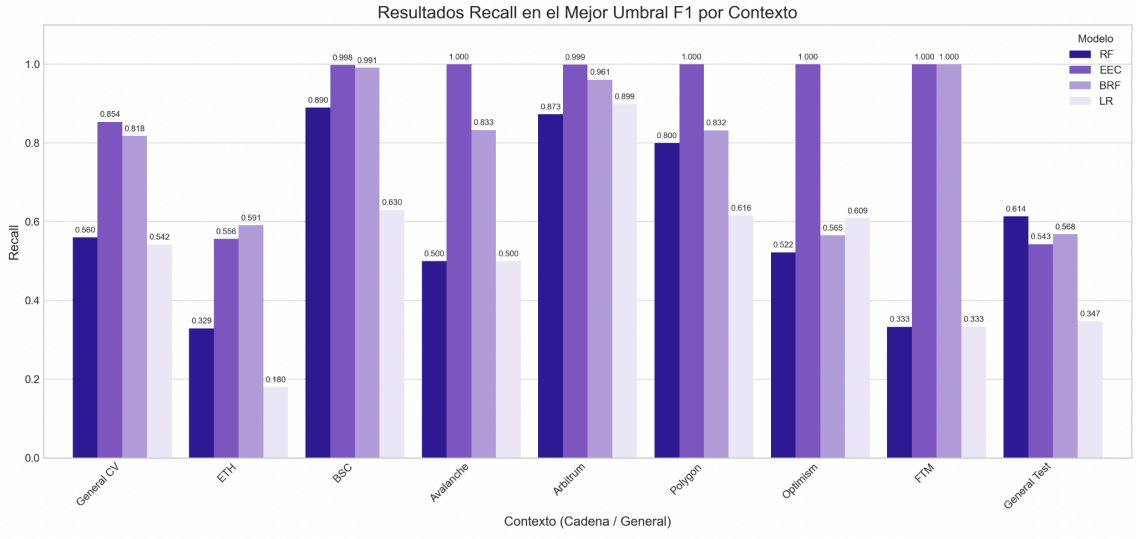

Figura 7 : Recall en el Mejor Umbral por Cadena, General Test y General VC

La *Tabla 4* y las *Figuras 5, 6* y *7* argumentan con fuerza sobre la influencia del contexto específico de la cadena. Cadenas como Arbitrum parecen presentar patrones de *opcodes* en contratos maliciosos que son más fácilmente discernibles por los modelos, llevando a *F1-Scores* excepcionalmente altos (superiores a 0.88 para todos excepto LR). En contraste, Avalanche se perfila como un entorno mucho más desafiante, donde la mayoría de los modelos lucharon por superar un *F1-Score* de 0.2. Es particularmente interesante notar

que BRF, a pesar de las dificultades generales en Avalanche, logró un F1 de 0.536 en VC, sugiriendo que su mecanismo de balanceo fue especialmente útil para extraer señales débiles en ese contexto durante el entrenamiento. Ethereum, a pesar de ser la cadena principal, mostró un rendimiento intermedio, lo que podría reflejar una mayor diversidad y sofisticación en los contratos desplegados. La tendencia general en VC fue que BRF y EEC dominaron en *Recall* (*Figura 7*), a menudo alcanzando valores cercanos al 100%, lo que les permitió obtener frecuentemente los F1-Scores más altos por cadena (*Figura 5*). No obstante, RF a menudo mantuvo una ventaja en *Precision* (*Figura 6*), como se vio claramente en Arbitrum y Optimism. Esta fase de validación cruzada, por lo tanto, estableció un escenario complejo: las técnicas de balanceo parecen excelentes para maximizar la detección inicial, pero su fiabilidad (*Precision*) y, crucialmente, su capacidad para generalizar este rendimiento a datos completamente nuevos, aún están por determinarse.

5.2 Evaluación del Desempeño en el Conjunto de Testeo

La verdadera medida de la utilidad práctica de un modelo predictivo es su capacidad para funcionar correctamente ante datos que no ha visto previamente. Por ello, la evaluación sobre el conjunto de testeo independiente (30,596 contratos) fue la etapa decisiva de este análisis comparativo. Los resultados, presentados en la *Tabla 5* y visualizados en la *Figura 8,* reconfiguraron significativamente las conclusiones iniciales.

Modelo	F1-Score (Test)	Precision (Test)	Recall (Test)
Random Forest (RF)	0.670	73.7%	61.4%
Easy Ensemble (EEC)	0.553	56.4%	54.3%
Balanced RF (BRF)	0.483	42.0%	56.8%
Reg. Logística (LR)	0.443	61.3%	34.7%

Tabla 5: Resumen del Rendimiento de los Modelos en el Conjunto de Testeo (en el mejor umbral F1).

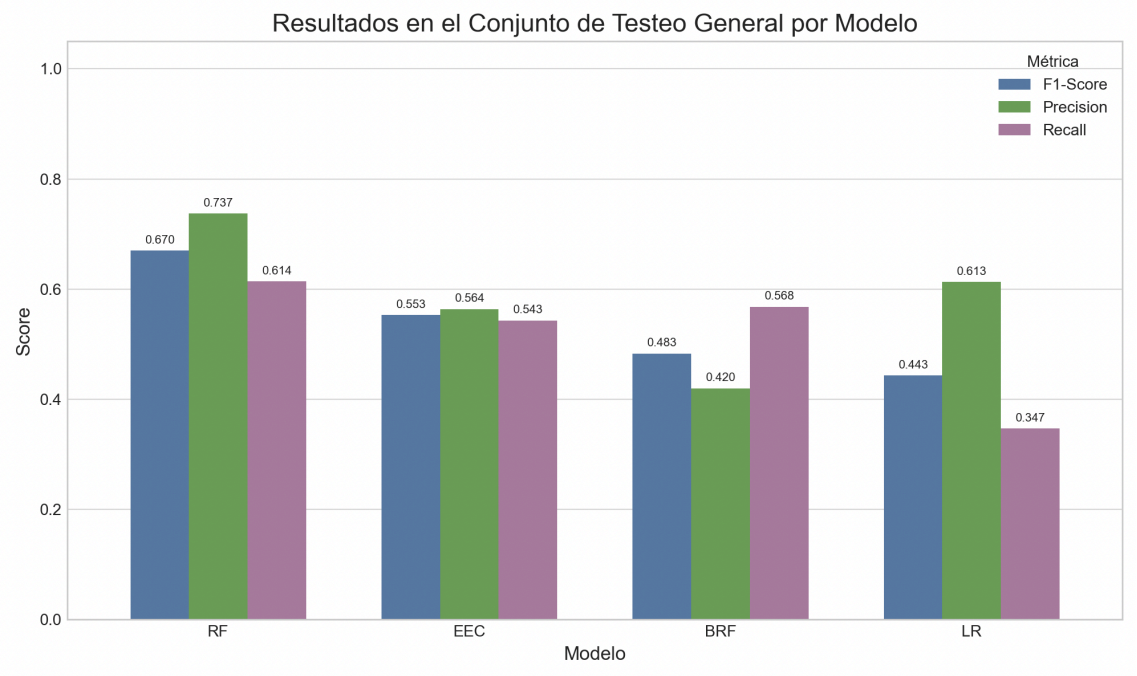

Figura 8 : Resultados en el Conjunto de Testeo General por Modelo, esto indica todas las chains juntas.

La *Tabla 5* y la *Figura 8* argumentan de forma contundente a favor del *Random Forest* (RF) estándar como el modelo más robusto y efectivo en términos de generalización. Contrario a las expectativas generadas por la VC, RF no solo obtuvo el *F1-Score* más alto (0.670) en el conjunto de testeo, sino que también logró la mejor *Precision* (73.7%) y el mejor *Recall* (61.4%) de manera simultánea. Este resultado es de suma importancia, porque indica que RF fue el modelo que mejor logró capturar los patrones subyacentes y verdaderamente distintivos de los contratos maliciosos a partir de los *opcodes*, de una manera que se tradujo eficazmente a datos nuevos y desconocidos. Logró el equilibrio más sólido entre identificar correctamente las amenazas y evitar acusaciones incorrectas.

La divergencia entre el rendimiento en VC y en *Test* fue particularmente instructiva en el caso de BRF y EEC. El EEC vio su *F1-Score* descender a 0.553, con una caída notable tanto en *Precision* como en *Recall* respecto a sus mejores valores en VC. La degradación fue aún más severa para BRF, cuyo *F1-Score* se redujo a 0.483. Aunque mantuvo un *Recall* razonable (56.8%), su *Precision* se desplomó a un 42.0%, el valor más bajo entre los modelos de ensamble en testeo. Este fenómeno es una manifestación clara de sobreajuste (*overfitting*).

Las intensas estrategias de submuestreo empleadas por BRF y EEC, si bien fueron efectivas para forzar la atención sobre la clase minoritaria durante el entrenamiento (resultando en un alto Recall en *Cross Validation*), hicieron que los modelos se especializaron en exceso. Aprendieron a memorizar patrones muy específicos de las particiones de entrenamiento, en lugar de las señales subyacentes y generalizables de malicia. Como resultado, su capacidad de predicción decayó significativamente al ver datos con distribución ligeramente diferente del conjunto de prueba, demostrando una falta de robustez. La Regresión Logística se mantuvo consistentemente como la opción menos performante.

5.3 Análisis Comparativo Integrado y Discusión de Limitaciones

Al comparar los resultados detallados de la validación cruzada (presentes en las *Tablas 3* y *4* y *Figuras 8,9* y *10*) con la evaluación final sobre el conjunto de *test* (*Tabla 5* y *Figura 8*), obtenemos la imagen completa, y esto nos revela algo interesante.
Se invirtió el orden de rendimiento: en la prueba final, el modelo *Random Forest* (RF) funcionó claramente mucho mejor que los modelos *Balanced Random Forest* (BRF) y *Ensemble Classifier* (EEC). Este cambio no es solo un detalle menor; realmente resalta lo fundamental que es probar los modelos con datos completamente nuevos, que no hayan visto antes. Esta prueba final nos muestra cómo funciona realmente el modelo en un escenario más realista.
Si nos hubiéramos fijado únicamente en las métricas de la validación cruzada, podríamos habernos hecho una idea equivocada. Esto es especialmente importante porque algunas técnicas que se usan al entrenar los modelos, como el submuestreo (*undersampling*) —que probablemente usan BRF y EEC para equilibrar los datos—, a veces pueden hacer que los modelos parezcan mejores durante la validación cruzada de lo que son en realidad al enfrentarse a datos nuevos. Confiar solo en esas métricas de validación cruzada podría llevarnos a elegir un modelo que parece prometedor al principio, pero que luego no da la talla cuando realmente importa.

Basándonos en toda la información, especialmente en los resultados de la *Tabla 5* y la *Figura 8,* el modelo *Random Forest* (RF) se establece como el que mejor funciona y el que recomendamos para esta tarea. Ha demostrado que aprende bien y luego aplica ese aprendizaje a datos nuevos, consiguiendo el mejor equilibrio entre acertar cuando dice que algo es malicioso (*Precision* del 73.7%) y encontrar una buena parte de los que realmente lo son (*Recall* del 61.4%). Esto lo convierte en la opción más fiable y equilibrada para usar en la

práctica. Aunque BRF y EEC parecían encontrar más casos maliciosos en la validación cruzada, su incapacidad para mantener ese nivel, sobre todo fallando más en sus predicciones en la prueba final, los hace menos interesantes para el uso real donde necesitamos confiar en las alertas.
Sin embargo, es importante ser realistas y reconocer las limitaciones que persisten, incluso con el modelo *Random Forest* seleccionado. Una cuestión clave es la diferencia de rendimiento entre cadenas, algo que vimos claramente en las *Figuras 5, 6 y 7*. Esto implica que el buen resultado general de *Random Forest* en la prueba (*Tabla 5*) es solo una media, su efectividad real puede ser mayor en algunas cadenas como Arbitrum y menor en otras como Avalanche o ETH. Esto sugiere que aplicar el mismo modelo general a todas partes podría no ser lo ideal, y quizás necesitemos ajustes específicos por cadena o incluso modelos distintos para optimizar los resultados localmente.
Otra consideración es que el mundo cambia, y los ataques también. El modelo se entrenó con datos de un momento específico, y a medida que los atacantes evolucionen sus tácticas, un modelo estático podría perder eficacia. La diferencia que ya observamos entre los resultados de validación cruzada y los de la prueba final nos da una pista de esta sensibilidad, indicando la necesidad de reentrenar o actualizar el modelo periódicamente.
Además, siempre existe el dilema entre *Precision* y *Recall*, especialmente en problemas como este donde los casos maliciosos son minoría. El punto de equilibrio que elegimos para RF (*Precision* 73.7%, Recall 61.4%), buscando optimizar la métrica F1, es solo una opción. Dependiendo del coste de los errores (¿es peor una falsa alarma o dejar pasar un contrato peligroso?), podríamos necesitar otro balance. Intentar detectar el 100% de los contratos maliciosos seguramente aumentaría mucho las falsas alarmas, y viceversa.
Finalmente, debemos recordar que el modelo solo mira una parte de la historia, basándose exclusivamente en los *opcodes* mediante TF-IDF. Es ciego a otra información potencialmente relevante como el código fuente, el historial de transacciones o interacciones fuera de la cadena. Ataques que no dejen una huella clara en la secuencia de opcodes podrían pasar desapercibidos.

5.4 Análisis de Atributos en TF-IDF

La discusión anterior se centró en la comparación del rendimiento de los algoritmos seleccionados. Sin embargo, una comprensión más profunda del éxito del modelo *Random Forest* requiere un análisis de los atributos que sustentan sus predicciones. Una de las decisiones metodológicas más importantes fue la utilización de n-gramas de opcodes, procesados mediante TF-IDF. Para este fin, se configuró la extracción de características para

considerar n-gramas de longitud variable, desde 1 hasta 4 (ngram_range=(1,4)). Este proceso implicó analizar no solo opcodes individuales (unigramas), sino también todas las secuencias contiguas de dos (bigramas), tres (trigramas) y cuatro opcodes (cuatrigramas) presentes en el corpus. De este gran universo de posibles secuencias, se seleccionaron únicamente las 30.000 más frecuentes del conjunto de entrenamiento. Esta selección constituyó el vocabulario final de atributos sobre el cual los modelos fueron entrenados.

En un problema con clases tan desbalanceadas, esta estrategia podría parecer contraintuitiva, ya que se podría suponer que los patrones característicos de los contratos maliciosos, al ser minoritarios, serían poco frecuentes y, por lo tanto, excluidos por este filtro de dimensionalidad.

Para investigar esta dinámica y comprender por qué estos atributos funcionan, se realizó un análisis post-hoc sobre las características aprendidas por el modelo.

Un análisis de la importancia de los atributos del modelo, presentado en la *Tabla 6* (Top 20 N-gramas más importantes para el modelo RF), nos muestra que en definitiva es una estrategia de detección multifacética. Es notable que el modelo no se enfoca en un único tipo de señal, sino que construye su decisión a partir de una visión holística que incluye la interacción con funciones estándar (como balanceOf y transfer), patrones de interacción con protocolos DeFi (como getReserves), y también la lógica interna de bajo nivel (como las secuencias de manipulación de la pila de los contratos).

N-grama	Importancia
70a08231	0.013944
addr	0.011438
add and dup3 add	0.011202
dup4 sub	0.009487
push4 ffffffff and push1	0.009412
swap1 push4 70a08231	0.007554
mload dup2 push4 ffffffff	0.007490
dup7 gas	0.007300

and push20 addr	0.007157
push1 not push1	0.006921
70a08231 push1 shl	0.006850
push1 add	0.006759
dup2 dup7 gas	0.006283
push2 unknown codecopy	0.006132
returndatasize push1 not push1	0.005982
dup3 add and	0.005882
push4 a9059cbb	0.005828
a9059cbb	0.005352
push4 0dfe1681	0.005241
push20 addr and push4	0.005216

Tabla 6: Top 20 N-gramas más importantes para el modelo RF (según Feature Importance). Nota: 70a08231 = balance Of(address)

Para una visión más granular, un análisis diferencial de los n-gramas más distintivos de cada clase resulta esclarecedor. La *Tabla 6.1* y la *Tabla 6.2* ilustran el caso del modelo *Random Forest* y muestran los patrones más representativos de cada categoría. La efectividad del modelo, a pesar de la selección de atributos frecuentes, puede explicarse a través de tres mecanismos complementarios que operan en conjunto.

N-grama	**Score TF-IDF Promedio (Malicioso)**	**Score TF-IDF Promedio (Legítimo)**	**Ratio de Malicia**
3c8a7d8d	0.000917	0.000000	917.55

push4 3c8a7d8d	0.000917	0.000000	917.55
a34123a7	0.000915	0.000000	916.16
push4 a34123a7	0.000915	0.000000	916.16
mstore push1 swap2 pop	0.001523	0.000001	769.03
unknown swap2 dup5	0.000914	0.000000	738.83
push1 swap1 swap3 swap2	0.000814	0.000000	718.08
swap1 push1 swap1 swap3	0.000800	0.000000	705.54
cd51b305	0.000686	0.000000	686.72
push4 cd51b305	0.000686	0.000000	686.72

Tabla 6.1: Top 10 N-gramas más distintivos de contratos maliciosos (ordenados por Ratio de Malicia descendente).

N-grama	**Score TF-IDF Promedio (Malicioso)**	**Score TF-IDF Promedio (Legítimo)**	**Ratio de Malicia**
push1 add swap1 swap2	0.000000	0.000873	0.0011
add swap1 swap2 swap1	0.000000	0.000860	0.0012
swap1 sub add swap1	0.000000	0.000259	0.0038
add swap1 log3	0.000002	0.000634	0.0041
sha3 push1 add sload	0.000000	0.000285	0.0047
add swap1 log3 pop	0.000001	0.000517	0.0048

swap2 add swap1 log3	0.000001	0.000385	0.0053
swap1 sha3 push1 add	0.000000	0.000202	0.0053
and iszero mul add	0.000000	0.000148	0.0067
iszero mul add swap1	0.000000	0.000146	0.0068

Tabla 6.2: Top 10 N-gramas más distintivos de contratos legítimos (ordenados por Ratio de Malicia ascendente).

El primer mecanismo es la detección por ausencia de la "normalidad". El modelo se convierte en un experto en reconocer los atributos más importantes de un contrato legítimo, donde patrones asociados al uso correcto de estructuras estándar, como el acceso a mapeos con sload o la emisión de eventos con log3, son fuertes indicadores de legitimidad. En consecuencia, un contrato malicioso puede ser detectado no solo por lo que contiene, sino por la ausencia de estas secuencias de código esperadas. Su estructura se desvía de la norma aprendida, lo que constituye una señal de alerta para el clasificador.
De manera complementaria, el segundo mecanismo es la identificación de patrones de ataque altamente distintivos. El análisis demuestra que la suposición de que todos los patrones maliciosos son filtrados es incorrecta. Ciertas secuencias, aunque raras en el corpus general, son tan exclusivas de la clase maliciosa que su score TF-IDF las eleva a un nivel de importancia crítico. La presencia de selectores de función como 3c8a7d8d, generalmente vinculada a la función *mint(address,int24,int24,uint128,bytes)* utilizada por administradores de manera oculta típicamente en los *honeypots*, actúan como "banderas rojas" que el modelo aprende a identificar con precisión.
Finalmente, la robustez del modelo *Random Forest* no reside en la dependencia de un único atributo, sino en su evaluación combinatoria y holística. Su fortaleza radica en la capacidad para evaluar cientos de estas pequeñas señales de manera conjunta. El modelo no se limita a una simple pregunta binaria sobre la presencia de un opcode malicioso, sino que evalúa un perfil completo. Es esta visión integrada, que equilibra múltiples formas de evidencia, la que permite una clasificación robusta y generalizable.
La naturaleza de estos atributos predictivos ofrece, además, una visión sobre las tácticas de los atacantes. La prominencia de selectores de función específicos de estafas sugiere que una porción significativa de los fraudes en el ecosistema no se basa en la explotación de vulnerabilidades complejas y

novedosas, sino en la reutilización de plantillas de código y "kits de desarrollo de estafas" que dejan una huella digital reconocible, como por ejemplo los *Inferno Drainners* que se comercializan de manera ilegal, compartiendo las ganancias con sus creadores.
En este contexto, la decisión de limitar el espacio de características a los 30.000 n-gramas más frecuentes se revela no como una limitación, sino como una decisión metodológica crucial para promover la generalización y evitar el sobreajuste. Un vocabulario sin restricciones, que podría incluir cientos de miles de n-gramas únicos, introduciría una dimensionalidad masiva. En ese escenario, aumenta significativamente el riesgo de que el modelo aprenda a asociar secuencias presentes en un único o en muy pocos contratos del conjunto de entrenamiento con la clase maliciosa. Estas reglas, basadas en patrones no representativos, carecerían de poder predictivo ante datos no vistos.
Por lo tanto, el límite de 30.000 características actúa como una forma de regularización implícita, enfocando al modelo en patrones que son más estables y prevalentes en el ecosistema. Al filtrar el "ruido" de los n-gramas extremadamente raros, que con mayor probabilidad representan artefactos de implementación específicos que patrones de ataque generalizables. De esta manera, se fuerza al modelo a construir su conocimiento sobre señales más robustas. El sólido rendimiento del modelo en el conjunto de prueba valida empíricamente esta estrategia, demostrando que su capacidad de generalización se logró no a pesar de esta decisión, sino en gran medida, gracias a ella.

5.5 Implicaciones Prácticas y Aplicabilidad del Modelo Seleccionado (RF)

Viendo los números concretos que obtuvo *Random Forest* en la prueba final, un *F1-Score* de 0.670, una *Precision* del 73.7% y un *Recall* del 61.4% (como muestra la *Tabla 5*), podemos hacernos una idea clara de lo útil que podría ser en la práctica. Si integráramos este modelo en una herramienta de análisis, por ejemplo, a través de una API, los usuarios podrían esperar que:
Primero, identifique y alerte sobre unos 61 de cada 100 contratos maliciosos reales que revise. Segundo, que cuando alerte sobre un contrato como malicioso, acierte unas 74 de cada 100 veces.
Este nivel de rendimiento ya supone una ayuda automática importante, sobre todo porque revisar manualmente todos los contratos es inviable. Podría funcionar bien como un primer filtro o para dar una puntuación de riesgo a los contratos. Sin embargo, también tenemos que pensar en los errores que

comete: se le escapan casi 39 de cada 100 contratos maliciosos (un 38.6% de falsos negativos), y su *Precision* del 73.7% implica que aproximadamente 26 de cada 100 alertas generadas serán falsas alarmas (Falsos Positivos). Es fundamental subrayar que estas falsas alarmas representan una fracción mínima del total de contratos legítimos (una Tasa de Falsos Positivos de solo 0.6%), por lo que el modelo no interfiere significativamente con el uso de aplicaciones válidas. No obstante, estos números nos dicen que no podemos confiar ciegamente en el modelo ni dejarlo decidir solo.
Su papel más adecuado es el de apoyo a la decisión. Sus predicciones pueden informar a los usuarios o a otros sistemas de seguridad, pero seguramente se necesitará una verificación extra, sobre todo si salta una alarma o en casos de alto riesgo. Una ventaja es que, al implementar el modelo final, podríamos ajustar su "sensibilidad" (el umbral de decisión) para que se adapte mejor a cuánto riesgo estamos dispuestos a asumir en cada caso (si preferimos minimizar los contratos malos que se escapan o las falsas alarmas). Esta flexibilidad se ilustra visualmente en la curva *Precision-Recall* del modelo.

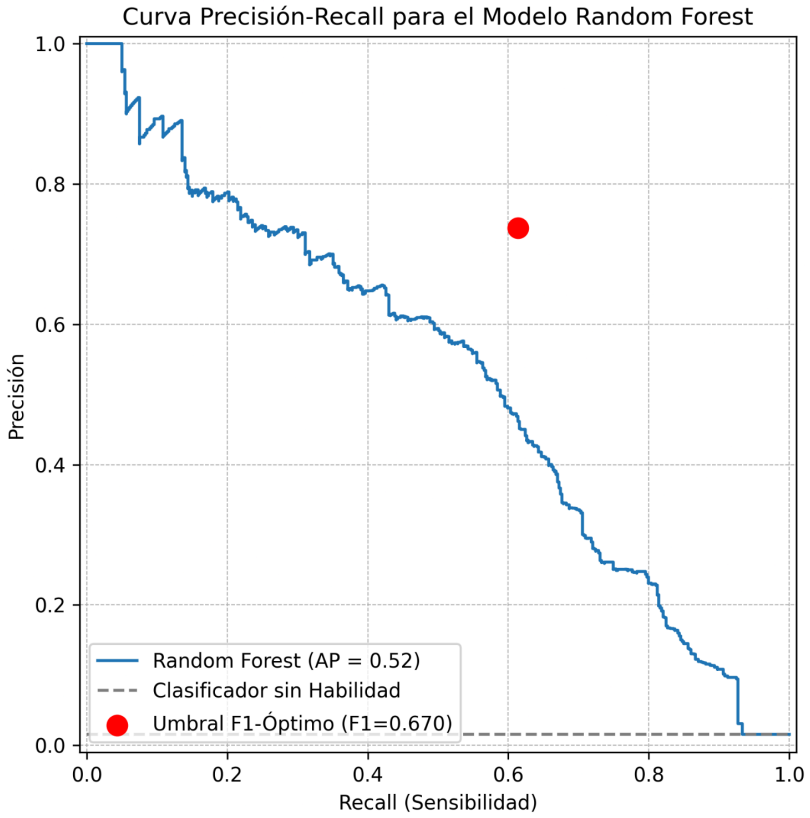

Figura 9 : Curva Precision-Recall para el modelo random Forest en Test

La *Figura 9* demuestra este compromiso operativo. El punto de equilibrio F1-óptimo es solo una de las posibles configuraciones. Un usuario podría elegir un umbral más estricto para priorizar una alta *Precision*, a costa de un menor *Recall*, o viceversa, adaptando el modelo a diferentes políticas de riesgo.
Algunas propuestas para poder mitigar este impacto pueden ser agregar información extra sobre las *addresses* o personas que deployean estos contratos. Gracias a que la *blockchain* es una base de datos transparente e inviolable (por lo menos en el registro y manipulación de la información) podríamos ver quien deployó ese contrato y obtener más información sobre los atacantes. Cuanto tiempo antes de deployar el contrato fue creada su *address*, por qué medio se fondearon, quien los fondeo y cuanto dinero se usó para fondearlos, cuantas transacciones hizo antes de fondear el contrato. Toda esta información podría ser relevante a la hora tomar la decisión si un contrato con un *Risk Score* alto es o no un contrato malicioso.
Existen protocolos que permiten mezclar activos y fondear nuevas *addresses* o contratos para que sea prácticamente imposible saber de donde salieron los fondos, lo único que sí se sabe es que salieron de ese protocolo. Algunos protocolos conocidos son Tornado Cash o FixFloat (Comunicado de Prensa OFAC).

Algunas propuestas para poder mitigar este impacto pueden ser agregar información extra sobre las *addresses* o personas que deployean estos contratos. Gracias a que la *blockchain* es una base de datos transparente e inviolable (por lo menos en el registro y manipulación de la información) podríamos ver quien deployó ese contrato y obtener más información sobre los atacantes. Cuanto tiempo antes de deployar el contrato fue creada su *address*, por qué medio se fondearon, quien los fondeo y cuanto dinero se usó para fondearlos, cuantas transacciones hizo antes de fondear el contrato. Toda esta información podría ser relevante a la hora tomar la decisión si un contrato con un *Risk Score* alto es o no un contrato malicioso.

6. Conclusiones

El presente capítulo sintetiza los resultados más relevantes obtenidos a lo largo de esta investigación orientada a la detección automatizada de contratos inteligentes maliciosos en entornos EVM mediante técnicas de aprendizaje automático. En las secciones que siguen, se exponen de manera estructurada los hallazgos centrales, las implicancias prácticas y teóricas derivadas del estudio, las limitaciones identificadas durante el proceso y, finalmente, se proponen líneas de investigación futuras con el fin de profundizar y ampliar este campo de conocimiento.

6.1 Resumen de Hallazgos y Contribuciones Clave

La investigación se centró en el desafío de identificar contratos inteligentes maliciosos en redes EVM, teniendo en cuenta que generalmente los usuarios no tienen acceso al código fuente y que los contratos maliciosos son mucho menos comunes que los no maliciosos. Comparamos el rendimiento de cuatro modelos de *Machine Learning*: *Random Forest* (RF), *Easy Ensemble Classifier* (EEC), *Balanced Random Forest* (BRF) y Regresión Logística (LR) entrenados sobre vectores TF-IDF derivados de secuencias de *opcodes*, lo cual constituye una representación abstracta pero accesible del comportamiento funcional de los contratos.

El análisis experimental reveló que el modelo *Random Forest* estándar se posicionó como la alternativa más robusta en términos de capacidad de generalización frente a datos no vistos. Con un *F1-Score* de 0.670, una *Precision* del 73.7% y un *Recall* del 61.4% sobre el conjunto de prueba independiente, RF superó en desempeño final tanto a BRF como a EEC, modelos específicamente diseñados para escenarios con clases desbalanceadas. Estos últimos, si bien mostraron métricas destacadas en la etapa de validación cruzada (*F1-Scores* de 0.699 y 0.664, respectivamente), presentaron una caída significativa en la evaluación final, sugiriendo un mayor grado de sobreajuste o menor capacidad de generalización en contextos reales.

Otro hallazgo importante fue la notable diferencia de rendimiento entre la validación cruzada y la prueba final para BRF y EEC. Esta caída nos demuestra los riesgos de usar estas técnicas de balanceo que pueden sobre ajustarse o ser menos estables y recalca la importancia fundamental de validar siempre con un conjunto de testeo independiente para elegir el modelo más fiable, sobre todo en problemas con datos desbalanceados.

También confirmamos que el rendimiento varía bastante entre las distintas *blockchains* analizadas (como se ve en las *Figuras 5, 6 y 7* y la *Tabla 4*). Que los modelos funcionen de forma diferente en redes como Arbitrum o Avalanche indica que el poder predictivo de malicia en los *opcodes* no son universales y dependen del entorno de cada red EVM. Esto hace difícil pensar en un único modelo perfecto para todas las cadenas sin adaptaciones.
Finalmente, la idea de implementar esto en la práctica mediante una API REST , usando el modelo RF, muestra que es factible aplicar estos resultados en herramientas de seguridad reales para el ecosistema *blockchain*.
En conjunto, las principales aportaciones de esta tesis pueden resumirse en:

1. Una evaluación comparativa que posiciona a *Random Forest* como el modelo con mejor desempeño global para la tarea propuesta.
2. La identificación de variabilidad en el rendimiento según el tipo de *blockchain*, lo que refuerza la necesidad de modelos únicos por *blockchain*.
3. La validación de la importancia crítica del testeo con datos independientes en problemas con clases desbalanceadas, como garantía de robustez metodológica.
4. El diseño e implementación de una API funcional, que traduce los resultados académicos en una solución práctica escalable y aplicable.

6.2 Implicaciones Prácticas y Teóricas

En la práctica, el resultado más directo es haber identificado a *Random Forest* como la herramienta más eficaz y fiable de las evaluadas. Su rendimiento medido en la prueba lo convierte en un candidato viable para integrarse en sistemas de seguridad, ofreciendo detección proactiva de contratos maliciosos. La API propuesta facilita su uso en plataformas DeFi, billeteras o herramientas de auditoría. Sin embargo, sus tasas de error significan que su mejor rol es como un sistema de alerta o evaluación de riesgo, apoyando la toma de decisiones dentro de una estrategia de seguridad más amplia. Poder ajustar su umbral de decisión es clave para adaptarlo a la tolerancia al riesgo de cada caso.
Desde el punto de vista teórico, la investigación aporta al estudio del manejo de datos desbalanceados, mostrando un caso donde un algoritmo estándar superó en generalización a técnicas especializadas. Esto sugiere que la efectividad de los métodos de balanceo depende del contexto y no es una garantía universal. Además, el análisis entre cadenas resalta la importancia del dominio específico en *Machine Learning*, mostrando cómo el contexto (la

blockchain) afecta significativamente el rendimiento. Este trabajo también se suma al conocimiento existente sobre la aplicación de *Machine Learning* al análisis de *bytecode*/*opcodes* para la seguridad *blockchain*.
Desde un punto de vista práctico, el modelo *Random Forest* propuesto se posiciona como un complemento valioso a las herramientas de análisis estático existentes como Slither o Mythril. Mientras que las herramientas *SAST* (análisis estático) son expertas en detectar vulnerabilidades de seguridad conocidas en el código, el enfoque de *Machine Learning* es capaz de identificar patrones de intencionalidad maliciosa que pueden no violar ninguna regla de seguridad formal, pero que están fuertemente correlacionados con estafas y fraudes. Su capacidad para operar a gran escala y en tiempo real sobre el bytecode lo hace ideal para sistemas de alerta temprana o para el filtrado inicial de grandes volúmenes de contratos.

6.3 Limitaciones del Estudio

Al interpretar los resultados, es importante tener en cuenta varias limitaciones. Primero, está la cuestión de la actualización con el tiempo o deriva conceptual; el modelo RF se entrenó con datos de un momento específico, y como las amenazas evolucionan, su rendimiento podría decaer, necesitando mecanismos para mantenerse al día. Segundo, existe una variación en el rendimiento entre cadenas que no se resolvió óptimamente; aunque RF fue el mejor globalmente, para aplicaciones que requieran máxima eficacia en una cadena donde fue más débil, podrían necesitarse modelos específicos. Tercero, la capacidad predictiva depende intrínsecamente de los datos de entrenamiento; ataques completamente novedosos podrían no ser detectados si difieren mucho de lo visto previamente. Finalmente, nuestro enfoque se basó exclusivamente en opcodes procesados con TF-IDF; incluir otras fuentes de datos, como análisis estructural del código o datos de transacciones, podría potencialmente mejorar los resultados.

6.4 Recomendaciones para Investigaciones Futuras

Basándonos en nuestros hallazgos y las limitaciones identificadas, se proponen varias direcciones para futuras investigaciones. Sería valioso desarrollar modelos que aprendan continuamente, investigando formas para que se adapten a las nuevas amenazas, ya sea mediante reentrenamiento periódico con algún proceso de *Machine Learning Operations*, aprendizaje sobre la marcha o detectando activamente cambios en los patrones. También se debería explorar la creación de modelos más específicos por contexto, ya sea

para cada *blockchain*, aunque su implementación sea más tediosa ya que habría que configurar las llamadas a la API por *blockchain* y no una general. Además de entrenar n modelos por según la cantidad de *blockchains* que se quieren cubrir. Otra línea interesante es evaluar representaciones de datos más avanzadas y modelos profundos, yendo más allá del TF-IDF para usar grafos de control de flujo, *embeddings* semánticos y arquitecturas de *Deep Learning*.
Además, se podría experimentar con la combinación de diferentes tipos de modelos mediante técnicas de ensamble avanzadas como *stacking* o *blending*, buscando un mejor rendimiento global. Es fundamental aplicar técnicas de Inteligencia Artificial Explicable (XAI) como SHAP o LIME para entender por qué los modelos toman sus decisiones, lo que puede aumentar la confianza, guiar mejoras y descubrir nuevas tácticas. Sería necesario realizar pruebas a largo plazo en entornos reales, colaborando con plataformas *blockchain* para evaluar el rendimiento práctico del modelo frente a amenazas reales.
En resumen, este trabajo ha identificado al modelo *Random Forest* como una solución sólida y la más generalizable de las estudiadas para detectar automáticamente contratos inteligentes maliciosos basándose en *opcodes*. Ha aportado una base empírica y una propuesta de implementación. No obstante, la seguridad en *blockchain* es un desafío constante. Las líneas futuras propuestas buscan superar las limitaciones actuales y seguir mejorando la *Precision,* adaptabilidad y fiabilidad de estas importantes herramientas de defensa contra atacantes.

7. Propuesta de Implementación Práctica: API REST

Para traducir los hallazgos de esta investigación en una herramienta funcional y de impacto real, se propone una implementación práctica del modelo seleccionado a través de una API REST (Application Programming Interface). Este enfoque permitiría que diversas plataformas del ecosistema *blockchain,* como billeteras virtuales, exploradores de bloques o exchanges descentralizados puedan integrar fácilmente la capacidad de detección de contratos maliciosos, ofreciendo una capa de seguridad proactiva a sus usuarios.

7.1 Arquitectura y Flujo de la API

El diseño conceptual de la API se centra en la simplicidad, la eficiencia y el uso de servicios e infraestructura existentes para garantizar la escalabilidad. El modelo, junto con el vectorizador TF-IDF, residiría en un servidor público,

expuesto a través de un endpoint con una URL pública para recibir las peticiones de análisis.

El flujo operativo sería el siguiente:

1. Recepción de la Petición: La API recibe una solicitud POST que contiene la dirección del contrato inteligente y el identificador de la cadena (chainId) donde fue desplegado.
2. Caché de Resultados: Antes de cualquier procesamiento, el sistema verificaría si la dirección del contrato ya ha sido analizada previamente consultando una base de datos interna o un sistema de caché. Si se encuentra un resultado, se devuelve inmediatamente, evitando el reprocesamiento y minimizando la latencia.
3. Obtención del Bytecode: Si el contrato es nuevo, el servidor utilizaría una clave de API del servicio Etherscan API V2 para realizar una llamada al endpoint correspondiente. Esta llamada, utilizando el chainId y la dirección del contrato, permite obtener de manera fiable el bytecode del contrato desplegado.
4. Conversión a Opcodes: Una vez obtenido el *bytecode* (representado como una cadena hexadecimal), el servidor utilizaría una librería especializada de Python como pyevmasm para desensamblarlo y convertirlo en su representación mnemotécnica legible: la secuencia de *opcodes*.
5. Predicción del Modelo: A continuación, se carga el modelo *Random Forest* serializado (.joblib) y el vectorizador TF-IDF. Se aplica la transformación TF-IDF a la nueva secuencia de *opcodes* y se alimenta al modelo para obtener un *Risk Score* probabilístico.
6. Almacenamiento y Respuesta: El resultado del análisis (dirección, score, clasificación, etc.) se almacena en la base de datos de caché para futuras consultas. Finalmente, la API devuelve una respuesta en formato JSON al cliente con el resultado del análisis.

7.2 Ejemplos de Uso en el Ecosistema

La utilidad práctica de la API se manifiesta al integrarla en las herramientas que usuarios y desarrolladores usan a diario. Se pueden distinguir dos modalidades de uso principales: el análisis individual y en tiempo real para la protección de transacciones, y el análisis masivo y asíncrono para la curación de datos a escala.

El primer caso de uso se centra en la protección inmediata del usuario en el punto más crítico de la interacción con la *blockchain*. Un escenario representativo es su integración en una billetera virtual. Cuando un usuario, atraído por la promesa de altos rendimientos o un nuevo *airdrop*, se dispone a interactuar con un nuevo y desconocido protocolo de finanzas descentralizadas, la billetera actúa como un intermediario de seguridad. Al momento en que el usuario intenta aprobar una transacción que involucra al nuevo contrato, la billetera intercepta la acción. Antes de presentar la ventana de confirmación, envía de forma transparente y automática la dirección del contrato a la API para un análisis instantáneo.

Si el modelo devuelve un score de riesgo alto, indicando una alta probabilidad de que el contrato sea malicioso (por ejemplo, un *honeypot*), la experiencia del usuario se interrumpe deliberadamente. En lugar de la típica ventana de confirmación de gas, la billetera despliega una advertencia de seguridad prominente, informando al usuario del riesgo detectado y recomendando encarecidamente rechazar la transacción. De esta manera, la API empodera al usuario con información crítica en tiempo real, permitiéndole evitar una pérdida financiera segura sin necesidad de poseer conocimientos técnicos de auditoría de contratos. Este flujo de trabajo transforma a la billetera de un simple gestor de claves a un sistema de seguridad proactivo. Se puede ver un ejemplo del flujograma de este caso en la *Figura 10.*

El segundo caso de uso aborda el desafío de la escala. Plataformas como los agregadores de datos de mercado, por ejemplo CoinGecko, necesitan procesar miles de nuevos tokens ERC-20 que se crean diariamente para mantener sus listados actualizados. La revisión manual de cada nuevo contrato es operativamente inviable. En este escenario, la API se integra en el backend de la plataforma como un primer filtro automatizado en el flujo de trabajo de indexación.

Un servicio de monitoreo detecta la creación de nuevos contratos en la red y, por cada uno, invoca a la API para obtener un score de riesgo. El sistema de la plataforma utiliza esta respuesta para tomar una decisión automatizada. Si el score es bajo, el contrato se considera seguro y el proceso de indexación continúa: se obtienen sus metadatos y se lista públicamente. Por el contrario, si el score de riesgo supera un umbral predefinido, el contrato es automáticamente descartado o derivado a una cola de revisión manual para un análisis más profundo por parte del equipo de seguridad.

Este enfoque de análisis masivo permite a la plataforma mantener un equilibrio entre la velocidad de listado y la seguridad del ecosistema. Se logra curar el contenido a gran escala, protegiendo a la comunidad de la abrumadora mayoría de tokens fraudulentos y manteniendo la confianza en la

plataforma como una fuente de datos fiable. Su flujograma se puede ver en la *Figura 11.*

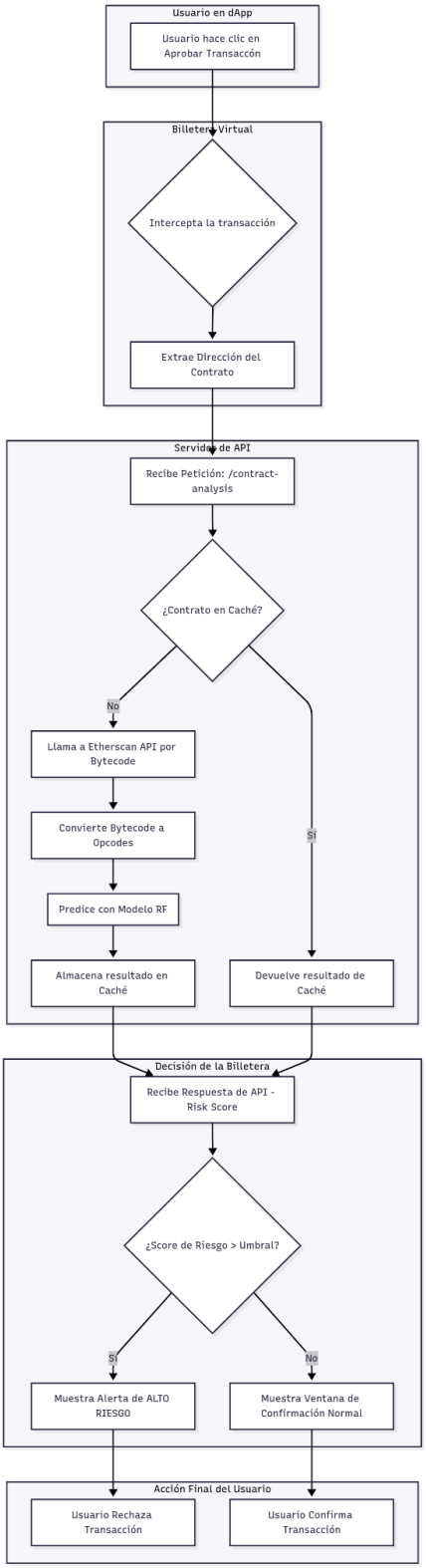

Figura 10: Ejemplo billetera virtual

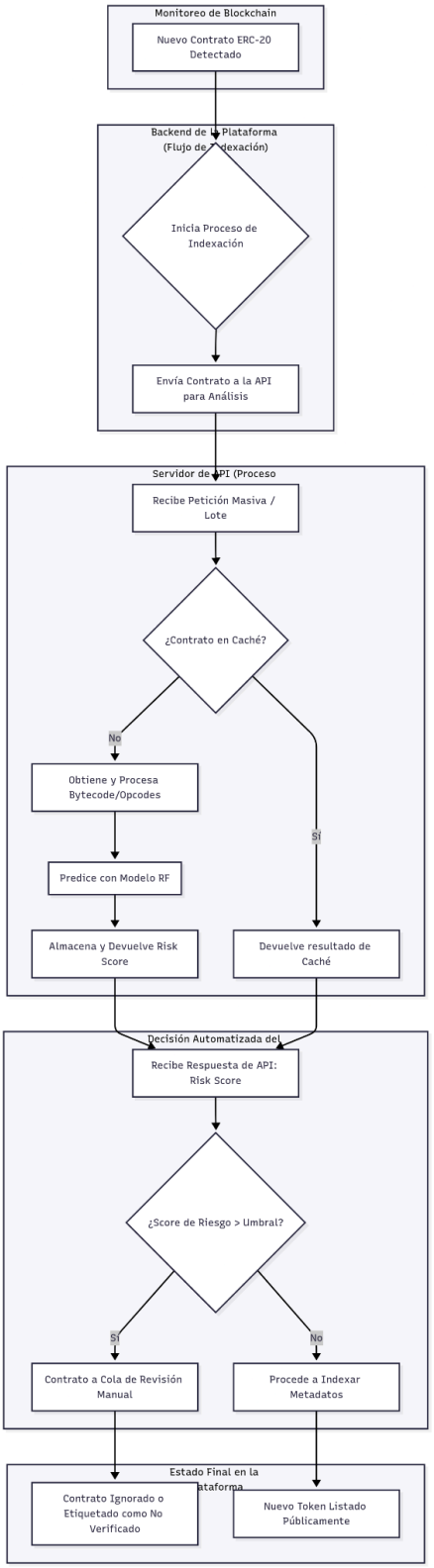

Figura 11: Ejemplo plataforma de tokens

Referencias

Antonopoulos, A. M., & Wood, G. (2018). Mastering Ethereum: Building smart contracts and DApps. O'Reilly Media.

Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A survey of attacks on Ethereum smart contracts (SoK). In Principles of Security and Trust (pp. 164–186). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-54455-6_8

Buterin, V. (2014). A next-generation smart contract and decentralized application platform. Ethereum White Paper. https://ethereum.org/en/whitepaper/

Chen, W., Zheng, Z., Cui, J., Chen, E., Xu, X., & Li, Z. (2020). Detecting ponzi schemes on ethereum: Towards healthier *blockchain* ecology. IEEE Access, 8, 29686-29698. https://doi.org/10.1109/ACCESS.2020.2971968

Entriken, W., Shirley, D., Evans, J., & Sachs, N. (2018). EIP-721: Non-fungible token standard. Ethereum Improvement Proposals, No. 721. https://eips.ethereum.org/EIPS/eip-721

Ethereum Foundation. (n.d.-a). Layer 2. Retrieved February 27, 2024, from https://ethereum.org/en/layer-2/

Ethereum Foundation. (n.d.-b). Proof-of-stake (PoS). Retrieved February 27, 2024, from https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/

Feist, J., Grieco, G., & Groz, B. (2019). Slither: A static analysis framework for smart contracts. In 2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&P) (pp. 356-363). IEEE. https://doi.org/10.1109/EuroSPW.2019.00049

Luu, L., Chu, D. H., Olickel, H., Saxena, P., & Hobor, A. (2016). Making smart contracts smarter. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (pp. 254–269). ACM. https://doi.org/10.1145/2976749.2978309

Lyu, X., Zhang, H., Wang, L., You, Z., & Mu, Y. (2020). A survey on the security and privacy of *blockchain* based systems. Security and Communication Networks, 2020, Article 6723712. https://doi.org/10.1155/2020/6723712

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf

Ng, W. K., V K, C. P., Lee, S. S., & Yeo, T. S. (2021). An Empirical Study of Smart Contract Vulnerabilities and Malicious Contract Detection Using Machine Learning Methods. Applied Sciences, 11(16), 7587. https://doi.org/10.3390/app11167587

Radomski, W., Cooke, A., Gudkov, P., & Operators, E. (2018). EIP-1155: Multi token standard. Ethereum Improvement Proposals, No. 1155. https://eips.ethereum.org/EIPS/eip-1155
Szabo, N. (1997). Formalizing and securing relationships on public networks. First Monday, 2(9). https://doi.org/10.5210/fm.v2i9.548
Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger (Ethereum Yellow Paper). Ethereum Project. https://ethereum.github.io/yellowpaper/paper.pdf
Xu, X., Wang, X., Gao, Z., Zhang, C., & Chen, X. (2019). A survey of *blockchain* technology: Architecture, consensus, and future trends. In 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS) (pp. 680-687). IEEE. https://doi.org/10.1109/ICSESS47205.2019.9040806
Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer.
Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence (Vol. 2, pp. 1137-1145). Morgan Kaufmann.
Tharwat, A. (2021). Classification assessment methods. Applied Computing and Informatics, 17(1), 168-192. https://doi.org/10.1016/j.aci.2018.08.003
U.S. Department of the Treasury. (2022, August 8). U.S. Treasury sanctions notorious virtual currency mixer Tornado Cash [Press release]. https://home.treasury.gov/news/press-releases/jy0916
Manning, C. D., Raghavan, P., & Schütze, H. (2008). Introduction to information retrieval. Cambridge University Press.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830.
Salton, G., & McGill, M. J. (1983). Introduction to modern information retrieval. McGraw-Hill.

Apéndice A. Composición del *Dataset*

Tabla A: Composición del *Dataset*

Dataset	Observaciones	No maliciosos	Maliciosos	%
Training	91,532	88,823	2,709	2.96%
Test	30,596	30,117	479	1.57%
Total	122,128	118,940	3,188	2.61%

Resumen de las divisiones del conjunto de datos utilizadas para el entrenamiento y la prueba del modelo, que muestra la distribución entre contratos maliciosos y no maliciosos.

Apéndice B. Resultados *Random Forest*

Tabla 7: *Performance Random Forest Cross Validation* (VC)

Dataset	Mejor F1	Precision	Recall	Threshold (T)
General (VC)	0.599	0.645	0.560	0.85
eth	0.414	0.557	0.329	0.86
bsc	0.728	0.616	0.890	0.50
avalanche	0.188	0.115	0.500	0.50
arbitrum	0.905	0.939	0.873	0.95
polygon	0.474	0.337	0.800	0.50
optimism	0.522	0.522	0.522	0.98
ftm	0.400	0.500	0.333	0.50

Métricas de rendimiento de Random Forest obtenidas durante la validación cruzada en el conjunto de entrenamiento. "General" representa las métricas agregadas de todas las chains. "Por chain" muestra la mejor puntuación F1 obtenida para cada cadena individual. Las métricas reportadas corresponden al umbral que arroja la puntuación F1 más alta para ese subconjunto.

Tabla 8: *Performance Random Forest* Test (Mejor *F1-Score*)

Dataset	Mejor F1	Precision	Recall	Threshold (T)
General (Test)	0.670	0.737	0.614	0.50

Métricas de rendimiento de Random Forest en el conjunto de prueba no observado ("conjunto de datos restante"). Las métricas reportadas corresponden al umbral que produce la puntuación F1 más alta.

Tabla 9: *Performance Random Forest* VC (eth) por Umbral

Threshold (T)	*F1-Score*	*Precision*	*Recall*
0.50	0.210	0.129	0.574
0.51	0.209	0.128	0.570
0.52	0.207	0.127	0.562
0.53	0.202	0.124	0.546
0.54	0.201	0.124	0.544
0.55	0.200	0.123	0.538
0.56	0.198	0.122	0.533
0.57	0.195	0.120	0.519
0.58	0.193	0.119	0.510
0.59	0.190	0.117	0.501
0.60	0.190	0.118	0.495
0.61	0.188	0.117	0.486

0.62	0.188	0.117	0.482
0.63	0.188	0.117	0.475
0.64	0.188	0.118	0.472
0.65	0.189	0.118	0.466
0.66	0.191	0.120	0.463
0.67	0.190	0.120	0.456
0.68	0.191	0.122	0.450
0.69	0.208	0.136	0.445
0.70	0.233	0.158	0.439
0.71	0.252	0.177	0.433
0.72	0.273	0.200	0.428
0.73	0.278	0.207	0.422
0.74	0.278	0.209	0.415
0.75	0.275	0.208	0.405
0.76	0.276	0.211	0.399
0.77	0.286	0.224	0.393
0.78	0.289	0.231	0.386
0.79	0.291	0.236	0.380
0.80	0.299	0.250	0.373
0.81	0.395	0.432	0.363
0.82	0.396	0.450	0.354
0.83	0.404	0.490	0.343
0.84	0.407	0.508	0.340
0.85	0.412	0.539	0.334
0.86	0.414	0.557	0.329
0.87	0.404	0.574	0.311

0.88	0.399	0.595	0.300
0.89	0.397	0.611	0.294
0.90	0.401	0.630	0.294
0.91	0.386	0.629	0.279
0.92	0.379	0.633	0.271
0.93	0.360	0.623	0.253
0.94	0.346	0.619	0.240
0.95	0.351	0.756	0.229
0.96	0.336	0.788	0.213
0.97	0.315	0.791	0.197
0.98	0.260	0.781	0.156
0.99	0.191	0.812	0.108

**Métricas de rendimiento de Random Forest durante la validación cruzada (sólo cadena eth) para diferentes umbrales de decisión.*

Tabla 10: *Performance Random Forest* VC (bsc) por Umbral

Threshold (T)	*F1-Score*	*Precision*	*Recall*
0.50	0.728	0.616	0.890
0.51	0.727	0.616	0.887
0.52	0.720	0.612	0.874
0.53	0.716	0.610	0.866
0.54	0.711	0.607	0.857
0.55	0.707	0.605	0.849
0.56	0.703	0.603	0.842
0.57	0.692	0.599	0.819

0.58	0.688	0.600	0.807
0.59	0.684	0.599	0.796
0.60	0.680	0.602	0.782
0.61	0.675	0.599	0.772
0.62	0.673	0.600	0.766
0.63	0.669	0.602	0.752
0.64	0.663	0.600	0.741
0.65	0.657	0.602	0.725
0.66	0.647	0.597	0.706
0.67	0.635	0.590	0.686
0.68	0.629	0.587	0.677
0.69	0.620	0.583	0.662
0.70	0.616	0.581	0.654
0.71	0.599	0.571	0.629
0.72	0.593	0.568	0.619
0.73	0.592	0.573	0.612
0.74	0.589	0.575	0.604
0.75	0.584	0.574	0.595
0.76	0.574	0.569	0.578
0.77	0.564	0.567	0.560
0.78	0.570	0.584	0.557
0.79	0.564	0.581	0.548
0.80	0.559	0.579	0.540
0.81	0.546	0.574	0.521
0.82	0.534	0.569	0.502
0.83	0.525	0.567	0.489

0.84	0.519	0.567	0.478
0.85	0.508	0.565	0.461
0.86	0.491	0.557	0.440
0.87	0.465	0.544	0.406
0.88	0.446	0.532	0.384
0.89	0.442	0.533	0.377
0.90	0.414	0.518	0.344
0.91	0.403	0.523	0.327
0.92	0.391	0.528	0.311
0.93	0.376	0.548	0.286
0.94	0.361	0.561	0.266
0.95	0.338	0.584	0.237
0.96	0.312	0.581	0.213
0.97	0.289	0.620	0.189
0.98	0.264	0.675	0.164
0.99	0.228	0.821	0.132

**Métricas de rendimiento de Random Forest durante la validación cruzada (sólo cadena de bsc) para diferentes umbrales de decisión.*

Tabla 11: *Performance Random Forest* VC (avalanche) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.171	0.103	0.500
0.51	0.173	0.105	0.500
0.52	0.175	0.106	0.500
0.56	0.157	0.095	0.444

0.59	0.168	0.104	0.444
0.65	0.149	0.092	0.389
0.69	0.152	0.095	0.389
0.71	0.132	0.082	0.333
0.73	0.133	0.083	0.333
0.75	0.135	0.085	0.333
0.78	0.136	0.086	0.333
0.79	0.138	0.087	0.333
0.80	0.071	0.045	0.167
0.84	0.048	0.031	0.111
0.85	0.025	0.016	0.056
0.87	0.025	0.016	0.056
0.88	0.026	0.017	0.056
0.89	0.030	0.021	0.056
0.90	0.036	0.027	0.056
0.91	0.038	0.029	0.056

**Métricas de rendimiento de Random Forest durante la validación cruzada (sólo cadena de avalanchas) para diferentes umbrales de decisión.*

Tabla 12: : *Performance Random Forest* VC (arbitrum) por Umbal

Threshold	F1-Score	Precision	Recall
0.50	0.879	0.808	0.965
0.51	0.880	0.809	0.965
0.52	0.880	0.809	0.964
0.53	0.878	0.808	0.961
0.54	0.876	0.808	0.958

0.56	0.875	0.808	0.956
0.57	0.876	0.810	0.954
0.58	0.874	0.809	0.951
0.60	0.878	0.816	0.950
0.61	0.877	0.816	0.948
0.62	0.879	0.820	0.948
0.63	0.880	0.825	0.943
0.64	0.879	0.829	0.935
0.65	0.882	0.834	0.935
0.66	0.885	0.839	0.935
0.67	0.886	0.842	0.934
0.68	0.885	0.842	0.933
0.69	0.887	0.847	0.932
0.70	0.887	0.849	0.930
0.72	0.887	0.848	0.928
0.73	0.890	0.857	0.926
0.74	0.890	0.859	0.922
0.75	0.888	0.860	0.918
0.76	0.887	0.861	0.914
0.77	0.887	0.865	0.910
0.78	0.891	0.876	0.906
0.79	0.889	0.877	0.901
0.80	0.887	0.877	0.898
0.82	0.888	0.878	0.898
0.83	0.890	0.883	0.898
0.84	0.891	0.885	0.897

0.85	0.892	0.888	0.896
0.86	0.891	0.888	0.894
0.87	0.891	0.892	0.889
0.88	0.890	0.893	0.887
0.89	0.892	0.899	0.885
0.90	0.895	0.907	0.884
0.91	0.897	0.913	0.882
0.92	0.900	0.922	0.880
0.93	0.900	0.926	0.876
0.94	0.902	0.932	0.874
0.95	0.905	0.939	0.873
0.96	0.904	0.945	0.866
0.97	0.900	0.952	0.854
0.98	0.891	0.967	0.825
0.99	0.876	0.986	0.787

**Métricas de rendimiento de Random Forest durante la validación cruzada (sólo cadena de arbitrum) para diferentes umbrales de decisión. Nota: Solo se incluyen los umbrales informados.*

Tabla 13: *Performance Random Forest* VC (polygon) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.474	0.337	0.800
0.51	0.470	0.334	0.792
0.52	0.467	0.332	0.784
0.54	0.459	0.328	0.768
0.55	0.457	0.326	0.760
0.56	0.453	0.324	0.752

0.57	0.447	0.321	0.736
0.58	0.444	0.319	0.728
0.59	0.436	0.314	0.712
0.60	0.440	0.320	0.704
0.61	0.437	0.319	0.696
0.62	0.426	0.312	0.672
0.63	0.424	0.313	0.656
0.64	0.418	0.310	0.640
0.65	0.418	0.312	0.632
0.66	0.411	0.308	0.616
0.67	0.406	0.305	0.608
0.68	0.402	0.302	0.600
0.69	0.402	0.305	0.592
0.70	0.403	0.306	0.592
0.71	0.396	0.301	0.576
0.72	0.374	0.288	0.536
0.73	0.380	0.300	0.520
0.74	0.362	0.288	0.488
0.75	0.354	0.284	0.472
0.77	0.350	0.282	0.464
0.78	0.353	0.288	0.456
0.79	0.345	0.284	0.440
0.80	0.347	0.290	0.432
0.81	0.344	0.290	0.424
0.82	0.352	0.301	0.424
0.83	0.349	0.301	0.416

0.84	0.356	0.311	0.416
0.85	0.359	0.321	0.408
0.86	0.368	0.336	0.408
0.87	0.375	0.352	0.400
0.88	0.380	0.368	0.392
0.89	0.373	0.370	0.376
0.90	0.359	0.367	0.352
0.91	0.318	0.343	0.296
0.92	0.299	0.344	0.264
0.93	0.306	0.381	0.256
0.94	0.310	0.413	0.248
0.95	0.316	0.437	0.248
0.96	0.295	0.431	0.224
0.97	0.264	0.421	0.192
0.98	0.249	0.477	0.168
0.99	0.056	0.235	0.032

**Métricas de rendimiento de Random Forest durante la validación cruzada (sólo cadena de polygon) para diferentes umbrales de decisión.*

Tabla 14: *Performance Random Forest* VC (optimism) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.286	0.175	0.783
0.51	0.272	0.167	0.739
0.52	0.274	0.168	0.739
0.53	0.260	0.160	0.696
0.60	0.276	0.172	0.696

0.63	0.283	0.178	0.696
0.64	0.286	0.180	0.696
0.65	0.288	0.182	0.696
0.66	0.291	0.184	0.696
0.67	0.275	0.174	0.652
0.69	0.297	0.192	0.652
0.70	0.300	0.195	0.652
0.71	0.283	0.184	0.609
0.73	0.289	0.189	0.609
0.75	0.292	0.192	0.609
0.78	0.301	0.200	0.609
0.81	0.308	0.206	0.609
0.82	0.311	0.209	0.609
0.83	0.315	0.212	0.609
0.86	0.318	0.215	0.609
0.87	0.329	0.226	0.609
0.88	0.310	0.213	0.565
0.89	0.317	0.220	0.565
0.90	0.329	0.232	0.565
0.91	0.308	0.218	0.522
0.92	0.312	0.222	0.522
0.95	0.320	0.231	0.522
0.96	0.381	0.300	0.522
0.97	0.444	0.387	0.522
0.98	0.522	0.522	0.522
0.99	0.486	0.643	0.391

**Métricas de rendimiento de Random Forest durante la validación cruzada (sólo cadena de optimism) para diferentes umbrales de decisión. Nota: Solo se incluyen los umbrales informados.*

Tabla 15: *Performance Random Forest* VC (ftm) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.250	0.200	0.333
0.51	0.333	0.333	0.333

**Métricas de rendimiento de Random Forest durante la validación cruzada (sólo cadena de optimism) para diferentes umbrales de decisión.*

Tabla 16: *Performance Random Forest* VC (General) por Umbral

Threshold (T)	F1-Score	Precision	Recall
0.50	*0.452*	*0.315*	*0.795*
0.51	*0.451*	*0.315*	*0.793*
0.52	*0.448*	*0.314*	*0.786*
0.53	*0.445*	*0.312*	*0.777*
0.54	*0.443*	*0.310*	*0.772*
0.55	*0.441*	*0.310*	*0.767*
0.56	*0.439*	*0.309*	*0.762*
0.57	*0.437*	*0.308*	*0.751*
0.58	*0.434*	*0.307*	*0.743*
0.59	*0.432*	*0.306*	*0.736*
0.60	*0.434*	*0.308*	*0.731*
0.61	*0.432*	*0.308*	*0.724*

0.62	*0.432*	*0.309*	*0.719*
0.63	*0.432*	*0.310*	*0.711*
0.64	*0.431*	*0.311*	*0.704*
0.65	*0.432*	*0.313*	*0.697*
0.66	*0.433*	*0.316*	*0.691*
0.67	*0.432*	*0.316*	*0.682*
0.68	*0.434*	*0.319*	*0.677*
0.69	*0.454*	*0.343*	*0.671*
0.70	*0.480*	*0.375*	*0.666*
0.71	*0.495*	*0.398*	*0.656*
0.72	*0.513*	*0.423*	*0.650*
0.73	*0.519*	*0.434*	*0.645*
0.74	*0.519*	*0.437*	*0.638*
0.75	*0.517*	*0.438*	*0.629*
0.76	*0.516*	*0.441*	*0.622*
0.77	*0.523*	*0.455*	*0.614*
0.78	*0.529*	*0.468*	*0.609*
0.79	*0.530*	*0.473*	*0.602*
0.80	*0.536*	*0.487*	*0.595*
0.81	*0.591*	*0.596*	*0.587*
0.82	*0.592*	*0.606*	*0.579*
0.83	*0.596*	*0.623*	*0.571*

0.84	*0.598*	*0.632*	*0.567*
0.85	*0.599*	*0.645*	*0.560*
0.86	*0.598*	*0.652*	*0.552*
0.87	*0.592*	*0.660*	*0.536*
0.88	*0.588*	*0.668*	*0.525*
0.89	*0.589*	*0.680*	*0.520*
0.90	*0.587*	*0.692*	*0.510*
0.91	*0.581*	*0.699*	*0.497*
0.92	*0.579*	*0.714*	*0.487*
0.93	*0.573*	*0.726*	*0.474*
0.94	*0.569*	*0.737*	*0.463*
0.95	*0.574*	*0.787*	*0.452*
0.96	*0.567*	*0.808*	*0.437*
0.97	*0.558*	*0.832*	*0.419*
0.98	*0.536*	*0.867*	*0.388*
0.99	*0.501*	*0.924*	*0.343*

Métricas de rendimiento de Random Forest durante la validación cruzada (agregadas en todas las cadenas) para diferentes umbrales de decisión.

Tabla 17: *Performance Random Forest* Test (General) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.670	0.737	0.614
0.51	0.668	0.752	0.601

0.52	0.656	0.766	0.574
0.53	0.650	0.778	0.557
0.54	0.641	0.791	0.539
0.55	0.642	0.819	0.528
0.56	0.636	0.837	0.514
0.57	0.620	0.862	0.484
0.58	0.597	0.869	0.455
0.59	0.567	0.885	0.418
0.60	0.533	0.880	0.382
0.61	0.519	0.893	0.365
0.62	0.498	0.888	0.347
0.63	0.475	0.896	0.324
0.64	0.456	0.907	0.305
0.65	0.438	0.914	0.288
0.66	0.426	0.917	0.278
0.67	0.403	0.939	0.257
0.68	0.372	0.941	0.232
0.69	0.345	0.944	0.211
0.70	0.325	0.940	0.196
0.71	0.309	0.978	0.184
0.72	0.280	0.987	0.163
0.73	0.258	0.986	0.148
0.74	0.206	0.982	0.115
0.75	0.178	0.979	0.098
0.76	0.147	0.974	0.079
0.77	0.080	0.952	0.042

0.78	0.068	0.944	0.035
0.79	0.049	0.923	0.025
0.80	0.029	1.000	0.015
0.81	0.025	1.000	0.013
0.82	0.017	1.000	0.008
0.83	0.012	1.000	0.006

**Métricas de rendimiento de Random Forest en el conjunto de prueba (agregadas en todas las cadenas) para diferentes umbrales de decisión.*

Apéndice C. Resultados *Logistic Regression*

Tabla 18: *Performance Logistic Regression* VC (Mejor F1-Score) por Cadena

Dataset	Mejor F1	Precision	Recall	Threshold (T)
General (VC)	0.539	0.536	0.542	0.57
eth	0.199	0.221	0.180	0.57
bsc	0.621	0.612	0.630	0.50
avalanche	0.188	0.115	0.500	0.50
arbitrum	0.881	0.863	0.899	0.50
polygon	0.440	0.342	0.616	0.51
optimism	0.315	0.212	0.609	0.70
ftm	0.400	0.500	0.333	0.50

**Métricas de rendimiento de la regresión logística obtenidas durante la validación cruzada en el conjunto de entrenamiento. "General" representa las métricas agregadas de todas las*

cadenas. "Por cadena" muestra la mejor puntuación F1 obtenida para cada cadena individual.

Tabla 19: *Performance Logistic Regression* Test (Mejor F1-Score)

Dataset	Mejor F1	Precision	Recall	Threshold (T)
General (Test)	0.443	0.613	0.347	0.50

** Métricas de rendimiento de la regresión logística en el conjunto de prueba no observado ("conjunto de datos restante"). Las métricas reportadas corresponden al umbral que produce la puntuación F1 más alta.*

Tabla 20: *Performance Logistic Regression* VC (General) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.492	0.436	0.566
0.51	0.492	0.438	0.562
0.52	0.508	0.465	0.559
0.53	0.512	0.475	0.556
0.54	0.520	0.490	0.554
0.55	0.529	0.509	0.550
0.56	0.529	0.514	0.544
0.57	0.539	0.536	0.542
0.58	0.537	0.536	0.537
0.59	0.535	0.537	0.534
0.60	0.532	0.537	0.528
0.61	0.531	0.538	0.524
0.62	0.530	0.539	0.520

0.63	0.527	0.540	0.515
0.64	0.521	0.536	0.506
0.65	0.519	0.539	0.501
0.66	0.516	0.539	0.495
0.67	0.516	0.542	0.492
0.68	0.516	0.548	0.487
0.69	0.512	0.550	0.480
0.70	0.512	0.551	0.478
0.71	0.509	0.552	0.472
0.72	0.505	0.551	0.467
0.73	0.504	0.555	0.462
0.74	0.501	0.555	0.456
0.75	0.498	0.555	0.451
0.76	0.496	0.559	0.446
0.77	0.492	0.558	0.440
0.78	0.497	0.577	0.436
0.79	0.493	0.577	0.430
0.80	0.484	0.573	0.420
0.81	0.482	0.575	0.414
0.82	0.476	0.573	0.407
0.83	0.472	0.573	0.401
0.84	0.459	0.571	0.384
0.85	0.458	0.575	0.381
0.86	0.367	0.526	0.282
0.87	0.298	0.471	0.217
0.88	0.296	0.478	0.215

0.89	0.294	0.481	0.212
0.90	0.292	0.484	0.209
0.91	0.222	0.413	0.151
0.92	0.218	0.425	0.147
0.93	0.214	0.429	0.143
0.94	0.214	0.436	0.142
0.95	0.212	0.441	0.140
0.96	0.208	0.456	0.135
0.97	0.209	0.501	0.132
0.98	0.184	0.503	0.113
0.99	0.092	0.386	0.052

**Métricas de rendimiento de regresión logística durante la validación cruzada (agregadas en todas las cadenas) para diferentes umbrales de decisión.*

Tabla 21: *Performance Logistic Regression* VC (eth) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.153	0.126	0.196
0.51	0.154	0.128	0.195
0.52	0.168	0.148	0.194
0.53	0.171	0.155	0.190
0.54	0.179	0.169	0.189
0.55	0.187	0.188	0.186
0.56	0.188	0.194	0.182
0.57	0.199	0.221	0.180
0.58	0.196	0.220	0.177

0.59	0.197	0.223	0.177
0.60	0.196	0.223	0.175
0.61	0.195	0.224	0.172
0.62	0.192	0.223	0.169
0.63	0.192	0.224	0.168
0.64	0.183	0.216	0.159
0.65	0.177	0.212	0.152
0.66	0.177	0.215	0.151
0.67	0.177	0.217	0.150
0.68	0.174	0.216	0.146
0.69	0.173	0.218	0.144
0.70	0.173	0.219	0.143
0.71	0.167	0.216	0.136
0.72	0.163	0.213	0.132
0.73	0.155	0.207	0.124
0.74	0.152	0.206	0.121
0.75	0.151	0.207	0.119
0.76	0.150	0.210	0.117
0.77	0.144	0.205	0.111
0.78	0.149	0.234	0.109
0.79	0.137	0.220	0.100
0.80	0.129	0.210	0.093
0.81	0.120	0.201	0.085
0.82	0.112	0.191	0.079
0.83	0.110	0.191	0.077
0.84	0.107	0.194	0.074

0.85	0.106	0.199	0.072
0.86	0.110	0.242	0.071
0.87	0.107	0.247	0.069
0.88	0.107	0.252	0.068
0.89	0.104	0.254	0.066
0.90	0.102	0.258	0.063
0.91	0.098	0.256	0.060
0.92	0.094	0.278	0.056
0.93	0.087	0.269	0.052
0.94	0.086	0.278	0.051
0.95	0.083	0.275	0.049
0.96	0.073	0.263	0.043
0.97	0.068	0.289	0.039
0.98	0.060	0.299	0.033
0.99	0.050	0.321	0.027

**Métricas de rendimiento de la regresión logística durante la validación cruzada (sólo cadena eth) para diferentes umbrales de decisión.*

Tabla 22: *Performance Logistic Regression* VC (bsc) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.621	0.612	0.630
0.51	0.613	0.608	0.618
0.52	0.608	0.605	0.610
0.53	0.603	0.602	0.604
0.54	0.603	0.603	0.603
0.55	0.597	0.600	0.595

0.56	0.589	0.596	0.583
0.57	0.586	0.595	0.578
0.58	0.582	0.593	0.572
0.59	0.577	0.590	0.565
0.60	0.566	0.584	0.549
0.61	0.558	0.579	0.539
0.62	0.553	0.577	0.531
0.63	0.547	0.575	0.522
0.64	0.533	0.567	0.504
0.65	0.533	0.569	0.501
0.66	0.521	0.562	0.486
0.67	0.516	0.561	0.478
0.68	0.512	0.564	0.469
0.69	0.503	0.562	0.455
0.70	0.500	0.561	0.451
0.71	0.488	0.554	0.435
0.72	0.477	0.547	0.423
0.73	0.474	0.548	0.417
0.74	0.466	0.545	0.408
0.75	0.456	0.537	0.396
0.76	0.444	0.532	0.381
0.77	0.432	0.525	0.367
0.78	0.423	0.520	0.356
0.79	0.415	0.516	0.347
0.80	0.388	0.496	0.318
0.81	0.379	0.494	0.307

0.82	0.361	0.479	0.289
0.83	0.339	0.462	0.268
0.84	0.273	0.402	0.207
0.85	0.265	0.396	0.199
0.86	0.254	0.389	0.189
0.87	0.189	0.319	0.134
0.88	0.184	0.316	0.129
0.89	0.176	0.308	0.123
0.90	0.168	0.298	0.117
0.91	0.161	0.291	0.111
0.92	0.153	0.280	0.105
0.93	0.148	0.280	0.100
0.94	0.148	0.283	0.100
0.95	0.142	0.276	0.096
0.96	0.132	0.278	0.087
0.97	0.129	0.297	0.082
0.98	0.120	0.310	0.075
0.99	0.117	0.365	0.070

**Métricas de rendimiento de la regresión logística durante la validación cruzada (sólo cadena bsc) para diferentes umbrales de decisión.*

Tabla 23: *Performance Logistic Regression* VC (avalanche) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.188	0.115	0.500
0.53	0.168	0.104	0.444
0.57	0.170	0.105	0.444

0.58	0.151	0.093	0.389
0.59	0.112	0.070	0.278
0.60	0.115	0.072	0.278
0.61	0.116	0.074	0.278
0.62	0.122	0.078	0.278
0.64	0.101	0.066	0.222
0.65	0.027	0.018	0.056
0.68	0.029	0.020	0.056
0.71	0.030	0.020	0.056
0.76	0.030	0.021	0.056
0.87	0.031	0.022	0.056
0.88	0.036	0.027	0.056

**Métricas de rendimiento de la regresión logística durante la validación cruzada (sólo cadena avalanche) para diferentes umbrales de decisión.*

Tabla 24: *Performance Logistic Regression* VC (arbitrum) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.881	0.863	0.899
0.51	0.880	0.862	0.898
0.54	0.878	0.862	0.894
0.55	0.877	0.862	0.893
0.57	0.877	0.863	0.893
0.58	0.878	0.864	0.893
0.59	0.878	0.864	0.892
0.60	0.877	0.865	0.889
0.61	0.876	0.865	0.888

0.62	0.877	0.866	0.888
0.63	0.877	0.867	0.887
0.64	0.876	0.866	0.886
0.65	0.877	0.869	0.885
0.66	0.875	0.869	0.882
0.68	0.878	0.876	0.881
0.69	0.875	0.875	0.874
0.70	0.876	0.877	0.874
0.71	0.877	0.879	0.874
0.72	0.876	0.879	0.873
0.73	0.876	0.881	0.872
0.74	0.874	0.880	0.869
0.75	0.875	0.881	0.869
0.76	0.874	0.882	0.867
0.77	0.874	0.882	0.866
0.78	0.872	0.881	0.862
0.79	0.873	0.883	0.862
0.80	0.871	0.883	0.860
0.81	0.872	0.886	0.859
0.82	0.872	0.887	0.858
0.83	0.872	0.889	0.856
0.84	0.871	0.889	0.854
0.85	0.870	0.889	0.852
0.86	0.684	0.844	0.575
0.87	0.559	0.807	0.427
0.88	0.560	0.812	0.427

0.89	0.559	0.820	0.424
0.90	0.559	0.826	0.422
0.91	0.400	0.769	0.270
0.92	0.395	0.770	0.266
0.93	0.393	0.769	0.264
0.94	0.393	0.781	0.262
0.95	0.394	0.788	0.262
0.96	0.393	0.795	0.261
0.97	0.397	0.828	0.261
0.98	0.344	0.814	0.218
0.99	0.094	0.595	0.051

**Métricas de rendimiento de la regresión logística durante la validación cruzada (sólo cadena arbitruml) para diferentes umbrales de decisión.*

Tabla 25: *Performance Logistic Regression* VC (polygon) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.439	0.341	0.616
0.51	0.440	0.342	0.616
0.52	0.439	0.344	0.608
0.53	0.436	0.342	0.600
0.54	0.437	0.344	0.600
0.56	0.415	0.330	0.560
0.57	0.408	0.327	0.544
0.58	0.389	0.314	0.512
0.59	0.384	0.310	0.504

0.60	0.379	0.307	0.496
0.61	0.374	0.303	0.488
0.62	0.373	0.305	0.480
0.63	0.351	0.293	0.440
0.64	0.335	0.281	0.416
0.65	0.338	0.284	0.416
0.66	0.327	0.276	0.400
0.67	0.317	0.270	0.384
0.68	0.307	0.268	0.360
0.69	0.293	0.259	0.336
0.71	0.281	0.250	0.320
0.73	0.289	0.263	0.320
0.74	0.266	0.247	0.288
0.75	0.253	0.236	0.272
0.76	0.256	0.241	0.272
0.77	0.251	0.239	0.264
0.78	0.257	0.250	0.264
0.79	0.258	0.252	0.264
0.81	0.259	0.254	0.264
0.82	0.262	0.260	0.264
0.83	0.263	0.262	0.264
0.84	0.265	0.266	0.264
0.85	0.258	0.260	0.256
0.86	0.239	0.246	0.232
0.87	0.233	0.243	0.224
0.88	0.213	0.227	0.200

0.89	0.214	0.229	0.200
0.90	0.215	0.231	0.200
0.91	0.217	0.238	0.200
0.92	0.220	0.245	0.200
0.93	0.215	0.245	0.192
0.94	0.208	0.240	0.184
0.95	0.205	0.244	0.176
0.96	0.213	0.268	0.176
0.97	0.230	0.333	0.176
0.98	0.238	0.367	0.176
0.99	0.247	0.415	0.176

**Métricas de rendimiento de la regresión logística durante la validación cruzada (sólo cadena polygon) para diferentes umbrales de decisión.*

Tabla 26: *Performance Logistic Regression* VC (optimism) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.280	0.182	0.609
0.54	0.286	0.187	0.609
0.65	0.289	0.189	0.609
0.67	0.311	0.209	0.609
0.70	0.315	0.212	0.609
0.75	0.295	0.200	0.565
0.79	0.299	0.203	0.565
0.80	0.282	0.194	0.522
0.81	0.286	0.197	0.522
0.84	0.289	0.200	0.522

0.85	0.293	0.203	0.522
0.86	0.296	0.207	0.522
0.87	0.300	0.211	0.522
0.90	0.304	0.214	0.522
0.91	0.114	0.085	0.174
0.92	0.118	0.089	0.174
0.93	0.125	0.098	0.174
0.95	0.127	0.100	0.174
0.96	0.133	0.108	0.174
0.97	0.143	0.121	0.174
0.98	0.043	0.043	0.043
0.99	0.047	0.050	0.043

**Métricas de rendimiento de la regresión logística durante la validación cruzada (sólo cadena de optimism) para diferentes umbrales de decisión.*

Tabla 27: Performance Logistic Regression VC (ftm) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.400	0.500	0.333

**Métricas de rendimiento de la regresión logística durante la validación cruzada (sólo cadena ftm) para el umbral 0.5. Nota: Los umbrales ≥ 0,51 resultaron en F1/Precision/Recall cero en esta prueba.*

Tabla 28: *Performance Logistic Regression* Test (General) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.443	0.613	0.347
0.51	0.424	0.609	0.326
0.52	0.405	0.612	0.303

0.53	0.385	0.618	0.280
0.54	0.343	0.602	0.240
0.55	0.304	0.576	0.207
0.56	0.256	0.544	0.167
0.57	0.223	0.523	0.142
0.58	0.170	0.455	0.104
0.59	0.155	0.446	0.094
0.60	0.150	0.453	0.090
0.61	0.116	0.432	0.067
0.62	0.107	0.446	0.061
0.63	0.089	0.407	0.050
0.64	0.079	0.389	0.044
0.65	0.076	0.408	0.042
0.66	0.065	0.395	0.035
0.67	0.051	0.394	0.027
0.68	0.028	0.292	0.015
0.69	0.020	0.294	0.010

**Métricas de rendimiento de la regresión logística en el conjunto de prueba (agregadas en todas las cadenas) para diferentes umbrales de decisión. Nota: Los umbrales ≥ 0,70 resultaron en F1/Precision/Recuperación cero en esta prueba.*

Apéndice D. Resultados *Easy Ensemble Classifier*

Tabla 29: *Performance Easy Ensemble Classifier* (General) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.638	0.495	0.896
0.51	0.658	0.526	0.880

0.52	0.664	0.543	0.854
0.53	0.663	0.563	0.806
0.54	0.639	0.565	0.733
0.55	0.603	0.563	0.650
0.56	0.563	0.572	0.553
0.57	0.530	0.601	0.474
0.58	0.508	0.683	0.404
0.59	0.464	0.722	0.342
0.60	0.215	0.605	0.131
0.61	0.021	0.280	0.011

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (agregadas en todas las cadenas) para diferentes umbrales de decisión.*

Tabla 30: *Performance* de *Easy Ensemble Classifier* (eth) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.433	0.311	0.708
0.51	0.464	0.354	0.673
0.52	0.474	0.382	0.625
0.53	0.477	0.418	0.556
0.54	0.415	0.401	0.430
0.55	0.366	0.394	0.341
0.56	0.271	0.360	0.217
0.57	0.177	0.319	0.123
0.58	0.118	0.426	0.069
0.59	0.064	0.351	0.035
0.60	0.031	0.254	0.017

0.61	0.012	0.375	0.006

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (sólo cadena eth) para diferentes umbrales de decisión.*

Tabla 31 : *Performance* de *Easy Ensemble Classifier* (bsc) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.775	0.633	0.998
0.51	0.773	0.632	0.992
0.52	0.760	0.626	0.967
0.53	0.735	0.616	0.912
0.54	0.716	0.612	0.863
0.55	0.645	0.581	0.726
0.56	0.592	0.565	0.623
0.57	0.530	0.558	0.505
0.58	0.436	0.554	0.359
0.59	0.289	0.516	0.201
0.60	0.124	0.368	0.075
0.61	0.051	0.353	0.027
0.62	0.018	0.462	0.009

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (sólo cadena bsc) para diferentes umbrales de decisión.*

Tabla 32: *Performance* de *Easy Ensemble Classifier* (avalanche) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.316	0.188	1.000
0.51	0.319	0.189	1.000

0.54	0.306	0.183	0.944
0.55	0.296	0.178	0.889
0.56	0.227	0.143	0.556
0.57	0.059	0.040	0.111

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (sólo cadena avalanche) para diferentes umbrales de decisión.*

Tabla 33: *Performance* de *Easy Ensemble Classifier* (arbitrum) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.892	0.806	0.999
0.51	0.890	0.806	0.995
0.52	0.888	0.805	0.989
0.53	0.882	0.806	0.973
0.54	0.877	0.816	0.947
0.55	0.867	0.822	0.916
0.56	0.855	0.835	0.875
0.57	0.860	0.867	0.854
0.58	0.863	0.904	0.825
0.59	0.858	0.922	0.803
0.60	0.446	0.873	0.299

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (sólo cadena arbitrum) para diferentes umbrales de decisión.*

Tabla 34: *Performance* de *Easy Ensemble Classifier* (polygon) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.553	0.382	1.000

0.52	0.547	0.378	0.984
0.53	0.514	0.362	0.888
0.54	0.468	0.337	0.768
0.55	0.407	0.302	0.624
0.56	0.306	0.251	0.392
0.57	0.262	0.246	0.280
0.58	0.200	0.232	0.176
0.59	0.143	0.197	0.112
0.60	0.085	0.179	0.056
0.61	0.029	0.167	0.016

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (sólo cadena polygon) para diferentes umbrales de decisión.*

Tabla 35: *Performance* de *Easy Ensemble Classifier* (optimism) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.346	0.209	1.000
0.53	0.333	0.202	0.957
0.54	0.328	0.200	0.913
0.55	0.271	0.168	0.696
0.56	0.286	0.183	0.652
0.57	0.217	0.145	0.435
0.58	0.278	0.204	0.435
0.59	0.209	0.159	0.304
0.60	0.269	0.241	0.304

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (sólo cadena optimism) para diferentes umbrales de decisión.*

Tabla 36: *Performance* de *Easy Ensemble Classifier* (ftm) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.600	0.429	1.000
0.52	0.444	0.333	0.667

**Métricas de rendimiento de Easy Ensemble Classifier durante la validación cruzada (sólo cadena ftm) para los diferentes umbrales de decisión.*

Tabla 37: *Performance* de *Easy Ensemble Classifier* en de VC (General) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.220	0.125	0.914
0.51	0.281	0.167	0.875
0.52	0.391	0.253	0.862
0.53	0.428	0.299	0.756
0.54	0.506	0.438	0.599
0.55	0.553	0.564	0.543
0.56	0.527	0.650	0.443
0.57	0.460	0.773	0.328
0.58	0.313	0.883	0.190
0.59	0.053	0.929	0.027

**Métricas de rendimiento de Easy Ensemble Classifier en el conjunto de prueba (agregadas en todas las cadenas) para diferentes umbrales de decisión.*

Apéndice E. Resultados Balanced Random Forest Classifier

Tabla 38: Rendimiento de BRFC (General) por Umbral.

Threshold	F1-Score	Precision	Recall
0.50	0.631	0.488	0.893
0.51	0.633	0.491	0.891
0.52	0.636	0.494	0.890
0.53	0.639	0.499	0.889
0.54	0.643	0.504	0.886
0.55	0.646	0.509	0.884
0.56	0.648	0.512	0.882
0.57	0.651	0.516	0.881
0.58	0.653	0.520	0.879
0.59	0.656	0.523	0.879
0.60	0.658	0.527	0.878
0.61	0.664	0.535	0.875
0.62	0.669	0.543	0.873
0.63	0.674	0.549	0.872
0.64	0.678	0.556	0.870
0.65	0.681	0.560	0.868
0.66	0.682	0.562	0.866
0.67	0.684	0.566	0.863
0.68	0.685	0.571	0.857
0.69	0.687	0.577	0.850
0.70	0.691	0.584	0.847
0.71	0.692	0.586	0.843
0.72	0.692	0.590	0.839
0.73	0.694	0.593	0.838
0.74	0.695	0.595	0.835

0.75	0.697	0.600	0.832
0.76	0.697	0.603	0.827
0.77	0.698	0.606	0.822
0.78	0.699	0.611	0.818
0.79	0.699	0.615	0.809
0.80	0.696	0.616	0.800
0.81	0.692	0.618	0.788
0.82	0.697	0.631	0.780
0.83	0.698	0.636	0.773
0.84	0.698	0.644	0.763
0.85	0.696	0.649	0.750
0.86	0.691	0.652	0.736
0.87	0.687	0.657	0.720
0.88	0.679	0.661	0.699
0.89	0.677	0.670	0.684
0.90	0.669	0.676	0.662
0.91	0.669	0.692	0.647
0.92	0.661	0.697	0.628
0.93	0.652	0.705	0.607
0.94	0.636	0.706	0.578
0.95	0.621	0.709	0.551
0.96	0.596	0.715	0.510
0.97	0.574	0.738	0.470
0.98	0.553	0.798	0.423
0.99	0.516	0.855	0.370

**Métricas de rendimiento de Balanced Random Forest Classifier durante la validación cruzada (agregadas en todas las cadenas) para diferentes umbrales de decisión.*

Tabla 39: *Perfomance* de BRFC (eth) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.421	0.301	0.699
0.51	0.423	0.304	0.696
0.52	0.426	0.308	0.693
0.53	0.431	0.314	0.689
0.54	0.436	0.320	0.683
0.55	0.439	0.326	0.675
0.56	0.443	0.331	0.672
0.57	0.448	0.336	0.670
0.58	0.451	0.341	0.664
0.59	0.455	0.346	0.664
0.60	0.459	0.351	0.663
0.61	0.469	0.364	0.658
0.62	0.475	0.374	0.652
0.63	0.482	0.382	0.650
0.64	0.490	0.394	0.647
0.65	0.496	0.403	0.644
0.66	0.498	0.406	0.643
0.67	0.501	0.412	0.638
0.68	0.507	0.423	0.632
0.69	0.512	0.432	0.627
0.70	0.519	0.443	0.626
0.71	0.518	0.446	0.619

0.72	0.518	0.449	0.611
0.73	0.520	0.453	0.610
0.74	0.520	0.457	0.605
0.75	0.521	0.462	0.597
0.76	0.522	0.467	0.591
0.77	0.521	0.471	0.583
0.78	0.520	0.476	0.573
0.79	0.514	0.474	0.561
0.80	0.501	0.468	0.539
0.81	0.488	0.465	0.513
0.82	0.487	0.477	0.498
0.83	0.483	0.479	0.487
0.84	0.484	0.488	0.481
0.85	0.478	0.487	0.470
0.86	0.468	0.485	0.452
0.87	0.466	0.493	0.442
0.88	0.463	0.497	0.434
0.89	0.462	0.504	0.427
0.90	0.452	0.503	0.410
0.91	0.439	0.505	0.388
0.92	0.418	0.495	0.362
0.93	0.401	0.489	0.339
0.94	0.375	0.477	0.309
0.95	0.352	0.463	0.284
0.96	0.323	0.451	0.252
0.97	0.292	0.452	0.215

0.98	0.249	0.485	0.168
0.99	0.198	0.529	0.122

**Métricas de rendimiento de Balanced Random Forest Classifier durante la validación cruzada (sólo cadena eth) para diferentes umbrales de decisión.*

Tabla 40: *Performance* de BRFC (bsc) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.776	0.634	0.998
0.52	0.775	0.634	0.997
0.56	0.774	0.634	0.995
0.58	0.775	0.635	0.995
0.59	0.776	0.636	0.995
0.60	0.777	0.637	0.995
0.61	0.777	0.638	0.994
0.62	0.780	0.642	0.992
0.63	0.781	0.645	0.991
0.64	0.782	0.645	0.991
0.65	0.780	0.644	0.988
0.66	0.779	0.644	0.985
0.67	0.778	0.644	0.982
0.68	0.775	0.643	0.977
0.69	0.772	0.644	0.963
0.70	0.775	0.648	0.962
0.71	0.774	0.649	0.959
0.72	0.774	0.651	0.956
0.73	0.775	0.652	0.954

0.74	0.775	0.652	0.954
0.75	0.777	0.655	0.953
0.76	0.774	0.654	0.948
0.77	0.773	0.655	0.942
0.78	0.774	0.660	0.936
0.79	0.773	0.665	0.922
0.80	0.776	0.670	0.922
0.81	0.774	0.672	0.915
0.82	0.778	0.680	0.907
0.83	0.778	0.682	0.906
0.84	0.771	0.682	0.887
0.85	0.762	0.682	0.863
0.86	0.758	0.685	0.849
0.87	0.740	0.682	0.808
0.88	0.719	0.675	0.769
0.89	0.704	0.671	0.740
0.90	0.680	0.667	0.694
0.91	0.680	0.684	0.677
0.92	0.674	0.695	0.654
0.93	0.665	0.705	0.629
0.94	0.645	0.705	0.595
0.95	0.616	0.717	0.540
0.96	0.543	0.703	0.443
0.97	0.485	0.712	0.368
0.98	0.442	0.767	0.311
0.99	0.319	0.799	0.199

**Métricas de rendimiento de Balance Random Forest Classifier durante la validación cruzada (sólo cadena bsc) para diferentes umbrales de decisión.*

Tabla 41: *Performance* de BRFC (avalanche) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.316	0.188	1.000
0.55	0.319	0.189	1.000
0.60	0.304	0.181	0.944
0.67	0.306	0.183	0.944
0.70	0.309	0.185	0.944
0.73	0.312	0.187	0.944
0.78	0.315	0.189	0.944
0.80	0.302	0.182	0.889
0.82	0.305	0.184	0.889
0.83	0.327	0.200	0.889
0.84	0.349	0.221	0.833
0.85	0.417	0.278	0.833
0.86	0.423	0.283	0.833
0.88	0.484	0.341	0.833
0.89	0.536	0.395	0.833
0.90	0.462	0.353	0.667
0.91	0.468	0.379	0.611
0.92	0.478	0.393	0.611
0.93	0.537	0.478	0.611
0.94	0.333	0.333	0.333
0.95	0.138	0.182	0.111

0.96	0.154	0.250	0.111
0.97	0.080	0.143	0.056
0.98	0.095	0.333	0.056

**Métricas de rendimiento de Balance RandomForest Classifier durante la validación cruzada (sólo cadena avalanche) para diferentes umbrales de decisión.*

Tabla 42: *Performance* de BRFC (arbitrum) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.892	0.806	0.999
0.58	0.893	0.807	0.999
0.59	0.893	0.808	0.999
0.60	0.894	0.809	0.998
0.62	0.895	0.813	0.997
0.63	0.898	0.817	0.997
0.65	0.897	0.817	0.996
0.66	0.898	0.818	0.996
0.67	0.899	0.819	0.996
0.68	0.894	0.818	0.987
0.69	0.896	0.824	0.983
0.70	0.897	0.829	0.977
0.71	0.898	0.832	0.975
0.72	0.899	0.834	0.975
0.73	0.900	0.836	0.975
0.74	0.899	0.835	0.973
0.75	0.900	0.839	0.971
0.76	0.900	0.841	0.969

0.77	0.901	0.842	0.969
0.78	0.903	0.846	0.969
0.79	0.907	0.853	0.967
0.80	0.906	0.855	0.963
0.81	0.908	0.859	0.963
0.82	0.913	0.868	0.962
0.83	0.914	0.871	0.961
0.84	0.912	0.875	0.953
0.85	0.911	0.878	0.948
0.86	0.909	0.879	0.941
0.87	0.913	0.887	0.940
0.88	0.908	0.888	0.928
0.89	0.907	0.893	0.921
0.90	0.911	0.908	0.913
0.91	0.913	0.918	0.909
0.92	0.911	0.919	0.902
0.93	0.910	0.924	0.896
0.94	0.906	0.930	0.883
0.95	0.904	0.933	0.877
0.96	0.904	0.943	0.868
0.97	0.902	0.963	0.848
0.98	0.889	0.979	0.815
0.99	0.885	0.987	0.803

**Métricas de rendimiento de Balance RandomForest Classifier durante la validación cruzada (sólo cadena arbitrum) para diferentes umbrales de decisión.*

Tabla 43: *Performance* de BRFC (polygon) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.553	0.382	1.000
0.60	0.554	0.383	1.000
0.61	0.551	0.382	0.992
0.62	0.556	0.386	0.992
0.63	0.557	0.388	0.992
0.64	0.554	0.386	0.984
0.66	0.553	0.386	0.976
0.67	0.553	0.387	0.968
0.68	0.554	0.388	0.968
0.69	0.550	0.388	0.944
0.70	0.551	0.389	0.944
0.71	0.549	0.389	0.936
0.72	0.551	0.390	0.936
0.73	0.560	0.399	0.936
0.74	0.561	0.401	0.936
0.75	0.568	0.408	0.936
0.76	0.567	0.409	0.920
0.77	0.569	0.412	0.920
0.79	0.574	0.419	0.912
0.81	0.567	0.415	0.896
0.82	0.576	0.428	0.880
0.83	0.569	0.426	0.856
0.84	0.568	0.429	0.840

0.85	0.579	0.444	0.832
0.86	0.573	0.446	0.800
0.87	0.553	0.437	0.752
0.88	0.514	0.426	0.648
0.89	0.514	0.449	0.600
0.90	0.521	0.471	0.584
0.91	0.530	0.497	0.568
0.92	0.504	0.489	0.520
0.93	0.438	0.453	0.424
0.94	0.381	0.415	0.352
0.95	0.361	0.402	0.328
0.96	0.307	0.367	0.264
0.97	0.269	0.355	0.216
0.98	0.204	0.405	0.136
0.99	0.056	0.222	0.032

**Métricas de rendimiento de Balance RandomForest Classifier durante la validación cruzada (sólo cadena polygon) para diferentes umbrales de decisión.*

Tabla 44: *Perfomance* de BRFC (*optimism*) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.346	0.209	1.000
0.72	0.338	0.206	0.957
0.75	0.341	0.208	0.957
0.78	0.344	0.210	0.957
0.79	0.349	0.214	0.957
0.80	0.352	0.216	0.957

0.82	0.376	0.234	0.957
0.83	0.372	0.233	0.913
0.84	0.385	0.244	0.913
0.85	0.362	0.232	0.826
0.86	0.346	0.222	0.783
0.87	0.353	0.228	0.783
0.88	0.364	0.237	0.783
0.89	0.383	0.254	0.783
0.90	0.341	0.231	0.652
0.91	0.370	0.259	0.652
0.92	0.400	0.288	0.652
0.93	0.441	0.333	0.652
0.94	0.444	0.350	0.609
0.96	0.483	0.400	0.609
0.97	0.491	0.433	0.565
0.98	0.468	0.458	0.478
0.99	0.462	0.562	0.391

**Métricas de rendimiento de Balance RandomForest Classifier durante la validación cruzada (sólo cadena optimism) para diferentes umbrales de decisión.*

Tabla 45: *Performance* de BRFC (ftm) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.600	0.429	1.000
0.56	0.444	0.333	0.667
0.86	0.250	0.200	0.333
0.90	0.333	0.333	0.333

0.91	0.400	0.500	0.333

**Métricas de rendimiento de Balance RandomForest Classifier durante la validación cruzada (sólo cadena ftm) para los distintos umbrales. Nota: En el resto de los umbrales la performance fue o.*

Tabla 46: *Performance* de BRFC (General) por Umbral

Threshold	F1-Score	Precision	Recall
0.50	0.193	0.108	0.885
0.51	0.198	0.111	0.885
0.52	0.204	0.115	0.885
0.53	0.210	0.119	0.881
0.54	0.215	0.123	0.875
0.55	0.221	0.127	0.871
0.56	0.227	0.131	0.871
0.57	0.233	0.135	0.862
0.58	0.240	0.139	0.860
0.59	0.248	0.145	0.860
0.60	0.254	0.149	0.858
0.61	0.261	0.154	0.858
0.62	0.267	0.158	0.854
0.63	0.272	0.162	0.846
0.64	0.278	0.167	0.837
0.65	0.283	0.171	0.831
0.66	0.292	0.177	0.825
0.67	0.300	0.183	0.825
0.68	0.307	0.188	0.823
0.69	0.313	0.193	0.820

0.70	0.321	0.200	0.814
0.71	0.328	0.206	0.804
0.72	0.340	0.216	0.802
0.73	0.352	0.225	0.800
0.74	0.359	0.232	0.791
0.75	0.369	0.241	0.785
0.76	0.384	0.254	0.783
0.77	0.395	0.266	0.768
0.78	0.404	0.276	0.756
0.79	0.410	0.284	0.741
0.80	0.424	0.298	0.737
0.81	0.430	0.309	0.708
0.82	0.431	0.316	0.678
0.83	0.440	0.330	0.660
0.84	0.453	0.347	0.651
0.85	0.461	0.361	0.635
0.86	0.466	0.377	0.608
0.87	0.476	0.397	0.593
0.88	0.483	0.420	0.568
0.89	0.467	0.432	0.507
0.90	0.465	0.457	0.472
0.91	0.460	0.487	0.436
0.92	0.458	0.524	0.407
0.93	0.417	0.538	0.340
0.94	0.402	0.591	0.305
0.95	0.361	0.645	0.251

0.96	0.289	0.672	0.184
0.97	0.185	0.699	0.106
0.98	0.056	0.636	0.029

**Métricas de rendimiento de Balance Random Forest Classifier en el conjunto de prueba (agregadas en todas las cadenas) para diferentes umbrales de decisión.*