UNIVERSIDAD DE CONCEPCIÓN
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

Detección de fraude transaccional mediante modelos de aprendizaje automático: Una aplicación a una entidad financiera chilena

Por: **Constanza Paz Luna Moreno**

Memoria de Título presentada a la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Concepción para optar al título profesional de Ingeniera Civil Matemática

Junio 2024
Concepción, Chile

Profesores Guía:
Guillermo Ferreira Cabezas
Pamela Meléndez Toso

Resumen

Con el avance continuo de la tecnología moderna, el volumen de transacciones financieras ha aumentado significativamente, lo que a su vez ha generado un incremento en los casos de fraude. Los estafadores están constantemente buscando nuevas tácticas y estrategias para llevar a cabo actividades ilegales, aprovechando las vulnerabilidades de los sistemas financieros. Por consiguiente, el desarrollo de tecnologías de protección contra el fraude se ha vuelto crucial para reducir las pérdidas en las instituciones financieras. Las técnicas de aprendizaje automático son una opción cada vez más estudiada e implementada en la detección de fraude transaccional.

Esta Memoria de Título aborda dicho desafío en colaboración con una Institución Financiera Chilena, centrándose en la implementación y comparación de diversos modelos de aprendizaje automático. Estos modelos se entrenan para identificar transacciones fraudulentas con precisión, al mismo tiempo que protegen la integridad de las transacciones legítimas mediante la minimización de las clasificaciones erróneas de fraude, garantizando así una experiencia positiva para el cliente.

Los modelos de aprendizaje automático seleccionados para esta tarea incluyen Regresión Logística, Redes Neuronales Artificiales, Máquinas de Vectores de Soporte, AdaBoost, CatBoost, Bosques Aleatorios y Naive Bayes, los cuales fueron comparados tanto entre sí como con el modelo XGBoost utilizado por la Institución Financiera.

Los resultados revelaron que, entre todos los modelos evaluados, CatBoost demostró el mejor rendimiento, convirtiéndose así en una herramienta poderosa para combatir el fraude y proteger los activos de los clientes.

En esta Memoria de Título, se presentan y comparan una variedad de métodos sólidos para detectar fraudes en transacciones financieras, los cuales podrían ser implementados por la Institución Financiera. Estos métodos contribuyen significativamente al fortalecimiento de las defensas del sector financiero contra esta creciente amenaza.

Índice general

Índice de Tablas

Índice de figuras

Índice de algoritmos

Capítulo 1

Introducción

1.1. Planteamiento del problema

En los últimos años, se ha observado a nivel global una masificación en el uso de sistemas de pago digitales, lo que ha generado un aumento en el volumen de transacciones para bancos y compañías de tarjetas de crédito y pagos. Chile no se queda atrás, mostrando un notable incremento en la adopción y utilización de medios de pago digitales.

En el Informe de Sistemas de Pago, publicado por el Banco Central de Chile (2023), se señala que los pagos digitales continúan creciendo rápidamente, tanto en monto como en número de transacciones. Varios factores contribuyen a este incremento, tales como mejoras en la conectividad y acceso a teléfonos inteligentes; nuevas tecnologías que facilitan la experiencia del usuario, como los pagos sin contacto y el uso de códigos QR; la implementación de botones de pago asociados al comercio electrónico, así como nuevos servicios ofrecidos por empresas de tecnología financiera.

Según lo declarado en dicho informe, los pagos digitales ascendieron a 110 % del PIB anualmente, de los cuales un 68 % corresponden a Transferencias Electrónicas de Fondos (TEF). El pago con tarjetas es el que representa un mayor número de transacciones, ascendiendo a 11 millones diarias, siendo las tarjetas de débito las más utilizadas.

En el ámbito internacional, se puede examinar el caso de PayPal Inc., una empresa de pagos de origen estadounidense con alcance global. Según los datos proporcionados por GlobalData (2022), en 2021 se llevaron a cabo cerca de 19.300 millones de transacciones a través de PayPal, generando un volumen total de pagos de 1,25 billones de dólares.

Sin embargo, PayPal no está exento de fraude. Según lo publicado por Aerospike (2022), empresa asociada a PayPal, la tasa de fraude de esta última oscila entre el 0,17 % y el 0,18 % de los ingresos. Con el aumento de las transacciones, tanto esta empresa como otras entidades financieras enfrentan un incremento en el fraude. Esto obliga a estas entidades a evaluar y desarrollar constantemente estrategias para prevenir y detectar el fraude de manera efectiva. Garantizar transacciones en línea seguras y mitigar el impacto del fraude se vuelve fundamental para mantener la confianza de los clientes al comprar productos y servicios.

En este escenario, el aprendizaje automático ha emergido como una herramienta indispensable. Su capacidad para analizar grandes volúmenes de datos de transacciones de manera rápida y eficiente lo ha vuelto muy popular para la detección de fraudes en tiempo real.

El aprendizaje automático permite no solo identificar patrones sutiles y comportamientos anómalos que podrían indicar fraude, sino que también evoluciona continuamente con la recopilación de datos. Es clave para identificar patrones emergentes y adaptarse rápidamente a nuevas formas de fraude, convirtiéndose así en un recurso esencial en la lucha contra el fraude en comercio electrónico y finanzas.

Es en este contexto que esta Memoria de Título aborda la problemática de clasificar si las transacciones realizadas a través de tarjetas de una Institución Financiera Chilena son fraudulentas. Para ello, se propone aplicar diversos modelos de aprendizaje automático y comparar su desempeño con el modelo básico de Regresión Logística, así como con el modelo implementado por la Institución Financiera.

1.2. Estado del arte

El avance tecnológico global ha sido un motor significativo en el crecimiento del campo de la inteligencia artificial, especialmente en el aprendizaje automático. Este progreso ha sido impulsado por un mejor acceso a internet, datos y capacidades de procesamiento computacional avanzadas, lo que facilita la integración para el entrenamiento y las operaciones en tiempo real. Como resultado, el aprendizaje automático y minería de datos se convierten en alternativas apropiadas para hacer frente al fraude transaccional (Moreira et al., 2022).

Debido a la creciente presencia de amenazas con graves consecuencias en el sector financiero, las instituciones financieras se ven obligadas a mejorar continuamente sus sistemas de inteligencia para detectar transacciones fraudulentas. Por lo tanto, varios investigadores han explorado diversas técnicas de aprendizaje automático para cumplir con este propósito. En la Tabla 1.2.1 se presenta evidencia bibliográfica reciente sobre la aplicación del aprendizaje automático en la modelación del fraude transaccional, cubriendo diferentes aspectos como el fraude en instituciones financieras y bancarias, así como el fraude en tarjetas de crédito y el comercio electrónico.

Tabla 1.2.1: Resumen de los trabajos relacionados sobre detección de fraude transaccional con técnicas de aprendizaje automático.

Año	Autores	Técnicas	Descripción
2019	Shpyrko and Koval	- Regresión Logística - Máquinas de vectores de soporte - K-Vecinos más cercanos - Árboles de decisión. - Redes Neuronales Artificiales	Su objetivo fue desarrollar un modelo capaz de detectar y bloquear al instante transacciones fraudulentas. Para ello, se utilizó una base de datos del sistema de pago con cuentas transaccionales. Se destacó el rendimiento de la regresión logística, que alcanzó una exactitud del 94 %, y de las redes neuronales, con una exactitud del 93.1 % sobre los datos submuestreados.
2020	Trivedi et al.	- Bosque aleatorio - Árboles de decisión - Redes neuronales artificiales - Máquina de vectores de soporte - Naïve Bayes - Regresión logística. - Gradient boosting	El artículo presenta un mecanismo efectivo de detección de fraude con tarjetas de crédito. Los conjuntos de datos utilizados incluyen información sobre transacciones realizadas con tarjetas de crédito por titulares de cuentas europeas. En este contexto, el método de bosque aleatorio muestra resultados superiores en comparación con otros clasificadores, obteniendo una precisión de 95 % y exactitud de 94 %.
2021	Chen and Han	- Naïve Bayes - Máquinas de vectores de soporte - CatBoost.	Este documento aborda los desafíos de detectar fraudes en la industria financiera. Los resultados de experimentos con un conjunto extenso de datos, proporcionados por una empresa especializada en la prevención de fraudes en transacciones en línea, indican que CatBoost ha logrado una mejora significativa en la exactitud de detección, alcanzando un 98.3 %.
2022	Gedela and Karthikeyan	- AdaBoost - Naïve Bayes - Regresión logística - Redes neuronaes artificiales - Árboles de decisión.	El artículo se centra en evaluar métodos de aprendizaje automático para la detección de fraudes con tarjetas de crédito, un problema crítico para los bancos emisores. Se observó que el algoritmo AdaBoost tuvo un rendimiento notablemente superior a otros modelos, superándolos en todas las métricas propuestas.

Tabla 1.2.1 – Continuación de la página anterior

Año	Autores	Técnicas	Descripción
2023	Gupta et al.	- Regresión logística - Árboles de decisión - XGboost - Redes neuronales artificiales.	Este estudio muestra cómo utilizar múltiples clasificadores y equilibrar los datos con enfoques de aprendizaje automático, para detectar fraudes con tarjetas de crédito. El clasificador XGBoost obtuvo los mejores resultados entre los cuatro algoritmos utilizados, tanto con datos desbalanceados como balanceados.
2024	Khalid et al.	- Máquinas de vectores de soporte - K-vecinos más cercanos - Bosque aleatorio - Bagging - Boosting - Ensamble de los modelos anteriores	Este artículo explora el uso de modelos de aprendizaje automático, centrándose en métodos de ensamble, para mejorar la detección de fraudes con tarjetas de crédito. Se evalúa el modelo utilizando datos de transacciones de titulares de tarjetas de crédito europeos, lo que proporciona un escenario realista. Lograr un equilibrio entre precisión y eficiencia computacional fue crucial. El modelo de ensamble demostró ser el más efectivo en mitigar falsos positivos y negativos.

1.3. Objetivos

El fraude transaccional está experimentando un aumento tanto en frecuencia como en sofisticación, lo que impulsa a las entidades financieras a evaluar y desarrollar continuamente estrategias para prevenir y detectar el fraude de manera efectiva.

En vista de esto, en colaboración con una Institución Financiera Chilena, el objetivo principal de esta Memoria de Título es abordar el problema del fraude transaccional mediante la implementación de diversos modelos de aprendizaje automático. Estos modelos tienen como finalidad identificar de manera precisa las transacciones con mayor probabilidad de ser fraudulentas, manteniendo una baja tasa de errores para no descuidar las transacciones legítimas.

En base a la revisión bibliográfica realizada, se han seleccionado los modelos de Regresión Logística, XGBoost (Modelo Institución), Redes Neuronales Artificiales, Máquinas de Vectores de Soporte, AdaBoost, CatBoost, Bosques Aleatorios y Naive Bayes para su estudio y aplicación, debido a su popularidad y buenos resultados en las investigaciones recientes.

Los datos proporcionados por la Institución Financiera Chilena son utilizados para el entrenamiento, validación y evaluación de los modelos. Se aplican los mismos datos a todos los modelos, lo que permite comparar su rendimiento. Además de determinar qué modelo ofrece la mejor precisión en la detección de fraude, se explorarán las peculiaridades de cada técnica. Esta Memoria de Título aspira a proporcionar herramientas de detección más sólidas, con el fin de fortalecer y diversificar las defensas contra el fraude transaccional en el sector financiero.

1.3.1. Objetivos específicos

- Implementar las técnicas estadísticas de Regresión Logística, XGBoost, Redes Neuronales Artificiales, Máquinas de Vectores de Soporte, AdaBoost, CatBoost, Bosques Aleatorios y Naive Bayes en conjuntos de datos de transacciones electrónicas reales para la detección de fraudes.
- Evaluar el rendimiento de los modelos de aprendizaje automático mediante el análisis de métricas ampliamente reconocidas en problemas de clasificación, para comprender su eficacia en la detección de fraudes.
- Comparar detalladamente los resultados obtenidos por cada técnica estadística, identificando fortalezas y debilidades, con el fin de proporcionar una visión completa de su aplicabilidad en la predicción de fraudes transaccionales.

Capítulo 2

Marco Teórico

2.1. Fraude transaccional

La Entidad Financiera que proporcionó los datos define el **fraude** como "el uso de la información de los clientes para realizar transacciones financieras, en las cuales dichos clientes indican no haber participado, ni haber autorizado a un tercero a realizarlo en representación de este".

Entre las acciones consideradas como fraudulentas, están:

- Usar los activos, bienes o servicios de la empresa para fines distintos a los establecidos por la compañía.
- Obtener y entregar información confidencial de los clientes tales como: Número de tarjetas, claves, teléfonos, dirección, correo electrónico, desde los sistemas de la empresa, con el fin usarla para fines no autorizados.
- Modificación de datos del cliente en el sistema sin la solicitud del titular de la tarjeta, como se define en los distintos procedimientos de atención al cliente.
- Uso malicioso de la información de la tarjeta, para realizar transacciones no autorizadas por el titular de la tarjeta en ambiente presencial y/o ambiente ausente.

La última acción involucra transacciones electrónicas, lo que se refiere a un tipo específico de fraude conocido como **fraude transaccional**. Torres (2022) lo define como "todas aquellas transacciones que el cliente no reconoce o que fueron realizadas sin su consentimiento, y este a su vez las reclama a la entidad como un consumo no reconocido".

La investigación reciente ha demostrado la eficacia de las técnicas de aprendizaje automático en la detección de transacciones fraudulentas. La inteligencia artificial y el aprendizaje automático son las principales tecnologías utilizadas para combatir el fraude en línea (PayPal, 2023). Estas técnicas tienen el potencial de evolucionar y detectar patrones de fraude previamente no identificados.

2.2. Aprendizaje automático

Murphy (2012) define el **aprendizaje automático** (también conocido como *machine learning*) como un conjunto de métodos capaces de detectar patrones en los datos de forma automática y utilizar estos patrones descubiertos para realizar predicciones sobre datos futuros.

El aprendizaje automático suele categorizarse en dos tipos principales. En primer lugar, está el **aprendizaje supervisado**, que se centra en aprender la relación entre las $p-$variables de entrada $\boldsymbol{x}$ y la variable de salida y. Esto se logra utilizando un conjunto de datos etiquetado, que consiste en pares de entrada-salida, que se denotan como $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^N$. Aquí, $\mathcal{D}$ se refiere al conjunto de entrenamiento y N representa el número de observaciones en dicho conjunto. Cada entrada de entrenamiento $\boldsymbol{x}_i$ es un vector de observaciones de $p-$variables, mientras que cada salida y_i puede ser una variable categórica perteneciente a un conjunto finito, es decir, $y_i \in \{1, \ldots, C\}$, o una variable numérica. Cuando y_i es categórica, esto da lugar a un problema conocido como clasificación o reconocimiento de patrones. Por otro lado, cuando y_i toma valores reales, se denomina problema de regresión.

La segunda categoría de aprendizaje automático es el **aprendizaje no supervisado**. Aquí, se proporcionan únicamente las entradas, denotadas como $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^N$, con el objetivo de descubrir patrones significativos en los datos. En contraste con el aprendizaje supervisado, en el aprendizaje no supervisado no se proporciona información explícita sobre las salidas esperadas o las categorías de los datos; además, no existe una métrica de error directa para evaluar el rendimiento del modelo.

Existe un tercer tipo de aprendizaje automático, conocido como **aprendizaje por refuerzo**, que se utiliza con menor frecuencia. El objetivo de este es aprender a tomar decisiones secuenciales en un entorno en función de las señales esporádicas de recompensa o castigo que se reciben.

2.2.1. Problema de clasificación

Un problema de clasificación es un tipo de aprendizaje supervisado que busca desarrollar una función capaz de asignar nuevas instancias de datos a una de las clases predefinidas, utilizando los patrones identificados durante el entrenamiento.

Matemáticamente, se define $\boldsymbol{x}$ como el conjunto de datos, mientras que y representa el conjunto de etiquetas o clases. Existe un mapeo f:

$$f : \boldsymbol{x} \longrightarrow y,$$

cuyos valores se conocen sólo para elementos de una muestra de aprendizaje finita $\mathcal{D} = (\boldsymbol{x}_i, y_i)_{i=1}^{N}$. El objetivo es construir una aproximación de f:

$$\hat{f} : \boldsymbol{x} \longrightarrow y,$$

que sea capaz de clasificar de manera precisa un objeto arbitrario $\boldsymbol{x}_i \in \boldsymbol{x}$. Dependiendo de la cardinalidad de y, se tienen diferentes problemas de clasificación (Shpyrko and Koval, 2019):

- **Clasificación binaria**. Es el caso más sencillo con $|y| = 2$. Las clases suelen representarse como "positiva" y "negativa", o bien como "1" y "0".
- **Clasificación multiclase**. El número de clases es mayor a 2.

Además, las clases pueden ser de los siguientes tipos:

- **Clases no superpuestas**. Cada objeto pertenece exclusivamente a una clase.
- **Clases ordinarias**. Un objeto puede pertenecer a varias clases simultáneamente.
- **Clases difusas**. Se determina el grado de pertenencia del objeto a cada una de las clases, generalmente expresado como un número en el rango de 0 a 1.

En el caso de la detección de fraude transaccional, se trata de un problema de clasificación binaria con clases no superpuestas, cuya formulación es la siguiente:

Se denota por N el número de registros de transacciones presentes en la base de entrenamiento. Los datos consisten en pares de la forma $(\boldsymbol{x}_i, y_i)$, donde $i = 1, \ldots, N$.

Cada observación $\boldsymbol{x}_i$ es un vector que contiene características de la i-ésima transacción, mientras que y_i es la variable de salida u objetivo. Esta variable indica la clase a la cual pertenece la transacción, siguiendo la siguiente codificación:

$$y_i = \begin{cases} 0 & \text{si} \quad \text{es una transacción normal,} \\ 1 & \text{si} \quad \text{es una transacción fraudulenta.} \end{cases}$$

Por lo tanto, el objetivo consiste en encontrar una función $\hat{f}$ que determine si una transacción es normal o fraudulenta. En otras palabras, se busca clasificar cada transacción $\boldsymbol{x}_i$ como fraudulenta o no, asignando la etiqueta 1 en caso de fraude y 0 en caso contrario.

2.3. Modelos de clasificación

Los modelos de clasificación son algoritmos cuya función es predecir la clase a la que pertenece un objeto, utilizando un conjunto de características y las relaciones existentes entre ellas. A continuación, se describirán los modelos de aprendizaje supervisado seleccionados para enfrentar esta tarea de clasificación.

2.3.1. Regresión Logística

La Regresión Logística (RL), cuyos fundamentos se remontan a los aportes de Cornfield et al. (1961), consiste en un tipo de análisis de clasificación utilizado para predecir el resultado de una variable categórica en función de las variables explicativas. Este enfoque ha sido ampliamente utilizado desde la década de 1980, impulsado por los avances en la capacidad informática.

Se utiliza para estimar la probabilidad de que un resultado se asocie con uno de dos resultados posibles, representados por la variable dicotómica y, que, con respecto a una variable explicativa x toma valores:

$$y = \begin{cases} 1 & \text{si} \quad \text{el evento ocurre} \\ 0 & \text{si} \quad \text{el evento no ocurre.} \end{cases}$$

Según lo planteado por James et al. (2013), la RL es una modificación al modelo de regresión clásico, donde se requiere utilizar una función $p(x)$ que produzca valores entre 0

y 1 para todos los valores de x. En lo que sigue, se utiliza la función logística:

$$p(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}.$$

Al manipular la función anterior y aplicar logaritmo, se llega a la función logit:

$$\text{logit}(p(x)) = \log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x.$$

Ahora se considera el problema de predecir una respuesta binaria utilizando múltiples predictores. El modelo puede generalizarse de la siguiente manera:

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p,$$

donde $x_1, \ldots, x_p$ son las $p-$variables independientes. De manera análoga al caso de regresión logística simple, se reescribe la ecuación se reescribe como:

$$p(x) = \frac{e^{\beta_0+\beta_1 x_1+\cdots+\beta_p x_p}}{1+e^{\beta_0+\beta_1 x_1+\ldots\beta_p x_p}}.$$

Posteriormente, se suele utilizar el método de máxima verosimilitud para estimar los parámetros $\beta_1, \ldots, \beta_p$ del modelo. El método de máxima verosimilitud proporciona los valores de los parámetros que maximizan la probabilidad de obtener el conjunto de datos observado.

2.3.1.1. Criterios de selección de variables

Es conocido que la RL, al ser un modelo relativamente simple, tiende a desempeñarse mejor en problemas que involucran un número reducido de variables explicativas. Por lo tanto, en el caso específico de RL, se han realizado distintos análisis de variables con el fin de identificar cuáles de ellas contribuyen de manera más efectiva al modelo. En seguida, se describen las técnicas utilizadas, en el orden en que se aplicaron:

- **Factor de inflación de la varianza**:

  Uno de los supuestos fundamentales de la RL es la ausencia de colinealidad perfecta entre las variables explicativas. Una manera de evaluar la multicolinealidad es calcular el factor de inflación de la varianza (VIF), que es un índice que mide en qué medida la varianza de un coeficiente de regresión estimado aumenta debido a la colinealidad.

El VIF de cada variable se calcula de la siguiente manera:

$$VIF(\hat{\beta}_j) = \frac{1}{1 - R_j^2},$$

donde R_j^2 es el coeficiente R^2 de una regresión de x_j sobre todas las demás variables predictoras. Como regla general, un valor de VIF que excede 5 o 10 indica una cantidad problemática de colinealidad (James et al., 2013). En este trabajo, se descartarán aquellas variables con un VIF superior a 10.

- **Eliminación progresiva (*backward*):**

  El método de selección de variables *backward* se enfoca en simplificar un modelo de regresión, utilizando como métrica el criterio de información de Akaike (AIC).

  El AIC es una medida de la calidad relativa de un modelo, que se emplea para seleccionar el modelo que mejor se ajusta a los datos, considerando tanto la bondad de ajuste como la complejidad del modelo. Se calcula mediante la siguiente fórmula:

  $$AIC = 2k - 2\ln(L),$$

  donde k es el número de parámetros del modelo y L es el máximo valor de la función de verosimilitud para dicho modelo.

  El método *backward* consiste en iniciar con un modelo que incluye todas las variables y, de manera iterativa, eliminar una variable a la vez del modelo. En cada paso, se elimina la variable que produce la mayor disminución posible en el AIC al ser retirada del modelo. El proceso se detiene cuando ya no es posible eliminar más variables sin aumentar el AIC.

  De este modo, el modelo final con las variables seleccionadas representa un equilibrio adecuado entre la bondad de ajuste y la complejidad del modelo (James et al., 2013).

- **Coeficiente de Kolmogorov-Smirnov**

  Si bien su formulación matemática se abordará más adelante, el coeficiente de Kolmogorov-Smirnov (KS), en el contexto de la selección de variables, se utiliza para evaluar el poder predictivo de una variable para distinguir entre diferentes valores de la variable objetivo. Las variables con un coeficiente KS más alto indican una relación más fuerte con la variable respuesta y una mejor capacidad para distinguir entre las clases (Khan, 2017).

Para seleccionar variables basadas en el coeficiente KS, se establece un umbral de 0.2. Luego, se seleccionan las variables cuyos valores de KS están por encima de dicho valor.

- ***Information Value***

  El *Information Value* (IV) es una métrica ampliamente utilizada en el análisis de variables en modelos predictivos, especialmente en la modelización de riesgos. Evalúa la capacidad predictiva de una variable explicativa en relación con la variable objetivo.

  El IV se calcula de la siguiente manera:

$$IV = \sum_{i=1}^{p} (P_{\text{fraude},i} - P_{\text{no-fraude},i}) \cdot \ln\left(\frac{P_{\text{fraude},i}}{P_{\text{no-fraude},i}}\right),$$

  donde $P_{\text{fraude},i}$ es la proporción de fraude para la i-ésima variable explicativa, y $P_{\text{no-fraude},i}$ es la proporción de transacciones normales según la i-ésima variable.

  Una variable con un IV entre 0,1 y 0,3 indica un poder predictivo moderado, por lo que se considera relevante (Lin, 2018). En esta Memoria de Título, se establece un umbral de 0,25 para la selección de variables.

Estas técnicas se aplicaron para reducir el número de variables en el modelo de RL, lo que permitió optimizar su desempeño y capacidad de generalización.

En contraste, para los otros modelos de aprendizaje automático, se incluyeron todas las variables disponibles sin aplicar ninguna técnica de selección. Esto se debe a que no es fundamental como en el caso de la regresión logística, lo que permite una mayor información en las variables independientes. Además, esta decisión facilita una mejor comparación con el modelo de la Institución.

2.3.2. XGBoost

Para comprender el algoritmo XGBoost, es esencial familiarizarse con las metodologías de *Boosting* y *Gradient Boosting*.

- **Boosting**

  El *Boosting* es una técnica de aprendizaje automático que consiste en combinar varios modelos débiles para crear un modelo fuerte. En este caso, los modelos débiles usados son árboles de decisión. La técnica implica entrenar secuencialmente una serie de árboles de decisión, donde cada árbol aprende de los errores cometidos por los árboles anteriores. El modelo final se obtiene a través del último árbol en esta secuencia, que incorpora el conocimiento acumulado para mejorar la precisión del modelo.

  A continuación se presenta el Algoritmo 2.3.1, que describe las etapas del proceso *Boosting*:

Algoritmo 2.3.1 *Boosting*

1. Sea $\hat{f}(x) = 0$ y $r_i = y_i \ \forall i$, siendo y_i el valor objetivo a predecir.
2. Para $b = 1, 2, \ldots, B$:
   (a) Crear un árbol $\hat{f}^b$ con d divisiones ($d+1$ nodos terminales) a los datos de entrenamiento (X, r).
   (b) Actualizar $\hat{f}$:
   $$\hat{f} \longleftarrow \hat{f}(x) + \lambda \hat{f}^b.$$
   (c) Actualizar los residuos:
   $$r_i \longleftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Salida:
   $$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

  Notar que el algoritmo cuenta con tres parámetros de ajuste: B, que corresponde al número de árboles; $\lambda \in (0, 1)$, que es un parámetro de contracción utilizado para regular la velocidad de aprendizaje de boosting; y d el número de nodos de cada árbol (James et al., 2013).

- ***Gradient boosting***

  En *Gradient Boosting*, desarrollado por Friedman (2001), se consideran los algoritmos de *boosting* como iteraciones de *Gradient Descent*, un método iterativo

de optimización para encontrar mínimos locales de una función.

Específicamente, el algoritmo *gradient boosting* busca una aproximación $\hat{f}(x)$ como una suma ponderada de M árboles de decisión $h_m(x)$:

$$\hat{f}(x) = \sum_{m=1}^{M} \gamma_m h_m(x).$$

A continuación, en el Algoritmo 2.3.2 se detallan las etapas de *gradient boosting*, cuyo input es un conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^{N}$, una función de pérdida diferenciable $L(y, f(x))$ y el número de iteraciones M:

Algoritmo 2.3.2 *Gradient Boosting*

1. Se calcula el valor inicial:
$$f_0(x) = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, \theta).$$
2. Para $m = 1, \ldots, M$:
   (a) Se calcula el gradiente, para $i = 1, \ldots, N$:
   $$r_{im} = -\frac{\partial}{\partial f(x_i)} L(y_i, f(x_i))\Big|_{f(x)=f_{m-1}}.$$
   (b) Se ajusta un árbol $h_m(x)$ utilizando los gradientes anteriores.
   (c) Se escoge el paso de dirección al gradiente:
   $$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i)).$$
   (d) Se actualiza la estimación de $f(x)$:
   $$f_m(x) = f_{m-1}(x) + \gamma_m h_m(x).$$
3. Se obtiene la predicción final $\hat{f}(x) = f_{M-1}(x) + \gamma_M h_M(x)$ en el paso M.

- **XGBoost**

  El algoritmo Extreme Gradient Boosting (XGBoost) desarrollado por Chen and Guestrin (2016) surge como una mejora del método *Gradient Boosting*. En este método, se busca minimizar la siguiente función:

$$\mathcal{L}(\phi) = \sum_{i} L(y_i, \hat{y}_i) + \sum_{k} \Omega(f_k),$$

donde L es una función diferenciable que mide la diferencia entre el valor real y_i y

su predicción $\hat{y}_i$, mientras que Ω es la función de regularización dada por:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2,$$

donde T representa el número de hojas en el árbol, w_j denota los pesos asociados a cada hoja, y γ y λ son constantes de penalización utilizadas para mitigar el riesgo de sobreajuste.

El Algoritmo 2.3.3 describe a XGBoost, cuyo input es el conjunto de entrenamiento $\{(x_i, y_i)\}_{i=1}^n$, la función de pérdida presentada, un número M de iteraciones y $\alpha \in \mathbf{R}^+$ una tasa de aprendizaje:

Algoritmo 2.3.3 XGBoost

1. Se calcula:
$$\hat{f}_0(x) = \arg\min_{\theta} \sum_{i=1}^{n} L(y_i, \theta).$$
2. Para $m = 1, \ldots, M$:
   (a) para $i = 1, \ldots, n$:
$$\begin{aligned} \hat{g}_m(x_i) &= \left[\frac{\partial L(y_i, (x_i))}{\partial f(x_i)}\right]_{f(x)=\hat{f}_{(m-1)}(x)}, \\ \hat{h}_m(x_i) &= \left[\frac{\partial^2 L(y_i, f(x_i))}{\partial f(x_i)^2}\right]_{f(x)=\hat{f}_{(m-1)}(x)}. \end{aligned}$$
   (b) Se calcula el resultado de un árbol utilizando el conjunto $\left\{x_i - \frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)}\right\}_{i=1}^n$ al resolver el siguiente problema de optimización:
$$\begin{aligned} \hat{\phi}_m &= \arg\min_{\phi} \sum_{i=1}^{n} \frac{1}{2}\hat{h}_m(x_i)\left[-\frac{\hat{g}_m(x_i)}{\hat{h}_m(x_i)} + \phi(x_i)\right]^2, \\ \hat{f}_m(x) &= \alpha\hat{\phi}_m(x). \end{aligned}$$
   (c) Se actualiza la predicción:
$$\hat{f}_{(m)}(x) = \hat{f}_{(m-1)}(x) + \hat{f}_m(x).$$
3. Se obtiene la predicción final:
$$\hat{f}(x) = \hat{f}_{(M)}(x) = \sum_{m=0}^{M} \hat{f}_m(x).$$

2.3.3. Redes Neuronales Artificiales.

Las Redes Neuronales Artificiales (*Artificial Neural Networks*, ANN), inspiradas en las redes neuronales biológicas, son modelos computacionales fundamentales en la inteligencia artificial. Las primeras ANN fueron desarrolladas por McCulloch and Pitts (1943). En términos simples, una ANN es un conjunto de unidades interconectadas, llamadas neuronas o nodos, que procesan la información de entrada para aprender patrones y realizar predicciones precisas.

Cada nodo de una ANN contiene una función de entrada que combina los valores de entrada en un solo valor. Esta función de entrada se expresa mediante la siguiente fórmula:

$$I(\hat{x}) = \sum_{i=1}^{n} w_i x_i - \theta,$$

donde x_i representa los valores de entrada, w_i corresponde a los pesos que ponderan la influencia de estas entradas en el cálculo de la salida de cada neurona, n es el número total de valores de entrada, y θ es el sesgo del nodo. Además, cada nodo también cuenta con una función de activación que procesa $I(\hat{x})$ para calcular un nuevo valor, el cual es transmitido a las neuronas adyacentes.

Existen diferentes tipos de ANN, sin embargo, este trabajo se enfocará específicamente en las **redes neuronales *feedforward***, también conocidas como redes neuronales de propagación hacia adelante.

Como su nombre lo indica, las redes neuronales *feedforward* procesan señales de manera unidireccional y sin ciclos a través de la red, desde la capa de entrada, pasando por capas ocultas donde se realizan cálculos intermedios para aprender y representar patrones, hasta llegar a la capa de salida. Esto implica que durante cada iteración de entrenamiento, las señales de activación avanzan siempre hacia adelante y nunca retroceden para revisitar nodos previamente procesados. Además, no existen conexiones directas entre nodos dentro de la misma capa. La Figura 2.3.1 ilustra un ejemplo de una red *feedforward* con una capa oculta.

Una red neuronal que carece de nodos ocultos se conoce como perceptrón. En este tipo de redes, el proceso de aprendizaje es más simple, ya que implica asignar pesos aleatorios a los nodos de entrada y luego iterar hacia una solución modificando

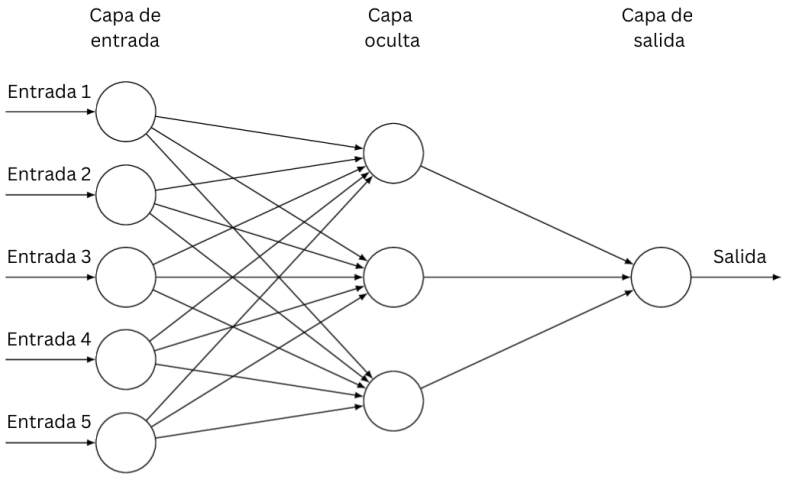


Figura 2.3.1: Estructura de una red neuronal *feedforward*.

estos pesos. Por otro lado, las redes neuronales con una o más capas ocultas son llamadas **Perceptrón Multicapa** (MLP por sus siglas en inglés). En la Figura 2.3.1 se muestra un MLP con una capa oculta.

A medida que la complejidad de la ANN aumenta, la tarea de actualizar los pesos se torna más desafiante, surgiendo así el algoritmo de ***backpropagation*** como una técnica efectiva. Este algoritmo tuvo su origen gracias a las contribuciones de Bryson and Ho (1969), y se popularizó con los avances realizados por Rumelhart et al. (1986).

El algoritmo *backpropagation* permite al MLP actualizar todos sus pesos con la condición de minimizar el error de salida. Usualmente se utiliza la siguiente función de error:

$$E = \frac{1}{2}\sum_{i=1}^{n} \|p_i - y_i\|^2,$$

donde p_i es el valor de salida predicho y y_i es el valor de salida objetivo de la instancia i del conjunto de entrenamiento.

Se utiliza el gradiente de la función de error para determinar la dirección óptima de ajuste. El gradiente se expresa como:

$$\nabla E = \left(\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_l}\right).$$

Calculando este gradiente para todos los l pesos en la red, es posible encontrar un mínimo de la función de error cuando $\nabla E = 0$. Cada peso w_i se actualiza utilizando la siguiente fórmula:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}.$$

Aquí, η representa la tasa de aprendizaje, que controla la velocidad a la que la red neuronal converge hacia una solución.

A continuación, se describe el algoritmo de *backpropagation* para un MLP:

1. **Propagación hacia adelante**: Los datos de entrada se introducen en la red, y en cada nodo de la capa de entrada y en cada uno de los nodos siguientes en las capas ocultas, se calcula la salida aplicando la función de activación a la suma ponderada de las entradas. Estas salidas se utilizan para generar las predicciones de la red.

2. **Cálculo del error en la capa de salida:** Se calcula el error en los nodos de salida comparando las salidas predichas con las salidas deseadas. Luego, se ejecuta la retropropagación hacia atrás utilizando las derivadas almacenadas en los nodos de salida. Este procedimiento se realiza para cada nodo de salida y en cada nodo j se calcula un valor de error retropropagado δ_j.

   $$\delta_c = (o_c - y_c) \cdot f'(z_c).$$

   Donde: o_c es el valor de salida del nodo actual en el paso de alimentación directa, y_c es el valor de salida deseado en el nodo actual, y $f'(\cdot)$ es la derivada de la función de activación aplicada en el punto z_c.

3. **Retropropagación hacia las capas ocultas:** El error retropropagado en cada nodo oculto c se calcula considerando los errores retropropagados de los nodos sucesores δ_s. Utilizando estos errores, el error retropropagado δ_c en el nodo actual se calcula como:

   $$\delta_c = f'(z_c) \sum_{s=1}^{S} w_{cs} \delta_s.$$

   Donde $f'(\cdot)$ es la derivada de la función de activación aplicada en el punto z_c, w_{cs} es el peso utilizado en la conexión entre el nodo sucesor s y el nodo actual c, y δ_s es el error retropropagado en el nodo sucesor s.

Luego, se utilizan estos errores para calcular las derivadas parciales de la función de error con respecto a los pesos de la conexión entre el nodo predecesor p y el nodo actual c:

$$\frac{\partial E}{\partial w_{pc}} = \delta_c o_p,$$

siendo o_p el valor de entrada recibido del nodo predecesor p, que también es el valor de salida del mismo nodo predecesor.

4. **Actualización de pesos:** Después de que se hayan calculado los valores de error retropropagados para todos los nodos, los pesos se actualizan en la dirección negativa del gradiente.

$$\Delta w_{pc} = -\eta \delta_c o_p,$$

donde η es la tasa de aprendizaje.

5. **Criterio de detención**: Los pasos anteriores se repiten iterativamente hasta que se alcance un criterio de convergencia, como una tolerancia predefinida o un número máximo de iteraciones (Rojas, 1996).

2.3.4. Máquinas de Vectores de Soporte.

Las Máquinas de Vectores de Soporte, comúnmente conocidas como SVM (por su nombre en inglés, *Support Vector Machine*), representan una técnica ampliamente utilizada para la clasificación binaria. Fueron desarrolladas por Cortes and Vapnik (1995), en los laboratorios de AT&T Bell. A continuación, se describirán los conceptos fundamentales de las SVM:

En un espacio de dimensión p, un hiperplano es un subespacio afín plano de dimensión $p-1$. Por ejemplo, en dos dimensiones, un hiperplano es un subespacio plano unidimensional, es decir, una línea. Matemáticamente, este ejemplo se define mediante el siguiente conjunto:

$$\{x = (x_1, x_2)^T : \beta_0 + \beta_1 x_1 + \beta_2 x_2 = 0\},$$

donde β_0, β_1 y β_2 son parámetros. Esto signfica que cualquier punto $x = (x_1, x_2)^T$ para el cual esta ecuación se cumple, representa un punto en el hiperplano. Lo

anterior se extiende fácilmente al entorno p-dimensional:

$$\{x = (x_1, x_2, \ldots, x_p)^T : \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0\}.$$

Por lo tanto, se puede interpretar el hiperplano como una división del espacio p-dimensional en dos mitades:

$$\{x \in \mathbb{R}^p = x^T\beta + \beta_0 < 0\}, \quad \text{y}$$
$$\{x \in \mathbb{R}^p = x^T\beta + \beta_0 > 0\},$$

donde $\beta = (\beta_1, \ldots, \beta_p)^T$ es un vector unitario y $\beta_0 \in \mathbb{R}$. Entonces, las máquinas de vectores de soporte consiste en elegir un hiperplano en un espacio de alta dimensionalidad, con el objetivo de separar de manera óptima los puntos pertenecientes a una clase de los pertenecientes a otra.

Una elección natural es el hiperplano de margen máximo, el cual representa el hiperplano de separación que se encuentra más lejos de las observaciones de entrenamiento. El margen se refiere a la distancia mínima entre las observaciones y este hiperplano. Por lo tanto, el hiperplano de margen máximo es aquel que maximiza esta distancia mínima.

Continuando con el plantamiento, se dispone de un conjunto de puntos de entrenamiento de tamaño N y p variables independientes. Cada dato es representado por un par (x_i, y_i), donde $x_i \in \mathbf{R}^p$, $y \in \{-1, 1\}$, para $i = 1, \ldots N$; siendo i el índice de la observación, x_i un vector con las variables de la i-ésima observación e y_i la clase a la cual pertenece.

Si las clases son linealmente separables, se desea encontrar un hiperplano tal que se pueda clasificar el punto x_i de acuerdo a la función:

$$f(x) = sign(x^T\beta + \beta_0) = \begin{cases} 1 & \text{si} \quad y_i = 1 \\ -1 & \text{si} \quad y_i = -1. \end{cases}$$

El hiperplano de margen máximo es la solución al problema de optimización:

$$\begin{aligned} & \underset{\beta, \beta_0, ||\beta||=1}{\text{máx}} \; M \\ \text{s.a} \quad & y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M, \; \forall i = 1, \ldots, n. \end{aligned}$$

donde M es el margen. Notar que $M = \frac{2}{\beta}$, por lo que maximizar el margen equivale a minimizar $f(M) = ||\beta||^2/2$. Si además se elimina la restricción sobre β, el problema equivale a:

$$\begin{aligned} & \min_{\beta,\beta_0} \frac{||\beta||^2}{2} \\ \text{s.a} \quad & y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq 1, \ \forall i = 1, \ldots, n. \end{aligned}$$

Si las clases no son linealmente separables, se pueden incorporar **variables de holgura** o ***slacks***, que sirven para aquellos datos que no pueden separarse perfectamente, permitiendo mediante penalizaciones que algunos puntos estén en el lado incorrecto del hiperplano de separación.

Las observaciones se pueden clasificar en separables si su variable de holgura correspondiente toma valor cero, si toma valor entre cero y uno, no separables pero correctamente clasificados y si es mayor que uno son ejemplos no separables y mal clasificados.

Sea $\xi = (\xi_1, \xi_2, \ldots, \xi_n)$ el vector de variables *slacks*. Luego, el problema toma la forma:

$$\begin{aligned} & \min_{\beta,\beta_0} \frac{||\beta||^2}{2} + C \sum_{i=1}^{n} \xi_i \\ \text{s.a} \quad & y_i(x_i^T \beta + \beta_0) \geq 1 - \xi_i, \ \forall i = 1, \ldots, n. \\ & \xi_i \geq 0, \ \forall i = 1, \ldots, n, \\ & \sum_{i=1}^{n} \xi_i \leq C; \end{aligned}$$

donde C es un parámetro de regularización. Cuando C es grande, el modelo SVM tiende a ajustarse mucho a los datos de entrenamiento, intentando clasificar cada punto correctamente, lo que puede llevar a una menor capacidad de generalización. Por otro lado, cuando C es pequeño, el modelo permite más errores de clasificación y tiene márgenes más amplios, lo que puede resultar en una clasificación más flexible y generalizable a nuevos datos.

Se tiene un problema cuadrático con restricciones lineales, que puede ser resuelto construyendo un Lagrangiano y transformándolo en el dual:

$$\begin{aligned} & \max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{s.a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0. \\ & 0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, n; \end{aligned}$$

donde $\alpha = (\alpha_1, \ldots \alpha_n)^n$ es un vector de multiplicadores de Lagrange. La función de decisión está dada por:

$$f(x) = sign(x^T \beta + \beta_0) = sign\left(\sum_{i=1}^{n} \alpha_i y_i \langle x, x_i \rangle \right).$$

Cuando el conjunto de datos no se puede separar por medio de una función lineal, se recurre a una nueva técnica consistente en la transformación del espacio original mediante una función no lineal hacia un espacio Hilbert dotado de un producto escalar, denominado **función kernel**.

Sea $\Phi : \mathcal{X} \longrightarrow \mathcal{F}$ la función de transformación que hace corresponder a cada vector de entrada x con un punto en el espacio de características $\mathcal{F}$, donde $\Phi(x) = (\phi_1(x), \ldots, \phi_p)$ y $\exists \ \phi_i(x)$ tal que $\phi_i(x)$ es una función no lineal. Por otra parte, kernel es una función $K : \mathcal{X} \times \mathcal{X} \longrightarrow \mathbb{R}$ tal que:

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle = \phi_1(x)\phi_1(x') + \cdots + \phi_p(x)\phi_p(x'), \quad \forall x, x' \in \mathcal{X}.$$

Boser et al. (1992) demostraron que para construir una función kernel no es necesario hacerlo a partir de un conjunto de funciones base $\Phi(x)$, sino que basta definir una función que sea simétrica y definida positiva. Así, las funciones de kernel transforman los datos originales en un espacio de características de mayor dimensionalidad donde la separación lineal es posible.

Algunos ejemplos de funciones Kernel, útiles en el contexto de las SVM, son:

- Polinomio de grado d: $K(x, x') = (1 + \gamma \langle x, x' \rangle)^d$,
- Base radial o gaussiano: $K(x, x') = \exp(-\gamma ||x - x'||^2)$, $\gamma > 0$,
- Red neuronal o sigmoidal: $K(x, x') = \tanh(\gamma \langle x, x' \rangle + r)$,

donde γ es el parámetro de ancho de banda del kernel. Volviendo a la formulación del problema no lineal, el planteamiento es análogo al anterior, sustituyendo el producto escalar por la función kernel:

$$\begin{aligned} & \max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \\ \text{s.a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0. \\ & 0 \leq \alpha_i \leq C, \ \forall i = 1, \ldots, n; \end{aligned}$$

y la función de decisión es (Campo, 2017):

$$f(x) = sign(x^t \beta + \beta_0) = sign\left(\sum_{i=1}^{n} \alpha_i y_i K(x_i, x_j) + \beta_0 \right).$$

2.3.5. AdaBoost

AdaBoost o *Adaptive Boosting*, propuesto por Freund and Schapire (1997), es un algoritmo de aprendizaje supervisado ampliamente utilizado para la clasificación. Es una implementación específica de la técnica de *boosting*, que se distingue por su enfoque adaptativo para ajustar los pesos de las muestras durante el entrenamiento.

El propósito de AdaBoost es construir incrementalmente un clasificador fuerte $G(x)$, optimizando los pesos y agregando un clasificador débil $G_m(x)$ uno a la vez, para $m = 1, \ldots, M$. Sea N es el número total de observaciones, y $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ el conjunto de datos de entrenamiento, donde x_i es el vector de características de la i-ésima observación, e $y_i \in \{-1, 1\}$ es su etiqueta de clase. El funcionamiento de AdaBoost se muestra en el Algoritmo 2.3.4.

Algoritmo 2.3.4 AdaBoost

1. Inicializar los pesos de las muestras: $w_i = \frac{1}{N}$ para $i = 1, \ldots, N$.
2. Para $m = 1$ hasta M:
   (a) Ajustar un clasificador débil $G_m(x)$ a los datos de entrenamiento usando pesos w_i.
   (b) Calcular el error ponderado del clasificador débil:
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \cdot I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i},$$
   donde $I(y_i \neq I(G_m(x_i)) = 1$ si $y_i \neq G_m(x_i)$ y 0 en caso contrario.
   (c) Calcular el peso asignado al clasificador débil:
   $$\alpha_m = \eta \log\left(\frac{1 - \text{err}_m}{\text{err}_m}\right),$$
   donde $\eta > 0$ es la tasa de aprendizaje.
   (d) Actualizar los pesos de las observaciones:
   $$w_i = w_i \cdot \exp(\alpha_m \cdot I(y_i \neq G_m(x_i)), \quad i = 1, \ldots, N.$$
3. Combinar los clasificadores débiles mediante una votación ponderada:
   $$G(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \cdot G_m(x)\right).$$

En resumen, el algoritmo funciona de la siguiente manera: se induce el clasificador débil actual $G_m(x)$ sobre las observaciones ponderadas. Luego, se calcula la tasa de error ponderado resultante, que se utiliza para determinar el peso α_m asignado a $G_m(x)$.

A continuación, se actualizan los pesos de las observaciones para la siguiente iteración. Aquellas observaciones mal clasificadas por $G_m(x)$ tienen sus pesos escalados por un factor $\exp(\alpha_m)$, lo que aumenta su influencia relativa para inducir el siguiente clasificador. En otras palabras, las observaciones mal clasificadas por $G_m(x)$ tienen un mayor peso en las iteraciones posteriores.

Finalmente, las predicciones de todos los clasificadores débiles se combinan mediante una mayoría ponderada de votos para producir la predicción final $G(x)$ (Hastie et al., 2009).

2.3.6. CatBoost

CatBoost (*Categorical Boosting*), desarrollado por la empresa Yandex gracias al trabajo de Prokhorenkova et al. (2018), es una biblioteca que implementa un algoritmo de *gradient boosting* basado en árboles de decisión.

A continuación, se resaltan las características distintivas de CatBoost en relación con otros modelos de boosting, lo que conlleva una explicación implícita de su funcionamiento.

- **Manejo de variables categóricas**

  CatBoost incorpora un tratamiento avanzado de variables categóricas mediante una técnica llamada *Target Statistics* (TS). Esta técnica se basa en reemplazar las categorías por estadísticas derivadas de la variable objetivo, lo que permite capturar de manera efectiva la relación entre las categorías y la variable objetivo.

  Para calcular estas estadísticas, CatBoost utiliza un enfoque de ordenamiento aleatorio, donde el conjunto de datos de entrenamiento se permuta aleatoriamente. Esta permutación asegura que las estadísticas de cada ejemplo se calculen usando solo la información de los ejemplos anteriores en la secuencia, evitando así la fuga de información.

- ***Boosting* ordenado**

  CatBoost se destaca entre otros modelos que utilizan *gradient boosting* por su capacidad para abordar el problema de la desviación de predicción inherente a este enfoque. Para comprender cómo lo aborda, es importante establecer ciertos puntos clave:

  Se considera el conjunto de datos de entrenamiento $D = \{(x_k, y_k)\}_{k=1}^{N}$, donde $x_k = (x_{1k}, \ldots, x_{pk})$ es un vector aleatorio de p características y $y_k \in \mathbb{R}$ es la variable objetivo, que puede ser binaria o una respuesta numérica. El objetivo de una tarea de aprendizaje es entrenar una función $F : \mathbb{R}^p \to \mathbb{R}$ que minimice la pérdida esperada $\mathcal{L}(F) := \mathbb{E}L(y, F(x))$, donde $L(.,.)$ es una función de pérdida suave.

  Como se presentó en la Sección 2.3.2, *gradient boosting* consiste en construir iterativamente una secuencia de aproximaciones $F_t : \mathbb{R}^p \to \mathbb{R}$, $t \geq 0$, tal que F_t se obtiene a partir de la aproximación anterior F_{t-1} de manera aditiva: $F_t = F_{t-1} + \eta h_t$, donde η es un tamaño de paso y la función $h_t : \mathbb{R}^m \to \mathbb{R}$ son los predictores base, en este caso árboles de decisión.

El problema de minimización suele abordarse tomando un paso de gradiente negativo. El paso de gradiente h_t se elige de tal manera que $h_t(x)$ se aproxime a $-g_t(x, y)$, donde $g_t(x, y) := \frac{\partial L(y,s)}{\partial s}|_{s=F_{t-1}(x)}$. Por lo general, se utiliza la aproximación de mínimos cuadrados:

$$h_t = \arg\min_{h \in H} \mathbb{E}\left[-g_t(x, y) - h(x)\right]^2. \tag{2.3.1}$$

En la práctica, suele aproximarse utilizando el mismo conjunto de datos D:

$$h_t = \arg\min_{h \in H} \frac{1}{N} \sum_{k=1}^{N} \left(-g_t(x_k, y_k) - h(x_k)\right)^2 \tag{2.3.2}$$

Tras analizar la Ecuación 2.3.2, Prokhorenkova et al. (2018) observaron que:

1. la distribución condicional del gradiente $g_t(x_k, y_k)|x_k$ (teniendo en cuenta la aleatoriedad de $D\backslash\{x_k\}$) está desplazada en comparación con la distribución en un ejemplo de prueba $g_t(x, y)|x$;
2. a su vez, el predictor base h_t definido por la Ecuación 2.3.1 está sesgado con respecto a la solución de la Ecuación 2.3.2;
3. esto, finalmente, afecta la capacidad de generalización del modelo entrenado F_t.

Estos problemas surgen debido a la fuga de objetivo, que ocurre cuando la información del objetivo se filtra en las características de entrenamiento. Los gradientes utilizados en cada paso se estiman utilizando los valores objetivo de los mismos puntos de datos en los que se construye el modelo actual F_{t-1}. Sin embargo, la distribución condicional $F_{t-1}(x_k)|x_k$ para un ejemplo de entrenamiento x_k generalmente difiere de la distribución $F_{t-1}(x)|x$ para un ejemplo de prueba x, lo que lleva al desplazamiento de predicción.

En CatBoost se implementó el método conocido como ***boosting* ordenado** para abordar este problema. El *boosting* ordenado consiste en introducir una permutación aleatoria del conjunto de datos en cada iteración y utilizar solo los datos anteriores a cada ejemplo en la permutación para el entrenamiento.

Se presenta el Algoritmo 2.3.5 que describe este procedimiento, donde I el número de árboles de decisión y $M_1, \ldots, M_N$ son modelos de soporte:

Algoritmo 2.3.5 *Boosting* Ordenado

1. Sea σ una permutación aleatoria de $[1, N]$.
2. $M_i = 0$ para $k = 1, \ldots, N$.
3. Para $t = 1$ hasta I:
   (a) Para $i = 1$ hasta N:
      i. $r_i \leftarrow y_i - M_{\sigma(i)-1}(x_i)$.
   (b) Para $i = 1$ hasta N:
      i. $\Delta M \leftarrow \text{LearnModel}((x_j, r_j) : \sigma(j) \leq i)$;
      ii. $M_i \leftarrow M_i + \Delta M$.
4. *return* M_N.

Notar que, r_i representa los residuos, y para calcular el residuo en un ejemplo, CatBoost utiliza un modelo entrenado sin él. Cada modelo se aprende utilizando solo los primeros i ejemplos de la permutación.

- **Construcción de árboles simétricos**

  CatBoost utiliza árboles simétricos como predictores base. Un árbol simétrico es un tipo de árbol de decisión donde se utiliza el mismo criterio de división en todo un nivel del árbol. Esto ofrece varias ventajas, como el balance, la resistencia al sobreajuste y la reducción significativa del tiempo de prueba (Prokhorenkova et al., 2018).

2.3.7. Bosques Aleatorios

Más conocido por su nombre en inglés *Random Forest* (RF), este algoritmo fue introducido por primera vez en un artículo escrito por Breiman (2001). Se trata de un algoritmo de aprendizaje automático supervisado utilizado tanto para problemas de clasificación como de regresión. En términos simples, es una técnica de ensamblaje que combina múltiples árboles de decisión durante el entrenamiento.

Para comprenderlo mejor, es necesario introducir el concepto de *bagging*.

- ***Bagging***

  Según lo planteado por James et al. (2013), La agregación *bootstrap*, o *bagging*, es una técnica fundamental en el aprendizaje estadístico, especialmente útil para reducir la varianza en métodos de aprendizaje automático, en particular los árboles de decisión.

  Para reducir la varianza y mejorar la precisión en el aprendizaje, se pueden construir múltiples modelos con distintos conjuntos de entrenamiento y luego se promedian

las predicciones. Sin embargo, utilizar múltiples conjuntos de entrenamiento es poco práctico. En su lugar, se recurre *bootstrap* para generar B muestras repetidas del conjunto de datos de entrenamiento, lo que resulta en B conjuntos bootstrap.

Después, para cada b-ésimo conjunto bootstrap, se entrena un árbol de decisión y se obtiene una función de predicción $\hat{f}^b(x)$. Luego, la predicción final $\hat{f}_{bagging}(x)$ se calcula como el promedio de todas las funciones de predicción:

$$\hat{f}_{bagging}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x).$$

La predicción final obtenida se conoce como *bagging*.

- ***Random Forest***

  El modelo RF (Breiman, 2001) es una modificación sustancial del *bagging* que construye una gran colección de árboles decorrelacionados y luego los promedia. En muchos casos, los bosques aleatorios tienen un rendimiento muy similar al del *boosting* y son más simples de entrenar y ajustar. Esto ha contribuido a su popularidad en el ámbito del aprendizaje automático.

  A continuación, se presenta el algoritmo de Bosque Aleatorio:

Algoritmo 2.3.6 Random Forest para Regresión o Clasificación

1. Para $b = 1$ hasta B:
   (a) Seleccionar una muestra bootstrap Z_b de tamaño N desde los datos de entrenamiento.

   (b) Entrenar un árbol T_b con la muestra bootstrap, repitiendo recursivamente los siguientes pasos para cada nodo terminal del árbol, hasta que se alcance el tamaño mínimo de nodo $n_{\min}$:
      i. Seleccionar m variables al azar de entre las p variables.
      ii. Elegir la mejor variable/punto de división entre las m variables.
      iii. Dividir el nodo en dos nodos hijo.

2. Obtener el conjunto de árboles $\{T_b\}_{b=1}^{B}$.

Para hacer predicción sobre una nueva observación x:

- *Regresión:* $\hat{f}_{\text{rf}}^{B}(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$
- *Clasificación:* Sea $\hat{C}_b(x)$ la predicción de clase del b-ésimo árbol de Random Forest. Luego $\hat{C}_{\text{rf}}^{B}(x) = \textit{voto mayoritario}\{\hat{C}_b(x)\}_{b=1}^{B}$.

La idea detrás de RF (Algoritmo 2.3.6) es mejorar la reducción de la varianza del *bagging* al disminuir la correlación entre los árboles. Esto se logra durante el proceso de crecimiento del árbol mediante la selección aleatoria de las variables de entrada. Específicamente, al construir un árbol con un conjunto *bootstrap*, se seleccionan aleatoriamente $m \leq p$ variables como candidatas para cada división. Usualmente, para problemas de clasificación se utiliza $m = \sqrt{p}$.

Un concepto relevante es la importancia de las variables, la cual revela la influencia de cada variable explicativa en la capacidad predictiva del modelo. En cada división de cada árbol, la mejora en el criterio de división es la medida de importancia atribuida a la variable de división, y esta se acumula en todos los árboles del bosque por separado para cada variable. Así, al final del entrenamiento, se obtiene una puntuación total de importancia para cada variable (Hastie et al., 2009).

2.3.8. Naive Bayes

El clasificador Naive Bayes (NB) es una técnica de aprendizaje supervisado ampliamente empleada, especialmente en tareas de clasificación de texto. Es capaz de manejar eficientemente tanto variables numéricas como categóricas.

Este método se fundamenta en el teorema de Bayes, un principio fundamental en la teoría de la probabilidad que proporciona un enfoque probabilístico para realizar predicciones basadas en la evidencia presente en los datos. Se considera un vector de características x con p dimensiones y una variable objetivo y. Según el teorema de Bayes, la probabilidad de clasificar x como perteneciente a la clase $c \in \{0, 1\}$, es decir, $y = c$, se puede expresar como:

$$P(y = c|x) = \frac{P(x|y = c) \cdot P(y = c)}{P(x)}.$$

Donde:

- $P(y = c|x)$ es la probabilidad de que la variable objetivo sea c dado el vector de características x, lo cual se conoce como la probabilidad a posteriori.
- $P(x|y = c)$ es la verosimilitud de observar el vector de características x dado que la variable objetivo es c.

- $P(y = c)$ es la probabilidad a priori de que la variable objetivo sea c.
- $P(x)$ es la probabilidad marginal de observar el vector de características x.

De esta manera, NB consiste en elegir aquella clase cuya probabilidad aposteriori sea mayor:

$$\hat{y} = \underset{c \in \{0,1\}}{\arg\max} \left(\frac{P(x|y = c) \cdot P(y = c)}{P(x)} \right).$$

Dado que $P(x)$ es independiente de c, esta expresión se simplifica como:

$$\hat{y} = \underset{c \in \{0,1\}}{\arg\max} \left(P(x|y = c) \cdot P(y = c) \right).$$

Por otro lado, El clasificador Naive Bayes asume que el efecto de una característica concreta en una clase es independiente de otras características. En términos matemáticos, esto se expresa como:

$$P(x|y = c) = \prod_{i=1}^{p} P(x_i|y = c)$$

Sin embargo, en escenarios del mundo real, es poco probable que esta suposición se cumpla por completo, lo que se considera ingenuo, de ahí el término 'naive'. A pesar de esta simplificación, el clasificador Naive Bayes aborda eficientemente problemas de clasificación, lo que lo hace más manejable desde el punto de vista computacional.

Aplicando este supuesto, la fórmula utilizada para la clasificación es la siguiente:

$$\hat{y} = \underset{c \in \{0,1\}}{\arg\max} \left(P(y = c) \prod_{i=1}^{p} P(x_i|y = c) \right),$$

donde $P(y = c)$, se calcula como la proporción de observaciones que pertenecen a la clase c, mientras que para calcular $P(x_i|y = c)$ en el caso de características numéricas, se modela la distribución de probabilidad de la característica x_i para cada clase c, generalmente utilizando una distribución gaussiana (Murphy, 2012).

2.4. Métricas de evaluación de modelos

En esta sección se exploran las principales métricas de evaluación utilizadas en los problemas de clasificación binaria, así como las métricas comúnmente empleadas para datos desbalanceados y en contextos financieros. Se abordan la interpretación de estas métricas y sus ventajas.

2.4.1. Matriz de confusión

Una matriz de confusión es una herramienta que permite visualizar el desempeño de un modelo de clasificación. En este caso, cuando la variable objetivo toma el valor 1, significa que el modelo clasificó la transacción como fraude. Por otro lado, si la variable objetivo toma el valor 0, indica que el modelo clasificó la transacción como normal o legítima.

Se considera la transacción fraudulenta como la clase positiva y la transacción normal como la clase negativa. La tabla 2.4.1 ejemplifica la matriz de confusión 2×2:

		Clase predicha	
		Negativa	Positiva
Clase real	Negativa	Verdaderos Negativos (VN)	Falsos Positivos (FP)
	Positiva	Falsos Negativos (FN)	Verdaderos Positivos (VP)

Tabla 2.4.1: Matriz de confusión.

Donde:

- **Verdaderos Negativos (VN)**: Número de transacciones normales que fueron correctamente clasificadas por el modelo.
- **Falsos Positivos (FP)**: Número de transacciones normales que fueron incorrectamente clasificadas como fraude.
- **Falsos Negativos (FN)**: Número de transacciones fraudulentas que fueron clasificadas de manera incorrecta como normales.
- **Verdaderos Positivos (VP)**: Número de transacciones fraudulentas que fueron clasificadas de manera correcta por el modelo.

Además de la matriz de confusión, se derivan otras métricas a partir de la información que esta proporciona, ofreciendo nuevas perspectivas para evaluar el rendimiento y la calidad del modelo.

2.4.2. Precisión

La precisión es la proporción de casos positivos que fueron correctamente identificados por el modelo con respecto al total de casos clasificados como positivos. Representa la probabilidad promedio de recuperar información relevante, indicando cuán interesantes y pertinentes son los resultados del modelo. (González, 2018). Matemáticamente, se expresa como:

$$\text{Precisión} = \frac{VP}{VP + FP}.$$

2.4.3. Exactitud

La exactitud o *accuracy* corresponde a la proporción de casos clasificados correctamente por el modelo con respecto al total de ejemplos. Evalúa la capacidad del modelo de clasificar de manera correcta las categorías de los casos positivos y negativos (González, 2018). Matemáticamente se define como:

$$\text{Exactitud} = \frac{VP + VN}{VP + VN + FP + FN}.$$

2.4.4. Sensibilidad

La sensibilidad, conocida en inglés como *Recall*, mide la capacidad de un modelo de clasificación para identificar todos los casos positivos reales en un conjunto de datos. Se calcula como la proporción de casos positivos que fueron correctamente identificados por el modelo, respecto al total de casos positivos reales:

$$\text{Sensibilidad} = \frac{VP}{VP + FN}.$$

2.4.5. Especificidad

Esta métrica mide la capacidad del modelo para identificar correctamente los casos negativos reales en un conjunto de datos. En otras palabras, la especificidad se refiere a la proporción de casos negativos reales que el modelo clasifica correctamente como negativos. La fórmula para calcular la especificidad es la siguiente:

$$\text{Especificidad} = \frac{VN}{VN + FP}.$$

2.4.6. Valor-F

El valor-F (o *F1-score*) es una métrica ampliamente reconocida para evaluar el rendimiento de un modelo en la clasificación entre la clase predicha y la clase real. Es especialmente útil cuando existe un desequilibrio entre las clases en los datos. El *F1-score* se calcula como la media armónica de la precisión y la sensibilidad, y se expresa de la siguiente manera:

$$\text{Valor-F} = 2 \cdot \frac{\text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}}.$$

2.4.7. Coeficiente de Correlación de Matthew

El coeficiente de correlación de Matthews, conocido por sus siglas MCC (del inglés *Matthews correlation coefficient*), se utiliza como medida de calidad en clasificaciones binarias. Toma en cuenta tanto los verdaderos como los falsos positivos y negativos, siendo especialmente útil cuando las clases tienen tamaños muy diferentes. Se calcula como:

$$MCC = \frac{VP \cdot VN - FP \cdot FN}{\sqrt{(VP + FP)(VP + FN)(VN + FP)(VN + FN)}}.$$

Este coeficiente toma valores en el rango de -1 a 1. Un valor de -1 indica un desacuerdo total entre las predicciones y las observaciones, mientras que un valor de 0 indica que las predicciones no son mejores que una elección aleatoria. Por otro lado, un valor de 1 representa una predicción perfecta (Cepeda, 2012).

2.4.8. Coeficiente de Gini

Para definir el coeficiente de Gini, primero es necesario entender el concepto de ROC y AUC:

El indicador *Receiver Operating Characteristics* (ROC) muestra la compensación entre la tasa de verdaderos positivos (TVP), también conocida como sensibilidad, y la tasa de falsos positivos (TFP). La TFP se calcula mediante la fórmula:

$$TFP = \frac{FP}{FP + VN} = 1 - \text{Especificidad}.$$

La Figura 2.4.1 presenta un gráfico bidimensional donde el eje vertical representa la TVP y el eje horizontal representa la TFP. En este contexto, la curva ROC, también conocida como Área Bajo la Curva (AUC), mide el área total bajo esta curva ROC y se utiliza para evaluar estadísticamente el desempeño de los modelos de clasificación binaria.

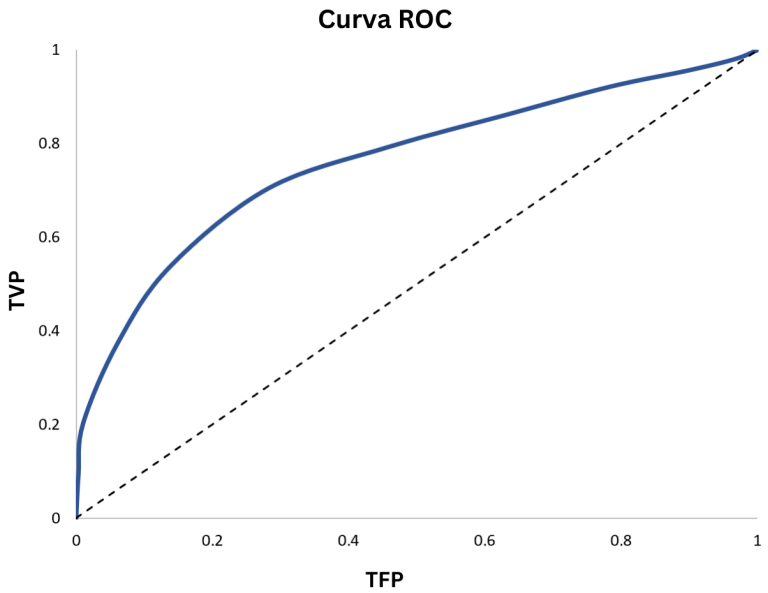


Figura 2.4.1: Ejemplo de curva ROC.

Un AUC alto indica un mejor rendimiento del modelo, ya que implica que el modelo puede discriminar de manera más efectiva entre las clases positivas y negativas. La tabla 2.4.2 proporciona intervalos de clasificación para los valores de AUC.

Niveles	Intervalos
Excelente	0.98 - 1.00
Muy Bueno	0.91 - 0.97
Bueno	0.76 - 0.90
Regular	0.75 - 0.61
Malo	0.50 - 0.60

Tabla 2.4.2: Intervalos de clasificación de valores AUC.
(Fuente: González (2018))

Una vez clarificado el AUC, se presenta el coeficiente de Gini, ampliamente utilizado en el contexto de modelos de riesgo. Este coeficiente mide la capacidad de discriminación del modelo al cuantificar la desigualdad en la distribución de las puntuaciones de predicción entre las clases positiva y negativa. Se calcula como:

$$\text{Gini} = 2 \cdot \text{AUC} - 1.$$

Un valor igual o superior a 0.5 indica un rendimiento satisfactorio del modelo. Cuanto más cercano a 1 esté el valor, mejor será la capacidad del modelo para clasificar correctamente (Gajowniczek et al., 2014).

2.4.9. Distancia de Kolmogorov-Smirnov

Conocido por sus siglas KS, el estadístico de Kolmogorov-Smirnov en el contexto de la curva ROC se utiliza para evaluar la capacidad discriminativa de un modelo de clasificación binaria. Se calcula como la máxima diferencia entre la Tasa de Verdaderos Positivos (TVP) y la Tasa de Falsos Positivos (TFP) (Khan, 2017):

$$KS = \text{máx}(TVP - TFP).$$

El KS toma valores entre 0 y 1, donde el valor 0 indica que la variable respuesta no discrimina y el valor 1 representa discriminación perfecta entre las clases positiva y negativa (Cepeda, 2012).

2.4.10. Métricas clave para la detección de fraude transaccional

La proporción de fraude en las transacciones suele ser considerablemente menor en comparación con el volumen total de transacciones. Por lo tanto, además de calcular las métricas comúnmente utilizadas en el contexto del aprendizaje supervisado, como la precisión y la exactitud, es importante incluir métricas que reflejen adecuadamente la capacidad predictiva del modelo, dando mayor énfasis a la detección de la clase positiva o fraude.

Una de las métricas principales a considerar es la sensibilidad, que mide la proporción de fraude correctamente clasificado en relación con el total de casos de fraude. Una alta sensibilidad demuestra que el modelo, a pesar del reducido número de fraudes, logró entrenarse para detectar la mayoría de las transacciones fraudulentas con éxito.

Sin embargo, también es importante acertar correctamente la clase negativa, ya que mantener una tasa reducida de falsos positivos suele ser una prioridad en las instituciones financieras. Según lo planteado por Oza (2018), "un sistema efectivo de detección de fraudes debería ser capaz de identificar transacciones fraudulentas con alta precisión y eficiencia. Si bien es crucial prevenir que los actores maliciosos lleven a cabo transacciones fraudulentas, también es fundamental garantizar que los usuarios legítimos no encuentren obstáculos para acceder al sistema de pagos. Un alto número de falsos positivos puede resultar en una mala experiencia para el cliente y llevarlos a buscar servicios en otro lugar".

Por lo tanto, otra métrica clave a tener en cuenta es el valor-F, especialmente útil cuando existe un desequilibrio entre las clases en los datos. Al considerar tanto la precisión como la sensibilidad, evita que una métrica se optimice a expensas de la otra, centrándose en el correcto acierto tanto de la clase positiva como negativa.

Capítulo 3

Implementación

3.1. Descripción de la base de datos

La muestra utilizada en este estudio se originó a partir de dos fuentes de información. La primera base de datos contiene el registro de todas las transacciones de comercio electrónico realizadas entre agosto de 2020 y junio de 2021, mientras que la segunda base de datos contiene las marcas de fraude o sospecha asociadas a estas transacciones. Es importante señalar que tanto los casos de fraude como los de sospecha fueron considerados como transacciones fraudulentas.

Durante los periodos mencionados, se registraron un total de 11.712.311 transacciones, de las cuales solo 24.314 fueron identificadas como fraude o sospecha, representando un 0,21 % del total.

Debido al reducido porcentaje de casos de fraude, el equipo encargado de desarrollar el modelo de predicción de fraude en la Entidad Financiera tomó la decisión de balancear la muestra mediante un submuestreo. Este proceso resultó en la selección de 242.190 registros, de los cuales 24.314 fueron clasificados como fraude o sospecha, representando ahora un 10,04 % del total.

La información disponible consistía en transacciones etiquetadas como fraude o no, cada una con su respectivo ID, fecha e información básica relacionada con el cliente. Cabe destacar que la variable que representa la etiqueta de fraude se llama FLAG_FRAUDE, la cual toma el valor 1 cuando se trata de una transacción fraudulenta y 0 cuando es una transacción legítima.

La Figura 3.1.1 ilustra el número de transacciones realizadas en cada mes, donde cada barra se divide en transacciones normales y fraudulentas, de acuerdo a la información proporcionada por FLAG_FRAUDE en cada transacción. En mayo del 2021 se alcanzó el máximo número de transacciones, totalizando 28.709, de las cuales 2.879 fueron identificadas como fraudulentas, lo que representa el 10,03 %. Se observa que la presencia de fraude se mantiene relativamente constante a lo largo de los meses, con el porcentaje de transacciones fraudulentas manteniéndose cerca del 10 % en cada mes.

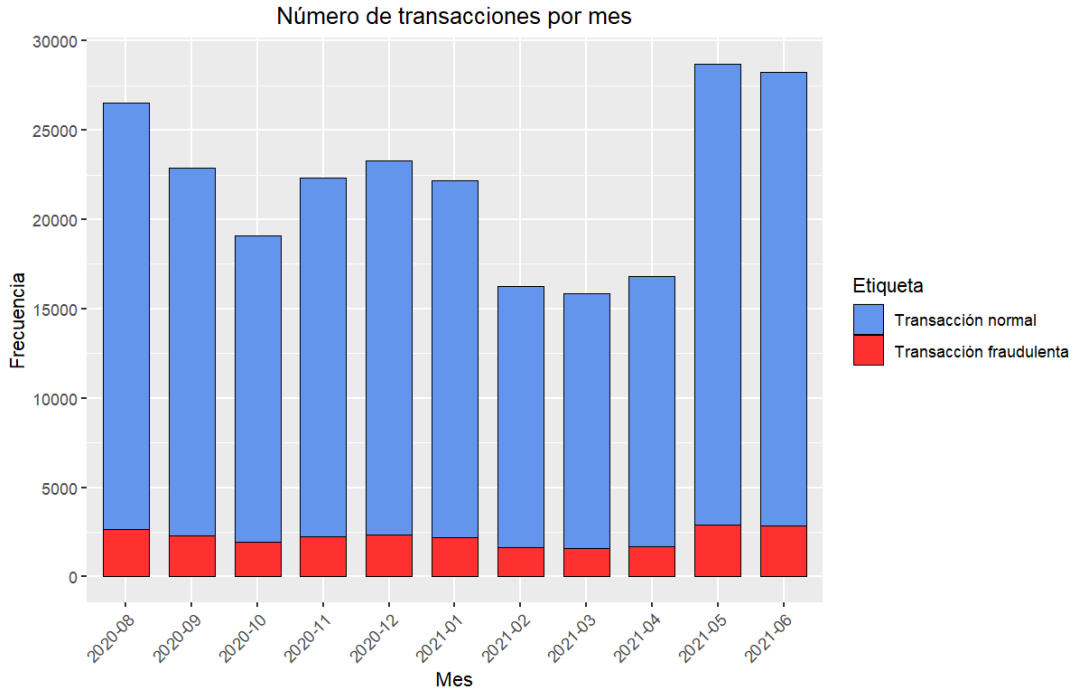


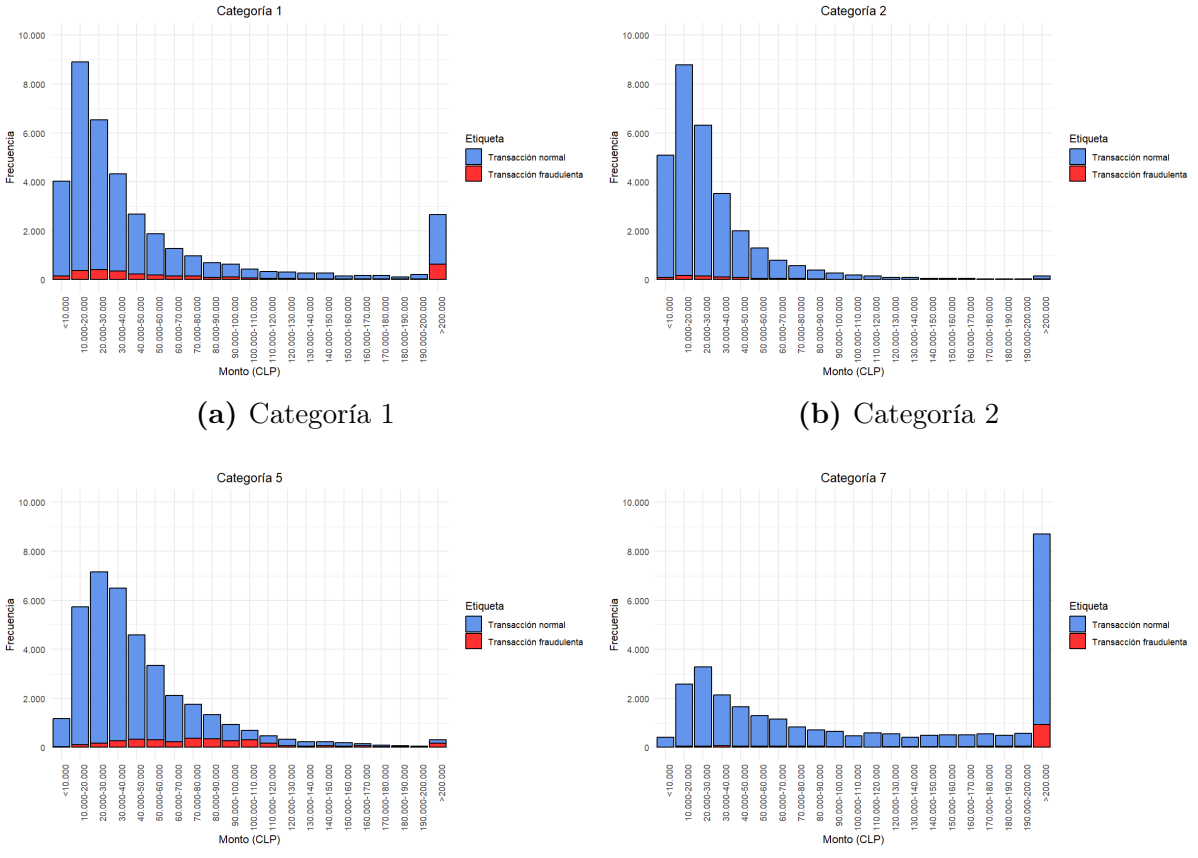
Figura 3.1.1: Frecuencia de transacciones normales y fraudulentas por mes

A partir de estos registros de transacciones, la entidad financiera generó 117 variables numéricas que formarán parte de los modelos que se presentarán más adelante. Estas variables se pueden clasificar según el tipo de información que proporcionan. Como ejemplo, algunas de las categorías de variables son:

- **Importe de la transacción**: El monto asociado a las operaciones se clasifica según la categoría a la que pertenecen los productos involucrados. Las categorías son las siguientes:
  - Categoría 1: Hombres, juvenil hombres, deportes.
  - Categoría 2: Damas, juvenil mujer.

- Categoría 3: Ropa interior, accesorios mujer, perfumería.
- Categoría 4: Niños.
- Categoría 5: Calzado.
- Categoría 6: Electrohogar, telefonía.
- Categoría 7: Electrohogar, electro blanco.
- Categoría 8: Blanco, menaje, decoración, regalos, gourmet y concesiones.

En la Figura 3.1.2, se muestra la frecuencia de transacciones para distintos rangos de dinero y para cada cuatro categorías de productos.



(a) Categoría 1 **(b)** Categoría 2

(c) Categoría 5 **(d)** Categoría 7

Figura 3.1.2: Número de transacciones por rango de importe total.

En la mayoría de las categorías expuestas, se observa que el mayor número de transacciones corresponde a montos pequeños, y lo mismo ocurre con las transacciones fraudulentas. Sin embargo, destaca un patrón distinto en la subfigura 3.1.2d. Dado que se trata de productos tecnológicos, estos suelen tener costos más elevados. Esto

lleva a una concentración de fraude en transacciones de montos considerables y con mayor frecuencia, ya que los productos más caros son más atractivos para actividades maliciosas.

- **Historial de transacciones**: Estas variables ofrecen información sobre el número de transacciones, el número de productos transaccionados, o la proporción del monto de la transacción en relación con el monto promedio o el monto total; ya sea en las últimas 24 horas, 7 días o 30 días; para un mismo cliente, misma dirección de envío, correo, teléfono o *fingerprint*. Estas variables son generales o específicas por categoría de producto.

  A modo de ejemplo, la figura 3.1.3 presenta la frecuencia de transacciones con respecto al número de productos de la categoría 5 en las últimas 24 horas con la misma dirección de envío. Se observa que la mayoría de transacciones fraudulentas involucran poco número de productos en este caso.

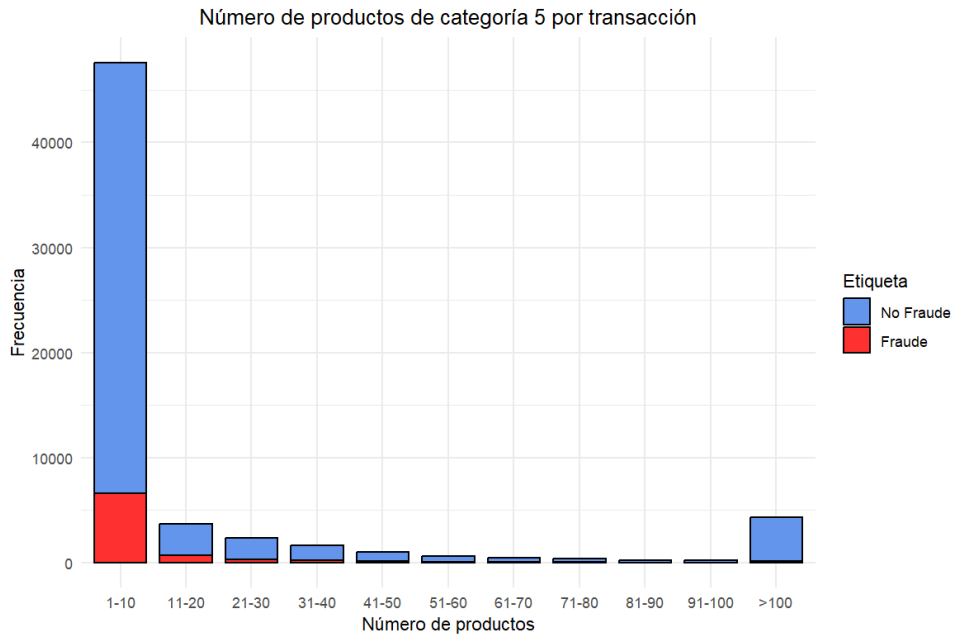
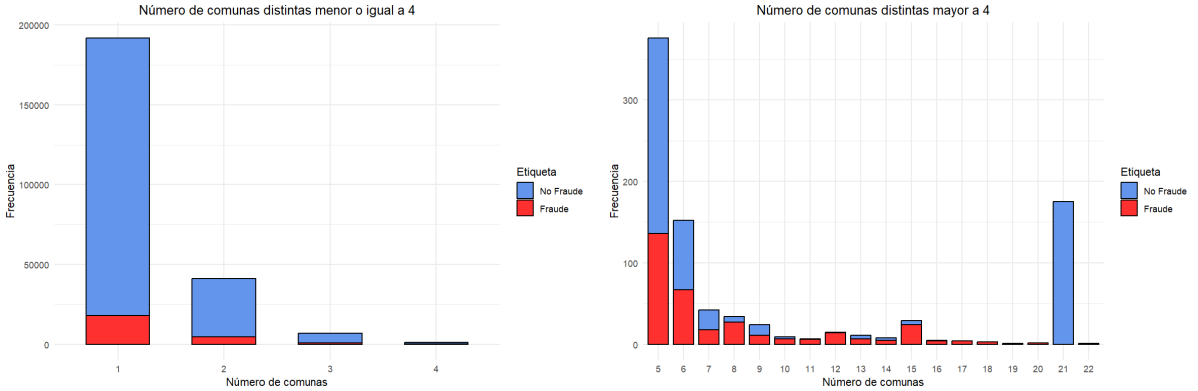


Figura 3.1.3: Número de productos de categoría 5 en las ultimas 24 horas con la misma dirección de envío

- **Discrepancias**: Estas variables señalan si hay diferencias entre las direcciones, correos y comunas informados por el cliente en el último mes, además de indicar si existen discrepancias de franja horaria en las transacciones recientes.



(a) 4 o menos comunas distintas **(b)** Más de cuatro comunas distintas

Figura 3.1.4: Número de transacciones según diferencias en las comunas.

En particular, la figura 3.1.4 presenta la variable que indica si ha habido un cambio de comuna informado por el cliente durante la transacción en relación con los últimos 30 días. Esta variable se divide en dos subfiguras debido a que la mayoría de las transacciones tienen valores bajos en dicha variable, indicando que hay pocos cambios de comunas.

En la subfigura 3.1.4b, se observa que, aunque el número de transacciones con múltiples cambios de comuna para un mismo cliente es bajo, la proporción de fraude tiende a ser significativamente alta.

3.1.1. Tratamiento de datos faltantes

La muestra presenta un considerable número de datos faltantes, y la mayoría de las funciones utilizadas para ajustar los modelos carecían de herramientas de imputación. Por esta razón, después de analizar la naturaleza de cada variable, se optó por imputar los valores faltantes con ceros al aplicar los modelos de clasificación. Se hace una excepción para el modelo XGBoost, el cual cuenta con herramientas incorporadas para manejar automáticamente los valores nulos.

3.1.2. Partición de la base de datos

Para el desarrollo del modelo, se optó por asignar aleatoriamente el 20 % de los datos como muestra de prueba, mientras que el 80 % restante se dividió en un 70 % para entrenamiento y un 30 % para la validación de parámetros. Así, la muestra total se segmentó en tres submuestras, con las siguientes características:

- Datos de entrenamiento: Contiene 135.626 registros, con una proporción de 10,04 % de FLAG_FRAUDE.
- Datos de validación: Consiste en 58.126 registros, con una proporción de 9,99 % de FLAG_FRAUDE.
- Datos de prueba: Comprende 48.438 registros, con una proporción de 10,10 % de FLAG_FRAUDE.

3.2. Aplicación de modelos

3.2.1. Regresión Logística

De acuerdo a lo expuesto en la sección 2.3.1.1, se aplicaron las cuatro metodologías propuestas para el análisis y selección de variables. Estas técnicas permiten identificar las variables que contribuyen significativamente al modelo y a su capacidad predictiva, ofreciendo distintos enfoques para seleccionar las más relevantes en el contexto de la RL.

El primer paso consistió en calcular el VIF de cada variable explicativa y descartar aquellas con un VIF superior a 10, pues este valor sugiere una fuerte colinealidad. Esto redujo el número de variables de 117 a 65.

Posteriormente, se aplicó el método de *backward* al modelo resultante para eliminar aquellas variables que podrían no aportar información significativa, lo que redujo el número de variables a 52.

Luego, se filtró según el coeficiente de Kolmogorov-Smirnov, manteniendo aquellas variables cuyo estadístico KS es superior a 0.2. Finalmente, para las 14 variables restantes después del filtrado por KS, se calculó el *Information Value* de cada una, elgiendo aquellas con un IV superior a 0,25, lo que resultó en una selección final de 12 variables.

Con las variables definidas, se aplicó el modelo RL en el entorno `R` y las métricas resultantes después del entrenamiento y la predicción con los datos de prueba se encuentran en la Tabla 3.2.1.

Métrica	**LR**
Precisión	0.6708
Exactitud	0.9098
Sensibilidad	0.2084
Especificidad	0.9885
Valor-F	0.3180
MCC	0.3404
Gini	0.6644
KS	0.5286

Tabla 3.2.1: Métricas de regresión logística.

		Predicha	
		Negativo	Positivo
Real	Negativo	43048	500
	Positivo	3871	1019

Tabla 3.2.2: Matriz de confusión del modelo de regresión logística.

Asimismo, se incluye la matriz de confusión 3.2.2, en la cual se observa que el número de falsos positivos es muy bajo. Esto indica que el modelo permitiría que las transacciones normales se procesen casi sin interrupciones. Sin embargo, el número de falsos negativos es muy extremadamente elevado, lo que señala una deficiencia del modelo en la detección de fraudes.

3.2.2. XGBoost

Este clasificador corresponde al elegido por la Institución Financiera, que servirá como modelo de referencia para futuras comparaciones. A continuación, se presentan los resultados obtenidos tras el proceso de entrenamiento del modelo XGBoost en `R` sobre las 117 variables, y la evaluación de sus predicciones utilizando los datos de prueba.

El proceso de entrenamiento se realizó replicando los procedimientos llevados a cabo por la Entidad Financiera. Los parámetros utilizados durante este proceso se detallan en la Tabla A1.1 del Apéndice. El reprocesamiento de XGBoost se llevó a cabo con el objetivo de explorar su capacidad predictiva y compararla con otros modelos de clasificación propuestos.

La Tabla 3.2.3 despliega las métricas de desempeño, mientras que la Tabla 3.2.4 muestra la matriz de confusión.

Métrica	**XGBoost**
Precisión	0.7909
Exactitud	0.9357
Sensibilidad	0.4935
Especificidad	0.9853
Valor-F	0.6077
MCC	0.5938
Gini	0.8381
KS	0.6928

Tabla 3.2.3: Métricas del modelo XGBoost.

		Predicha	
		Negativo	Positivo
Real	Negativo	42910	638
	Positivo	2477	2413

Tabla 3.2.4: Matriz de confusión del modelo XGBoost.

3.2.3. Redes Neuronales Artificiales

La implementación de ANN en el problema de detección de fraude transaccional se llevó a cabo en `Python`, utilizando la función `MLPClassifier` de la biblioteca `scikit-learn`. Las ANN requieren que los datos sean previamente estandarizados para funcionar correctamente, ya que esto ayuda a reducir el impacto de las diferencias en la escala y magnitud de las variables en los datos (Ibáñez, 2023). La estandarización se realizó mediante la importación de `StandardScaler`, función también perteneciente a `scikit-learn`.

3.2.3.1. Ajuste de hiperparámetros

Durante el proceso, se exploraron diversas combinaciones de parámetros y se realizaron diversas evaluaciones. A continuación, se describirá la metodología empleada y se examinarán los modelos más destacados de cada configuración de parámetros.

a. Optimizador: Durante el entrenamiento, se utilizó el algoritmo `Adam` para optimizar los pesos de la red neuronal. `Adam` es un optimizador estocástico basado en gradiente propuesto por Kingma and Ba (2014). Este optimizador se distingue por su eficacia, especialmente en problemas que involucran grandes conjuntos de datos y/o numerosos parámetros.

Asimismo, se realizaron pruebas con el optimizador `SGD` (Descenso de Gradiente Estocástico); sin embargo, los resultados no fueron favorables. La principal limitación de `SGD` fue que la red proporcionaba resultados aceptables solo para configuraciones muy reducidas de capas y neuronas. En contraste, `Adam` demostró mayor flexibilidad al ajustar estos parámetros, como se evidenciará más adelante.

b. Funciones de activación: Se evaluaron dos funciones de activación para las capas ocultas. La primera es la función logística sigmoide (`logistic`), definida como:

$$f(x) = \frac{1}{1 + \exp(-x)},$$

y la segunda es la función unitaria lineal rectificada (`ReLU`), definida como:

$$f(x) = \text{máx}(0, x).$$

c. Configuración de capas y nodos

En este ítem, se prueban distintas estructuras de redes neuronales utilizando el optimizador `Adam` y ambas funciones de activación mencionadas.

c.1 Redes neuronales con capas descendentes.

Una estructura usual en las redes neuronales es la de capas descendentes, donde las capas iniciales tienen un mayor número de nodos para capturar características más generales y simples, mientras que las capas posteriores tienen menos nodos para enfocarse en detalles más específicos y discriminativos.

Como notación, se ejemplifica con la siguiente estructura de red neuronal: $(10, 5)$, la cual significa que la red cuenta con dos capas ocultas, teniendo 10 neuronas en la primera capa oculta y 5 en la segunda.

Se entrenaron las siguientes combinaciones de capas ocultas: $(10, 5)$, $(10, 5, 2)$, $(50, 10)$, $(100, 50)$, $(150, 100, 50)$, $(200, 150, 100, 50)$ y $(250, 200, 150, 100, 50)$;

utilizando tanto la función de activación `logistic` como `ReLU`. Se optó por la combinación $(250, 200, 150, 100, 50)$ con función de activación `ReLU`, debido a que, al evaluar el rendimiento del modelo con datos de validación, esta configuración obtuvo el mejor valor-F y una sensibilidad alta. A dicha red se le llamará ANN-1. Las métricas se encuentran disponibles en la Tabla 3.2.5.

Función de activación	**Configuración ANN**	**Sensibilidad**	**Valor-F**
`logistic`	(10,5)	0.4338	0.5521
	(10,5,2)	0.4586	0.5680
	(50,10)	0.4968	0.5799
	(100,50)	0.5120	0.5708
	(150,100,50)	0.4903	0.5643
	(200,150,100,50)	0.5165	0.5789
	(250,200,150,100,50)	0.5602	0.5919
`ReLU`	(10,5)	0.4382	0.5588
	(10,5,2)	0.4615	0.5711
	(50,10)	0.5061	0.5925
	(100,50)	0.5731	0.5717
	(150,100,50)	0.5476	0.5919
	(200,150,100,50)	0.5326	0.5960
	(250,200,150,100,50) (ANN-1)	**0.5361**	**0.5983**

Tabla 3.2.5: Métricas de ANN con capas descendentes.

c.2 Redes neuronales con capas ascendentes.

En una estructura ascendente, las capas iniciales cuentan con menos nodos y aprenden representaciones simples, mientras que las capas profundas las combinan para formar características más complejas. Este enfoque es beneficioso para resolver problemas complejos y manejar datos de alta dimensionalidad. Sin embargo, este tipo de estructura puede requerir más tiempo computacional durante el entrenamiento del modelo.

Se realizaron entrenamientos con diversas combinaciones de capas ocultas, incluyendo $(5, 10)$, $(2, 5, 10)$, $(10, 50)$, $(50, 100)$, $(50, 100, 150)$, $(50, 100, 150, 200)$ y $(50, 100, 150, 200, 250)$. En esta instancia, la combinación $(50, 100, 150)$ destacó al lograr el mayor valor-F, por lo que fue seleccionada. La red elegida se identifica como ANN-2. Las métricas correspondientes se detallan en la Tabla 3.2.6.

Función de activación	Configuración ANN	Sensibilidad	Valor-F
`logistic`	(5,10)	0.4660	0.5670
	(2,5,10)	0.4493	0.5409
	(10,50)	0.4562	0.5617
	(50,100)	0.5084	0.5644
	(50,100,150)	0.5294	0.5804
	(50,100,150,200)	0.5254	0.5726
	(50,100,150,200,250)	0.5120	0.5749
`ReLU`	(5,10)	0.5140	0.5937
	(2,5,10)	0.4469	0.5371
	(10,50) (ANN-2)	**0.5094**	**0.5982**
	(50,100)	0.5333	0.5814
	(50,100,150)	0.5292	0.5767
	(50,100,150,100)	0.5109	0.5741
	(50,100,150,200,250)	0.5268	0.5870

Tabla 3.2.6: Métricas de ANN con capas ascendentes.

c.3 Redes neuronales con capas uniformes.

Se experimentó con configuraciones de redes en las que el número de neuronas permaneció constante en cada capa oculta. La repetición de nodos en varias capas facilita el aprendizaje de representaciones intermedias complejas y específicas en distintas etapas; sin embargo, esto conlleva a un mayor costo computacional.

Se entrenaron redes con estructuras de tres y cuatro capas, variando el número de nodos en cada capa entre 10, 50, 100, 150 y 200. Esto resultó en un total de 10 configuraciones de redes. Sin embargo, cada configuración fue evaluada dos veces, una utilizando la función de activación `logistic` y otra utilizando `relu`. Por lo tanto, en total se evaluaron 20 modelos.

La combinación que obtuvo mejores resultados con los datos de validación fue la red neuronal $(200, 200, 200, 200)$ con función de activación `ReLU`, de acuerdo a la Tabla 3.2.7. El modelo de red neuronal elegido con esta configuración de nodos repetidos se denomina ANN-3.

Función de activación	Configuración ANN	Sensibilidad	Valor-F
logistic	(10,10,10)	0.4632	0.5661
	(50,50,50)	0.4827	0.5671
	(100,100,100)	0.5127	0.5623
	(150,150,150)	0.5929	0.5832
	(200,200,200)	0.5140	0.5744
	(10,10,10,10)	0.4298	0.5479
	(50,50,50,50)	0.4767	0.5561
	(100,100,100,100)	0.4972	0.5572
	(150,150,150,150)	0.5385	0.5785
	(200,200,200,200)	0.4973	0.5726
ReLU	(10,10,10)	0.4536	0.5700
	(50,50,50)	0.5289	0.5776
	(100,100,100)	0.5263	0.5851
	(150,150,150)	0.5426	0.5955
	(200,200,200)	0.5034	0.5882
	(10,10,10,10)	0.5049	0.5964
	(50,50,50,50)	0.4884	0.5640
	(100,100,100,100)	0.5101	0.5795
	(150,150,150,150)	0.5192	0.5872
	(200,200,200,200) (ANN-3)	**0.5359**	**0.5997**

Tabla 3.2.7: Métricas de ANN con capas uniformes.

d. Validación cruzada

A continuación, se realizó una validación cruzada para identificar los valores óptimos de dos parámetros clave en la configuración de la red neuronal.

El primero es `alpha`, que corresponde al parámetro de regularización L2 y controla la penalización aplicada a los pesos grandes en la red neuronal. El segundo parámetro, `learning_rate_init`, establece la tasa de aprendizaje inicial, determinando el tamaño de los pasos dados durante la optimización de los pesos de la red.

Por defecto, en los modelos anteriores se utilizaron los valores predeterminados `alpha`=0,0001 y `learning_rate_init`=0,001. Para los modelos ANN-1, ANN-2 y ANN-3, se llevó a cabo una búsqueda en cuadrícula utilizando tres valores diferentes para `alpha` y tres valores diferentes para `learning_rate_init`, lo que resulta en un total de nueve escenarios por modelo. La Tabla 3.2.8 presenta las combinaciones de hiperparámetros que se probarán.

Esta búsqueda se llevó a cabo utilizando una validación cruzada de 5 particiones, y se evaluó el rendimiento del modelo para cada combinación de hiperparámetros utilizando la métrica *f1-score*.

Escenario	Hiperparámetros	
	`alpha`	`learning_rate_init`
1	10^{-5}	10^{-4}
2		10^{-3}
3		10^{-2}
4	10^{-4}	10^{-4}
5		10^{-3}
6		10^{-2}
7	10^{-3}	10^{-4}
8		10^{-3}
9		10^{-2}

Tabla 3.2.8: Búsqueda en cuadrícula de hiperparámetros para ANN.

Así, se generaron tres nuevos modelos basados en la validación cruzada, cuyas métricas al realizar predicciones con respecto a los datos de prueba se encuentran en la Tabla 3.2.9:

Modelo ANN	Hiperparámetros		Sensibilidad	Valor-F
	`alpha`	`learning_rate_init`		
ANN-1	10^{-4}	10^{-2}	0.5241	0.6160
ANN-2	10^{-4}	10^{-3}	0.5076	0.5953
ANN-3	10^{-3}	10^{-3}	0.5372	0.6097

Tabla 3.2.9: Métricas de ANN con validación cruzada de `alpha` y `learning_rate_init`.

3.2.3.2. Selección modelo ANN

En resumen, se exploraron diversas configuraciones de capas ocultas, de las cuales se seleccionaron tres modelos de cada categoría. Posteriormente, se llevó a cabo una búsqueda en cuadrícula con validación cruzada para ajustar los parámetros de regularización L2 y la tasa de aprendizaje inicial, lo que dio como resultado los tres modelos finales, cuyos parámetros y métricas principales se presentaron en la Tabla 3.2.9. Ahora se presenta la Tabla 3.2.10 que contiene la totalidad de métricas para los tres modelos, evaluados en los datos de prueba.

Se destaca que el modelo ANN-3 posee la mayor sensibilidad, mientras que el modelo ANN-1 exhibe el mayor *f1-score*. Considerando el conjunto de otras métricas, el modelo ANN-1 muestra un desempeño superior, lo cual fundamenta su elección como la mejor opción. La Tabla 3.2.11 muestra la matriz de confusión del modelo ANN-1.

Métricas	**ANN-1**	**ANN-2**	**ANN-3**
Precisión	0.7470	0.7196	0.7047
Exactitud	0.9340	0.9303	0.9306
Sensibilidad	0.5241	0.5076	0.5372
Especificidad	0.9801	0.9778	0.9747
Valor-F	0.6160	0.5953	0.6097
MCC	0.5921	0.5686	0.5786
Gini	0.8274	0.8232	0.7304
KS	0.6926	0.6758	0.6212

Tabla 3.2.10: Métricas finales de los modelos ANN.

		Predicha	
		Negativo	Positivo
Real	Negativo	42680	868
	Positivo	2408	2563

Tabla 3.2.11: Matriz de confusión del modelo ANN final.

3.2.4. Máquinas de Vectores de Soporte

Inicialmente, se aplicaron SVM utilizando `R`, específicamente la función `svm` de la biblioteca `e1071`. No obstante, para explorar alternativas adicionales, se llevaron a cabo pruebas en `Python` utilizando las funciones `SVC` y `LinearSVC` de la biblioteca `scikit-learn`. Esta decisión se tomó con la intención de ampliar las opciones disponibles y evaluar si había margen para optimizar la velocidad de los resultados obtenidos inicialmente en `R`.

Es importante destacar que, durante las pruebas en `Python`, se observó una mejora significativa al estandarizar previamente los datos utilizando `StandardScaler` de `scikit-learn`. En contraste, en el análisis realizado en `R`, los datos se utilizaron en su forma original, sin requerir una estandarización previa.

3.2.4.1. Ajuste de hiperparámetros

La intención es realizar una búsqueda en cuadrícula de hiperparámetros, explorando diversos valores del parámetro de regularización `C` que varíen en escala logarítmica. Para el parámetro de ajuste de kernel `gamma`, se adopta una estrategia similar, incluyendo el valor predeterminado de gamma:

$$\frac{1}{\text{N° de características}} = \frac{1}{117} \approx 0{,}009.$$

Luego, se prueban valores próximos a este, en un total de 9 escenarios, según se muestra en la tabla 3.2.12.

Escenario	**Hiperparámetros**	
	`C`	`gamma`
1	1	0,001
2		0,009
3		0,1
4	10	0,001
5		0,009
6		0,1
7	100	0,001
8		0,009
9		0,1

Tabla 3.2.12: Búsqueda en cuadrícula de hiperparámetros para SVM.

Cabe mencionar que esta búsqueda en cuadrícula se aplica a SVM con kernel radial, polinomial y sigmoidal. Para las SVM lineales, el parámetro `gamma` no es aplicable, por lo que solo se prueban valores de `C`: 0,1, 1, 10 y 100.

- **R**

  En `R`, se realizó una búsqueda en cuadrícula utilizando el kernel radial (RBF) para SVM, que es capaz de modelar relaciones no lineales entre los datos. Después de entrenar los modelos en `R`, se realizaron predicciones con los datos de validación, obteniendo las métricas que se muestran en la tabla 3.2.13.

SVM kernel radial		**Sensibilidad**	**Valor-F**
`C`=1	`gamma`=0.001	0.3600	0.4888
`C`=1	`gamma`=$\frac{1}{117}$	0.4109	0.5378
`C`=1	`gamma`=0.1	0.2613	0.3875
`C`=10	`gamma`=0.001	0.3845	0.5198
`C`=10	`gamma`=$\frac{1}{117}$	0.4260	0.5545
`C`=10	`gamma`=0.1	0.2500	0.3695
`C`=100	`gamma`=0.001	0.3893	0.5281
`C`=100	`gamma`=$\frac{1}{117}$	0.3953	0.5261
`C`=100	`gamma`=0.1	0.1981	0.3056

Tabla 3.2.13: Métricas de SVM con kernel radial en `R`.

Desde la tabla anterior se observa que la SVM de kernel radial, con `C`=10 y `gamma`=$\frac{1}{117}$, muestra la mayor sensibilidad y el valor-F más alto, con valores del 42,6 % y 55,45 %,

respectivamente. Esto sugiere que el modelo exhibe una precisión moderada, con una baja cantidad de falsos positivos.

Cabe destacar que se llevaron a cabo algunas pruebas con kernel polinomial y lineal en R. Sin embargo, debido al prolongado tiempo de computación y a los resultados deficientes obtenidos, se decidió cancelar la búsqueda en cuadrícula para estos kernels.

- **Python**

  En esta sección se describe el proceso realizado en `Python`. En primer lugar, cabe destacar que, al igual que en las ANN, se llevó a cabo una búsqueda en cuadrícula con validación cruzada para encontrar la combinación óptima de parámetros que maximizara el *F1-score*, utilizando los parámetros especificados en la Tabla 3.2.12. Adicionalmente, se estable un número máximo de iteraciones que el algoritmo de entrenamiento de SVM realizará antes de detenerse.

  Inicialmente, se intentó entrenar modelos con kernel polinomial de grados 2 y 3. Sin embargo, los resultados obtenidos fueron insatisfactorios, ya que etiquetaba a la gran mayoría de observaciones como fraude. En el apéndice A2.1 se presenta una tabla con las métricas que respaldan la validación del modelo.

  Posteriormente, se llevó a cabo una búsqueda en cuadrícula con validación cruzada utilizando los kernels lineal, radial y sigmoidal. La Tabla 3.2.14 contiene los valores de sensibilidad y valor-F de los modelos, tras predecir sobre el conjunto de validación:

Kernel	**Hiperparámetros**		**Sensibilidad**	**Valor-F**
Lineal	`C`=100		0.2792	0.4054
Radial	`C`=10	`gamma`=0.001	0.5988	0.4948
Sigmoidal	`C`=1	`gamma`=0.001	0.3258	0.4115

Tabla 3.2.14: Métricas de SVM con distintos kernel en `Python`.

  Como resultado, se determinó que la SVM con kernel radial, utilizando los parámetros `C`=10 y `gamma`=0,001, proporciona el mejor rendimiento.

3.2.4.2. Selección modelo SVM

En resumen, se realizaron entrenamientos de SVM mediante una búsqueda en cuadrícula de los hiperparámetros `C` y `gamma`, para distintos kernel. Este proceso se llevó a cabo tanto en `R` como en `Python`, donde se aplicaron técnicas de validación cruzada en este último.

Así, se seleccionaron los mejores modelos:

- SVM-1: Máquina de vectores de soporte entrenada en `R`, con kernel radial, `C`=10 y `gamma`=$\frac{1}{117}$.
- SVM-2: Máquina de vectores de soporte entrenada en `Python`, con kernel radial, `C`=10 y `gamma`=0,001.

Se presentan todas sus métricas en la Tabla 3.2.15, luego de evaluar los modelos con los datos de prueba. Adicionalmente, se muestran las matrices de confusión de los modelos (Tablas 3.2.16 y 3.2.17).

Métrica	**SVM-1**	**SVM-2**
Precisión	0.8024	0.4274
Exactitud	0.9330	0.8783
Sensibilidad	0.4468	0.6033
Especificidad	0.9876	0.9092
Valor-F	0.5740	0.5003
MCC	0.5683	0.4417
Gini	0.7528	0.6597
KS	0.6465	0.5339

Tabla 3.2.15: Métricas finales de los modelos SVM.

Al comparar las métricas presentadas, se evidencia una clara ventaja del modelo SVM-1 sobre el modelo SVM-2, con la excepción de la sensibilidad. Al examinar la matriz de confusión 3.2.17, se confirma la alta sensibilidad del modelo SVM-2, ya que predice mucho más de la mitad de las 4890 observaciones de fraude. Sin embargo, la cantidad de falsos positivos es alarmante, lo que se refleja en sus resultados deficientes en las demás métricas.

		Predicha	
		Negativo	Positivo
Real	Negativo	43010	538
	Positivo	2705	2185

Tabla 3.2.16: Matriz de confusión del modelo SVM-1.

		Predicha	
		Negativo	Positivo
Real	Negativo	39595	3953
	Positivo	1940	2950

Tabla 3.2.17: Matriz de confusión del modelo SVM-2.

De acuerdo con lo observado en la matriz de confusión 3.2.16, en el modelo SVM-1 apenas 538 transacciones normales son incorrectamente clasificadas como fraude, lo que explica sus altas métricas.

En conclusión, se elige el modelo SVM-1, ya que, a pesar de sus deficiencias en la detección del fraude, su baja tasa de falsos positivos garantiza una mejor experiencia para el cliente al no interferir en sus transacciones.

Sin embargo, es importante destacar que la librería de `Python` ofrece la posibilidad de reducir el tiempo de ejecución mediante la opción de establecer un límite en el número de iteraciones. Esto significa que el algoritmo puede detenerse incluso si no ha convergido completamente. Aunque esto afectó a los resultados, también permitió explorar una variedad de kernels debido a los tiempos de ejecución más cortos.

3.2.5. AdaBoost

La implementación del clasificador AdaBoost se llevó a cabo utilizando `Python`, específicamente mediante la función `AdaBoostClassifier` del módulo `scikit-learn`.

3.2.5.1. Ajuste de hiperparámetros

Se realizó una búsqueda en cuadrícula de hiperparámetros para optimizar los valores de `n_estimators` y `learning_rate` del modelo AdaBoost. `n_estimators` define el número de árboles utilizados en el proceso de *boosting*, mientras que `learning_rate` controla el peso aplicado a cada clasificador en cada iteración. La Tabla 3.2.18 muestra las combinaciones de parámetros evaluadas.

Escenario	**Hiperparámetros**	
	`n_estimators`	`learning_rate`
1	50	0,01
2		0,1
3		1,0
4	100	0,01
5		0,1
6		1,0
7	500	0,01
8		0,1
9		1,0

Tabla 3.2.18: Búsqueda en cuadrícula de hiperparámetros para AdaBoost.

Se utiliza validación cruzada de k pliegues para evaluar cada combinación de hiperparámetros, calculando el rendimiento mediante *F1-score*. Luego, se selecciona la mejor combinación según esta métrica. Así, los parámetros finales del modelo serían `n_estimators`=500 y `learing_rate`=1.

3.2.5.2. Selección modelo AdaBoost

La Tabla 3.2.19 muestra todas las métricas del modelo AdaBoost final después de realizar predicciones sobre los datos de prueba. También se adjunta su matriz de confusión 3.2.20.

Métrica	**AdaBoost**
Precisión	0.7647
Exactitud	0.9300
Sensibilidad	0.4427
Especificidad	0.9847
Valor-F	0.5608
MCC	0.5490
Gini	0.8183
KS	0.6725

Tabla 3.2.19: Métricas del modelo AdaBoost.

		Predicha	
		Negativo	Positivo
Real	Negativo	42882	666
	Positivo	2725	2165

Tabla 3.2.20: Matriz de confusión del modelo AdaBoost.

Este modelo destaca por su bajo número de falsos positivos, lo que significa que son pocas las transacciones legítimas que son clasificadas erróneamente. Sin embargo, su sensibilidad indica que solo detecta el 44 % del fraude.

3.2.6. CatBoost

Se aplicó el algoritmo CatBoost utilizando la biblioteca del mismo nombre en `Python`. Los parámetros seleccionados se detallan en la Tabla A3.1 del Apéndice.

A continuación, se muestra la Tabla 3.2.21 que incluye las métricas después de realizar predicciones sobre los datos de prueba, junto con su correspondiente matriz de confusión 3.2.22.

Métrica	**CatBoost**
Precisión	0.8260
Exactitud	0.9425
Sensibilidad	0.5448
Especificidad	0.9871
Valor F	0.6566
MCC	0.6428
Gini	0.8627
KS	0.7237

Tabla 3.2.21: Métricas del modelo CatBoost.

		Predicha	
		Negativo	Positivo
Real	Negativo	42987	561
	Positivo	2226	2664

Tabla 3.2.22: Matriz de confusión del modelo CatBoost.

Los resultados del clasificador CatBoost son altamente alentadores. En particular se destaca por su alta precisión, así como por el valor alcanzado en los coeficientes Gini y KS, que subrayan la eficacia del modelo en discriminar entre las clases positiva y negativa.

3.2.7. Bosques Aleatorios

En esta sección, se detalla la implementación del modelo RF en `R`, utilizando una librería `RandonForest`. Los parámetros empleados durante el entrenamiento se enumeran en la Tabla A4.1 del apéndice. Posteriormente, se exhibe la Tabla de métricas 3.2.23 y la Tabla matriz de confusión 3.2.23 obtenidas tras evaluar el rendimiento del modelo con los datos de prueba.

Métrica	**RF**
Precisión	0.8088
Exactitud	0.9356
Sensibilidad	0.4740
Especificidad	0.9874
Valor-F	0.5977
MCC	0.5892
Gini	0.7769
KS	0.6663

Tabla 3.2.23: Métricas del modelo RF.

		Predicha	
		Negativo	Positivo
Real	Negativo	43000	548
	Positivo	2572	2318

Tabla 3.2.24: Matriz de confusión del modelo RF.

Este modelo cuenta con una alta precisión, lo que significa que tiene una capacidad notable para clasificar correctamente las transacciones en general. Aproximadamente el 80 % de las transacciones identificadas como fraudulentas por este modelo son genuinamente fraudulentas.

Adicionalmente, en el Apéndice se adjunta Tabla A4.2, que presenta los 15 valores de importancia alcanzados por las 15 mejores variables. La importancia se determinó mediante el algoritmo Random Forest, utilizando el promedio de la disminución en el índice de Gini. Esta medida refleja las variables más relevantes para la clasificación en el modelo.

3.2.8. Naive Bayes

Se empleó el clasificador NB para abordar el problema de detección de fraude transaccional, utilizando el entorno de programación `R` y la librería `e1071`. Cabe mencionar que la implementación tradicional de NB, no hay hiperparámetros por ajustar. Se presenta la Tabla de métricas 3.2.25 y la Tabla matriz de confusión 3.2.26.

Métrica	**NB**
Precisión	0.3906
Exactitud	0.8694
Sensibilidad	0.5245
Especificidad	0.9081
Valor F	0.4478
MCC	0.3808
Gini	0.6474
KS	0.5250

Tabla 3.2.25: Métricas del modelo NB.

El modelo exhibe una sensibilidad moderada (52.45 %); no obstante, el resto de las métricas son deficientes, como se evidencia en la matriz de confusión. Esta situación se atribuye al elevado número de falsos positivos.

		Predicha	
		Negativo	Positivo
Real	Negativo	39547	4001
	Positivo	2325	2565

Tabla 3.2.26: Matriz de confusión del modelo NB.

3.3. Comparación de modelos

Se presenta una Tabla comparativa 3.3.1 con la mejor opción de cada clasificador propuesto, tras evaluarlos en un mismo conjunto de prueba.

Modelo	**Precisión**	**Exactitud**	**Recall**	**Especificidad**	**Valor-F**	**MCC**	**Gini**	**KS**
RL	0.6708	0.9098	0.2084	0.9885	0.3180	0.3404	0.6644	0.5286
XGBoost	0.7909	0.9357	0.4935	0.9853	0.6077	0.5938	0.8381	0.6928
ANN	0.7470	0.9340	0.5241	0.9801	0.6160	0.5921	0.8274	0.6926
SVM	0.8024	0.9330	0.4468	0.9876	0.5740	0.5683	0.7528	0.6465
AdaBoost	0.7647	0.9300	0.4427	0.9847	0.5608	0.5490	0.8183	0.6725
CatBoost	**0.8260**	**0.9425**	**0.5448**	**0.9871**	**0.6566**	**0.6428**	**0.8627**	**0.7237**
RF	0.8088	0.9356	0.4740	0.9874	0.5977	0.5892	0.7769	0.6663
NB	0.3906	0.8694	0.5245	0.9081	0.4478	0.3808	0.6474	0.5250

Tabla 3.3.1: Comparación de métricas entre los modelos de aprendizaje automático.

A partir de la Tabla 3.3.1 se observa que en las métricas de precisión, exactitud, sensibilidad, valor-F, MCC, coeficiente de Gini y KS, CatBoost obtivo el mayor valor, mientras que en la especificidad, el mayor valor lo obtuvo RL. Estos resultados sugieren que el modelo CatBoost generalmente tuvo el mejor rendimiento en la mayoría de las métricas evaluadas en comparación con los otros modelos de aprendizaje automático.

Capítulo 4

Discusión

En este capítulo se analizan los resultados obtenidos tras evaluar el desempeño de diferentes clasificadores: Regresión Logística, XGBoost, Redes Neuronales Artificiales, Máquinas de Vectores de Soporte, AdaBoost, CatBoost, Bosques Aleatorios y Naive Bayes. Estos resultados se contextualizan en relación con la literatura existente sobre detección de fraude transaccional y se ofrece una interpretación crítica de su relevancia.

4.1. Interpretación de resultados

Es importante recordar que el objetivo de esta Memoria de Título es aplicar diversas metodologías de aprendizaje automático en los datos de transacciones proporcionados por la Entidad Financiera, con el fin de evaluar su rendimiento, rentabilidad y compararlos.

Se aplicaron estos clasificadores utilizando el mismo conjunto de datos de transacciones proporcionado por la Entidad Financiera. Cada modelo se entrenó, validó y evaluó bajo las mismas condiciones para garantizar comparaciones equitativas, con la excepción de la Regresión Logística, donde no se utilizaron todas las 117 variables disponibles.

En el capítulo de Implementación, se presentan los resultados de diversas métricas para varios modelos de aprendizaje automático (ver Tabla 3.3.1), y se destacan los siguientes puntos:

4.1.1. Modelos de referencia

Pese a que se simplificó el problema reduciendo el número de características, el modelo RL exhibió el peor rendimiento, con una alta especificidad (98.85 %) pero una sensibilidad extremadamente baja (20.84 %); es decir, clasifica indiscriminadamente casi todas las transacciones como legítimas.

Por otro lado, el modelo XGBoost, utilizado por la Institución Financiera, mostró resultados favorables en métricas como precisión y exactitud. Los lineamientos para la construcción de modelos de la Institución se centran en alcanzar valores sólidos de KS y Gini, ya que estos reflejan la capacidad predictiva del modelo. En este caso, los valores obtenidos se sitúan dentro de un rango considerado aceptable según los criterios establecidos.

4.1.2. Modelo con mejor desempeño

Los resultados revelan que el modelo CatBoost destaca en la mayoría de las métricas evaluadas, abarcando tanto las métricas estándar de precisión (82.60 %) y exactitud (94.25 %), así como aquellas más especializadas para contextos de datos desbalanceados, como el valor-F (65.66 %) y MCC (0.6428). Además, CatBoost exhibe un rendimiento sobresaliente en términos del coeficiente de Gini (0.8627) y KS (0.7237), lo que señala su excepcional capacidad para discernir entre transacciones fraudulentas y legítimas.

Aunque CatBoost mostró una especificidad ligeramente menor en comparación con los modelos RL, SVM y RF, su tasa de falsos positivos sigue siendo baja, lo que respalda su buen desempeño en otras métricas.

Estos resultados sugieren que CatBoost es altamente efectivo para clasificar transacciones, siendo crucial en entornos financieros donde minimizar los falsos negativos y positivos es de suma importancia para la satisfacción del cliente y la prevención del fraude.

4.1.3. Modelos con peor desempeño

A pesar de que la mayoria de los modelos obtienen resultados aceptables, algunos modelos como RL y NB mostraron un desempeño significativamente inferior, este último con una precisión de 39.06 % y un valor-F de 44.78 %. Esto podría deberse a la inadecuación del modelo para este tipo de problema, ya sea por las suposiciones simplificadas y su sensibilidad a ciertas características del conjunto de datos.

4.2. Consideraciones de implementación y eficiencia

En esta sección se discuten las consideraciones de implementación y la eficiencia de los diferentes modelos de aprendizaje automático utilizados en el estudio. Se destacan los desafíos encontrados durante el proceso de implementación y se proporciona un análisis del tiempo de ejecución de los modelos.

4.2.1. Consideraciones de implementación

En el proceso de implementación de los modelos de aprendizaje automático, surgieron diversas consideraciones que influenciaron tanto el diseño de la metodología como la eficiencia de los modelos.

Respecto al resto de metodologías empleadas, se encontró necesario realizar una búsqueda más exhaustiva de los hiperparámetros más convenientes para los modelos ANN, SVM y AdaBoost, lo que implicó mayor trabajo a la hora de decidir qué direcciones tomar, considerando en algunos casos el número elevado de parámetros y opciones posibles.

Por otro lado, los modelos RF y CatBoost no requirieron de un ajuste exhaustivo para obtener buenos resultados. Variar sus hiperparámetros principales generalmente conllevaba un aumento en el tiempo de ejecución y/o diferencias poco significativas en los resultados.

4.2.2. Análisis del tiempo de ejecución

La Tabla 4.2.1 muestra el tiempo aproximado de ejecución de los códigos de la versión final de cada modelo. Esto también es un factor importante a la hora de evaluar la rentabilidad de los modelos.

Modelo	**Tiempo de ejecución**
RL	1 min
XGBoost	7 min
ANN	22 min
SVM	12 hrs
AdaBoost	6 min
CatBoost	5 min
RF	16 min
NB	3 min

Tabla 4.2.1: Tiempo de ejecución de los modelos de aprendizaje automático.

Se observa que SVM fue, por mucho, el modelo más lento. No obstante, las pruebas realizadas en `Python` demostraron que su velocidad en este entorno mejoraba significativamente, aunque a expensas de una precisión menor en la detección de fraude. Estos hallazgos sugieren que las SVM podrían beneficiarse de implementaciones alternativas en Python para futuras aplicaciones.

Aunque ANN y Adaboost no requirieron tanto tiempo de cómputo según la Tabla 4.2.1, aquí sólo se refleja el tiempo necesario para entrenar el modelo con la combinación de hiperparámetros final. Dado que se probaron muchas combinaciones, en ANN, SVM y Adaboost se entrenaron al menos 9 modelos previos a elegir los hiperparámetros finales, por lo que implementar estas metodologías resultó ser bastante lento, en especial las dos primeras.

4.3. Comparación con estudios previos

En esta sección se comparan los resultados obtenidos con la literatura existente, especialmente con aquellas publicaciones descritas en la Tabla 1.2.1 del Estado del arte. En dicha tabla se observa que los resultados sobre el modelo ganador varían entre los distintos artículos, lo que motivó la exploración de diversas metodologías en este estudio.

Los modelos presentados en esta Memoria de Título muestran una exactitud similar a los estudios realizados por Shpyrko and Koval (2019) y Trivedi et al. (2020). Sin embargo, comparado otros estudios previos, métricas como el valor-F y el *recall* son ligeramente inferiores. Esta diferencia puede atribuirse a varias razones. Primero, los datos utilizados en este estudio cuentan con un 10 % de fraude; es decir, están moderadamente desbalanceados, y en algunas investigaciones suelen probar distintas técnicas de muestreo para equilibrar las clases, muchas veces igualando la cantidad de transacciones legítimas y fraudulentas, como se observa en Gupta et al. (2023). Esto se hace porque los modelos de aprendizaje automático, en algunas ocasiones, pueden verse sesgados hacia la clase mayoritaria.

Cabe mencionar que en este trabajo se priorizó obtener un bajo número falsos positivos, ya que en el contexto de las instituciones bancarias, la clasificación errónea de transacciones legítimas causa insatisfacción al cliente, lo que se considera más perjudicial que el fraude en sí mismo (Abdallah et al., 2016). Es posible que en otros estudios se haya priorizado disminuir los falsos negativos, aunque esto aumente el número de falsos positivos.

Además, la limitación de recursos computacionales y el tiempo disponible para el ajuste de hiperparámetros de los modelos convencionales, como SVM y ANN, pudieron restringir el desempeño óptimo de los modelos.

CatBoost sigue la tendencia observada en otros estudios, donde los modelos basados en *boosting* tienden a superar a otros modelos en tareas de clasificación complejas, en concordancia con Chen and Han (2021), Gedela and Karthikeyan (2022) y Gupta et al. (2023). Sin embargo, el rendimiento de modelos como RF y ANN, aunque sea ligeramente inferior en este caso, siguen siendo competitivos y pueden ofrecer ventajas en otros contextos.

Capítulo 5

Conclusión

En el transcurso de esta Memoria de Título, se ha llevado a cabo un exhaustivo desarrollo y evaluación de diversos sistemas de detección de fraude transaccional, haciendo uso de una variedad de modelos de aprendizaje automático.

Se realizó una detallada revisión de estudios previos relacionados con el tema. Durante este análisis, se observó claramente el creciente interés en el uso de técnicas de aprendizaje automático para la detección de fraudes, un fenómeno impulsado por el vertiginoso avance tecnológico y el aumento del volumen de transacciones electrónicas. Además, se llevó a cabo una investigación teórica profunda sobre los modelos seleccionados para esta tarea. Este proceso no solo contribuyó a un mayor entendimiento del funcionamiento de cada modelo, sino que también facilitó la optimización de los parámetros para lograr el mejor rendimiento posible.

Gracias a los recursos proporcionados por una Institución Financiera Chilena, se dispuso de datos de transacciones reales basados en el comportamiento de gasto del cliente, que incluían información sobre el monto de la transacción, los intervalos de tiempo desde la última compra, la categoría del artículo, la dirección del cliente, entre otros. Durante la aplicación de las metodologías propuestas, se enfrentaron diversos desafíos inherentes a la complejidad del problema, como el desbalanceo de clases, la presencia de datos faltantes, la calibración de hiperparámetros y el tiempo de cómputo considerable.

Tras evaluar los modelos con diversas métricas, se llevó a cabo el objetivo principal de esta Memoria: un análisis comparativo entre los resultados de los modelos de aprendizaje automático, contrastándolos con modelos base como la Regresión Logística y el modelo de la Institución XGBoost. Cada decisión tomada estuvo orientada a obtener clasificadores

capaces de lograr un equilibrio entre la predicción precisa de fraudes y la preservación de la validez de las transacciones legítimas.

Los resultados obtenidos son prometedores y representan una contribución significativa al análisis de fraudes transaccionales en este entorno financiero. Destaca especialmente la eficacia del modelo CatBoost sobre los otros modelos. Esta innovadora variación del *gradiente boosting* ofrece una gran precisión predictiva, mediante la incorporación técnicas como el *boosting* ordenado y un control eficaz del sobreajuste. Además, en la práctica demostró ser un clasificador versátil, rápido y relativamente sencillo de aplicar.

A pesar de las limitaciones encontradas, el estudio proporciona una base sólida para futuras aplicaciones. Los modelos presentados lograron un desempeño competitivo y ofrecieron perspectivas valiosas para la implementación de sistemas de detección de fraude transaccional en entornos financieros reales.

5.1. Trabajos futuros

En futuras investigaciones, resultaría relevante explorar técnicas adicionales para abordar el desbalance de clases, como el sobremuestreo. Asimismo, sería beneficioso probar diversos métodos de selección de variables con el fin de analizar y mitigar el posible *overlapping* (superposición entre clases). Se podría, además, desarrollar métodos más robustos para manejar los datos faltantes, además de considerar el uso de diferentes bibliotecas o librerías para explorar una mayor variedad de ajustes de hiperparámetros. Sería interesante también analizar ensambles de modelos complejos de aprendizaje supervisado, como se ha realizado en estudios previos (Khalid et al., 2024).

Por otro lado, se sugiere investigar la integración de técnicas de aprendizaje profundo, especialmente en contextos con datos secuenciales. Métodos como las redes neuronales recurrentes muestran promesas significativas en la detección del fraude en transacciones y podrían ser explorados en trabajos futuros.

Bibliografía

Abdallah, A., Maarof, M. A., and Zainal, A. (2016). Fraud detection system: A survey. *Journal of Network and Computer Applications*, 68:90–113.

Aerospike (2022). *PayPal tackles fraud prevention with data.* https://aerospike.com/resources/case-study/paypal-fraud-prevention-with-data/.

Banco Central de Chile (2023). *Informe de Sistemas de Pago.* https://www.bcentral.cl/documents/33528/4387486/Informe-de-Sistemas-de-Pago-agosto-2023.pdf/.

Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. pages 144–152.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Bryson, A. E. and Ho, Y.-C. (1969). *Applied Optimal Control.* Wiley.

Campo, E. (2017). *Introducción a las máquinas de vector soporte (SVM) en aprendizaje supervisado.* (Trabajo de Grado), Universidad de Zaragoza.

Cepeda, D. (2012). *Detección de fraude en tarjetas de crédito.* (Trabajo de grado), Universidad de Chile, Chile.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794.

Chen, Y. and Han, X. (2021). Catboost for fraud detection in financial transactions. In *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, pages 176–179. IEEE.

Cornfield, J., Gordon, T., and Smith, W. N. (1961). Quantal response curves for experimentally uncontrolled variables. *Bulletin of the International Statistical Institute*, 38:97–115.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20:273–297.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232.

Gajowniczek, K., Zabkowski, T., and Szupiluk, R. (2014). Estimating the roc curve and its significance for classification models' assessment. *Quantitative Methods in Economics*, 15(2):382–391.

Gedela, B. and Karthikeyan, P. R. (2022). Credit card fraud detection using adaboost algorithm in comparison with various machine learning algorithms to measure accuracy, sensitivity, specificity, precision and f-score. In *2022 International Conference on Business Analytics for Technology and Security (ICBATS)*, pages 1–6. IEEE.

GlobalData (2022). *Total Payment Volume on PayPal: Global (Q3 2015 – Q1 2022)*. https://www.globaldata.com/data-insights/financial-services/total-payment-volume-on-paypal-global/.

González, E. (2018). *Detección de Fraude en Tarjetas de Crédito Mediante Técnicas de Minería de Datos*. (Trabajo de grado), Universidad Santo Tomás, Colombia.

Gupta, P., Varshney, A., Khan, M. R., Ahmed, R., Shuaib, M., and Alam, S. (2023). Unbalanced credit card fraud detection data: a machine learning-oriented comparative study of balancing techniques. *Procedia Computer Science*, 218:2575–2584.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, volume 2. Springer, New York.

Ibáñez, J. (2023). *Por qué la normalización es clave e importante en Machine Learning y Ciencia de Datos*. https://jorgeiblanco.medium.com/por-qu%C3%A9-la-normalizaci%C3%B3n-es-clave-e-importante-en-machine-learning-y-ciencia-de-datos-4595f15d5be0.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An Introduction to Statistical Learning: with Applications in R*. Springer.

Khalid, A. R., Owoh, N., Uthmani, O., Ashawa, M., Osamor, J., and Adejoh, J. (2024). Enhancing credit card fraud detection: an ensemble machine learning approach. *Big Data and Cognitive Computing*, 8(1):6.

Khan, R. (2017). *Binary classifier evaluation metrics: error rate, KS statistic, AUROC, lift, gains table*. https://rstudio-pubs-static.s3.amazonaws.com/303414_fb0a43efb0d7433983fdc9adcf87317f.html. South Dakota State University.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lin, A. Z. (2018). Using information value, information gain and gain ratio for detecting two-way interaction effect. In *SAS Global Forum 2018 Conference Proceedings*.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115–133.

Moreira, M. L., Junior, C. D. S. R., de Lima Silva, D. F., de Castro Junior, M. A. P., de Araújo Costa, I. P., Gomes, C. F. S., and dos Santos, M. (2022). Exploratory analysis and implementation of machine learning techniques for predictive assessment of fraud in banking systems. *Procedia Computer Science*, 214:117–124.

Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.

Oza, A. (2018). Fraud detection using machine learning. *TRANSFER*, 528812(4097):532909.

PayPal (2023). *A Global Study into the True Cost of Online Fraud*. https://www.paypal.com/us/brc/article/true-cost-of-online-fraud-report.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A., and Gulin, A. (2018). Catboost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, 31.

Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Shpyrko, V. and Koval, B. (2019). Fraud detection models and payment transactions analysis using machine learning. *SHS Web Of Conferences*, 65, 02002.

Torres, L. (2022). *Análisis del comportamiento de fraude transaccional en una entidad financiera a nivel nacional frente al marco de la pandemia COVID-19*. Ciencia Unisalle. Universidad de La Salle.

Trivedi, N. K., Simaiya, S., Lilhore, U. K., and Sharma, S. K. (2020). An efficient credit card fraud detection model based on machine learning methods. *International Journal of Advanced Science and Technology*, 29(5):3414–3424.

Apéndice A

Tablas adicionales

A1. XGBoost

Parámetro	Valor parámetro
objective	binary:logistic
eval_metric	logloss
eta	0.03
gamma	5
subsample	0.6
colsample_bytree	1
max_depth	5
nthread	5
nrounds	3000
early_stopping_rounds	50

Tabla A1.1: Valores de los parámetros para el modelo XGBoost

A2. Maquinas de vectores de soporte.

SVM kernel polinomial			Sensibilidad	Valor F
degree=2	C=1	gamma=0,001	0,9991	0,1815
degree=3	C=1	gamma=0,001	0,9983	0,1814

Tabla A2.1: Métricas de SVM con kernel polinomial en Python.

A3. CatBoost

Parámetro	Valor parámetro
`iterations`	800
`learning_rate`	0.1
`loss_function`	`Logloss`
`eval_metric`	`Logloss`
`depth`	10
`random_seed`	0

Tabla A3.1: Valores de los parámetros para el modelo CatBoost.

A4. Bosques Aleatorios

Parámetro	Valor parámetro
`ntree`	100
`mtry`	10
`replace`	`TRUE`
`importance`	`TRUE`

Tabla A4.1: Valores de los parámetros para el modelo RF.

Variables	Importancia
variable_3	893.8857
variable_5	729.6903
variable_2	660.9127
variable_1	639.9963
variable_4	624.7525
variable_39	539.9414
variable_6	513.0601
variable_13	481.3139
variable_8	425.1173
variable_9	394.7986
variable_30	394.0372
variable_16	380.2091
variable_10	379.0968
variable_70	373.3973
variable_7	312.8358

Tabla A4.2: Variables con mayor importancia según RF.