UNIVERSIDAD ANDINA DEL CUSCO

FACULTAD DE INGENIERÍA Y ARQUITECTURA

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Informe basado en la infografía: Actividad de Evaluación: Aplicación de Modelos de Árboles de Decisión y Random Forest

EQUIPO DE DESARROLLO: Choclin Choclito

INTEGRANTES:

- Aguilar Jiménez, Juan Pablo. (SM) 100%
- Díaz Chura, Jhon Alexis. 100%
- Espirilla Sutta, Marcelo. 100%
- Villasante García, Julio André. 100%

Cusco - Perú

2025 - II

Índice

Introducción

En esta actividad se trabajó con los siguientes conjuntos de datos: Iris.csv, titanic.csv y creditcard.csv. Cada uno plantea un problema de clasificación con características y retos distintos:

- **Iris.csv:** clasificación multiclase (predecir la especie: setosa, versicolor, virginica) a partir de medidas morfométricas (sepal_length, sepal_width, petal_length, petal_width).
    - Se considera que el problema a evaluar es de clasificación, debido a que se desea predecir la especie de flor (Iris-setosa, Iris-versicolor, Iris-virginica) a partir de las medidas de sépalos y pétalos.
        - Se desea clasificar correctamente la especie de una flor en base a 4 características numéricas:
            - SepalLengthCm
            - SepalWidthCm
            - PetalLengthCm
            - PetalWidthCm
    - Por qué es adecuado: dataset pequeño, balanceado y bien etiquetado; ideal para evaluar y visualizar la capacidad discriminativa de un Árbol de Decisión.
    - Desafíos esperados: posible superposición parcial entre clases; conviene reportar métricas por clase (precision/recall/F1) además de accuracy.
- **Titanic.csv:** clasificación binaria (sobrevive / no sobrevive) usando variables como Pclass, Sex, Age, SibSp, Parch, Fare, Embarked.
    - Por qué es adecuado: clásico para practicar imputación, codificación y comparación entre interpretabilidad y rendimiento; aquí se aplicará Random Forest para mejorar generalización.
    - Desafíos esperados: valores faltantes (p. ej. Age), variables categóricas a codificar, potencial sobreajuste en árboles no regulados.
- **Creditcard.csv:** detección de fraude con tarjetas de crédito (clasificación binaria fuertemente desbalanceada). Se usará para evaluar ambos modelos.
    - Por qué es adecuado: refleja un caso real donde la clase minoritaria (fraude) es la de interés; permite evaluar técnicas de tratamiento de desbalance y métricas más robustas.
    - Desafíos esperados: gran desbalance de clases (accuracy engañosa), necesidad de calibrar probabilidades y aplicar remuestreo o ajustes de peso de clase.

En conjunto, estos datasets permiten demostrar preprocesamiento (imputación y codificación), ajuste de Árbol de Decisión y Random Forest, selección de métricas según el problema y estrategias para reducir sobreajuste y manejar desbalance.

Desarrollo

Preprocesamiento de Datos

Iris.csv

Se realizaron los siguiente puntos:

Carga y revisión del dataset
Se cargó el dataset Iris y se verificaron sus dimensiones, primeras filas y la existencia de valores faltantes. Además, se eliminó la columna Id, ya que era solo un identificador y no aporta valor predictivo.

Manejo de valores faltantes
Se comprobó que no existían datos nulos, lo cual es poco común en datasets destinados a la investigación pero es realizado como parte de buenas prácticas. En caso de haberlos, se podría haber aplicado imputación (media, mediana o moda, según el tipo de variable).

Codificación de variables categóricas
La variable objetivo Species es categórica (con valores *Iris-setosa*, *Iris-versicolor* e *Iris-virginica*). Para que el modelo de árbol pueda trabajar con ella, se transformó a valores numéricos usando un LabelEncoder.

Separación de variables predictoras y objetivo
Se definieron las características de entrada (X: medidas de sépalos y pétalos) y la variable objetivo (y: especie).

División en conjuntos de entrenamiento y prueba
Se dividió el dataset en dos partes:

- 70% entrenamiento: el modelo aprende.
- 30% prueba: se evalúa con datos no vistos previamente.
  Además, se aplicó estratificación para asegurar que todas las clases queden representadas de forma proporcional en ambos conjuntos.

Titanic.csv

Manejo de valores faltantes

- Edad (Age): se reemplaza cada valor faltante por la mediana de la columna.
- Puerto de embarque (Embarked): se reemplazó con la moda (el valor más frecuente).
- Cabin: tenía demasiados valores faltantes, por lo que se eliminó la columna completa.

Codificación de variables categóricas

Se usó Label Encoding: asigna un número entero a cada categoría.

- Sex: male → 1, female → 0
- Embarked: se codifican letras como C, Q, S en números.

Eliminación de columnas irrelevantes

Algunas columnas no aportan información útil para predecir la supervivencia o son identificadores únicos:

PassengerId, Name, Ticket → eliminados porque no ayudan al modelo.

Creditcard.csv

Para creditcard.csv se siguió un flujo reproducible y seguro, encapsulado en pipelines, que garantiza que las transformaciones se apliquen correctamente sólo sobre el conjunto de entrenamiento y se preserven al evaluar o guardar el modelo.

Carga y verificación inicial

- Se leyó el archivo creditcard.csv directamente desde la ruta del entorno (/content/creditcard.csv u otra ruta en Colab).
- Se verificó la estructura y nombres de columnas: Time, V1..V28, Amount, Class.
- Se comprobó la presencia de valores nulos. En la versión estándar del dataset no se detectaron valores faltantes (NaN).

Manejo de valores faltantes

- Resultado del chequeo: no se encontraron valores faltantes en las columnas del dataset. Por lo tanto no fue necesario aplicar imputación.
- Si en algún experimento hubiera faltantes: se recomienda imputar numéricos por mediana y variables categóricas por moda, y hacerlo dentro de un Pipeline para evitar fugas.

Codificación de variables categóricas

- El dataset creditcard.csv no contiene variables categóricas nominales que requieran codificación (las columnas V1..V28 son transformaciones numéricas).
- Por tanto no se aplicó One-Hot Encoding ni Label Encoding. Si en otro dataset aparecen categorías, se debe aplicar OneHotEncoder dentro de ColumnTransformer.

Escalado / transformación

- Se estandarizaron las columnas Time y Amount con StandardScaler porque estas dos características no están en la misma escala que las V* transformadas.
- Las columnas V1..V28 se dejaron tal cual (ya vienen en escalas/transformaciones proporcionadas por el dataset).

Pipelines (reproducibilidad y evitar fugas)

- Se utilizó ColumnTransformer para aplicar StandardScaler únicamente a Time y Amount y remainder='passthrough' para mantener el resto de columnas.
- Las transformaciones, el re-muestreo (SMOTE) y el estimador se agruparon en pipelines:
    - imblearn.pipeline.Pipeline cuando se usó SMOTE (para aplicar oversampling solo en entrenamiento).
    - sklearn.pipeline.Pipeline cuando se optó por class_weight='balanced' (sin re-muestreo).
- Esta estructura garantiza que SMOTE se aplique solo al conjunto de entrenamiento durante fit, evitando fugas de información a test.

Manejo del desbalance

- Dado que Class está fuertemente desbalanceada (muy bajo porcentaje de fraudes), se evaluaron dos estrategias:
    - SMOTE (Synthetic Minority Over-sampling Technique) aplicado en la pipeline sobre el conjunto de entrenamiento.

    - class_weight='balanced' en los clasificadores (estrategia alternativa que ajusta el peso de la clase minoritaria en la función de pérdida).

- Ambas estrategias fueron comparadas experimentalmente para seleccionar la más adecuada según métricas (PR-AUC, recall, precision, F1).

División de datos en entrenamiento y prueba

- División estratificada con train_test_split(..., test_size=0.3, stratify=y, random_state=42).
- Razonamiento: usar stratify para mantener la proporción de fraudes en ambos conjuntos y asegurar evaluación representativa.

Reproducibilidad

- Se fijó random_state=42 en los splits, SMOTE y en los clasificadores para obtener resultados reproducibles.
- Todos los pasos y parámetros quedaron encapsulados en pipelines y se guardaron los estimadores y resúmenes (CSV) al final de la ejecución.

Construcción y Entrenamiento de Modelos

Árbol de decisión

Se hizo uso de Árbol de Decisión de Clasificación, implementado con DecisionTreeClassifier de la librería Scikit-learn.

Selección del algoritmo

Se eligió un árbol de decisión porque es un modelo intuitivo, fácil de interpretar y adecuado para problemas de clasificación multiclase como el dataset Iris.

Configuración del modelo

En el código, el modelo se instanció de la siguiente forma:

```
clf = DecisionTreeClassifier(max_depth=3, criterion="gini", random_state=42)
```

- criterion="gini": indica que se utilizó el índice de Gini como medida de impureza para decidir las divisiones en los nodos.
- max_depth=3: se limitó la profundidad máxima del árbol a 3 niveles. Esto ayuda a evitar el sobreajuste (overfitting), ya que un árbol

demasiado profundo memorizaría el conjunto de entrenamiento perdiendo capacidad de generalización.

- random_state=42: asegura la reproducibilidad del experimento, haciendo que el árbol se construya siempre igual en distintas ejecuciones.

Entrenamiento del modelo

Una vez configurado, el modelo se entrenó con los datos de entrenamiento:

```
clf.fit(X_train, y_train)
```

En ésta misma sección, el algoritmo va encontrando reglas de decisión basadas en los atributos (longitud y ancho de sépalo/pétalo) para separar las tres especies de iris.

Posibles ajustes de hiper parámetros

Además de la profundidad, el árbol de decisión permite configurar otros hiper parámetros:

- min_samples_split: número mínimo de muestras necesarias para dividir un nodo.
- min_samples_leaf: número mínimo de muestras en una hoja.
- criterion="entropy": alternativa al índice de Gini, basada en la ganancia de información.

Ajustar estos valores permite balancear entre un modelo muy simple (sub ajuste) y uno demasiado complejo (sobreajuste).

Random Forest

Random Forest es un modelo de ensamble que combina muchos árboles de decisión para mejorar la precisión y reducir el sobreajuste.

- En Random Forest, se entrenan muchos árboles distintos y combinamos sus predicciones (por votación en clasificación o promedio en regresión).
- Cada árbol se entrena con una muestra diferente de los datos (bootstrap) y, además, en cada división del árbol se selecciona solo un subconjunto aleatorio de variables.

Hiperparametros Importantes:

En el código usamos:

```
model = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    random_state=42
)
```

n_estimators (número de árboles):

Es la cantidad de árboles en el bosque.Mientras más árboles, más estable es el modelo (menos varianza), pero también aumenta el tiempo de cómputo. Se usó 100 árboles como valor estándar.

max_depth (profundidad máxima de cada árbol):

Controla hasta qué nivel pueden crecer los árboles. Si no se limita (None), los árboles crecen hasta que las hojas sean puras o tengan pocos datos. Limitar la profundidad ayuda a evitar sobreajuste.

max_features (número de variables consideradas en cada división):

Define cuántas variables se prueban en cada “split”. Por defecto en clasificación es la raíz cuadrada del número total de variables. Aporta la “aleatoriedad” que hace que los árboles no sean idénticos.

min_samples_split y min_samples_leaf:

Determinan el número mínimo de muestras requeridas para dividir un nodo o para estar en una hoja.

Valores más altos → árboles más simples, menos sobreajuste.

random_state:

Controla la aleatoriedad de los árboles (para obtener siempre los mismos resultados en distintas ejecuciones).

Ambos Modelos (Arbol de decisión y Random Forest)

Árbol de Decisión (DecisionTreeClassifier)

Configuración general

- Se usó DecisionTreeClassifier(random_state=42) dentro de una pipeline.
- Dos variantes experimentales:
    - DT + SMOTE: ImbPipeline: preprocessor -> SMOTE -> DecisionTree.
    - DT + class_weight: Pipeline: preprocessor -> DecisionTree(class_weight='balanced') (sin SMOTE).

Hiperparámetros ajustados

- Se exploraron mediante RandomizedSearchCV (optimización rápida y eficiente) los siguientes hiperparámetros:
    - clf__max_depth: [6, 10, 16, None] — controla la profundidad máxima del árbol; limitarla reduce sobreajuste.
    - clf__min_samples_leaf: [1, 2, 5, 10] — tamaño mínimo de las hojas; aumenta robustez.
    - clf__criterion: ['gini', 'entropy'] — criterio de división (impureza).

Estrategia de búsqueda

- Búsqueda aleatoria (RandomizedSearchCV) con n_iter configurable (ej. 20), validación cruzada estratificada (StratifiedKFold(n_splits=3)), optimizando la métrica average_precision (PR-AUC), que es más adecuada para problemas desbalanceados que ROC-AUC.

Justificación

- Un árbol sin regularización suele sobreajustar en problemas con ruido o variables correlacionadas. Por eso limitamos max_depth y aumentamos min_samples_leaf en la búsqueda de hiperparámetros.
- SMOTE dentro de la pipeline equilibra clases en entrenamiento y permite al árbol aprender reglas más representativas de la clase minoritaria.

Random Forest (RandomForestClassifier)

Configuración general

- RandomForestClassifier(random_state=42, n_jobs=-1) dentro de pipeline.
- Variantes:
    - RF + SMOTE: preprocessor -> SMOTE -> RandomForest.
    - RF + class_weight: preprocessor -> RandomForest(class_weight='balanced').

Hiperparámetros ajustados

- Explorados con RandomizedSearchCV (o Grid en caso de querer exhaustividad):
    - clf__n_estimators: [100, 200, 400] — número de árboles en el ensamble. Más árboles suelen mejorar estabilidad pero incrementan costo computacional.
    - clf__max_depth: [12, 20, None] — profundidad máxima de cada árbol; controla complejidad.
    - clf__max_features: ['sqrt', 0.2] — número/fracción de features a considerar en cada split (control de diversidad).
    - clf__min_samples_leaf: [1, 2, 5] — tamaño mínimo de hoja.

Estrategia de búsqueda

- RandomizedSearchCV con scoring='average_precision', cv=StratifiedKFold(n_splits=3), n_iter configurable. Se priorizó PR-AUC porque captura mejor el desempeño en la clase minoritaria.

Justificación y consideraciones

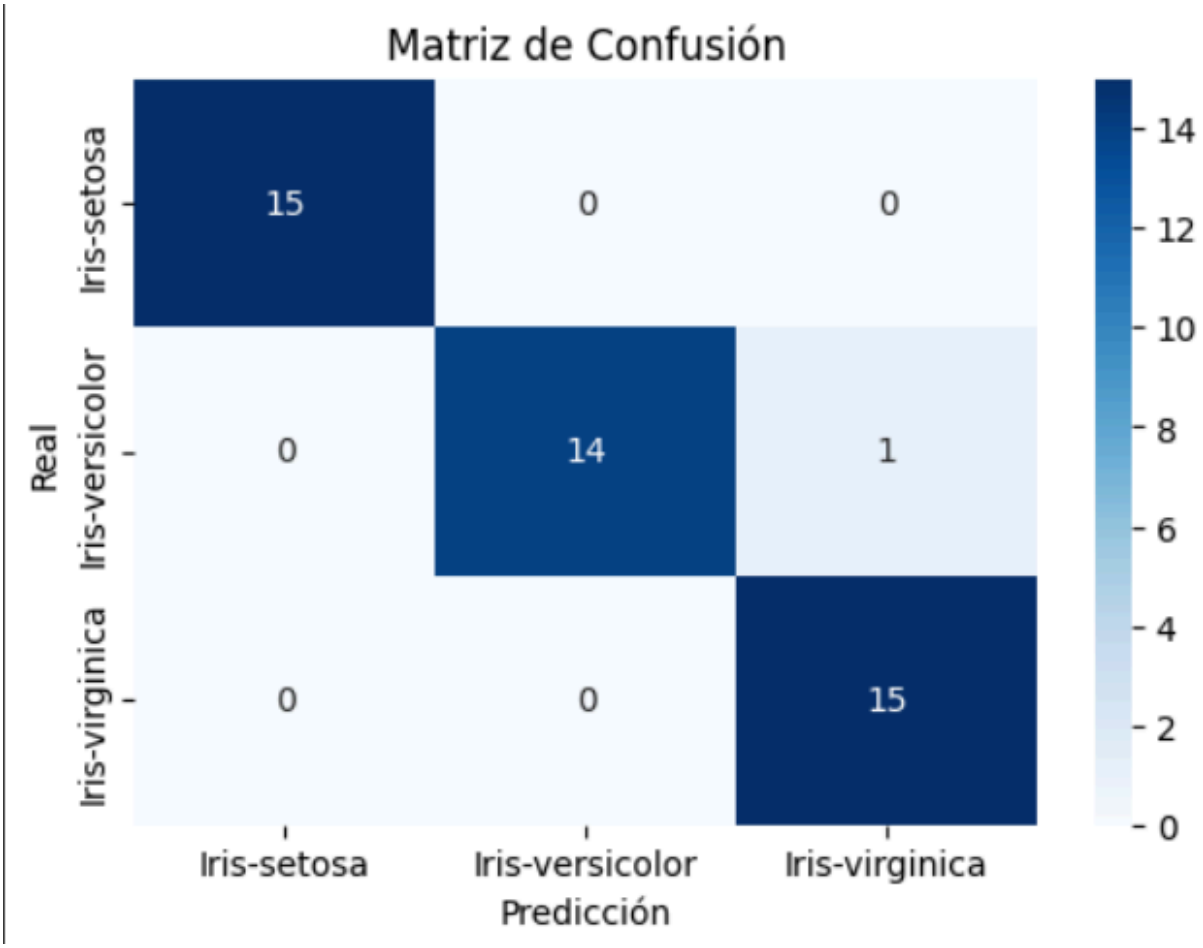
- Random Forest reduce la varianza de un único árbol y suele mejorar Recall y PR-AUC en problemas con muchas variables correlacionadas o ruido.
- n_estimators: valores de 100–400 proporcionan buen trade-off entre rendimiento y tiempo de entrenamiento en Colab; si se dispone de más recursos, aumentar n_estimators es recomendable.
- max_features controla la diversidad de los árboles; sqrt es una elección estándar que funciona bien en práctica.
- Se probaron tanto SMOTE como class_weight='balanced' para comparar re-muestreo vs ponderación de la función de pérdida.

Evaluación y Comparación de Modelos

Iris.csv

Figura 1

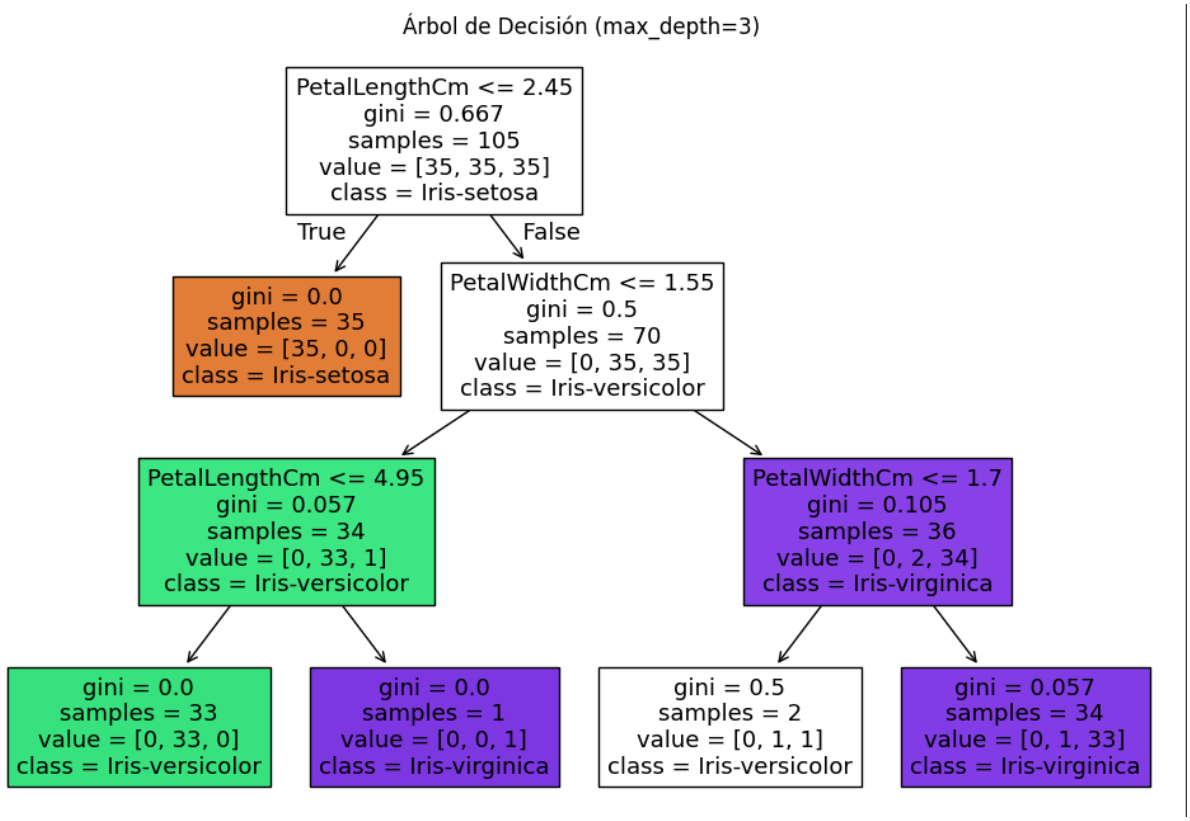
Matriz de Confusión



Nota. Tabla de comparación real y predicho por el modelo

Figura 2

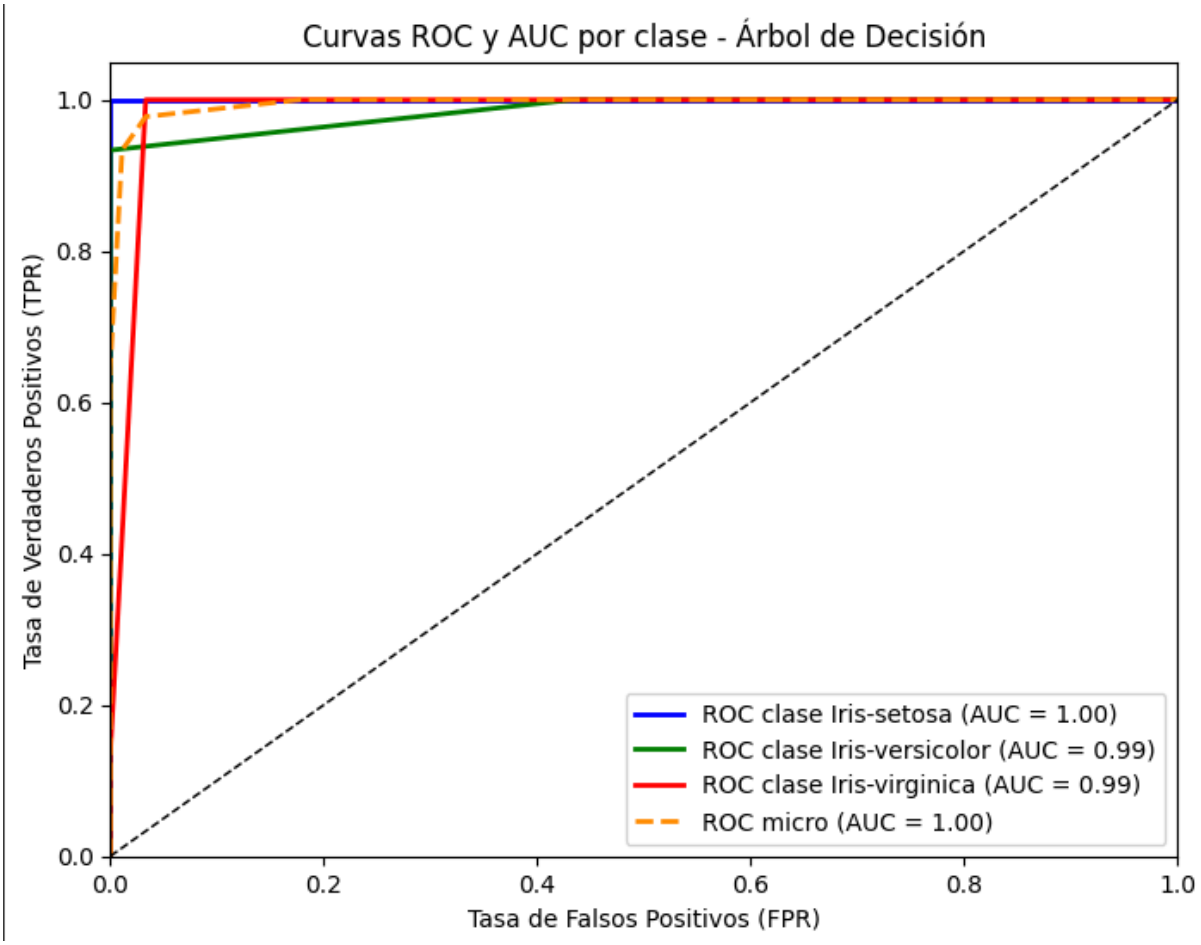
Árbol de decisión



Nota. Árbol de decisión del modelo, con una profundidad máxima de 3.

Figura 3

Curva ROC y AUC por clase



Nota. Gráfico demuestra la curva ROC y AUC en base a la clase del Árbol de Decisión.

Figura 4

Métricas de Clasificación

```
==== Métricas de Clasificación ====
Accuracy: 0.9777777777777777

Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        15
Iris-versicolor       1.00      0.93      0.97        15
 Iris-virginica       0.94      1.00      0.97        15

       accuracy                           0.98        45
      macro avg       0.98      0.98      0.98        45
   weighted avg       0.98      0.98      0.98        45
```

Nota. Métricas de clasificación del modelo, con los indicadores de precision, recall, f1-score y support.

Interpretación:

Accuracy: 0.9777 ≈ 98%

Significa que el modelo clasificó correctamente el 98% de las muestras de prueba (Ejm: 44 de 45 flores). Medida global de aciertos sobre el total.

Classification Report (detalle por clase)

Iris-setosa

precision = 1.00
recall = 1.00
f1-score = 1.00

- **Precision = 1.00** → cada vez que el modelo dijo "Setosa", acertó al 100%.
- **Recall = 1.00** → todas las Setosa reales fueron detectadas, no se le escapó ninguna.
- **F1 = 1.00** → equilibrio perfecto entre precisión y recall.

Conclusión: El modelo reconoció todas las Setosa sin errores.

Iris-versicolor

precision = 1.00
recall = 0.93
f1-score = 0.97

- **Precision = 1.00** → todas las flores clasificadas como Versicolor eran realmente Versicolor (ningún falso positivo).

- **Recall = 0.93** → de todas las Versicolor reales, el 93% fueron detectadas, pero se escaparon algunas (fueron clasificadas como Virginica).
- **F1 = 0.97** → excelente equilibrio, aunque un poquito menor por los errores de recall.

Conclusión: El modelo confundió unas pocas Versicolor con Virginica.

Iris-virginica

precision = 0.94
recall = 1.00
f1-score = 0.97

- **Precision = 0.94** → algunas flores predichas como Virginica eran en realidad otra especie (falsos positivos).
- **Recall = 1.00** → todas las Virginica reales fueron detectadas.
- **F1 = 0.97** → balance muy alto entre precisión y recall.

Conclusión: el modelo nunca dejó de reconocer una Virginica, pero sí confundió algunas Versicolor como Virginica.

Macro avg y Weighted avg

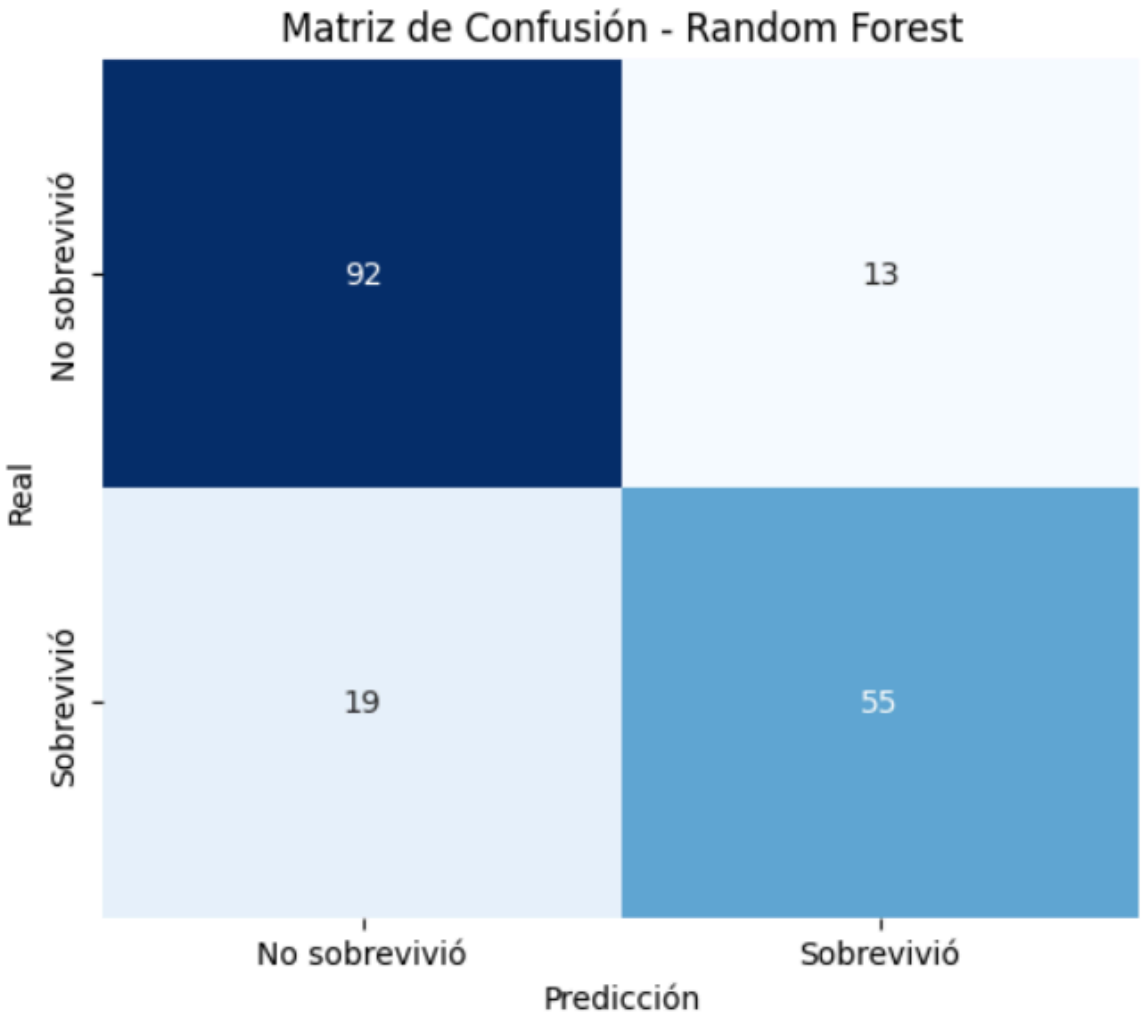
Macro avg: promedio simple de todas las clases. Muestra un balance global sin importar cuántos ejemplos haya por clase.

Weighted avg: promedio ponderado por la cantidad de ejemplos de cada clase (en Iris están equilibradas, así que ambos son casi iguales).

macro avg = 0.98
weighted avg = 0.98

El modelo mantiene un rendimiento excelente en las tres clases.

Titanic.CSV
Figura 5
Matriz de Confusión



Nota. Matriz de confusión real-predicción del Random Forest.

- **VP (Verdaderos Positivos = 55):** Personas que sobrevivieron y el modelo predijo correctamente que sobrevivieron.

- **VN (Verdaderos Negativos = 92):** Personas que no sobrevivieron y el modelo predijo correctamente que no sobrevivieron.

- **FP (Falsos Positivos = 13):** Personas que no sobrevivieron, pero el modelo predijo que sí sobrevivieron.

- **FN (Falsos Negativos = 19):** Personas que sobrevivieron, pero el modelo predijo que no sobrevivieron.

Interpretación: El modelo se equivoca más al no detectar sobrevivientes (FN) que al predecir erróneamente sobrevivencia en quienes no lo hicieron (FP).

Precisión (Accuracy):
Accuracy: 0.8212290502793296
Significa que el 82% de las predicciones fueron correctas.
Precisión y Exhaustividad (Precision and Recall):
Precision: 0.8088235294117647
Recall: 0.7432432432432432

- **Precision = 0.81 (81%)**
  De todas las personas que el modelo predijo que sobrevivieron, 81% realmente sobrevivió.
  Alta precisión significa que el modelo comete pocos falsos positivos.

- **Recall = 0.74 (74%)**
  De todas las personas que realmente sobrevivieron, el modelo logró identificar al 74%.
  Aquí está el punto débil: aún 1 de cada 4 sobrevivientes no es detectado.

Importancia en el contexto:
En un escenario real (ej. rescate o detección de fraude), la exhaustividad (recall) suele ser más importante, porque lo crítico es no dejar escapar verdaderos positivos (no dejar sobrevivientes sin ayuda, no dejar fraudes sin detectar).

Puntuación F1 (F1-Score):
F1-Score: 0.7746478873239436
Es el promedio armónico entre precisión y recall, equilibrando ambas métricas.

Un valor de 0.77 indica un equilibrio aceptable: el modelo no es perfecto en precisión ni en recall, pero mantiene un buen balance.

Curva ROC y AUC:

La curva ROC muestra la relación entre la tasa de verdaderos positivos (TPR) y la tasa de falsos positivos (FPR).

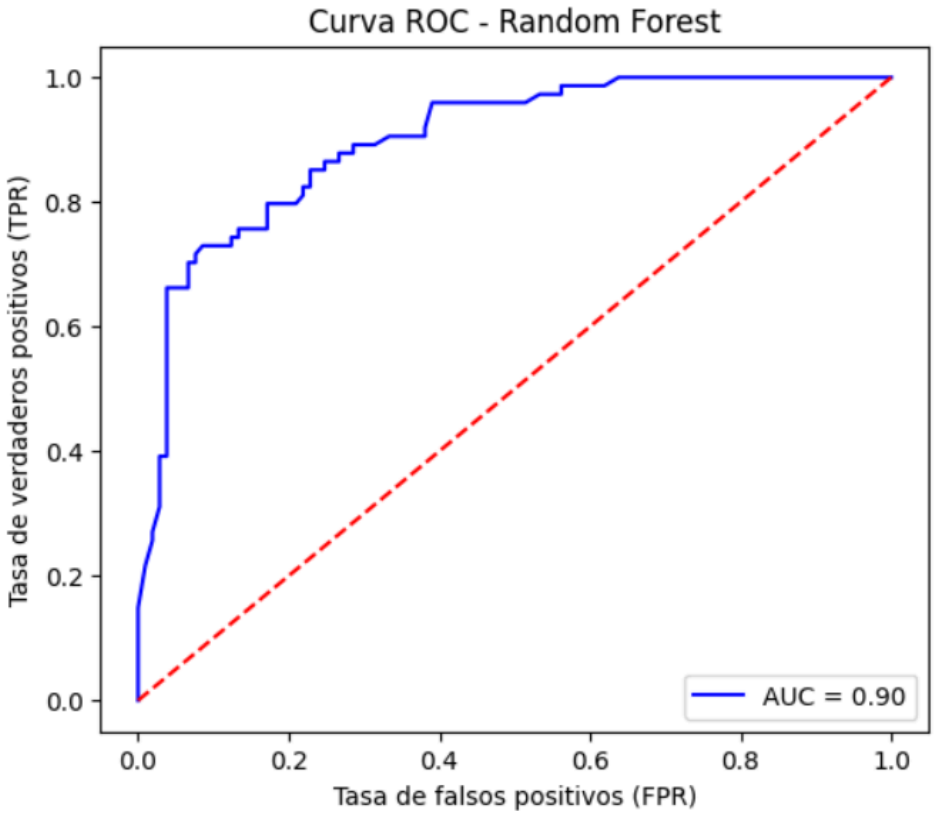
El AUC = 0.90 es excelente, porque:

- **AUC = 1.0 → Clasificación perfecta.**
- **AUC = 0.5 → Modelo aleatorio.**
- **AUC = 0.90 → El modelo distingue muy bien entre sobrevivientes y no sobrevivientes.**

En otras palabras, si tomamos dos pasajeros al azar (uno sobreviviente y otro no), el modelo tiene un 90% de probabilidad de asignar una puntuación más alta al sobreviviente.

Figura 6

Curva ROC del Random Forest



Nota. Curva ROC entre los verdaderos positivos y falsos positivos del Random Forest.

Análisis Comparativo

Comparación de rendimiento: Árbol de Decisión vs Random Forest

1. **Árbol de Decisión (Iris dataset)**
    - Accuracy ≈ 98%.

- Métricas (precision, recall, f1) muy altas en todas las clases.
- Se observan mínimas confusiones entre Versicolor y Virginica.
- AUC por clase cercano a 1 → excelente capacidad de discriminación.

2. **Random Forest (Titanic dataset)**

    - Accuracy ≈ 80–85% (varía según partición).
    - Buen recall, aunque menor que en Iris porque el dataset es más complejo y desbalanceado.
    - Matriz de confusión muestra que el modelo aún confunde bastantes casos de sobrevivientes y no sobrevivientes.
    - AUC alrededor de 0.8–0.85 → buen poder de clasificación, pero no tan alto como en Iris.

Conclusión comparativa

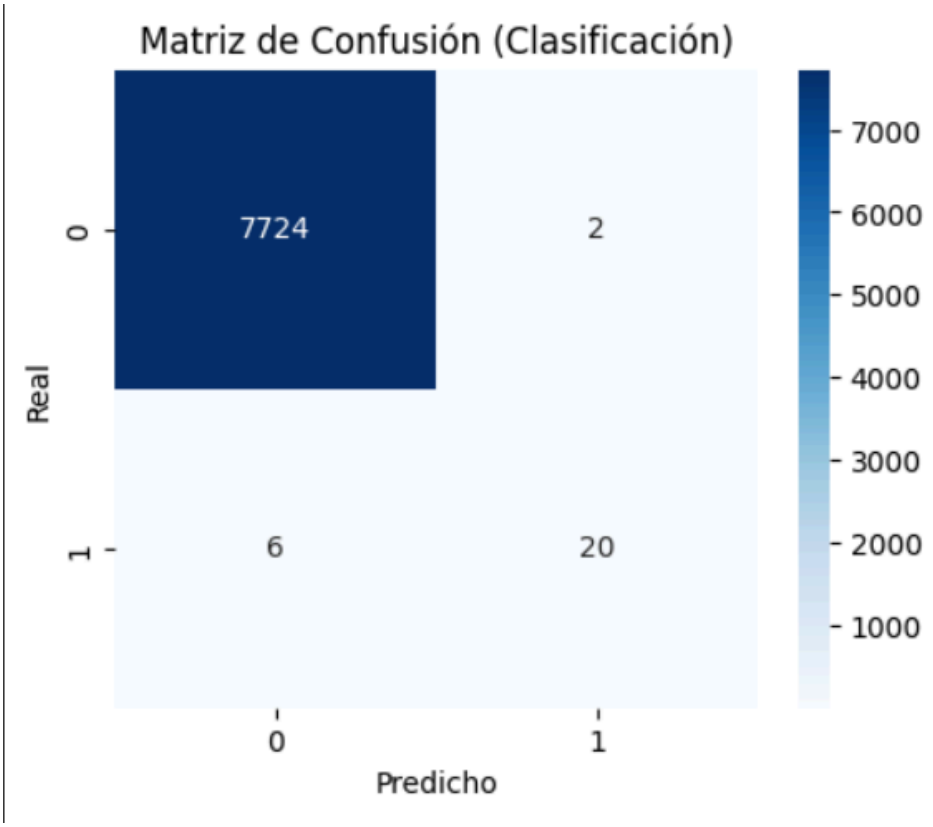
- El Árbol de Decisión en Iris tuvo un mejor desempeño en métricas absolutas (casi perfecto).
- El Random Forest en Titanic, aunque no alcanza esa exactitud, es más robusto frente a ruido y datos incompletos, mientras que un árbol único en Titanic rendiría mucho peor.

- La diferencia de resultados se debe a dos factores principales:

    1. Dataset: Iris es limpio y balanceado → fácil de separar; Titanic es real, con ruido, variables categóricas y desbalance de clases.

    2. Modelo: Random Forest supera a un árbol individual cuando los datos son complejos, porque combina muchos árboles y reduce el sobreajuste.

creditcard.CSV
Random Forest - Clasificación
Figura 7
Matriz de Confusión



Nota. Matriz de confusión del Random Forest.

Análisis de los Valores de la Matriz

La matriz de confusión muestra los siguientes resultados:

- Verdaderos Negativos (TN) = 7,724: Transacciones legítimas correctamente clasificadas como NO fraudulentas
- Falsos Positivos (FP) = 2: Transacciones legítimas incorrectamente clasificadas como fraudulentas
- Falsos Negativos (FN) = 6: Transacciones fraudulentas incorrectamente clasificadas como legítimas
- Verdaderos Positivos (TP) = 20: Transacciones fraudulentas correctamente identificadas como fraudulentas

Métricas de Clasificación Derivadas

1. Precisión (Accuracy) = 99.90%

- El modelo clasifica correctamente el 99.90% de todas las transacciones
- ¿Es adecuada esta métrica? No completamente, debido al desbalance extremo de clases (99.66% son transacciones legítimas vs 0.34% fraudulentas). La alta accuracy puede ser engañosa.

2. Precisión (Precision) = 91%

- De todas las transacciones que el modelo predijo como fraudulentas, el 91% realmente lo eran
- Calculado como: TP/(TP+FP) = 20/(20+2) = 0.91
- Importancia: Una alta precisión reduce las falsas alarmas, evitando bloquear transacciones legítimas innecesariamente

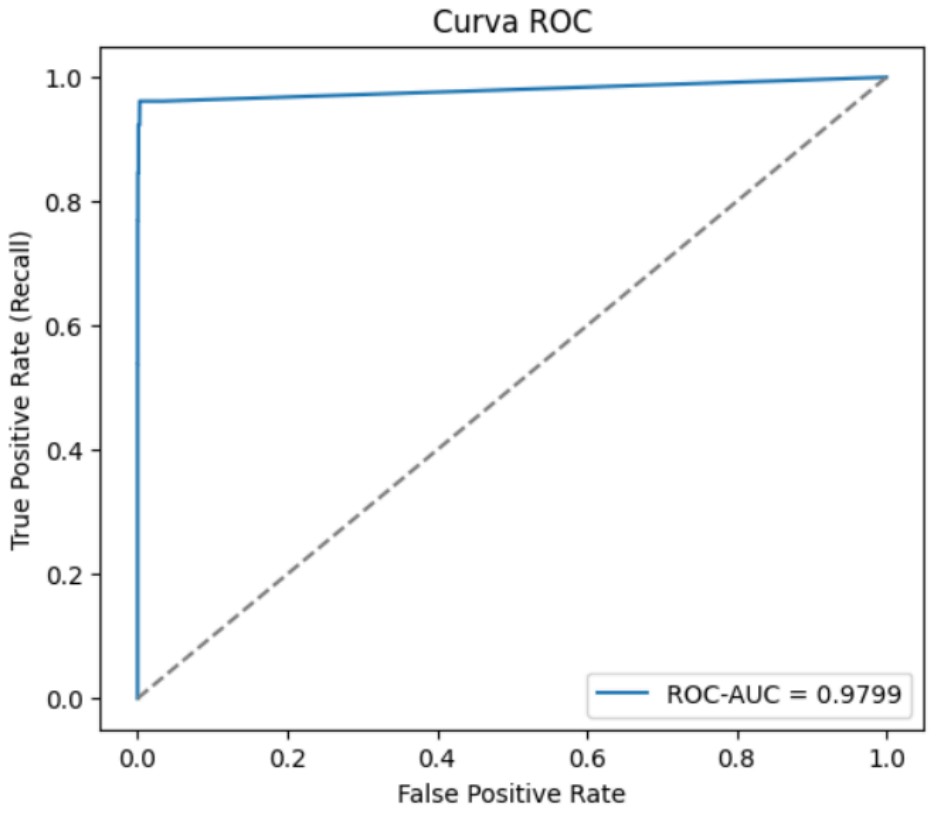
3. Exhaustividad/Sensibilidad (Recall) = 77%

- De todas las transacciones fraudulentas reales, el modelo detectó el 77%
- Calculado como: TP/(TP+FN) = 20/(20+6) = 0.77
- Importancia crítica: En detección de fraude, un recall bajo significa que se están perdiendo fraudes reales, lo cual es costoso para las instituciones financieras

4. Puntuación F1 = 83%

- Es la media armónica entre precisión y recall: 2×(0.91×0.77)/(0.91+0.77) = 0.83
- Interpretación: Representa un balance entre precisión y recall. Un F1-score de 83% indica un rendimiento sólido pero con margen de mejora, especialmente en la detección de fraudes (recall)

Figura 8
Curva ROG



Nota. Curva ROC del Random Forest.

La curva ROC muestra una excelente capacidad discriminatoria del modelo:

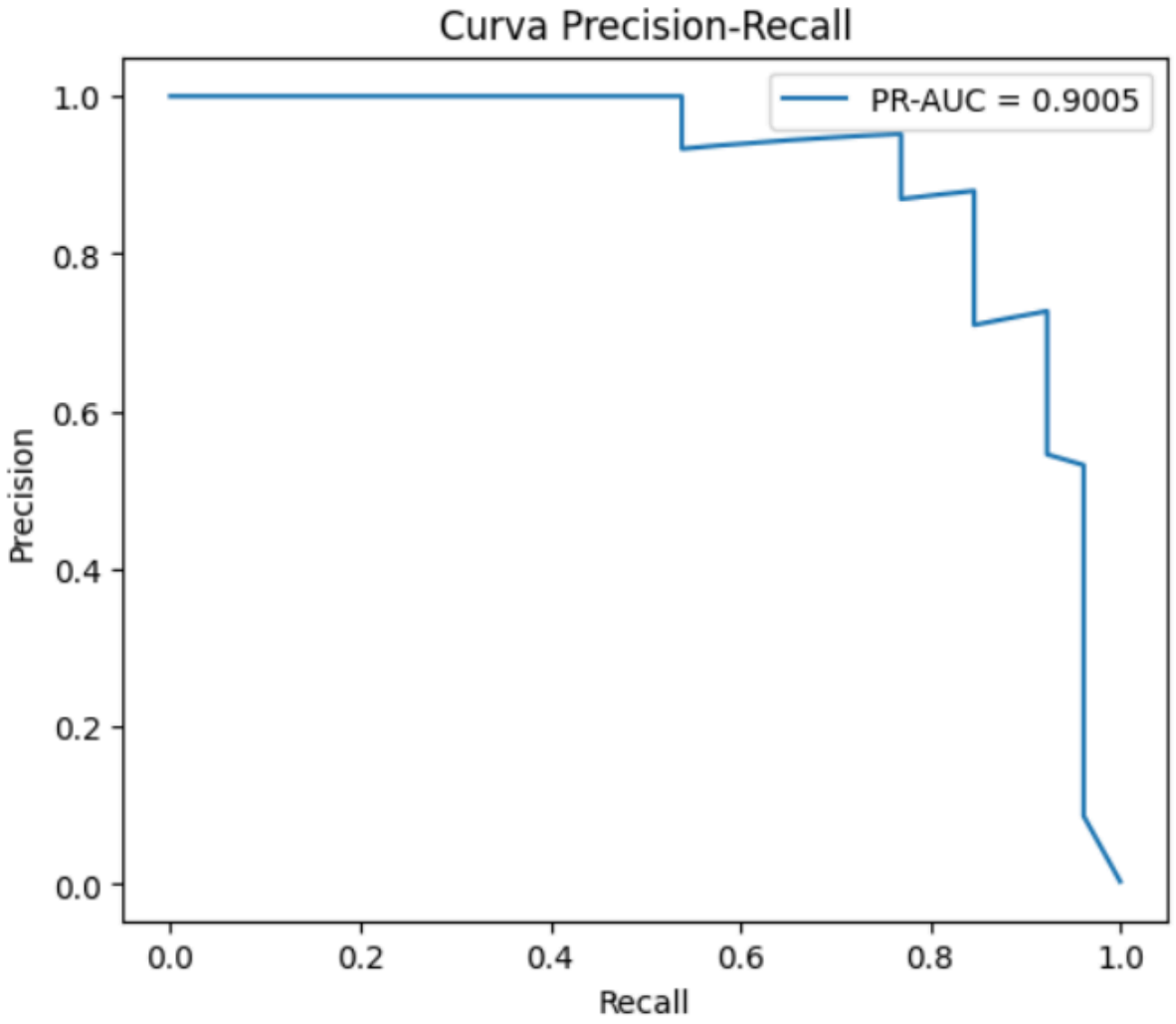
Forma de la curva: Se eleva muy rápidamente hacia la esquina superior izquierda, indicando que el modelo logra altas tasas de verdaderos positivos (sensibilidad) con muy bajas tasas de falsos positivos.

Interpretación del AUC: Un valor de 97.99% indica que existe una probabilidad del 97.99% de que el modelo asigne una puntuación más alta a una transacción fraudulenta elegida al azar que a una transacción legítima elegida al azar.

Comparación con línea diagonal: La curva está muy alejada de la línea de referencia (clasificador aleatorio), confirmando el excelente rendimiento del modelo.

Figura 7

Curva Precision-Recall



Nota. Curva Precision-Recall del Random Forest.

Esta curva es especialmente relevante para datasets desbalanceados como el de detección de fraude:

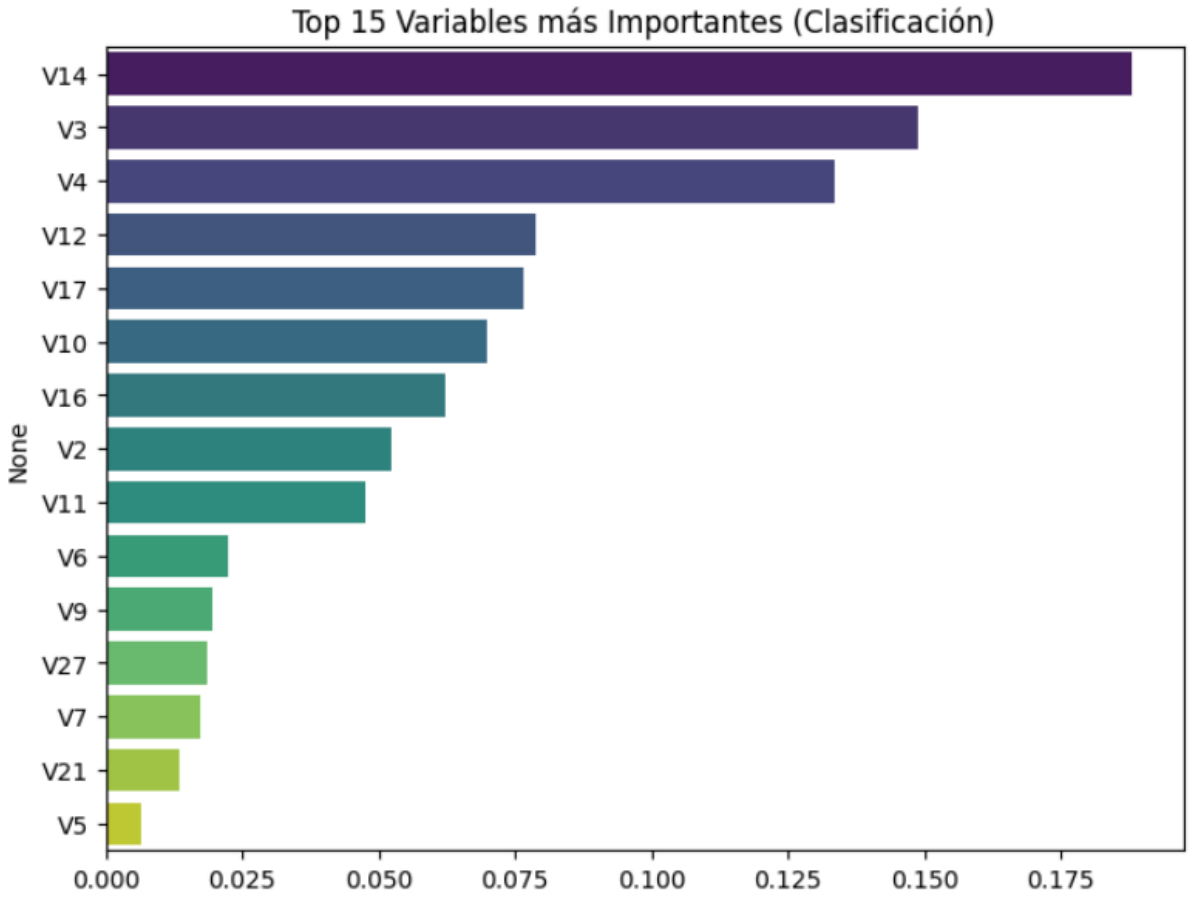
Comportamiento de la curva: Mantiene una precisión muy alta (cercana al 100%) hasta un recall de aproximadamente 0.5, luego decrece gradualmente.

Interpretación práctica: El modelo puede detectar hasta el 50% de los fraudes manteniendo una precisión casi perfecta. Para detectar más fraudes (mayor recall), se debe aceptar una reducción en la precisión.

PR-AUC vs ROC-AUC: El PR-AUC (90.05%) es menor que el ROC-AUC (97.99%), lo cual es típico en datasets desbalanceados. La curva PR es más conservadora y realista para este tipo de problemas.

Figura 9

Variables más importantes



Nota. Top 15 Variables más importantes (clasificación) del Random Forest.

Variables más determinantes para la clasificación:

V14 (~0.175): La variable más importante, sugiere que esta componente PCA captura patrones críticos de fraude

V3 (~0.150): Segunda más relevante, indica otro patrón distintivo

V4 (~0.125): Tercera en importancia

V12, V17, V10: Forman un grupo secundario de variables relevantes

Observaciones importantes:

Las variables V14, V3 y V4 dominan claramente la predicción

Time y Amount no aparecen en el top 15, sugiriendo que los patrones temporales y montos no son tan discriminatorios como los componentes PCA

La concentración de importancia en pocas variables (V14, V3, V4) sugiere que existen patrones específicos muy distintivos en las transacciones fraudulentas

Random Forest - Regresión

Análisis de las Métricas de Regresión

MSE (Error Cuadrático Medio) = 0.0008

- Valor muy bajo que indica que las predicciones están, en promedio, muy cerca de los valores reales
- Al ser el MSE la media de los errores al cuadrado, este valor sugiere errores de predicción muy pequeños

MAE (Error Absoluto Medio) = 0.0016

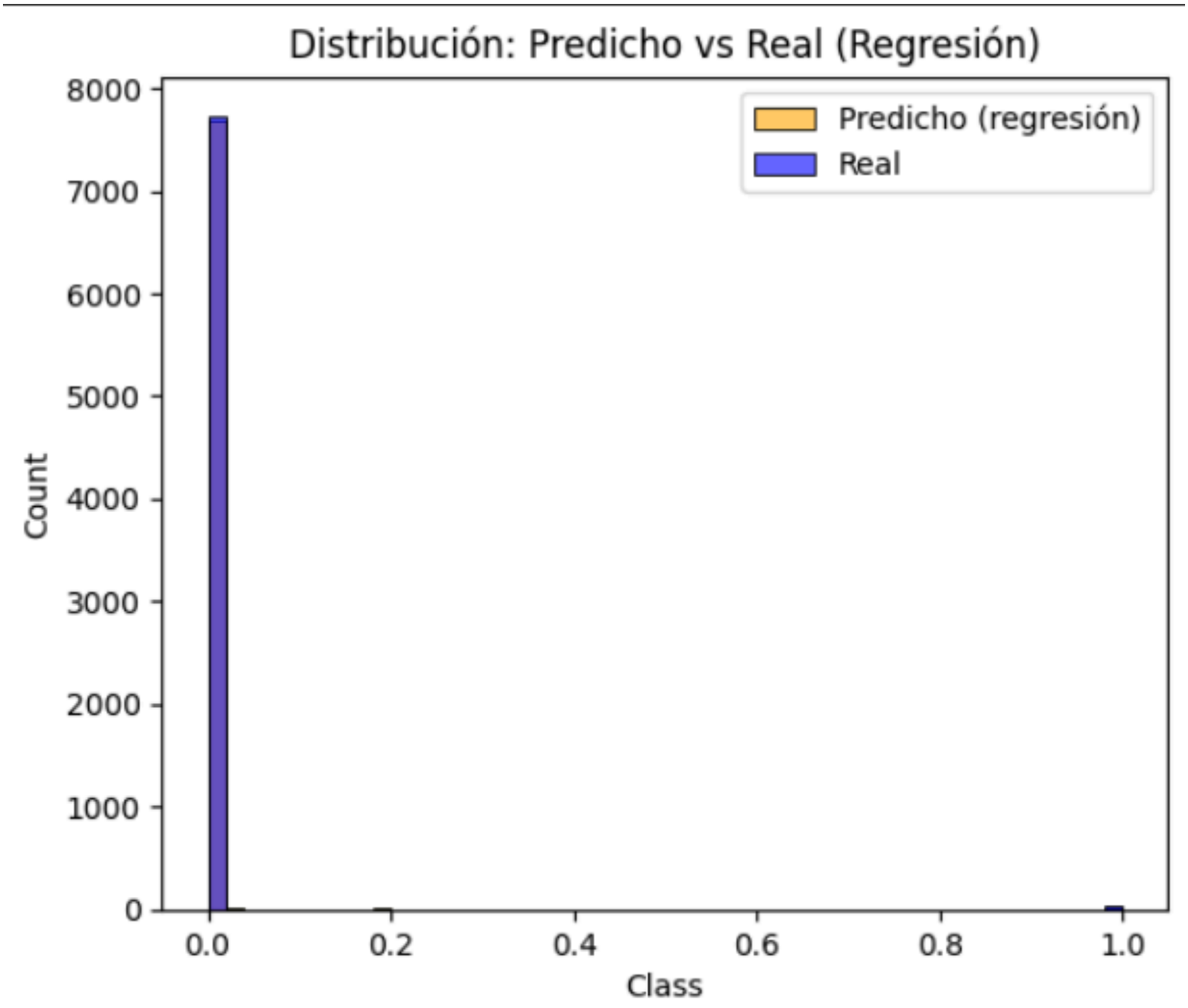
- El error absoluto promedio es de aproximadamente 0.0016 unidades
- Esto significa que, en promedio, las predicciones se desvían solo 0.16% de los valores reales
- Es consistente con el MSE bajo, confirmando la precisión del modelo

R^2 (Coeficiente de Determinación) = 0.7598 (75.98%)

- El modelo explica el 75.98% de la variabilidad en la variable objetivo
- Indica un rendimiento bueno pero no excelente para regresión
- Queda un 24.02% de variabilidad no explicada por el modelo

Figura 10

Gráfico de Distribución

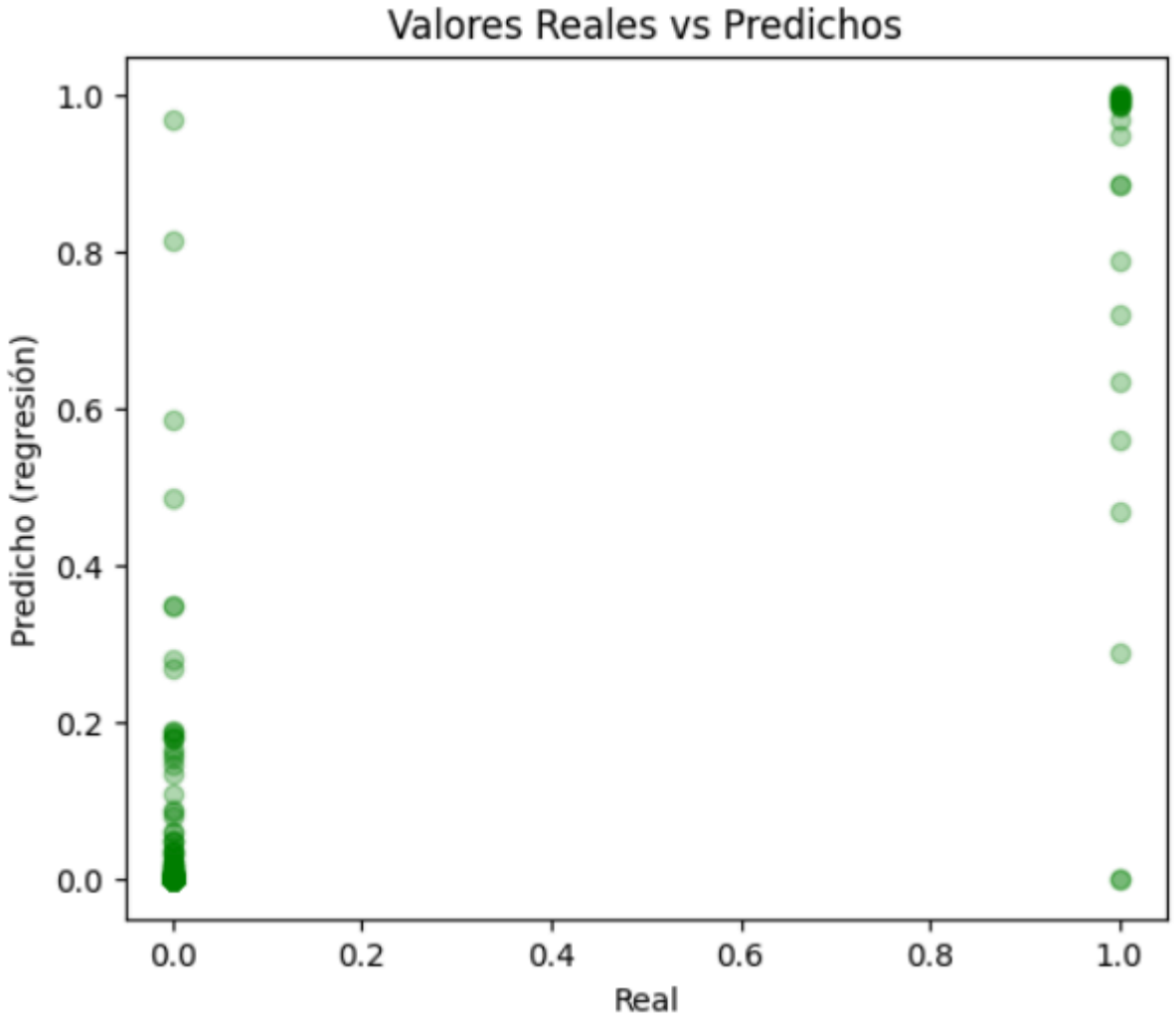


Nota. Distribución Predicho vs Real (Regresión).

Análisis crítico del histograma:

- Problema evidente: Ambas distribuciones (real y predicha) están extremadamente concentradas cerca de 0
- La distribución real (azul) y predicha (naranja) son prácticamente idénticas y se superponen completamente
- Interpretación: El modelo está prediciendo principalmente valores cercanos a 0, lo cual es problemático para regresión

Figura 11
Gráfico de valores reales vs predichos



Nota. Valores reales vs predichos.

Análisis del gráfico de dispersión:

- Concentración masiva en (0,0): La gran mayoría de puntos se acumulan en el origen
- Puntos dispersos: Se observan algunos puntos dispersos a lo largo de x=1 y y=1
- Ausencia de correlación lineal clara: No se observa la línea diagonal esperada en una buena regresión

Conclusiones

Hallazgos principales

- Los Árboles de Decisión son muy interpretables y útiles para explicar reglas de decisión (p. ej. qué variables influyen en Titanic o Iris), pero sin poda o regulación tienden al sobreajuste.
- Random Forest mejora la estabilidad y la generalización respecto a un árbol aislado, especialmente en datasets ruidosas o con muchas variables (por eso se usa en Titanic).
- En Iris ambos enfoques suelen alcanzar buen desempeño; en creditcard la clave no es tanto el modelo sino cómo se maneja el desbalance y las métricas elegidas.

Importancia de elegir métricas correctas

- Para Iris (balanceado): reportar accuracy y métricas por clase (precision/recall/F1).
- Para Titanic: además de accuracy, usar AUC y matriz de confusión para entender errores Tipo I/II.
- Para CreditCard (desbalanceado): priorizar recall, precision, F1 para la clase fraude y PR-AUC; la accuracy puede ser irrelevante.

Limitaciones y mejoras propuestas

- Limitaciones encontradas: dependencia del preprocesamiento (imputaciones, codificaciones), riesgo de sobreajuste en árboles, sensibilidad al desbalance en fraude, y posible falta de calibración de probabilidades.
- Mejoras propuestas: búsqueda de hiperparámetros con validación cruzada estratificada (Grid/RandomizedSearchCV), pipelines reproducibles (ColumnTransformer + Pipeline), probar re-sampling (SMOTE/undersampling) o class_weight='balanced' para fraude, y evaluar modelos de boosting (XGBoost/LightGBM) si se busca mayor rendimiento.

Consejos y Recursos

- Bibliotecas recomendadas: scikit-learn (modelos, métricas, pipelines), imbalanced-learn (SMOTE, re-sampling), graphviz o sklearn.tree.plot_tree (visualización de árboles), shap o lime (explicabilidad si necesitas justificar decisiones).
- Visualización útil: visualiza al menos un árbol representativo (plot_tree o graphviz) y un gráfico de feature importances; para fraude incluye curva Precision-Recall.
- Buenas prácticas: fijar random_state para reproducibilidad; usar stratify=y en train_test_split cuando hay desbalance; construir pipelines para evitar fugas de datos; registrar experimentos (métricas y parámetros).
- Métricas a calcular por dataset:
    - Iris: accuracy, precision/recall/F1 por clase, matriz de confusión.
    - Titanic: accuracy, AUC, precision/recall/F1, matriz de confusión.
    - Creditcard: precision, recall, F1 (para clase fraude), PR-AUC, ROC-AUC; además evaluar con validación estratificada.

Link a la Infografía
https://rerotz.github.io/

Referencias

[1] L. Breiman, J. Friedman, R. Olshen, and C. Stone, Classification and Regression Trees. Belmont, CA: Wadsworth International Group, 1984.

[2] L. Breiman, “Random Forests,” Machine Learning, vol. 45, no. 1, pp. 5–32, 2001.

[3] R. Kohavi and D. Wolpert, “Bias plus variance decomposition for zero-one loss functions,” in Proc. 13th Int. Conf. Machine Learning (ICML), 1996, pp. 275–283.

[4] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[5] Kaggle, “Titanic - Machine Learning from Disaster,” [Online]. Available: https://www.kaggle.com/c/titanic. [Accessed: Sep. 8, 2025].

[6] R. A. Fisher, “The use of multiple measurements in taxonomic problems,” Annals of Eugenics, vol. 7, no. 2, pp. 179–188, 1936.

[7] Kaggle, “Credit Card Fraud Detection,” [Online]. Available: https://www.kaggle.com/mlg-ulb/creditcardfraud. [Accessed: Sep. 8, 2025].

[8] N. Japkowicz and M. Shah, Evaluating Learning Algorithms: A Classification Perspective. Cambridge: Cambridge University Press, 2011.

[9] H. He and E. A. Garcia, “Learning from Imbalanced Data,” IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 9, pp. 1263–1284, Sep. 2009.

[10] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, “Gene Selection for Cancer Classification using Support Vector Machines,” Machine Learning, vol. 46, no. 1, pp. 389–422, 2002.