INFB6074 - INFRAESTRUCTURA PARA CS. DE DATOS
INGENIERÍA CIVIL EN CIENCIA DE DATOS

Actividad N° 3

" Comparación de Algoritmos para Detección de Fraudes Financieros. "

Dr. Ing. Michael Miranda Sandoval

Mayo de 2025

UTEM UNIVERSIDAD TECNOLÓGICA METROPOLITANA del Estado de Chile

ACTIVIDAD N° 3

INTEGRANTES:

Nombre: Felipe Martínez González **Email:** fmartinezgo@utem.cl
Nombre: Christian Pérez Flores **Email:** cperezfl@utem.cl
Nombre: Benjamín Zamorano Soto **Email:** bzamoranos@utem.cl

1. Introducción

En este informe se presenta el desarrollo y evaluación de un sistema de detección de fraude en transacciones con tarjetas de crédito utilizando técnicas de aprendizaje automático. Esta actividad permite comprender y experimentar la aplicación práctica de diversos algoritmos de machine learning para la identificación de patrones fraudulentos en datos financieros. El objetivo principal es entrenar y evaluar modelos capaces de clasificar transacciones como fraudulentas o legítimas, determinando su rendimiento mediante métricas como precisión, exhaustividad (recall) y F1-score. Para ello, se utilizan algoritmos como Regresión Logística, Random Forest, Redes Neuronales, Árboles de Decisión y Análisis Discriminante Lineal, aplicándolos a un conjunto de datos de transacciones financieras. En este documento se detalla cada etapa del desarrollo, desde el preprocesamiento de los datos y la ingeniería de características hasta el entrenamiento, evaluación y comparación de los modelos. Finalmente, se analiza el rendimiento de cada algoritmo y se discuten las implicaciones de los resultados obtenidos.

2. Detalle de las Actividades

2.1 Explicación Previa a Cada Parte de la Secuencia Lógica

1. **Importación de Bibliotecas**: La primera etapa consiste en la carga de bibliotecas necesarias para el desarrollo del análisis. Se incluyen paquetes para manipulación de datos (pandas, numpy), visualización (matplotlib, seaborn) y modelado predictivo (scikit-learn). Esta preparación inicial es fundamental para habilitar las funciones y herramientas que serán utilizadas en las siguientes etapas.

2. **Carga del Conjunto de Datos**: Posteriormente, se procede a la carga del conjunto de datos desde un archivo CSV que contiene registros de transacciones con tarjetas de crédito, obtenido desde Kaggle. El dataset a explorar El dataset contiene 284,807 transacciones con tarjeta de crédito realizadas por europeos en dos días de septiembre de 2013, de las cuales solo 492 son fraudes, lo que refleja un fuerte desbalance de clases. Las variables han sido transformadas mediante PCA (V1 a V28), excepto 'Time' (segundos desde la primera transacción) y 'Amount' (monto de la transacción). La variable objetivo 'Class' indica fraude (1) o transacción legítima (0). Esta acción permite disponer de los datos en memoria para su exploración y análisis posterior. El dataset constituye la base del estudio.

3. **Exploración Inicial de Datos**: En esta etapa se examinan las primeras y últimas observaciones del conjunto de datos, se identifican posibles valores nulos (Fig. 1) y se analizan estadísticas descriptivas. Esta revisión preliminar permite obtener una visión general de la estructura del dataset, así como detectar inconsistencias que deben ser corregidas.

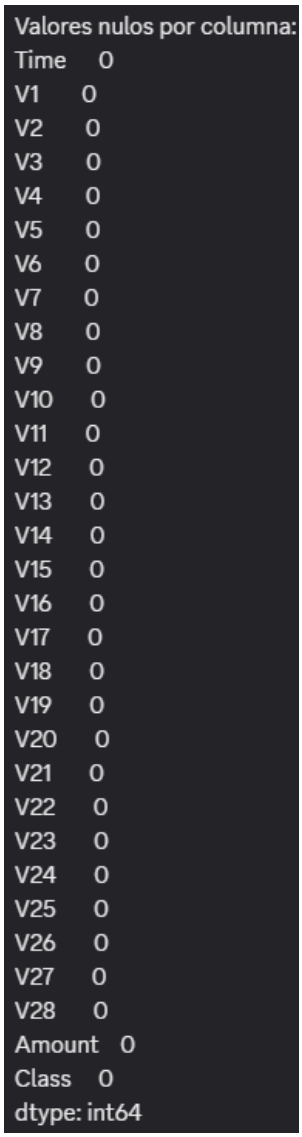

Figure 1: Exploración de posibles valores nulos en las distintas columnas, en el dataset específico se ve que no existen, por lo que no es necesario corregir.

4. **Limpieza de Datos**: Una vez explorados los datos, se realiza un proceso de limpieza que incluye la eliminación de registros duplicados (Fig. 2). Esta acción busca mejorar la calidad de los datos y evitar la introducción de sesgos o errores durante el entrenamiento de los modelos.

```
Tamaño del DataFrame antes de eliminar duplicados: 284807
Tamaño del DataFrame después de eliminar duplicados: 283726
Cantidad de filas duplicadas restantes: 0
```

Figure 2: Muestra de filas duplicadas, las que son limpiadas del dataset antes de ser trabajadas.

5. **Preparación del Conjunto de Datos**: Esta fase incluye la selección de variables de interés, la separación entre variables predictoras (atributos) y la variable objetivo (clase), así como la normalización o transformación de los datos si es necesario (Fig. 3). El propósito es dejar el

conjunto de datos en condiciones óptimas para el entrenamiento de los modelos de aprendizaje automático.

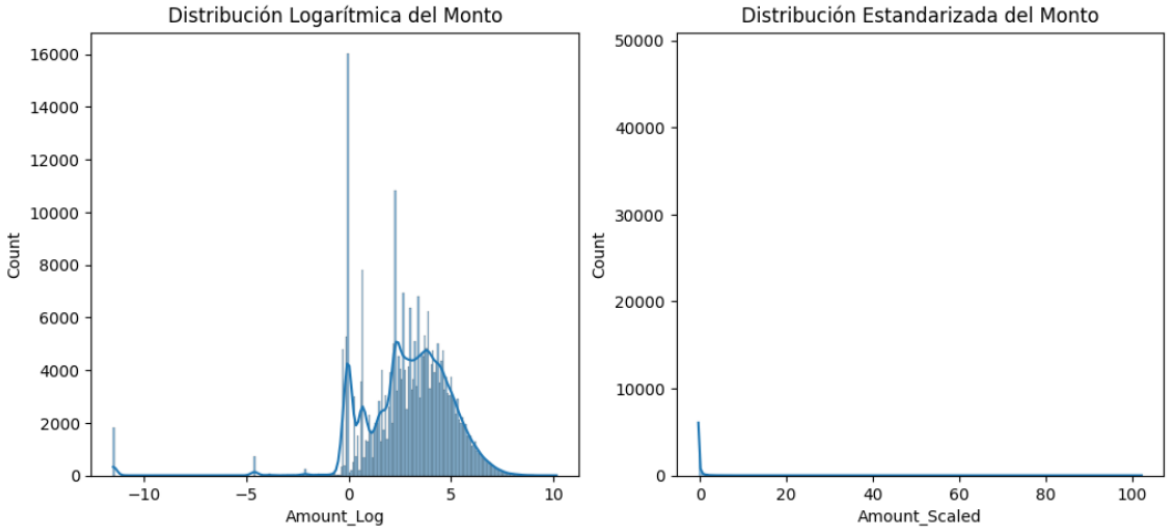

Figure 3: Gráfico que muestra las distribuciones logarítmica y estandarizada de la variable monto luego de su transformación. Por otro lado se transformó la variable de tiempo.

6. **División del Conjunto de Datos**: Se divide el conjunto de datos en dos subconjuntos: uno de entrenamiento y otro de prueba (Fig. 4). Esta partición permite evaluar el desempeño de los modelos en datos no utilizados durante el entrenamiento, con el fin de obtener una medida realista de su capacidad de generalización.

```
# Dividir los datos en características (X) y variable objetivo (y)
X = df.drop('Class', axis=1)  # Excluimos 'Class'
y = df['Class']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)


# Escalar las características (después de la división)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figure 4: División del conjunto de datos, con clase como la variable objetivo.

7. **Entrenamiento de Modelos**: En esta etapa se entrenan diversos modelos de clasificación supervisada (Fig. 5), tales como regresión logística, árboles de decisión, Random Forest, perceptrón multicapa (MLP) y análisis discriminante lineal (LDA). Cada modelo se entrena utilizando los datos previamente preparados y se ajusta a los patrones identificados en el conjunto de entrenamiento.

```python
modelos = {
    "Regresión Logística": LogisticRegression(random_state=42, solver='liblinear'),
    "Random Forest": RandomForestClassifier(random_state=42),
    "Red Neuronal": MLPClassifier(random_state=42, max_iter=300, early_stopping=True), # Usa early stopping
    "Árbol de Decisión": DecisionTreeClassifier(random_state=42),
    "Análisis Discriminante Lineal": LinearDiscriminantAnalysis(),
}
```

Figure 5: Lista de modelos utilizados para la detección de fraudes y su respectiva llamada.

8. **Evaluación de Modelos**: Una vez entrenados, los modelos son evaluados utilizando métricas de desempeño, entre las que se destaca el área bajo la curva ROC (AUC-ROC) (Fig. 6). Esta métrica permite comparar el rendimiento de los distintos clasificadores en la tarea de detección de transacciones fraudulentas. Se generan gráficos que representan visualmente el comportamiento de cada modelo (Fig. 7).

```
Entrenando: Regresión Logística
Entrenando: Random Forest
Entrenando: Red Neuronal
Entrenando: Árbol de Decisión
Entrenando: Análisis Discriminante Lineal

Tabla de Resultados (DataFrame):
                      Modelo  Precisión (Fraude)  Recall (Fraude)  F1-Score (Fraude)      AUC  Support (Fraude)
          Regresión Logística            0.858586         0.598592           0.705394 0.969664             142.0
                Random Forest            0.956140         0.767606           0.851562 0.934554             142.0
                 Red Neuronal            0.867257         0.690141           0.768627 0.949575             142.0
            Árbol de Decisión            0.755725         0.697183           0.725275 0.848403             142.0
Análisis Discriminante Lineal            0.868852         0.746479           0.803030 0.975721             142.0
```

Figure 6: Resultados entregados por los distintos modelos en forma de tabla, que muestra distintas métricas de desempeño.

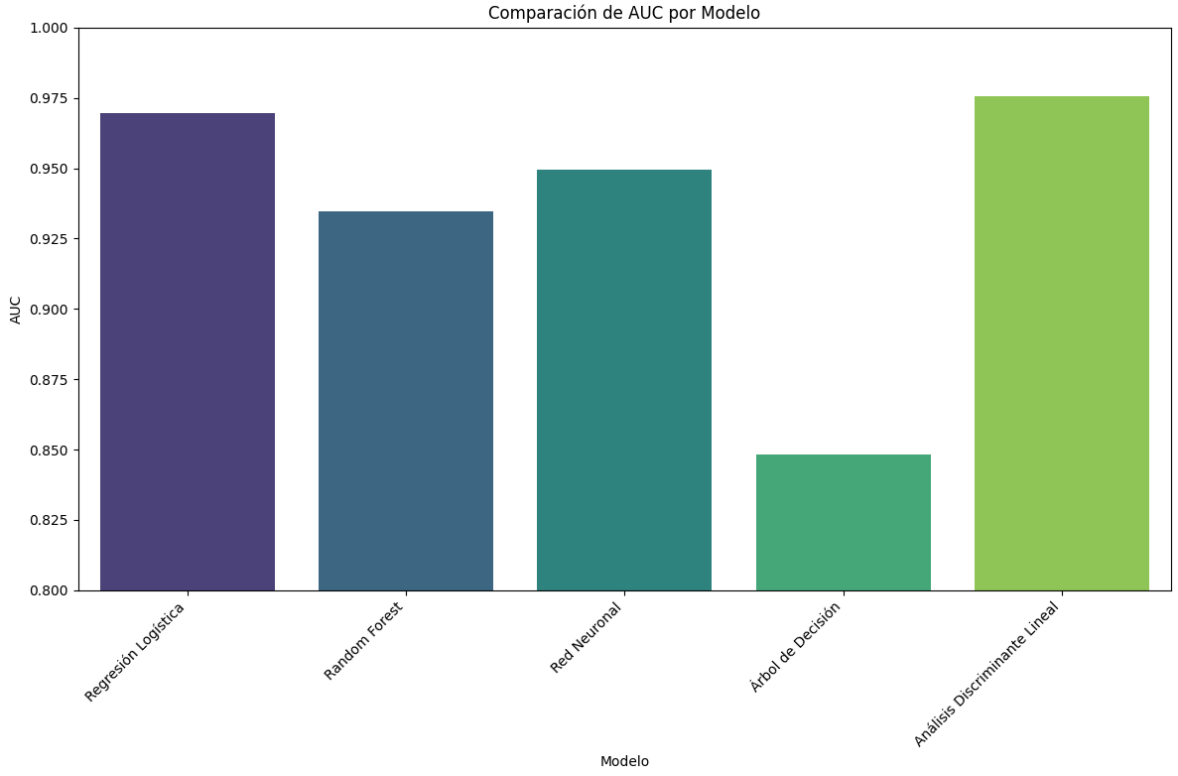

Figure 7: Gráfico comparativo que representa el comportamiento de los modelos. Se puede observar que el modelo de Análisis de Discriminante Lineal es el que mejor comportamiento presenta, con un AUC de 0.9757.

3. Conclusión

El análisis comparativo de las arquitecturas paralela, distribuida y Big Data permite comprender las distintas formas en que una solución de detección de fraudes, como la desarrollada en este proyecto, puede ser implementada y escalada según el contexto. En el entorno académico y experimental en que se ejecutó el proyecto, la arquitectura paralela implícita en Scikit-learn resultó ser suficiente y eficiente, permitiendo realizar pruebas y entrenamientos en tiempos razonables con los recursos disponibles localmente. No obstante, si se proyecta esta solución hacia un entorno productivo o comercial, como el de una institución bancaria, sería necesario considerar el uso de arquitecturas distribuidas para asegurar la disponibilidad, escalabilidad y capacidad de análisis en tiempo real. Finalmente, en contextos de grandes volúmenes de información y alta velocidad de generación de datos, la adopción de una arquitectura Big Data se vuelve indispensable para garantizar un procesamiento eficiente y continuo. Este análisis evidencia que, aunque el enfoque actual es apropiado para el alcance del proyecto, la elección de la arquitectura adecuada será clave si se busca una implementación robusta en escenarios reales.

4. Referencias

1. ULB Machine Learning Group. (2018). Credit Card Fraud Detection Dataset. Kaggle. https://www.kaggle.com/data ulb/creditcardfraud/data

2. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

3. The Pandas Development Team. (2024). pandas-dev/pandas: Pandas (version 2.2.2) [Software]. Zenodo. https://doi.org/10.5281/zenodo.8081452. Documentación: https://pandas.pydata.org/

4. Caswell, T. A., Droettboom, M., Hunter, J., et al. (2023). Matplotlib: Visualization with Python (version 3.8.2). https://matplotlib.org/stable/

5. Scikit-learn Developers. (2024). Scikit-learn documentation (version 1.4.2). https://scikit-learn.org/stable/documentation.html

6. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B. E., Bussonnier, M., Frederic, J., ... Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In Positioning and Power in Academic Publishing: Players, Agents and Agendas (pp. 87–90). IOS Press. https://jupyter.org/

7. Intel Corporation. (2023). Introduction to Parallel Programming and Multi-Core Processing. Recuperado de https://www.intel.com/

8. Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., Stoica, I. (2016). Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM, 59(11), 56–65. https://doi.org/10.1145/293466

9. Dean, J., Ghemawat, S. (2008). MapReduce: simplified data processing on large clusters. Communications of the ACM, 51(1), 107–113. https://doi.org/10.1145/1327452.1327492

10. OpenAI. (2025). ChatGPT (versión GPT-4.5) [Modelo de lenguaje de IA]. Utilizado como herramienta de asistencia para generar fragmentos de código y apoyar en la escritura de scripts en Python, además de asistencia para redactar y estructurar el presente informe. https://openai.com/chatgpt