Arah Rojas Blanco (A00834299)
Valeria Aguilar Meza (A01741304)
Miguel Angel Barrientos Ballesteros (A01637150)
Mariela Quintanar de la Mora (A01642675)

Tecnológico de Monterrey

Inteligencia artificial avanzada para la ciencia de datos I (Gpo 101)
Clave: TC3006C

Septiembre 2025

Resumen Este trabajo aborda la detección de fraude bancario utilizando el conjunto de datos Bank Account Fraud (BAF), caracterizado por fuerte desbalance, sesgo y dinámica temporal. Se implementó un *pipeline* reproducible y modular que integra limpieza, particionado temporal (meses 1–5/6/7–8), y modelado con XGBoost, Random Forest, MLP y LightGBM, incorporando estrategias de rebalanceo (ninguna, *Random Oversampling*, SMOTE). El umbral de decisión se calibró para operar con **FPR** $\leq 5\,\%$, priorizando la recuperación de fraudes con control de falsos positivos. En pruebas sobre la **Variante III**, **LightGBM** mostró el mejor compromiso entre eficiencia y desempeño, alcanzando **AUC** ≈ 0.954, **Accuracy** ≈ 0.94 y **Recall@5 %FPR** entre 0.76–0.77, con **FPR** real ≈ 0.056; además, las curvas de aprendizaje evidenciaron bajo sesgo y varianza controlada. La robustez entre configuraciones BASE y ALT, junto con la orquestación por `run.py` y la generación automática de métricas y gráficas, valida la solución propuesta para entornos reales, aunque con importantes áreas de mejora.

1. Introducción

El fraude bancario es un problema común en las instituciones financieras, por lo cual es necesario utilizar técnicas de inteligencia artificial para detectar y prevenir transacciones fraudulentas. Para este proyecto se utilizó el Bank Account Fraud (BAF) Dataset [3,4], el cual se caracteriza por ser desbalanceado, sesgado y dinámico.

Durante la implementación se limpiaron los datos y se evaluaron distintos modelos de *machine learning*, incluyendo XGBoost [2], Random Forest, MLP y LightGBM [5], con el objetivo de identificar el más adecuado para la clasificación de fraudes. Se optó por utilizar LightGBM ya que destacó por su precisión y robustez, siendo la mejor opción para este proyecto.

Adicionalmente, se exploraron técnicas de rebalanceo como SMOTE [1], con el fin de mitigar el fuerte desbalance de clases presente en el dataset y mejorar la detección de la clase minoritaria.

2. Metodología

2.1. Pipeline

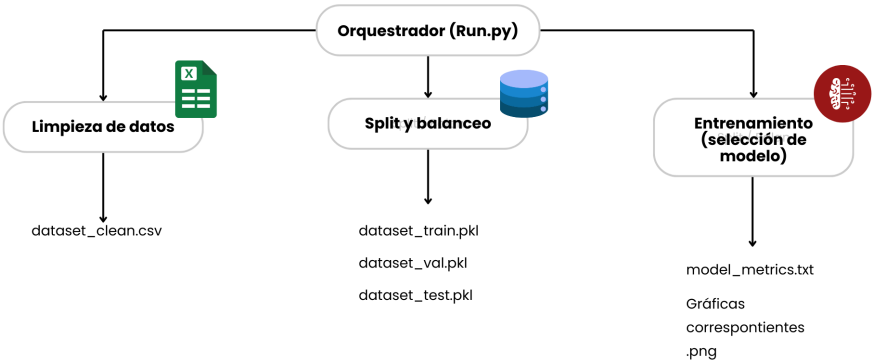

Figura 1. Pipeline Overview

El proyecto implementa un *pipeline* completo para el preprocesamiento y modelado del conjunto de datos de fraude bancario (*Bank Account Fraud*, BAF). Este flujo de trabajo automatiza el proceso desde la limpieza de los datos crudos hasta el entrenamiento de distintos modelos de aprendizaje automático, garantizando reproducibilidad y modularidad.

El `pipeline` consta de tres etapas principales:

1. **Limpieza de datos** Se cargan los archivos originales (`OriginalData/*.csv`) y se aplican transformaciones como: normalización de valores, detección de valores atípicos, transformación de variables, imputación de datos faltantes y codificación de variables categóricas. El resultado de esta etapa se guarda en la carpeta `cleandata/`, produciendo un archivo limpio por cada base.
2. **Preparación de particiones** A partir de los datos limpios, se generan divisiones temporales en tres conjuntos:
   - **Entrenamiento:** meses 1–5
   - **Validación:** mes 6
   - **Prueba:** meses 7–8

   Cada división se guarda como un archivo `.pkl` que incluye el nombre de la variante y el *dataframe*. Esta estrategia respeta la naturaleza temporal de los datos y evita fugas de información.

3. **Modelado** Una vez preparados los conjuntos, se entrena el modelo seleccionado. El `pipeline` soporta actualmente cuatro algoritmos:
   - **XGBoost** (`-mode xgboost`)
   - **MLP** (`-mode mlp`)
   - **LightGBM** (`-mode lgbm`)
   - **Random Forest** (`-mode randomforest`)

   En esta etapa se pueden aplicar técnicas de balanceo opcionales en el conjunto de entrenamiento: *ninguna* (`none`), *sobremuestreo aleatorio* (`ros`) o *SMOTE* (`smote`). El modelo entrenado se guarda en la carpeta `models/`, junto con métricas impresas en consola como AUC-ROC, AUC-PR, precisión, *recall*, F1 y matriz de confusión.

Orquestación del pipeline El archivo `run.py` actúa como orquestador de estas etapas mediante el argumento `-mode`, que selecciona entre `clean`, `prepare`, `xgboost`, `mlp`, `randomforest` o `lgbm`. Los principales parámetros disponibles son:

- `-datacsv`: ruta al archivo CSV de entrada (obligatorio en limpieza y preparación).
- `-preparedpath`: ruta a los conjuntos preparados (obligatorio en modelado).
- `-resampler`: estrategia de balanceo (`none`, `ros`, `smote`).
- `-seed`: semilla aleatoria para reproducibilidad (por defecto: 42).
- `-smote-k`: número de vecinos para SMOTE (por defecto: 5).

Resumen del flujo de trabajo El flujo puede resumirse de la siguiente manera:

1. Datos crudos → `cleandata/*.csv`
2. Datos limpios → `prepared_data/prepared_*`
3. Entrenamiento de modelos → `results/*.png,.txt`

Este diseño modular permite extender fácilmente el pipeline para incluir nuevos modelos o técnicas de balanceo sin necesidad de modificar las etapas anteriores.

3. Métodos

3.1. XGBoost

Con el objetivo de seleccionar el mejor modelo para el problema de clasificación binaria, se empleó **XGBoost** [2], un método de *gradient boosting* sobre árboles que construye un *ensamble aditivo* de árboles de decisión de manera secuencial. El modelo combina *shrinkage* (tasa de aprendizaje), *submuestreo* por filas y columnas, y límites de profundidad para regularizar la complejidad y mitigar el sobreajuste. Dado el fuerte desbalance de clases, se utilizó `scale_pos_weight` (razón #negativos/#positivos) y se eligió `AUC-PR` (`aucpr`) como métrica principal de validación.

Configuración principal.

- **Estructura y regularización:** `max_depth=6`, `learning_rate=0.03`, `n_estimators=5000` con `early_stopping_rounds=200`.
- **Submuestreo estocástico:** `subsample=0.9`, `colsample_bytree=0.9`.
- **Objetivo y métrica:** `objective=binary:logistic`, `eval_metric=aucpr`.
- **Desbalance:** `scale_pos_weight=`$\frac{N_{\text{neg}}}{N_{\text{pos}}}$ (calculado con `compute_scale_pos_weight`) y **SMOTE** aplicado sólo en *train*.
- **Reproducibilidad y desempeño:** `random_state=seed`, `n_jobs=-1`. El *early stopping* seleccionó la mejor iteración alrededor de ∼1.6k rondas, cuando la `aucpr` se estabilizó.

Resultados. (Variante III; *resampler* = `smote`)

- **Entrenamiento (TRAIN):** AUC-ROC = **0.9998**, AUC-PR = **0.9998**; Precision = 0.9996, Recall = 0.9936, F1 = 0.9966.
  Matriz de confusión [tn fp; fn tp]: [603,558 238; 3,865 599,931].
- **Validación (VAL):** AUC-ROC = **0.9551**, AUC-PR = **0.5354**; Precision = 0.7765, Recall = 0.4143, F1 = 0.5403.
  Matriz: [145,994 160; 786 556].
- **Prueba (TEST):** AUC-ROC = **0.9527**, AUC-PR = **0.5388**; Precision = 0.7801, Recall = 0.3906, F1 = 0.5206.
  Matriz: [238,728 292; 1,616 1,036].

En conjunto, XGBoost logra **AUC-ROC** y **AUC-PR** aceptables en validación y prueba, aunque se observa una importande diferencia respecto al desempeño casi perfecto en entrenamiento, lo cual indica *overfitting* inicial pese a la regularización y al *early stopping*. Esto tiene sentido por la naturaleza altamente desbalanceada y dinámica del BAF.

3.2. MLP

Para abordar el problema de clasificación binaria de fraude, se implementó una red neuronal multicapa (Multi-Layer Perceptron, MLP) utilizando `PyTorch`. El modelo se diseñó con tres capas densas: una capa de entrada con dimensión equivalente al número de atributos, una capa oculta de 256 neuronas, otra capa oculta de 128 neuronas y una capa de salida de dos neuronas (fraude / no fraude). Las funciones de activación empleadas fueron `ReLU`, y se incorporó `Dropout` con probabilidad de 0,2 para mitigar el sobreajuste.

El entrenamiento se realizó con el optimizador `Adam`, tasa de aprendizaje de 1×10^{-3} y `weight decay` de 1×10^{-4}. Para manejar el desbalanceo severo entre clases, se exploraron dos mecanismos: (i) reponderación de la función de pérdida `CrossEntropyLoss` mediante pesos inversos a la frecuencia de clase, y (ii) técnicas de sobremuestreo aplicadas únicamente al conjunto de entrenamiento (Random Oversampling o SMOTE). El criterio de parada temprana (*early stopping*) se estableció con base en la métrica F1 ponderada del conjunto de validación.

Durante el entrenamiento se monitorearon curvas de pérdida, *accuracy* y F1 ponderado para los conjuntos de entrenamiento y validación. Adicionalmente, se generaron curvas de *recall* y F1 específicas de la clase positiva (fraude), con el fin de evaluar el comportamiento del modelo sobre la clase minoritaria.

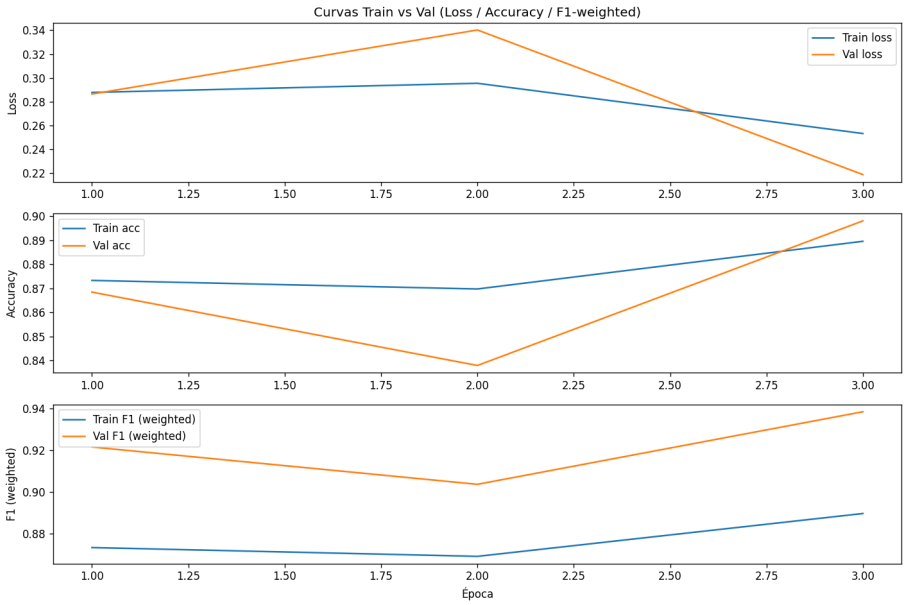

Figura 2. Curvas de entrenamiento y validación del MLP (Loss, Accuracy y F1 ponderado).

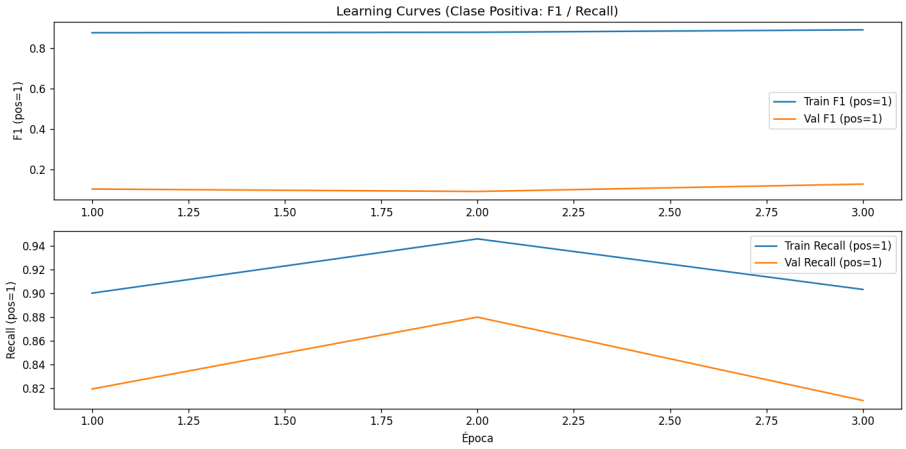

Figura 3. Curvas de aprendizaje del MLP para la clase positiva (F1 y Recall).

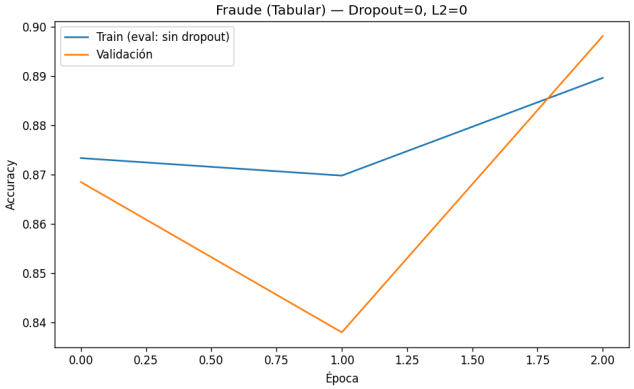

Figura 4. Evolución de la *accuracy* en entrenamiento y validación para el MLP.

En la evaluación final sobre el conjunto de prueba, el modelo obtuvo los siguientes resultados:

- **Loss:** 0.2508
- **Accuracy:** 0.8872
- **F1 ponderado:** 0.9306
- **Recall ponderado:** 0.8872
- **F1 clase positiva (fraude):** 0.0882
- **Recall clase positiva (fraude):** 0.4970

Estos valores muestran que el MLP logra un buen desempeño general en términos de exactitud y métricas globales ponderadas, sin embargo, la clase positiva presenta dificultades: aunque se alcanzó un *recall* cercano al 50 %, el F1 de fraude fue bajo, lo cual indica que aún existen problemas de precisión en la detección de transacciones fraudulentas. Este resultado es consistente con la naturaleza desbalanceada del conjunto de datos, y resalta la necesidad de seguir afinando técnicas de muestreo o ajustes en la arquitectura para mejorar el desempeño en la clase minoritaria.

3.3. Random Forest

Se entrenó un modelo Random Forest Classifier con 100 árboles y `class_weight=balanced` para manejar el desbalance de clases. La partición de los datos fue 70 % para entrenamiento y 30 % para prueba.

Los resultados de desempeño se resumen en la Tabla 1.

Cuadro 1. Resultados del modelo Random Forest en train y test

Dataset	**AUC-ROC**	**AUC-PR**	**Precision**	**Recall**
Train	1.0000	1.0000	1.0000	0.9804
Test	0.7352	0.0440	0.0000	0.0000

La matriz de confusión en entrenamiento y prueba se muestra en la Tabla 2.

Cuadro 2. Matriz de confusión del Random Forest

Train	Predicción	
	No Fraude	Fraude
No Fraude	4317	0
Fraude	1	50

Test	Predicción	
	No Fraude	Fraude
No Fraude	1850	0
Fraude	22	0

En la Figura 5 y Figura 6 se muestran las curvas ROC y Precision-Recall respectivamente. Aunque la curva ROC presenta un área de 0.7352 en test, la curva Precision-Recall revela que el modelo no logra detectar fraudes (recall = 0). Esto indica un problema de *overfitting* y de fuerte desbalance de clases.

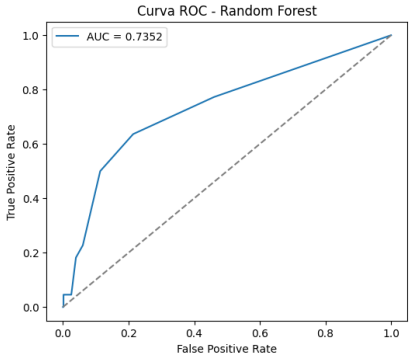

Figura 5. Curva ROC del modelo Random Forest

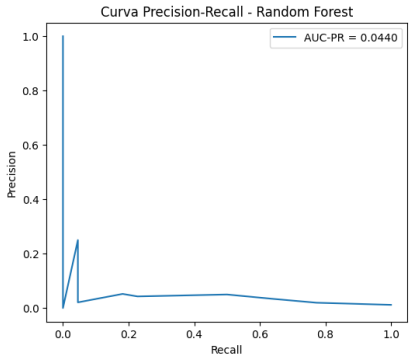

Figura 6. Curva Precision-Recall del modelo Random Forest

3.4. LightGBM

El modelo final adoptado fue **LightGBM** (Light Gradient Boosting Machine), un algoritmo de *gradient boosting* basado en árboles de decisión. LightGBM emplea técnicas avanzadas de optimización que lo hacen muy eficiente: utiliza un algoritmo basado en histogramas para binarizar características, acelerando el entrenamiento y reduciendo el uso de memoria, y adopta una estrategia de crecimiento hoja a hoja (*leaf-wise*), que selecciona dinámicamente la división de mayor ganancia en cada paso. Estas características permiten que LightGBM procese conjuntos de datos muy grandes o de alta dimensionalidad con gran rapidez y escalabilidad.

Se eligió LightGBM para la detección de fraude bancario por varias razones:

- **Eficiencia y escalabilidad.** Su implementación optimizada permite entrenar modelos complejos en menos tiempo que métodos clásicos de *boosting*, algo crítico cuando se manejan millones de transacciones.
- **Manejo de datos desbalanceados.** LightGBM incluye parámetros específicos (por ejemplo, `is_unbalance`, o `scale_pos_weight`) para ponderar las clases minoritarias.
- **Precisión y flexibilidad.** En general, LightGBM suele ofrecer alto rendimiento predictivo en problemas de clasificación compleja, y su amplia variedad de hiperparámetros (profundidad, número de hojas, regularización, etc.) permite ajustar finamente el modelo a las particularidades del problema.

Estas ventajas hicieron de LightGBM la opción más adecuada frente a otros algoritmos de *machine learning* para nuestro proyecto de detección de fraude bancario. Los resultados y el análisis del desempeño de LightGBM se encuetran en la sección de Solución.

4. Solución

4.1. Datos

El conjunto de datos empleado que demostró el mejor desempeño en el modelo fue (`Variant III.csv`) el cual pasó por un robusto proceso de limpieza y preparación de datos. Este proceso incluyó las siguientes etapas:

1. **Revisión inicial de los datos.** Se cargaron los datasets originales y se revisó el número de filas, columnas y tipos de variables. Se identificaron las variables numéricas y categóricas, así como la columna objetivo (`fraud_bool`). Se verificaron duplicados y se evaluó la cantidad de valores faltantes en cada columna.
2. **Tratamiento de valores fuera de rango.** Se definieron rangos para cada variable, a partir de la documentación del conjunto de datos BAF. Todos los valores que cayeron fuera de estos rangos fueron reemplazados por `NaN`, para ser tratados posteriormente como datos faltantes.
3. **Detección y tratamiento de outliers.** Se analizaron las variables numéricas del subconjunto no fraude para detectar valores atípicos mediante el criterio del rango intercuartílico (IQR). Los *outliers* fueron identificados gráficamente con histogramas y diagramas de caja, y se documentó en qué variables aparecían.

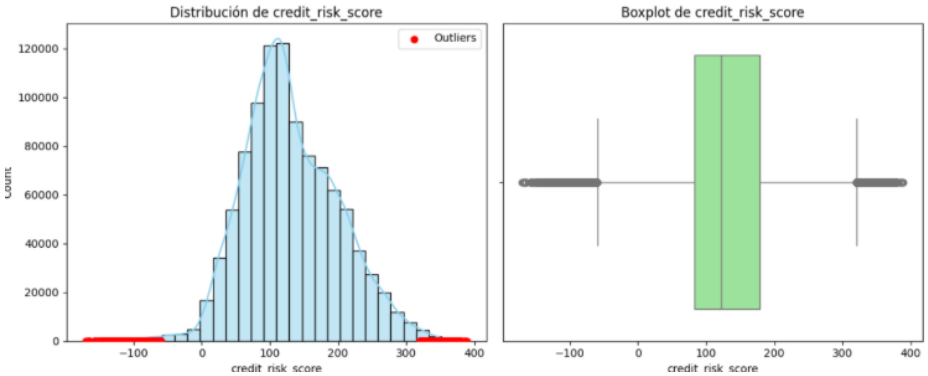

Figura 7. Ejemplo de gráficos de detección de outliers para la variable credit risk score

4. **Transformaciones de variables.** Para mejorar la distribución de las variables sesgadas, se aplicaron distintas transformaciones como Yeo-Johnson, Log1p (logarítmica), Log10 y raíz cuadrada (Sqrt). Estas transformaciones redujeron la asimetría y la influencia de valores extremos.
5. **Imputación de valores faltantes.** Para las variables donde el valor cero tenía un significado especial (*"no aplica"*), los valores faltantes se reemplazaron con 0. En otras variables numéricas, los `NaN` fueron reemplazados con la mediana de cada columna, evitando perder información y manteniendo la robustez frente a valores extremos.
6. **Codificación de variables categóricas.** Se aplicó *One-Hot Encoding* a las variables categóricas; cada categoría fue transformada en una nueva columna binaria (0/1).

4.2. Modelo

Para LightGBM se entrenaron dos configuraciones principales, aprovechando su eficiencia en el manejo de grandes volúmenes de datos y su capacidad para tratar clases desbalanceadas:

- **Modelo BASE:** configuración estándar con `num_leaves=31`, regularización $L_2 = 1{,}0$ y manejo de desbalance automático cuando no se aplicaba sobremuestreo.
- **Modelo ALT:** configuración alternativa más compleja, con `num_leaves=63`, `min_data_in_leaf=80` y regularización $L_2 = 2{,}0$, buscando un mayor ajuste del modelo.

Ambos modelos se entrenaron con `SMOTE` aplicado al conjunto de entrenamiento para equilibrar las clases, dejando a LightGBM únicamente el ajuste fino de la frontera de decisión.

Durante el entrenamiento, se utilizó `early stopping` con un máximo de 1000 iteraciones y paciencia de 100 rondas, monitoreando el conjunto de validación. El

umbral de clasificación se seleccionó en validación restringiendo el **FPR** $\leq$ **5 %**, lo cual permite priorizar la detección de fraudes minimizando falsos positivos.

4.3. Refinamiento

La fase de refinamiento se centró en ajustar los parámetros de LightGBM y optimizar su desempeño. Entre las técnicas y ajustes aplicados destacan:

- **Regularización L2 (`lambda_l2`):** Se incorporó penalización L2 en la función objetivo para evitar sobreajuste. Este término de regularización tiende a reducir los pesos del modelo y mejora su generalización. Ajustar `lambda_l2` permite controlar la complejidad: valores mayores inducen más penalización sobre pesos grandes, haciendo el modelo más conservador.
- **Hiperparámetros clave:** Se mejoraron parámetros estructurales de los árboles. El parámetro `num_leaves` (número máximo de hojas por árbol) regula la complejidad: valores más altos permiten árboles más profundos y complejos, pero aumentan el riesgo de ajuste a ruido. Por su parte, `min_data_in_leaf` establece el mínimo de instancias requeridas en una hoja; incrementarlo hace que el árbol sea más conservador al requerir más datos para una división. En combinación, estos parámetros regularizan la capacidad de aprendizaje del modelo.
- **Validación temprana (Early Stopping):** Para prevenir el sobreentrenamiento y reducir el tiempo de cómputo, se empleó *early stopping*. Esto implica monitorear la métrica de evaluación en un conjunto de validación y detener el entrenamiento cuando deja de mejorar tras un número fijo de iteraciones. De este modo, el entrenamiento se interrumpe antes de que el modelo se ajuste al ruido del set de entrenamiento, mejorando la generalización.
- **Comparación de modelos BASE y ALT:** Se definieron dos versiones del modelo para medir el impacto de los ajustes. El modelo **BASE** fue entrenado con los parámetros por defecto de LightGBM, mientras que el modelo **ALT** incorporó los hiperparámetros ajustados mencionados (mayor `num_leaves`, ajuste de `min_data_in_leaf`, regularización, etc.). La comparación de sus resultados en validación permitió confirmar que los ajustes mejoran métricas clave sin sacrificar estabilidad.
- **Selección de umbral de clasificación:** Finalmente, al evaluar el modelo binario se ajustó el umbral de probabilidad para clasificar una transacción como fraude. En este tipo de problema es crítico limitar los falsos positivos, por lo que se eligió un umbral que garantizara una tasa de falsos positivos (FPR) de a lo sumo 5 %. En términos de la curva ROC, esto equivale a operar en un punto donde el FPR $\leq$ 0,05, equilibrando así sensibilidad y precisión. Las métricas finales (F1-score, precisión y *recall*) se midieron con este umbral, siguiendo recomendaciones de la documentación.

5. Resultados

5.1. Explicación de outputs

El proyecto fue implementado para que, tras entrenar y evaluar *LightGBM*, se generen los siguientes outputs:

- `lgbm_results.txt`: archivo de texto con las métricas por versión de modelo (**BASE** y **ALT**): AUC, Accuracy, Recall@5 %FPR, FPR real, mejor iteración.
- `lgbm_errorlearning_curve_[Versión].png`: curva de aprendizaje que compara **error** en **entrenamiento** y **validación** a lo largo de las **iteraciones**.
- `lgbm_learning_curve_[Versión].png`: curva de aprendizaje que compara **AUC** en **entrenamiento** y **validación** a lo largo de las **iteraciones**.

5.2. Evaluación

Análisis de métricas obtenidas Tras evaluar con el conjunto de prueba de la **Variante III**, se obtuvieron:

- **Modelo BASE:** AUC = 0.9540, Accuracy = 0.9424, F1 = 0.2271, Recall@5 %FPR = 0.7711, FPR real = 0.0557, mejor iteración = 592.
- **Modelo ALT:** AUC = 0.9537, Accuracy = 0.9440, F1 = 0.2299, Recall@5 %FPR = 0.7621, mejor iteración = 519.

El **AUC** alto indica elevada capacidad para diferenciar fraude vs. no fraude. Dado el contexto del problema, se prioriza **Recall@5 %FPR** para maximizar la detección de fraudes fijando un *límite operativo* de falsos positivos del 5 %. Un valor cercano a 0.76 implica que el modelo identifica alrededor del 76 % de fraudes bajo esa restricción, mientras que el **FPR real** de 0.055 sugiere que el punto operativo en prueba quedó muy próximo al 5 %.

El **F1** (en torno a 0.23) es bajo, lo que refleja **baja precisión** (muchos falsos positivos) como resultado del umbral seleccionado para sostener un **recall** alto; este es un *trade-off* esperado al optimizar Recall@5 %FPR. Aun así, la baja precisión deja margen de mejora (por ejemplo, calibración de probabilidades o estrategias alternativas de rebalanceo). En conjunto, los resultados son coherentes con los objetivos del caso y con la regularización aplicada.

Finalmente, el desempeño es **similar entre BASE y ALT**, lo que sugiere **robustez** del enfoque ante variaciones de configuración y/o partición.

Análisis de sesgo La **curva de aprendizaje de error** (Fig. 8) muestra el error a lo largo de las iteraciones en entrenamiento y validación. En **entrenamiento**, el error desciende rápidamente y se aproxima a cero antes de las ∼50 iteraciones, manteniéndose prácticamente nulo después; esto sugiere **bajo sesgo**. En **validación**, el error se estabiliza tras ∼250 iteraciones alrededor de ∼0.045. Aunque mayor que en entrenamiento (como es esperable), el nivel observado también indica **bajo sesgo** en el punto de convergencia.

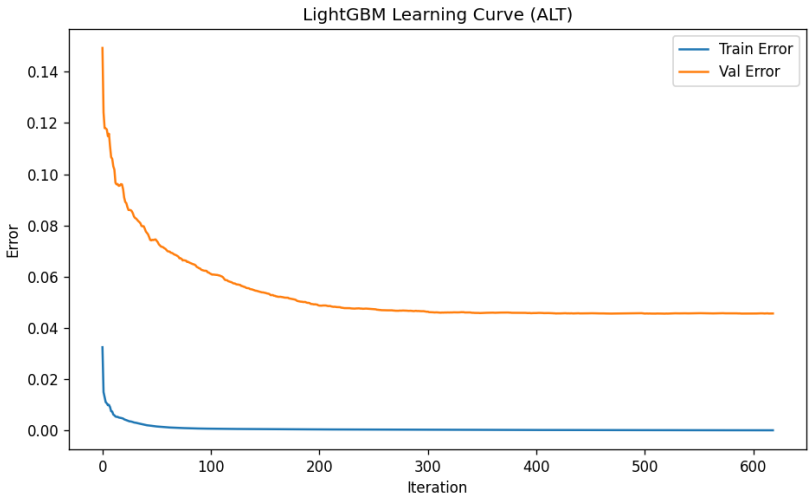

Figura 8. Curva de aprendizaje de **error** (entrenamiento vs. validación) para LightGBM.

Análisis de varianza En la misma curva de aprendizaje de error, el **gap** entre entrenamiento y validación es **moderado y estable** (aprox. 0.04 en el eje del error), lo que indica **varianza presente pero controlada**. No se aprecian incrementos tardíos del error de validación que evidencien sobreajuste marcado; por el contrario, la separación se estabiliza, consistente con una varianza **aceptable**.

Nivel de ajuste Con **bajo sesgo** y **varianza controlada**, el modelo evidencia **buen ajuste** (ni *underfitting* ni *overfitting*) y generaliza adecuadamente en prueba. Esta conclusión es consistente con la **curva de aprendizaje de AUC** (Fig. 9), donde las métricas se estabilizan a medida que avanzan las iteraciones, apoyando la elección de la mejor iteración mediante *early stopping*.

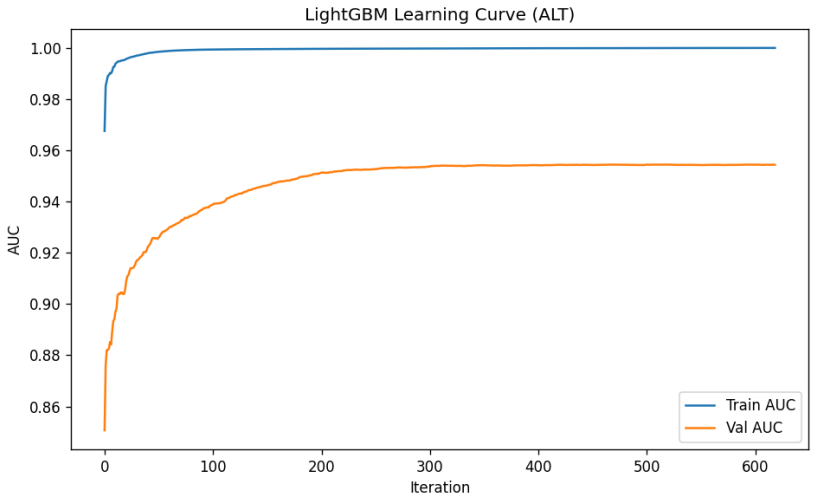

Figura 9. Curva de aprendizaje de **AUC** (entrenamiento vs. validación) para LightGBM.

6. Conclusiones

El presente proyecto permitió diseñar y evaluar un *pipeline* para el entrenamiento y evaluación de modelos de ML enfocadaos a la detección de fraude bancario, desde la limpieza y preparación de los datos hasta la implementación y comparación de estos distintos modelos. Los resultados obtenidos evidencian que **LightGBM** destacó como la alternativa más robusta y consistente frente al desbalance y la dinámica temporal de los datos. En particular, la capacidad de mantener un **AUC** elevado ($\approx$0.95) y un **Recall@5 %FPR** cercano al 76 % confirma que obtuvo el mejor desempeño en un problema en el que se buscó maximizar la detección de fraudes sin incrementar de forma desproporcionada los falsos positivos. Asimismo, el diseño modular del `pipeline` garantiza reproducibilidad, escalabilidad y la posibilidad de incorporar nuevos modelos o técnicas de balanceo de forma sencilla. En conjunto, este trabajo demuestra que la combinación del preprocesamiento adecuado de datos, con algoritmos avanzados como LightGBM proporcionó un solución adecuada para enfrentar el reto del fraude bancario, aportando una base sólida para futuras implementaciones del equipo relacionadas a Machine Learning, incluso en contextos reales.

Conclusiones Personales

Miguel Ángel Barrientos. El proyecto fue un reto importante en mi desarrollo profesional, pues me obligó a integrar nuevos conocimientos de los cuales tenía muy pocas bases. Tras haber concluido, considero que las habilidades que adquirí

en relación con *Machine Learning* y estadística me permitirán desarrollar soluciones robustas para problemas futuros de aprendizaje automático, diseñándolas conscientemente y tomando decisiones adecuadas al problema específico, tal como sucedió en la detección de fraude bancario. Además, el proyecto me permitió aplicar conocimientos previos sobre arquitectura de software, implementando un *pipeline* pensado en la eficacia, el uso intuitivo, la escalabilidad y la modularidad de cada fase.

Arah Rojas Blanco. Este reto fue una experiencia integradora que me permitió aplicar lo aprendido en estadística y *machine learning*, desde la limpieza de datos hasta la comparación de modelos. Pude identificar sus ventajas y desventajas, y comprobar la importancia de preparar bien los datos y ajustar parámetros para lograr mejores resultados. En lo personal, esta práctica fortaleció mi capacidad de análisis y me mostró cómo usar la teoría en un problema real como la detección de fraude.

Valeria Aguilar Meza. Como estudiante de Ciencia de Datos, llegué a este proyecto con una base previa en varios conceptos del curso. Lo valioso fue llevar esos conocimientos de la teoría a la práctica. El proyecto me permitió aplicar técnicas que antes solo entendía de forma abstracta, y al mismo tiempo me retó a profundizar en nuevas metodologías y herramientas que ampliaron mi visión de la disciplina.

Mariela Quintanar de la Mora. En este proyecto de *Machine Learning* aprendí a implementar y comparar modelos como Random Forest, limpiar y preparar datos, y usar librerías útiles. También comprendí conceptos importantes como sesgo y varianza, lo que me ayudó a evaluar y mejorar los modelos. Todo esto fortaleció mis habilidades teóricas y prácticas en inteligencia artificial.

Descripción de contribuciones

- **Miguel Ángel Barrientos:** Implementación de XGBoost; detección de *outliers*; diseño e implementación general del *pipeline* (orquestador, integración de scripts de limpieza y modelos, README); división (*split*) de conjuntos de datos; balanceo con diferentes técnicas de *oversampling*; configuración final con Variante III y LightGBM; reporte de análisis.
- **Arah Rojas Blanco:** Implementación inicial de LightGBM para configuraciones Base y Alt; tratamiento de valores fuera de rango; transformaciones de variables; imputación de valores faltantes; reporte de análisis.
- **Valeria Aguilar Meza:** Implementación del modelo MLP; codificación de variables categóricas; reporte de análisis.
- **Mariela Quintanar de la Mora:** Implementación del modelo Random Forest; reporte de análisis.

Referencias

1. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: Synthetic minority over-sampling technique. Journal of Artificial Intelligence Research **16**, 321–357 (2002)
2. Chen, T., Guestrin, C.: Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 785–794 (2016)
3. Jesus, S.: Bank account fraud dataset (neurips 2022). `https://www.kaggle.com/datasets/sgpjesus/bank-account-fraud-dataset-neurips-2022` (2022), accessed: 2025-09-21
4. Jesús, S., Pombal, J., Alves, D., Cruz, A.F., Saleiro, P., Ribeiro, R.P., Gama, J., Bizarro, P.: Turning the tables: Biased, imbalanced, dynamic tabular datasets for ml evaluation. In: Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS 2022), Track on Datasets and Benchmarks (2022)
5. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y.: Lightgbm: A highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems 30 (2017)