Crimen financiero

Detección de fraude en tarjetas de crédito aplicando aprendizaje automático

Álvaro Artola Moreno

Máster Universitario en Ingeniería de Telecomunicación

Smart Cities

Tutor/a de TF
Rubén Molina Casasnovas
Profesor/a responsable de la asignatura
Carlos Monzo Sánchez

Junio 2023

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Ficha del Trabajo Final

Título del trabajo:	Crimen financiero – detección de fraude en tarjetas de crédito aplicando aprendizaje automático.
Nombre del autor/a:	Álvaro Artola Moreno
Nombre del Tutor/a de TF:	Rubén Molina Casanovas
Nombre del/de la PRA:	Carlos Monzo Sánchez
Fecha de entrega:	06/2023
Titulación o programa:	Máster Universitario en Ingeniería de Telecomunicación
Área del Trabajo Final:	Smart Cities
Idioma del trabajo:	Castellano
Palabras clave	**fraude, aprendizaje automático**, crédito, débito, regresión, inteligencia artificial, **clasificación**

Resumen del Trabajo

El proyecto se enfoca en la implementación de modelos predictivos basados en técnicas de aprendizaje automático para prevenir el fraude financiero en transacciones con tarjetas de crédito y débito. La gran cantidad de datos disponibles para su análisis hace que se convierta en una oportunidad para construir modelos predictivos que permitan detectar patrones y comportamientos anómalos en las transacciones.

Se han analizado diversas técnicas de aprendizaje automático, como el análisis de regresión logística, los árboles de decisión, las redes neuronales y la clasificación por vecinos más cercanos, seleccionándose por su capacidad para trabajar con grandes volúmenes de datos y su capacidad para adaptarse a diferentes tipos de modelos.

Los resultados obtenidos en el desarrollo del proyecto son muy prometedores, ya que los modelos predictivos implementados han demostrado una alta precisión en la detección de transacciones anómalas sospechosas de fraude. La identificación temprana de estas actividades fraudulentas puede ser de gran utilidad para las entidades financieras, ya que les permite actuar rápidamente para evitar pérdidas económicas y

proteger a sus clientes de posibles ataques fraudulentos.

Abstract

The goal of this project is to implement predictive models utilising machine learning techniques in order to prevent financial fraud in transactions made using credit and debit cards. The vast amount of data available for analysis presents an opportunity to construct predictive models which can detect anomalous patterns and behaviours in transactions.

Several machine learning techniques have been evaluated, including logistic regression analysis, decision trees, neural networks, and k-nearest neighbour classification, with an emphasis on their ability to handle large volumes of data and adapt to various types of models.

The results of this project are highly promising, as the implemented predictive models have demonstrated a high degree of accuracy in detecting suspicious anomalous transactions. Early identification of fraudulent activities can be of great benefit to financial institutions, enabling them to take swift action to prevent economic losses and safeguard their clients from potential fraudulent attacks.

Índice

Contents

Lista de Figuras

1. Introducción

En este primer capítulo de la memoria nos situaremos en el contexto y comentaremos la justificación del trabajo. Se fijarán los objetivos, su impacto ético social, la planificación de éste para llevarlo a cabo, un breve sumario de los productos obtenidos y se finalizará con una breve descripción del resto de capítulos.

1.1. Contexto y justificación del Trabajo

La motivación de este proyecto surge a partir de dos factores:

- En los últimos años, cuando el mundo estaba bloqueado por la COVID-19 y los movimientos restringidos debido a una emergencia absoluta, millones de personas conocieron o intensificaron las compras en línea. Esta situación ayudó a las plataformas de comercio electrónico a registrar ventas históricas. Sin embargo, mientras esto sucedía, el índice de fraudes financieros en línea también aumentó considerablemente. Los casos de fraude en línea con tarjetas de crédito y débito experimentaron un aumento histórico del 225% durante la pandemia en 2020 en comparación con 2019. Según el informe de la NCRB, el recuento de fraudes con tarjetas de crédito y débito se situó en 1194 en 2020, frente a 367 en 2019.
- La imperiosa necesidad de la banca hacia una migración tecnológica como la que empresas con, a priori, menor poder adquisitivo, ya han realizado y que potenciaría evitar la presencia de fraude entre sus consumidores o al menos detectarla con anterioridad y poder ganar así, mayor capacidad de reacción.

1.2. Objetivos del Trabajo

El objetivo principal del proyecto, por tanto, será aprovechar las últimas tendencias en análisis de datos, así como la comparación de los algoritmos más novedosos dentro del concepto Aprendizaje Automático – Inteligencia Artificial.

El término fraude con tarjeta de crédito puede definirse como el acceso no autorizado a tarjetas de pago como tarjetas de crédito o débito para pagar por el uso de servicios o bienes. Los hackers o defraudadores pueden obtener los datos confidenciales de la tarjeta a través de sitios web no seguros. Cuando un estafador compromete la tarjeta de crédito/débito de un individuo, todos los implicados en el proceso sufren, desde el individuo cuyos datos confidenciales han sido filtrados hasta las empresas (generalmente bancos) que emiten la tarjeta de crédito y el comerciante que finaliza la transacción con la compra.

Esto hace que sea extremadamente esencial identificar las transacciones fraudulentas desde el principio. Las instituciones financieras y empresas como las de comercio electrónico están tomando medidas firmes para señalar a los defraudadores que entran en el sistema. Diversas tecnologías avanzadas de aprendizaje automático están en juego, evaluando cada transacción y deteniendo a los usuarios fraudulentos en su inicio utilizando datos de comportamiento y patrones de transacción. El proceso de diferenciar automáticamente entre usuarios fraudulentos y auténticos es conocido como "detección de fraude con tarjetas de crédito". Se pretende que en un futuro entidades financieras, al igual que las de comercio electrónico sean capaces, a raíz de estas investigaciones de detectar y mitigar el fraude con mayor precisión y que dichos casos terminen de ser noticia tan a menudo.

1.3. Impacto en sostenibilidad, ético-social y de diversidad

Dimensión sostenibilidad

El resultado de este trabajo final no tiene un impacto en aspectos de sostenibilidad medioambiental, aunque sí que es cierto que puede interpretarse como una mejora en la huella ecológica en un futuro. Tras los análisis que se plantean concluir, un punto final importante será que las instituciones acaben integrando algoritmos que ayuden a mitigar el fraude siendo necesarios menos recursos que los que a día de hoy se utilizan. Reducir los procesos de auditoría es un ejemplo de lo que se podría lograr gracias a la digitalización.
Es importante tener en cuenta los Objetivos de Desarrollo Sostenible para Smart Cities [1].

Dimensión comportamiento ético y de responsabilidad social

Desde una perspectiva ética y de responsabilidad social, tiene un claro impacto positivo. Se pretende salvaguardar los datos de los usuarios bancarios, así como la seguridad de las personas. Preservar la identidad de los consumidores que ven como hackean sus cuentas para llevar a cabo acciones delictivas es un claro gesto positivo hacia una buena responsabilidad social.

Además, se pretende que las instituciones bancarias mejoren sus herramientas de detección y que, con ello, estemos en una sociedad más justa.

1.4. Enfoque y método seguido

Diseñar y desarrollar un producto requiere seguir una metodología de trabajo. Debido a multitud de opciones posibles se debe escoger aquella que mejor se adapte a cada situación,

característica y necesidad. En definitiva, se puede desarrollar algoritmos nuevos, modificar los existentes o ampliar y utilizar alguna métrica que lo mejore.
Considerando que la propuesta del trabajo final surge como motivación personal para ampliar los conocimientos algorítmicos en relación con las finanzas, y más específicamente al fraude financiero, se evidencia la necesidad de comprobar de entre la multitud de opciones disponibles dentro del concepto de aprendizaje automático cuáles son los modelos que con mayor precisión podrán detectar el fraude crediticio. No sólo se buscará una precisión lo más exacta posible, sino que la detección se lleve a cabo en los orígenes de las transacciones, es decir, con la mayor prontitud posible para evitar que las cuantías de dicho fraude no sean tan elevadas.

Abordando el proyecto desde una perspectiva detallada en sus fases de elaboración, para concretar las bases estructurales previas, se sigue una metodología de investigación documental, analizando los distintos tipos de fraude que se dan en la actualidad.

El método final consiste en la comprobación de los resultados, comparando los modelos aplicados en las mismas bases de datos y analizando cuál será el más eficiente en precisión y tiempo.

1.5. Planificación del trabajo

De igual modo que se debe seguir una metodología de trabajo para la buena orientación en la consecución de los objetivos, la planificación está intrínsecamente relacionada y resulta indispensable para la buena organización de las tareas definidas en el método. El trabajo, sigue el principio de flexibilidad, de forma que considera la existencia de riesgos y posibles nuevas necesidades, pudiendo ser modificada durante el progreso del proyecto en cada fase de evaluación, sin alterar el producto final, tal y como se ha descrito en el apartado anterior.

La organización de las tareas ha sido realizada siguiendo una tabla de hitos. Se pueden identificar las entregas parciales estipuladas, en las secciones de sombreado azul. Secundariamente, en cada bloque temporal, se organiza de la definición de las tareas metodológicas, así como el cálculo de sus tiempos de elaboración (duración en días). Se expone con más detalle, las fechas exactas en las que se estima y recomienda comenzar y finalizar cada una de las tareas específicas, teniendo en cuenta para su definición, factores como el esfuerzo requerido, la dificultad, los posibles riesgos o necesidades que exijan una mayor dedicación, etc. Por último, se muestra una referencia detallada del coste(C) que supone cada tarea concreta, y la suma total de su global, siguiendo el siguiente parámetro para su cálculo:

C. tareas generales	Bajo (B)	Bajo - Medio (B+)	Medio (M)	Medio - Alto (M+)	Alto (A)	Muy Alto (A+)
C. subtareas	1	2	3	4	5	6

Se otorga un valor numérico (C. subtareas) a cada tarea específica, siendo 1 el coste muy bajo, y 6 el coste de desarrollo muy elevado

ID	Nombre	Duración (días)	Fecha de inicio	Fecha final	Coste (C)
1	PEC 1	11	01/03/2023	12/03/2023	**B+**
1.1	Propuesta de título	1	04/03/2023	05/03/2023	1
1.2	Resumen	1	11/03/2023	12/03/2023	4
1.3	Palabras clave	2	10/03/2023	12/03/2023	1
1.4	Objetivos del proyecto	2	10/03/2023	12/03/2023	3
1.5	Impacto en sostenibilidad, ético-social y de diversidad	2	10/03/2023	12/03/2023	3
1.6	Enfoque y método seguido	2	10/03/2023	12/03/2023	1
1.7	Planificación del proyecto	2	10/03/2023	12/03/2023	1
1.8	Índice preliminar de la memoria	2	10/03/2023	12/03/2023	1
2	PEC 2	13	13/03/2023	26/03/2023	**M+**
2.1	Redacción del estado del arte	12	14/03/2023	26/03/2023	4
3	PEC 3	53	27/03/2023	19/05/2023	**A**
3.1	Elección de la base de datos	5	31/03/2023	05/04/2023	6
3.2	Entendimiento de la base de datos	10	05/04/2023	15/04/2023	3
3.3	Métricas empleadas	12	15/04/2023	27/04/2023	5
3.4	Modelos analizados	14	01/05/2023	15/05/2023	5
3.5	Rendimiento de los modelos	13	05/05/2023	18/05/2023	5
3.6	Analisis exploratorio de los datos	10	05/04/2023	15/04/2023	4
3.7	Distribución de las clases	11	05/04/2023	16/04/2023	3
3.8	Resultados y conclusiones	15	15/05/2023	30/05/2023	5
3.9	Anexo	12	18/05/2023	30/05/2023	2
4	PEC 4	23	20/05/2023	12/06/2023	
5	PEC 5	9	13/06/2023	22/06/2023	
6	DEFENSA	0	06/07/2023	06/07/2023	

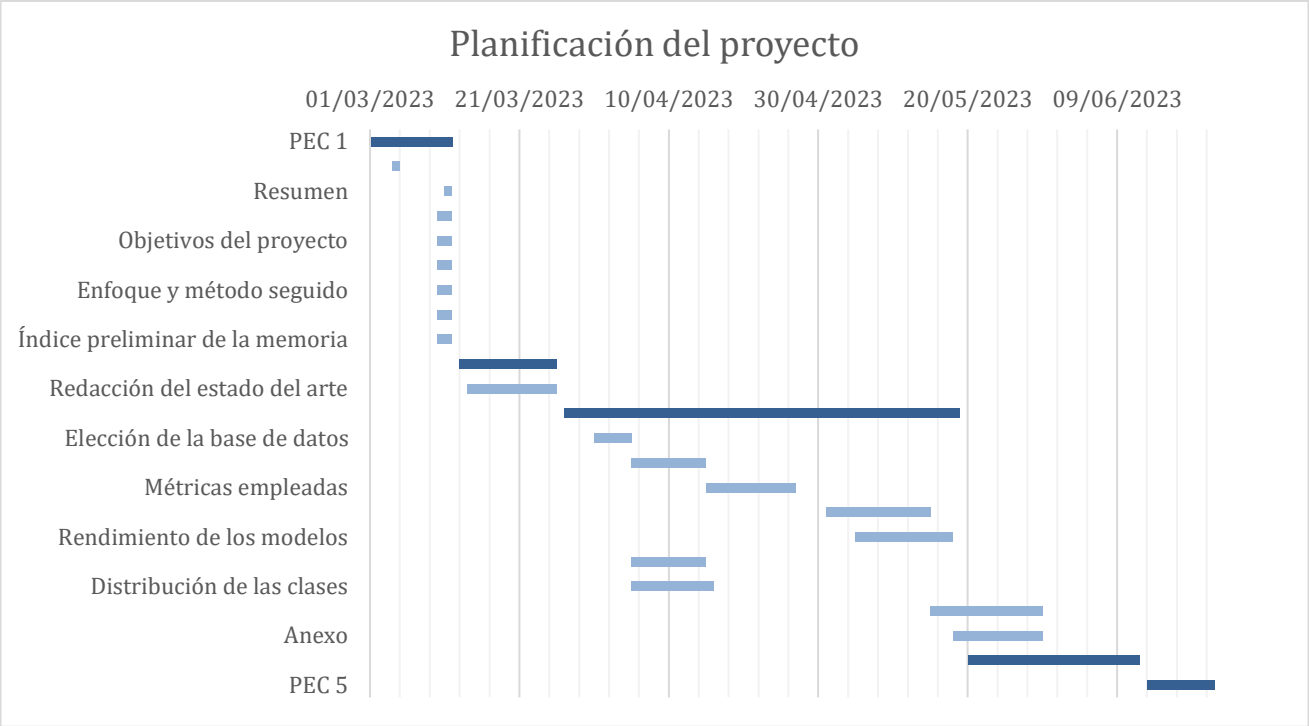

Figura 1: Planificación del proyecto

1.6. Breve sumario de productos obtenidos

En este apartado, se presenta un breve sumario de los productos obtenidos. A lo largo del Trabajo Fin de Máster (TFM), se ha explorado y analizado información relevante relacionada con técnicas y modelos utilizados en la detección y prevención de crimen financiero. A continuación, se resumen los principales hallazgos y productos derivados de estas investigaciones.

Conceptos y Definiciones Clave:
Se ha proporcionado una explicación detallada sobre los conceptos fundamentales relacionados con el crimen financiero, como el fraude, el lavado de dinero y la corrupción. Además, se han definido términos importantes como el sesgo, la asimetría y la matriz de correlación, que son fundamentales para comprender y analizar los datos asociados al crimen financiero.

Modelos de Aprendizaje Automático:
Se ha realizado un estudio exhaustivo de diferentes modelos de aprendizaje automático utilizados en la detección y predicción del crimen financiero. Se han explorado modelos como la Regresión Logística, K Vecinos Más Cercanos (KNN), Máquinas de Soporte Vectorial (SVM), Árboles de Decisión, Random Forest y XGBoost, y se han presentado explicaciones detalladas de cada uno de ellos, incluyendo sus ventajas y limitaciones.

Métricas de Evaluación:
Se han descrito y analizado diversas métricas utilizadas para evaluar el rendimiento de los modelos de crimen financiero. Entre ellas se encuentran el Área bajo la Curva ROC (ROC-AUC), Precisión, Recall y F1-Score. Se ha destacado la importancia de utilizar métricas apropiadas para conjuntos de datos desequilibrados, y se han proporcionado ejemplos de su aplicación en el contexto del crimen financiero.

Preprocesamiento de Datos:
Se ha abordado la importancia del preprocesamiento de datos en el análisis de crimen financiero. Se ha explicado la necesidad de técnicas como el escalado de características (Feature Scaling) y el uso de transformadores de potencia (PowerTransformer) para abordar la asimetría y mejorar la distribución de los datos.

Validación Cruzada y Optimización de Hiperparámetros:
Se ha destacado la importancia de la validación cruzada y se ha explicado el método de validación Stratified K-Fold como una técnica adecuada para conjuntos de datos desequilibrados. Además, se ha presentado GridSearchCV como una herramienta para optimizar los hiperparámetros de los modelos de crimen financiero, permitiendo encontrar la mejor configuración para obtener un rendimiento óptimo.

El sumario de los productos obtenidos en este análisis de crimen financiero proporciona una visión general de los conceptos, modelos y técnicas relevantes para la detección y prevención de crimen financiero. Estos productos pueden ser utilizados como base para el desarrollo de soluciones efectivas en la lucha contra el crimen financiero, ayudando a las organizaciones y entidades a protegerse de las amenazas y tomar decisiones informadas basadas en datos.

1.7. Breve descripción de los capítulos de la memoria

Este primer capítulo describe cómo se plantea el proyecto final, las principales motivaciones y la visión actual sobre el crimen financiero con el objetivo de profundizar en las últimas técnicas disponibles y conseguir mitigar los actos delictivos.

A continuación, el segundo capítulo describe el estado del arte del crimen financiero, comentando los antecedentes, técnicas empleadas, el impacto socioeconómico que este conlleva, así como el marco regulatorio. Además, se destacarán algunas herramientas y soluciones.

En el tercer capítulo es donde se explica por completo el desarrollo del proyecto, desde la elección de la base de datos utilizada para el análisis, así como los modelos probados.

Será después en el cuarto capítulo donde se detallen los resultados obtenidos aplicando los modelos descritos en el capítulo anterior, así como las conclusiones e investigaciones futuras.

Finalmente, también se encuentra un quinto capítulo a modo de glosario para definir aquellos términos que precisen más detalle para su entendimiento, un sexto donde está toda la bibliografía empleada para la elaboración del proyecto y el último capítulo, que corresponde con algunas gráficas obtenidas en la investigación del mejor modelo y algunas capturas del código empleado para el desarrollo del tercer capítulo.

2. Estado del arte

El capítulo del Estado del Arte en este trabajo proporciona una descripción general de la investigación y el conocimiento actual en el campo del crimen financiero, con un enfoque específico en el fraude en tarjetas de crédito y débito. Este capítulo tiene como objetivo evaluar y sintetizar críticamente la literatura existente sobre el tema, e identificar lagunas y limitaciones en el conocimiento actual. Los conocimientos adquiridos a partir de este análisis informarán el desarrollo de los modelos predictivos que se implementarán en este proyecto para prevenir y detectar el fraude financiero.

2.1. Introducción

El crimen financiero ligado a tarjetas de crédito y débito ha existido desde que se introdujeron las primeras tarjetas de crédito en la década de 1950. Sin embargo, el problema se ha vuelto cada vez más grave a medida que la tecnología ha avanzado y las transacciones electrónicas se han vuelto más comunes [2].

En la década de 1960, los delincuentes comenzaron a utilizar tecnologías de clonación para crear copias falsas de tarjetas de crédito. La clonación de tarjetas se convirtió en un problema importante en la década de 1970, cuando se introdujeron las primeras tarjetas con banda magnética. Los delincuentes utilizaban dispositivos de skimming [3] para robar información de la banda magnética de las tarjetas y luego crear copias falsas.

En la década de 1990, el fraude en línea se convirtió en un problema importante a medida que se popularizó el uso de Internet para las transacciones financieras. Los delincuentes comenzaron a utilizar técnicas de phishing y malware [4] para robar información de tarjetas de crédito y débito de los usuarios en línea.

En los últimos años, los delincuentes han utilizado tecnologías cada vez más avanzadas para cometer crímenes financieros relacionados con tarjetas de crédito y débito. Esto incluye la utilización de malware en terminales de puntos de venta, así como el uso de técnicas de ingeniería social [4], para engañar a los usuarios para que revelen su información de tarjeta.

En respuesta a estos desafíos, las instituciones financieras han adoptado medidas cada vez más rigurosas para prevenir y detectar el crimen financiero relacionado con tarjetas de crédito y débito. Esto incluye la implementación de medidas de seguridad avanzadas, como la autenticación de dos factores y el monitoreo en tiempo real de transacciones. También se han establecido leyes y regulaciones [6], [7], [8] para proteger a los consumidores y responsabilizar a los delincuentes.

A pesar de estos avances, el crimen financiero relacionado con tarjetas de crédito y débito sigue siendo un problema importante en todo el mundo. Los delincuentes siguen encontrando formas de eludir las medidas de seguridad y robar información de tarjetas, lo que significa que la lucha contra el crimen financiero relacionado con tarjetas de crédito y débito continuará siendo un desafío en el futuro.

2.2. Antecedentes

Desde hace varias décadas, los crímenes financieros en tarjetas de crédito han sido objeto de estudio por parte de diversos investigadores y expertos en el tema. Algunos de los trabajos anteriores que han abordado este tema incluyen:

- Investigaciones empíricas: Se han realizado diversas investigaciones para conocer la magnitud del problema de los crímenes financieros en tarjetas de crédito y las características de los delitos cometidos. Por ejemplo: "Detección del fraude con tarjetas de crédito en la era de las tecnologías disruptivas: Una revisión sistemática " realizado por la el Journal of King Saud University – Computer and Information Sciences, que examina los métodos utilizados por los delincuentes para cometer fraude con tarjetas de crédito en línea y propone medidas para prevenirlo [9].

- Estudios de casos: También se han llevado a cabo estudios de casos para analizar cómo se llevan a cabo los crímenes financieros en tarjetas de crédito y cómo pueden prevenirse. Por ejemplo, el estudio "Detección de Fraude en tarjetas de crédito: un caso de estudio" realizado por el Technocrats Institute of Tehcnology, Bhopla, en India que analiza un caso real de fraude y propone medidas para prevenir este tipo de delitos [10].

- Revisiones bibliográficas: Finalmente, se han realizado diversas revisiones bibliográficas para sintetizar la información existente sobre los crímenes financieros en tarjetas de crédito y las medidas para prevenirlos. Por ejemplo, la revisión "Prevención del fraude con tarjetas de crédito: una revisión de la literatura" realizada desde la Universidad de Salford - Manchester, que resume los principales enfoques y medidas preventivas propuestas en la literatura académica y técnica [11].

2.3. Técnicas

Las técnicas de crimen financiero más comunes utilizadas por los delincuentes para atacar tarjetas de crédito:

- Skimming: Los delincuentes instalan dispositivos electrónicos en los lectores de tarjetas en cajeros automáticos, terminales de pago o en gasolineras para copiar los datos de la banda magnética de la tarjeta. Los datos obtenidos se utilizan para crear una copia falsa de la tarjeta y realizar compras fraudulentas.

- Phishing: Los atacantes envían correos electrónicos o mensajes de texto falsificados que parecen legítimos, pero que en realidad dirigen al usuario a sitios web maliciosos donde se les pide que ingresen información sensible, como números de tarjeta de crédito o contraseñas. Con esta información, los delincuentes pueden realizar compras fraudulentas o incluso robar la identidad de la víctima.

- Malware: Los delincuentes pueden utilizar malware para instalar programas maliciosos en los dispositivos de los usuarios y robar información confidencial, como números de tarjeta de crédito. También pueden utilizar el malware para acceder a las cuentas bancarias en línea y realizar transferencias fraudulentas.

- Fraude en línea: Los delincuentes pueden utilizar sitios web falsificados o no seguros para obtener información de tarjetas de crédito de los usuarios. También pueden utilizar técnicas de ataque de fuerza bruta para descubrir los números de tarjeta de crédito o las contraseñas de los usuarios.

- Robo de identidad: Los delincuentes pueden utilizar información personal robada, como números de seguridad social o fechas de nacimiento, para abrir cuentas bancarias o solicitar préstamos en nombre de la víctima. También pueden utilizar esta información para solicitar tarjetas de crédito y realizar compras fraudulentas.

Es importante destacar que los usuarios pueden protegerse de estas técnicas utilizando contraseñas seguras, navegando solo en sitios web seguros y no compartiendo información confidencial con terceros no confiables. Además, es fundamental estar atentos a las transacciones en las tarjetas de crédito y reportar cualquier actividad sospechosa a la brevedad posible [12].

2.4. Impacto económico y social

El crimen financiero en tarjetas de crédito tiene un impacto significativo tanto a nivel económico como social. Esto influye en la evolución de las infraestructuras de las ciudades hacia Smart Cities [13]. En primer lugar, los individuos que son víctimas de estos delitos

pueden sufrir pérdidas económicas significativas, ya que los delincuentes pueden utilizar la información de sus tarjetas de crédito para realizar compras fraudulentas o para retirar dinero de sus cuentas bancarias. Esto puede tener un impacto negativo en el crédito de la persona afectada y puede tomar tiempo y recursos considerables para solucionar el problema.

Además, a nivel global, el crimen financiero en tarjetas de crédito también puede tener un impacto significativo en la economía. Los bancos y las empresas emisoras de tarjetas de crédito pueden sufrir pérdidas económicas significativas debido al fraude y al robo de identidad. Además, los comerciantes también pueden verse afectados, ya que pueden ser responsables de las pérdidas económicas que resulten de transacciones fraudulentas con tarjetas de crédito.

En cuanto al impacto social, el crimen financiero en tarjetas de crédito puede tener una afectación a la privacidad y seguridad de las personas. Cuando los delincuentes obtienen información personal y financiera de los individuos, esto puede resultar en un robo de identidad y puede poner en riesgo la información personal y financiera de la persona afectada. Además, esto puede generar desconfianza en los sistemas financieros y en las empresas que manejan información financiera, lo que puede afectar negativamente la percepción pública de la seguridad y la confianza en estos sistemas.

En resumen, el crimen financiero en tarjetas de crédito tiene un impacto económico y social significativo tanto a nivel individual como a nivel global [14]. Es importante que los individuos estén conscientes de los riesgos y tomen medidas preventivas para protegerse contra estos delitos, y que las empresas y los gobiernos implementen medidas para mejorar la seguridad y la protección de la información financiera.

2.5. Marco regulatorio

El marco regulatorio y las políticas públicas para prevenir y combatir el crimen financiero en tarjetas de crédito varían según el país o la región, pero en general, buscan proteger a los consumidores y las empresas de los riesgos asociados con estos delitos. Algunas de las principales políticas y regulaciones que se han implementado incluyen:

- Normativas de protección de datos: Las normativas de protección de datos, como el Reglamento General de Protección de Datos (RGPD) [15] en la Unión Europea y la Ley de Privacidad de la Información del Consumidor de California (CCPA) [16] en los Estados Unidos, buscan proteger la información personal de los consumidores y establecer reglas para su uso y almacenamiento por parte de las empresas.

- Responsabilidad de los proveedores de servicios financieros: Los proveedores de servicios financieros, como los bancos y las empresas emisoras de tarjetas de crédito, están sujetos a regulaciones específicas que establecen requisitos para la

protección de los datos de los clientes y para la prevención del fraude. Estas regulaciones también establecen responsabilidades específicas para los proveedores de servicios financieros en caso de que se produzcan violaciones de seguridad o fraudes.

- Identificación y autenticación de usuarios: Las políticas y regulaciones de identificación y autenticación de usuarios establecen requisitos para la verificación de la identidad de los usuarios y la autenticación de sus transacciones. Esto incluye la implementación de medidas de seguridad, como el uso de contraseñas y la autenticación de dos factores [16], para prevenir el uso no autorizado de tarjetas de crédito.

- Monitoreo y análisis de transacciones: Las empresas y los proveedores de servicios financieros suelen implementar sistemas de monitoreo y análisis de transacciones para detectar y prevenir fraudes en tiempo real. Esto incluye el análisis de patrones de comportamiento y el uso de herramientas de inteligencia artificial y aprendizaje automático para detectar actividades sospechosas.

- Cooperación entre agencias y entidades reguladoras: Las agencias gubernamentales y las entidades reguladoras trabajan en colaboración para prevenir y combatir el crimen financiero en tarjetas de crédito. Esto incluye el intercambio de información y la colaboración en investigaciones y en la implementación de políticas y regulaciones.

En definitiva, el marco regulatorio y las políticas públicas para prevenir y combatir el crimen financiero en tarjetas de crédito buscan proteger a los consumidores y las empresas de los riesgos asociados con estos delitos. Esto incluye la implementación de medidas de seguridad y de protección de datos, así como la cooperación entre agencias y entidades reguladoras para prevenir y combatir el fraude financiero.

2.6. Herramientas y soluciones

En los últimos años, se han desarrollado varias herramientas y soluciones tecnológicas para prevenir y detectar el crimen financiero en tarjetas de crédito. Algunas de las principales herramientas y soluciones incluyen:

- Autenticación de dos factores: La autenticación de dos factores requiere que los usuarios proporcionen dos formas de identificación, como una contraseña y un código de seguridad enviado a su teléfono móvil, antes de realizar una transacción. Esto ayuda a prevenir el uso no autorizado de tarjetas de crédito.

- Monitoreo de transacciones: Los sistemas de monitoreo de transacciones pueden detectar patrones y comportamientos sospechosos en el uso de tarjetas de crédito. Estos sistemas pueden alertar a los proveedores de servicios financieros o a los consumidores sobre transacciones sospechosas o no autorizadas.

- Análisis de datos: El análisis de datos y el uso de herramientas de inteligencia artificial y aprendizaje automático pueden ayudar a detectar patrones y comportamientos sospechosos en el uso de tarjetas de crédito. Esto puede permitir a los proveedores de servicios financieros tomar medidas preventivas antes de que se produzca un fraude.

- Tarjetas de crédito virtuales: Las tarjetas de crédito virtuales son números de tarjetas de crédito únicos generados para cada transacción en línea. Esto ayuda a prevenir el uso no autorizado de tarjetas de crédito al limitar la cantidad de transacciones que se pueden realizar con un número de tarjeta de crédito específico.

- Encriptación de datos: La encriptación de datos ayuda a proteger la información personal y financiera de los consumidores durante las transacciones en línea. Esto puede ayudar a prevenir la interceptación de datos de tarjetas de crédito por parte de delincuentes.

- Educación y concienciación: La educación y la concienciación son herramientas importantes para prevenir el crimen financiero en tarjetas de crédito. Los consumidores y las empresas deben estar informados sobre las técnicas utilizadas por los delincuentes y sobre las medidas que pueden tomar para proteger sus tarjetas de crédito.

En resumen, las herramientas y soluciones tecnológicas para prevenir y detectar el crimen financiero en tarjetas de crédito incluyen la autenticación de dos factores, el monitoreo de transacciones, el análisis de datos, las tarjetas de crédito virtuales, la encriptación de datos, la educación y la concienciación. Estas herramientas pueden ayudar a prevenir el fraude y proteger la información financiera de los consumidores y las empresas.

2.7. Conclusiones

En conclusión, este trabajo ha permitido realizar un estado del arte sobre el crimen financiero en tarjetas de crédito, destacando las diferentes técnicas utilizadas por los delincuentes para cometer estos delitos y el impacto económico y social que tienen sobre las personas y las empresas.

Se ha encontrado que el skimming, el phishing y el malware son algunas de las técnicas más utilizadas por los delincuentes, y que estas se han sofisticado con el tiempo, lo que hace

necesario el desarrollo constante de herramientas y soluciones tecnológicas para prevenir y detectar estos delitos.
Asimismo, se ha observado que el crimen financiero en tarjetas de crédito tiene un impacto económico y social significativo, que va desde la pérdida de dinero por parte de los individuos hasta la afectación de la confianza en los sistemas financieros.

En cuanto al marco regulatorio y las políticas públicas, se ha constatado la necesidad de fortalecer la regulación en cuanto a la protección de datos y la responsabilidad de los proveedores de servicios financieros, así como de fomentar la cooperación internacional en la prevención y el combate del crimen financiero en tarjetas de crédito.

Por último, se ha concluido que la investigación en el campo del crimen financiero en tarjetas de crédito debe continuar, especialmente en lo que respecta a la implementación y evaluación de nuevas herramientas y soluciones tecnológicas y a la adaptación de las políticas públicas y marcos regulatorios a los desafíos que plantean las nuevas técnicas de los delincuentes"

3. Desarrollo del proyecto

En este capítulo se analizará la fuente de datos utilizada para el análisis de los mejores algoritmos predictivos en cuanto a detección de fraude en transacciones bancarias asociadas a tarjetas de crédito se refiere.

En primer lugar, se describirá la base de datos de transacciones bancarias de tarjetas de crédito, el proceso de recopilación de datos, así como posibles desafíos.

Como continuación y dada la necesidad, se describirá la estrategia seguida para la limpieza y preprocesamiento de los datos, incluyendo si fuera necesario la eliminación de datos incorrectos o faltantes, la normalización y la transformación de variables.

Además, se procederá a realizar una descripción estadística de los datos, como medidas de dispersión y distribución de las variables relevantes. Para ello, será útil llevar a cabo una visualización de los datos mediante gráficos y diagramas que nos ayuden a identificar patrones, relaciones o anomalías. De aquí, podremos destacar también características importantes o variables relevantes para el análisis posterior.

Por último, en una subsección se realizará la selección y aplicación de modelos de inteligencia artificial. Se describirán los modelos empleados, una justificación de su uso y su adecuación para abordar el problema de análisis de las transacciones bancarias de tarjetas de crédito. Esto incluirá detalles sobre los parámetros y configuraciones utilizados para cada modelo.

3.1. Elección de la base de datos

En general La obtención de información abierta o pública sobre transacciones bancarias de tarjetas de crédito puede ser sumamente complicada debido a diversas razones:

Confidencialidad y privacidad: Las transacciones bancarias de tarjetas de crédito implican datos altamente confidenciales y sensibles, como números de tarjetas, fechas de vencimiento, códigos de seguridad y detalles de transacciones específicas. Por razones de seguridad y privacidad, las instituciones financieras suelen tener estrictas políticas y regulaciones para proteger esta información y no la hacen públicamente disponible.

Normativas y leyes: Las transacciones bancarias están sujetas a regulaciones y leyes específicas en cada país para proteger los derechos de los consumidores y prevenir actividades delictivas, como el fraude financiero. Estas regulaciones, como la Ley de Protección de Datos y la Ley de Privacidad, restringen el acceso y la divulgación de información personal y financiera, lo que dificulta aún más obtener datos abiertos de este tipo.

Acuerdos comerciales y confidencialidad empresarial: Las instituciones financieras, las compañías de tarjetas de crédito y otros actores del sector suelen tener acuerdos comerciales y acuerdos de confidencialidad con sus clientes y socios comerciales. Estos acuerdos protegen la información de los clientes y los datos comerciales sensibles, lo que limita la disponibilidad de datos abiertos relacionados con transacciones bancarias de tarjetas de crédito.

En consecuencia, obtener una base de datos abierta y pública que contenga información real de transacciones bancarias de tarjetas de crédito puede ser extremadamente difícil debido a los desafíos mencionados anteriormente. Como alternativa, es común utilizar conjuntos de datos sintéticos o simulados que intentan representar características realistas de las transacciones reales sin comprometer la privacidad y la confidencialidad de los datos reales.

En el caso de la base de datos "Credit Card Fraud Detection" de Kaggle, aunque no representa datos reales, proporciona un conjunto de datos sintéticos creado para simular transacciones bancarias de tarjetas de crédito, incluyendo transacciones fraudulentas y no fraudulentas. Esta opción ha sido la escogida para poder analizar diferentes modelos predictivos con el afán de buscar cuál de las opciones es más precisa sin violar la privacidad de los datos reales de los clientes o incurrir en infracciones legales.

3.1.1. Entendiendo la base de datos

El conjunto de datos se obtiene del sitio web de Kaggle y consta de un total de 284,807 transacciones; de estas, 492 son fraudulentas. Dado a estar altamente desequilibrado, ha sido necesario tratarlo antes de intentar modelar futuros comportamientos para bases de datos similares.

El conjunto de datos contiene transacciones realizadas con tarjetas de crédito en septiembre de 2013 por titulares de tarjetas europeos. Este conjunto de datos presenta transacciones que ocurrieron en dos días, donde tenemos 492 fraudes de un total de 284,807 transacciones. Como se mencionaba anteriormente, el conjunto de datos está altamente desequilibrado, la clase positiva (fraudes) únicamente representa el 0.172% de todas las transacciones.

Además, por las causas expuestas en el subapartado anterior, solo contiene variables de entrada numéricas resultado de una transformación PCA (Principal Component Analysis, por sus siglas en inglés) [17].

La transformación PCA es una técnica que se utiliza para reducir la dimensionalidad de un conjunto de datos. Básicamente, lo que hace es encontrar una nueva representación de los datos mediante la combinación lineal de las variables originales.

El objetivo principal de la transformación PCA es encontrar las direcciones en las que los datos tienen más variabilidad. Estas direcciones se llaman componentes principales.

Al combinar las variables originales en estas nuevas direcciones, podemos reducir la cantidad de variables necesarias para describir los datos, manteniendo la mayor cantidad posible de información.

La idea principal detrás de la transformación PCA es que las variables originales pueden estar correlacionadas entre sí, lo que significa que contienen información redundante. Al combinarlas en componentes principales, podemos capturar la mayor parte de la variabilidad de los datos con menos variables.

Desafortunadamente, debido a problemas de confidencialidad, la base de datos no proporciona las características originales y más información de fondo sobre los datos. Las características V1, V2, ... V28 son los componentes principales obtenidos con PCA, las únicas características que no se han transformado con PCA son 'Time' y 'Amount'. La característica 'Time' contiene los segundos transcurridos entre cada transacción y la primera transacción en el conjunto de datos. La característica 'Amount' es el monto de la transacción, esta característica se puede utilizar, por ejemplo, para un aprendizaje dependiente del costo sensible. La característica 'Class' es la variable de respuesta y toma el valor 1 en caso de fraude y 0 en caso contrario.

Dada la relación de desequilibrio de clases, se medirá la precisión utilizando el Área Bajo la Curva de Precisión-Recall (AUPRC) ya que la precisión de la matriz de confusión no es significativa para la clasificación desequilibrada.

esta métrica proporciona una evaluación más precisa y significativa del rendimiento de un modelo de detección de fraude en transacciones bancarias cuando existe una disparidad entre la cantidad de casos positivos (fraudes) y negativos (transacciones legítimas).

En escenarios de desequilibrio de clases, es común que la mayoría de las transacciones sean legítimas, mientras que los casos de fraude son una pequeña proporción del conjunto de datos. Por ejemplo, en el caso de las transacciones bancarias, la cantidad de transacciones legítimas supera con creces el número de casos de fraude.

3.1.2. Métricas empleadas

La evaluación de modelos es una etapa crítica en el proceso de modelización, ya que nos permite medir y comprender el rendimiento de un modelo predictivo o clasificador. Las métricas de evaluación desempeñan un papel fundamental en este proceso, ya que nos proporcionan medidas objetivas y cuantitativas sobre qué tan bien está funcionando nuestro modelo en términos de precisión, generalización y capacidad para capturar patrones en los datos.

La importancia de las métricas radica en varios aspectos. En primer lugar, nos permiten comparar y seleccionar entre diferentes modelos o algoritmos. Al utilizar métricas consistentes y apropiadas, podemos identificar cuál de los modelos candidatos se ajusta mejor a nuestros datos y objetivos específicos.

Además, las métricas nos ayudan a comprender el desempeño de un modelo en diferentes aspectos. Por ejemplo, la precisión nos indica la proporción de predicciones correctas realizadas por el modelo, mientras que el recall nos muestra qué tan bien el modelo puede capturar las instancias positivas. Estas métricas nos brindan información valiosa sobre las fortalezas y debilidades del modelo en relación con los diferentes tipos de errores que puede cometer.

Las métricas también son fundamentales para evaluar el impacto de los umbrales de decisión en un modelo. En muchos casos, los modelos de clasificación requieren establecer un umbral de probabilidad o confianza para tomar decisiones. Las métricas nos permiten evaluar cómo cambia el rendimiento del modelo a medida que ajustamos este umbral, lo cual es esencial para tomar decisiones informadas sobre el punto de corte óptimo.

- ROC-AUC (Área bajo la curva ROC): La curva ROC (Receiver Operating Characteristic) es una representación gráfica que muestra el rendimiento de un modelo de clasificación binaria a medida que se ajusta el umbral de decisión. El área bajo la curva (AUC) es una métrica que cuantifica la capacidad del modelo para distinguir entre clases positivas y negativas. Un valor de AUC cercano a 1 indica un modelo con un rendimiento excelente, mientras que un valor cercano a 0.5 indica un rendimiento similar al azar [18].

- Precisión: La precisión es una métrica que mide la proporción de predicciones positivas correctas (verdaderos positivos) en relación con el total de predicciones positivas realizadas por el modelo. Se calcula como TP / (TP + FP), donde TP son los verdaderos positivos y FP son los falsos positivos. La precisión proporciona información sobre la exactitud del modelo al predecir las instancias positivas [18].

- F1-score: El F1-score es una medida que combina la precisión y el recall (también conocido como sensibilidad) en un solo valor. Se calcula como 2 * (precisión * recall) / (precisión + recall). El F1-score proporciona una medida equilibrada del rendimiento del modelo, especialmente cuando hay un desequilibrio entre las clases [19].

- Recall (sensibilidad): El recall es una métrica que mide la proporción de instancias positivas correctamente identificadas por el modelo en relación con el número total de instancias positivas en los datos de prueba. Se calcula como TP / (TP + FN), donde TP son los verdaderos positivos y FN son los falsos negativos. El recall es útil cuando es importante capturar la mayoría de las instancias positivas y minimizar los falsos negativos [20].

3.2. Modelos analizados

En primer lugar, es interesante destacar que los modelos de aprendizaje automático se pueden clasificar en dos categorías principales: modelos supervisados y modelos no supervisados. Estas categorías representan diferentes enfoques para el aprendizaje a partir de los datos y tienen aplicaciones en diversos problemas y escenarios.

Modelos Supervisados:

Los modelos supervisados se basan en datos etiquetados, donde cada instancia del conjunto de datos de entrenamiento está asociada con una etiqueta o clase predefinida. El objetivo de los modelos supervisados es aprender una función que mapee las variables predictoras (características) a las etiquetas correspondientes. Estos modelos se utilizan para problemas de clasificación y regresión.

En la clasificación, el objetivo es asignar una instancia a una o varias clases predefinidas. Por ejemplo, un modelo de clasificación puede predecir si un correo electrónico es spam o no spam, o si una imagen contiene un gato o un perro. Los modelos de clasificación supervisados comunes incluyen la Regresión Logística, los Árboles de Decisión, las Máquinas de Vectores de Soporte (SVM) y las Redes Neuronales.

En la regresión, el objetivo es predecir un valor numérico continuo. Por ejemplo, un modelo de regresión puede predecir el precio de una casa en función de sus características. Algunos modelos supervisados utilizados para la regresión son la Regresión Lineal, los Bosques Aleatorios y las Redes Neuronales.

La principal diferencia entre los modelos supervisados radica en los algoritmos y las técnicas utilizadas para aprender la función objetivo a partir de los datos de entrenamiento. Cada modelo tiene sus propias fortalezas y debilidades, y la elección del modelo depende del problema específico y los requisitos del dominio.

Modelos No Supervisados:

En contraste, los modelos no supervisados se utilizan cuando no se dispone de etiquetas o clases predefinidas en los datos de entrenamiento. Estos modelos exploran la estructura inherente en los datos y buscan patrones, grupos o relaciones ocultas sin la guía de etiquetas externas.

El objetivo de los modelos no supervisados es encontrar patrones interesantes y estructuras subyacentes en los datos sin necesidad de un conocimiento previo. Estos modelos son útiles para la segmentación de datos, la reducción de dimensionalidad, la detección de anomalías y la exploración de datos.

Algunos ejemplos comunes de modelos no supervisados son el Análisis de Componentes Principales (PCA), el Agrupamiento K-Means, las Redes Neuronales Autoencoders y los Algoritmos de Asociación.

A diferencia de los modelos supervisados, los modelos no supervisados no requieren etiquetas o clases predefinidas en los datos de entrenamiento. En su lugar, se utilizan técnicas estadísticas y algoritmos de aprendizaje para encontrar patrones y estructuras emergentes en los datos.

En definitiva, los modelos supervisados se basan en datos etiquetados y se utilizan para problemas de clasificación y regresión, mientras que los modelos no supervisados se enfocan en descubrir patrones y estructuras en los datos sin la guía de etiquetas externas. Ambos enfoques tienen sus propias aplicaciones y desafíos, y la elección del modelo depende de la naturaleza del problema y los objetivos del análisis de datos.

En cuanto a lo que a este trabajo concierne y como se ha detallado en la explicación de la base de datos, los modelos empleados son puramente supervisados. Sin embargo, para la obtención de las variables que por motivos de confidencialidad se desconocen, ha sido empleada la técnica PCA correspondiente al segundo grupo de modelos.

3.2.1. Regresión Logística

La Regresión Logística [21] es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación, donde el objetivo es predecir la pertenencia a una o varias categorías discretas. Aunque el nombre incluye "regresión", la Regresión Logística se utiliza para problemas de clasificación, no para problemas de regresión.

El objetivo principal de la Regresión Logística es modelar la probabilidad de que una instancia pertenezca a una clase específica. Dado un conjunto de variables predictoras (características), la Regresión Logística estima la probabilidad de que la instancia pertenezca a la clase positiva utilizando una función logística, también conocida como función sigmoide.

La función sigmoide [22] toma cualquier valor de entrada y lo transforma en un valor entre 0 y 1. Esta transformación se utiliza para modelar la probabilidad condicional de que una instancia pertenezca a la clase positiva dado un conjunto de variables predictoras. La función sigmoide se define como:

$$p = 1/(1 + e^{-z})$$

Donde p es la probabilidad estimada, e es la base del logaritmo natural y z es una combinación lineal de las variables predictoras ponderadas por los coeficientes del modelo. La ecuación se puede expresar como:

$$z = b_0 + b_1x_1 + b_2x_2 + \cdots + b_2x_2$$

Donde b_0, b_1, $b_2 \ldots$ b_n son los coeficientes del modelo y x_1, x_2, …, x_n son las variables predictoras.

Para ajustar un modelo de Regresión Logística, se utiliza el método de máxima verosimilitud para estimar los coeficientes que maximizan la probabilidad conjunta de los datos observados. Una vez ajustado el modelo, se pueden realizar predicciones calculando la probabilidad estimada y aplicando un umbral para asignar la clase correspondiente.

La Regresión Logística tiene varias ventajas. Es un modelo lineal que es relativamente rápido de entrenar y aplicar, incluso en conjuntos de datos grandes. Además, proporciona una interpretación directa de los coeficientes del modelo, lo que permite comprender cómo cada variable afecta la probabilidad de pertenecer a una clase específica.

Sin embargo, la Regresión Logística también tiene algunas limitaciones. Es un modelo lineal y asume una relación lineal entre las variables predictoras y la probabilidad de pertenencia a la clase positiva. Si existe una relación no lineal, puede ser necesario aplicar transformaciones a las variables predictoras o utilizar técnicas más avanzadas.

3.2.2. El algoritmo KNN (k-nearest neighbours)

El algoritmo KNN (k-nearest neighbors) es un método de aprendizaje supervisado utilizado para la clasificación y regresión. Es un algoritmo simple pero efectivo que se basa en el principio de que las instancias similares tienden a estar en la misma clase o tener valores similares.

En el caso de la clasificación, el algoritmo KNN se basa en encontrar los k vecinos más cercanos a una instancia de prueba en el espacio de características. Estos vecinos se determinan calculando la distancia entre la instancia de prueba y las instancias de entrenamiento en función de sus características. La distancia más comúnmente utilizada es la distancia euclidiana, pero también se pueden utilizar otras medidas de distancia.

Una vez que se han identificado los k vecinos más cercanos, el algoritmo KNN asigna la clase más frecuente entre los vecinos como la clase de la instancia de prueba. En el caso de problemas de clasificación binaria, esto implica asignar la clase mayoritaria de los vecinos. En problemas de clasificación multiclase, se puede utilizar una estrategia de votación para determinar la clase asignada.

En cuanto a la regresión, el algoritmo KNN se utiliza para predecir un valor numérico continuo en función de los valores de las instancias vecinas más cercanas. En este caso, en lugar de asignar una clase, se puede tomar como predicción el valor medio o la mediana de los valores de las instancias vecinas.

El parámetro k en el algoritmo KNN representa el número de vecinos más cercanos que se tienen en cuenta para realizar la clasificación o la regresión. La elección adecuada de k puede ser crucial y depende del conjunto de datos y el problema específico. Un valor k demasiado pequeño puede llevar a un sobreajuste (overfitting), mientras que un valor k demasiado grande puede llevar a un subajuste (underfitting).

El algoritmo KNN tiene algunas ventajas, como su simplicidad y facilidad de implementación. Además, es no paramétrico, lo que significa que no hace suposiciones sobre la distribución de los datos. Sin embargo, también tiene algunas limitaciones, como su sensibilidad a la presencia de atributos irrelevantes y la necesidad de almacenar y calcular distancias para todas las instancias de entrenamiento, lo que puede ser costoso computacionalmente en conjuntos de datos grandes [23].

3.2.3. SVM (Support Vector Machine)

El algoritmo SVM (Support Vector Machine) es un método de aprendizaje supervisado utilizado para la clasificación y regresión. Su objetivo principal es encontrar el hiperplano óptimo que mejor separa las instancias de diferentes clases en el espacio de características.

En el caso de la clasificación, el SVM busca encontrar el hiperplano que maximice la distancia (margen) entre las instancias de diferentes clases más cercanas a él, llamadas vectores de soporte. Estos vectores de soporte son las instancias críticas que influyen en la posición y orientación del hiperplano de decisión. El hiperplano de decisión es una superficie de separación lineal que divide el espacio de características en regiones correspondientes a las diferentes clases.

Sin embargo, en algunos casos, las instancias pueden no ser linealmente separables en el espacio de características. Para abordar este problema, el SVM utiliza una técnica llamada "kernel trick". Un kernel es una función que permite mapear las instancias a un espacio de mayor dimensionalidad donde es más probable que sean linealmente separables. Al aplicar el kernel, se puede encontrar un hiperplano de decisión no lineal en el espacio de características original.

La ecuación matemática para el hiperplano de decisión en su forma más básica se puede expresar de la siguiente manera:

$$w^T * x + b = 0$$

Donde:

- w es el vector normal al hiperplano.
- x es el vector de características de una instancia de entrada.
- b es el sesgo o término de sesgo.

La expresión $w^T * x$ representa el producto escalar entre w y x.

Para realizar la clasificación, se evalúa la posición de una instancia de prueba x en relación con el hiperplano de decisión calculando el lado izquierdo de la ecuación. Si el resultado es mayor que 0, se asigna a una clase positiva, y si es menor que 0, se asigna a una clase negativa.

En el caso de SVM con kernel, la ecuación se modifica para incluir el mapeo no lineal a un espacio de mayor dimensionalidad. La forma general de la ecuación es la siguiente:

$$\sum(\alpha_i * y_i * K(x_i, x)) + b = 0$$

Donde:
α_i es un coeficiente asociado a cada instancia de entrenamiento.
y_i es la etiqueta de clase correspondiente a cada instancia de entrenamiento.
$K(x_i, x)$ es la función kernel que mapea las instancias de entrenamiento x_i y la instancia de prueba x a un espacio de mayor dimensionalidad.

La función kernel permite el uso de una variedad de transformaciones no lineales en el algoritmo SVM, como el kernel lineal, el kernel polinomial y el kernel gaussiano (RBF).

Estas ecuaciones matemáticas son fundamentales en el algoritmo SVM y describen la relación entre las instancias de entrenamiento, el hiperplano de decisión y el proceso de clasificación.

El SVM también puede manejar problemas de clasificación multiclase mediante la técnica de "one-vs-all" o "one-vs-one". En el enfoque "one-vs-all", se entrena un clasificador SVM separado para cada clase, donde se considera una clase como positiva y el resto como negativas. En el enfoque "one-vs-one", se entrenan clasificadores SVM binarios para cada par de clases posibles y se utiliza una estrategia de votación para determinar la clase asignada.

En cuanto a la regresión, el SVM también se puede utilizar para predecir valores numéricos continuos. En este caso, el SVM busca encontrar una función de regresión que se ajuste a los datos de entrenamiento y minimice el error de predicción. El enfoque de regresión SVM se basa en encontrar un conjunto de vectores de soporte que definan una función lineal o no lineal que se ajuste a los datos de entrenamiento.

3.2.4. Árbol de decisión (Decision Tree)

Un árbol de decisión es un modelo de aprendizaje supervisado que se utiliza para resolver problemas de clasificación y regresión. Este modelo se basa en la estructura de un árbol, donde cada nodo interno representa una característica o atributo, cada rama representa una decisión o regla y cada hoja representa una clase o un valor numérico.

El proceso de construcción de un árbol de decisión comienza con un conjunto de datos de entrenamiento que contiene instancias etiquetadas. El objetivo es dividir el conjunto de datos de manera óptima en cada nodo interno, de modo que las instancias que caigan en un mismo nodo compartan características similares. Para lograr esto, se utilizan medidas de impureza o ganancia de información, como la entropía o el índice de Gini, para evaluar la calidad de las divisiones.

En cada nodo interno, se selecciona la característica que mejor discrimina las clases o reduce la impureza del conjunto de datos. Se realiza una partición basada en los valores posibles de esa característica, creando ramas que conducen a otros nodos internos o a hojas. Este proceso se repite recursivamente hasta que se alcanza un criterio de parada, como alcanzar una profundidad máxima, un número mínimo de instancias en un nodo o una impureza mínima.

Una vez que el árbol de decisión ha sido construido, se puede utilizar para clasificar nuevas instancias o predecir valores numéricos. Para clasificar una instancia, se sigue el camino desde la raíz hasta una hoja a través de las ramas, tomando las decisiones de acuerdo con los valores de las características en cada nodo interno. La clase o valor numérico asociado a la hoja en la que se llega se asigna como la predicción.

Los árboles de decisión tienen varias ventajas. Son fáciles de entender e interpretar, ya que las reglas y decisiones se pueden visualizar de manera intuitiva. Además, son capaces de manejar tanto características numéricas como categóricas y pueden manejar conjuntos de datos grandes y de alta dimensionalidad.

Sin embargo, los árboles de decisión también tienen algunas limitaciones. Pueden ser propensos al sobreajuste si se construyen demasiado complejos, lo que puede llevar a un rendimiento deficiente en datos no vistos. Además, los árboles de decisión pueden ser sensibles a pequeños cambios en los datos de entrenamiento y pueden ser menos efectivos en problemas con relaciones no lineales.

Para abordar estas limitaciones, se han propuesto diversas técnicas, como la poda del árbol, el ensamblado de árboles (como el bosque aleatorio) y los gradient boosting, que combinan múltiples árboles para mejorar la precisión y la generalización del modelo [25].

Una de las ventajas del SVM es que puede manejar conjuntos de datos de alta dimensionalidad y es eficiente en el uso de memoria debido a que solo utiliza los vectores de soporte para definir el hiperplano de decisión. Además, el SVM es efectivo incluso en casos donde las clases están parcialmente superpuestas en el espacio de características. Sin embargo, el SVM puede ser computacionalmente costoso en conjuntos de datos grandes y puede requerir una cuidadosa selección del kernel y los parámetros asociados [24].

3.2.5. Bosque Aleatorio (Random Forest)

Random Forest (bosque aleatorio) es un algoritmo de aprendizaje supervisado que combina múltiples árboles de decisión para realizar clasificación o regresión. Cada árbol de decisión en el bosque se construye de manera independiente utilizando una muestra aleatoria de los datos de entrenamiento y realizando selecciones aleatorias de características.

El proceso de construcción de un bosque aleatorio se puede resumir en los siguientes pasos:

I. Seleccionar una muestra aleatoria con reemplazo (bootstrapping) del conjunto de datos de entrenamiento. Esta muestra se utilizará para construir un árbol de decisión.

II. Para cada nodo en el árbol de decisión, se selecciona un subconjunto aleatorio de características. Esto se hace para garantizar que cada árbol tenga diversidad en las características consideradas.

III. Se construye un árbol de decisión en base a la muestra de datos y las características seleccionadas. Se utiliza algún criterio de división, como la ganancia de información o la reducción de impureza, para determinar cómo se divide el conjunto de datos en cada nodo.

IV. Los pasos I a III se repiten para construir un número predeterminado de árboles en el bosque.

Una vez que se han construido todos los árboles del bosque, el bosque aleatorio puede realizar clasificación o regresión utilizando los siguientes enfoques:

- Para clasificación: cada árbol en el bosque emite una votación y la clase más frecuente entre todos los árboles se selecciona como la predicción final.

- Para regresión: cada árbol en el bosque predice un valor y los valores predichos se promedian para obtener la predicción final.

La idea clave detrás de un bosque aleatorio es que la combinación de múltiples árboles de decisión reduce el sobreajuste y mejora la generalización. Cada árbol en el bosque se

entrena de forma independiente y puede aprender diferentes aspectos de los datos, lo que ayuda a capturar la complejidad del problema.

Los bosques aleatorios tienen varias ventajas. Son muy efectivos en problemas de clasificación y regresión, pueden manejar conjuntos de datos grandes y de alta dimensionalidad, y proporcionan una medida de la importancia de las características. Además, son menos propensos al sobreajuste en comparación con un solo árbol de decisión [26].

3.2.6. XGBoost (Extreme Gradient Boosting)

GBoost (Extreme Gradient Boosting) es un algoritmo de aprendizaje automático que utiliza el enfoque de impulso (boosting) para construir modelos de clasificación y regresión de alta precisión. Es conocido por su capacidad para manejar conjuntos de datos grandes, alta velocidad de entrenamiento y excelentes resultados en competencias de aprendizaje automático.

La idea principal detrás de XGBoost es combinar múltiples modelos débiles, como árboles de decisión, en un modelo fuerte y más preciso. El algoritmo funciona de la siguiente manera:

I. Se construye un árbol de decisión inicial como el primer modelo base.

II. Se calcula el error residual entre las predicciones del modelo actual y las etiquetas verdaderas en el conjunto de datos de entrenamiento.

III. Se construye un nuevo árbol de decisión para capturar los errores residuales del modelo actual. El objetivo es ajustar el modelo a los errores pasados.

IV. Se actualizan las predicciones agregando las predicciones del nuevo árbol al modelo actual.

Los pasos II a IV se repiten hasta que se alcance un número predeterminado de árboles o se cumpla un criterio de parada.

Una de las características distintivas de XGBoost es su enfoque en la optimización del modelo. Utiliza una función objetivo regularizada que combina una medida de error y un término de penalización para evitar el sobreajuste. Además, utiliza técnicas de poda y recorte (pruning) para mejorar la generalización y la eficiencia del modelo.

Otra característica importante de XGBoost es la capacidad de manejar características numéricas y categóricas, así como datos faltantes. Puede manejar automáticamente características categóricas codificándolas de forma adecuada y utilizar estrategias específicas para manejar valores faltantes en los datos.

XGBoost también proporciona una función de importancia de características que ayuda a identificar las características más relevantes para el modelo. Esto permite realizar análisis de características y selección para comprender qué características contribuyen más a las predicciones [27].

3.3. Rendimiento de los modelos

Para analizar el rendimiento de los modelos, es importante conocer los fenómenos de overfitting y underfitting.

El overfitting (sobreajuste) ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos o datos de prueba. En otras palabras, el modelo memoriza los datos de entrenamiento en lugar de capturar los patrones subyacentes en los datos. Como resultado, el modelo puede tener un rendimiento excelente en los datos de entrenamiento, pero se desempeña mal en datos no vistos.

Los signos comunes de overfitting incluyen una precisión o puntaje alto en los datos de entrenamiento, pero una precisión o puntaje bajo en los datos de prueba o validación. El modelo puede ser demasiado complejo, con muchos parámetros o características, lo que le permite ajustarse a ruido o peculiaridades únicas de los datos de entrenamiento que no son representativas de la población en general.

Por otro lado, el underfitting (subajuste) ocurre cuando un modelo no es lo suficientemente complejo como para capturar los patrones subyacentes en los datos de entrenamiento. El modelo no se ajusta lo suficiente a los datos y, como resultado, su rendimiento tanto en los datos de entrenamiento como en los datos de prueba puede ser deficiente.

Los signos comunes de underfitting incluyen una baja precisión o puntaje en los datos de entrenamiento y una baja precisión o puntaje en los datos de prueba o validación. El modelo puede ser demasiado simple o tener pocos parámetros o características para capturar la complejidad de los datos.

El objetivo es encontrar un equilibrio entre el overfitting y el underfitting, donde el modelo se ajuste adecuadamente a los datos de entrenamiento y generalice bien a nuevos datos. Esto se puede lograr mediante técnicas como la selección de características relevantes, la regularización para penalizar la complejidad del modelo y la validación cruzada para evaluar el rendimiento en conjuntos de datos no vistos.

El overfitting y el underfitting son conceptos importantes que considerar al desarrollar modelos de aprendizaje automático, ya que afectan directamente el rendimiento y la capacidad de generalización del modelo. En general, se busca evitar tanto el overfitting como el underfitting para obtener un modelo que se ajuste bien y pueda hacer predicciones precisas en datos nuevos o no vistos [28].

Para la realización de este trabajo, se han aplicado tres técnicas de oversampling comunes: Random Oversampling, SMOTE (Synthetic Minority Over-sampling Technique) y ADASYN (Adaptive Synthetic Sampling).

Random Oversampling:
Random Oversampling es una técnica simple de oversampling en la que se duplican aleatoriamente ejemplos de la clase minoritaria para equilibrar la distribución de clases en un conjunto de datos desequilibrado. Al agregar copias adicionales de ejemplos de la clase minoritaria, se aumenta su representación y se evita el sesgo hacia la clase mayoritaria. Esta técnica puede generar problemas de sobreajuste si se duplican demasiados ejemplos y no se controla adecuadamente [29].

SMOTE (Synthetic Minority Over-sampling Technique):
SMOTE es una técnica de oversampling que genera ejemplos sintéticos de la clase minoritaria al tomar muestras de los vecinos cercanos en el espacio de características. En lugar de duplicar ejemplos existentes, SMOTE crea ejemplos sintéticos interpolando características entre ejemplos de la clase minoritaria cercanos. Esto ayuda a superar el problema del sobreajuste y mejora la capacidad del modelo para capturar la distribución subyacente de los datos de la clase minoritaria [30].

ADASYN (Adaptive Synthetic Sampling):
ADASYN es una técnica de oversampling que se basa en SMOTE, pero introduce una adaptación basada en densidad. ADASYN genera ejemplos sintéticos de la clase minoritaria en función de la densidad de los ejemplos en su vecindario. Pone más énfasis en la generación de ejemplos en regiones de la clase minoritaria con menor representación, lo que ayuda a abordar el problema de desequilibrio en áreas de mayor dificultad [31].

Además de todas estas técnicas, para ahondar más en la evaluación del rendimiento y seleccionar los hiperparámetros óptimos se ha empleado la validación cruzada.

Dos componentes clave en la validación cruzada son los métodos StratifiedKFold y GridSearchCV.

StratifiedKFold:
StratifiedKFold es un método de particionamiento de datos utilizado en la validación cruzada. A diferencia de la validación cruzada tradicional, donde se seleccionan muestras aleatorias para cada partición, StratifiedKFold asegura que cada partición tenga una distribución similar de clases. Esto es especialmente útil cuando se trabaja con conjuntos de datos desequilibrados, donde una clase puede tener muchas más muestras que otras. StratifiedKFold garantiza que cada partición contenga una proporción representativa de cada clase, lo que ayuda a evitar el sesgo en la evaluación del modelo [32].

GridSearchCV:
GridSearchCV es una técnica utilizada para encontrar los hiperparámetros óptimos de un modelo de aprendizaje automático mediante la búsqueda exhaustiva de una cuadrícula de combinaciones posibles. Permite especificar un conjunto de hiperparámetros y sus valores correspondientes, y evalúa el rendimiento del modelo utilizando validación cruzada en cada combinación. GridSearchCV realiza una búsqueda sistemática en la cuadrícula de hiperparámetros y devuelve el conjunto de valores que produce el mejor rendimiento según una métrica definida, como la precisión o el puntaje F1 [33].

3.4. Análisis exploratorio de los datos

Los parámetros incluidos en la base de datos, por ser una base de datos oficial ya tenían tratados los valores no disponibles (NAs) y habían sido estructurados en concordancia con el PCA mencionado anteriormente.

Analizando cada uno de los parámetros (ver Anexo I) observamos que todos son float64, salvo la variable diferenciadora de la clase de cuenta que estamos tratando, es decir, fraudulenta o no fraudulenta.

Esto, además, confirmaba que los posibles outliers ya habían sido tratados

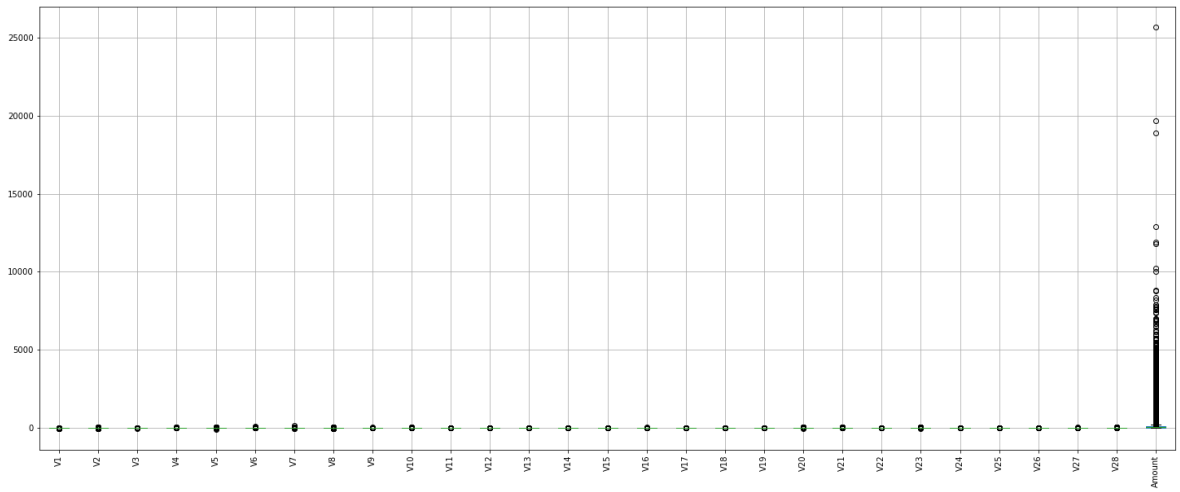

Figura 2: Asimetrías en la muestra

3.4.1. Distribución de las clases

En el campo de la detección de fraude en tarjetas de crédito, uno de los aspectos cruciales a tener en cuenta es la distribución de las clases fraudulentas y no fraudulentas. En un conjunto de datos típico como el analizado, la mayoría de las transacciones son legítimas y no fraudulentas, mientras que solo una pequeña proporción representa transacciones

fraudulentas. Esta asimetría en la distribución de clases plantea desafíos significativos para desarrollar modelos de aprendizaje automático efectivos

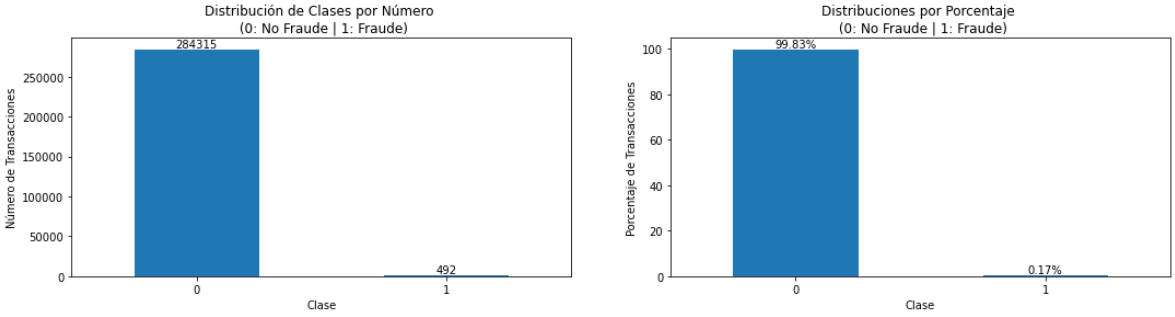

Figura 3: Distribución de clases I

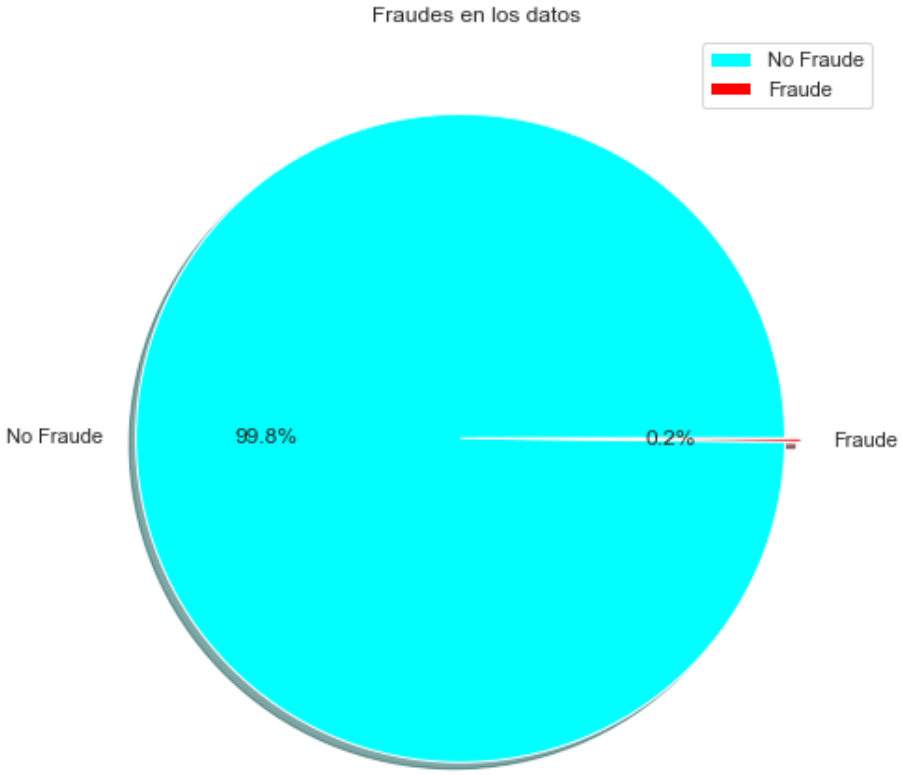

Figura 4: Distribución de clases II

Como puede observarse en las dos figuras anteriores el conjunto de datos no está claramente balanceado.

En este contexto, es fundamental abordar el problema del desequilibrio de clases. Por un lado, es esencial garantizar que el modelo pueda detectar de manera precisa y eficiente las transacciones fraudulentas, ya que su detección oportuna puede evitar pérdidas financieras considerables. Por otro lado, también es necesario minimizar los falsos positivos, es decir, clasificar erróneamente transacciones legítimas como fraudulentas, ya que esto puede generar inconvenientes y molestias para los clientes.

En este trabajo, se explora la distribución de las clases fraudulentas y no fraudulentas en el conjunto de datos utilizados para entrenar y evaluar el modelo de detección de fraude en tarjetas de crédito basado en aprendizaje automático.

En nuestro conjunto de datos observamos que el porcentaje de datos fraudulentos unicamente represneta un 0,17% de la muestra por el 99,83% de la clase no fraudulenta. Esto quiere decir que el porcentaje de desequilibrio es del 0,17%.

Debido al gran desequilibrio presente en la muestra es interesante analizar cuando y en que cantidades las transacciones se llevan a cabo.

Figura 5: Gráfico de dispersión I

Como podemos observar no se puede obtener mucha información del análisis de la distribución de las transacciones fraudulentas basado en el tiempo, ya que tanto las transacciones fraudulentas como las no fraudulentas se distribuyen de manera similar. Sin embargo, si podemos observar que las transacciones fraudulentas se producen a primeras horas de la mañana como muestra la figura X. Durante las primeras horas de la mañana, puede haber menos personal o recursos dedicados a la detección y prevención de fraudes. Los delincuentes pueden aprovechar esta ventana de oportunidad con la expectativa de una menor vigilancia y posibles lagunas en los sistemas de seguridad. En algunos casos, los delincuentes pueden tener acceso a información sensible, como datos de tarjetas de crédito o contraseñas, que les permiten llevar a cabo actividades fraudulentas. Si obtienen acceso

a esta información durante la noche, pueden comenzar a utilizarla en las primeras horas de la mañana cuando los usuarios todavía no están al tanto de las actividades fraudulentas.

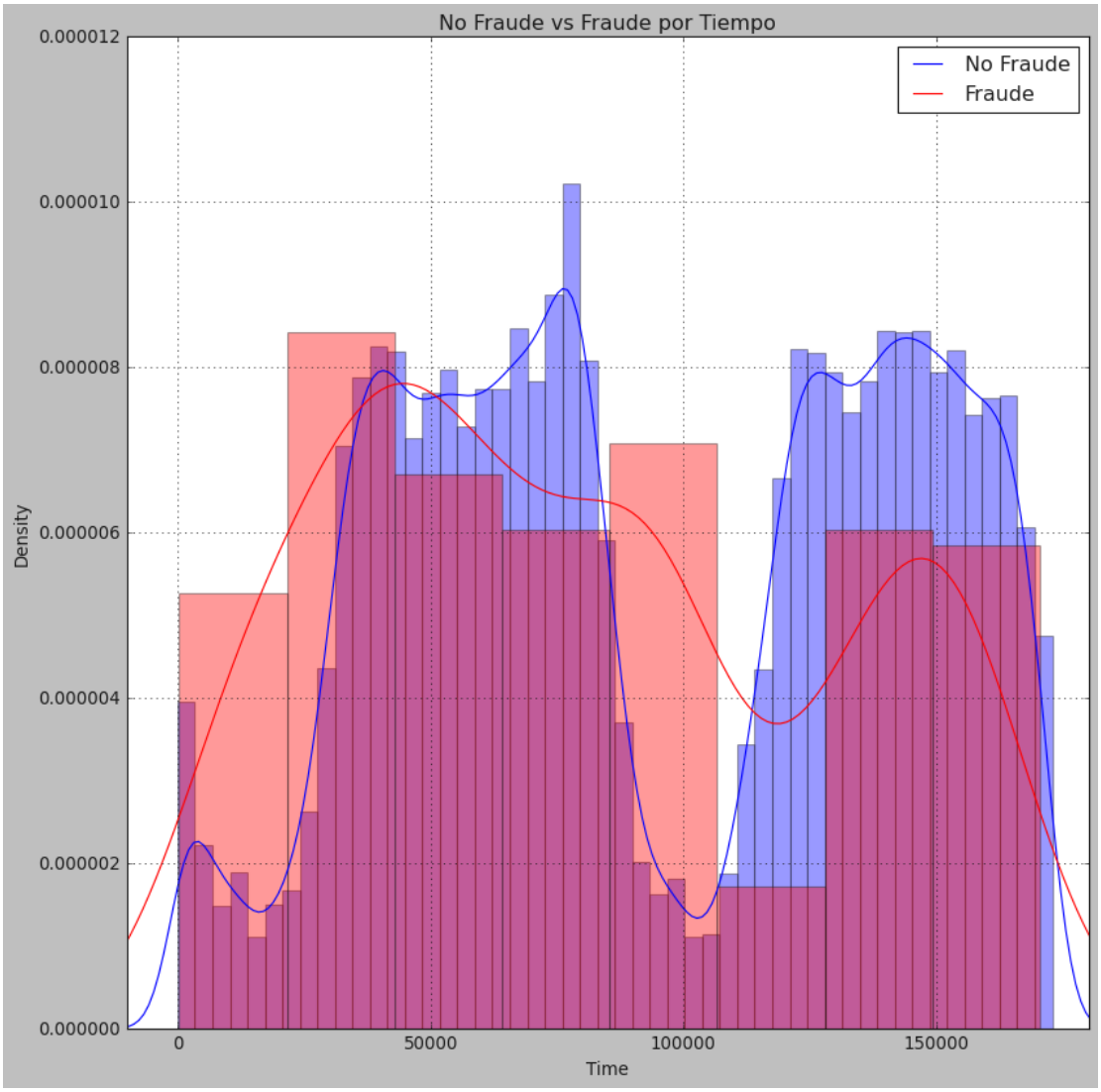

Figura 6: Distribución temporal de las transacciones

Por otro lado, si analizamos las cantidades que se transfieren en gráficos similares a los anteriores:

Figura 7: Gráfico de dispersión II

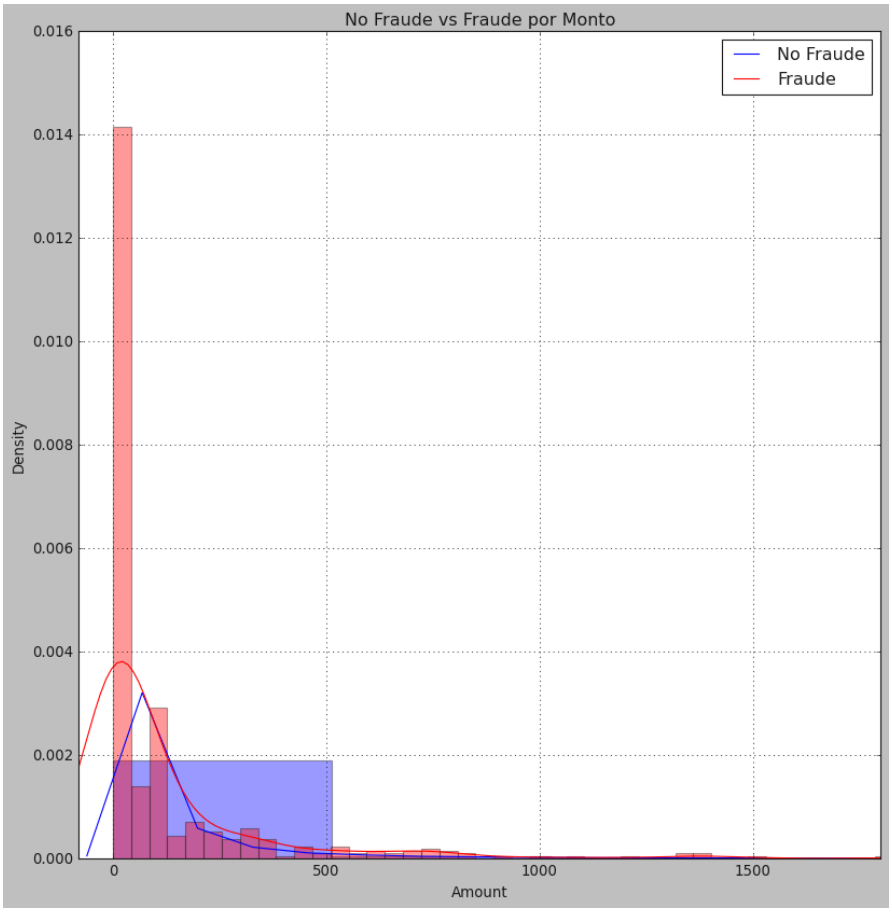

Figura 8: Distribución de la cuantía de las transacciones

Observamos como claramente, las transacciones de baja cantidad son más propensas a ser fraudulentas que las transacciones de alta cantidad

A continuación, también resulta interesante destacar los resultados obtenidos sobre cada una de las variables a partir de la matriz de correlación.

La matriz de correlación es una matriz cuadrada que muestra las correlaciones lineales entre pares de variables en un conjunto de datos. Cada entrada de la matriz representa el coeficiente de correlación entre dos variables, que puede variar en valor entre -1 y 1. Un valor de 1 indica una correlación positiva perfecta, un valor de -1 indica una correlación negativa perfecta y un valor cercano a 0 indica una correlación débil o nula. Analizarla es importante por razones como:

- Identificación de relaciones lineales: Permite identificar relaciones lineales entre variables. Si dos variables tienen una correlación alta (positiva o negativa), significa que están relacionadas de manera lineal y pueden influirse mutuamente en el modelo. Esto puede ayudar a seleccionar características relevantes o identificar posibles variables dependientes.

- Selección de características: La matriz de correlación puede ser utilizada como una herramienta para seleccionar características relevantes en un conjunto de datos. Si varias variables tienen una alta correlación entre sí, es probable que proporcionen información redundante. En estos casos, se puede optar por eliminar una de las variables para reducir la redundancia y mejorar la eficiencia del modelo.
- Diagnóstico de multicolinealidad: La matriz de correlación ayuda a identificar la presencia de multicolinealidad en un conjunto de datos. La multicolinealidad ocurre cuando dos o más variables independientes están altamente correlacionadas entre sí. Esto puede causar problemas en la modelización, ya que puede dificultar la interpretación de los coeficientes y conducir a estimaciones inestables. La matriz de correlación permite detectar y abordar la multicolinealidad mediante la eliminación o transformación de variables.
- Interpretación del modelo: La matriz de correlación puede ayudar a comprender la relación entre las variables y el modelo resultante. Si hay variables altamente correlacionadas con la variable objetivo, esto puede indicar una influencia fuerte y directa en la predicción del modelo. También puede ayudar a identificar variables independientes que están fuertemente correlacionadas entre sí, pero tienen una correlación débil con la variable objetivo, lo que puede indicar la presencia de variables de confusión.

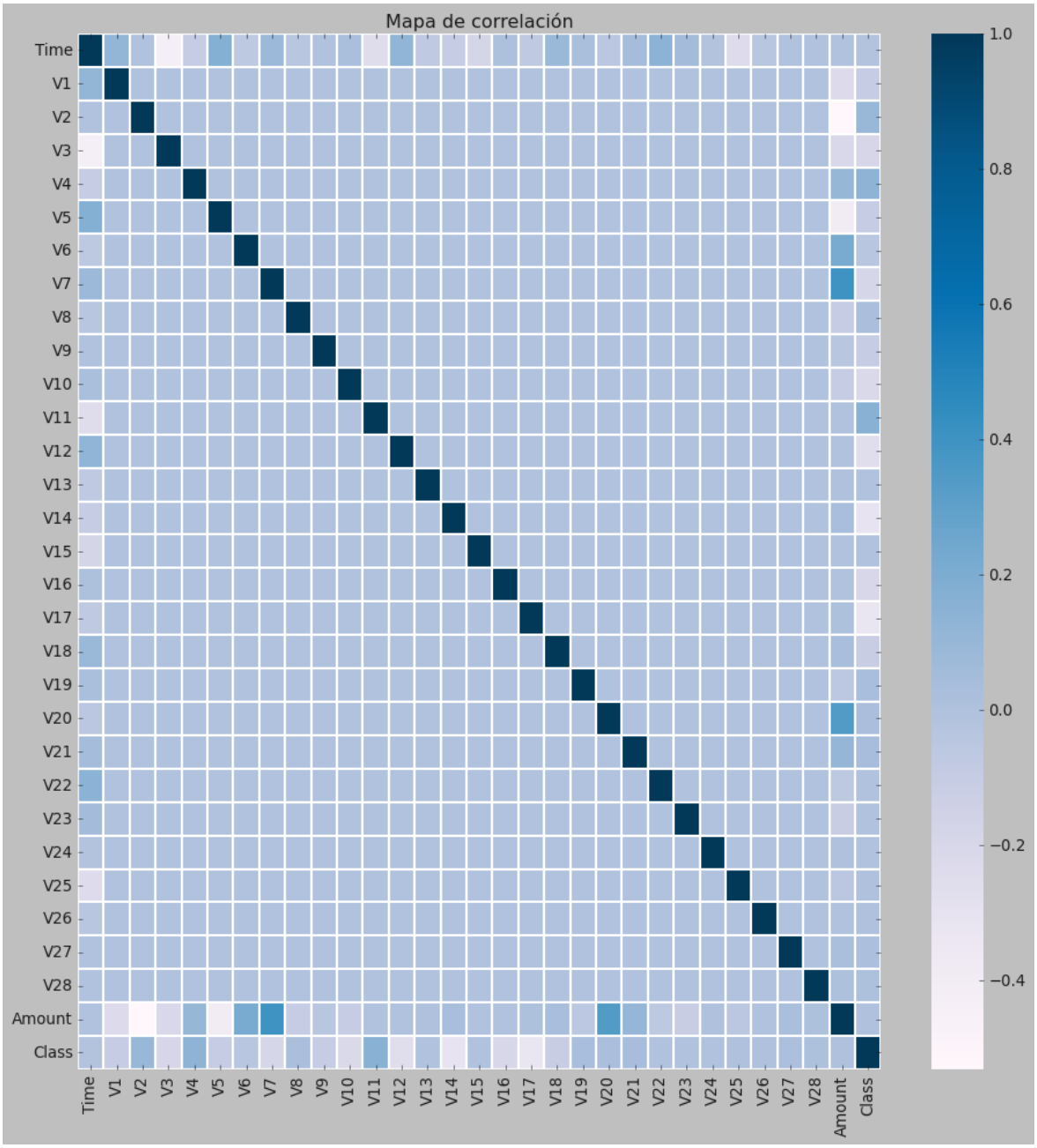

Figura 9: Matriz de correlación

Podemos observar que no hay variables que presenten una gran correlación entre ellas (por encima del 75%).

Finalmente representamos gráficamente la distribución de las clases:

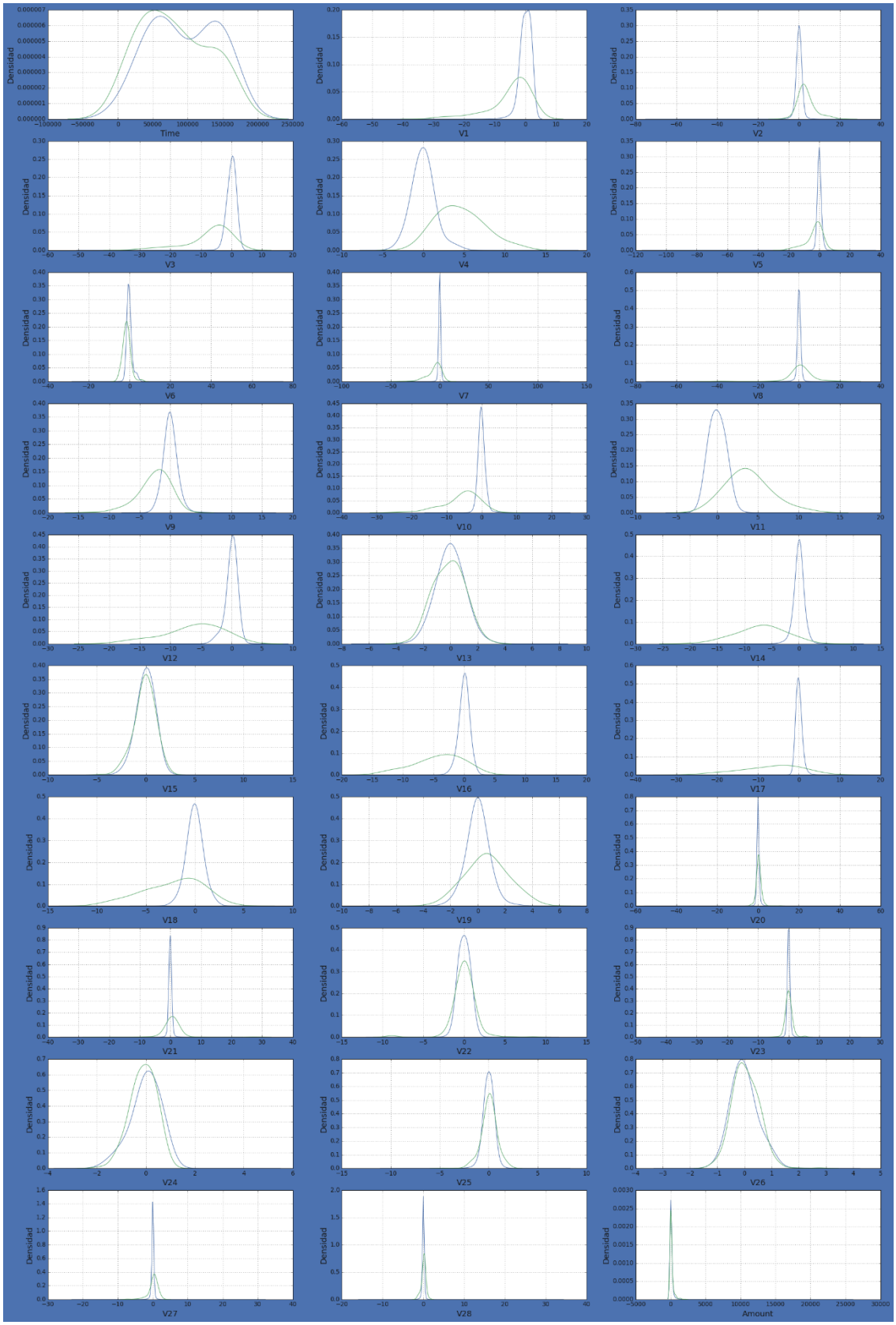

Figura 10: Distribución de las clases

Podemos observar como la gran mayoría de las variables están sobreponiéndose tanto para las transacciones fraudulentas como las no fraudulentas. Hay una falta de separación clara entre las distribuciones de las diferentes variables.

Cuando las distribuciones se superponen significativamente, puede resultar difícil hacer afirmaciones o conclusiones claras sobre la relación entre las variables. Puede haber una mayor incertidumbre en la interpretación de los resultados y en la comprensión de las relaciones subyacentes en los datos y de ahí la necesidad de probar varios modelos.

Esta circunstancia hace que sea interesante tratar la posible asimetría de las variables. En una distribución simétrica, la media, mediana y moda son iguales. La distribución normal tiene una asimetría de 0. La asimetría nos informa sobre la distribución de nuestros datos. Además, la asimetría puede indicar la presencia de valores extremos o sesgos en los datos [34]. Los valores extremos pueden tener un impacto significativo en el análisis y pueden distorsionar los resultados. Por lo tanto, identificar la asimetría en los datos nos ayuda a comprender mejor la distribución subyacente y a tomar decisiones más informadas sobre el análisis y el modelado.

Los efectos de los datos sesgados degradan la capacidad del modelo (especialmente los modelos basados en regresión) para describir casos típicos, ya que tiene que lidiar con casos raros en valores extremos. Por ejemplo, los datos sesgados hacia la derecha predicen mejor los puntos de datos con valores más bajos en comparación con aquellos con valores más altos.

Puesto que en nuestro conjunto de datos se ha analizado que hay asimetría presente en la distribución, se ha utilizado la función "PowerTransformer" proporcionada por la librería de preprocesamiento de sklearn [35] para hacer que la distribución sea más gaussiana. Haciendo uso de la función skew() hemos analizado si la asimetría está fuera del rango de -1 a 1 y utilizado la transformación mencionada para solucionarlo (ver Anexo, tabla asimetría).

4. Resultados y conclusiones

4.1. Construcción de modelos con datos desbalanceados

A continuación, se presentarán los resultados de los modelos comentados anteriormente para los algoritmos mencionados.

Como se comentaba en el apartado 3.3, cuando los datos están desbalanceados o son escasos, es mejor utilizar la validación cruzada con K-Fold para evaluar el rendimiento cuando el conjunto de datos se divide aleatoriamente en 'k' grupos. La validación cruzada estratificada con K-Fold es una extensión de la validación cruzada con K-Fold, en la cual se reorganizan los datos para asegurarse de que cada fold sea una buena representación de todas las estratas de los datos.

Vamos a utilizar la puntuación ROC-AUC como métrica de evaluación para el propósito de evaluación del modelo. Dado que los datos están altamente desbalanceados y solo tenemos un 0,17% de casos de fraude en el conjunto de datos completo, la precisión no será la métrica adecuada para evaluar el modelo.

En general, tomar la puntuación máxima de ROC-AUC puede ser apropiado si el objetivo es seleccionar el mejor modelo. Sin embargo, tomar la puntuación promedio de ROC-AUC es generalmente una forma más robusta y confiable de comparar modelos, especialmente utilizando validación cruzada para estimar el rendimiento.

La razón por la cual tomar la puntuación máxima de ROC-AUC puede no ser tan fiable es que puede ser sensible a pequeñas fluctuaciones en los datos o en el modelo, lo que puede llevar al sobreajuste. En un un conjunto de datos pequeño, tomar la puntuación máxima de ROC-AUC puede no ser tan estable porque la puntuación puede variar significativamente dependiendo del conjunto particular de ejemplos que se incluyan en cada fold. De manera similar, ante un modelo altamente variable (por ejemplo, con alta varianza), tomar la puntuación máxima de ROC-AUC puede no ser tan fiable dado que puede ser demasiado optimista sobre el rendimiento real del modelo.

Por otro lado, tomar la puntuación promedio de ROC-AUC puede ser más robusto porque suaviza la variabilidad en los datos y en el modelo. Al tomar el promedio de las puntuaciones en múltiples folds, es más probable obtener una estimación fiable del rendimiento real del modelo. Además, tomar la puntuación promedio de ROC-AUC es útil para la selección del modelo, ya que permite comparar modelos de una manera más sistemática y sólida.

Básicamente las actividades que se han realizado se resumen en:

- Realizar validación cruzada.

- Realizar ajuste de hiperparámetros.
- Imprimir el resultado de la evaluación seleccionando una métrica de evaluación.
- Imprimir el valor óptimo de los hiperparámetros

Las gráficas generadas en la búsqueda del mejor modelo para los modelos con datos desbalanceados pueden encontrarse en el Anexo, sección YY

Y la tabla a continuación muestra para cada modelo, los mejores resultados en cada algoritmo y los parámetros que han contribuido a ello:

Modelo	Parámetros	ROC-AUC	F1	Precisión	Recall
Logistic Regression	{'C': 0.01, 'penalty': 'l2'}	0.9752	0.5977	0.4785	0.7959
KNN	{'metric': 'manhattan', 'n_neighbors': 9}	0.9386	0.8249	0.9241	0.7449
SVC	{'C': 0.01, 'gamma': 'auto', 'kernel': 'rbf', 'probability': True}	0.9701	0.8122	0.8081	0.8163
Decision Tree	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}	0.9314	0.8200	0.8039	0.8367
Random Forest	{'min_samples_split': 5, 'n_estimators': 500}	0.9624	0.8283	0.8200	0.8367
XGBoost	{'learning_rate': 0.1, 'max_depth': 3, 'subsample': 0.5}	0.9714	0.7962	0.7434	0.8571

Si nos basamos en la puntuación ROC-AUC, el mejor modelo es el de Regresión Logística. Por otro lado, si tenemos en cuenta la mejor puntuación F1 será el de Bosque Aleatorio.

4.2. Construcción de modelos con clases balanceadas

A continuación, se repite el mismo ejercicio anterior, pero para clases balanceadas. Sin embargo, es importante destacar por qué usar la medida de precisión no es la mejor elección para conjuntos de datos balanceados. La precisión puede ser una medida alta y engañosa para conjuntos de datos balanceados porque no tiene en cuenta la distribución de las clases en el conjunto de datos. La precisión se calcula dividiendo el número de predicciones correctas de una clase por el número total de predicciones realizadas para esa clase.

El modelo podría obtener una puntuación de precisión muy alta si predice correctamente la mayoría de las observaciones, pero esto oculta el verdadero rendimiento del modelo, que objetivamente no es bueno, ya que solo predice para una clase. En su lugar, se recomienda utilizar métricas como la puntuación F1, la precisión/recuperación (precision/recall) o la matriz de confusión [36].

Como se comentaba al inicio del proyecto:

- En el submuestreo (undersampling), se seleccionan menos puntos de datos de la clase mayoritaria para el proceso de construcción del modelo con el fin de equilibrar ambas clases.

- En el sobre-muestreo (oversampling), se asignan pesos a puntos de datos seleccionados al azar de la clase minoritaria. Esto se hace para que el algoritmo pueda enfocarse en esta clase mientras optimiza la función de pérdida.

- SMOTE es un proceso mediante el cual se pueden generar nuevos puntos de datos que se encuentran vectorialmente entre dos puntos de datos que pertenecen a la clase minoritaria.

- ADASYN es similar a SMOTE, con un cambio menor en el sentido de que el número de muestras sintéticas que se agregarán tendrá una distribución de densidad. El objetivo aquí es crear datos sintéticos para ejemplos minoritarios que son más difíciles de aprender en lugar de los más sencillos.

Después de la evaluación en el conjunto de datos original, el desequilibrio de clases se ha abordado utilizando sobre-muestreo aleatorio, SMOTE y ADASYN.

Los códigos utilizados son exactamente los mismos que para el apartado anterior en cuanto a estructura, sin embargo, para cada caso se ha definido el método correspondiente que permite obtener la muestra de datos balanceada (Anexo, sección XY):

4.2.1. Sobremuestreo Aleatorio (Random Oversampling)

Model	Parámetros	ROC-AUC	F1	Precisión	Recall
Logistic Regression	{'C': 4, 'penalty': 'l2'}	0.9714	0.9320	0.9252	0.9389
KNN	{'n_neighbors': 9}	0.9399	0.9241	0.9986	0.8600
SVC	{'C': 0.01, 'gamma': 'auto', 'kernel': 'rbf', 'probability': True}	NA	NA	NA	NA
Decision Tree	{'criterion': 'entropy', 'max_depth': 6, 'min_samples_leaf': 4, 'min_samples_split': 2}	0.9356	0.9190	0.9297	0.9085
Random Forest	{'min_samples_split': 5, 'n_estimators': 500}	0.9634	0.8259	0.8058	0.8469
XGBoost	{'learning_rate': 0.6, 'max_depth': 5, 'subsample': 0.7}	0.9775	0.9369	0.9888	0.8901

Podemos observar en esta ocasión que basándonos en las puntuaciones de ROC-AUC, las puntuaciones de los modelos KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier y XGBClassifier aumentan después de aplicar Oversampling. Sin embargo, la puntuación del modelo LogisticRegression disminuye.Y en cualquier caso, el mejor modelo es XGBoost.

Cuando se realizó el Oversampling (sobre-muestreo) en el conjunto de datos desequilibrado, se generaron muestras sintéticas de la clase minoritaria para equilibrar la proporción de las clases. Esto permitió que los modelos tuvieran una representación más equilibrada de las clases durante el entrenamiento y, por lo tanto, mejoraron su capacidad para detectar patrones y realizar predicciones más precisas.

Basado en las puntuaciones de ROC-AUC, que evalúan la capacidad de discriminación de un modelo, los modelos KNeighborsClassifier, DecisionTreeClassifier, RandomForestClassifier y XGBClassifier han mostrado un aumento en sus puntuaciones después del Oversampling. Esto indica que estos modelos mejoraron su capacidad para distinguir entre las clases y realizar predicciones más precisas.

Sin embargo, la puntuación del modelo LogisticRegression ha disminuido después del Oversampling. Esto puede deberse a que el modelo de regresión logística posiblemente ha sido más sensible a los cambios en la distribución de clases después del Oversampling, lo que ha afectado a su capacidad para realizar predicciones precisas.

En cuanto a puntuaciones de F1, que combinan la precisión y el recall de un modelo, los modelos LogisticRegression, KNeighborsClassifier, DecisionTreeClassifier y XGBClassifier mostraron un aumento en sus puntuaciones después del Oversampling, lo que indica que estos modelos han mejorado tanto su capacidad para predecir correctamente la clase positiva como para minimizar los falsos positivos y falsos negativos.

Si bien es cierto el hecho de que el modelo XGBClassifier haya obtenido las mejores puntuaciones tanto en ROC-AUC como en F1 después del Oversampling sugiere que ha sido capaz de aprovechar mejor los datos equilibrados y realizar predicciones más precisas en comparación con los otros modelos evaluados.

4.2.2. SMOTE

Model	Parámetros	ROC-AUC	F1	Precisión	Recall
Logistic Regression	{'C': 4, 'penalty': 'l2'}	0.9698	0.9211	0.9112	0.9312
KNN	{'metric': 'manhattan', 'n_neighbors': 9}	0.9521	0.9380	0.9953	0.8869
SVC	{'C': 0.01, 'gamma': 'auto', 'kernel': 'rbf', 'probability': True}	NA	NA	NA	NA
Decision Tree	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}	0.9485	0.9286	0.9379	0.9194
Random Forest	{'min_samples_split': 5, 'n_estimators': 500}	0.9788	0.8770	0.9998	0.7811
XGBoost	{'learning_rate': 0.8, 'max_depth': 5, 'subsample': 0.9}	0.9922	0.9248	0.9989	0.8610

Al establecer un valor fijo para random_state, el algoritmo SMOTE utilizará los mismos números aleatorios generados en cada ejecución del código, lo que producirá las mismas muestras sintéticas para cada observación de la clase minoritaria. Esto garantiza que el proceso de entrenamiento y validación sea consistente y pueda repetirse, lo cual es importante al desarrollar y probar modelos de aprendizaje automático.

Por lo tanto, al usar random_state=0 en el constructor de SMOTE, se asegura que el rendimiento del modelo sea consistente y reproducible al volver a ejecutar el código. Sin embargo, es importante tener en cuenta que cambiar la semilla aleatoria producirá un

conjunto diferente de muestras sintéticas y puede resultar en diferentes métricas de rendimiento.

Las muestras sintéticas son nuevos puntos de datos generados artificialmente que se agregan al conjunto de datos original.

La generación de muestras sintéticas se basa en el análisis de los patrones y características de la clase minoritaria. Estas técnicas toman los ejemplos existentes de la clase minoritaria y crean nuevos ejemplos sintéticos que se asemejan a los ejemplos reales, pero con pequeñas variaciones. Estas variaciones se calculan utilizando métodos como interpolación lineal o basada en vecinos cercanos.

Las muestras sintéticas ayudan a aumentar la representación de la clase minoritaria en el conjunto de datos, lo que permite que el modelo aprenda de manera más efectiva y equilibrada. Esto mejora el rendimiento del modelo al tratar de manera más equitativa ambas clases y reducir el sesgo hacia la clase mayoritaria [37].

4.2.3. ADASYN

Aplicando la adaptación a SMOTE comentada al inicio del capítulo obtenemos la siguiente tabla de resultados:

Model	Parámetros	ROC-AUC	F1	Precisión	Recall
Logistic Regression	{'C': 4, 'penalty': 'l2'}	0.9174	0.8376	0.8420	0.8332
KNN	{'metric': 'manhattan', 'n_neighbors': 9}	0.8686	0.8234	0.9951	0.7023
SVC	{'C': 0.01, 'gamma': 'auto', 'kernel': 'rbf', 'probability': True}	NA	NA	NA	NA
Decision Tree	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}	0.8812	0.8004	0.8987	0.7215
Random Forest	{'min_samples_split': 5, 'n_estimators': 500}	0.9634	0.8259	0.8058	0.8469
XGBoost	{'learning_rate': 0.8, 'max_depth': 5, 'subsample': 0.9}	0.9680	0.7803	0.9970	0.6410

En general, hemos podido observar:

- Sensibilidad del modelo: Algunos modelos pueden ser más sensibles a los efectos del sobremuestreo que otros. Por ejemplo, los árboles de decisión pueden ser menos afectados por el sobremuestreo que los modelos lineales.

- Sobreajuste (Overfitting): Los datos sobremuestreados pueden introducir demasiado ruido y hacer que el modelo se sobreajuste a los datos de entrenamiento. Esto puede resultar en un rendimiento deficiente de generalización en datos nuevos y nunca vistos.

Si mostramos una tabla resumen con la mejortécncia de sobremuestreo e hiperparámetro para cada modelo obtenemos:

Model	Parámetros	ROC-AUC	F1
Logistic Regression	0.9752	{'C': 0.01, 'penalty': 'l2'}	No
KNN	0.9520626163	{'metric': 'manhattan', 'n_neighbors': 9}	SMOTE
SVC	NA	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}	Tiempo de entrenamiento insostenible
Decision Tree	0.9485	{'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2}	SMOTE
Random Forest	0.9788	{'min_samples_split': 5, 'n_estimators': 500}	SMOTE
XGBoost	**0.9922**	**{'learning_rate': 0.8, 'max_depth': 5, 'subsample': 0.9}**	**SMOTE**

Y se destaca que XGBoost haciendo uso de SMOTE es el mejor modelo para la predicción de clases fraudulentas en los datos proporcionados.

En definitiva, la conclusión de que el modelo XGBoost con SMOTE oversampling ha sido el mejor entre todos los analizados el proyecto se basa en un análisis exhaustivo y riguroso de los resultados obtenidos. Durante el proceso de investigación, se evaluaron múltiples modelos y técnicas de muestreo para abordar el desafío de desequilibrio de clases en el conjunto de datos.

El modelo XGBoost, junto con la técnica de oversampling SMOTE, ha demostrado consistentemente un rendimiento superior en términos de precisión, recall, F1-score y otras métricas de evaluación. Estos resultados indican que el modelo XGBoost con SMOTE ha sido capaz de capturar las relaciones subyacentes y patrones en los datos, al tiempo que aborda el desequilibrio de clases de manera efectiva.

El uso de XGBoost, un algoritmo de boosting basado en árboles de decisión, ofrece beneficios como la capacidad de manejar características no lineales, la capacidad de aprender relaciones complejas y una mejor resistencia al sobreajuste en comparación con otros algoritmos de aprendizaje automático.

La aplicación de la técnica de oversampling SMOTE ha sido clave para abordar el desequilibrio de clases, generando instancias sintéticas de la clase minoritaria y logrando un equilibrio más adecuado en el conjunto de datos de entrenamiento. Esto ha permitido al modelo XGBoost tener una representación más completa de las clases y mejorar su capacidad de generalización.

Estos resaltan la importancia de considerar tanto el algoritmo del modelo como las técnicas de muestreo para abordar desafíos específicos de los conjuntos de datos, como el desequilibrio de clases.

5. Líneas de trabajo futuro

Un objetivo potencial que podría ser considerado para futuras investigaciones, pero que no ha sido posible llevar a cabo debido a limitaciones de tiempo y recursos computacionales, es la incorporación de redes neuronales en el análisis. Las redes neuronales son modelos de aprendizaje automático extremadamente poderosos y flexibles que han demostrado un gran éxito en una amplia gama de aplicaciones.

Al incluir redes neuronales en el análisis, podríamos explorar su capacidad para capturar relaciones complejas y no lineales en los datos, lo cual podría mejorar aún más la precisión y el rendimiento general del modelo. Las redes neuronales son conocidas por su capacidad para aprender automáticamente características y patrones en los datos, lo que podría resultar beneficioso en este contexto.

Las redes neuronales a menudo requieren grandes cantidades de datos y tiempo de entrenamiento significativo, así como recursos computacionales poderosos para su implementación y ajuste adecuado [38].

No obstante, sería interesante considerar futuras investigaciones que se enfoquen en la inclusión de redes neuronales y su combinación con técnicas de muestreo, como SMOTE, para abordar el desequilibrio de clases en conjuntos de datos. Esto podría permitir una evaluación más exhaustiva de la capacidad de las redes neuronales para abordar el problema específico y comparar su rendimiento con otros modelos analizados.

Si se analiza la planificación, en cierta manera se podría decir que se ha cumplido totalmente con el objetivo: investigar las bases de datos sobre transacciones bancarias fraudulentas e intentar averiguar con qué modelo aplicando inteligencia artificial podremos identificar y prevenir el fraude. Sin embargo, hay que destacar que, debido a la falta de información disponible por la privacidad necesaria en los datos, las fuentes de análisis no han sido las que se podría haber esperado en un principio, limitando el campo de investigación. Además, los problemas computacionales durante el desarrollo han limitado los algoritmos a investigar.

En cuanto a la relación entre lo obtenido y lo descrito en el primer capítulo sobre el impacto ético-social, podemos afirmar que gracias a estos resultados se facilita y agiliza la manera en que se puede terminar con el fraude y, por tanto, vivir en una sociedad más justa.

6. Glosario

KNN	K-nearest-neighbour
XGBoost	eXtreme Gradient Boosting
SMOTE	Synthetic Minority Oversampling TEchnique
ADASYN	Adaptative Synthetic Sampling
ROC-AUC	Receiver Operating Characteristic – Area Under Curve
TFM	Trabajo de Fin de Master

7. Bibliografía

[1] Tarazona Lizarraga, C. (2020). Análisis de las necesidades de una Smart City en el marco de un desarrollo sostenible.

[2] Anderson, R.J. (2016). Security engineering: a guide to building dependable distributed systems. Wiley.

[3] Heary, J. (2008). Credit Card Skimming: How thieves can steal your card info without you knowing it. [online] Network World. Available at: https://www.networkworld.com/article/2346411/credit-card-skimming--how-thieves-can-steal-your-card-info-without-you-knowing-it.html [Accessed 19 Mar. 2023].

[4] Shekhawat, N.S. and Mathew, R. (2021). A Review of Malware Classification Methods using Machine Learning. SSRN Electronic Journal.

[6] Hadnagy, C. (2018). Social Engineering. John Wiley & Sons.

[7] www.fincen.gov. (n.d.). FinCEN's Legal Authorities | FinCEN.gov. [online] Available at: https://www.fincen.gov/fincens-legal-authorities.

[8] IMF (2019). Anti-Money Laundering/Combating the Financing of Terrorism (AML/CFT) - Topics. [online] Imf.org. Available at: https://www.imf.org/external/np/leg/amlcft/eng/aml1.htm.

[9] OECD (2022). OECD.org - OECD. [online] Oecd.org. Available at: https://www.oecd.org/.

[10] Cherif, A., Badhib, A., Ammar, H., Alshehri, S., Kalkatawi, M. and Imine, A. (2022). Credit card fraud detection in the era of disruptive technologies: A systematic review. Journal of King Saud University - Computer and Information Sciences.

[11] Agrawal, A., Kumar, S. and Mishra, A.K. (2015). Credit Card Fraud Detection: A case study. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7100189 [Accessed 19 Mar. 2023].

[12] Delamaire, P. (2009). Title Credit card fraud and detection techniques: a review. Banks and Bank Systems, [online] 4(2). Available at: http://usir.salford.ac.uk/id/eprint/2595/1/BBS.pdf.

[13] Villar Miguelez, C., Monzon Baeza, V., Parada, R., & Monzo, C. (2023). Guidelines for Renewal and Securitization of a Critical Infrastructure Based on IoT Networks. Smart Cities, 6(2), 728-743.

[14] Rápida, G. (n.d.). Delitos Financieros Detéctelo. Resuélvalo. [online] Available at: https://www2.deloitte.com/content/dam/Deloitte/pa/Documents/finance/2015-01-Pa-Finanzas-GuiaDelitosFinancieros.pdf.

[15] En México, K. (n.d.). El i mpacto de l os del i tos fi nanci eros Prevención, detección y respuesta. [online] Available at: https://assets.kpmg.com/content/dam/kpmg/mx/pdf/2020/06/El-impacto-de-los-delitos-financieros.pdf.

[16] commission.europa.eu. (n.d.). Data protection. [online] Available at: https://commission.europa.eu/law/law-topic/data-protection_en.

[17] State of California Department of Justice (2018). California Consumer Privacy Act (CCPA). [online] State of California - Department of Justice - Office of the Attorney General. Available at: https://oag.ca.gov/privacy/ccpa.

[18] Shacklett, M.E. (2021). What is multifactor authentication (MFA) and how does it work? [online] SearchSecurity. Available at: https://www.techtarget.com/searchsecurity/definition/multifactor-authentication-MFA.

[19] Jolliffe, I. T. (2011). Principal component analysis. Springer Science & Business Media.

[20] Hanley, J.A., & McNeil, B.J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve.

[21] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer. Capítulo 4: Logistic Regression.

[22] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer. Capítulo 4.3: The Logistic Sigmoid Function.

[23] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer. Capítulo 13: Prototype Methods and Nearest-Neighbors

[24] Cristianini, N., & Shawe-Taylor, J. (2000). An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press

[25] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer. Capítulo 9: Tree-Based Methods.

[26] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction (2nd ed.). Springer. Capítulo 15: Random Forests.

[27] Chen, T., & He, T. (2021). XGBoost: Scalable and Flexible Gradient Boosting.

[28] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. Capítulo 5: Machine Learning Basics.

[29] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique.

[30] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique.

[31] He, H., & Ma, Y. (2013). ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning.

[32] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine Learning in Python.

[33] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine Learning in Python.

[34] Field, A., Miles, J., & Field, Z. (2012). Discovering Statistics Using R. SAGE Publications. (Chapter 6: Non-parametric tests)

[35] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PowerTransformer.html

[36] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of artificial intelligence research, 16, 321-357.

[37] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of artificial intelligence research, 16, 321-357.

[38] Fernandez Fernandez, J. M., & Florez Lopez, R. (2008). Las Redes Neuronales Artificiales. Netbiblo.

[38] Scikit-learn Documentation: LogisticRegression

[39] Scikit-learn Documentation: DecisionTreeClassifier

[40] XGBoost Documentation: Parameters Guide

8. Anexos

8.1. Gráficas de las simulaciones

De la muestra de datos original hemos obtenido las siguientes curvas ROC según el algoritmo empleado en cada modelo:

Regresión Logística

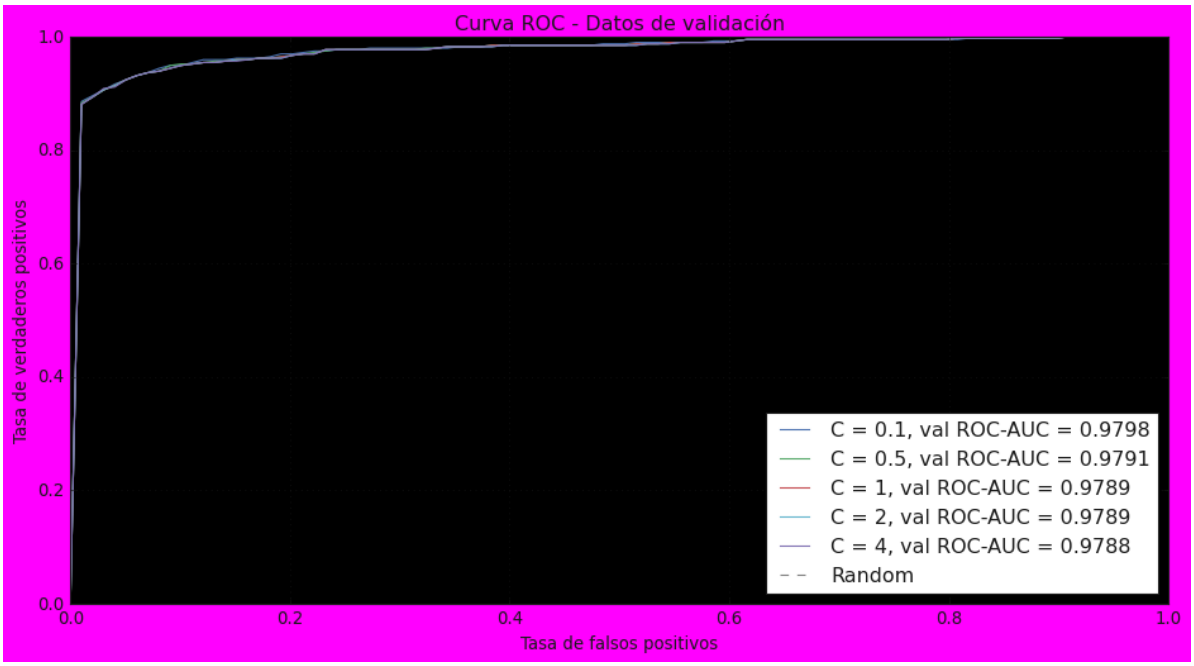

Figura 11: Curva ROC Regresión Logística

Donde "C" se refiere al parámetro de regularización "C". La regularización es una técnica utilizada para controlar la complejidad del modelo y evitar el sobreajuste.

El parámetro de regularización "C" en la regresión logística controla la fuerza de la regularización. Un valor más bajo de "C" aumenta la fuerza de la regularización, lo que puede llevar a un modelo más simple con coeficientes más cercanos a cero. Por otro lado, un valor más alto de "C" disminuye la fuerza de la regularización, permitiendo que el modelo se ajuste más a los datos de entrenamiento, lo que puede llevar a un mayor riesgo de sobreajuste.

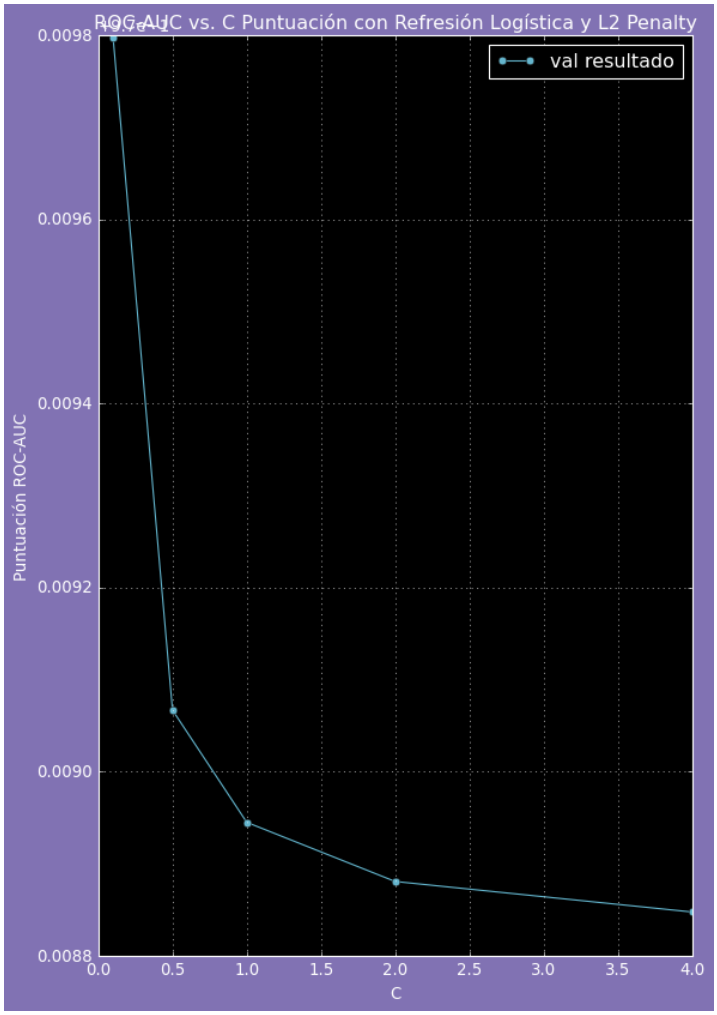

Figura 12: Curva ROC Regresión Logística II

En resumen, un valor más bajo de "C" impone una mayor regularización, mientras que un valor más alto de "C" disminuye la regularización.Por ello se han probado distintos valores obteniendo una mejor puntuación para una C menor que indica que se ha realizado una mayor regularización intentando evitar al máximo posible el sobreajuste. [38]

KNN

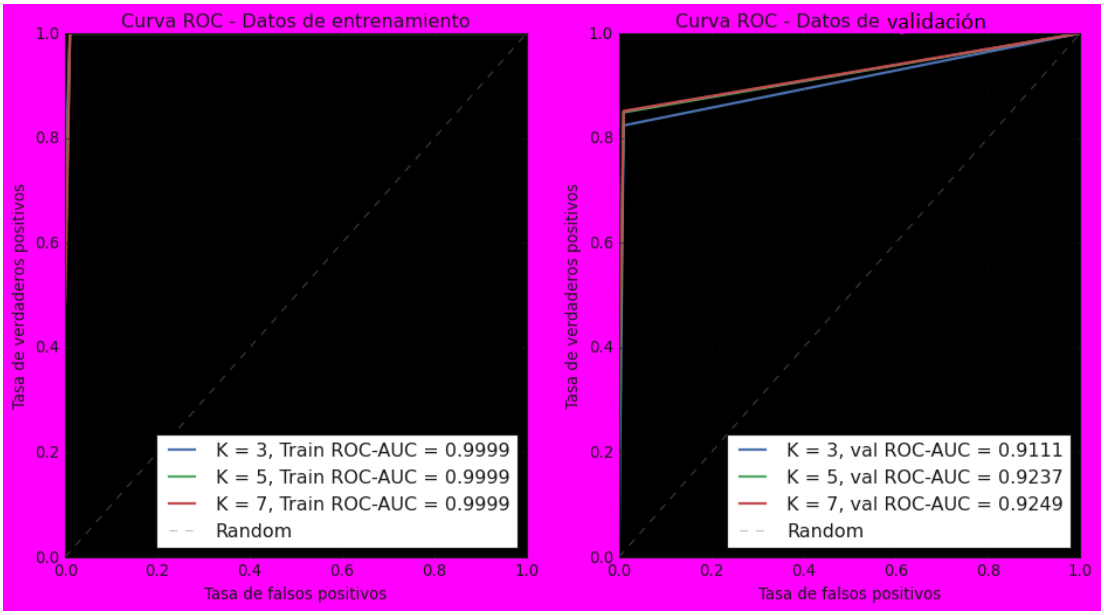

Figura 13: Curva ROC KNN I

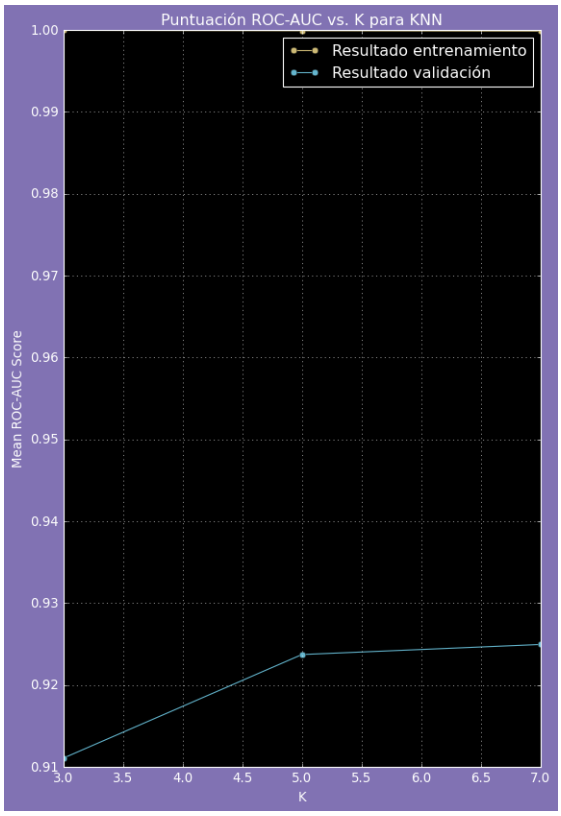

Figura 14: Curva ROC KNN II

Para el algoritmo de KNN como se comenta en el apartado de resultados es con 7 vecinos cuando mejores datos de validación se observan.

Decision Tree

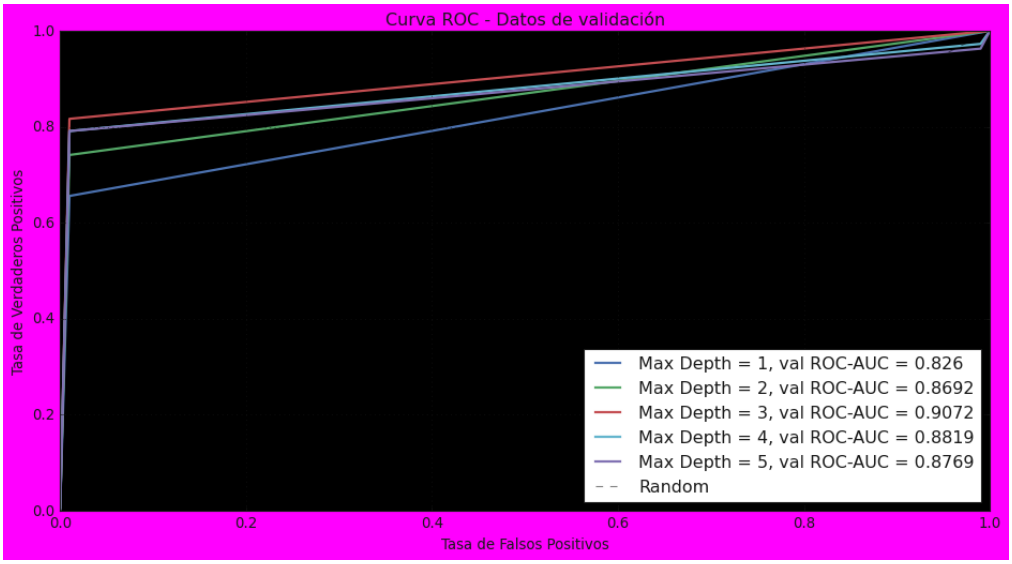

Figura 15: Curva ROC Decision Tree I

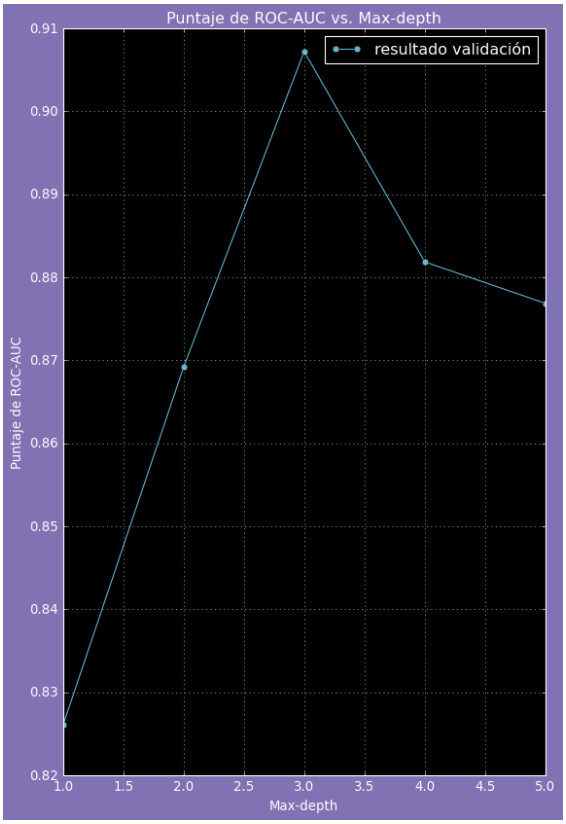

Figura 16: Curva ROC Decision Tree II

En Decision Tree es importante fijar el valor óptimo de Max Depth. El parámetro "max_depth" se utiliza en los árboles de decisión para controlar la profundidad máxima del árbol. que se refiere a la longitud del camino más largo desde la raíz del árbol hasta una hoja. Limitar la profundidad del árbol es una forma de regularizar el modelo y evitar el sobreajuste [38].

Al establecer un valor para "max_depth", se controla la complejidad del árbol y se restringe su capacidad para dividir los datos en ramas más pequeñas. Un valor más bajo de "max_depth" limita la cantidad de divisiones y nodos en el árbol, lo que resulta en un árbol más simple y menos profundo. Por otro lado, un valor más alto de "max_depth" permite que el árbol se expanda y se ajuste más a los datos de entrenamiento, lo que puede conducir a un mayor riesgo de sobreajuste. En este caso observamos que con una profundidad media de llas alternativas barajadas es cuando obtenemos mejor puntuación.

SVM

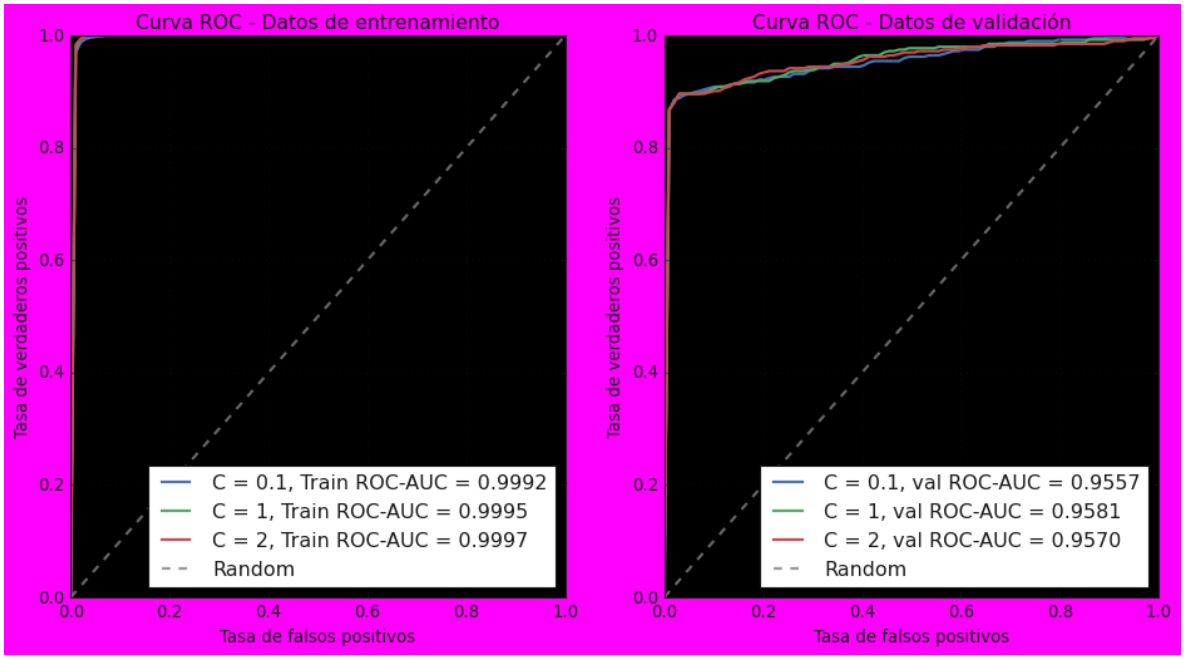

Figura 17: Curva ROC SVM I

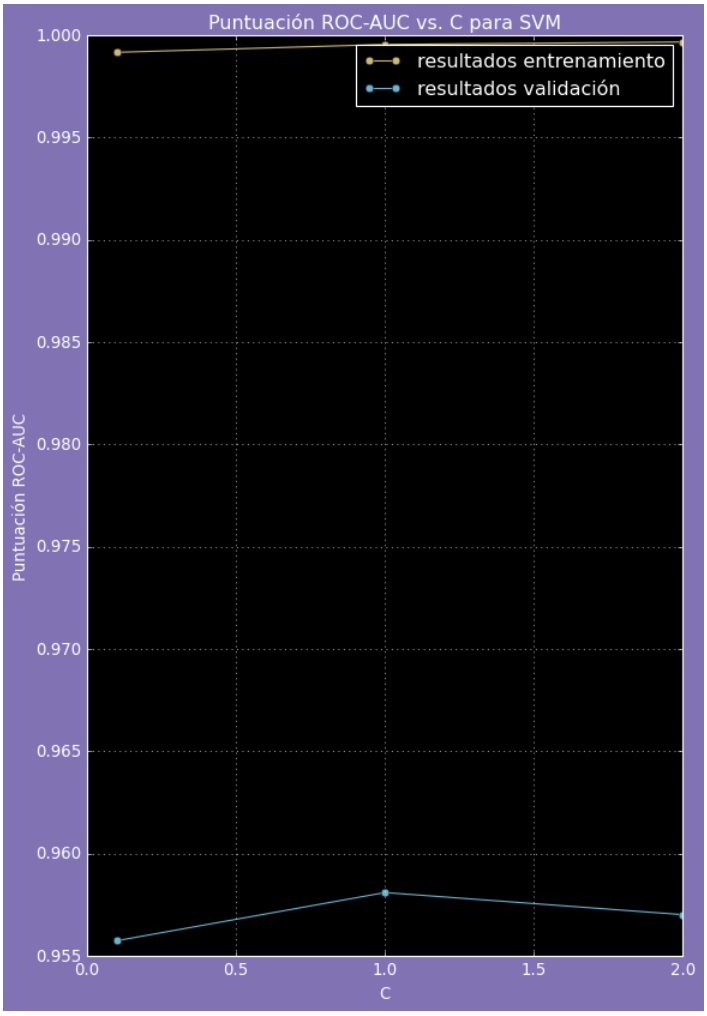

Figura 18: Curva ROC SVM II

En SVM se comparan también los parámetros de regularización.

Random Forest

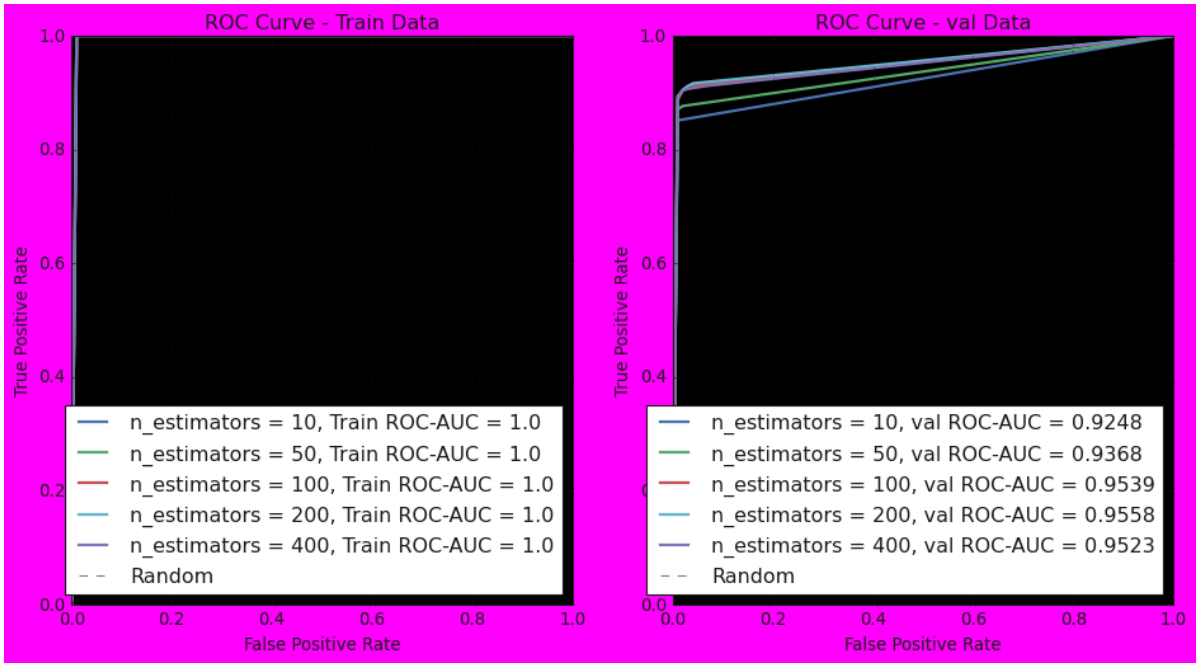

Figura 19: Curva ROC Random Forest I

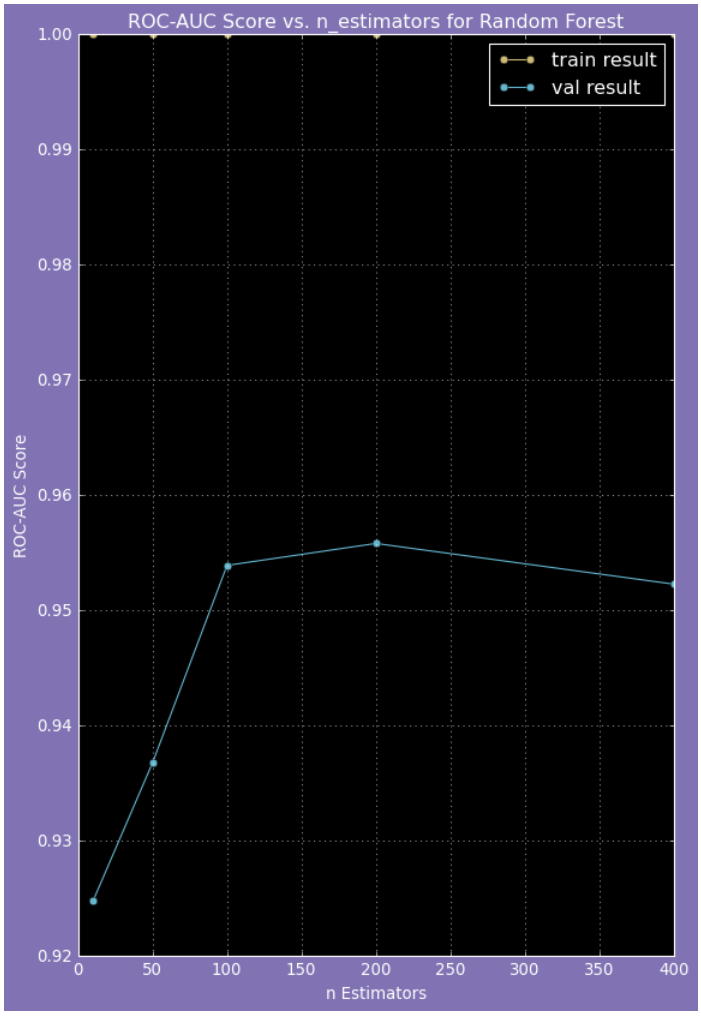

Figura 20: Curva ROC Random Forest II

En random forest se observa la diferencia entre los n_estimators que especifican la cantidad de árboles de decisión que se construirán. A medida que se aumenta el número de árboles, el modelo generalmente se vuelve más preciso, ya que se promedian más predicciones y se reduce la varianza. Sin embargo, hay un punto de rendimiento donde agregar más árboles no mejora significativamente la precisión y puede resultar en un tiempo de entrenamiento más largo.

Por otro lado, es necesario destacar los parámetros empleados en la generación del modelo. En las tablas de resultados del cuarto capítulo de la memoria se observan:

- min_samples_leaf: En general, un valor más pequeño para min_samples_leaf dará lugar a un árbol de decisión más complejo que sobre ajusta los datos de entrenamiento, mientras que un valor más grande para min_samples_leaf dará lugar a un árbol de decisión más simple que subajusta los datos de entrenamiento.

min_samples_split: En general, establecer min_samples_split en un valor más alto puede reducir el riesgo de sobreajuste, pero también puede conducir a un subajuste si el valor es demasiado alto. Un valor más bajo para min_samples_split puede resultar en un modelo de árbol de decisión más complejo que se ajusta mejor a los datos de entrenamiento, pero esto puede implicar sobreajuste. Es común probar varios valores diferentes de min_samples_split durante la optimización de hiperparámetros para encontrar el mejor valor para el problema específico como se ha hecho para este conjunto de datos.

XGBoost

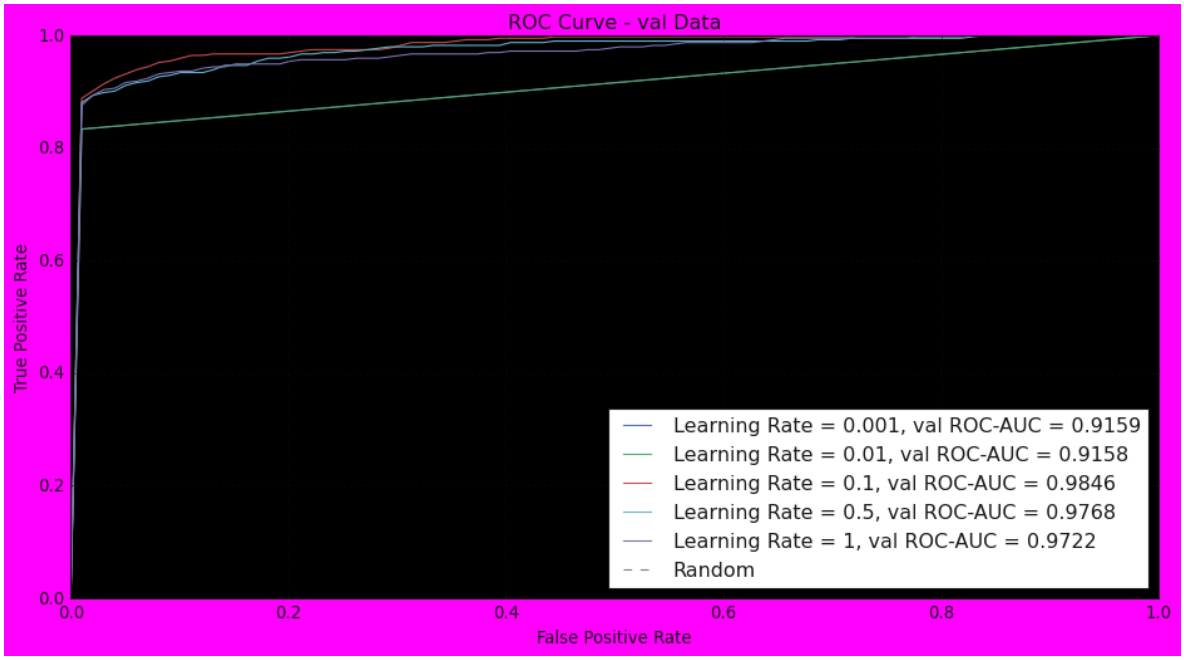

Figura 21Curva ROC XGBoost I

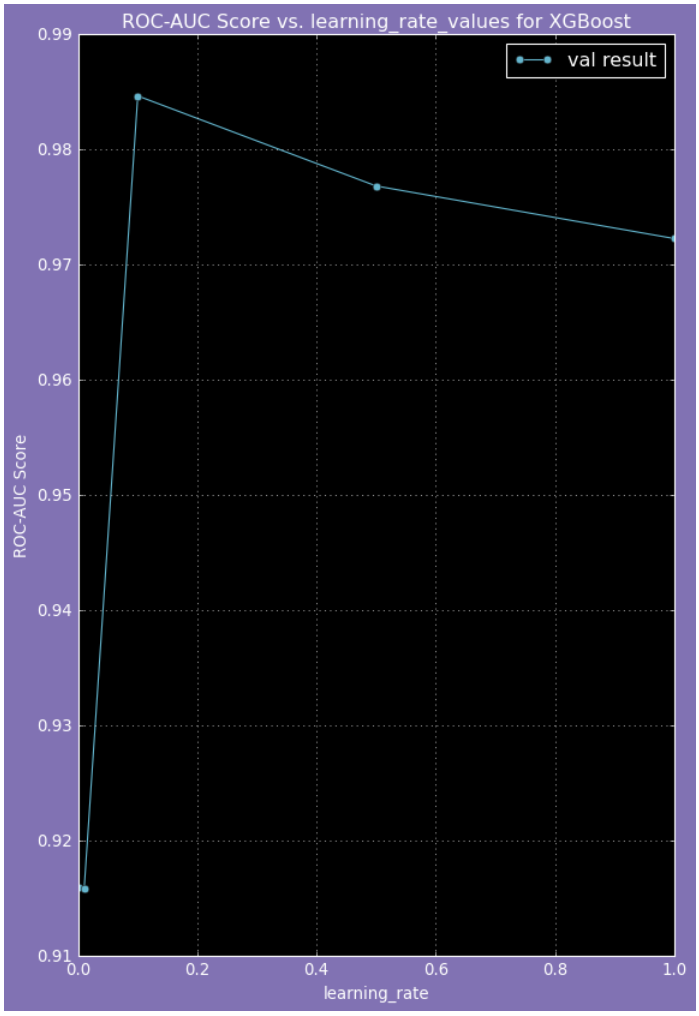

Figura 22: Curva ROC XGBoost II

Por último, en el XGBoost se comparan las tasas de aprendizaje (learning rates) hiperparámetro utilizado en el algoritmo de XGBoost, que es una implementación mejorada del algoritmo de Gradient Boosting. La tasa de aprendizaje controla la contribución de cada árbol al modelo final.

En XGBoost, se agrega un nuevo árbol de decisión en cada iteración para mejorar gradualmente el modelo. La tasa de aprendizaje determina cuánto se ajustan los valores de los árboles existentes en cada iteración. Una tasa de aprendizaje baja significa que los árboles se ajustan lentamente, mientras que una tasa de aprendizaje alta permite ajustes más rápidos.

Una tasa de aprendizaje alta puede conducir a un modelo más rápido, pero también puede hacer que el modelo se sobreajuste a los datos de entrenamiento. Por otro lado, una tasa de aprendizaje baja puede ayudar a evitar el sobreajuste, pero puede requerir más iteraciones para que el modelo alcance un buen rendimiento [40].

Las gráficas generadas en oversampling son:

Logistic Regression:

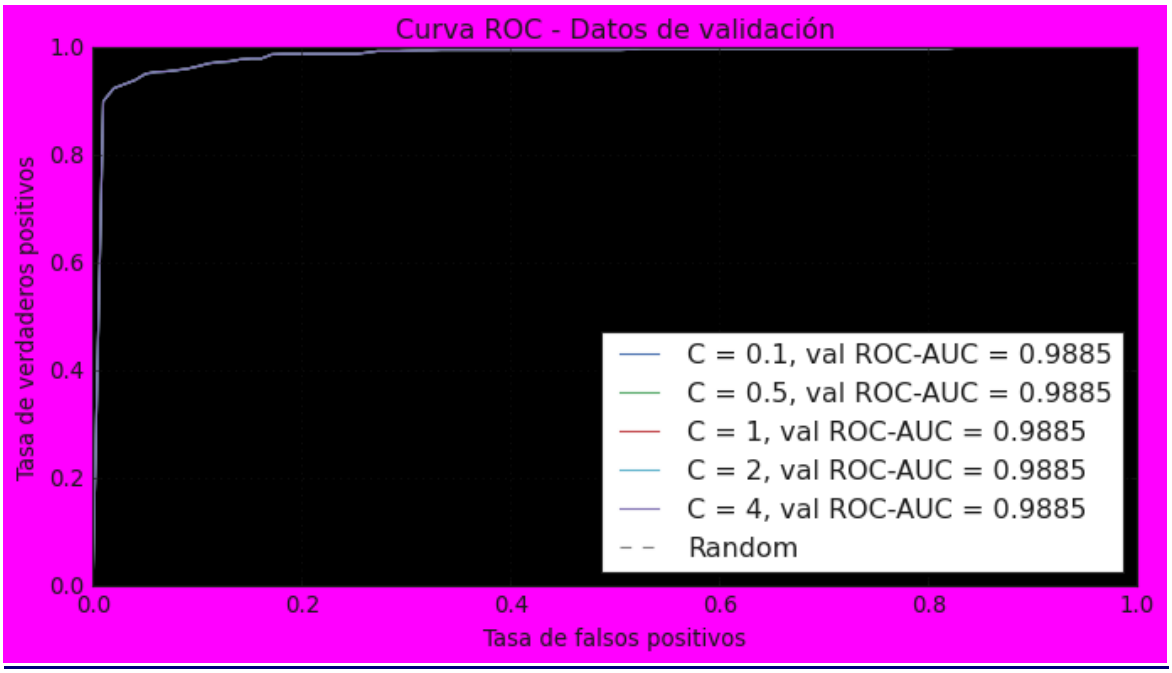

Figura 23: Curva ROC Logistic Regression - Oversampling

KNN

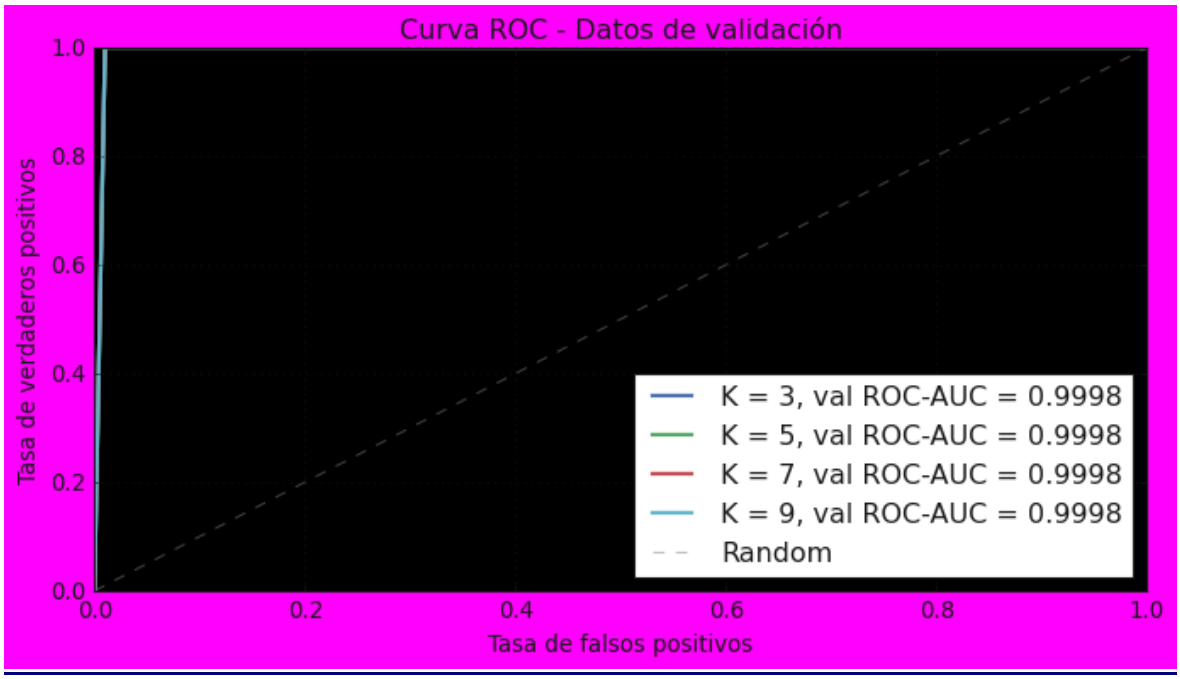

Figura 24: Curva ROC KNN - Oversampling

Decision Tree

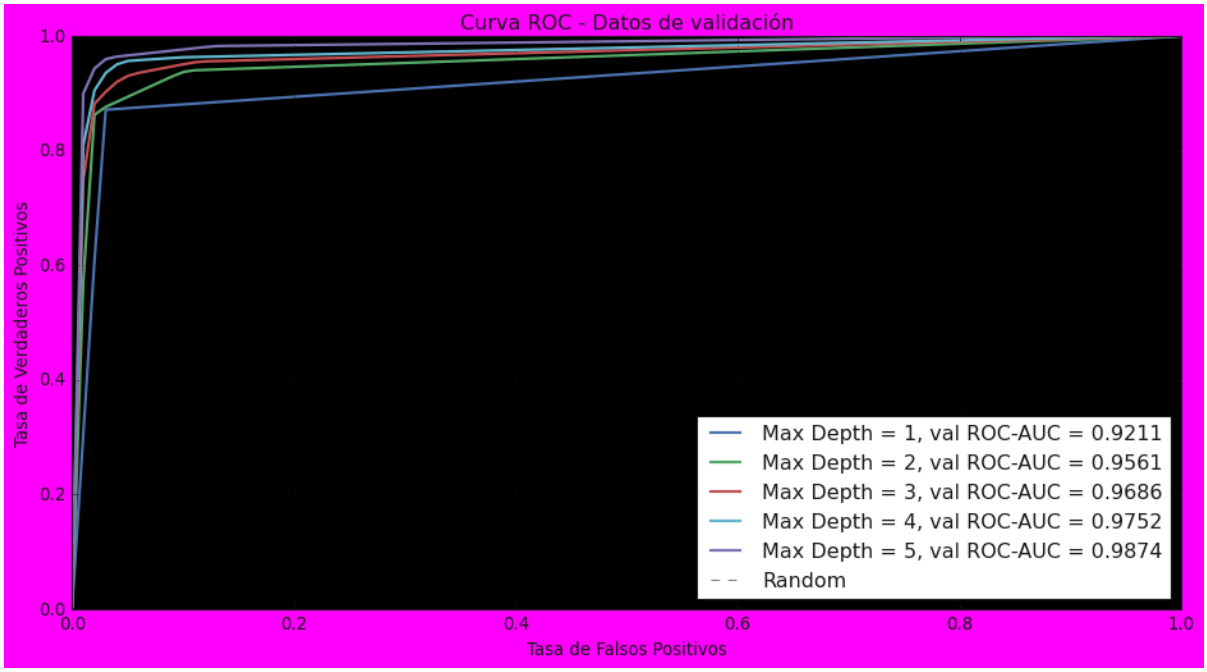

Figura 25: Curva ROC Decision Tree - Oversampling

Random Forest

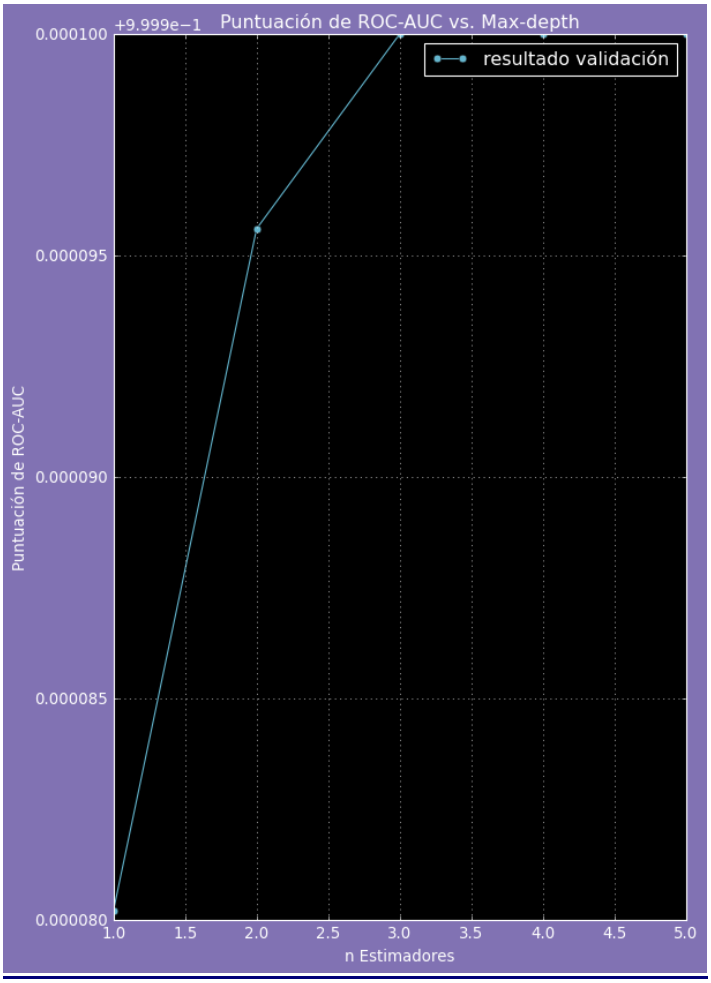

Figura 26: Curva ROC Random Forest Oversampling

XGBoost

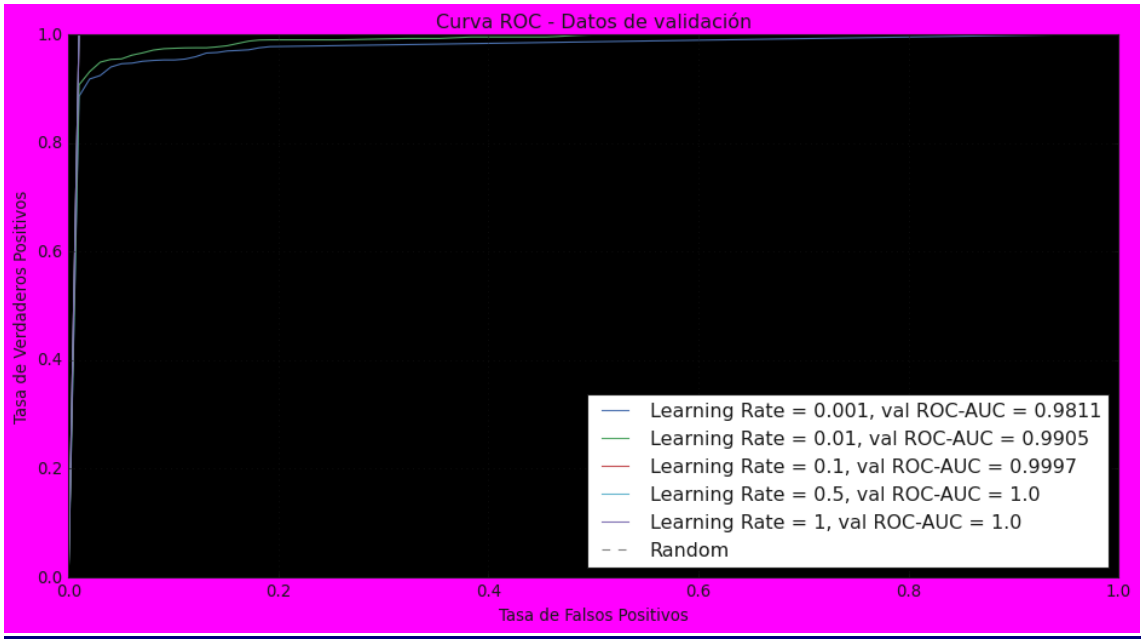

Figura 27: Curva ROC XGBoost - Oversampling

Para la SMOTE obtenemos resultados similares en este sentido:

Logistic Regression

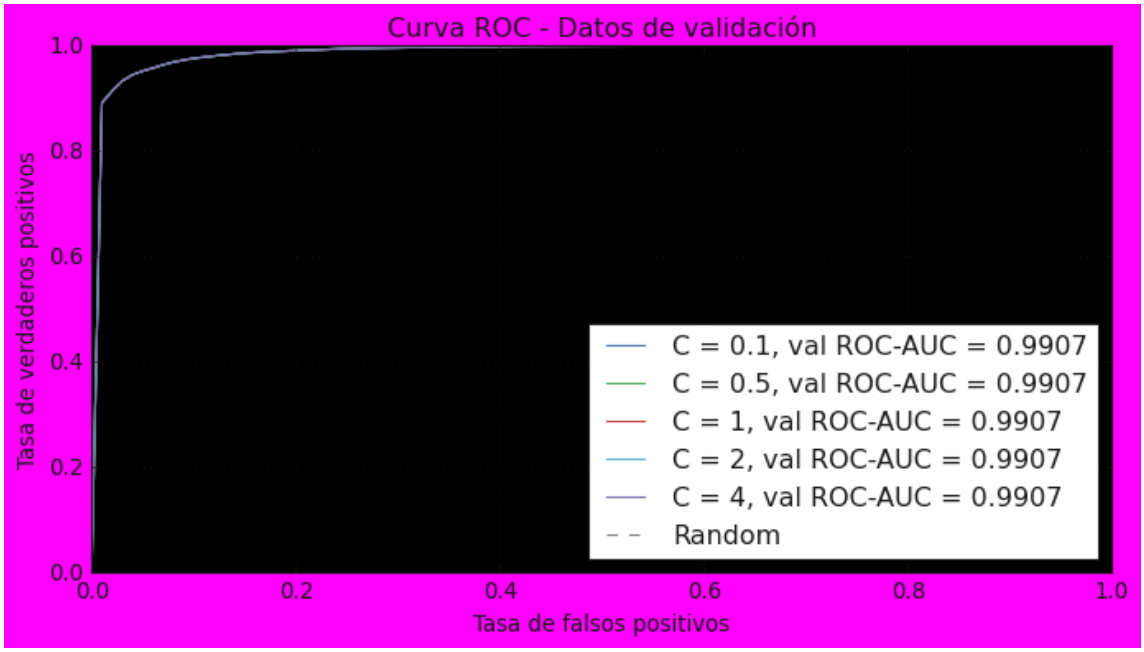

Figura 28: Curva ROC Logistic Regression - SMOTE I

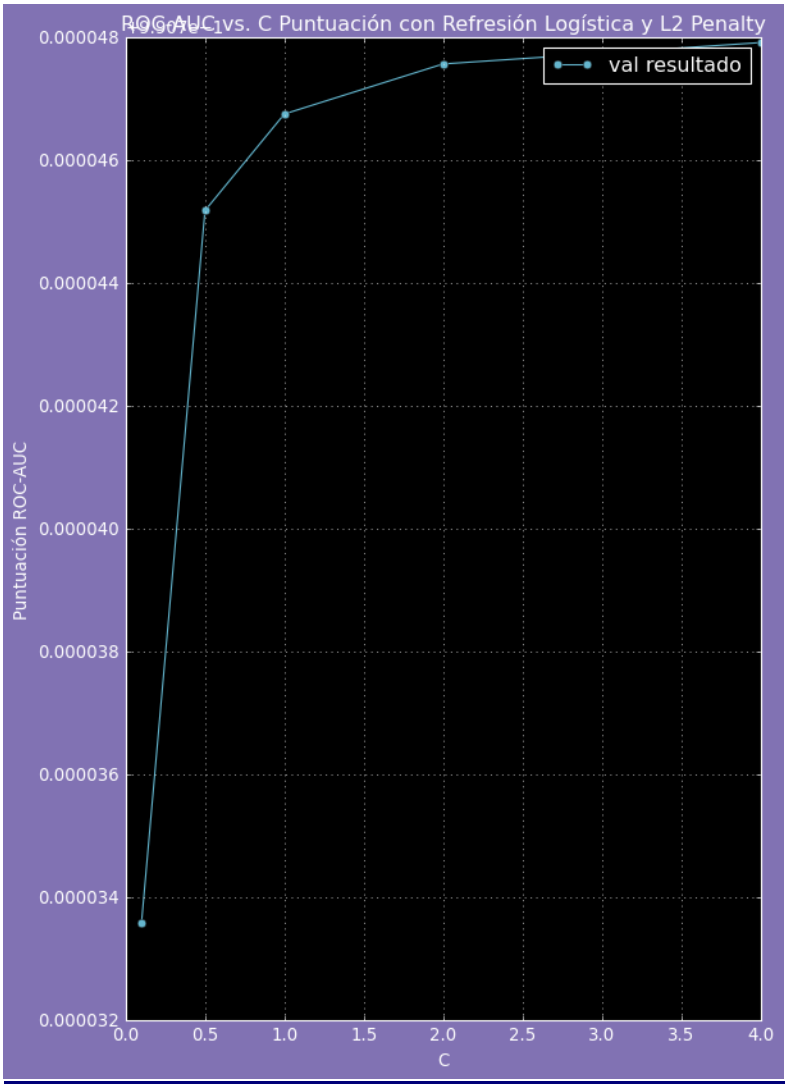

Figura 29: Curva ROC Logistic Regression - SMOTE II

Decision Tree:

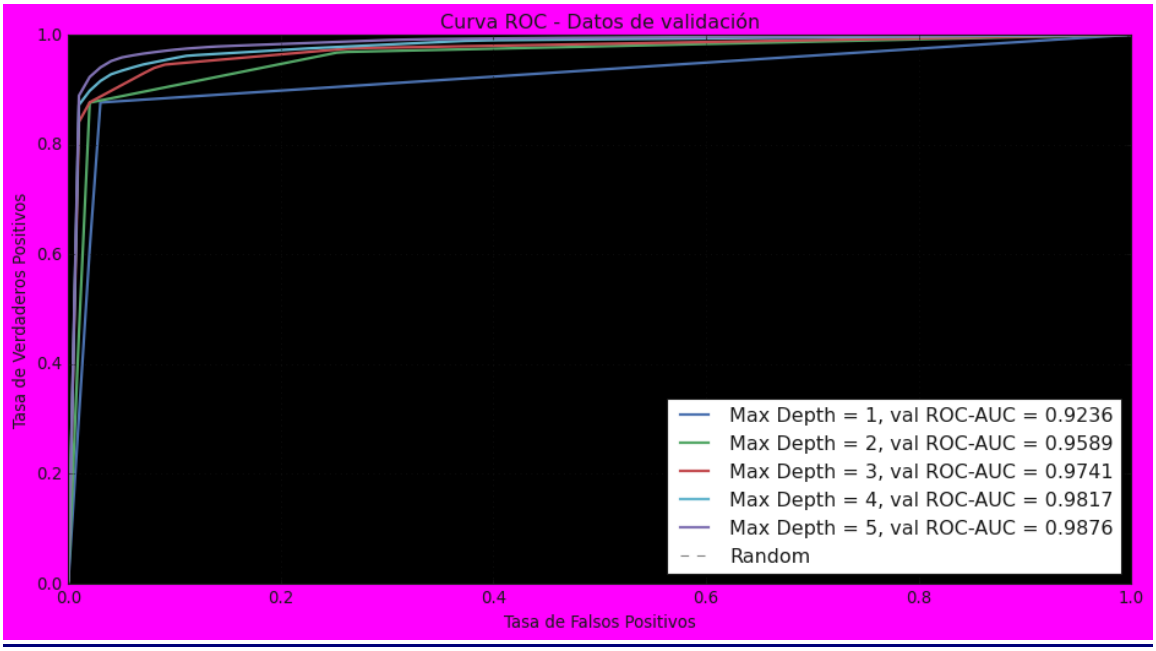

Figura 30: Curva ROC Decision Tree - SMOTE I

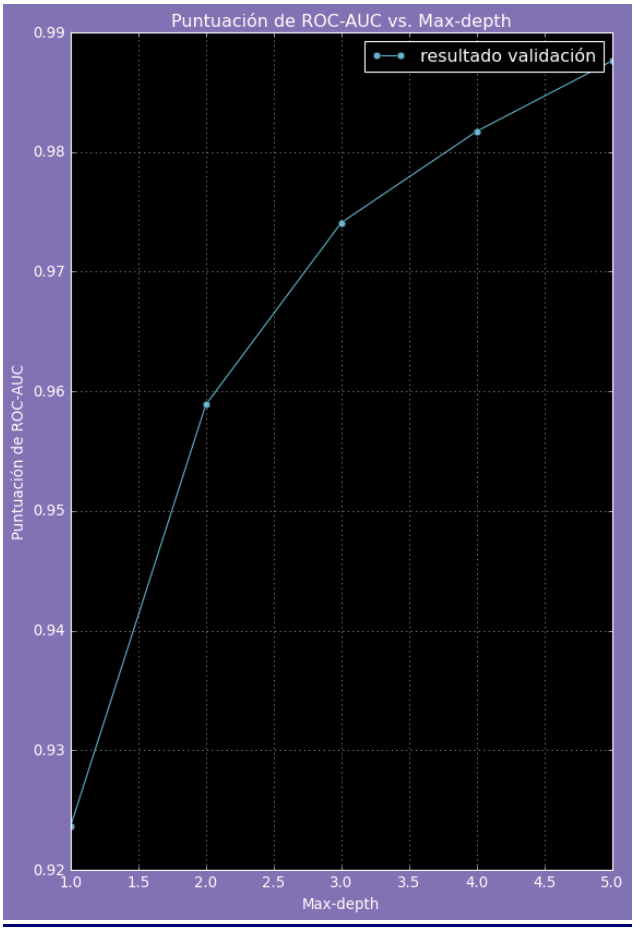

Figura 31: Curva ROC Decision Tree - SMOTE II

Random Forest:

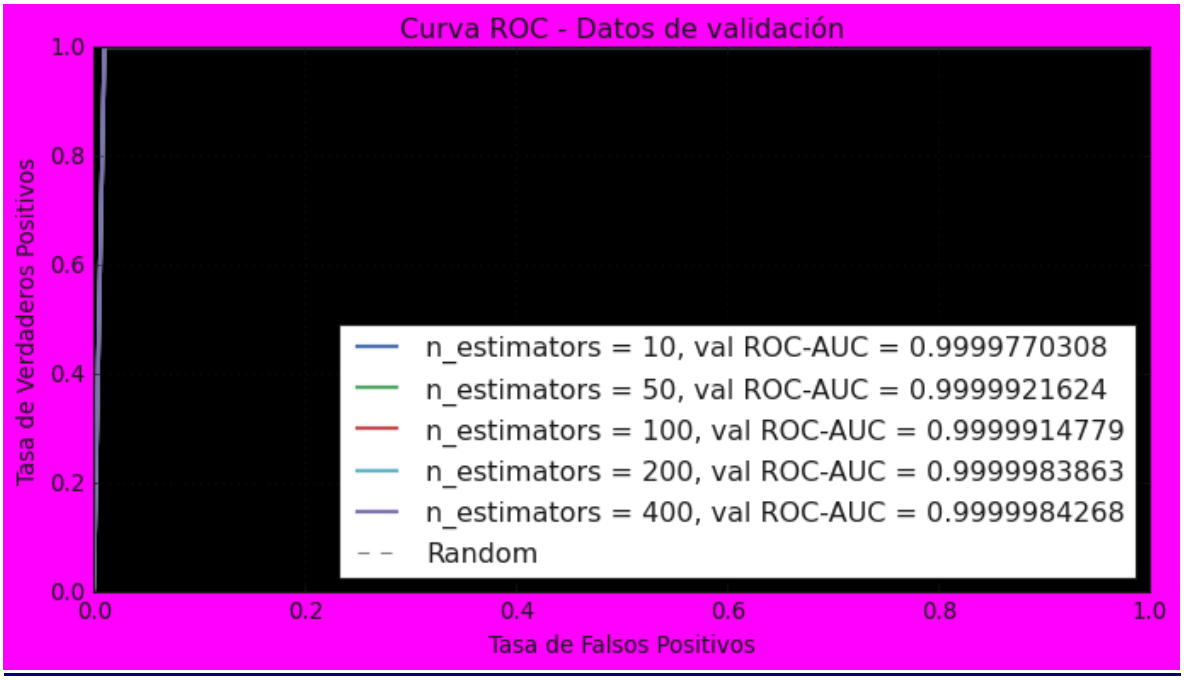

Figura 32: Curva ROC Random Forest - SMOTE

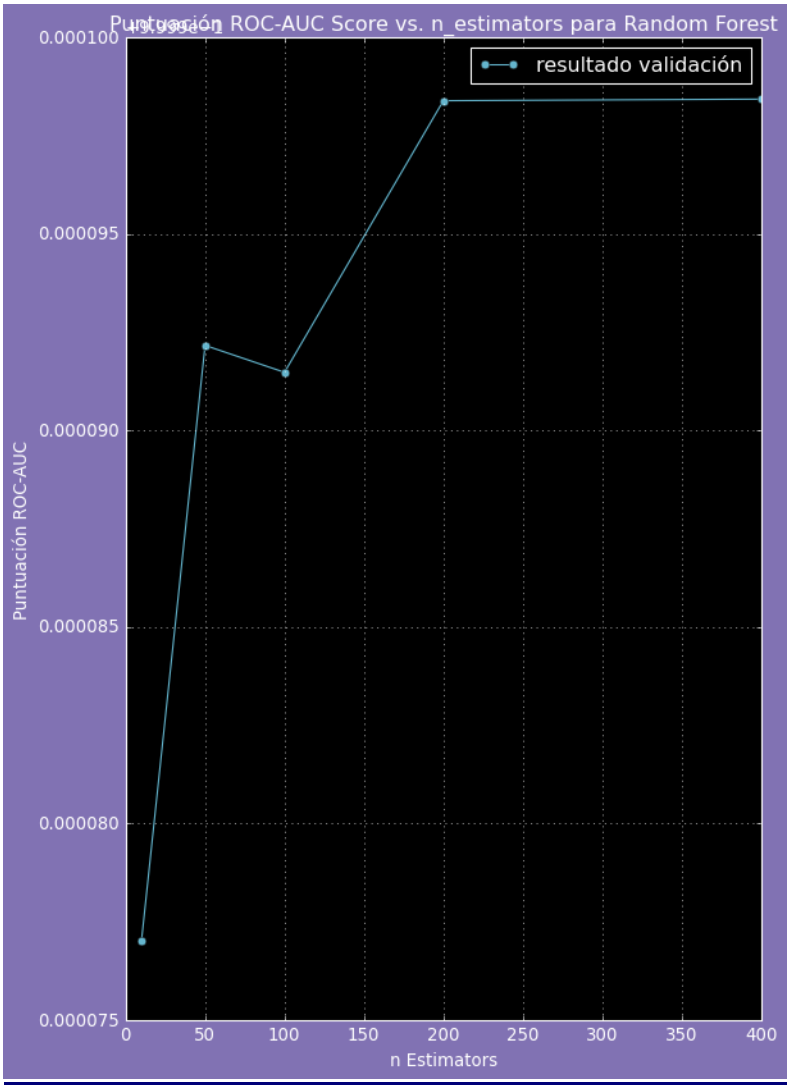

Figura 33: Curva ROC Random Forest - SMOTE II

XGBoost:

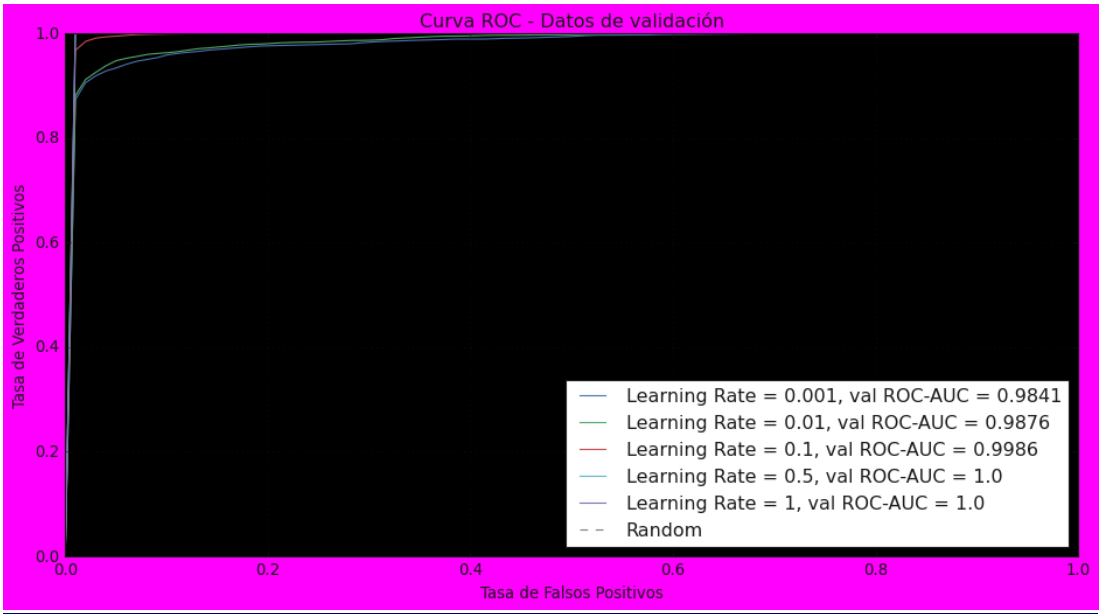

Figura 34: Curva ROC XGBoost - SMOTE I

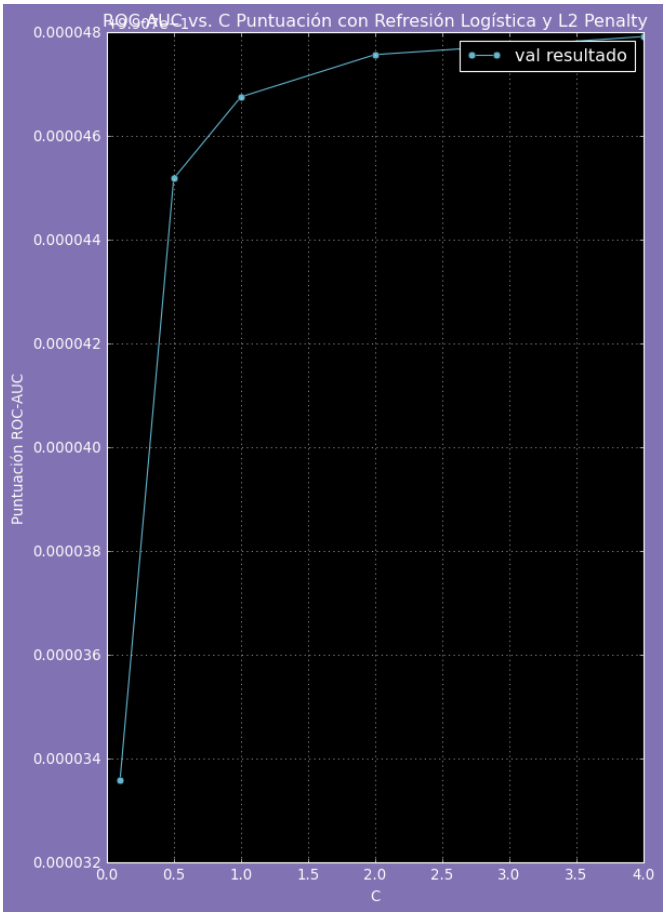

Figura 35: Curva ROC XGBoost - SMOTE II

8.2. Codificación

En este apartado del anexo se plasman las líneas de código empleadas para la generación de los modelos. Si bien es cierto que el script está compuesto por cerca de 6500 líneas esto es debido a que para cada algoritmo la estructura ha sido muy similar y únicamente se han ido adaptando las funciones definidas en las librerías de Python en función del modelo.

De la misma manera, para las representaciones gráficas el código ha sido reutilizado facilitando así la reproducción de las características más destacables en cada ocasión.

En cuanto a las librerías:

```
"""
Created on Mon Apr 10 15:24:43 2023

@author: Alvaro Artola Moreno
"""
import os
import numpy as np
import pandas as pd
pd.set_option("display.max_columns", None)
pd.set_option('float_format', '{:f}'.format)

import matplotlib.pyplot as plt
import matplotlib.colors as colors
import matplotlib.patches as mpatches
from matplotlib import colors
#%matplotlib inline
import seaborn as sns

from sklearn import metrics
from sklearn import preprocessing

from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import  RobustScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer

from sklearn.manifold import TSNE
from sklearn.decomposition import PCA, TruncatedSVD

from sklearn.model_selection import KFold, GridSearchCV, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
import xgboost as xgb
from sklearn import linear_model

from sklearn.metrics import roc_auc_score

from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import f1_score, classification_report
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score, accuracy_score, classification_report

from sklearn.pipeline import make_pipeline
from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import NearMiss

from scipy import interp
```

De donde hay que destacar:

- import seaborn as sns: Importa la biblioteca seaborn, que proporciona una interfaz de alto nivel para crear visualizaciones estadísticas atractivas y informativas.

- from sklearn import metrics: Importa el submódulo metrics de la biblioteca scikit-learn (sklearn), que contiene funciones para la evaluación de modelos de aprendizaje automático.

- from sklearn import preprocessing: Importa el submódulo preprocessing de scikit-learn, que proporciona funciones para la transformación y preprocesamiento de datos antes de ajustar un modelo.

- from sklearn.model_selection import ...: Importa varias clases y funciones relacionadas con la selección de modelos y evaluación del rendimiento, como train_test_split, KFold, GridSearchCV, etc.

- from sklearn.linear_model import ...: Importa clases de modelos lineales de scikit-learn, como LogisticRegression y LinearRegression.

- from sklearn.ensemble import ...: Importa clases de modelos de ensamble, como RandomForestClassifier, que implementa el algoritmo de bosques aleatorios.

- from sklearn.neighbors import ...: Importa clases relacionadas con el aprendizaje basado en vecinos más cercanos, como KNeighborsClassifier.

- from sklearn.tree import ...: Importa clases relacionadas con árboles de decisión, como DecisionTreeClassifier.

- from xgboost import ...: Importa clases y funciones relacionadas con el algoritmo XGBoost, como XGBClassifier.

- import linear_model: Importa el módulo linear_model de scikit-learn.

- from sklearn.metrics import ...: Importa varias métricas de evaluación de modelos, como roc_auc_score, f1_score, precision_score, etc.

- from imblearn.metrics import ...: Importa métricas específicas para conjuntos de datos desequilibrados de la biblioteca imbalanced-learn, como classification_report_imbalanced.

- from sklearn.pipeline import make_pipeline: Importa la función make_pipeline de scikit-learn, que se utiliza para crear pipelines de procesamiento de datos y modelos de aprendizaje automático.

- from imblearn.pipeline import make_pipeline as imbalanced_make_pipeline: Importa la función make_pipeline de la biblioteca imbalanced-learn, pero se le asigna el alias imbalanced_make_pipeline para evitar conflictos con make_pipeline de scikit-learn.

- from imblearn.over_sampling import SMOTE: Importa la clase SMOTE de imbalanced-learn, que se utiliza para aplicar la técnica de sobremuestreo sintético para abordar el desequilibrio de clases.

- from imblearn.under_sampling import NearMiss: Importa la clase NearMiss de imbalanced-learn, que se utiliza para aplicar la técnica de submuestreo para abordar el desequilibrio de clases.

- from scipy import interp: Importa la función interp del submódulo scipy para realizar interpolaciones.

Para la obtención de las características de la base de datos:

```
# Características presentes en los datos
print("=============================================")
print("Tipos de datos de las columnas en la base de datos:", data.dtypes)
print("=============================================")
print("Forma de la base de datos:", data.shape)
print("=============================================")
print("Información sobre la base de datos:", data.info())
data.head()

# Verificación de los valores faltantes presentes en cada columna
total = data.isnull().sum().sort_values(ascending=False)
percent = (data.isnull().sum() / data.isnull().count() * 100).sort_values(ascending=False)
pd.concat([total, percent], axis=1, keys=['Total', 'Porcentaje'])

# Tratamiento de outliers. A pesar de que la base de datos original ha sido tratada con PCA
# y se asume que los outliers ya han sido tratados

# Calcular el IQR (Rango Intercuartílico) para cada columna
Q1 = data.quantile(0.25)
Q3 = data.quantile(0.75)
IQR = Q3 - Q1
print(IQR)

# Eliminar valores atípicos basados en el IQR
"""
El código que se muestra comentado a continuación pretende eliminar valore atípicos de la muestra, sin embargo,
con esto eliminamos por completo las transacciones fraudulentas y, por tanto, insertamos un bias que no nos permite seguir con el
análisis del mejor modelo para predecir futuras transacciones fraudulentas.

"""
# data = data[~((data < (Q1 - 1.5 * IQR)) | (data > (Q3 + 1.5 * IQR))).any(axis=1)]
"""
Como alternativa tenemos utilizamos Median Imputation para deshacernos de los valores atípicos.
Esto nos premite reemplazar los valores extremos por valores en la media.

"""

plt.figure(figsize =(25,10))
data.boxplot(column = ['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11', 'V12', 'V13', 'V14', 'V15', 'V16', 'V17',
plt.xticks(rotation = 90)
plt.show()
```

```python
# Observacion de la distribucion de las clases presentes en la base de datos
"""
Este código calcula el número de transacciones clasificadas como "No fraudulentas" y "Fraudulentas" en un DataFrame dado "data".
Luego, calcula el porcentaje de cada clase y lo almacena en las variables "normal_share" y "fraud_share", respectivamente.
Finalmente, se imprime el porcentaje de transacciones no fraudulentas y fraudulentas en el DataFrame utilizando una cadena de formato.

"""

classes = data['Class'].value_counts()
normal_share = round(classes[0] / data['Class'].count() * 100, 2)
fraud_share = round(classes[1] / data['Class'].count() * 100, 2)
print("No fraudulentas: {} %".format(normal_share))
print("Fraudulentas: {} %".format(fraud_share))

print("===============================================")

"""
Este código calcula el porcentaje de transacciones normales y fraudulentas en el conjunto de datos,
y calcula el porcentaje de desequilibrio entre las dos clases.
La entrada es un DataFrame de Pandas "data" que contiene datos de transacciones,
con una columna "Class" que indica si la transacción es normal (0) o fraudulenta (1).
La salida incluye el porcentaje de transacciones normales y transacciones fraudulentas,
así como el porcentaje de desequilibrio entre ambas.
El porcentaje de desequilibrio se calcula dividiendo el porcentaje de transacciones fraudulentas por
el porcentaje de transacciones normales, y multiplicando el resultado por 100.
"""

classes = data['Class'].value_counts()
normal_share = classes[0] / data['Class'].count() * 100
fraud_share = classes[1] / data['Class'].count() * 100

print("Normal_share =", normal_share, "\n", "Fraud_share =", fraud_share)
print("===============================================")

imbalance = (fraud_share / normal_share) * 100
print('Porcentaje de desequilibrio = ' + str(imbalance))

```

```python

Por último, también podemos observar la distribución en un gráfico circular (pie chart)
"""

with plt.style.context('dark_background'):
    plt.figure(figsize=(20, 6), facecolor='m')

fig, ax = plt.subplots(1, 2, figsize=(18, 4))

classes.plot(kind='bar', rot=0, ax=ax[0])
ax[0].set_title('Distribución de Clases por Número \n (0: No Fraude | 1: Fraude)')
ax[0].set_ylabel('Número de Transacciones')
ax[0].set_xlabel('Clase')

for i in ax[0].containers:
    ax[0].bar_label(i, label_type='edge', labels=i.datavalues.astype(int))

(classes / data['Class'].count() * 100).plot(kind='bar', rot=0, ax=ax[1])
ax[1].set_title('Distribuciones por Porcentaje \n (0: No Fraude | 1: Fraude)')
ax[1].set_ylabel('Porcentaje de Transacciones')
ax[1].set_xlabel('Clase')

for i in ax[1].containers:
    ax[1].bar_label(i, label_type='edge', labels=[f'{val:.2f}%' for val in i.datavalues])

plt.show()

sns.set(style="whitegrid")
labels = ["No Fraude", "Fraude"]
sizes = data["Class"].value_counts(sort=True)

colors = ["aqua", "red"]
explode = (0.05, 0)

plt.figure(figsize=(8, 8))
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct="%1.1f%%", shadow=True)
plt.title("Fraudes en los datos")
plt.legend()
plt.show()
```

```
    Notas:
    Este código crea un gráfico de dispersión de la variable Tiempo en el eje x y la variable Clase en el eje y.
    Los puntos en el gráfico están codificados por colores utilizando el parámetro cmap en función de si una transacción es fraudulenta o no.
    Agregamos una barra de colores para mostrar la correspondencia entre el color y la clase.
    También se agrega una leyenda para aclarar los colores utilizados para cada clase.

    """
    # A pesar de estar definida al inicio da error cuando no se vuelve a escribir aqui
    from matplotlib import colors

    cmap = colors.ListedColormap(['white', 'red'])

    with plt.style.context('dark_background'):
        plt.figure(figsize=(25, 10), facecolor='m')

    # Utiliza el parámetro cmap para codificar por colores los puntos
    plt.scatter(x=data["Time"], y=data["Class"], c=data["Class"], cmap=cmap)

    # Establece los límites de la barra de colores
    plt.colorbar(ticks=[0, 1], orientation='horizontal', aspect=30, pad=0.15)

    plt.title("Gráfico de dispersión Tiempo vs Clase", fontsize=25)
    plt.xlabel("Clase", fontsize=15)
    plt.ylabel("Tiempo", fontsize=15)
    plt.grid()

    # Crea una leyenda con las dos etiquetas de clase
    elementos_leyenda = [plt.scatter([], [], marker='o', color='white', label='0 - No Fraudulento'),
                         plt.scatter([], [], marker='o', color='red', label='1 - Fraudulento')]
    plt.legend(handles=elementos_leyenda, loc='upper center')

    plt.show()
```

```
    Notas:
    Esta función utiliza la biblioteca matplotlib para crear un gráfico de dispersión de Amount vs Class con puntos codificados
    por colores basados en la etiqueta de Clase (0 - No Fraudulento o 1 - Fraudulento). El mapa de colores utilizado para los puntos
    se crea utilizando colors.ListedColormap de la biblioteca matplotlib. La barra de colores para el mapa de colores se crea
    utilizando plt.colorbar con marcas en los valores 0 y 1 y una orientación de 'horizontal'. Los parámetros aspect y pad
    se establecen en 30 y 0.15, respectivamente, para ajustar la posición y el tamaño de la barra de colores.

    El título del gráfico de dispersión, la etiqueta del eje x y la etiqueta del eje y se establecen utilizando plt.title,
    plt.xlabel y plt.ylabel, respectivamente.
    Los tamaños de fuente para el título y las etiquetas de los ejes se pueden ajustar utilizando el parámetro fontsize.

    Se crea una leyenda utilizando plt.legend y el parámetro handles para crear gráficos de dispersión con puntos vacíos y los
    colores y etiquetas correspondientes para cada valor de Clase.

    Finalmente, se utiliza plt.grid para mostrar una cuadrícula en el gráfico de dispersión y se llama a plt.show para mostrar
    el gráfico.

    """
    cmap = colors.ListedColormap(['white', 'red'])

    with plt.style.context('dark_background'):
        plt.figure(figsize=(20,10), facecolor='m')

    # Utiliza el parámetro cmap para codificar por colores los puntos
    plt.scatter(x=data["Amount"], y=data["Class"], c=data["Class"], cmap=cmap)

    # Establece los límites de la barra de colores
    plt.colorbar(ticks=[0, 1], orientation='horizontal', aspect=30, pad=0.15)

    plt.title("Gráfico de dispersión Amount vs Class", fontsize=25)
    plt.ylabel("Clase", fontsize=15)
    plt.xlabel("Amount", fontsize=15)
    plt.grid()

    # Crea una leyenda con las dos etiquetas de clase
    elementos_leyenda = [plt.scatter([], [], marker='o', color='white', label='0 - No Fraudulento'),
                         plt.scatter([], [], marker='o', color='red', label='1 - Fraudulento')]
    plt.legend(handles=elementos_leyenda, loc='upper center')

    plt.show()
```

```
    Comentarios:
    Esta función crea una cuadrícula de gráficos de densidad utilizando la función kdeplot() de seaborn. La cuadrícula
    se estructura en 10 filas y 3 columnas para acomodar todas las variables en el DataFrame de entrada. La columna Class
    se excluye de la lista de variables a trazar.

    Los datos se separan primero por Class en dos subconjuntos, t0 y t1, utilizando el método loc() en el DataFrame. Luego,
    los subconjuntos se trazan utilizando kdeplot() con un ancho de banda de 0.5 para estimar la función de densidad. Los dos
    subconjuntos se distinguen por diferentes colores.


    """
    # Graficar todas las variables en displot para visualizar la distribución
    variables = list(data.columns.values)

    # Eliminar la columna Class de la lista
    variables.remove("Class")

    i = 0
    t0 = data.loc[data['Class'] == 0]
    t1 = data.loc[data['Class'] == 1]

    with plt.style.context('dark_background'):
        plt.figure()
    fig, ax = plt.subplots(10, 3, figsize=(30, 45), facecolor='b')

    for variable in variables:
        i += 1
        plt.subplot(10, 3, i)
        sns.kdeplot(t0[variable], bw=0.5, label="0")
        sns.kdeplot(t1[variable], bw=0.5, label="1")
        plt.xlabel(variable, fontsize=16)
        plt.ylabel("Densidad", fontsize=16)
        locs, labels = plt.xticks()
        plt.tick_params(axis='both', which='major', labelsize=12)
        plt.grid()
    plt.show()

    # Eliminamos columnas innecesarias para la modelización
    data = data.drop("Time", axis=1)
    data.head()

    """
    División de la base de datos para obtener una muestra de entrenamiento y otra de validación

    """
    # y= #class variable
    y= data["Class"]
    X = data.drop("Class", axis = 1)
    y.shape,X.shape
```

```
"""
Utilizamos la función train_test_split() de scikit-learn para dividir el conjunto de datos X y sus etiquetas correspondientes
"y" en conjuntos de entrenamiento y prueba. La división se realiza con una proporción de 80/20 para el conjunto de prueba y el valor
de random_state se establece en 42 para garantizar reproducibilidad.

El parámetro stratify se establece en "y", lo que significa que la división se realiza de manera que se preserva la proporción
de muestras para cada clase en "y" en los conjuntos de entrenamiento y prueba.

La salida resultante es una tupla que contiene la forma del conjunto de entrenamiento (X_train y y_train) y la forma del
 conjunto de prueba (X_test y y_test).

La función train_test_split() se utiliza comúnmente en aprendizaje automático para dividir un conjunto de datos en conjuntos de entrenamiento
y prueba, a fin de evaluar el rendimiento de un modelo en datos no vistos.

El conjunto de entrenamiento se utiliza para ajustar el modelo, mientras que el conjunto de prueba se utiliza para evaluar el rendimiento
del modelo en nuevos datos no vistos.

"""

# Dividir los datos en una proporción de 80:20 para entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape

# Verificando la distribución de las etiquetas de clase
print("Recuento de transacciones fraudulentas para el conjunto de datos completo:", np.sum(y))
print("Recuento de transacciones fraudulentas para el conjunto de entrenamiento:", np.sum(y_train))
print("Recuento de transacciones fraudulentas para el conjunto de prueba:", np.sum(y_test))

# Guardamos el conjunto de prueba para evaluación
X_test_saved = X_test.copy()
y_test_saved = y_test.copy()
print("Conjunto de prueba guardado (X_test y y_test)")

# Dado que ya se ha realizado PCA en el conjunto de datos desde las características V1 hasta V28, solo escalaremos el campo Amount
scaler = RobustScaler()

# Escalar los datos de entrenamiento
X_train[["Amount"]] = scaler.fit_transform(X_train[["Amount"]])

# Transformar los datos de prueba
X_test[["Amount"]] = scaler.transform(X_test[["Amount"]])

X_train.head()
X_test.head()
```

```
"""
Definimos una función para trazar histogramas de las variables en un conjunto de datos dado para ver la asimetría.

Parámetros:
X_train: un DataFrame de pandas con las variables que se van a trazar como columnas.

Retorno:
Una cuadrícula de histogramas, donde cada histograma representa la distribución de una variable en el DataFrame de entrada.

Comentarios:
Los histogramas son útiles para comprender la distribución de una variable y para identificar cualquier posible asimetría.
Si un histograma es simétrico, indica que los datos están distribuidos de forma normal, mientras que un histograma que se
sesga hacia la izquierda o hacia la derecha sugiere que los datos están sesgados en esa dirección.
Al trazar el histograma de cada variable en el conjunto de entrenamiento, podemos tener una idea de la distribución de los
datos y si es necesario realizar alguna transformación antes de modelar.

"""

variables = X_train.columns

with plt.style.context('dark_background'):
    fig, axes = plt.subplots(10, 3, figsize=(30, 45), facecolor='b')
axes = axes.flatten()

for i, ax in enumerate(axes):
    if i < len(variables):
        sns.histplot(X_train[variables[i]], ax=ax)
        ax.set_title(variables[i], fontsize=20)
        ax.set_ylabel("Conteo", fontsize=20)  # establecer etiqueta del eje y del subplot
        ax.tick_params(axis='both', labelsize=15)
        ax.set_xlabel('')  # establecer cadena vacía como etiqueta del eje x del subplot

plt.tight_layout()
plt.show()

"""
Verificamos la asimetría de las características y para ello creamos una lista de nombres de variables llamada var,
y luego calculamos la asimetría de cada variable utilizando el método skew() de un DataFrame de Pandas.
Los valores de asimetría los almacenamos en una lista llamada skew_list.

"""
var = X_train.columns
skew_list = []
for i in var:
    skew_list.append(X_train[i].skew())

tmp = pd.concat([pd.DataFrame(var, columns=["Características"]), pd.DataFrame(skew_list, columns=["Asimetría"])], axis=1)
tmp.set_index("Características", inplace=True)
tmp
```

```
Existe asimetría en la distribución de las características mencionadas anteriormente. Por tanto, podemos hacer uso de
Power Transformer proporcionado por la biblioteca de preprocesamiento de sklearn para hacer que la distribución sea más gaussiana
Filtraremos las características que tienen una asimetría mayor que +1 o menor que -1:

"""
asimetricas = tmp.loc[(tmp["Asimetría"] > 1) | (tmp["Asimetría"] < -1)].index
asimetricas.tolist()

"""
En la clase PowerTransformer de scikit-learn, el parámetro 'copy' especifica si se debe realizar una copia
de los datos de entrada antes de aplicar la transformación o no.
De forma predeterminada, 'copy' se establece en True, lo que significa que los datos de entrada se copian
 a un nuevo objeto antes de aplicar la transformación.

La función preprocessing.PowerTransformer() es utilizada para transformar los datos y lograr una distribución
más similar a una distribución gaussiana.
Aquí, estamos aplicando la transformación a las características que tienen una asimetría mayor que 1 o menor que -1.

Se utiliza 'fit_transform' tanto para ajustar el transformador a los datos de entrenamiento como para transformar los datos,
mientras que 'transform' se utiliza solo para transformar nuevos datos basados en un transformador previamente ajustado.

"""

# Aplicamos preprocessing.PowerTransformer(copy=False) para ajustar y transformar los datos de entrenamiento y prueba
pt = preprocessing.PowerTransformer(method='yeo-johnson', copy=True) # crea una instancia de la clase PowerTransformer.
pt.fit(X_train)

X_train_pt = pt.transform(X_train)
X_test_pt = pt.transform(X_test)

y_train_pt = y_train
y_test_pt = y_test

print(X_train_pt.shape)
print(y_train_pt.shape)

var = X_train.columns
with plt.style.context('dark_background'):
    fig = plt.figure(figsize=(30,45), facecolor='b')
    fig.suptitle('Histogramas de las variables', fontsize=30)
    i=0
    for col in var:
        i += 1
        ax = fig.add_subplot(10,3, i)
        sns.histplot(X_train[col], ax=ax)
        ax.set_title(col, fontsize=20)
        ax.set_ylabel('Count', fontsize=15)
        ax.set_xlabel('')
    fig.subplots_adjust(hspace=0.5, wspace=0.2)
    plt.show()
```

Una vez se tienen codificadas las estadísticas el siguiente paso es la cosntrucción de los modelos. Como se comentaba anteriormente todos siguen la misma estructura y por sencillez tan solo se muestran los que se han realizado con la muestra original, es decir, con los datos desequilibrados:

```
"""
Construcción de modelos con datos desequilibrados

"""
# Class imbalance
y_train.value_counts()/y_train.shape

# Realizamos validación cruzada en X_train y y_train
# Inicializamos validador cruzado StratifiedKFold
skf = StratifiedKFold(n_splits=3, random_state=None, shuffle=False)
```

```
print("=========================================================================================================")

print("=================================== Regresión Logística: ================================")

# # Definimos los valores de C
C_values = [0.1, 0.5, 1, 2, 4]

# Inicializamos listas para almacenar las puntuaciones medias de ROC-AUC así como las tasas
# de falsos positivos medias para los datos de entrenamiento y validación
mean_roc_auc_scores_val = []
mean_precision_scores_val = []
mean_recall_scores_val = []
mean_f1_scores_val = []
mean_fpr = np.linspace(0, 1, 100)


fig, ax2= plt.subplots(figsize=(14, 7), facecolor='fuchsia')  # Set facecolor to black for dark background
# Establecemos el color del fondo para los subplots
ax2.set_facecolor('black')

# Recorremos cada valor de C
for c in C_values:
    print("C =", c, "Penalty = L2")
    cv_scores_val = []
    precision_val = []
    recall_val = []
    f1_val = []
    fprs_val = []
    tprs_val = []

    # Validacion cruzada
    for train_index, val_index in skf.split(X_train_pt, y_train_pt):
        start_time = time.time()
        print("Train:", train_index, "val:", val_index)
        X_train_cv, X_val_cv = X_train_pt[train_index], X_train_pt[val_index]
        y_train_cv, y_val_cv = y_train_pt.iloc[train_index], y_train_pt.iloc[val_index]

        logreg_classifier = linear_model.LogisticRegression(penalty='l2', C=c)
        logreg_classifier.fit(X_train_cv, y_train_cv)

        y_val_pred = logreg_classifier.predict_proba(X_val_cv)
        val_score = roc_auc_score(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        cv_scores_val.append(val_score)


        y_val_pred_binary = (y_val_pred[:, 1] >= 0.5).astype(int)

        precision_val.append(precision_score(y_val_cv, y_val_pred_binary))

        recall_val.append(recall_score(y_val_cv, y_val_pred_binary))
        f1_val.append(f1_score(y_val_cv, y_val_pred_binary))

        fpr_val, tpr_val, _ = roc_curve(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        tprs_val.append(interp(mean_fpr, fpr_val, tpr_val))
        tprs_val[-1][0] = 0.0
```

```
        fprs_val.append(fpr_val)
        elapsed_time = time.time() - start_time
        print("Tiempo transcurrido:", elapsed_time, "segundos")

    # Calcular las puntuaciones medias de ROC-AUC para los datos de validación

    mean_roc_auc_val = np.mean(cv_scores_val)
    mean_roc_auc_scores_val.append(mean_roc_auc_val)

    mean_precision_val = np.mean(precision_val)
    mean_precision_scores_val.append(mean_precision_val)

    mean_recall_val = np.mean(recall_val)
    mean_recall_scores_val .append(mean_recall_val)

    mean_f1_val = np.mean(f1_val)
    mean_f1_scores_val.append(mean_f1_val)

    print("Mejor puntuación media de ROC-AUC para los datos de validación:", mean_roc_auc_val)
    print("Puntuación media de precisión para el mejor C:", mean_precision_val)
    print("Puntuación media de recall para el mejor C:", mean_recall_val)
    print("Puntuación media de F1 para el mejor C:", mean_f1_val)

    print("-----------------------------------------------------------------------------------------------------------")

    # Plot ROC curve for the current value of C
    ax2.plot(mean_fpr, np.mean(tprs_val, axis=0), label='C = ' + str(c) + ', val ROC-AUC = ' + str(round(mean_roc_auc_val, 4)))


print("-----------------------------------------------------------------------------------------------------------")
print("\n")
print("==============================La Regresión Logística se ha ejecutado exitosamente:==================================")
print("\n")
print("-----------------------------------------------------------------------------------------------------------")
print("\n")
print("====================================Trama de la curva ROC AUC:=========================================")
print("\n")

# Plot ROC curve for random classifier
ax2.plot([0, 1], [0, 1], linestyle='--', color='black', label='Random', alpha=0.5)

# Establecer etiquetas y título para las curvas ROC
ax2.set_xlabel('Tasa de falsos positivos')
ax2.set_ylabel('Tasa de verdaderos positivos')
ax2.set_title('Curva ROC - Datos de validación')
ax2.legend(loc="lower right")
ax2.grid(True)
plt.show()

print("===========================================================================================================")

```

```
680     with plt.style.context('dark_background'):
681         plt.figure(figsize=(8, 12), facecolor='m')
682 
683         # Puntuaciones ROC-AUC vs C
684         plt.plot(C_values, mean_roc_auc_scores_val, 'co-')
685         plt.xlabel('C')
686         plt.ylabel('Puntuación ROC-AUC')
687         plt.title('ROC-AUC vs. C Puntuación con Refresión Logística y L2 Penalty')
688         plt.legend(['val resultado'], loc='upper right')
689         plt.grid()
690 
691         plt.show()
692 
693     print("\n")
694     print("=============================================================================================================")
695     print("\n")
696 
697     print("=========================Mejores puntuaciones promedio de ROC-AUC y mejores hiperparámetros:====================
698 
699     # Print mean ROC-AUC scores for train and val data for all values of C
700     print("Mean ROC-AUC scores for val data for all values of C: \n", mean_roc_auc_scores_val)
701 
702     # Find the index of maximum mean ROC-AUC score for val data
703     best_index = np.argmax(mean_roc_auc_scores_val)
704     best_c = C_values[best_index]
705     best_mean_roc_auc = mean_roc_auc_scores_val[best_index]
706 
707     print("Mejor valor de C:", best_c)
708     print("Mejor puntuación ROC-AUC media para los datos de validación:", best_mean_roc_auc)
709     print("Puntuación media de precisión para el mejor valor de C:", mean_precision_scores_val[best_index])
710     print("Puntuación media de recall para el mejor valor de C:", mean_recall_scores_val[best_index])
711     print("Puntuación media de F1 para el mejor valor de C:", mean_f1_scores_val[best_index])
712 
713     print("=============================================================================================================")
714     print("\n")
715     print("=============================================================================================================")
716 
717     print("=============================================================================================================")
718 
719     print("============ Sintonización de múltiples hiperparámetros (GridSearchCV) + Puntuación ROC_AUC del Mejor Modelo: =====
720 
721     """
722     Parámetros de Regresión Logística para validación cruzada estratificada con K-fold
723 
724     Con C el parámetro de regularización para la regresión logística.
725     folds es un objeto que define cuántos pliegues se deben utilizar en la validación cruzada.
726 
727     """
```

```
# Definir los parámetros a buscar en la cuadrícula
params = {"C": [0.01, 0.1, 0.5], "penalty": ["l1", "l2"]}

# Crear el clasificador de regresión logística
logreg_classifier = linear_model.LogisticRegression()

start_time = time.time()

# Crear el objeto GridSearchCV con validación cruzada estratificada
model_GridSearch = GridSearchCV(logreg_classifier,
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = skf,
                        n_jobs=-1, # 100% de la CPU
                        verbose = 1,
                        pre_dispatch = 6, # evitar fallo de memoria
                        return_train_score=True)

# Ajustar el objeto GridSearchCV para realizar la búsqueda de cuadrícula
model_GridSearch.fit(X_train_pt, y_train_pt)
end_time = time.time()

print("Tiempo transcurrido: {:.2f} segundos".format(end_time - start_time))

# Imprimir los resultados
print("Mejor puntuación ROC-AUC: ", model_GridSearch.best_score_)
print("Mejores hiperparámetros: ", model_GridSearch.best_params_)

cv_results = model_GridSearch.cv_results_

# Imprimir las puntuaciones medias de prueba para cada combinación de hiperparámetros
print("Puntuaciones medias de prueba:")
for mean_score, params in zip(cv_results["mean_test_score"], cv_results["params"]):
    print(params, mean_score)

# Imprimir el rango de cada combinación de hiperparámetros basado en la puntuación media de prueba
print("\nRango de cada combinación de hiperparámetros:")
for rank, params in enumerate(cv_results["params"]):
    print(rank+1, params)

# Imprimir la desviación estándar de las puntuaciones de prueba para cada combinación de hiperparámetros
print("\nDesviación estándar de las puntuaciones de prueba:")
for std_score, params in zip(cv_results["std_test_score"], cv_results["params"]):
    print(params, std_score)
```

```python
print("==================================== KNN: ====================================")

# import matplotlib.pyplot as plt
# import numpy as np
# from sklearn.model_selection import KFold
# from sklearn.neighbors import KNeighborsClassifier
# from sklearn.metrics import roc_auc_score, roc_curve
# from scipy import interp
# import time

# Numero de vecinos escogidos en la muestra
K_values = [3, 5, 7]

# Inicializar listas para almacenar las puntuaciones promedio de ROC-AUC
# y las tasas promedio de falsos positivos para los datos de entrenamiento y validación.
mean_roc_auc_scores_train = []
mean_roc_auc_scores_val = []
mean_fpr = np.linspace(0, 1, 100)

# Subplots para las curvas ROC
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 7), facecolor='fuchsia')
# Set background color for subplots
ax1.set_facecolor('black')
ax2.set_facecolor('black')

# Define the KFold object
# skf = KFold(n_splits=3, shuffle=False, random_state=42)

# Iteracion para cada valor de K
for k in K_values:
    print("K =", k)
    cv_scores_train = []
    cv_scores_val = []
    fprs_train = []
    tprs_train = []
    fprs_val = []
    tprs_val = []

    # Validación cruzada
    for train_index, val_index in skf.split(X_train_pt, y_train_pt):
        start_time = time.time()
        print("Entrenamiento:", train_index, "val:", val_index)
        X_train_cv, X_val_cv = X_train_pt[train_index], X_train_pt[val_index]
        y_train_cv, y_val_cv = y_train_pt.iloc[train_index], y_train_pt.iloc[val_index]

        knn_classifier = KNeighborsClassifier(n_neighbors=k)
        knn_classifier.fit(X_train_cv, y_train_cv)

        y_train_pred = knn_classifier.predict_proba(X_train_cv)
        train_score = roc_auc_score(y_true=y_train_cv, y_score=y_train_pred[:, 1])
        cv_scores_train.append(train_score)

        y_val_pred = knn_classifier.predict_proba(X_val_cv)
        val_score = roc_auc_score(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        cv_scores_val.append(val_score)

        fpr_train, tpr_train, _ = roc_curve(y_true=y_train_cv, y_score=y_train_pred[:, 1])
        tprs_train.append(interp(mean_fpr, fpr_train, tpr_train))
        tprs_train[-1][0]

        fpr_val, tpr_val, _ = roc_curve(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        tprs_val.append(interp(mean_fpr, fpr_val, tpr_val))
        tprs_val[-1][0]

        elapsed_time = time.time() - start_time
        print("Tiempo transcurrido:", elapsed_time, "segundos")

    # Calcular las puntuaciones promedio de ROC-AUC y las tasas promedio de falsos positivos para los datos de entrenamiento y validación.
    mean_roc_auc_train = np.mean(cv_scores_train)
    mean_roc_auc_scores_train.append(mean_roc_auc_train)
    mean_tpr_train = np.mean(tprs_train, axis=0)
    mean_fpr_train = mean_fpr
    ax1.plot(mean_fpr_train, mean_tpr_train, linestyle='-', lw=2, label='K = ' + str(k) + ', Train ROC-AUC = ' + str(round(mean_roc_auc_train, 4)).format(k))

    mean_roc_auc_val = np.mean(cv_scores_val)
    mean_roc_auc_scores_val.append(mean_roc_auc_val)
    mean_tpr_val = np.mean(tprs_val, axis=0)
    mean_fpr_val = mean_fpr
    ax2.plot(mean_fpr_val, mean_tpr_val, linestyle='-', lw=2, label='K = ' + str(k) + ', val ROC-AUC = ' + str(round(mean_roc_auc_val, 4)).format(k))

    print("Puntuación promedio de ROC-AUC para los datos de entrenamiento:", mean_roc_auc_train)
    print("Media de ROC-AUC para los datos de validación:", mean_roc_auc_val)
```

```
# ROC curves de entrenamiento y validación
ax1.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Random', alpha=0.5)
ax1.set_xlim([0, 1])
ax1.set_ylim([0, 1])
ax1.set_xlabel('Tasa de falsos positivos')
ax1.set_ylabel('Tasa de verdaderos positivos')
ax1.set_title('Curva ROC - Datos de entrenamiento')
ax1.legend(loc='lower right')
ax1.grid(True)

ax2.plot([0, 1], [0, 1], color='gray', linestyle='--', label='Random', alpha=0.5)
ax2.set_xlim([0, 1])
ax2.set_ylim([0, 1])
ax2.set_xlabel('Tasa de falsos positivos')
ax2.set_ylabel('Tasa de verdaderos positivos')
ax2.set_title('Curva ROC - Datos de entrenamiento')
ax2.legend(loc='lower right')
ax2.grid(True)

plt.show()

print("=============================================================================================================")

# Estilos y tamaños de la gráfica
with plt.style.context('dark_background'):
    plt.figure(figsize=(8, 12), facecolor='m')

    # Plot ROC-AUC scores vs C
    plt.plot(K_values, mean_roc_auc_scores_train, 'yo-')
    plt.plot(K_values, mean_roc_auc_scores_val, 'co-')
    plt.xlabel('K')
    plt.ylabel('Mean ROC-AUC Score')
    plt.title('Puntuación ROC-AUC vs. K para KNN')
    # plt.xscale('log')
    plt.legend(['Resultado entrenamiento', 'Resultado validación'], loc='upper right')
    plt.grid()

    plt.show()
```

```
print("==========================Mejores puntuaciones promedio de ROC-AUC y mejores hiperparámetros:==============================")

# Print mean ROC-AUC scores for train and val data for all values of K
print("Puntuaciones promedio de ROC-AUC para los datos de entrenamiento para todos los valores de K: \n", mean_roc_auc_scores_train)
print("Puntuaciones promedio de ROC-AUC para los datos de validación para todos los valores de K: \n", mean_roc_auc_scores_val)

# Find the index of maximum mean ROC-AUC score for val data
best_index = np.argmax(mean_roc_auc_scores_val)
best_k = K_values[best_index]
best_mean_roc_auc = mean_roc_auc_scores_val[best_index]

print("Mejor K:", best_k)
print("Mejor puntuación promedio de ROC-AUC para los datos de validación:", best_mean_roc_auc)


print("=============================================================================================================")
print("\n")
print("=============================================================================================================")


print("============ Sintonización de múltiples hiperparámetros (GridSearchCV) + Puntuación ROC_AUC del Mejor Modelo KNN: ==========

"""
Parámetros de Regresión Logística para validación cruzada estratificada con K-fold en este caso para el modelo KNN

"""
from sklearn.model_selection import GridSearchCV

params = {'n_neighbors': [5, 7, 9], 'metric': ['euclidean', 'manhattan']}

# Clasificador KNN
knn_classifier = KNeighborsClassifier()

start_time = time.time()

# Create a GridSearchCV object with stratified Cross validation
model_GridSearch = GridSearchCV(knn_classifier,
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = skf,
                        n_jobs=-1, # 100% de la CPU
                        verbose = 1,
                        pre_dispatch = 6, # evitamos problemas de memoria
                        return_train_score=True)
# Ajustamos el objeto GridSearchCV y realizamos la sintonización de hiperparámetros
model_GridSearch.fit(X_train_pt, y_train_pt)


end_time = time.time()
print("Tiempo transcurrido: {:.2f} seconds".format(end_time - start_time))

#print the evaluation result by choosing a evaluation metric
print('Mejor puntuación ROC AUC: ', model_GridSearch.best_score_)
```

```
#print the optimum value of hyperparameters
print('Mejores hiperparámetros: ', model_GridSearch.best_params_)

cv_results = model_GridSearch.cv_results_

# Imprimir las puntuaciones medias de prueba para cada combinación de hiperparámetros
print("Puntuaciones medias de prueba:")
for mean_score, params in zip(cv_results["mean_test_score"], cv_results["params"]):
    print(params, mean_score)

# Imprimir el rango de cada combinación de hiperparámetros basado en la puntuación media de prueba
print("\nRango de cada combinación de hiperparámetros:")
for rank, params in enumerate(cv_results["params"]):
    print(rank+1, params)

# Imprimir la desviación estándar de las puntuaciones de prueba para cada combinación de hiperparámetros
print("\nDesviación estándar de las puntuaciones de prueba:")
for std_score, params in zip(cv_results["std_test_score"], cv_results["params"]):
    print(params, std_score)
```

```
995  print("==========================================SVM Classifier:==========================================")
996
997  # Valores de C
998  C_values = [0.1, 1, 2]
999
1000 # Inicialización de listas para almacenar las puntuaciones promedio de ROC-AUC y las tasas promedio de falsos positivos
1001 # para los datos de entrenamiento y validación
1002 mean_roc_auc_scores_train = []
1003 mean_roc_auc_scores_val = []
1004 mean_fpr = np.linspace(0, 1, 100)
1005
1006 # fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
1007 fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 7), facecolor='fuchsia')  # Set facecolor to black for dark background
1008 # Set background color for subplots
1009 ax1.set_facecolor('black')
1010 ax2.set_facecolor('black')
1011
1012 # Iteracion para cada valor de C
1013 for c in C_values:
1014     print("C =", c, "Kernel = rbf")
1015     cv_scores_train = []
1016     cv_scores_val = []
1017     fprs_train = []
1018     tprs_train = []
1019     fprs_val = []
1020     tprs_val = []
1021
1022     # Validación cruzada
1023     for train_index, val_index in skf.split(X_train_pt, y_train_pt):
1024         start_time = time.time()
1025         print("Train:", train_index, "val:", val_index)
1026         X_train_cv, X_val_cv = X_train_pt[train_index], X_train_pt[val_index]
1027         y_train_cv, y_val_cv = y_train_pt.iloc[train_index], y_train_pt.iloc[val_index]
1028
1029         # svm_classifier = svm.SVC(kernel='linear', C=c, probability=True)
1030         svm_classifier = svm.SVC(kernel='rbf', C=c, probability=True)
1031         svm_classifier.fit(X_train_cv, y_train_cv)
1032
1033         y_train_pred = svm_classifier.predict_proba(X_train_cv)
1034         train_score = roc_auc_score(y_true=y_train_cv, y_score=y_train_pred[:, 1])
1035         cv_scores_train.append(train_score)
1036
1037         y_val_pred = svm_classifier.predict_proba(X_val_cv)
1038         val_score = roc_auc_score(y_true=y_val_cv, y_score=y_val_pred[:, 1])
1039         cv_scores_val.append(val_score)
1040
1041         fpr_train, tpr_train, _ = roc_curve(y_true=y_train_cv, y_score=y_train_pred[:, 1])
1042         tprs_train.append(interp(mean_fpr, fpr_train, tpr_train))
1043         tprs_train[-1][0] = 0.0
1044         fprs_train.append(fpr_train)
1045
1046         fpr_val, tpr_val, _ = roc_curve(y_true=y_val_cv, y_score=y_val_pred[:, 1])
1047         tprs_val.append(interp(mean_fpr, fpr_val, tpr_val))
1048         tprs_val[-1][0] = 0.0
1049         fprs_val.append(fpr_val)
```

```
        elapsed_time = time.time() - start_time
        print("Time elapsed:", elapsed_time, "seconds")
    # Calcular las puntuaciones promedio de ROC-AUC para los datos de entrenamiento y validación
    mean_roc_auc_train = np.mean(cv_scores_train)
    mean_roc_auc_scores_train.append(mean_roc_auc_train)

    mean_roc_auc_val = np.mean(cv_scores_val)
    mean_roc_auc_scores_val.append(mean_roc_auc_val)

    print("Puntuación de ROC-AUC para los datos de entrenamiento =", mean_roc_auc_train)
    print("Puntuación de ROC-AUC para los datos de validación =", mean_roc_auc_val)
    print("-----------------------------------------------------------------------------------------------------")

    # Curvas ROC para los datos de entrenamiento
    ax1.plot(mean_fpr, np.mean(tprs_train, axis=0), linestyle='-', lw=2, label='C = {}, Train ROC-AUC = {:.4f}'.format(c, mean_roc_auc_train))

    # Curvas ROC para los datos de validación
    ax2.plot(mean_fpr, np.mean(tprs_val, axis=0), linestyle='-', lw=2, label='C = {}, val ROC-AUC = {:.4f}'.format(c, mean_roc_auc_val))

print("=======================================El SVM se ha ejecutado correctamente:=======================================")

print("-----------------------------------------------------------------------------------------------------")
print("\n")
print("=====================================Trazando la curva ROC-AUC:=====================================")
print("\n")

# gráficas para los datos de entrenamiento
ax1.plot([0, 1], [0, 1], linestyle='--', lw=2, color='gray', label='Random', alpha=0.8)
ax1.set_xlim([0.0, 1.0])
ax1.set_ylim([0.0, 1.0])
ax1.set_xlabel('Tasa de falsos positivos', fontsize=12)
ax1.set_ylabel('Tasa de verdaderos positivos', fontsize=12)
ax1.set_title('Curva ROC - Datos de entrenamiento', fontsize=14)
ax1.legend(loc="lower right")
ax1.grid(True)

# gráficas para los datos de validación
ax2.plot([0, 1], [0, 1], linestyle='--', lw=2, color='gray', label='Random', alpha=0.8)
ax2.set_xlim([0.0, 1.0])
ax2.set_ylim([0.0, 1.0])
ax2.set_xlabel('Tasa de falsos positivos', fontsize=12)
ax2.set_ylabel('Tasa de verdaderos positivos', fontsize=12)
ax2.set_title('Curva ROC - Datos de validación', fontsize=14)
ax2.legend(loc="lower right")
ax2.grid(True)

# Show the plot
plt.show()
```

```
# Set plot style and figure size
with plt.style.context('dark_background'):
    plt.figure(figsize=(8, 12), facecolor='m')

    # Plot ROC-AUC scores vs C
    plt.plot(C_values, mean_roc_auc_scores_train, 'yo-')
    plt.plot(C_values, mean_roc_auc_scores_val, 'co-')
    plt.xlabel('C')
    plt.ylabel('Puntuación ROC-AUC')
    plt.title('Puntuación ROC-AUC vs. C para SVM')
    # plt.xscale('log')
    plt.legend(['resultados entrenamiento', 'resultados validación'], loc='upper right')
    plt.grid()

    plt.show()

print("\n")
print("=============================================================================================================")
print("\n")

print("============================Mejores puntuaciones promedio de ROC-AUC y mejores hiperparámetros:============================")

# Imprimir las puntuaciones promedio de ROC-AUC para los datos de entrenamiento y validación para todos los valores de C
print("Puntuaciones promedio de ROC-AUC para los datos de entrenamiento para todos los valores de C: \n", mean_roc_auc_scores_train)
print("Puntuaciones promedio de ROC-AUC para los datos de validación para todos los valores de C: \n", mean_roc_auc_scores_val)

# Encontrar el índice del puntaje promedio de ROC-AUC máximo para los datos de validación
best_index = np.argmax(mean_roc_auc_scores_val)
best_c = C_values[best_index]
best_mean_roc_auc = mean_roc_auc_scores_val[best_index]

print("Mejor C:", best_c)
print("Mejor puntuación promedio de ROC-AUC para los datos de validación:", best_mean_roc_auc)
```

```
print("============ Sintonización de múltiples hiperparámetros (GridSearchCV) + Puntuación ROC_AUC del Mejor Modelo SVM: =

"""
Parámetros de Regresión Logística para validación cruzada estratificada con K-fold en este caso para el modelo SVM

"""
from sklearn import svm

params = {"C": [0.01, 0.1], 'kernel': ['rbf'], 'gamma': ['auto']}

# Definimos el clasificador SVM
svm_classifier = svm.SVC()

start_time = time.time()

# Creación de un objeto GridSearchCV con validación cruzada estratificada
model_GridSearch = GridSearchCV(svm_classifier,
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = skf,
                        n_jobs=-1, # uso del 100% de la CPU
                        verbose = 1,
                        pre_dispatch = 6, # evitamos problemas de memoria
                        return_train_score=True)

# Ajustamos el objeto GridSearchCV y realizar la sintonización de hiperparámetros
model_GridSearch.fit(X_train_pt, y_train_pt)

end_time = time.time()
print("Tiempo transcurrido: {:.2f} segundos".format(end_time - start_time))

# Print the evaluation result by choosing a evaluation metric
print('Mejor puntuación de ROC AUC: ', model_GridSearch.best_score_)

# Print the optimum value of hyperparameters
print('Mejores hiperparámetros: ', model_GridSearch.best_params_)


cv_results = model_GridSearch.cv_results_

# Imprimir las puntuaciones medias de prueba para cada combinación de hiperparámetros
print("Puntuaciones medias de prueba:")
for mean_score, params in zip(cv_results["mean_test_score"], cv_results["params"]):
    print(params, mean_score)

# Imprimir el rango de cada combinación de hiperparámetros basado en la puntuación media de prueba
print("\nRango de cada combinación de hiperparámetros:")
for rank, params in enumerate(cv_results["params"]):
    print(rank+1, params)

# Imprimir la desviación estándar de las puntuaciones de prueba para cada combinación de hiperparámetros
print("\nDesviación estándar de las puntuaciones de prueba:")
for std_score, params in zip(cv_results["std_test_score"], cv_results["params"]):
    print(params, std_score)
```

```
print("========================= Árbol de decisión - Decision Tree:============================================")


# Valores de max_depth
max_depth_values = [1, 2, 3, 4, 5]

# Inicializamos las  listas para almacenar los puntajes promedio de ROC-AUC y las tasas promedio de
# falsos positivos para los datos de validación
mean_roc_auc_scores_val = []
mean_precision_scores_val = []
mean_recall_scores_val = []
mean_f1_scores_val = []
mean_fpr = np.linspace(0, 1, 100)

fig, ax2= plt.subplots(figsize=(14, 7), facecolor='fuchsia')
ax2.set_facecolor('black')

# Recorremos cada valor de max_depth
for max_depth in max_depth_values:
    print("Max Depth =", max_depth)
    cv_scores_val = []
    precision_val = []
    recall_val = []
    f1_val = []
    fprs_val = []
    tprs_val = []


    # Validación cruzada
    for train_index, val_index in skf.split(X_train_pt, y_train_pt):
        start_time = time.time()
        print("Train:", train_index, "val:", val_index)
        X_train_cv, X_val_cv = X_train_pt[train_index], X_train_pt[val_index]
        y_train_cv, y_val_cv = y_train_pt.iloc[train_index], y_train_pt.iloc[val_index]

        # Crear un clasificador de árbol de decisión con el valor de max_depth especificado
        dt_classifier = DecisionTreeClassifier(max_depth=max_depth)
        dt_classifier.fit(X_train_cv, y_train_cv)

        y_val_pred = dt_classifier.predict_proba(X_val_cv)
        val_score = roc_auc_score(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        cv_scores_val.append(val_score)

        y_val_pred_binary = (y_val_pred[:, 1] >= 0.5).astype(int)
        precision_val.append(precision_score(y_val_cv, y_val_pred_binary))
        recall_val.append(recall_score(y_val_cv, y_val_pred_binary))
        f1_val.append(f1_score(y_val_cv, y_val_pred_binary))

        fpr_val, tpr_val, _ = roc_curve(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        tprs_val.append(interp(mean_fpr, fpr_val, tpr_val))
        tprs_val[-1][0] = 0.0
        fprs_val.append(fpr_val)
        elapsed_time = time.time() - start_time
        print("Tiempo transcurrido:", elapsed_time, "segundos")
```

```
        # Calculamos las puntuaciones promedio de ROC-AUC para los datos de validación
        mean_roc_auc_val = np.mean(cv_scores_val)
        mean_roc_auc_scores_val.append(mean_roc_auc_val)

        mean_precision_val = np.mean(precision_val)
        mean_precision_scores_val.append(mean_precision_val)

        mean_recall_val = np.mean(recall_val)
        mean_recall_scores_val .append(mean_recall_val)

        mean_f1_val = np.mean(f1_val)
        mean_f1_scores_val.append(mean_f1_val)

        print("Puntuación de ROC-AUC para los datos de validación =", mean_roc_auc_val)
        print("Puntuación promedio de precisión para los datos de validación:", mean_precision_val)
        print("Puntuación promedio de recall para los datos de validación:", mean_recall_val)
        print("Puntuación promedio de f1 para los datos de validación:", mean_f1_val)
        print("-----------------------------------------------------------------------------------------------------------")

        # Curva ROC para el valor actual de C
        ax2.plot(mean_fpr, np.mean(tprs_val, axis=0), linestyle='-', lw=2, label='Max Depth = ' + str(max_depth) + ', val ROC-AUC = ' + str(round(mean_roc_auc_val, 4)))


    print("==================================El Árbol de Decisión se ha ejecutado correctamente:====================================")

    print("-----------------------------------------------------------------------------------------------------------")
    print("\n")
    print("==================================== Curva ROC-AUC:=========================================")
    print("\n")

    # Curva ROC para el clasificador aleatorio
    ax2.plot([0, 1], [0, 1], linestyle='--', color='black', label='Random', alpha=0.5)

    ax2.set_xlabel('Tasa de Falsos Positivos')
    ax2.set_ylabel('Tasa de Verdaderos Positivos')
    ax2.set_title('Curva ROC - Datos de validación')
    ax2.legend(loc="lower right")
    ax2.grid(True)
    plt.show()

    print("=============================================================================================================")

    with plt.style.context('dark_background'):
        plt.figure(figsize=(8, 12), facecolor='m')

        # Graficar los puntajes de ROC-AUC en función de C
        plt.plot(max_depth_values, mean_roc_auc_scores_val, 'co-')
        plt.xlabel('Max-depth')
        plt.ylabel('Puntaje de ROC-AUC')
        plt.title('Puntaje de ROC-AUC vs. Max-depth')
        # plt.xscale('log')
        plt.legend(['resultado validación'], loc='upper right')
        plt.grid()

        plt.show()
```

```
print("==========================Mejores puntajes promedio de ROC-AUC y mejores hiperparámetros:============================")


print("Puntuaciones promedio de ROC-AUC para los datos de validación para todos los valores de Max Depth: \n", mean_roc_auc_scores_val)

# Encontrar el índice de la puntuación promedio de ROC-AUC máximo para los datos de validación
best_index = np.argmax(mean_roc_auc_scores_val)
best_max_depth = max_depth_values[best_index]
best_mean_roc_auc = mean_roc_auc_scores_val[best_index]

print("Mejor Max Depth:", best_max_depth)
print("Mejor puntuación promedio de ROC-AUC para los datos de validación:", best_mean_roc_auc)
print("Puntuación promedio de precisión para los datos de validación con el mejor valor de C", mean_precision_scores_val[best_index])
print("Puntuación promedio de recall para los datos de validación con el mejor valor de C", mean_recall_scores_val[best_index])
print("Puntuación promedio de f1 para los datos de validación con el mejor valor de C", mean_f1_scores_val[best_index])


print("=============================================================================================================")
print("\n")
print("=============================================================================================================")

print("============ Sintonización de múltiples hiperparámetros (GridSearchCV) + Puntuación ROC_AUC del Mejor Modelo Decision Tree: =====

"""
Parámetros de Regresión Logística para validación cruzada estratificada con K-fold en este caso para el modelo Decision Tree

"""

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
import time

# Define the Decision Tree classifier
dt_classifier = DecisionTreeClassifier(random_state=42)

# Define the parameters to be tuned
params = {'max_depth': [2, 3, 4],
          'min_samples_split': [2, 5, 10],
          'min_samples_leaf': [1, 2, 4],
          'criterion': ['gini', 'entropy']}

# Create a GridSearchCV object with stratified cross-validation
start_time = time.time()
model_GridSearch = GridSearchCV(dt_classifier,
                        param_grid = params,
                        scoring= 'roc_auc',
                        cv = skf,
                        n_jobs=-1, # using 100% of CPU
                        verbose = 1,
                        return_train_score=True)

# Fit the GridSearchCV object and perform hyperparameter tuning
model_GridSearch.fit(X_train_pt, y_train_pt)

end_time = time.time()
```

```
print("================================================Random Forest:=============================================")

# Define your values of n_estimators
n_values = [10, 50, 100, 200, 400]

# Initialize lists to store mean ROC-AUC scores and mean false positive rates for train and val data
mean_roc_auc_scores_train = []
mean_roc_auc_scores_val = []
mean_fpr = np.linspace(0, 1, 100)

# fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 7), facecolor='fuchsia')  # Set facecolor to black for dark backgroun
# Set background color for subplots
ax1.set_facecolor('black')
ax2.set_facecolor('black')


# Loop through each value of n_estimators
for n in n_values:
    print("n_estimators =", n)
    cv_scores_train = []
    cv_scores_val = []
    fprs_train = []
    tprs_train = []
    fprs_val = []
    tprs_val = []

    # Perform cross-validation
    for train_index, val_index in skf.split(X_train_pt, y_train_pt):
        start_time = time.time()
        print("Train:", train_index, "val:", val_index)
        X_train_cv, X_val_cv = X_train_pt[train_index], X_train_pt[val_index]
        y_train_cv, y_val_cv = y_train_pt.iloc[train_index], y_train_pt.iloc[val_index]

        rf_classifier = RandomForestClassifier(n_estimators=n)
        rf_classifier.fit(X_train_cv, y_train_cv)

        y_train_pred = rf_classifier.predict_proba(X_train_cv)
        train_score = roc_auc_score(y_true=y_train_cv, y_score=y_train_pred[:, 1])
        cv_scores_train.append(train_score)

        y_val_pred = rf_classifier.predict_proba(X_val_cv)
        val_score = roc_auc_score(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        cv_scores_val.append(val_score)

        fpr_train, tpr_train, _ = roc_curve(y_true=y_train_cv, y_score=y_train_pred[:, 1])
        tprs_train.append(interp(mean_fpr, fpr_train, tpr_train))
        tprs_train[-1][0] = 0.0
        fprs_train.append(fpr_train)

        fpr_val, tpr_val, _ = roc_curve(y_true=y_val_cv, y_score=y_val_pred[:, 1])
        tprs_val.append(interp(mean_fpr, fpr_val, tpr_val))
        tprs_val[-1][0] = 0.0
        fprs_val.append(fpr_val)
        elapsed_time = time.time() - start_time
```

```
        print("Time elapsed:", elapsed_time, "seconds")

    # Calculate mean ROC-AUC scores for train and val data
    mean_roc_auc_train = np.mean(cv_scores_train)
    mean_roc_auc_scores_train.append(mean_roc_auc_train)

    mean_roc_auc_val = np.mean(cv_scores_val)
    mean_roc_auc_scores_val.append(mean_roc_auc_val)

    print("ROC-AUC Score for train data =", mean_roc_auc_train)
    print("ROC-AUC Score for val data =", mean_roc_auc_val)
    print("-----------------------------------------------------------------------------------------------------------")

    # Plot ROC curve for the current value of C
    ax1.plot(mean_fpr, np.mean(tprs_train, axis=0), linestyle='-', lw=2, label='n_estimators = ' + str(n) + ', Train ROC-AUC = ' + str(round(mean_roc_auc_train, 4)))
    ax2.plot(mean_fpr, np.mean(tprs_val, axis=0), linestyle='-', lw=2, label='n_estimators = ' + str(n) + ', val ROC-AUC = ' + str(round(mean_roc_auc_val, 4)))


print("================================Random Forest se ha ejectuado correctamente:=====================================")
```

```
print("====================================Trazando la curva ROC AUC:====================================")
print("\n")

# Plot ROC curve for random classifier
ax1.plot([0, 1], [0, 1], linestyle='--', color='black', label='Random', alpha=0.5)
ax2.plot([0, 1], [0, 1], linestyle='--', color='black', label='Random', alpha=0.5)

# Set labels and title for ROC curves
ax1.set_xlabel('False Positive Rate')
ax1.set_ylabel('True Positive Rate')
ax1.set_title('ROC Curve - Train Data')
ax1.legend(loc="lower right")
ax1.grid(True)  # Add grid to the plot

ax2.set_xlabel('False Positive Rate')
ax2.set_ylabel('True Positive Rate')
ax2.set_title('ROC Curve - val Data')
ax2.legend(loc="lower right")
ax2.grid(True)  # Add grid to the plot

# Show the plot
plt.show()

print("============================================================================================================")

# Set plot style and figure size
with plt.style.context('dark_background'):
    plt.figure(figsize=(8, 12), facecolor='m')

    # Plot ROC-AUC scores vs C
    plt.plot(n_values, mean_roc_auc_scores_train, 'yo-')
    plt.plot(n_values, mean_roc_auc_scores_val, 'co-')
    plt.xlabel('n Estimators')
    plt.ylabel('ROC-AUC Score')
    plt.title('ROC-AUC Score vs. n_estimators for Random Forest')
    # plt.xscale('log')
    plt.legend(['train result', 'val result'], loc='upper right')
    plt.grid()

    plt.show()
```

```
print("==========================Best Mean ROC-AUC scores and Best hyperparameters:==============================")

# Print mean ROC-AUC scores for train and val data for all values of n Estimators
print("Mean ROC-AUC scores for train data for all values of n Estimators: \n", mean_roc_auc_scores_train)
print("Mean ROC-AUC scores for val data for all values of n Estimators: \n", mean_roc_auc_scores_val)

# Find the index of maximum mean ROC-AUC score for val data
best_index = np.argmax(mean_roc_auc_scores_val)
best_n_estimators = n_values[best_index]
best_mean_roc_auc = mean_roc_auc_scores_val[best_index]

print("Best n Estimators:", best_n_estimators)
print("Best Mean ROC-AUC score for val data:", best_mean_roc_auc)



print("=================================================================================================================")
print("\n")
print("=================================================================================================================")

print("============ Sintonización de múltiples hiperparámetros (GridSearchCV) + Puntuación ROC_AUC del Mejor Modelo Random Forest:

"""
Parámetros de Regresión Logística para validación cruzada estratificada con K-fold en este caso para el modelo Random Forest

"""

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import time

# Define the Random Forest classifier
rf_classifier = RandomForestClassifier()

# Define the hyperparameters grid for tuning
params = {
    "n_estimators": [500],
    "min_samples_split": [5, 7],
    }

start_time = time.time()

# Create a GridSearchCV object with stratified Cross validation
model_GridSearch = GridSearchCV(rf_classifier,
                        param_grid=params,
                        scoring='roc_auc',
                        cv=skf,
                        n_jobs=-1,
                        verbose=1,
                        # pre_dispatch=6,
                        return_train_score=True)

# Fit the GridSearchCV object and perform hyperparameter tuning
model_GridSearch.fit(X_train_pt, y_train_pt)

end_time = time.time()
```

```
1633    print("=========================================XGBoost Classifier:=========================================")
1634
1635
1636    # Define your values of C
1637    learning_rate_values = [0.001, 0.01, 0.1, 0.5, 1]
1638
1639    # Initialize lists to store mean ROC-AUC scores and mean false positive rates for val data
1640    mean_roc_auc_scores_val = []
1641    mean_precision_scores_val = []
1642    mean_recall_scores_val = []
1643    mean_f1_scores_val = []
1644    mean_fpr = np.linspace(0, 1, 100)
1645
1646    fig, ax2 = plt.subplots(figsize=(14, 7), facecolor='fuchsia')
1647    ax2.set_facecolor('black')
1648
1649    # Loop through each value of C
1650    for learning_rate in learning_rate_values:
1651        print("Learning Rate =", learning_rate)
1652        cv_scores_val = []
1653        precision_val = []
1654        recall_val = []
1655        f1_val = []
1656        fprs_val = []
1657        tprs_val = []
1658
1659        # Perform cross-validation
1660        for train_index, val_index in skf.split(X_train_pt, y_train_pt):
1661            start_time = time.time()
1662            print("Train:", train_index, "val:", val_index)
1663            X_train_cv, X_val_cv = X_train_pt[train_index], X_train_pt[val_index]
1664            y_train_cv, y_val_cv = y_train_pt.iloc[train_index], y_train_pt.iloc[val_index]
1665
1666            xgb_classifier = xgb.XGBClassifier(learning_rate = learning_rate, n_estimators=100, max_depth=3, gamma=0, subsample=0.8,
1667                                               colsample_bytree=0.8, objective='binary:logistic', reg_alpha=0.005,
1668                                               reg_lambda=1, random_state=42)
1669            xgb_classifier.fit(X_train_cv, y_train_cv)
1670
1671            y_val_pred = xgb_classifier.predict_proba(X_val_cv)
1672            val_score = roc_auc_score(y_true=y_val_cv, y_score=y_val_pred[:, 1])
1673            cv_scores_val.append(val_score)
1674
1675            y_val_pred_binary = (y_val_pred[:, 1] >= 0.5).astype(int)
1676
1677            precision_val.append(precision_score(y_val_cv, y_val_pred_binary))
1678            recall_val.append(recall_score(y_val_cv, y_val_pred_binary))
1679            f1_val.append(f1_score(y_val_cv, y_val_pred_binary))
1680
1681            fpr_val, tpr_val, _ = roc_curve(y_true=y_val_cv, y_score=y_val_pred[:, 1])
1682            tprs_val.append(interp(mean_fpr, fpr_val, tpr_val))
1683            tprs_val[-1][0] = 0.0
1684            fprs_val.append(fpr_val)
1685            elapsed_time = time.time() - start_time
1686            print("Time elapsed:", elapsed_time, "seconds")
```

```
1688        # Calculate mean ROC-AUC scores for val data
1689
1690        mean_roc_auc_val = np.mean(cv_scores_val)
1691        mean_roc_auc_scores_val.append(mean_roc_auc_val)
1692
1693        mean_precision_val = np.mean(precision_val)
1694        mean_precision_scores_val.append(mean_precision_val)
1695
1696        mean_recall_val = np.mean(recall_val)
1697        mean_recall_scores_val .append(mean_recall_val)
1698
1699        mean_f1_val = np.mean(f1_val)
1700        mean_f1_scores_val.append(mean_f1_val)
1701
1702        print("ROC-AUC Score for val data =", mean_roc_auc_val)
1703        print("Mean precision val score:", mean_precision_val)
1704        print("Mean recall val score:", mean_recall_val)
1705        print("Mean f1 val score:", mean_f1_val)
1706        print("-----------------------------------------------------------------------------------------------------")
1707
1708        # Plot ROC curve for the current value of C
1709        ax2.plot(mean_fpr, np.mean(tprs_val, axis=0), label='Learning Rate = ' + str(learning_rate) + ', val ROC-AUC = ' + str(round(mean_roc_auc_val, 4)))
1710
1711
1712    print("=====================================XGBoost se ha ejecutado correctamente:=====================================")
```

```
1716    print("=====================================Trazando la curva ROC AUC:=========================================")
1717    print("\n")
1718
1719    # Plot ROC curve for random classifier
1720    ax2.plot([0, 1], [0, 1], linestyle='--', color='black', label='Random', alpha=0.5)
1721
1722    # Set labels and title for ROC curves
1723    ax2.set_xlabel('False Positive Rate')
1724    ax2.set_ylabel('True Positive Rate')
1725    ax2.set_title('ROC Curve - val Data')
1726    ax2.legend(loc="lower right")
1727    ax2.grid(True)  # Add grid to the plot
1728
1729    # Show the plot
1730    plt.show()
1731
1732    print("=============================================================================================================")
1733
1734    # Set plot style and figure size
1735    with plt.style.context('dark_background'):
1736        plt.figure(figsize=(8, 12), facecolor='m')
1737
1738        # Plot ROC-AUC scores vs C
1739        plt.plot(learning_rate_values, mean_roc_auc_scores_val, 'co-')
1740        plt.xlabel('learning_rate')
1741        plt.ylabel('ROC-AUC Score')
1742        plt.title('ROC-AUC Score vs. learning_rate_values for XGBoost')
1743        plt.legend(['val result'], loc='upper right')
1744        plt.grid()
1745
1746        plt.show()
```

```
print("==========================Best Mean ROC-AUC scores and Best hyperparameters:==============================")

# Print mean ROC-AUC scores for val data for all values of learning_rate_values
print("Mean ROC-AUC scores for val data for all values of Learning Rate: \n", mean_roc_auc_scores_val)

# Find the index of maximum mean ROC-AUC score for val data
best_index = np.argmax(mean_roc_auc_scores_val)
best_learning_rate = learning_rate_values[best_index]
best_mean_roc_auc = mean_roc_auc_scores_val[best_index]

print("Best Learning Rate :", best_learning_rate)
print("Best Mean ROC-AUC score for val data:", best_mean_roc_auc)
print("Mean precision val score for best C", mean_precision_scores_val[best_index])
print("Mean recall val score for best C", mean_recall_scores_val[best_index])
print("Mean f1 val score for best C", mean_f1_scores_val[best_index])


print("=============================================================================================================")
print("\n")
print("=============================================================================================================")

print("============ Sintonización de múltiples hiperparámetros (GridSearchCV) + Puntuación ROC_AUC del Mejor Modelo XGBoost:

"""
Parámetros de Regresión Logística para validación cruzada estratificada con K-fold en este caso para el modelo XGBoost

"""

import xgboost as xgb
from sklearn.model_selection import GridSearchCV, StratifiedKFold

params = {
    'learning_rate': [0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.5, 0.7, 0.9],
}

# Define the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(objective='binary:logistic', eval_metric='auc') # As the number of classes are 2

start_time = time.time()

# Create a GridSearchCV object with stratified Cross validation
model_GridSearch = GridSearchCV(xgb_classifier,
                        param_grid=params,
                        scoring='roc_auc',
                        cv=skf,
                        n_jobs=-1,
                        verbose=1,
                        return_train_score=True)

# Fit the GridSearchCV object and perform hyperparameter tuning
model_GridSearch.fit(X_train_pt, y_train_pt)

end_time = time.time()
```

Una vez se han desarrollado los modelos, se lleva a cabo la validación cruzada:

```
"""
Observación final sobre datos desbalanceados

A. VALIDACIÓN CRUZADA.

A raiz de los resultados en las líneas de código anteriores podremos evaluar sobre el set de prueba

La evaluación la realizaremos sobre el modelo que mejor resultado presente y para ello:
    - Aplicaremos el mejor hiperparámetro en el modelo
    - Predeciremos en la base de datos de prueba

"""

X_test_saved.head()
y_test_saved.head()
# As PCA is already performed on the dataset from V1 to V28 features, we are scaling only Amount field
scaler = RobustScaler()

# Transforming the test data
X_test_saved["Amount"]] = scaler.fit_transform(X_test_saved[["Amount"]])
X_test_saved.head()
```

```
from sklearn.metrics import f1_score, precision_score, recall_score

print("=============================================================================================")
print("\n")

# inicializar el modelo con los hiperparámetros óptimos
start_time = time.time()
clf = linear_model.LogisticRegression(penalty='l2', C=0.01)
clf.fit(X_train_pt, y_train_pt)

# predecir en el conjunto de prueba para obtener la probabilidad
y_pred_proba = clf.predict_proba(X_test_saved)
# calcular la puntuación ROC-AUC
roc_auc = roc_auc_score(y_true=y_test_saved, y_score=y_pred_proba[:, 1])
print("Puntuación LogisticRegression ROC-AUC en el conjunto de prueba =", roc_auc)
# predicción en el conjunto de prueba para obtener las etiquetas de clase.
y_pred = clf.predict(X_test_saved)
# calculo de F1-score, precision, and recall
f1 = f1_score(y_test_saved, y_pred)
preci_sion = precision_score(y_test_saved, y_pred)
re_call = recall_score(y_test_saved, y_pred)

print("LogisticRegression F1-Score en el conjunto de prueba =", f1)
print("LogisticRegression Precision en el conjunto de prueba =", preci_sion)
print("LogisticRegression Recall en el conjunto de prueba =", re_call)
end_time = time.time()
print("Tiempo empleado: {:.2f} segundos".format(end_time - start_time))


print("=============================================================================================")

# Inicializar el modelo con los hiperparámetros óptimos.
start_time = time.time()
clf = KNeighborsClassifier(n_neighbors=9, metric='manhattan')
clf.fit(X_train_pt, y_train_pt)
# predecir en el conjunto de prueba para obtener la probabilidad.
y_pred_proba = clf.predict_proba(X_test_saved)
# calcular la puntuación ROC-AUC
roc_auc = roc_auc_score(y_true=y_test_saved, y_score=y_pred_proba[:, 1])
print("Puntuación KNeighbors Classifier ROC-AUC Score en el conjunto de prueba =", roc_auc)
# predecir en el conjunto de prueba para obtener las etiquetas de clase.
y_pred = clf.predict(X_test_saved)
# calcular F1-score, precision, and recall
f1 = f1_score(y_test_saved, y_pred)
preci_sion = precision_score(y_test_saved, y_pred)
re_call = recall_score(y_test_saved, y_pred)

print("KNeighbors Classifier F1-Score en el conjunto de prueba =", f1)
print("KNeighbors Classifier Precision en el conjunto de prueba =", preci_sion)
print("KNeighbors Classifier Recall en el conjuuto de prueba =", re_call)
end_time = time.time()
print("Tiempo empleado: {:.2f} segundos".format(end_time - start_time))

print("=============================================================================================")
```

```
1925    print("=============================================================================================================")
1926
1927    # Inicializar el modelo con los hiperparámetros óptimos.
1928    start_time = time.time()
1929    clf = DecisionTreeClassifier(criterion='entropy', max_depth=3, min_samples_leaf=1, min_samples_split=2)
1930    clf.fit(X_train_pt, y_train_pt)
1931    # predecir en el conjunto de prueba para obtener la probabilidad.
1932    y_pred_proba = clf.predict_proba(X_test_saved)
1933    # calcular la puntuación ROC-AUC
1934    roc_auc = roc_auc_score(y_true=y_test_saved, y_score=y_pred_proba[:, 1])
1935    print("Decision Tree Classifier ROC-AUC Score en el conjunto de prueba =", roc_auc)
1936    # predecir en el conjunto de prueba para obtener las etiquetas de clase.
1937    y_pred = clf.predict(X_test_saved)
1938    # calcular F1-score, precision, and recall
1939    f1 = f1_score(y_test_saved, y_pred)
1940    preci_sion = precision_score(y_test_saved, y_pred)
1941    re_call = recall_score(y_test_saved, y_pred)
1942
1943    print("Decision Tree Classifier F1-Score en el conjunto de prueba =", f1)
1944    print("Decision Tree Classifier Precision en el conjunto de prueba =", preci_sion)
1945    print("Decision Tree Classifier Recall en el conjunto de prueba =", re_call)
1946    end_time = time.time()
1947    print("Tiempo empleado: {:.2f} segundos".format(end_time - start_time))
1948
1949    print("=============================================================================================================")
1950    # Inicializar el modelo con los hiperparámetros óptimos.
1951    start_time = time.time()
1952    clf = XGBClassifier(learning_rate=0.1, max_depth=3, subsample=0.5, objective='binary:logistic', eval_metric='auc')
1953    clf.fit(X_train_pt, y_train_pt)
1954    # predecir en el conjunto de prueba para obtener la probabilidad.
1955    y_pred_proba = clf.predict_proba(X_test_saved)
1956    # calcular la puntuación ROC-AUC
1957    roc_auc = roc_auc_score(y_true=y_test_saved, y_score=y_pred_proba[:, 1])
1958    print("XGBOOST Classifier ROC-AUC Score en el conjunto de prueba =", roc_auc)
1959    # predecir en el conjunto de prueba para obtener las etiquetas de clase.
1960    y_pred = clf.predict(X_test_saved)
1961    # calcular F1-score, precision, and recall
1962    f1 = f1_score(y_test_saved, y_pred)
1963    preci_sion = precision_score(y_test_saved, y_pred)
1964    re_call = recall_score(y_test_saved, y_pred)
1965
1966    print("XGBOOST Classifier F1-Score en el conjunto de prueba =", f1)
1967    print("XGBOOST Classifier Precision en el conjunto de prueba =", preci_sion)
1968    print("XGBOOST Classifier Recall en el conjunto de prueba =", re_call)
1969    end_time = time.time()
1970    print("Time taken: {:.2f} seconds".format(end_time - start_time))
```

```
start_time = time.time()
clf = svm.SVC(probability=True, C=0.01, gamma='auto', kernel='rbf')
clf.fit(X_train_pt, y_train_pt)
y_pred_proba = clf.predict_proba(X_test_saved)
y_pred = clf.predict(X_test_saved)
roc_auc = roc_auc_score(y_true=y_test_saved, y_score=y_pred_proba[:,1])
f1 = f1_score(y_true=y_test_saved, y_pred=y_pred)
preci_sion = precision_score(y_true=y_test_saved, y_pred=y_pred)
re_call = recall_score(y_true=y_test_saved, y_pred=y_pred)
print("Puntuación SVM Classifier ROC-AUC en el conjunto de prueba =", roc_auc)
print("F1 Score en el conjunto de prueba =", f1)
print("Precision en el conjunto de prueba =", preci_sion)
print("Recall en el conjunto de prueba =", re_call)
end_time = time.time()
print("Tiempo empleado: {:.2f} segundos".format(end_time - start_time))

print("========================================================================================================================")
# Inicializar el modelo con los hiperparámetros óptimos.
start_time = time.time()
clf = RandomForestClassifier(min_samples_split=5, n_estimators=500)
clf.fit(X_train_pt, y_train_pt)
# predecir en el conjunto de prueba para obtener la probabilidad.
y_pred_proba = clf.predict_proba(X_test_saved)
# calcular la puntuación ROC-AUC
roc_auc = roc_auc_score(y_true=y_test_saved, y_score=y_pred_proba[:, 1])
print("Random Forest Classifier ROC-AUC Score en el conjunto de prueba =", roc_auc)
# predecir en el conjunto de prueba para obtener las etiquetas de clase.
y_pred = clf.predict(X_test_saved)
# calcular F1-score, precision, and recall
f1 = f1_score(y_test_saved, y_pred)
preci_sion = precision_score(y_test_saved, y_pred)
re_call = recall_score(y_test_saved, y_pred)

print("Random Forest Classifier F1-Score en el conjunto de prueba =", f1)
print("Random Forest Classifier Precision en el conjunto de prueba =", preci_sion)
print("Random Forest Classifier Recall en el conjunto de prueba =", re_call)
end_time = time.time()
print("Tiempo empleado: {:.2f} segundos".format(end_time - start_time))
```

```
"""
B. TEST SET
Obtendremos a partir de las líneas de código anteriores la tabla de resultados para cada uno de los modelos desarrollados
"""


print("=================================================================================================================")
print("\n")
print("=================================================================================================================")

"""

Obtendremos además, las características más importantes para entender la base de datos

"""
var_imp = []
for i in clf.feature_importances_:
    var_imp.append(i)
print('Top var =', var_imp.index(np.sort(clf.feature_importances_)[-1])+1)
print('2a Top var =', var_imp.index(np.sort(clf.feature_importances_)[-2])+1)
print('3a Top var =', var_imp.index(np.sort(clf.feature_importances_)[-3])+1)

# La variable en el índice 16 y la variable en el índice 13 parecen ser las dos variables principales.
top_var_index = var_imp.index(np.sort(clf.feature_importances_)[-1])
second_top_var_index = var_imp.index(np.sort(clf.feature_importances_)[-2])

X_train_1 = X_train.to_numpy()[np.where(y_train==1.0)]
X_train_0 = X_train.to_numpy()[np.where(y_train==0.0)]

np.random.shuffle(X_train_0)

import matplotlib.pyplot as plt
# %matplotlib inline
plt.rcParams['figure.figsize'] = [20, 20]

plt.scatter(X_train_1[:, top_var_index], X_train_1[:, second_top_var_index], label='Actual Class-1 Examples')
plt.scatter(X_train_0[:X_train_1.shape[0], top_var_index], X_train_0[:X_train_1.shape[0], second_top_var_index],
            label='Actual Class-0 Examples')
plt.legend()
```

A partir de aquí la diferencia ha sido con respecto a la definición del RandomOverSampler:

```
2079  print("============ I. Random Oversampling - Sobre-muestreo aleatorio: ======================")
2080
2081  from imblearn.over_sampling import RandomOverSampler
2082
2083  # Definición de RandomOverSampler
2084  ros = RandomOverSampler(sampling_strategy='minority', random_state=42)
2085
2086  # Re-muestrear los datos de entrenamiento utilizando RandomOverSampler.
2087  X_ros_train_pt, y_ros_train_pt = ros.fit_resample(X_train_pt, y_train_pt)
2088
```

```
3585  # Definición de SMOTE
3586  smote = over_sampling.SMOTE(random_state=0)
3587
3588  # Resample los datos de entrenamiento utilizando SMOTE.
3589  X_smote_train_pt, y_smote_train_pt = smote.fit_resample(X_train_pt, y_train_pt)
3590
3591  import warnings
```

```
5171    # Definición de ADASYN
5172    ada = over_sampling.ADASYN(random_state=0)
5173
5174    # Remuestreo datos de entrenamiento utilizando ADASYN.
5175    X_adasyn_train_pt, y_adasyn_train_pt = ada.fit_resample(X_train_pt, y_train_pt)
5176
```

El resto de codificación es la misma que la mostrada hasta ahora.