GRADO EN FÍSICA

Detección de Fraudes con Algoritmos de Machine Learning

Presentado por:

INÉS TORRES RODRÍGUEZ

Dirigido por:

HECTOR GISBERT MULLOR

CURSO ACADÉMICO 2024-2025

Resumen

El presente Trabajo de Fin de Grado aborda la detección de fraudes financieros mediante la aplicación de algoritmos de machine learning. A partir de un conjunto de datos con más de seis millones de transacciones bancarias, se ha desarrollado un estudio comparativo entre dos modelos supervisados: *random forest* y redes neuronales artificiales.

Se ha llevado a cabo un análisis exploratorio y un cuidadoso preprocesamiento de los datos, incluyendo la creación de nuevas variables relevantes y técnicas de normalización. Posteriormente, ambos modelos fueron entrenados, evaluados y comparados según distintas métricas de rendimiento, como la precisión, el recall, el F1-score y la matriz de confusión.

Los resultados obtenidos evidencian que el modelo basado en *random forest* ofrece un rendimiento más sólido y eficiente, con tiempos de entrenamiento reducidos y una capacidad predictiva superior en la detección de transacciones fraudulentas. Aunque las redes neuronales mostraron ciertas limitaciones en este estudio, especialmente en la identificación de fraudes, se reconoce su potencial en contextos más adecuados desde el punto de vista computacional.

Este trabajo pone de manifiesto la utilidad del aprendizaje automático como herramienta para combatir el fraude en tiempo real, aportando soluciones escalables, precisas y adaptables a las necesidades del sector financiero actual.

Abstract

This Final Degree Project addresses the detection of financial fraud through the application of machine learning algorithms. Based on a dataset containing over six million banking transactions, a comparative study was conducted between two supervised models: Random Forest and artificial neural networks.

An exploratory analysis and thorough data preprocessing were carried out, including the creation of relevant new variables and normalization techniques. Both models were then trained, evaluated, and compared using various performance metrics, such as precision, recall, F1-score, and the confusion matrix.

The results show that the Random Forest model delivers more robust and efficient performance, with reduced training times and superior predictive capability in detecting fraudulent transactions. Although neural networks showed some limitations in this study—particularly in fraud detection—their potential is acknowledged in more suitable computational environments.

This project highlights the usefulness of machine learning as a tool to combat fraud in real time, providing scalable, accurate, and adaptable solutions to meet the current needs of the financial sector.

1. Introducción

La detección de fraudes es un proceso utilizado en diversos sectores, como los sectores bancario y financiero, de la salud, seguros, y de l administración pública. El fraude, entendido como la manipulación ilegal de información para obtener beneficios de manera ilícita, ha evolucionado rápidamente. Esto se debe a que, con el gran avance de la tecnología y el comercio digital, cada vez son más los métodos usados por los delincuentes para extraer información de sistemas de pago, produciéndose un incremento de la actividad fraudulenta [5].

Los métodos tradicionales de detección de fraude han consistido principalmente en sistemas basados en reglas, donde se establecen condiciones fijas, como restringir el número máximo de transacciones diarías o la cantidad transferida, para marcar transacciones sospechosas. Sin embargo, estos enfoques presentan limitaciones importantes, ya que dependen de la constante actualización de reglas, que pueden ser fácilmente evadidas por estafadores que cambian sus tácticas con el tiempo. Además, generan altas tasas de falsos positivos, afectando a clientes legítimos y perjudicando la experiencia de usuario.

Con el fin de solucionar este problema, se puede recurrir a la aplicación de distintos algoritmos de *machine learning*. Estos modelos permiten el análisis de grandes volúmenes de datos, pudiendo identificar patrones y, por tanto, detectar cuando se produce un resultado inusual, pudiendo clasificarlo como fraude.

Sin embargo, existen desafíos que afectan la precisión de los algoritmos de machine learning, dificultando la detección de fraudes, como una distribución de datos desbalanceada, alta tasa de falsos positivos, detección en tiempo real, el uso de inteligencia artificial generativa y cambio de concepto.

1.1. El potencial de *machine learning* en la detección de fraude

Los algoritmos de aprendizaje automático son capaces de procesar grandes conjuntos de datos de manera más rápida y eficiente que los humanos, lo cual resulta especialmente útil en la detección de fraudes, ya que diariamente se realizan millones de transacciones, algo imposible para un humano. En cambio, un modelo previamente entrenado tiene la capacidad de detectar patrones anómalos en tiempo real y con una alta precisión.

Esta implementación de los algoritmos ofrece múltiples ventajas. En primer lugar, permite identificar transacciones sospechosas con gran rapidez y eficiencia, siendo esto realmente útil para prevenir pérdidas económicas. Además, al estar entrenado para reconocer patrones, es capaz de detectar nuevas formas de fraude, incluso si estás no han sido vistas anteriormente, ya que suelen presentar anomalías respecto a las transacciones legítimas. Esto resulta especialmente conveniente en la actualidad, ya que, a diferencia de unos años atrás, los tipos de fraude son ahora cada vez más diversos y difíciles de identificar.

Otra ventaja que ofrece es una destacable reducción del tiempo de revisión manual, ya que los datos ya han sido vistos y clasificados por el modelo, de manera que los agentes de operaciones de riesgo pueden centrarse únicamente en revisar aquellas transacciones que el modelo ha identificado como potencialmente fraudulentas. Esto se traduce en una mayor eficiencia operativa por parte del equipo de trabajadores y, por tanto, una disminución de costes para las empresas, debido a que ya no hay necesidad de contar con grandes equipos de analistas de riesgo.

Otra ventaja es la posibilidad de aprender en tiempo real, ya que, una vez entrenado, el modelo puede recibir nuevos datos de forma continua y realizar predicciones al instante. Esta actualización de los datos que se le envían puede ser tan rápida como se desee en la institución financiera que lo utiliza [11].

En conclusión, la implementación de *machine learning* en la detección de fraudes no solo mejora la

precisión y la velocidad del proceso, sino que también representa una solución económica rentable y adaptable a nuevas amenazas, convirtiéndose en una herramienta imprescindible en el ámbito financiero actual.

1.2. Objetivos

1.2.1 Objetivo general

Desarrollar y evaluar distintos algoritmos de *machine learning* capaces de detectar transacciones fraudulentas de manera eficiente y rápida, mediante un estudio comparativo entre ellos, con el objetivo de determinar que algoritmo es más eficaz para identificar detectar si una transacción es fraudulenta o no.

1.2.2 Objetivos específicos

- Estudiar los fundamentos de la inteligencia artificial, su evolución histórica y su impacto actual en diferentes sectores, especialmente en el ámbito financiero.
- Analizar en profundidad los modelos de clasificación más utilizados en *machine learning*, como árboles de decisión, *random forest* y redes neuronales artificiales.
- Investigar diferentes métricas de evaluación utilizadas en problemas de clasificación binaria, como precisión, *recall*, *F1-score*, matriz de confusión y curva ROC.
- Comparar las metodologías tradicionales de detección de fraude con los enfoques basados en *machine learning*.
- Explorar los conceptos clave sobre tipos de fraude financiero y las estrategias más comunes utilizadas por los estafadores en entornos digitales.
- Buscar diferentes conjuntos de datos (datasets) sobre transacciones bancarias y seleccionar el más adecuado para el análisis y entrenamiento de modelos predictivos.

- Realizar un análisis exploratorio de datos para detectar patrones y anomalías.
- Crear variables adicionales derivadas que aporten valor al modelo predictivo en términos de comportamiento sospechoso.
- Seleccionar las variables más relevantes mediante análisis de correlación y métricas de importancia de características.
- Implementar un modelo basado en *random forest* y evaluar su rendimiento.
- Implementar una red neuronal profunda ajustando hiperparámetros y arquitectura, y evaluar su rendimiento.
- Comparar los resultados obtenidos por ambos modelos mediante las métricas.
- Determinar cuál de los modelos propuestos es más eficaz en la detección de fraudes, tanto en un conjunto de datos reducido como en uno de mayor volumen.
- Reflexionar sobre las limitaciones encontradas durante el desarrollo del trabajo y proponer líneas de mejora.

2. Marco teórico

2.1. Inteligencia Artificial

La inteligencia artificial (IA) es la capacidad de las máquinas para usar algoritmos, aprender de los datos y utilizar lo aprendido en la toma de decisiones de manera similar a un ser humano. Según Rouhiainen (2018), *"la inteligencia artificial es la habilidad de los ordenadores para hacer actividades que normalmente requieren inteligencia humana"* [10].

El interés por la IA ha crecido exponencialmente en los últimos años debido a su impacto en diversos sectores como la industria, la medicina y la tecnología. Sin embargo, su origen se remonta a la decada de 1950, cuando comenzaron las primeras teorías sobre la posibilidad de que las máquinas imitaran el pensamiento humano. Su auge real no llegó hasta finales de la primera década del siglo XXI gracias a dos factores clave:

- **Mayor capacidad de cómputo:** La evolución del hardware permitió el procesamiento de algoritmos cada vez más complejos de manera eficiente.
- **Aparición del big data:** La acumulación masiva de datos proporcionó la base para el aprendizaje de modelos avanzados, permitiendo que las máquinas mejoraran su rendimiento a partir del análisis de grandes volúmenes de información.

Estos avances han permitido que la IA pase de ser un concepto teórico a una tecnología aplicada en la vida cotidiana, optimizando procesos y mejorando la toma de decisiones automatizada.

2.1.1 Evolución de la Inteligencia Artificial

El desarrollo de la IA ha pasado por diversas etapas clave que han definido su crecimiento y aplicación en el mundo moderno. A continuación, se presentan algunos de los hitos más relevantes en su evolución:

Los primeros avances: El Test de Turing Uno de los primeros intentos de definir la inteligencia artificial fue el Test de Turing, propuesto por Alan Turing en 1950 en su artículo *Computing Machinery and Intelligence*. Esta prueba consistía en que un evaluador debía mantener una conversación en lenguaje natural con otra persona y con una máquina, sin saber cuál era cuál. Si el evaluador no podía diferenciar a la máquina del humano en al menos el 70% del tiempo, la máquina sería considerada inteligente.

Turing replanteó la pregunta sobre si las máquinas pueden pensar, reformulándola en términos de imitación: *"¿Pueden las computadoras digitales simular a un ser humano en el juego de imitación?"*. Este planteamiento dio lugar a la exploración de la IA como un campo de estudio formal.

Crecimiento y Limitaciones Iniciales

Aunque la inteligencia artificial comenzó oficialmente en la década de 1950, su desarrollo inicial estuvo limitado por diversos factores:

- La baja capacidad de cómputo de los ordenadores de la época impedía la ejecución de modelos avanzados.
- La falta de grandes volúmenes de datos hacía que los modelos no pudieran aprender de manera efectiva.

Durante varias décadas, la IA se mantuvo como un campo de investigación teórico con aplicaciones limitadas. No fue hasta finales del siglo XX que se superaron estas barreras, permitiendo avances más significativos.

Avances en el aprendizaje automático y profundo

A finales del siglo XX y principios del XXI, dos avances clave permitieron el despegue definitivo de la IA:

- **Aumento en la capacidad de cómputo:** El desarrollo de procesadores más potentes y la computación en la nube hicieron posible la ejecución de algoritmos complejos en tiempos

reducidos.

- **Big data:** La disponibilidad de enormes cantidades de datos permitió el entrenamiento de modelos de IA con mayor precisión y eficiencia.

Estos avances llevaron al desarrollo del aprendizaje profundo (Deep Learning), una técnica basada en redes neuronales artificiales que ha revolucionado el campo de la inteligencia artificial. Entre sus aplicaciones más destacadas se encuentran:

- **Redes Neuronales Convolucionales (CNN):** Utilizadas en visión por computadora para tareas como el reconocimiento de imágenes y la detección de objetos.
- **Redes Neuronales Recurrentes (RNN):** Aplicadas en el procesamiento del lenguaje natural y traducción automática.
- **Asistentes virtuales y chatbots:** Sistemas como Siri, Alexa y Google Assistant que utilizan IA para interactuar con los usuarios de manera natural.

Hoy en día, la inteligencia artificial sigue evolucionando rápidamente, con aplicaciones en vehículos autónomos, diagnóstico médico, finanzas y muchas otras áreas. Su crecimiento continuo sugiere que su impacto en la sociedad seguirá expandiéndose en los próximos años.

2.2. *Machine learning*

El *machine learning* es una rama de la IA que se centra en la búsqueda de patrones en los datos. Consiste en, una vez resuelto el problema, ser capaz de reconocer la situación problemática y poder solucionarlo usando la estrategia aprendida [9].

La finalidad del *machine learning* es que sea capaz de tomar decisiones sobre cual es el curso mas apropiado que debe seguir la resolucion de un problema y modificar estas decisiones cuando

las condiciones así lo requieran. Da a las computadoras la capacidad de aprender sin ser programados de manera explícita, es decir, el programa aprende a partir de la experiencia.

A diferencia de la programación tradicional, donde en primera instancia se procesa un conjunto de datos de entrada y, por medio de una serie de reglas, se genera una salida, en *machine learning* los datos de entrada y las salidas esperadas constituyen los datos iniciales que, mediante un proceso de entrenamiento, producen reglas. Estas reglas conforman un modelo, siendo dicho modelo el resultado de detectar en los datos patrones o tendencias que se pueden usar para hacer predicciones sobre datos nunca vistos.

Este proceso es realizado por distintos algoritmos, que al ser correctamente ajustados, permiten que el modelo realice buenos resultados predictivos. Dichos algoritmos se clasifican en distintas categorías, dependiendo del tipo de problema que se esté abordando. Los principales tipos de algoritmos de machine learning son:

- Aprendizaje supervisado.
- Aprendizaje no supervisado.
- Aprendizaje por refuerzo.

Los algoritmos más utilizados son los de aprendizaje supervisado, siendo los más destacables los modelos de clasificación y regresión, entre los cuales se encuentran: Máquinas de Soporte Vectorial (SVM), K-Nearest Neighbors (KNN), árboles de decisión, random forest y redes neuronales [**ibm-ml**].

2.3. Modelos de clasificación y regresión

2.3.1 Árboles de decisión

Un árbol de decisión es un modelo predictivo que divide el espacio de los predictores agrupando observaciones con valores similares para la variable respuesta o dependiente.

Para dividir el espacio en subregiones se debe aplicar una serie de reglas, con el fin de que cada subregión contenga la mayor proporción posible de datos de la misma clase.
La estructura de un árbol de decisión consta de distintos nodos, para verlo de manera más clara se representa en la imágen 1

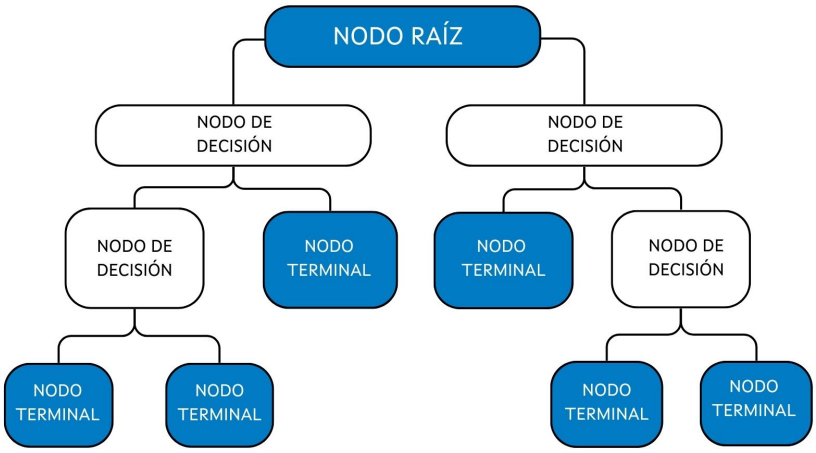

Figure 1: Árbol de decisión [1].

- **Primer nodo o nodo raíz:** produce la primera división en función de la variable más importante.
- **Nodos internos:** dividen el conjunto de datos según otras variables. Cada nodo interno representa una característica que se debe considerar para tomar una decisión.
- **Ramas:** representan la decisión tomada en función de una determinada condición.

- **Nodos terminales u hojas:** indican la clasificación definitiva, es decir, representan el resultado de la decisión.

Entrenamiento

El proceso de entrenamiento de un árbol de decisión se divide en dos etapas:

1. División sucesiva de los predictores generando regiones no solapantes R_1, R_2, ...
   Teóricamente, las regiones podrían tener cualquier forma, pero con el fin de simplificar el proceso de construcción y facilitar la interpretación, se limita su forma a regiones rectangulares.

2. Predicción de la variable respuesta en cada región.
   Es necesario establecer una metodología que permita crear las regiones, o lo que es equivalente, decidir donde se introducen las divisiones. Es en este paso en el que se diferencian los algoritmos de regresión y clasificación.

Proceso de selección de divisiones

Independientemente de la medida empleada como criterio de selección de las divisiones, el proceso siempre es el mismo:

1. Para cada posible división se calcula el valor de la medida en cada uno de los nodos resultantes.

2. Se suman los dos valores, ponderando cada uno por la fracción de observaciones que contiene cada nodo. Este paso es muy importante, ya que no es lo mismo dos nodos puros con 2 observaciones que dos nodos puros con 100 observaciones. Matemáticamente:

$$\frac{n \text{ observaciones nodo A}}{n \text{ observaciones totales}} \cdot \text{pureza A} + \frac{n \text{ observaciones nodo B}}{n \text{ observaciones totales}} \cdot \text{pureza B}. \qquad (1)$$

3. La división con menor o mayor valor (dependiendo de la medida empleada) se selecciona como punto de corte óptimo.

Ventajas y limitaciones de los árboles de decisión

Los modelos basados en árboles de decisión presentan una gran cantidad de ventajas, lo que los hace realmente útiles para diversos tipos de problemas. Una de sus principales ventajas es la facilidad de interpretación, incluso en situaciones en las que las relaciones entre predictores son complejas. Los modelos que están basados en un solo árbol pueden ser representados gráficamente, siempre y cuando el número de predictores sea mayor de 3, de manera que permite visualizar gráficamente la estructura de decisión, tal y como se ha mostrado en la imagen 1.

Otra ventaja extremadamente útil es su flexibilidad para manejar tanto predictores numéricos como categóricos, sin necesidad de realizar transformaciones previas como la creación de variables *dummy* o *one-hot-encoding*. No solo pueden trabajar con distintos predictores, sino que no requieren que los datos sigan distribuciones específicas.

En cuanto al preprocesamiento de los datos, los árboles de decisión destacan por necesitar un tratamiento mínimo en comparación con otros métodos de *machine learning*, como KNN o SVM. Además, son robustos frente a la presencia de *outliers*, ya que su estructura de particionamiento de datos reduce su impacto. También ofrecen soluciones cuando existen valores faltantes, en el en caso en que el valor de una variable no esté disponible es posible realizar predicciones utilizando el grupo de observaciones del último nodo alcanzado.

Este modelo resulta muy útil en la fase de exploración de datos debido a que es capaz de seleccionar las variables más importantes de forma automática, además de construir reglas interpretables. También cabe destacar que es un tipo de modelo que es perfectamente aplicable tanto a problemas de regresión como de clasificación.

En cuanto a las desventajas que presenta este modelo, se debe destacar que la capacidad predictiva de un solo árbol de decisión es bastante inferior a la obtenida mediante otros modelos, como redes neuronales profundas o *random forest*, por eso mismo surgió la necesidad de crear métodos más robustos como *random forest*, que, como se explica en el siguiente apartado, consiste en entrenar diversos árboles de decisión independientes y luego combinar sus predicciones, mejorándose así la

capacidad predictiva, esto se conoce como un ensemble de árboles. Además, son sensibles a conjuntos de datos desbalanceados, lo cual puede empeorar el rendimiento en tareas de clasificación. Cuando un árbol de decisión trata con predictores continuos, se pierde parte de su información al categorizar las variables en el momento de la división de los nodos, lo cual puede limitar la capacidad del modelo para captar pequeñas variaciones en los datos. También, si la creación de las ramificaciones del árbol, es decir, su proceso de construcción se consigue mediante el algoritmo de *recursive binary splitting*, los predictores continuos tienen mayor probabilidad de contener algún punto de corte óptimo, por lo que su probabilidad de ser seleccionados para las divisiones es mayor, pudiéndose crear un sesgo hacia esas variables.

Finalmente, se debe tener en cuenta que el modelo es incapaz de explorar fuera del rango de predictores observado en los datos de entrenamiento, por lo que las predicciones realizadas sobre datos fuera de ese rango tienden a ser erróneas, simplemente lo clasifica como la última regla que encontró. Es importante también que se debe limitar su crecimiento, ya que si esto no se realiza, este modelo presenta tendencia al sobreajuste.

2.3.2 *Random forest*

Random forest es un algoritmo de aprendizaje automático basado en la combinación de múltiples árboles de decisión mediante la técnica bagging. Al usar bagging, lo que realmente se está haciendo es que cada árbol de decisión vea distintas porciones de los datos, ningún árbol ve todos los datos de entrenamiento, de manera que cada árbol es entrenado con una muestra aleatoria de los datos [2].

Esto resulta fundamental, ya que, al combinar los distintos modelos que han cometido errores de maneras diferentes, estos errores tienden a cancelarse entre sí, logrando una mayor capacidad de generalización y reduciendo el riesgo de sobreajuste respecto a un único árbol de decisión. Esta estructura se ve en la imagen 2.

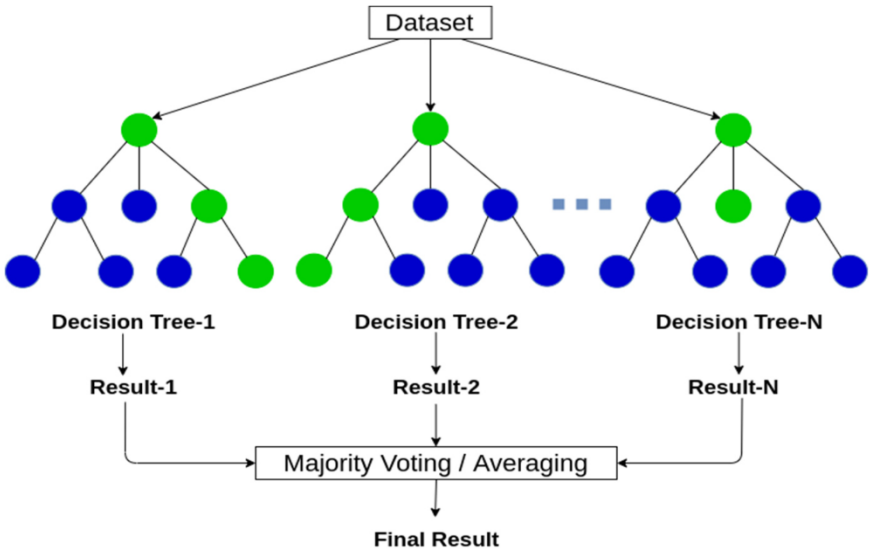

Figure 2: Random Forest [3].

En estos modelos se cuenta con una doble aleatoriedad, gracias a la cual se mejora la independencia entre los distintos árboles de decisión:

- **Aleatoriedad a nivel de registro:** cada árbol utiliza una muestra aleatoria de los registros del conjunto de datos de original.
- **Aleatoriedad a nivel de características:** durante el proceso de construcción de cada árbol, en cada punto de decisión (nodo) se selecciona aleatoriamente un subconjunto de las variables disponibles. Es decir, cada modelo emplea características diferentes y aleatorias.

Ventajas y limitaciones de *random forest*

El modelo *random forest* presenta una serie de ventajas, convirtiéndolo en un modelo muy utilizado en el aprendizaje automático. Una de las principales ventajas es que, al usar múltiples árboles de decisión, consigue disminuir la varianza de la predicción en comparación con un modelo basado en un solo árbol, generando así modelos más estables y generalizables a datos nuevos.

Otra gran ventaja es que no requiere realizar suposiciones sobre la distribución de los datos, lo

cual lo hace aplicable a una gran variedad de problemas. Esto provoca que la preparación de los datos sea mínima, ya que puede trabajar tanto con variables categóricas como numéricas. Además funciona sin realizar un ajuste de hiperparámetros, lo que se traduce en un modelo robusto al sobreajuste.

Es capaz de manejar miles de variables de entrada, convirtiéndolo en un modelo perfecto para problemas de *big data*, y identificando los parámetros más significativos para la predicción.

Este modelo también tiene con una serie de desventajas que se deben tener en cuenta. Entre estas, se puede destacar la pérdida de interpretación, debido a que a medida que se aumenta el número de árboles de decisión, más complejo se vuelve entender cómo el modelo toma las decisiones. Aunque trabaja excelentemente en tareas de clasificación, en problemas de regresión su eficiencia es menor, ya que, al estar compuesto por varios árboles de decisión, presenta ese mismo problema de no poder explorar datos fuera del rango de valores del conjunto de entrenamiento.

Otro inconveniente es el poco control que se tiene sobre el modelo, ya que al ser un conjunto de varios árboles es más complejo personalizarlo. Además, si se trabaja con conjuntos de datos pequeños, este modelo tiende a sobreajustarse, por lo que para datasets pequeños es más conveniente trabajar con un solo árbol de decisión o un conjunto pequeño, debido a que al ser un dataset con pocos datos no es necesaria tanta complejidad ni tanto tiempo de entrenamiento, ya que *random forest* tiende a ser un modelo computacionalmente costoso en el caso de trabajar con un gran número de árboles o grandes volúmenes de datos.

2.3.3 Redes neuronales

Neurona biológica vs. neurona artificial

En una neurona biológica, las señales eléctricas se transmiten a través de dendritas hacia el cuerpo de la célula, donde se procesan. Si la señal es lo suficientemente fuerte, se envían a otras neuronas a través del axón. Su aprendizaje se basa en la adquisición de nuevos conocimientos a través de experiencias propias.

Una neurona artificial recibe entradas (*inputs*) de múltiples fuentes y las procesa a través de una función de activación. Cada entrada tiene un peso asociado que representa la importancia o intensidad de esa señal. La neurona artificial suma todas estas señales ponderadas, cuyo resultado es pasado por una función de activación que determina la salida (*output*) de la neurona. Esta salida es entonces transmitida a las neuronas en la siguiente capa, de forma similar a como una neurona biológica transmite impulsos eléctricos. Su aprendizaje requiere un conjunto de datos de entrada además de un proceso de oprimización. La comparación de ambas se presenta en la imagen 3.

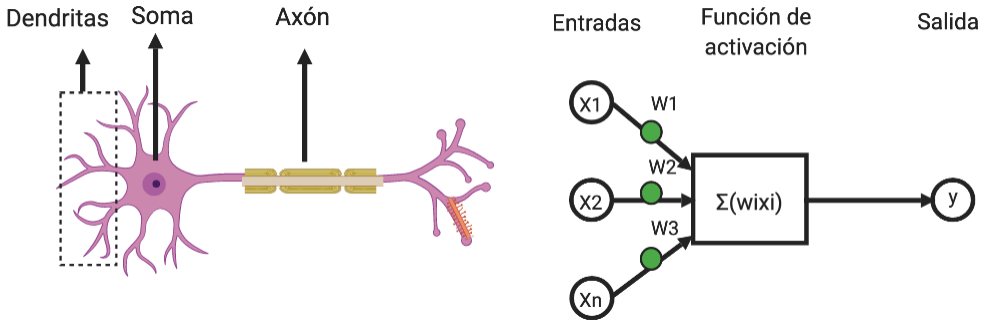

Figure 3: Neurona biológica vs. artificial [7].

Arquitectura de una red neuronal.

Una red neuronal artificial se compone de múltiples neuronas organizadas en capas. Siendo su estrucutra típica la siguiente:

- **Capa de entrada:** recibe los datos de entrada, y cada neurona que se encuentra ubicada en esta capa representa una característica o variable del conjunto de datos.
- **Capas ocultas:** la red neuronal puede contener varias capas ocultas, en las cuales ocurre el procesamiento interno. Las neuronas en estas capas realizan transformaciones de los datos a través de funciones de activación.
- **Capa de salida:** en esta se genera la predicción o resultado final de la red. El número de neuronas en esta capa depende del problema.

Tanto la topología de la red, es decir, cómo están organizadas y conectadas las neuronas, como la dimensión de las capas ocultas, cuya elección depende de la complejidad del problema y de la

cantidad de datos, tienen un impacto directo en su capacidad para aprender y generalizar patrones en los datos.

Entrenamiento de una red neuronal

El entrenamiento de una red neuronal es el proceso mediante el que la red ajusta sus parámetros internos, principalmente los pesos y sesgos de las conexiones entre neuronas, con el fin de aprender patrones a partir de los datos y así minimizar el error. Para esto, se utiliza la técnica de descenso de gradiente.

El descenso del gradiente es una técnica de optimización utilizada para ajustar los parámetros de una red neuronal, con el objetivo de minimizar la función de pérdida, lo que equivale a reducir el error del modelo en sus predicciones. La principal idea es representar el error como una función continua de los pesos e intentar obtener configuraciones de valores que nos permitan obtener el mínimo de esta función.

Para poder explicar el descenso del gradiente, primero debemos entender que es el gradiente de una función. Este representa la dirección en la que dicha función crece más rápidamente, por lo que, en el contexto de aprendizaje automático, el gradiente de la función de pérdida con respecto a los pesos indica la dirección de mayor crecimiento de la pérdida, por lo que, al moverse en la dirección opuesta, se consigue reducir el error.

Debido a la complejidad de una red neuronal, donde existen muchos parámetros que deben ser optimizados de manera eficiente, para poder minimizar el error, se utiliza el algoritmo de *back-propagation*, que consta de dos fases:

1. **Propagación hacia adelante:** se toman unos valores de entrada (x), que se transmiten a través de la red neuronal. En cada neurona, se calcula una suma ponderada de las entradas más un sesgo, y luego se aplica una función de activación para obtener la activación de la neurona. Con esto, obtendremos la salida final de la red, cuyo valor debe ser comparado

con el de la salida real, cuya diferencia se calcula utilizando una función de pérdida. Esta diferencia es conocida como el error.

2. **Retropropagación:** usando el error obtenido en la propagación hacia adelante, se calculan los gradientes de la función de pérdida con respecto a los pesos y sesgos mediante la regla de la cadena. Luego, estos gradientes se propagan hacia atrás para ajustar los parámetros y minimizar el error.

Esta técnica se repite durante múltiples épocas, hasta que la red alcanza un nivel óptimo de rendimiento, es decir, cuando el error deja de disminuir significativamente.

Hiperparámetros

Los hiperparámetros son valores externos al modelo, los cuales deben ser definidos manualmente antes del entrenamiento. A continuación son explicados:

1. **Funciones de activación**

   La finalidad de las funciones de activación es introducir no linealidad en el modelo, permitiendo que la red aprenda y represente relaciones complejas entre las entradas y las salidas. Las típicas funciones de activación son:

   - **Sigmoide:** es utilizada en la capa de salida en problemas de clasificación binaria, devolviendo un valor comprendido entre 0 y 1.
   - **Softmax:** es una generalización de la función *sigmoide* para problemas de clasificación multiclase. Los valores que devuelve asociados a cada probabilidad también se encuentran entre 0 y 1.
   - **Tangente hiperbólica:** se utiliza cuando se requeiren salidas centradas alrededor de 0 y se necesite una función suave y no lineal. Su rango se encuentra comprendido entre -1 y 1.
   - **ReLU:** es la función de activación más utilizada en redes neuronales profundas debido a su mayor eficiencia y rendimiento.

2. **Tasa de aprendizaje**

   La tasa de aprendizaje establece a que velocidad el modelo ajusta sus parámetros durante cada iteración. Cuanto mayor sea esta, más rápido se ajustará el modelo, pero puede generar un rendimiento inestable si este es ajustado demasiado deprisa. Es por esto que suele escogerse una tasa de aprendizaje más baja, ya que aunque tome más tiempo de cómputo, garantiza encontrar el punto de pérdida mínima.

3. **Optimizadores**

   Los optimizadores son algoritmos o métodos cuya finalidad es cambiar los parámetros del modelo de aprendizaje automático para reducir las pérdidas. Aunque el más conocido es el descenso del gradiente, el cual ya ha sido explicado, este puede ser excesivamente lento si usamos grandes conjuntos de datos, por lo que se recurre al uso de distintos optimizadores:

   - **Adagrad:** adapta la tasa de aprendizaje para cada parámetro en función de la frecuencia de actualización de sus gradientes, permitiendo que los parámetros que cambian con mayor rapidez al inicio tengan tasas de aprendizaje menores, y viceversa
   - **RMSprop:** la tasa de aprendizaje se calcula como la media exponencial de los gradientes de iteraciones anteriores.
   - **Adam:** genera una tasa de aprendizaje adaptativa para cada parámetro

4. **Tamaño del lote (batch size)**

   Determina el número de muestras que el modelo debe calcular antes de actualizar sus parámetros. Por lo general, cuanto mayor sea este número, menor rendimiento del modelo, aunque si se ajusta correctamente con la tasa de aprendizaje este puede mejorar.

5. **Número de epocas (epochs)**

   Establece la cantidad de veces que el modelo se expone a todos los datos del conjunto de entrenamiento. A medida que esta se aumenta, también se mejora el rendimiento de la red, aunque existe la posibilidad de que se produzca un sobreajuste.

6. **Función de pérdida**

   Esta función es definida antes del entrenamiento de la red y su finalidad es medir el error entre las predicciones que ha hecho el modelo y los valores reales. A diferencia del resto de hiperparámetros, esta no se ajusta, ya que su elección depende únicamente del problema que se esté resolviendo.

   - **Error cuadrático medio (MSE):** es la función de pérdida más utilizada en problemas de regresión. Calcula la media de las diferencias al cuadrado entre los valores previstos y los reales
   - **Error medio absoluto (MSE):** esta mide la diferencia absoluta media entre los valores de las predicciones y los reales. Al igual que el MSE, se utiliza en problemas de regresión, especialmente en el caso en el que haya valores atípicos.
   - **Entropía cruzada binaria:** se utiliza para problemas de clasificación, concretamente para los problemas en los que hay dos clases. Los valores que esta devuelve están comprendidos entre 0 y 1. En el caso de la detección de fraudes, 0 es cuando es una transacción legítima y 1 es fraudulenta, dependiendo del valor que devuelva para cada entrada, es posible conocer que tipo de transacción se ha realizado.
   - **Entropía cruzada categórica:** utilizada también para problemas de clasificación, aunque en este caso pueden existir más de dos clases, por lo que genera un valor posible asociado a cada clase. Los valores de salida, al igual que la *entropía cruzada binaria* también están comprendidos entre 0 y 1, y la suma de todos ellos debe dar 1. Suele ser combinada con la función de activación *softmax*.
   - **Arquitectura de la red**

     Aunque esta ha sido explicada anteriormente, se debe nombrar que, tanto el número de capas ocultas como el número de neuronas por capa, son hiperparámetros. Estas afectan a la complejidad del modelo y a su capacidad de predicción.

Ventajas y limitaciones de las redes neuronales

Entre las principalaes ventajas de las redes neuronales artificiales se puede destacar la elevada capacidad de aprendizaje, permitiéndo aprender patrones complejos y no lineales en grandes conjuntos de datos. Además, es un modelo altamente flexible, siendo capaz de trabajar con datos de diversa naturaleza, como texto, imágenes, señales, secuencias temporales o audio, entre muchas otras. Son capaces también de entrenar el modelo en tiempo real, ya que permiten que los datos sean actualizados a medida que se van generando.

Aunque es un modelo con importantes ventajas, también cuenta con una serie de desventajas, por ejemplo la necesidad de un conjunto grande de datos para su entrenamiento, puesto que si este conjunto es demasiado pequeño podría llegar a producirse un sobreajuste, ya que este modelo tiene una alta capacidad de aprendizaje. Las redes neuronales tienen estructuras internas muy complejas, por lo que resulta difícil entender el razonamiento detrás de cada una de sus predicciones.
Tal y como se ha nombrado, constan de una elevada cantidad de hiperparámetros que deben ser ajustados, lo cual puede resultar en un entrenamiento costoso tanto en tiempo como computacionalmente. Aunque pueden trabajar con una gran diversidad de datos, son muy sensibles a la escala de estos, por lo que se requiere aplicar técnicas de normalización para asegurarse de que todos tengan una escala similar [8] [9].

2.4. Métricas de evaluación

2.4.1 Matriz de confusión

En el caso en el que nuestro problema sea un problema de clasificación binaria, se pueden generar dos errores: falsos positivos y falsos negativos. Para poder evaluar estos, se recurre a la matriz de confusión, la cual se estructura como se ve en la siguiente tabla 1.

A partir de los valores que presente la matriz, se pueden calcular distintas métricas

	Predicción: No Fraude	**Predicción: Fraude**
Real: No Fraude	TN (Verdaderos Negativos)	FP (Falsos Positivos)
Real: Fraude	FN (Falsos Negativos)	TP (Verdaderos Positivos)

Table 1: Matriz de Confusión para Detección de Fraudes

- **Accuracy:** indica la proporción de predicciones correctas sobre el total de predicciones realizadas.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{TP} + \text{FN}}. \tag{2}$$

- **Precision:** es la capacidad de ser específico, indica la proporción de predicciones positivas que son correctas:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{3}$$

- **Recall:** mide cuántos de los casos positivos reales fueron detectados:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4}$$

- **F1-Score:** mide el equilibrio entre precision y recall. Resulta especialmente útil cuando hay una gran diferencia entre los falsos positivos y los falsos negativos:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{5}$$

  Cuanto más próximo del valor 1 se encuentre, mejor es la predicción del modelo.

2.4.2 Curva ROC

La curva ROC es un método alternativo para evaluar la calidad de un modelo de clasificación binaria. Es especialmente útil para modelos en los que las clases estén desbalanceadas.

Esta curva ilustra la capacidad del modelo para discriminar entre clases positivas y negativas,

graficando la tasa de verdaderos positivos (TPR) frente a la tasa de falsos positivos (FPR).

- **TPR:** también conocida como sensibilidad, mide la proporción de instancias positivas correctamente clasificadas:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{6}$$

- **FPR:** mide la proporción de instancias negativas que han sido incorrectamente clasificadas como positivas:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}. \tag{7}$$

Tal y como se observa en la imagen 4, una curva ROC que se acerca más a la esquina superior izquierda indica un mejor rendimiento del modelo, ya que refleja una alta tasa de verdaderos positivos y una baja tasa de falsos positivos, de manera que cuanto más se curve hacia esa esquina, mayor es la capacidad del modelo para discriminar entre las clases, es decir, muestra una mejor precisión. En cambio, si la curva se acerca más hacia la diagonal, indica un rendimiento similar al de un clasificador aleatorio, mientras que si está por debajo de la diagonal, acercándose a la esquina inferior derecha, se puede decir que un clasificador aleatorio obtendría mejores resultados que el modelo entrenado.

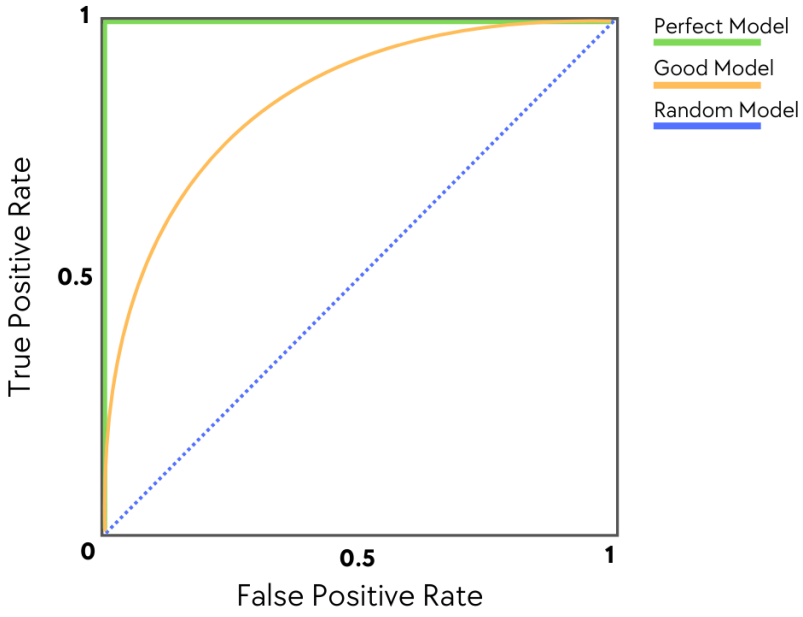

Figure 4: Curva ROC [12].

3. Fraude

El fraude en transacciones financieras hace referencia a cualquier actividad maliciosa que busca obtener beneficios económicos no autorizados mediante la manipulación de sistemas de pago digitales o plataformas financieras. Este tipo de fraude suele aprovechar vulnerabilidades en la infraestructura tecnológica o en los procesos de validación para realizar transacciones ilegítimas, suplantar identidades o evadir controles de seguridad establecidos.

El fraude incluye [4] [13]:

- ***Phising*:** consiste en hacerse pasar por otra persona, es decir, suplantación de la identidad. Funciona enviando correos electrónicos procedentes de una fuente que parece confiable, y mediante enlaces o archivos adjuntos, robar información personal del destinatario del correo, como pueden ser los datos bancarios.

- ***Spoofing*:** esta estafa tiene el mismo formato que el phising, archivos fraudulentos a través de un correo electrónico, pero la finalidad de este es que, al descargar dicho archivo, se instala automáticamente en el dispositivo un malware (software malicioso). Al infiltrarse en el dispositivo, su finalidad es robar datos o tomar el control del sistema.

- **Tiendas online fraudulentas:** diversas webs fraudulentas suplantan la identidad de empresas en las que la gente normalmente confía, pero ofreciendo precios mucho más económicos. En el momento de realizar el pedido, la web roba los datos del cliente, y nunca se realiza la entrega del pedido.

- **Cuentas de *trading* financiadas:** empresas que ofrecen la posibilidad de acceder a una cuenta de compra y venta de acciones. Al proporcionar capital, incremental la captación de usuarios, pero para poder acceder a las cuentas de *trading*, se debe realizar un curso con coste. Una vez aboando el dinero para realizar el curso, este no es devuelto, además de no ofrecerles el acceso a las cuentas.

- **Compromiso del correo electrónico empresarial:** tras un estudio social sobre el usuario se le envía un correo en el que se le demanda un pago que tiene pendiente. El correo puede parecer legítimo, por lo que, al seguir las instrucciones explicadas, el destinatario del correo transferirá el dinero a una cuenta dirigida por los estafadores.
- **Fraude caritativo:** el estafador se hace pasar por una entidad benéfica, de manera que varias personas realizan donaciones mediante transferencias bancarias. El dinero transferido nunca llega a utilizarse con un fin benéfico, sino que acaba en cuentas controladas por los autores del fraude.
- ***Skimming*:** los atacantes utilizan un dispositivo que conectan a lectores de tarjetas, clonando la información de la banda magnética de la tarjeta. Con esta información se crean tarjetas falsificadas o realizan compras fraudulentas.

3.1. Detección de transacciones fraudulentas

3.1.1 Detección tradicional

Para poder detectar si la transacción realizada es legítima o fraudulenta, anteriormente se realizaban métodos tradicionales, los cuales se basaban en reglas fijas y revisiones manuales.
Los pasos de la detección de fraude tradicional son los siguientes:

1. **Métodos basados en reglas:** se utilizaban sistemas basados en reglas que analizaban las transacciones según los criterios predefinidos. Entre estos criterios se encuentran:
    - Compras inusualmente elevadas.
    - Ubicación sospechosa.
    - Frecuencia de transacciones.
2. **Revisión manual:** una vez las reglas estaban definidas, los bancos contaban con equipos de analistas de fraude, quienes revisaban las transacciones detectadas como inusuales para comprobar si verdaderamente eran fraudulentas.

3. **Solución:** como respuesta a las actividades sospechosas, los bancos implementaban medidas de seguridad adicionales como:

    - Límites de gasto.
    - Códigos de verificación, como por ejemplo por SMS o correo electrónico.
    - Confirmaciones manuales por parte del cliente o del banco.

3.1.2 Detección con algoritmos de *machine learning*

Actualmente, se utilizan algoritmos de *machine learning* para la detección de fraudes, debido a que aprenden automáticamente de los datos y pueden detectar patrones complejos en tiempo real, además de tener una menor tasa de falsos positivos.
Los pasos para la detección de fraude con algoritmos de *machine learning* son los siguientes:

1. **Recolección y preparación de datos:** los algoritmos de *machine learning* necesitan un gran volumen de datos para aprender patrones y generar predicciones. Este proceso implica la recolección de datos y sus transformaciones necesarias.

2. **Entrenamiento del modelo:** una vez se tienen recopilados los datos, se deben entrenar con distintos modelos, cuya elección depende de la cantidad de datos que se tenga o de la finalidad del problema.

    - **Modelos supervisados:** estos modelos aprenden patrones de transacciones legítimas y fraudulentas en función de datos.
    - **Modelos no supervisados:** se utilizan cuando no se tienen suficientes datos etiquetados.

3. **Predicción en tiempo real:** cuando se realiza una transacción, el modelo de *machine learing* procesa rápidamente los datos nuevos que han sido recolectados para predecir si esta es fraudulenta o no.

4. **Ajuste y mejora del modelo:** dado que los fraudes evolucionan, los modelos deben actualizarse continuamente con nuevas transacciones etiquetadas para mejorar su precisión.

Ambos enfoques, el tradicional y el basado en *machine learning*, no son excluyentes. Su combinación puede ofrecer una solución más eficaz. Al clasificar el tipo de transacción mediante algoritmos, hay casos que pueden estar clasificados erróneamente, por lo que la revisión manual de un experto permite analizar los casos clasificados incorrectamente, permitiendo así reducir el número de falsos positivos y falsos negativos.

4. Metodología

4.1. Datos utilizados

El conjunto de datos utilizado para el estudio se encuentra disponible en [6].

El dataset contiene un total de 6362620 filas, y las siguientes once columnas:

- ***step*:** indica el número de paso o tiempo en la simulación,
- ***type*:** muestra el tipo de transacción:
  - *CASH_OUT*: retirada de efectivo de la cuenta bancaria,
  - *DEBIT*: cargos automáticos en la cuenta,
  - *PAYMENT*: pagos de factura o servicios,
  - *TRANSFER*: transferencia de dinero entre cuentas.
- ***amount*:** cantidad de dinero transferida en la transacción,
- ***nameOrig*:** ID del remitente,
- ***nameDest*:** ID del destinatario,
- ***newbalanceOrg*:** saldo de la cuenta de origen después de la transacción,
- ***oldbalanceOrg*:** saldo de la cuenta de origen antes de la transacción,
- ***newbalanceDest*:** saldo de la cuenta de destino después de la transacción,
- ***oldbalanceOrg*:** saldo de la cuenta de destino antes de la transacción,
- ***isFlaggedFraud*:** proprociona información sobre que transacciones son marcadas como potencialmente por el sistema (1 para marcada y 0 para no marcada),
- ***isFraud*:** indica que transacciones son fraudulentas.

4.2. Lenguaje de programación

El lenguaje de programación utilizado para crear el código es Python. Python es un lenguaje de programación muy popular en el desarrollo de aplicaciones web, software, ciencia de datos y aprendizaje automático. Su sintaxis sencilla y su eficiencia lo convierten en una herramienta ideal tanto para principiantes como para expertos. Además, es multiplataforma, de código abierto y gratuito, lo que facilita su integración con distintos sistemas y acelera significativamente el proceso de desarrollo.

Concretamente, se ha utilizado la versión de Python 3.10.11.

4.3. Librerías

En Python, las librerías (también llamadas bibliotecas o módulos) son conjuntos de funciones, clases y métodos predefinidos que permiten ampliar las capacidades del lenguaje sin necesidad de programar desde cero. Estas herramientas especializadas facilitan tareas como el análisis de datos, la visualización, la manipulación de archivos o la implementación de modelos de machine learning, entre muchas otras.

Pandas

Pandas es una de las libreráis más usadas en el lenguaje de Python debido a su excelente funcionamiento para trabajar con grandes conjuntos de datos. Esta se utiliza para la manipulación y análisis de datos estructurados en forma de DataFrames.

Matplotlib

Utilizada para la visualización de datos, permitiendo la generación de diferentes tipos de gráficos con apariencia sencilla a partir de los datos y, lo que facilita su comprensión.

Seaborn

Es una librería creada a partir de Matplotlib, por lo que su utilidad es también la creación y visualización de gráficos a partir de los datos, con la diferencia de que estos son de mayor calidad.

Scikit-learn

Es una de las librerías de Machine Learning más utilizada para la implementación de algoritmos de aprendizaje, tanto supervisado como no supervisado.

TensorFlow

TensorFlow es una plataforma de código abierto desarrollada por Google que permite construir, entrenar y desplegar modelos de aprendizaje automático y profundo. Su versatilidad la hace ideal para tareas complejas como la detección de transacciones fraudulentas. Además, ofrece herramientas de visualización que facilitan el análisis de la arquitectura y el comportamiento del modelo durante el entrenamiento.

5. Desarrollo

5.1. Análisis exploratorio de datos

Antes de empezar con la construcción de modelos de predicción, se realizó un análisis exploratorio del conjunto de datos con el objetivo de comprender su estructura, identificar patrones relevantes, detectar posibles anomalías y evaluar la distribución de las variables disponibles. Este proceso es fundamental en proyectos de *machine learning*, ya que permite tomar decisiones informadas respecto al preprocesamiento, la selección de características y la estrategia de modelado a seguir.

Se empieza con la carga del conjunto de datos y se aplica un análisis descriptivo estadístico para conocer el tamaño y, debido a que la variable *isFraud* es de tipo booleana, se explora el nivel de desbalance de datos. Este se ve claramente en la imagen 5.

Esta exploración da como resultado un conjunto de datos altamente desbalanceado, siendo el 99.87% de transacciones no fraudulentas y resultando solo en un 0.13% de transacciones fraudulentas.

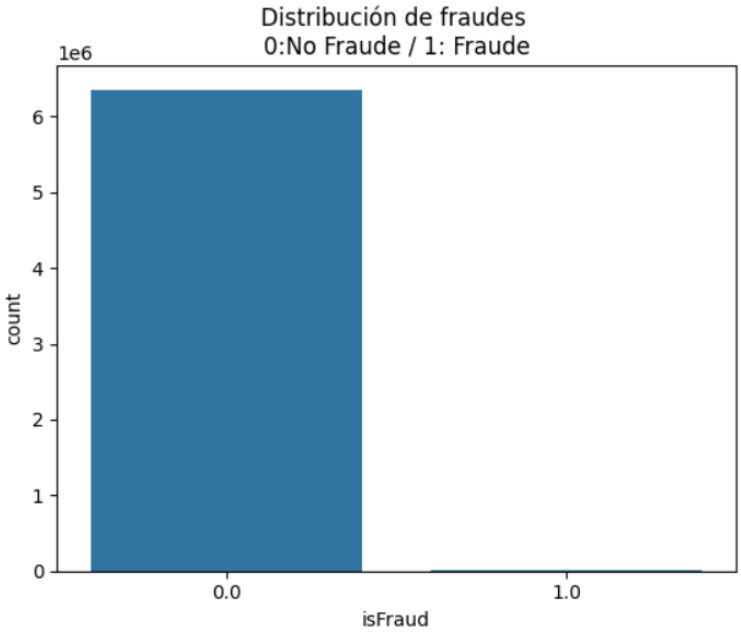

Figure 5: Distribución de clases

5.2. Preprocesamiento de datos

Como se comentó anteriormente, la variable TYPE incluye 4 categorías de transacciones distintas. Con el fin de facilitar su tratamiento, se aplica get_dummies, una función de pandas que genera una nueva columna para cada categoría. De esta manera, se crean 4 variables nuevas en formato binario (type_CASH_OUT, type_DEBIT, type_PAYMENT y type_TRANSFER) cuyo valor es 1 si la transacción corresponde a ese tipo y 0 en caso contrario.

Además, para que los algoritmos de aprendizaje automático puedan aprender y generalizar patrones a partir de los datos, es necesario que todas las variables sean numéricas, ya que modelos como *random forest* o redes neuronales, que son los utilizados en este trabajo, no son capaces de interpretar información a partir de datos categóricos. Por este motivo, en la parte de preprocesamiento de datos, se incluye la transformación de las variables, convirtiendo todas las numéricas a formato float, y las dos variables de texto (nameOrig y nameDest) son transformadas a números enteros mediante el uso del codificador LabelEncoder, cuya selección es debida a que su función es convertir datos de texto a números aleatorios, sin aportarles ningún significado específico.

Se debe destacar que, para el modelo entrenado con la red neuronal, debido a que este algoritmo es muy sensible a la escala de los datos y el dataset contiene variables (*"amount"*, *"oldbalanceOrg"*, *"newbalanceOrig"*, *"oldbalanceDest"*, *"newbalanceDest*) cuyos valores son extremadamente más elevados que los del resto de variables, se aplica la técnica de normalización MinMaxScaler, cuya elección se debe a que es un método que normaliza todos los datos en un rango de 0 a 1.
Aunque tradicionalmente la normalización de los datos se aplica justo antes de entrenar el modelo, en el caso de este TFG, se ha realizado el estudio para ambos casos, al aplicarlo justo antes del entrenamiento del modelo, o antes de estudiar las correlaciones entre las variables, la elección de la segunda opción se basa en los resultados obtenidos, debido a que se obtiene un modelo más eficiente y con más transacciones fraudulentas detectadas.

Una vez aplicada la normalización de las variables para el modelo basado en una red neuronal y siendo ya todas las variables de tipo numérico, se empieza a estudiar la correlación entre ellas. Este estudio tiene como objetivo conocer cuales eran las variables más relacionadas con las transacciones fraudulentas, lo que puede sugerir que algunas de ellas presentan un comportamiento anómalo cuando se realizan fraudes.

Para ello, se calcula una matriz de correlación entre todas las variables mediante el coeficiente de correlación de Pearson. La matriz de correlación es una tabla que muestra como las distintas variables se relacionan entre sí y indica si hay correlación positiva o negativa.

Después se utiliza la función de mapa de calor de seaborn para crear la gráfica acorde a una escala de colores que resaltaba las correlaciones tanto positivas (rojo) como negatiavs (azul), lo cual resulta útil para observar con mayor facilidad cómo se relacionan las variables. (imagen 6)

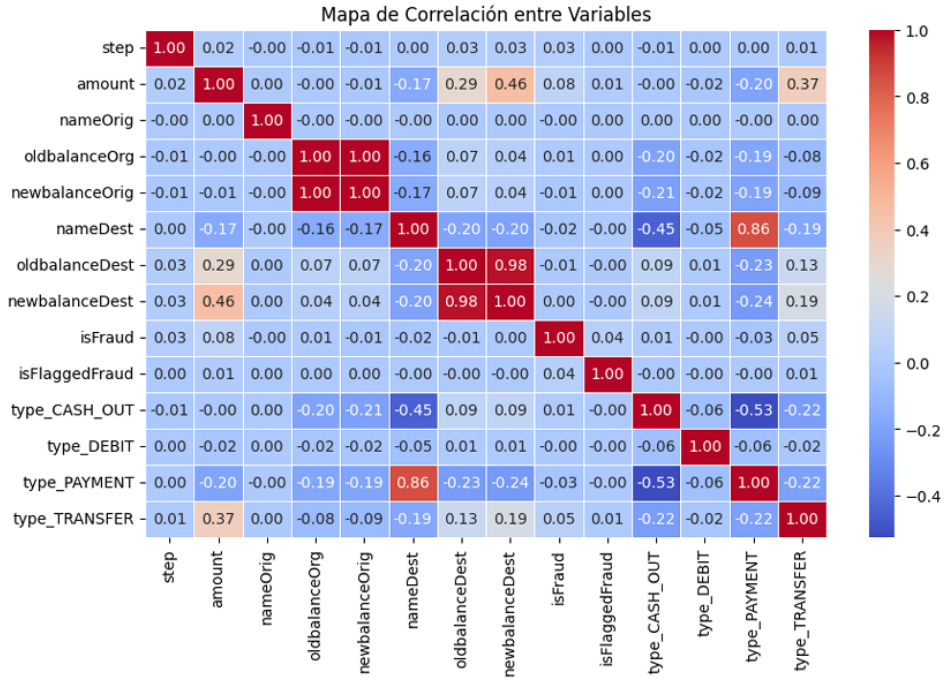

Figure 6: Mapa de correlación entre las variables originales del dataset.

Como se puede observar en la imagen 6, ninguna variable tiene una alta relación con la de *isFraud*, también la variable *nameOrig* no muestra correlación con ninguna, debido a que cada transac-

ción ha sido realizada por un usuario distinto. Y las correlaciones más altas, como *nameDest* y *type_PAYMENT*, se deben a que las transacciones de tipo PAYMENT corresponden a pagos automáticos domiciliados, por lo que están dirigidas a un conjunto recurrente de destinatarios, generalmente las empresas proveedoras de servicios, lo que justifica la fuerte relación entre este tipo de transacción y el destinatario.

Debido a que ninguna variable mostro una fuerte relación con isFraud o con alguna otra que pueda indicar alguna anomalía, creo nuevas variables a partir de las ya existentes, siendo estas nuevas variables:

- **saldo_cambio_origen:** representa la variación del saldo de la cuenta emisora. Cuanto más cercano este valor al de amount, mayor indicio de que la transacción es legítima, mientras que si es muy diferente, puede indicar que la transacción es fraudulenta.
- **saldo_cambio_destino:** esta variable muestra la variación entre el saldo de la cuenta receptora al realizarse la transacción. Si el resultado es 0 puede indicar que la transacción nunca llegó al destinatario, pudiendo ser una indicación de que la transacción fue fraudulenta.
- **proporcion_transferencia:** es la proporción entre la cantidad de dinero transferida (amount) y el saldo total disponible el emisor (oldbalanceOrg). Ayuda a detectar si el emisor transfirió gran parte de su saldo.
- **saldo_cero_origen:** es una variable booleana que indica si el emisor se quedo sin saldo después de realizar la transacción, lo cual, si es así, puede dar sospechas de que es un fraude.
- **destino_sin_saldo:** también es una variable booleana, que en este caso muestra si el destinatario no tenía saldo antes de realizarse la transacción.
- **transacción_alto_riesgo:** esta variable booleana evidencia si en una transacción del tipo TRANSFER o CASH_OUT el monto transferido supera el 90% del saldo original.

- **frecuencia_transacciones_origen:** muestra el número de transacciones que ha realizado un mismo emisor.
- **frecuencia_transacciones_destino:** muestra el número de veces que un destinatario ha recibido transacciones.

Teniendo estas nuevas variables, se crea un nuevo dataset que contiene tanto las originales como las nuevas variables, y se grafica un mapa de correlación, al igual que se ha hecho anteriormente. Este mapa es representado en la figura 7.

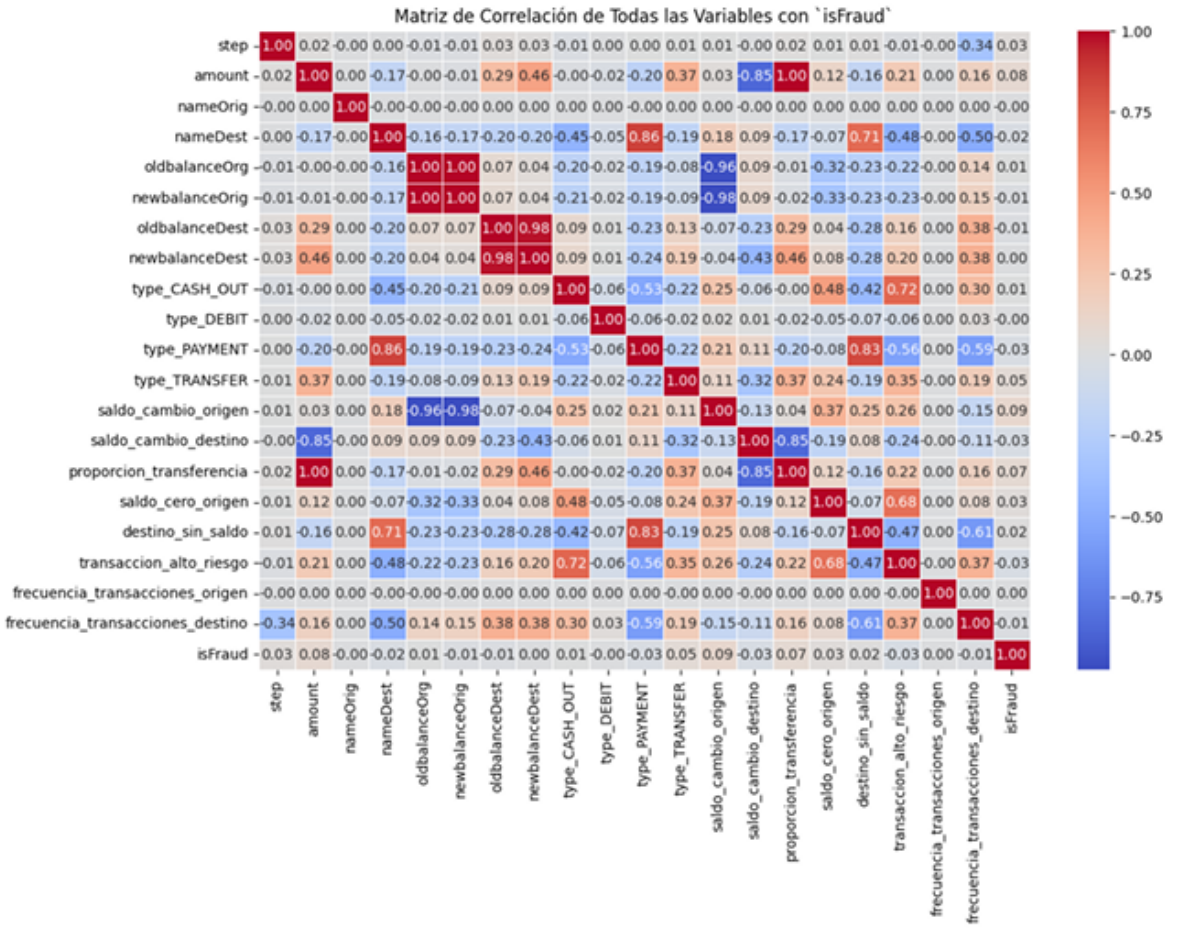

Figure 7: Segundo mapa de correlación con las nuevas variables.

En este nuevo mapa, tampoco aparece ninguna variable con gran relación con *isFraud*, y las nuevas altas relaciones que tenemos provienen de que las nuevas variables son creadas a partir de las orig-

inales, mostrándose por tanto una correlación en el mapa.

Una vez finalizada esta etapa de preprocesamiento de los datos, se procede a la implementación y entrenamietno de algoritmos de *machine learning* para la detección de fraudes. En el caso de este trabajo, se implementan dos metodologías distintas con el objetivo de comparar su eficacia. Por un lado se implementa una metodología basada en el modelo de aprendizaje automático *random forest*, y por otro, una basada en una red neuronal profunda. Ambos enfoques van a ser explicados detalladamente en los siguientes apartados.

5.3. Metodología 1: Random Forest

Este modelo es especialmente resistente al sobreajuste, además de ideal para trabajar con grandes volúmenes de datos, ya que cada árbol entrena sobre un subconjunto de los datos, cuyos resultados son combinados para generar una predicción más precisa.
Esto lo convierte en un algoritmo conveniente para los problemas de clasificación binaria, como la detección de transacciones fraudulentas.

5.3.1 Selección de variables

En la primera etapa se identificaron aquellas variables que fueran más relevantes para la detección de fraudes, con el objetivo de reducir la dimensionalidad del problema y mejorar tanto la eficiencia como la precisión de modelo que más adelante será entrenado.

En primer lugar, se crea un conjunto de datos menor al original, pero que mantiene el mismo desbalance, concretamente con 20000 muestras en las que la transacción es legítima y otras 20 en las que es fraudulenta, obteniendo una selección de variables válida para trabajar con ellas.
Se separaran las variables independientes (las cuales contienen información útil para la predicción) de la variable objetivo (isFraud), aunque se descartaron tres variables (step, nameOrig y nameDest) ya que step es simplemente el número de intervalo de tiempo y nameOrig y nameDest son los iden-

tificadores de los emisores y receptores de transacciones, cuyos valores podrían aportar ruido al modelo, ya que, aunque nameDest pueda tener valores repetidos, esto se debe a que en algunos tipos de transacción, como las de *PAYMENT*, el destinatario es una empresa de servicios como el agua o la luz.

A continuación, se entrenó un modelo de Random Forest utilizando 40 árboles de decisión, cantidad determinada tras realizar múltiples pruebas comparativas. Este entrenamiento inicial se llevó a cabo sobre el conjunto de datos completo, sin dividirlo en subconjuntos de entrenamiento y prueba, con el único propósito de identificar las variables más relevantes dentro del modelo.

Después del entrenamiento, usando la métrica feature_importances_ que muestra que variables son más útiles para predecir fraudes, se obtiene un listado ordenado de mayor a menor importancia.

Figure 8: 10 Variables más importantes.

5.3.2 Entrenamiento y evaluación

Tras la selección de las 10 variables más importantes, se entrena un nuevo modelo de *random forest* utilizando solo estas. El conjunto de datos es distribuido en dos subconjuntos, entrenamiento (70%) y prueba (30%), permitiéndole al modelo poder aprender patrones y detectar anomalías en el entrenamiento, y comprobar su capacidad predictiva al aplicar lo aprendido en nuevos datos que

no han sido vistos.

Para conocer que tan bien el modeo ha aprendido en la fase de entrenamiento, grafico una matriz de confusión, tal y como se ve en la imagen 9.

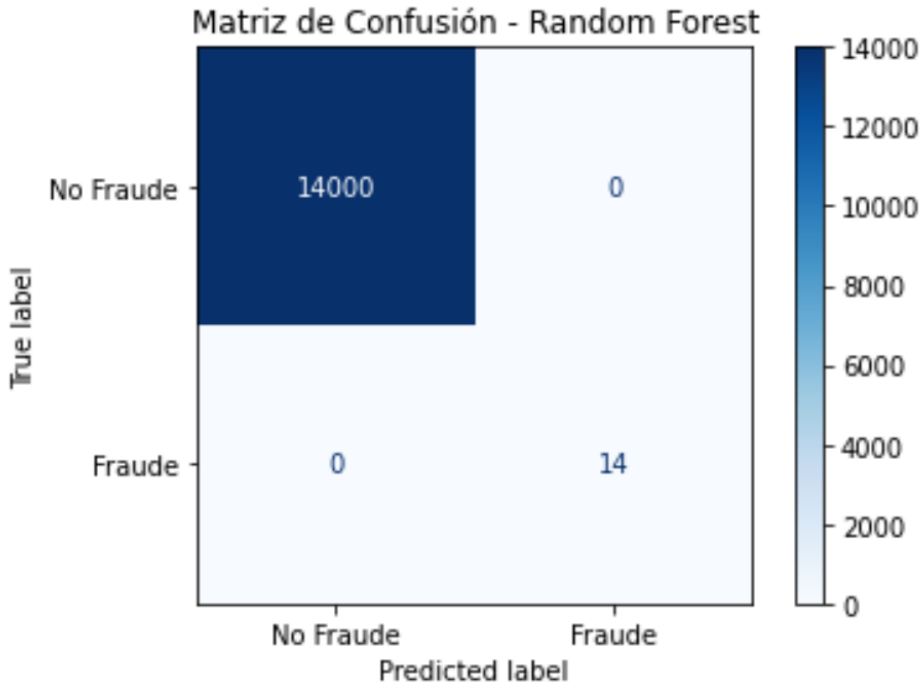

Figure 9: Matriz de confusión para el conjunto de entrenamiento del modelo *random forest*

A su vez también genero un reporte de clasificación, obteniendo unas métricas perfectas:

- **precisión:** 1.00 (para ambas clases),
- ***recall*:** 1.00 (para ambas clases),
- ***f1-score*:** 1.00 (para ambas clases),
- ***accuracy*:** 1.00.

Para obtener información sobre la capacidad predictiva del modelo, genero otra matriz de confusión para el conjunto de prueba, la cual se representa en la imagen 10.

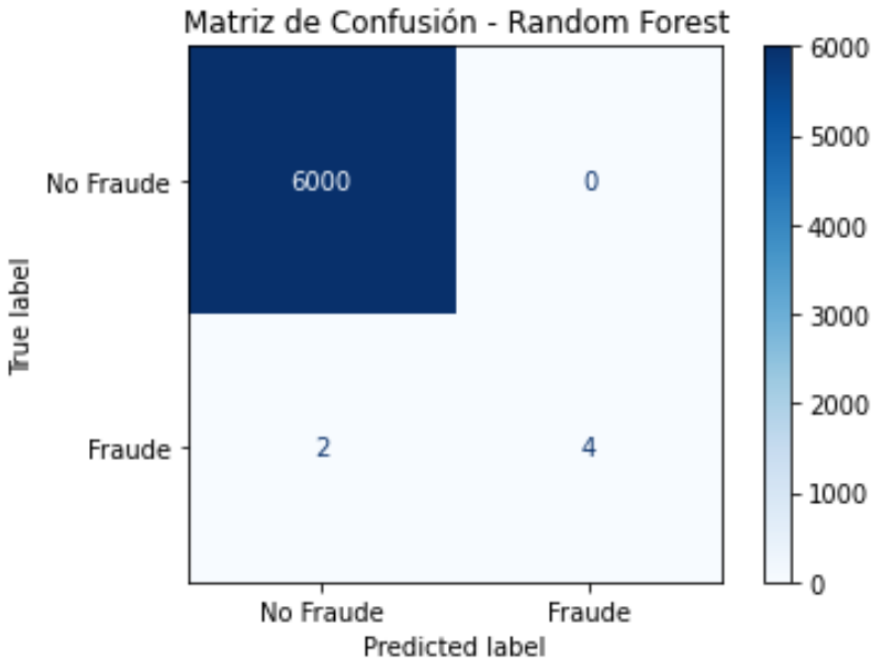

Figure 10: Matriz de confusión para el conjunto de prueba del modelo *random forest*

Y muestro sus métricas asociadas, las cuales son:

Clase	**Precisión**	**Recall**	**F1-Score**
Clase 0 (no fraude)	0.9997	1.00	0.9998
Clase 1 (fraude)	1.00	0.6667	0.8000

Table 2: Métricas de desempeño por clase

- **Accuracy:** 0.9997

Estos resultados muestran que, en un conjunto de datos reducido, el algoritmo *Random Forest* ofrece un excelente rendimiento al clasificar transacciones no fraudulentas. Sin embargo, aunque también identifica correctamente muchas transacciones fraudulentas, su capacidad predictiva en esta clase es inferior, lo que sugiere una menor sensibilidad frente a casos de fraude.

Una vez observada la capcidad predictiva del modelo entrenado con *random forest* para un conjunto pequeño de datos, compruebo si es capaz de clasificar correctamente cuando se realiza el mismo entrenamiento pero para un conjunto de datos mayor, en este caso para las 6 millones de

filas que contiene el original.

El proceso de entrenamiento y prueba es exactamente el mismo que el anterior, se seleccionan como variables independientes las seleccionadas, y como variable objetivo *isFraud*. Se realiza la misma división del conjunto de datos (70% entrenamiento y 30% prueba).

Para el conjunto de entrenamiento se generan la siguiente matriz de confusión, mostrada en la imagen 11:

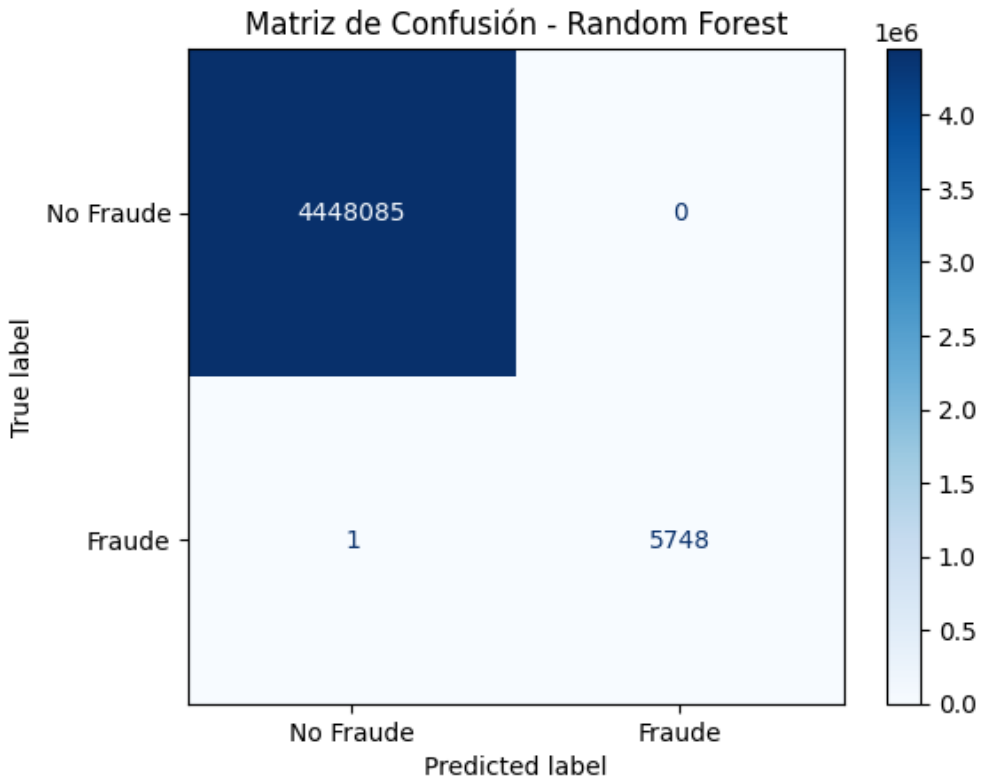

Figure 11: Matriz de confusión del conjunto de entrenamiento,

y las métricas para ambas clases son:

Clase	**Precisión**	**Recall**	**F1-Score**
Clase 0 (no fraude)	0.9999	1.00	0.9999
Clase 1 (fraude)	1.00	0.9998	0.9999

Table 3: Métricas de desempeño por clase

- **Accuracy:** 0.9999

Observamos unos resultados prácticamente perfectos, donde solo 1 fraude no ha sido clasificado como tal.

Finalmente, para comprobar si realmente el modelo entrenado con *random forest* tiene una buena capacidad predictiva, se muestra una matriz de confusión (imagen 12):

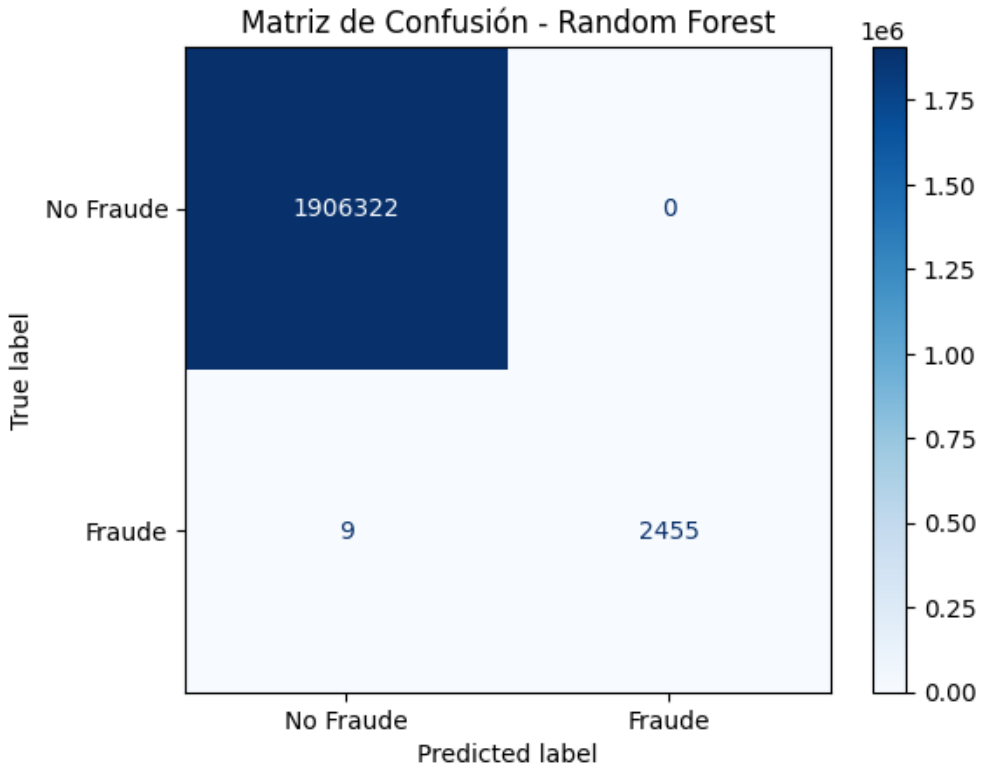

Figure 12: Matriz de confusión del conjunto de prueba.

Junto con esta matriz, se ha calculado también el reporte de clasificación, mostrando las siguientes métricas:

Clase	**Precisión**	**Recall**	**F1-Score**
Clase 0 (no fraude)	0.9999	1.00	0.9999
Clase 1 (fraude)	1.00	0.9963	0.9981

Table 4: Métricas de desempeño por clase

- **Accuracy:** 0.9999

Al analizar los resultados obtenidos, se concluye que el algoritmo Random Forest ofrece un rendimiento predictivo sobresaliente en conjuntos de datos amplios. No obstante, al aplicarse

sobre volúmenes de datos reducidos, su desempeño disminuye de forma notable, especialmente en la detección de transacciones fraudulentas, que constituyen el foco principal de este estudio.

5.4. Metodología 2: Red Neuronal Artificial

5.4.1 Selección de variables

En esta metodología, el proceso de selección de variables utilizado es de caracter estadístico y exploratorio. Se realiza un análisis basado en la diferencia de correlaciones. Esta opción permite conocer que variables presentan relaciones considerablemente distintas con el resto de las variables en función del valor de la variable isFraud, es decir, se puede observar cómo varían las correlaciones entre variables para transacciones legítimas y fraudulentas.

El primer paso es reducir la dimensionalidad del conjunto de datos. Para eso se toma como muestra, al igual que en *random forest*, 20000 transacciones legítimas y 20 fraudulentas, facilitando así el entrenamiento de la red.
Después se divide el conjunto de datos en dos subconjuntos, uno en el que las transacciones sean solamente fraudulentas (isFraud = 1) y otro donde sean solamente legítimas (isFraud = 0). Al igual que se hizo anteriormente una matriz de correlación con todo el conjunto de datos, ahora se generan dos matrices, una representativa de cada subconjunto. Estas matrices permiten observar de manera rápida y sencilla la relación lineal entre pares de variables.

Una vez ambas matrices han sido generadas, restando los valores de correlación de las transicciones legítimas a las fraudulenmtas, se obtiene una matriz de diferencia de correlaciones, la cual se muestra en la imagen 13, y al igual que feature_importances_ en Random Forest, es realmente útil para la reducción de dimensionalidad y selección de variables, ya que permite ver de manera clara cuales son las relaciones de variables que más varían dependiendo de que tipo de transacción

se trate.

Para cada variable, se calcula la suma del valor absoluto de sus diferencias de correlación con respecto a todas las demás. Esta métrica refleja el grado total de cambio en el comportamiento de una variable y finalmente se seleccionan las 10 más importantes, las cuales se muestran en la imagen 14.

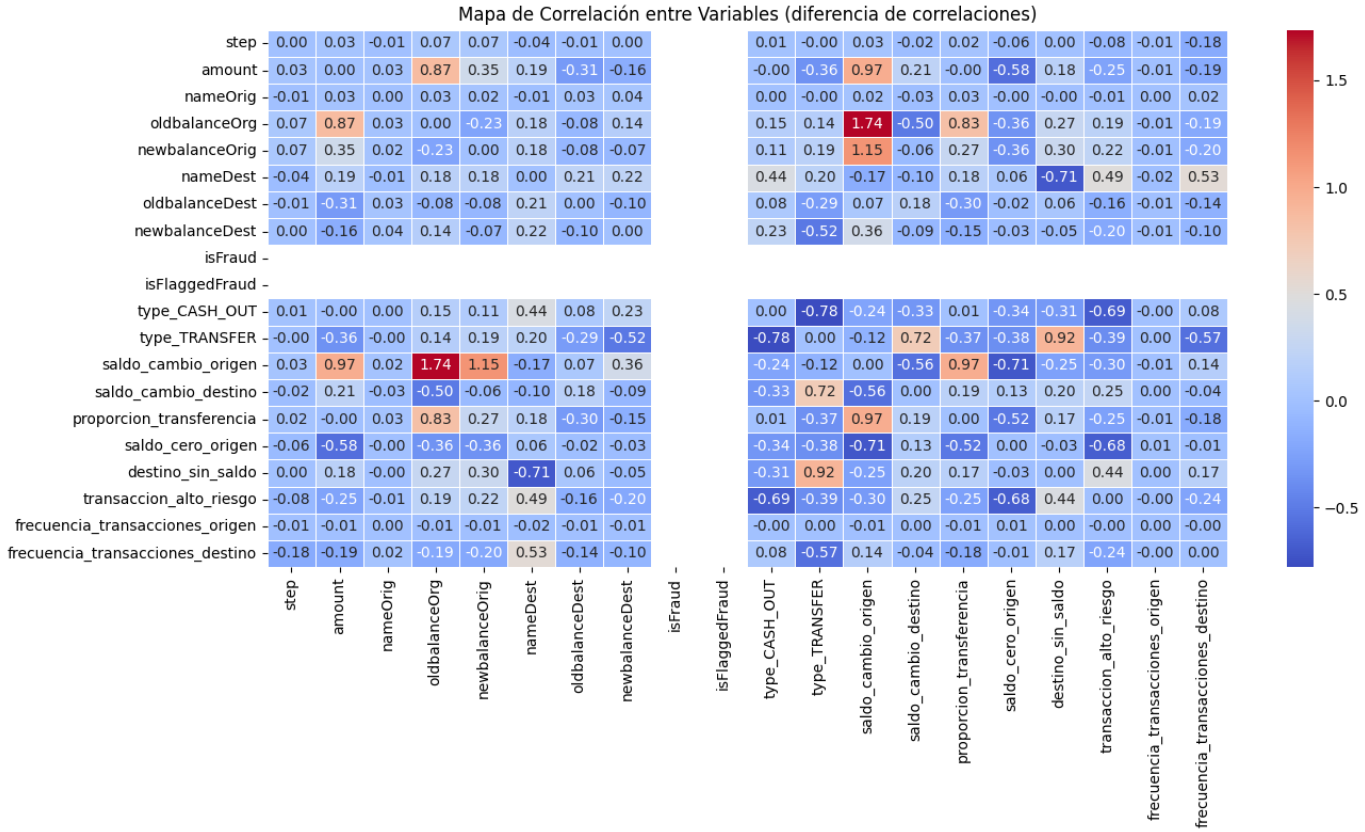

Figure 13: Mapa de diferencia de correlaciones.

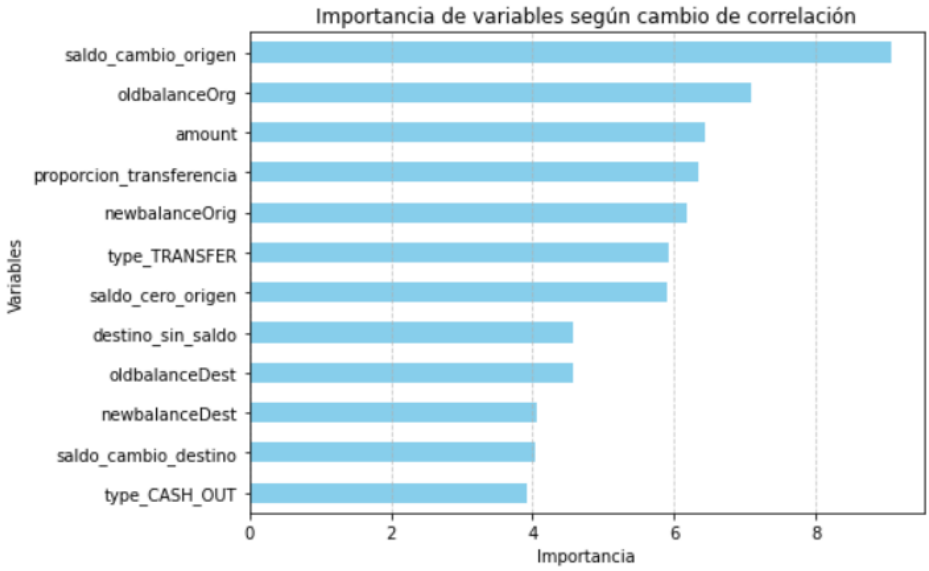

Figure 14: Importancia de las variables según la diferencia de correlaciones.

5.4.2 Entrenamiento del modelo

Tras la selección de características más relevantes, se procede a la etapa de entrenamiento del modelo mediante el uso de una red neuronal.

A partir del conjunto de datos reducido, se toma como variables independientes las 10 extraidas de la selección de características (correlación), y como variable objetivo isFraud. Se procede entonces a la división del conjunto de datos, destinando un 70% de estos al entrenamiento y un 30% a prueba, permitiendo esta distribución entrenar el modelo de manera eficiente y obtener una evaluación final válida.

Después de realizar la división del conjunto de datos, pasamos a la definición de la red neuronal, la cual será utilizada para predecir los fraudes. Esta red neuronal consta de 4 capas ocultas con tamaños decrecientes (128, 64, 32 y 16 neuronas), todas utilizando la función de activación ReLU y con un *Dropout* del 20%, con el fin de evitar el sobreajuste. La capa de salida contiene una sola neurona debido a que el problema que se busca resolver en este trabajo es de clasificación binaria (fraude 1, no fraude 0) y se usa la función de activación sigmoid ya que se desea obtener

una probabilidad comprendida entre 0 y 1.

Una vez se ha definido el modelo el modelo, en este caso la red neuronal, se compila utilizando el optimizador Adam con una tasa de aprendizaje de 0.001. Debido a que el problema que se resuelve en este trabajo es de clasificación binaria, la función de pérdida seleccionada fue binary crossentropy, y además, se especificó el cálculo de la métrica accuracy para evaluar el rendimiento del modelo durante la fase de entrenamiento, al igual que se definció la técnica de *early_stop* con una paciencia de 3, es decir, que si el valor de val_loss no mejora pasadas 3 épocas, se detiene el entrenamiento, evitando obtener un sobreajuste.

Al haber definido la aquitectura de la red, se procede al verdadero entrenamiento del modelo. Durante este proceso, se observan métricas de rendimiento, cuyos valores eran realmente positivos desde las primeras épocas.

Durante la primera época, el modelo alcanzó una accuracy del 99.70% sobre los datos de entrenamiento, mientras que en el subconjunto de validación automático generado por el propio modelo se obtuvo una precisión del 99.86%, con funciones de pérdida respectivas de 3.64% y 0.96%. Estos resultados iniciales indican que el modelo fue capaz de aprender patrones relevantes desde las primeras fases del entrenamiento.
A lo largo de las siguientes épocas, las métricas se mantuvieron en valores muy altos y estables. Por ejemplo, en la segunda época, la pérdida de entrenamiento se redujo al 0.51%, y la exactitud alcanzó el 99.91%, mientras que la validación se mantuvo constante en torno al 99.86%. El valor más bajo de pérdida sobre el conjunto de validación se registró en la época 6, con un 0.70%.

5.4.3 Evaluación del modelo

Una vez finaliza el entrenamiento de la red neuronal, el siguiente paso es comprobar que tan bien el modelo es capaz de predecir que transacciones son fraudulentas y cuales no.

Para ello, se evalua el rendimiento del modelo sobre el mismo conjunto de datos utilizado durante el entrenamiento. Comprobando así si ha aprendido adecuadamente los patrones existentes en los datos.

Se realizan predicciones con el modelo entenado, de manera que para cada transacción se obtiene una probabilidad de que esta sea fraudulenta. El modelo, en lugar de devolver una clasificación binaria (fraude o no fraude), proprociona probabilidades que se encuentran comprendidas entre 0 y 1. Para poder interpretar estas probabilidades como una decisión final, es necesaria la definición de un umbral de clasificación para así transformarlas a etiquetas binarias.
En este caso, se establece un umbral de clasificación de 0.7, lo cual significa que una transacción solo es clasificada como fraudulenta si la probabilidad supera dicho valor, de lo contrario es considerada como legítima. Esta selección de umbral se debe a que, al ser un conjunto de datos que el modelo ya ha visto, es razonable ser más estricto al clasificar una transacción como fraudulenta, exigiendo un mayor nivel de confianza.

Una vez aplicadas dichas predicciones, para evaluar el rendimiento del modelo sobre el conjunto de entrenamiento, se utiliza una matriz de confusión. Esta permite ver de manera clara cuantas transacciones, tanto fraudulentas como legítimas, han sido correcta o incorrectamente clasificadas. Esta matriz se observa en la imagen 15.

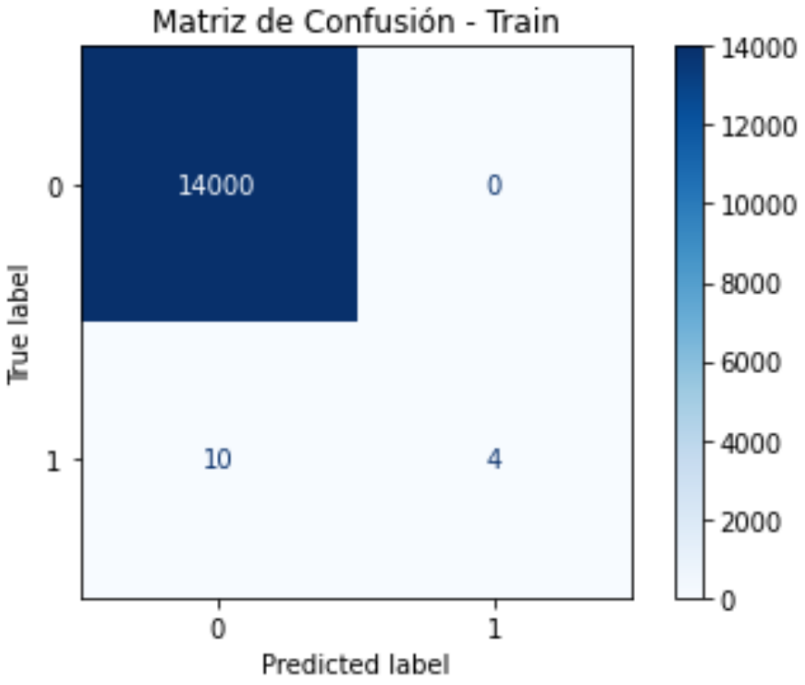

Figure 15: Matriz de confusión sobre el conjunto de entrenamiento.

A esta matriz se le acompaña un reporte de clasificación, el cual incluye las siguientes métricas:

Clase	**Precisión**	**Recall**	**F1-Score**
Clase 0 (No fraude)	0.9993	1.000	0.9996
Clase 1 (Fraude)	1.0000	0.2857	0.4444

Table 5: Métricas de rendimiento del modelo sobre el conjunto de prueba

- **Accuracy:** 0.9993

Para obtener información sobre la capacidad predictiva del modelo, genero una nueva matriz de confusión para el conjunto de prueba, la cual se representa en la imagen 16

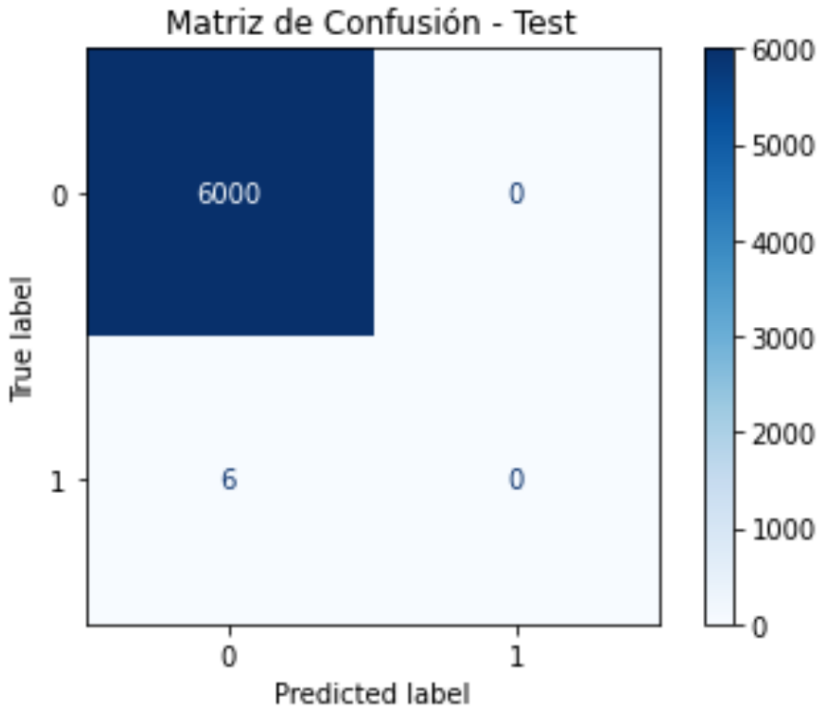

Figure 16: Matriz de confusión para el conjunto de prueba.

Cuyas métricas son:

Clase	Precisión	Recall	F1-Score
Clase 0 (No fraude)	0.999	1.000	0.9995
Clase 1 (Fraude)	0.0000	0.0000	0.0000

Table 6: Métricas de rendimiento del modelo sobre el conjunto de prueba

- **Accuracy:** 0.9990

Aunque el modelo alcanza una *accuracy* del 99.90%, las métricas por clase revelan un fuerte desbalance en su rendimiento. La clasificación de las transacciones no fraudulentas (clase 0) es prácticamente perfecta, con una precisión del 99.9% y un recall del 100%. Sin embargo, el modelo no detecta ningún caso de fraude (clase 1), obteniendo valores nulos en precisión, recall y F1-score. Este resultado refleja que el modelo ha aprendido a predecir únicamente la clase mayoritaria, ignorando por completo la minoritaria. En contextos como la detección de fraude, donde los datos están fuertemente desbalanceados, la *accuracy* resulta engañosa.

Después de observar la capacidad predictiva de la red neuronal apra el conjunto de datos reducido, se realiza el mismo proceso de entrenamiento pero para el conjunto de datos original, el cual contiene más de 6 millones de filas.

Se mantienen las mismas proporciones en la división del conjunto de datos, utilizando un 70% para entrenamiento y un 30% para prueba. No obstante, se modifica la arquitectura de la red neuronal, añadiendo una capa oculta adicional. La nueva estructura cuenta con cinco capas ocultas compuestas por 256, 128, 64, 32 y 16 neuronas, respectivamente. Esta ampliación se debe a que, al trabajar con un conjunto de datos más extenso, es necesario dotar al modelo de una mayor capacidad representacional para que pueda aprender de manera eficaz los patrones subyacentes en los datos.

Para el conjunto de entrenamiento se genera la matriz de confusión mostrada en la imagen 17

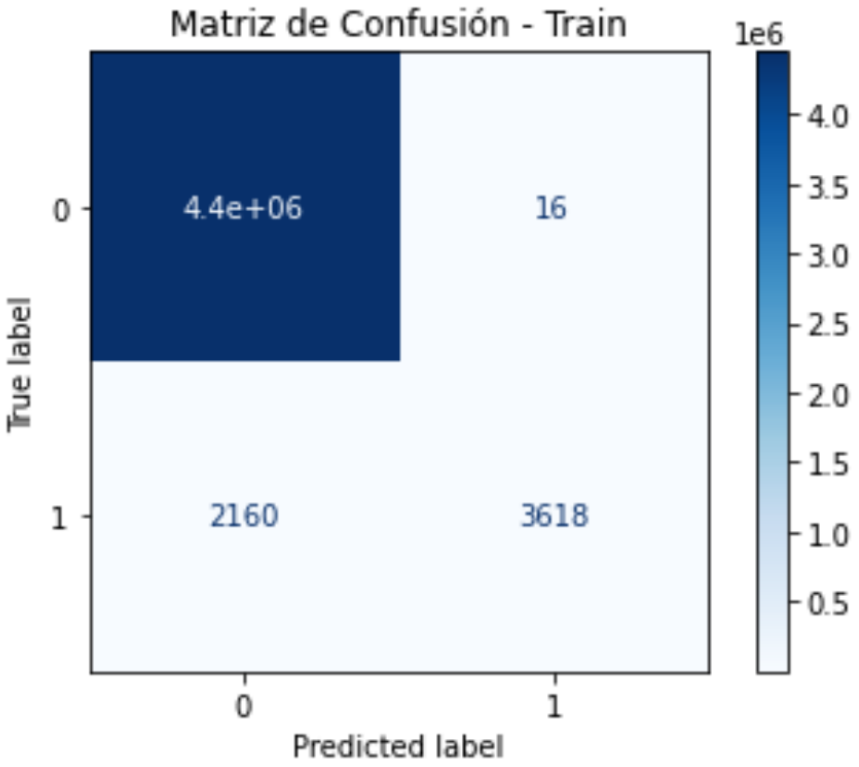

Figure 17: Matriz de confusión para el conjunto de entrenamiento.

Cuyas métricas son:

Clase	Precisión	Recall	F1-Score
Clase 0 (No fraude)	0.9995	1.000	0.9998
Clase 1 (Fraude)	0.9956	0.6262	0.7688

Table 7: Métricas de rendimiento del modelo sobre el conjunto de prueba.

- **Accuracy:** 0.9995

Para conocer la capacidad predictiva del modelo, genero una matriz de confusión para el conjunto de prueba, mostrada en la imagen 18

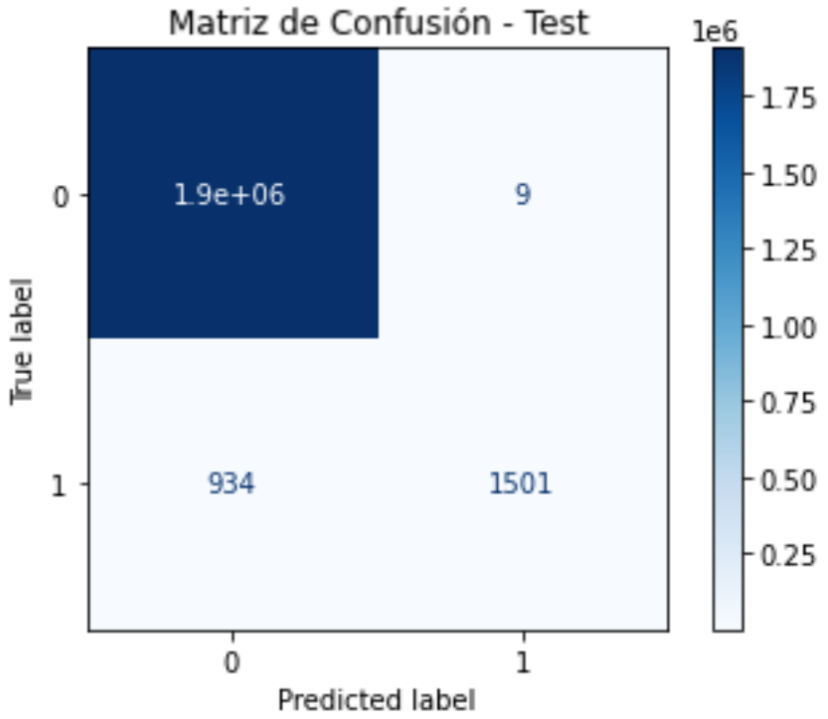

Figure 18: Matriz de confusión para el conjunto de prueba.

Se han calculado también las siguietnes métricas:

Clase	Precisión	Recall	F1-Score
Clase 0 (No fraude)	0.9995	1.000	0.9998
Clase 1 (Fraude)	0.9994	0.6164	0.7610

Table 8: Métricas de rendimiento del modelo sobre el conjunto de prueba.

- **Accuracy:** 0.9995

Los resultados de las métricas muestran un rendimiento muy elevado del modelo al clasificar transacciones en el conjunto de prueba. La accuracy global es del 99.95%, lo que indica que el modelo acierta en la gran mayoría de las predicciones. Sin embargo, es importante analizar las métricas por clase para evaluar su comportamiento de forma más detallada, especialmente en un problema desbalanceado como la detección de fraude.

Para la clase 0 (no fraude), el rendimiento es prácticamente perfecto: la precisión alcanza un 99.95% y el recall es del 100%, lo que significa que el modelo identifica correctamente la gran mayoría de las transacciones legítimas.

En cuanto a la clase 1 (fraude), el modelo también presenta una precisión muy alta (99.94%), lo que indica que casi todas las transacciones que predijo como fraude realmente lo eran. Sin embargo, el recall en esta clase es sensiblemente más bajo, con un valor del 61.64%. Esto implica que el modelo solo detectó aproximadamente el 62% de los fraudes reales, dejando sin identificar el resto.

En resumen, aunque el modelo es capaz de detectar fraudes cuando lo hace, aún se le escapan una parte considerable de los casos fraudulentos, lo cual es relevante en contextos donde la prioridad es minimizar los falsos negativos. Esto sugiere que aún hay margen de mejora, especialmente en la sensibilidad hacia la clase minoritaria.

5.5. Comparación de ambas metodologías

Tras haber evaluado ambas metodologías, es posible realizar una comparación de estas basada en distintos factores que han sido observados, como la capacidad predictiva, la eficiendia y la complejidad de los dos modelos.

En cuanto a la capacidad predictiva, al analizar las figuras 12 y 18, se observa que el modelo entrenado con Random Forest ofrece un rendimiento significativamente superior. En concreto, genera un número notablemente menor de falsos positivos y falsos negativos. Este modelo logra clasificar correctamente la totalidad de las transacciones legítimas, y únicamente 9 transacciones

fraudulentas no son identificadas como tales.

Por el contrario, aunque la red neuronal comete pocos errores en la clasificación de transacciones legítimas (solo 9 casos incorrectos), su rendimiento en la detección de fraudes es considerablemente inferior, ya que 934 de las 2435 transacciones fraudulentas son clasificadas erróneamente.

Esta diferencia también se refleja en las métricas de rendimiento: los valores obtenidos para ambas clases en la tabla 4,correspondientes al modelo Random Forest, son claramente superiores a los presentados en la tabla 8, que corresponden a la red neuronal.

n términos de complejidad, la arquitectura de la red neuronal resulta significativamente más compleja que la del modelo *random forest*. Este último está compuesto por múltiples árboles de decisión, cada uno de los cuales es relativamente simple y fácilmente interpretable de forma individual, ya que opera sobre subconjuntos específicos de los datos. Esto permite examinar de manera más clara las decisiones tomadas por cada árbol y comprender su lógica interna.

Por el contrario, la red neuronal cuenta con un número considerable de parámetros que se ajustan durante el proceso de entrenamiento. Aunque esto dificulta su interpretación, dichos parámetros son fundamentales, ya que permiten a la red aprender relaciones no lineales entre las variables. Además, su estructura está formada por una combinación de múltiples capas ocultas, cuyo número y configuración son definidos tras diversas pruebas y análisis de los resultados obtenidos. Esta complejidad estructural, aunque necesaria para alcanzar un buen rendimiento, representa un desafío adicional en términos de interpretabilidad.

Respecto a la eficiencia, es decir, el tiempo de entrenamiento y el consumo de recursos computacionales, se observan grandes diferencias.

Random forest demuestra ser consideradamente más rápido durante el entrenamiento, lo cual se debe a que la construcción de árboles de decisión individuales puede realizarse de forma paralela, lo que acelera significativamente el proceso de entrenamiento, requiriendo por tanto un tiempo menor, concretamente 37 minutos para el conjunto de entrenamiento y 8 minutos para el conjunto

de prueba. Además, al ser un algoritmo que no requiere un ajuste de hiperparámetros, contribuye a mantener bajo el uso de memoria y procesamiento.

Por el contrario, la red neuronal implicó un coste computacional mucho mayor. Aunque el número de épocas necesarias para alcanzar un rendimiento óptimo fue reducido gracias a la implementación del mecanismo de *early stopping*, el tiempo por época fue significativamente más alto. En total, el entrenamiento de la red neuronal tomó 211 minutos, y su evaluación sobre el conjunto de prueba 57 minutos, superando ampliamente los tiempos registrados por *random forest*. Esta diferencia se explica por la mayor complejidad del modelo: como se ha mencionado previamente, la red neuronal cuenta con un elevado número de capas ocultas y neuronas, lo que implica el ajuste de numerosos parámetros durante el entrenamiento. Para lograr un rendimiento óptimo, es necesario calcular gradientes y realizar actualizaciones de pesos mediante backpropagation, lo que incrementa notablemente el tiempo de cómputo. Además, al almacenar los estados intermedios durante el proceso de entrenamiento, el modelo requiere más memoria. Esta demanda computacional se ve aún más acentuada en el contexto de este trabajo, al haberse utilizado un ordenador portátil sin GPU ni otro tipo de acelerador integrado, lo que limita significativamente la velocidad de procesamiento.

6. Conclusiones

A lo largo de este trabajo se ha llevado a cabo un estudio sobre la aplicación de algoritmos de *machine learning* en la detección de fraudes financieros, con el objetivo de evaluar su efectividad ante grandes volúmenes de datos. Para ello, se ha seguido un proceso completo que ha abarcado desde el análisis exploratorio hasta la implementación de dos modelos predictivos: *random forest* y redes neuronales artificiales. Esto ha permitido comparar su rendimiento, ventajas y limitaciones frente a un problema real de clasificación con clases desbalanceadas.

Ambos modelos han sido capaces de aprender patrones a partir de un conjunto de datos compuesto por más de seis millones de transacciones, logrando identificar correctamente una gran proporción de ellas. Los resultados muestran que la inteligencia artificial no solo es aplicable a contextos reales y complejos como el fraude financiero, sino que permite alcanzar niveles de precisión muy superiores a los que se podrían lograr de forma manual.

No obstante, el análisis comparativo entre ambos enfoques pone de manifiesto diferencias importantes. El modelo basado en *random forest* ha demostrado ser el más eficaz, alcanzando una tasa de falsos positivos y falsos negativos extremadamente baja, junto con un tiempo de entrenamiento notablemente inferior. Su estructura interpretativa, su bajo coste computacional y su excelente rendimiento general lo convierten en la opción más adecuada para el caso de estudio.

En contraste, el modelo basado en redes neuronales ha ofrecido un rendimiento inferior, especialmente en la detección de transacciones fraudulentas. Aunque es capaz detectar una parte considerable de los fraudes, deja sin clasificar correctamente una proporción significativa de ellos.

En conclusión, este estudio demuestra que no existe un algoritmo universalmente óptimo, y que la elección del modelo debe basarse en los recursos disponibles, las características del problema y los objetivos específicos. Para el caso particular de este trabajo, donde se busca maximizar la detección de fraudes con eficiencia y fiabilidad, *random forest* se presenta como la solución más

adecuada. No obstante, el uso de redes neuronales sigue siendo una alternativa válida, especialmente en escenarios donde se pueda disponer de mayor capacidad computacional y se requiera capturar patrones más complejos.

Finalmente, dada la naturaleza dinámica y creciente del fraude financiero, se refuerza la necesidad de seguir desarrollando sistemas automatizados e inteligentes capaces de adaptarse y detectar nuevas formas de fraude con rapidez y precisión. El uso de algoritmos de aprendizaje automático se perfila como un componente esencial para afrontar este desafío de forma eficiente y escalable.

6.1. Trabajo futuro

En futuras líneas de trabajo, me gustaría profundizar en el uso de redes neuronales, especialmente explorando su comportamiento con una configuración computacional más adecuada, como el uso de un equipo con GPU o aceleradores especializados, que permita reducir significativamente los tiempos de entrenamiento y mejorar la eficiencia del proceso. Esto facilitaría el ajuste de arquitecturas más complejas y la realización de más pruebas sin los condicionantes del tiempo de cómputo.

Asimismo, contar con mayores recursos permitiría realizar una optimización más precisa de los umbrales de decisión en la fase de predicción. Esta optimización podría mejorar notablemente el rendimiento del modelo, adaptándolo mejor a los objetivos específicos del problema, como la minimización de falsos negativos en la detección de fraudes.

En resumen, una futura línea de investigación consistiría en explorar el verdadero potencial de las redes neuronales bajo condiciones óptimas, combinando una mayor capacidad de procesamiento con técnicas de ajuste fino que permitan obtener un modelo competitivo, eficiente y ajustado a las necesidades reales de detección de fraude.

7. Bibliografía

[1] *Árbol de decisiones*. (n.d.). Xunta de Galicia. https://espazoabalar.edu.xunta.gal/es/agora-dixital/lab/ia/arbol-de-decisiones

[2] Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[3] Brital, A. (2021). *Random forest algorithm explained*. https://anasbrital98.github.io/blog/2021/Random-Forest/

[4] Conekta. (2023). Fraudes por internet: Qué son y cómo prevenirlos. http://conekta.com/blog/fraudes-por-internet

[5] Experian, D. (2024). El auge del fraude en línea: Nuevas modalidades. *DataCrédito Experian*. https://www.datacredito.com.co/blogs/datablog/el-auge-del-fraude-en-linea-nuevas-modalidades/

[6] JainilCoder. (2023). *Online payment fraud detection dataset*. Kaggle. https://www.kaggle.com/datasets/jainilcoder/online-payment-fraud-detection

[7] Lab, F. (2019). *Introducción a las redes neuronales pt. i*. https://futurelab.mx/redes%20neuronales/inteligencia%20artificial/2019/06/25/intro-a-redes-neuronales-pt-1/

[8] Matich, J. (2001). *Redes neuronales artificiales*. Universidad Tecnológica Nacional (UTN), Facultad Regional Rosario. https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monograias/matich-redesneuronales.pdf

[9] Moreno, A., Armengol, E., Béjar, J., Belanche, L., Cortés, U., Gavaldà, R., Gimeno, J. M., López, B., Martín, M., & Sànchez, M. (1994). *Aprendizaje automático*. Edicions de la Universitat Politècnica de Catalunya. https://upcommons.upc.edu/bitstream/handle/2099.3/36157/9788483019962.pdf?sequence=1

[10] Pedraza Caro, J. D. (2023). La inteligencia artificial en la sociedad: Explorando su impacto actual y los desafíos futuros. https://oa.upm.es/75068/1/TFG_JAROD_DAVID_PEDRAZA_CARO.pdf

[11] SEON. (n.d.). Machine learning para detectar fraude: Qué es y cómo funciona. https://seon.io/es/recursos/machine-learning-para-detectar-fraude/

[12] Soni, P. (2024). *Roc-auc analysis – a deep dive*. https://www.blog.trainindata.com/auc-roc-analysis/

[13] UNIR. (2023). Tipos de fraudes en internet más comunes y cómo evitarlos. https://www.unir.net/revista/derecho/fraudes-internet/

8. Anexo

8.1. Anexo 1: Código Random Forest

```
#LIBRERIAS
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,
    confusion_matrix, ConfusionMatrixDisplay
from sklearn.utils import class_weight
from sklearn.metrics import precision_score, recall_score, f1_score,
    accuracy_score


#RANDOM FOREST


#LEER CSV
data_rf = pd.read_csv(r"C:\Users\Ines\OneDrive - SINEN Solucions Informatiques
    , SL\Escritorio\TFG\onlinefraud.csv")
```

```
data_rf.head(10)


#ANALISIS EXPLORATORIO DE DATOS


# Dimension del dataset
print(f"Numero total de transacciones: {data_rf.shape[0]}")
print(f"Numero de columnas: {data_rf.shape[1]}")


# Mostrar cantidad y proporcion de fraudes
print(data_rf['isFraud'].value_counts())
fraud_percentage = data_rf['isFraud'].mean() * 100
print(f"Porcentaje de fraudes: {fraud_percentage:.4f}%")


# Graficar
sns.countplot(x=data_rf["isFraud"])
plt.title("Distribucion de fraudes")
plt.show()


#PREPROCESAMIENTO DE DATOS


data_rf = pd.get_dummies(data_rf, columns=["type"], drop_first=True)


# Pasar las columnas numericas enteras a float
data_rf["step"] = data_rf["step"].astype(float)
data_rf["isFraud"] = data_rf["isFraud"].astype(float)
data_rf["isFlaggedFraud"] = data_rf["isFlaggedFraud"].astype(float)
data_rf["type_CASH_OUT"] = data_rf["type_CASH_OUT"].astype(float)
data_rf["type_DEBIT"] = data_rf["type_DEBIT"].astype(float)
data_rf["type_PAYMENT"] = data_rf["type_PAYMENT"].astype(float)
data_rf["type_TRANSFER"] = data_rf["type_TRANSFER"].astype(float)


# Pasar las columnas string a numericas
le = LabelEncoder()
```

```
data_rf["nameOrig"] = data_rf["nameOrig"].fillna("unknown").astype(str)
data_rf["nameOrig"] = le.fit_transform(data_rf["nameOrig"])

data_rf["nameDest"] = data_rf["nameDest"].fillna("unknown").astype(str)
data_rf["nameDest"] = le.fit_transform(data_rf["nameDest"])

#Creacion de nuevas variables
data_rf["saldo_cambio_origen"] = data_rf["oldbalanceOrg"]- data_rf["
    newbalanceOrig"]
data_rf["saldo_cambio_destino"] = data_rf["oldbalanceDest"]- data_rf["
    newbalanceDest"]
data_rf["proporcion_transferencia"] = data_rf["amount"] / (data_rf["
    oldbalanceOrg"] + 1) # +1 para evitar division por 0
data_rf["saldo_cero_origen"] = (data_rf["newbalanceOrig"] == 0).astype(int)
data_rf["destino_sin_saldo"] = (data_rf["oldbalanceDest"] == 0).astype(int)
data_rf["transaccion_alto_riesgo"] = ((data_rf["type_CASH_OUT"] == 1) |(
    data_rf["type_TRANSFER"] == 1)) & (data_rf["amount"] > data_rf["
    oldbalanceOrg"] * 0.9)
data_rf["transaccion_alto_riesgo"] = data_rf["transaccion_alto_riesgo"].astype
    (int)
data_rf["frecuencia_transacciones_origen"] = data_rf.groupby("nameOrig")["
    nameOrig"].transform("count")
data_rf["frecuencia_transacciones_destino"] = data_rf.groupby("nameDest")["
    nameDest"].transform("count")

data_rf = data_rf.drop(columns=['type_PAYMENT', 'type_DEBIT'])

#SELECCION DE VARIABLES

# Dataset reducido
fraudes = data_rf[data_rf['isFraud'] == 1].sample(n=20, random_state=42)
no_fraudes = data_rf[data_rf['isFraud'] == 0].sample(n=20000, random_state=42)
```

```
reducido_rf = pd.concat([fraudes, no_fraudes]).sample(frac=1, random_state=42)

# Seleccionar variables independientes y variable objetivo
X = reducido_rf.drop(columns=['isFraud', 'step', 'nameOrig', 'nameDest'])
y = reducido_rf['isFraud']

print(y.unique())
y.fillna(0, inplace=True)

# Entrenar un modelo de Random Forest
rf = RandomForestClassifier(n_estimators=40, random_state=42)
rf.fit(X, y)
# Obtener importancia de caracteristicas
importances = pd.Series(rf.feature_importances_, index=X.columns)
importances.sort_values(ascending=False, inplace=True)
importances

# Seleccion
top_10 = importances.sort_values(ascending=False).head(10).sort_values(
    ascending=True)

# Grafico de barras
plt.figure(figsize=(10, 5))
top_10.plot(kind='barh')
plt.title("Importancia de las 10 Variables Mas Relevantes segun Random Forest"
    )
plt.ylabel("Variables")
plt.xlabel("Importancia")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

#ENTRENAMIENTO Y PREDICCION
```

```
X1 = reducido_rf.drop(columns=['isFraud', 'step', 'nameOrig', 'nameDest'])
y1 = reducido_rf['isFraud']

X1_train_rf, X1_test_rf, y1_train_rf, y1_test_rf = train_test_split(X1, y1,
    test_size=0.3, random_state=42, stratify=y1)

rf = RandomForestClassifier(n_estimators=40, random_state=42)
rf.fit(X1_train_rf, y1_train_rf)

# Predicciones sobre el conjunto de entrenamiento
y1_pred_train_rf = rf.predict(X1_train_rf)

# Matriz de confusion
cm_rf = confusion_matrix(y1_train_rf, y1_pred_train_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=["No
    Fraude", "Fraude"])
disp_rf.plot(cmap="Blues", values_format='d')
plt.title("Matriz de Confusion - Random Forest")
plt.show()

# Reporte de clasificacion
print("Reporte de clasificacion - Random Forest:")
print(classification_report(y1_train_rf, y1_pred_train_rf, digits=4))

# Predicciones sobre el conjunto de prueba
y1_pred_test_rf = rf.predict(X1_test_rf)

cm_rf = confusion_matrix(y1_test_rf, y1_pred_test_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=["No
    Fraude", "Fraude"])
disp_rf.plot(cmap="Blues", values_format='d')
plt.title("Matriz de Confusion - Random Forest")
```

```
plt.show()

print("Reporte de clasificacion - Random Forest:")
print(classification_report(y1_test_rf, y1_pred_test_rf, digits=4))

#ENTRENAMIENTO Y PREDICCION CON VARIABLES SELECCIOANDAS

top_vars = importances.sort_values(ascending=False).head(10).index.tolist()

X2 = reducido_rf[top_vars]
y2 = reducido_rf['isFraud']

X2_train_rf, X2_test_rf, y2_train_rf, y2_test_rf = train_test_split(X2, y2,
    test_size=0.3, random_state=42, stratify=y2)

rf = RandomForestClassifier(n_estimators=40, random_state=42)
rf.fit(X2_train_rf, y2_train_rf)

# Predicciones sobre el conjunto de entrenamiento
y2_pred_train_rf = rf.predict(X2_train_rf)

# Matriz de confusion
cm_rf = confusion_matrix(y2_train_rf, y2_pred_train_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=["No
    Fraude", "Fraude"])
disp_rf.plot(cmap="Blues", values_format='d')
plt.title("Matriz de Confusion - Random Forest")
plt.show()

# Reporte de clasificacion
print("Reporte de clasificacion - Random Forest:")
print(classification_report(y2_train_rf, y2_pred_train_rf, digits=4))
```

```
# Predicciones sobre el conjunto de prueba
y2_pred_test_rf = rf.predict(X2_test_rf)

cm_rf = confusion_matrix(y2_test_rf, y2_pred_test_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=["No
    Fraude", "Fraude"])
disp_rf.plot(cmap="Blues", values_format='d')
plt.title("Matriz de Confusion - Random Forest")
plt.show()

print("Reporte de clasificacion - Random Forest:")
print(classification_report(y2_test_rf, y2_pred_test_rf, digits=4))

#ENTRENAMIENTO Y PREDICCION DATASET ORIGINAL

top_vars = importances.sort_values(ascending=False).head(10).index.tolist()
X5 = data_rf[top_vars]
y5 = data_rf['isFraud']

# Division correcta
X5_train_rf, X5_test_rf, y5_train_rf, y5_test_rf = train_test_split(X5, y5,
    test_size=0.3, random_state=42, stratify=y5)

rf = RandomForestClassifier(n_estimators=40, random_state=42)
rf.fit(X5_train_rf, y5_train_rf)

# Predicciones sobre el entrenamiento
y5_pred_train_rf = rf.predict(X5_train_rf)

# Matriz de confusion
cm_rf = confusion_matrix(y5_train_rf, y5_pred_train_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=["No
    Fraude", "Fraude"])
```

```
disp_rf.plot(cmap="Blues", values_format='d')
plt.title("Matriz de Confusion - Random Forest")
plt.show()

# Reporte de clasificacion
print("Reporte de clasificacion - Random Forest:")
print(classification_report(y5_train_rf, y5_pred_train_rf, digits=8))

# Predicciones sobre prueba
y5_pred_test_rf = rf.predict(X5_test_rf)

cm_rf = confusion_matrix(y5_test_rf, y5_pred_test_rf)
disp_rf = ConfusionMatrixDisplay(confusion_matrix=cm_rf, display_labels=["No
    Fraude", "Fraude"])
disp_rf.plot(cmap="Blues", values_format='d')
plt.title("Matriz de Confusion - Random Forest")
plt.show()

print("Reporte de clasificacion - Random Forest:")
print(classification_report(y5_test_rf, y5_pred_test_rf, digits=8))
```

8.2. Anexo 2: Código red neuronal

```
#LIBRERIAS
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.ensemble import RandomForestRegressor
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import MeanSquaredError
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.inspection import permutation_importance
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,
   confusion_matrix, ConfusionMatrixDisplay
from sklearn.utils import class_weight
from sklearn.metrics import precision_score, recall_score, f1_score,
   accuracy_score

#RED NEURONAL

#LEER CSV

data_rn = pd.read_csv(r"C:\Users\Ines\OneDrive - SINEN Solucions Informatiques
   , SL\Escritorio\TFG\onlinefraud.csv")

data_rn = pd.get_dummies(data_rn, columns=["type"], drop_first=True)
```

```
# Convertir todas las columnas numericas a float
data_rn["step"] = data_rn["step"].astype(float)
data_rn["isFraud"] = data_rn["isFraud"].astype(float)
data_rn["isFlaggedFraud"] = data_rn["isFlaggedFraud"].astype(float)
data_rn["type_CASH_OUT"] = data_rn["type_CASH_OUT"].astype(float)
data_rn["type_DEBIT"] = data_rn["type_DEBIT"].astype(float)
data_rn["type_PAYMENT"] = data_rn["type_PAYMENT"].astype(float)
data_rn["type_TRANSFER"] = data_rn["type_TRANSFER"].astype(float)

le = LabelEncoder()
# Para nameOrig
data_rn["nameOrig"] = data_rn["nameOrig"].fillna("unknown").astype(str)
data_rn["nameOrig"] = le.fit_transform(data_rn["nameOrig"])
# Para nameDest
data_rn["nameDest"] = data_rn["nameDest"].fillna("unknown").astype(str)
data_rn["nameDest"] = le.fit_transform(data_rn["nameDest"])

#Normalizacion
scaler = MinMaxScaler()
cols_a_escalar = ["amount", "oldbalanceOrg", "newbalanceOrig", "oldbalanceDest
    ", "newbalanceDest"]
data_rn[cols_a_escalar] = scaler.fit_transform(data_rn[cols_a_escalar])

data_rn.head()

#SELECCION DE VARIABLES

# Dataset reducido
fraudes = data_rn[data_rn['isFraud'] == 1].sample(n=20, random_state=42)
no_fraudes = data_rn[data_rn['isFraud'] == 0].sample(n=20000, random_state=42)
reducido_rn = pd.concat([fraudes, no_fraudes]).sample(frac=1, random_state=42)

# Matriz de correlacion
```

```
correlation_matrix = reducido_rn.corr()
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f',
    linewidths=0.5)
plt.title('Mapa de Correlacion entre Variables')
plt.show()

reducido_rn["saldo_cambio_origen"] = reducido_rn["oldbalanceOrg"]- reducido_rn
    ["newbalanceOrig"]
reducido_rn["saldo_cambio_destino"] = reducido_rn["oldbalanceDest"]-
    reducido_rn["newbalanceDest"]
reducido_rn["proporcion_transferencia"] = reducido_rn["amount"] / (reducido_rn
    ["oldbalanceOrg"] + 1) # +1 para evitar division por 0
reducido_rn["saldo_cero_origen"] = (reducido_rn["newbalanceOrig"] == 0).astype
    (int)
reducido_rn["destino_sin_saldo"] = (reducido_rn["oldbalanceDest"] == 0).astype
    (int)
reducido_rn["transaccion_alto_riesgo"] = ((reducido_rn["type_CASH_OUT"] == 1)
    |(reducido_rn["type_TRANSFER"] == 1)) & (reducido_rn["amount"] >
    reducido_rn["oldbalanceOrg"] * 0.9)
reducido_rn["transaccion_alto_riesgo"] = reducido_rn["transaccion_alto_riesgo"
    ].astype(int)
reducido_rn["frecuencia_transacciones_origen"] = reducido_rn.groupby("nameOrig
    ")["nameOrig"].transform("count")
reducido_rn["frecuencia_transacciones_destino"] = reducido_rn.groupby("
    nameDest")["nameDest"]

reducido_rn.head()

# Matriz de correlacion con todas las variables
df_corr = reducido_rn
# Calcular la matriz de correlacion
correlation_matrix = df_corr.corr()
```

```
plt.figure(figsize=(12,8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f",
    linewidths=0.5)
plt.title("Matriz de Correlacion de Todas las Variables con `isFraud`")
plt.show()

reducido_rn = reducido_rn.drop(columns=['type_PAYMENT', 'type_DEBIT', '
    frecuencia_transaccion']

# Matriz correlacion solo fraudes
fraudes = reducido_rn[reducido_rn['isFraud'] == 1]
fraudes_numero = fraudes.select_dtypes(include='number')

correlation_fraude = fraudes_numero.corr()

plt.figure(figsize=(14, 8))
sns.heatmap(correlation_fraude, annot=True, cmap='coolwarm', fmt='.2f',
    linewidths=0.5)
plt.title('Mapa de Correlacion entre Variables (solo transacciones
    fraudulentas)')
plt.tight_layout()
plt.show()

# Matriz de correlacion solo NO fraudes
no_fraudes = reducido_rn[reducido_rn['isFraud'] == 0]
no_fraudes_numero = no_fraudes.select_dtypes(include='number')

correlation_no_fraude = no_fraudes_numero.corr()

plt.figure(figsize=(14, 8))
sns.heatmap(correlation_no_fraude, annot=True, cmap='coolwarm', fmt='.2f',
    linewidths=0.5)
```

```
plt.title('Mapa de Correlacion entre Variables (solo transacciones no
    fraudulentas)')
plt.tight_layout()
plt.show()

# 2. Restar ambas matrices
diferencia_corr = correlation_fraude- correlation_no_fraude
diferencias = diferencia_corr.abs().sum().sort_values(ascending=False)

# 4. Seleccionar variables
top_variables = diferencias.drop('isFraud', errors='ignore').head(12).index.
    tolist()
print("Variables con mayor cambio de correlacion:", top_variables)

top_variables = diferencias.drop('isFraud', errors='ignore').head(12)

# Crear la grafica
plt.figure(figsize=(8, 5))
top_variables.sort_values().plot(kind='barh', color='skyblue')
plt.xlabel('Importancia')
plt.ylabel('Variables')
plt.title('Importancia de variables segun cambio de correlacion')
plt.tight_layout()
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.show()

# Matriz confusion (resta)
plt.figure(figsize=(14, 8))
sns.heatmap(diferencia_corr, annot=True, cmap='coolwarm', fmt='.2f',linewidths
    =0.5)
plt.title('Mapa de Correlacion entre Variables (diferencia de correlaciones)')
plt.tight_layout()
plt.show()
```

```
#ENTRENAMIENTO

top_variables = diferencias.drop('isFraud', errors='ignore').head(12).index.
    tolist()
X_rn = reducido_rn[top_variables]
y_rn = reducido_rn['isFraud']

# Dividir el conjunto de datos en entrenamiento (70%) y prueba (30%)
X_train_rn, X_test_rn, y_train_rn, y_test_rn = train_test_split(X_rn, y_rn,
    test_size=0.3, random_state=42, stratify=y_rn)

# 4. Escalar
scaler = StandardScaler()
X_train_rn = scaler.fit_transform(X_train_rn)
X_test_rn = scaler.transform(X_test_rn)
print("Tamano X_train_rn:", X_train_rn.shape)
print("Tamano y_train_rn:", y_train_rn.shape)
print("Tamano X_test_rn:", X_test_rn.shape)
print("Tamano y_test_rn:", y_test_rn.shape)

# Arquitectura
num_features = X_train_rn.shape[1]

# Crear modelo
model = Sequential([
# Capa de entrada y primera capa oculta
Dense(128, activation='relu', input_shape=(num_features,)),
Dropout(0.2),
# Segunda capa oculta
Dense(64, activation='relu'),
# Tercera capa oculta
Dense(32, activation='relu'),
```

```
# Cuarta capa oculta
Dense(16, activation='relu'),
# Capa de salida para clasificacion
Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy',
metrics=['accuracy'])
print('Resumen del modelo:')
model.summary()

early_stop = EarlyStopping(monitor='val_loss', patience=3,restore_best_weights
    =True)

entreno = model.fit(X_train_rn, y_train_rn,
validation_split=0.2,
epochs=15,
batch_size=32,
callbacks=[early_stop],
verbose=1)

# EVALUACION DE LAS PREDICCIONES
# 1. Predecir en el conjunto de entrenamiento
y_pred_prob_rn = model.predict(X_train_rn)

# 2. Convertir a etiquetas (0 o 1)
y_pred_rn = (y_pred_prob_rn > 0.7).astype(int)

# 3. Matriz de confusion
cm = confusion_matrix(y_train_rn, y_pred_rn)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
```

```
plt.title("Matriz de Confusion - Train")
plt.show()

# Reporte de clasificacion
print("Reporte de clasificacion:")
print(classification_report(y_train_rn, y_pred_rn, digits=4))

# 1. Predecir en el conjunto de prueba
y_pred_prob_test_rn = model.predict(X_test_rn)

# 2. Convertir a etiquetas (0 o 1)
y_pred_test_rn = (y_pred_prob_test_rn > 0.5).astype(int)

cm = confusion_matrix(y_test_rn, y_pred_test_rn)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
plt.title("Matriz de Confusion - Test")
plt.show()

print("Reporte de clasificaci'on:")
print(classification_report(y_test_rn, y_pred_test_rn, digits=4))

#ENTRENAMIENTO Y PREDICCION DATASET ORIGINAL

#Crear variables
data_rn["saldo_cambio_origen"] = data_rn["oldbalanceOrg"]- data_rn["
    newbalanceOrig"]
data_rn["saldo_cambio_destino"] = data_rn["oldbalanceDest"]- data_rn["
    newbalanceDest"]
data_rn["proporcion_transferencia"] = data_rn["amount"] / (data_rn["
    oldbalanceOrg"] + 1) # +1 para evitar division por 0
data_rn["saldo_cero_origen"] = (data_rn["newbalanceOrig"] == 0).astype(int)
data_rn["destino_sin_saldo"] = (data_rn["oldbalanceDest"] == 0).astype(int)
```

```
data_rn["transaccion_alto_riesgo"] = ((data_rn["type_CASH_OUT"] == 1) |(
    data_rn["type_TRANSFER"] == 1)) & (data_rn["amount"] > data_rn["
    oldbalanceOrg"] * 0.9)
data_rn["transaccion_alto_riesgo"] = data_rn["transaccion_alto_riesgo"].astype
    (int)
data_rn["frecuencia_transacciones_origen"] = data_rn.groupby("nameOrig")["
    nameOrig"].transform("count")
data_rn["frecuencia_transacciones_destino"] = data_rn.groupby("nameDest")["
    nameDest"].transform("count")


data_rn.head()


#ENTRENAMIENTO


top_variables = diferencias.drop('isFraud', errors='ignore').head(12).index.
    tolist()
X2_rn = data_rn[top_variables]
y2_rn = data_rn['isFraud']


# Dividir el conjunto de datos en entrenamiento (70%) y prueba (30%)
X2_train_rn, X2_test_rn, y2_train_rn, y2_test_rn = train_test_split(X2_rn,
    y2_rn, test_size=0.3, random_state=42)



# 4. Escalar
scaler = StandardScaler()
X2_train_rn = scaler.fit_transform(X2_train_rn)
X2_test_rn = scaler.transform(X2_test_rn)
print("Tamano X2_train_rn:", X2_train_rn.shape)
print("Tamano y2_train_rn:", y2_train_rn.shape)
print("Tamano X2_test_rn:", X2_test_rn.shape)
print("Tamano y2_test_rn:", y2_test_rn.shape)
```

```
# Arquitectura
num_features = X2_train_rn.shape[1]
# Crear modelo
model = Sequential([
# Capa de entrada y primera capa oculta
Dense(512, activation='relu', input_shape=(num_features,)),
Dropout(0.2),
# Segunda capa oculta
Dense(256, activation='relu'),
Dropout(0.2),
# Tercera capa oculta
Dense(128, activation='relu'),
Dropout(0.2),
# Cuarta capa oculta
Dense(64, activation='relu'),
Dropout(0.2),
# Quinta capa oculta
Dense(32, activation='relu'),
# Capa de salida para clasificacion
Dense(1, activation='sigmoid')
])

model.compile(optimizer=Adam(learning_rate=0.001),
loss='binary_crossentropy',
metrics=['accuracy'])
print('Resumen del modelo:')
model.summary()

early_stop = EarlyStopping(monitor='val_loss', patience=3,restore_best_weights
    =True)

entreno = model.fit(X2_train_rn, y2_train_rn,
validation_split=0.2,
```

```
epochs=30,
batch_size=32,
callbacks=[early_stop],
verbose=1)

#PREDICCIONES

# 1. Predecir en el conjunto de entrenamiento
y2_pred_prob_rn = model.predict(X2_train_rn)

# 2. Convertir a etiquetas (0 o 1)
y2_pred_rn = (y2_pred_prob_rn > 0.7).astype(int)

# 3. Matriz de confusion
cm = confusion_matrix(y2_train_rn, y2_pred_rn)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
plt.title("Matriz de Confusion - Train")
plt.show()

# Reporte de clasificacion
print("Reporte de clasificacion:")
print(classification_report(y2_train_rn, y2_pred_rn, digits=4))

# 1. Predecir en el conjunto de prueba
y2_pred_prob_test_rn = model.predict(X2_test_rn)

# 2. Convertir a etiquetas (0 o 1)
y2_pred_test_rn = (y2_pred_prob_test_rn > 0.5).astype(int)

cm = confusion_matrix(y2_test_rn, y2_pred_test_rn)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1])
disp.plot(cmap='Blues')
```

```
plt.title("Matriz de Confusion - Test")
plt.show()


print("Reporte de clasificacion:")
print(classification_report(y2_test_rn, y2_pred_test_rn, digits=4))
```