Comparación de Modelos de Clasificación Supervisada: Regresión Logística vs. Árbol de Decisión

Jorge Eliecer Delgado Cortes

Johan Mauricio Espinoza Espejo

Johan Alexander Arevalo Contreras

Universidad de Cundinamarca

Introducción a Machine Learning

Edwar Daniel Perez Lopez

29 de octubre de 2025

Ubaté Cundinamarca

Contenido

1. Introducción 3
1.1. Justificación del Dataset 3
1.2. Metodología 3
2. Implementación y Resultados 3
2.1. Preparación de Datos 4
2.2. Modelo 1: Regresión Logística 5
2.3. Modelo 2: Árbol de Decisión 7
3. Análisis y Discusión 8
3.1. Análisis de la Similitud en Resultados 8
4. Conclusiones 9
Referencias 10

1. Introducción

La clasificación supervisada es una tarea fundamental en el aprendizaje automático (*Machine Learning*) que consiste en entrenar un modelo para predecir una etiqueta categórica (una clase) basándose en un conjunto de variables de entrada (características). El objetivo de este informe es implementar y comparar el rendimiento de dos algoritmos de clasificación distintos, la Regresión Logística y el Árbol de Decisión, sobre un mismo conjunto de datos.

1.1. Justificación del Dataset

Para este ejercicio, se seleccionó el dataset loan_approval.csv, obtenido del repositorio Kaggle. La elección de este conjunto de datos se justifica por varias razones clave:

1. Idoneidad para la Tarea: El dataset presenta un problema de clasificación binaria claro (la variable objetivo loan_approved es Falso/Verdadero), lo cual es ideal para los algoritmos seleccionados.
2. Calidad de los Datos: Un análisis exploratorio inicial (ver *Figura 1*) reveló que el dataset estaba completo y limpio, sin valores nulos (missing values) en ninguna de las columnas de interés. Esto eliminó la necesidad de técnicas complejas de preprocesamiento o imputación de datos.
3. Sugerencia del Curso: Fue el conjunto de datos proporcionado como ejemplo en el material de clase, facilitando el seguimiento de la actividad práctica.

1.2. Metodología

La metodología consistió en cargar el dataset, seleccionar las variables predictoras (income, credit_score, etc.) y la variable objetivo (loan_approved). Posteriormente, los datos se dividieron en un 75% para entrenamiento y un 25% para pruebas (test_size=0.25). Finalmente, se entrenaron ambos modelos con el conjunto de entrenamiento y se evaluaron sus métricas de rendimiento (precisión y matriz de confusión) usando el conjunto de pruebas.

2. Implementación y Resultados

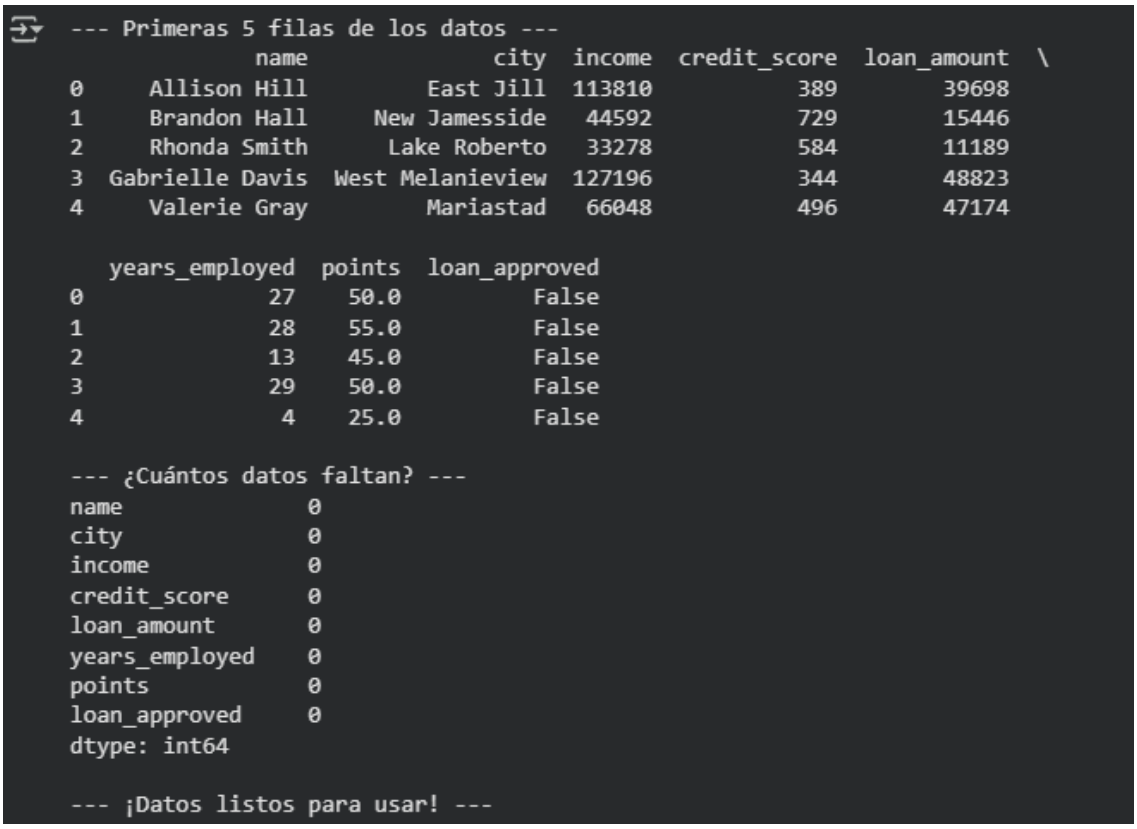
El análisis se realizó utilizando la biblioteca scikit-learn en un entorno de Google Colab.

2.1. Preparación de Datos

El primer paso fue cargar el dataset, inspeccionar su estructura y prepararlo para el modelado.

```
[9]
✓ 0s
# --- 1. Importación de herramientas necesarias ---

# Se importa "pandas" para trabajar con los datos en formato de tabla.
import pandas as pd

# Se importa "train_test_split" para separar los datos en conjuntos.
from sklearn.model_selection import train_test_split

# --- 2. Carga del archivo de datos ---

# Se lee el archivo "loan_approval.csv".
df = pd.read_csv("/loan_approval.csv")

# --- 3. Revisión de los datos (para el informe) ---

# Se muestran las primeras 5 filas para entender la estructura de los datos.
print("--- Primeras 5 filas de los datos ---")
print(df.head(), "\n")

# Se cuentan cuántos datos faltan en cada columna.
print("--- ¿Cuántos datos faltan? ---")
print(df.isnull().sum(), "\n")

# --- 4. Preparación de los datos para el análisis ---

# Se define "X" (las características) con las columnas que se usarán para predecir.
X = df[['income', 'credit_score', 'loan_amount', 'years_employed', 'points']]

# Se define "y" (la variable objetivo) que es lo que se quiere predecir.
# Se convierten los valores True/False a 1/0 para que el modelo los entienda.
y = df['loan_approved'].astype(int)

# --- 5. División de los datos para entrenamiento y prueba ---

# Se separan los datos en conjuntos de entrenamiento (para que el modelo aprenda)
# y prueba (para evaluar su rendimiento).
# Se usa el 25% de los datos para probar (test_size=0.25).
# random_state=42 asegura que la división sea la misma en cada ejecución.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

print("--- ¡Datos listos para usar! ---")
```

Figura 1. *Código de exploración inicial del dataset y comprobación de valores nulos.*



```
--- Primeras 5 filas de los datos ---
                name              city  income  credit_score  loan_amount  \
0     Allison Hill         East Jill  113810           389        39698
1     Brandon Hall     New Jamesside   44592           729        15446
2     Rhonda Smith      Lake Roberto   33278           584        11189
3  Gabrielle Davis  West Melanieview  127196           344        48823
4     Valerie Gray         Mariastad   66048           496        47174

   years_employed  points  loan_approved
0              27    50.0          False
1              28    55.0          False
2              13    45.0          False
3              29    50.0          False
4               4    25.0          False

--- ¿Cuántos datos faltan? ---
name              0
city              0
income            0
credit_score      0
loan_amount       0
years_employed    0
points            0
loan_approved     0
dtype: int64

--- ¡Datos listos para usar! ---
```

Figura 2. *Resultados de la exploración inicial del dataset y comprobación de valores nulos.*

2.2. Modelo 1: Regresión Logística

Se implementó un modelo de Regresión Logística, un algoritmo lineal que estima probabilidades usando la función sigmoide.

```
[14]
✓ 0 s
```

```python
# --- 1. Importación del Modelo 1 y las Métricas ---

# Se importa el modelo de Regresión Logística
from sklearn.linear_model import LogisticRegression

# Se importan las herramientas para evaluar el modelo
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# --- 2. Creación y Entrenamiento del Modelo 1 ---

# Se crea una instancia del modelo. max_iter=1000 se usa para aumentar las iteraciones.
modelo_log = LogisticRegression(max_iter=1000)

# Se "entrena" el modelo utilizando los datos de entrenamiento (X_train, y_train).
modelo_log.fit(X_train, y_train)

# --- 3. Evaluación del Modelo 1 ---

# Se solicitan las predicciones del modelo para los datos de prueba (X_test).
y_pred_log = modelo_log.predict(X_test)

# --- 4. Mostrar los Resultados (Para el informe) ---

print("--- RESULTADOS: Regresión Logística ---")

# La Precisión (Accuracy) compara las respuestas correctas (y_test) con las predicciones (y_pred_log).
# Indica el porcentaje de aciertos.
print("Precisión:", accuracy_score(y_test, y_pred_log))

# La Matriz de Confusión muestra cuántos "Sí" y "No" fueron predichos correctamente.
print("\nMatriz de Confusión:\n", confusion_matrix(y_test, y_pred_log))

# El Reporte de Clasificación proporciona un resumen detallado de las métricas.
print("\nReporte de Clasificación:\n", classification_report(y_test, y_pred_log))
```

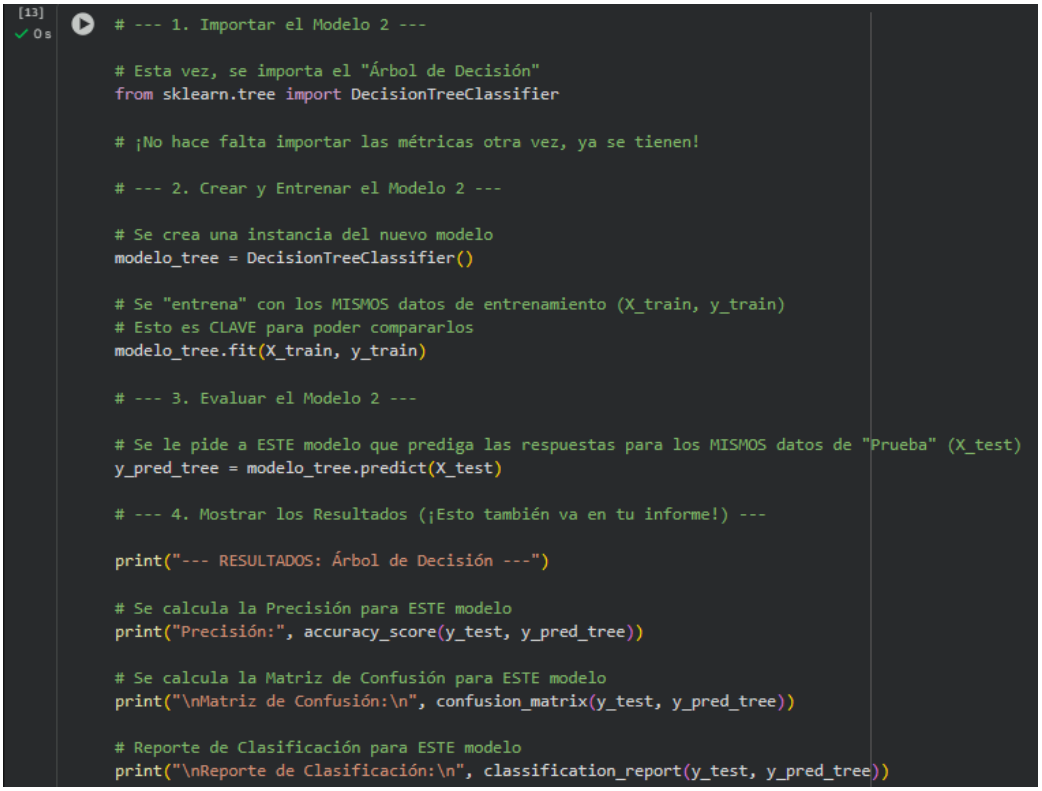
Figura 3. *Código del rendimiento del modelo de Regresión Logística.*

```
--- RESULTADOS: Regresión Logística ---
Precisión: 1.0

Matriz de Confusión:
 [[278   0]
 [  0 222]]

Reporte de Clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       278
           1       1.00      1.00      1.00       222

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500
```

Figura 4. *Resultados de rendimiento del modelo de Regresión Logística.*

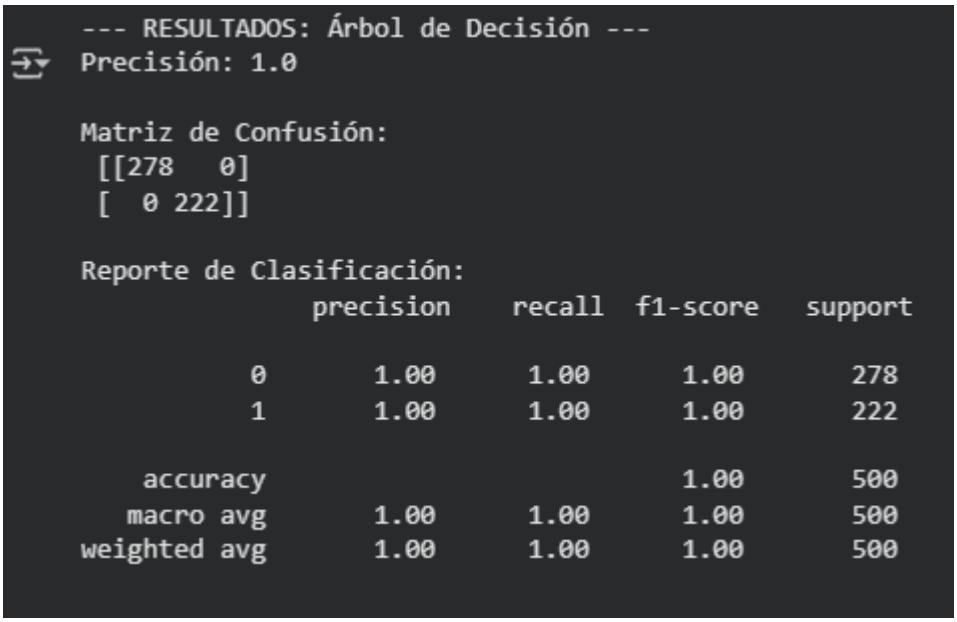
2.3. Modelo 2: Árbol de Decisión

Se implementó un modelo de Árbol de Decisión, un algoritmo no lineal que divide los datos basándose en reglas de decisión.



```
[13]
✓ 0s
# --- 1. Importar el Modelo 2 ---

# Esta vez, se importa el "Árbol de Decisión"
from sklearn.tree import DecisionTreeClassifier

# ¡No hace falta importar las métricas otra vez, ya se tienen!

# --- 2. Crear y Entrenar el Modelo 2 ---

# Se crea una instancia del nuevo modelo
modelo_tree = DecisionTreeClassifier()

# Se "entrena" con los MISMOS datos de entrenamiento (X_train, y_train)
# Esto es CLAVE para poder compararlos
modelo_tree.fit(X_train, y_train)

# --- 3. Evaluar el Modelo 2 ---

# Se le pide a ESTE modelo que prediga las respuestas para los MISMOS datos de "Prueba" (X_test)
y_pred_tree = modelo_tree.predict(X_test)

# --- 4. Mostrar los Resultados (¡Esto también va en tu informe!) ---

print("--- RESULTADOS: Árbol de Decisión ---")

# Se calcula la Precisión para ESTE modelo
print("Precisión:", accuracy_score(y_test, y_pred_tree))

# Se calcula la Matriz de Confusión para ESTE modelo
print("\nMatriz de Confusión:\n", confusion_matrix(y_test, y_pred_tree))

# Reporte de Clasificación para ESTE modelo
print("\nReporte de Clasificación:\n", classification_report(y_test, y_pred_tree))
```

Figura 5. *Código del rendimiento del modelo de Árbol de Decisión.*



```
--- RESULTADOS: Árbol de Decisión ---
Precisión: 1.0

Matriz de Confusión:
 [[278   0]
 [  0 222]]

Reporte de Clasificación:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       278
           1       1.00      1.00      1.00       222

    accuracy                           1.00       500
   macro avg       1.00      1.00      1.00       500
weighted avg       1.00      1.00      1.00       500
```

Figura 6. *Resultados de rendimiento del modelo de Árbol de Decisión.*

3. Análisis y Discusión

Para facilitar la comparación, los resultados de ambos modelos se consolidan en la siguiente tabla.

Tabla 1. *Comparativa de Métricas de Rendimiento*

Métrica	Modelo 1: Regresión Logística	Modelo 2: Árbol de Decisión
Precisión (Accuracy)	1.0	1.0
Matriz de Confusión	[[278, 0], [0, 222]]	[[278, 0], [0, 222]]

3.1. Análisis de la Similitud en Resultados

El hallazgo más notable es que ambos modelos, a pesar de funcionar de maneras fundamentalmente diferentes, obtuvieron un rendimiento idéntico y perfecto (100% de precisión). No se registró ni un solo error (falso positivo o falso negativo) en las 500 muestras del conjunto de prueba.

Esta similitud se debe, muy probablemente, a la naturaleza del propio dataset:

1. Problema Linealmente Separable: La Regresión Logística es un modelo lineal. Su éxito perfecto sugiere que los datos son "linealmente separables"; es decir, existe una "línea" (o hiperplano) que puede dividir perfectamente a los clientes "Aprobados" (1) de los "No Aprobados" (0) basándose en las características dadas.
2. Capacidad del Árbol de Decisión: Un Árbol de Decisión, aunque es un modelo no lineal, también es perfectamente capaz de encontrar esta separación. Si existe una regla simple (ej. "SI credit_score > 700 Y income > 50000 ENTONCES 1..."), el árbol la encontrará y la usará para clasificar sin errores.

La similitud no se debe a que los modelos sean iguales, sino a que el problema planteado por este dataset es lo suficientemente "limpio" o "fácil" como para que ambos enfoques (lineal y no lineal basado en reglas) pudieran encontrar la solución óptima y perfecta.

4. Conclusiones

Se implementaron y compararon exitosamente dos algoritmos de clasificación supervisada. Ambos, la Regresión Logística y el Árbol de Decisión, demostraron ser capaces de predecir la aprobación de préstamos en el dataset de prueba con un 100% de precisión.

Se concluye que, para este conjunto de datos específico, ambos modelos son igualmente viables y efectivos. La elección entre uno u otro en un escenario real podría depender de otros factores, como la necesidad de interpretabilidad.

Referencias

Edward, A. D. (2023). *Loan Approval Dataset* (Version 1). Kaggle. https://www.kaggle.com/datasets/anishdevedward/loan-approval-dataset

McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 445, 51–56. https://doi.org/10.25080/Majora-92bf1922-00a

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.

Perez, E. D. (2025). *Semana 11 - Las máquinas aprenden a clasificar* [Diapositivas de clase]. Introducción a Machine Learning. Universidad de Cundinamarca.