Lab_S10_BankMarketing_D_Escriba

June 1, 2025

1 LABORATORIO: Análisis Comparativo de Árboles de Decisión y Métodos de Ensamblado en Campañas de Marketing Bancario

2 Introducción al caso

En el contexto actual de digitalización financiera, los bancos buscan optimizar sus campañas de marketing directo para aumentar la tasa de conversión de sus productos. Este laboratorio simula el rol de un analista de datos en una institución financiera que busca predecir si un cliente contratará un depósito a plazo fijo a partir de sus características personales, historial financiero y comportamiento previo con el banco.

Se utilizará el dataset Bank Marketing, proveniente de campañas telefónicas reales realizadas en Portugal, y se pondrá en práctica una comparación entre modelos de árboles de decisión simples ydiferentes métodos de ensamblado.

3 Objetivos del laboratorio

• Preprocesar adecuadamente un conjunto de datos mixto (numérico y categórico). • Entrenar y comparar modelos de clasificación basados en árboles de decisión y ensamblado. • Aplicar técnicas de evaluación apropiadas para contextos de desbalance de clases. • Interpretar los resultados obtenidos desde un enfoque técnico y comercial. • Reflexionar sobre la aplicabilidad, limitaciones y ética del uso de modelos automatizados en decisiones comerciales.

Preparacion de Datos

3.1 Bank Marketing

The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

[ ]:
```
# ============================
# 1. IMPORTACIÓN DE LIBRERÍAS
# ============================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from ucimlrepo import fetch_ucirepo
from sklearn.model_selection import train_test_split, StratifiedKFold,␣
 ↪cross_val_score, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import confusion_matrix, classification_report,␣
 ↪accuracy_score, recall_score, f1_score, roc_auc_score
from sklearn.preprocessing import StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier,␣
 ↪GradientBoostingClassifier, AdaBoostClassifier, ExtraTreesClassifier
from sklearn.impute import SimpleImputer

import warnings
warnings.filterwarnings('ignore')
```

```
Requirement already satisfied: ucimlrepo in /usr/local/lib/python3.11/dist-
packages (0.0.7)
Requirement already satisfied: pandas>=1.0.0 in /usr/local/lib/python3.11/dist-
packages (from ucimlrepo) (2.2.2)
Requirement already satisfied: certifi>=2020.12.5 in
/usr/local/lib/python3.11/dist-packages (from ucimlrepo) (2025.4.26)
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-
packages (from pandas>=1.0.0->ucimlrepo) (2.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.11/dist-packages (from pandas>=1.0.0->ucimlrepo)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-
packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
packages (from pandas>=1.0.0->ucimlrepo) (2025.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-
packages (from python-dateutil>=2.8.2->pandas>=1.0.0->ucimlrepo) (1.17.0)
```

[ ]:
```
# ========================================
# 2. CARGA DE DATOS
# ========================================

# Cargar dataset desde ucirepo
# fetch dataset
bank_marketing = fetch_ucirepo(id=222)

# data (as pandas dataframes)
X = bank_marketing.data.features
y = bank_marketing.data.targets
```

```
df = pd.concat([X, y], axis=1)

# Ver distribución de clases
print("\nDistribución de clases (no = no se suscribe, yes = si se suscribe):")
print(df['y'].value_counts())
print("\n")
print(df['y'].value_counts(normalize=True))
print("\n\n")


# ============================
# 4. PREPROCESAMIENTO
# ============================

# Guardamos "y" como objetivo
y = df['y'].map({'no': 0, 'yes': 1})

# Eliminamos y del DataFrame para procesar solo características
X = df.drop(columns=['y'])

#En base al laboratorio anterior, Eliminamos por irrelevancia o sesgo␣
 ↪predictivo:
#day_of_week: poco predictivo
#duration: muy predictiva pero solo se conoce después de llamar , por lo tanto,␣
 ↪para un modelo realista, debe eliminarse

X = X.drop(columns=['day_of_week', 'duration'], errors='ignore')

# Identificamos las columnas categóricas
categorical_columns = X.select_dtypes(include=['object']).columns.tolist()

# Primero, vamos a manejar los valores NaN en columnas categóricas
# Podemos reemplazarlos con 'unknown' o la moda
for col in categorical_columns:
    X[col] = X[col].fillna('unknown')

# Aplicamos Label Encoding a todas las columnas categóricas
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    X[column] = le.fit_transform(X[column])
    label_encoders[column] = le
    print(f"Mapeo para {column}: {dict(zip(le.classes_, le.transform(le.
 ↪classes_)))}")

# Manejamos los valores NaN en columnas numéricas
numeric_columns = X.select_dtypes(include=['number']).columns.tolist()
```

```
imputer = SimpleImputer(strategy='median')
X[numeric_columns] = imputer.fit_transform(X[numeric_columns])

# Detectar automáticamente las categóricas (ya excluimos 'y')
cat_vars = X.select_dtypes(include=['object']).columns.tolist()

# Definir el preprocesador
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore', sparse_output=False),␣
 ↪cat_vars),
        # Puedes agregar escalado si lo necesitas más adelante
    ],
    remainder='passthrough'  # deja pasar otras columnas sin cambios
)

# Aplicar transformación
X_encoded = preprocessor.fit_transform(X)

# Estandarizar los datos: necesario porque las variables tienen diferentes␣
 ↪escalas
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# División entrenamiento-prueba con estratificación (mantiene proporción de␣
 ↪clases)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42
)
```

```
Distribución de clases (no = no se suscribe, yes = si se suscribe):
y
no     39922
yes     5289
Name: count, dtype: int64


y
no     0.883015
yes    0.116985
Name: proportion, dtype: float64


Mapeo para job: {'admin.': np.int64(0), 'blue-collar': np.int64(1),
'entrepreneur': np.int64(2), 'housemaid': np.int64(3), 'management':
```

```
np.int64(4), 'retired': np.int64(5), 'self-employed': np.int64(6), 'services':
np.int64(7), 'student': np.int64(8), 'technician': np.int64(9), 'unemployed':
np.int64(10), 'unknown': np.int64(11)}
Mapeo para marital: {'divorced': np.int64(0), 'married': np.int64(1), 'single':
np.int64(2)}
Mapeo para education: {'primary': np.int64(0), 'secondary': np.int64(1),
'tertiary': np.int64(2), 'unknown': np.int64(3)}
Mapeo para default: {'no': np.int64(0), 'yes': np.int64(1)}
Mapeo para housing: {'no': np.int64(0), 'yes': np.int64(1)}
Mapeo para loan: {'no': np.int64(0), 'yes': np.int64(1)}
Mapeo para contact: {'cellular': np.int64(0), 'telephone': np.int64(1),
'unknown': np.int64(2)}
Mapeo para month: {'apr': np.int64(0), 'aug': np.int64(1), 'dec': np.int64(2),
'feb': np.int64(3), 'jan': np.int64(4), 'jul': np.int64(5), 'jun': np.int64(6),
'mar': np.int64(7), 'may': np.int64(8), 'nov': np.int64(9), 'oct': np.int64(10),
'sep': np.int64(11)}
Mapeo para poutcome: {'failure': np.int64(0), 'other': np.int64(1), 'success':
np.int64(2), 'unknown': np.int64(3)}
```

---

4 Preguntas para desarrollo

4.1 Árboles de Decisión

4.1.1 ¿Qué criterio de división (gini, entropy) ofrece mejor precisión, recall o F1-score en este caso?

[ ]:
```python
# ========================================
#   COMPARACIÓN DE CRITERIOS DE DIVISIÓN
# ========================================

criterios = ['gini', 'entropy']
modelos = {}

print("\n COMPARACIÓN DE CRITERIOS DE DIVISIÓN")
print("="*50)

for criterio in criterios:
    modelo = DecisionTreeClassifier(
        criterion=criterio,
        max_depth=8,
        min_samples_split=10,
        min_samples_leaf=5,
        random_state=42,
        class_weight='balanced'
    )
```

```
    modelo.fit(X_train, y_train)
    y_pred = modelo.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred, average='macro')

    modelos[criterio] = modelo

    print(f" Criterio {criterio.upper()}:")
    print(f"   Precisión: {accuracy:.4f}")
    print(f"   Recall: {recall:.4f}")
    print(classification_report(y_test, y_pred, target_names=["no", "yes"]))
    print()
```

```
 COMPARACIÓN DE CRITERIOS DE DIVISIÓN
==================================================
 Criterio GINI:
  Precisión: 0.8583
  Recall: 0.6923
              precision    recall  f1-score   support

          no       0.93      0.91      0.92      7985
         yes       0.41      0.48      0.44      1058

    accuracy                           0.86      9043
   macro avg       0.67      0.69      0.68      9043
weighted avg       0.87      0.86      0.86      9043


 Criterio ENTROPY:
  Precisión: 0.8628
  Recall: 0.6882
              precision    recall  f1-score   support

          no       0.93      0.92      0.92      7985
         yes       0.42      0.46      0.44      1058

    accuracy                           0.86      9043
   macro avg       0.67      0.69      0.68      9043
weighted avg       0.87      0.86      0.87      9043
```

Respuesta Precisión: - ENTROPY ofrece una mejor precisión general (0.8628) en comparación con GINI (0.8583). Aunque la diferencia es pequeña, ENTROPY tiene una ligera ventaja en la precisión general del modelo.

Recall: - GINI tiene un mejor recall general (0.6923) en comparación con ENTROPY (0.6882).

Sin embargo, en detalle, para la clase “yes” (suscripciones logradas), ENTROPY tiene un recall de 0.46, mientras que GINI tiene un recall de 0.48. Esto indica que ambos criterios tienen dificultades para detectar bien la clase “yes”, aunque GINI es ligeramente mejor en este aspecto.

F1-score: - El F1-score ponderado es muy similar entre ambos criterios, con ENTROPY ligeramente mejor (0.87) en comparación con GINI (0.86). Esto confirma que el equilibrio entre precisión y recall es similar en ambos casos.

Conclusión:

Si bien ambos criterios de división (GINI y ENTROPY) ofrecen resultados muy similares, ENTROPY tiene una ligera ventaja en términos de precisión general y F1-score ponderado. Sin embargo, GINI tiene un mejor recall general, aunque la diferencia es mínima.

En general, ninguno de los criterios alcanza un estado óptimo, especialmente en la detección de la clase “yes”. Por lo tanto, aunque ENTROPY tiene una ligera ventaja en precisión y F1-score, ambos criterios necesitan mejorar para alcanzar una optimización mejor en la detección de ambas clases.

4.1.2 Analiza el árbol obtenido: ¿qué variables aparecen como más importantes? ¿Son coherentes con el contexto del marketing bancario?

[ ]:
```
for criterio in criterios:
    modelo = modelos[criterio]

    # Obtener importancias de las variables del modelo entrenado
    importancias = modelo.feature_importances_

    # Crear un DataFrame con nombres de las variables e importancias
    importancias_df = pd.DataFrame({
        'Variable': X.columns,
        'Importancia': importancias
    }).sort_values(by='Importancia', ascending=False)

    # Mostrar las 5 variables más importantes
    print(f"\n Variables más importantes según el árbol {criterio.upper()}:")
    print(importancias_df.head(5))
```

```
 Variables más importantes según el árbol GINI:
   Variable  Importancia
8   contact     0.272957
9     month     0.201860
11    pdays     0.144226
6   housing     0.113830
0       age     0.086976

 Variables más importantes según el árbol ENTROPY:
```

```
   Variable  Importancia
8   contact     0.249110
9     month     0.195391
11    pdays     0.151103
6   housing     0.100921
0       age     0.083624
```

Respuesta Las variables más importantes según los árboles GINI y ENTROPY son consistentes con el contexto del marketing bancario. Estas variables reflejan aspectos clave que pueden influir en la respuesta de los clientes a una campaña de marketing:

- Tipo de contacto: Importante para la efectividad de la comunicación.
- Mes de la campaña: Relevante para aprovechar tendencias estacionales.
- Días desde el último contacto: Indica la recurrencia y la reciente interacción.
- Préstamo hipotecario: Refleja la carga financiera y la estabilidad del cliente.
- Edad: Relacionada con las necesidades y preferencias financieras según la etapa de vida.

Estos resultados son coherentes con las prácticas comunes en el marketing bancario, donde la comunicación efectiva, la situación financiera del cliente y la etapa de vida son factores cruciales para el éxito de las campañas.

4.1.3 ¿Qué profundidad y número de nodos terminales presenta cada árbol? ¿Cómo se relaciona esto con el sobreajuste?

[ ]:
```python
for criterio in criterios:
  modelos[criterio] = modelo
  print(f" Criterio {criterio.upper()}:")
  print(f"   Profundidad del árbol: {modelo.get_depth()}")
  print(f"   Número de hojas: {modelo.get_n_leaves()}")
  print()
```

```
 Criterio GINI:
   Profundidad del árbol: 8
   Número de hojas: 143

 Criterio ENTROPY:
   Profundidad del árbol: 8
   Número de hojas: 143
```

Respuesta Se encontro la profundidad y numero de hojas.

Asimismo, ambos árboles tienen la misma profundidad y número de hojas. Una profundidad de 8 y 143 hojas son valores moderados que no sugieren sobreajuste. Si los árboles fueran más profundos o tuvieran más hojas, podrían capturar ruido en los datos y sobreajustarse. En este caso, los valores actuales parecen equilibrados.

4.2 Optimización del Árbol

###Realiza una búsqueda de hiperparámetros con GridSearchCV. ¿Qué combinación ofrece el mejor rendimiento?

[ ]:
```
# ========================================
#   OPTIMIZACIÓN DE HIPERPARÁMETROS
# ========================================

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [3, 5, 7, 10, None],
    'min_samples_split': [2, 5, 10, 20],
    'min_samples_leaf': [1, 2, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'class_weight': ['balanced']
}

grid_search = GridSearchCV(
    estimator=DecisionTreeClassifier(random_state=42),
    param_grid=param_grid,
    cv=5,
    scoring='accuracy',
    n_jobs=-1
)

grid_search.fit(X_train, y_train)
mejor_modelo = grid_search.best_estimator_
y_pred_opt = mejor_modelo.predict(X_test)
acc_opt = accuracy_score(y_test, y_pred_opt)
recall_opt = recall_score(y_test, y_pred_opt, average='macro')  # Macro para␣
 ↪evaluar todas las clases por igual
f1_opt = f1_score(y_test, y_pred_opt, average='macro')

print(" Resultados de optimización:")
print("\n Mejor combinación:")
for clave, valor in grid_search.best_params_.items():
    print(f"   {clave}: {valor}")

print(f"\n Métricas del modelo optimizado:")
print(f"   Precisión (Accuracy): {acc_opt:.4f}")
print(f"   Recall: {recall_opt:.4f}")
print(f"   F1-Score: {f1_opt:.4f}")
print(f" Precisión media CV: {grid_search.best_score_:.4f}")
```

```
Resultados de optimización:

Mejor combinación:
```

```
  class_weight: balanced
  criterion: gini
  max_depth: None
  max_features: sqrt
  min_samples_leaf: 1
  min_samples_split: 2

Métricas del modelo optimizado:
  Precisión (Accuracy): 0.8389
  Recall: 0.5943
  F1-Score: 0.5973
Precisión media CV: 0.8337
```

4.2.1 ¿La precisión del árbol mejoró significativamente tras la optimización? Justifica si vale la pena el aumento de complejidad

La precisión del árbol disminuyó ligeramente tras la optimización, pasando de 0.8583 a 0.8389. Aunque el modelo optimizado muestra una precisión media en validación cruzada de 0.8337, lo que indica una generalización mejor, el recall y el F1-score también disminuyeron. Dado que la complejidad del modelo no aumentó significativamente, pero el rendimiento general se mantuvo similar o incluso disminuyó, no parece que valga la pena el aumento de complejidad. En este caso, sería recomendable explorar otras opciones de hiperparámetros o incluso probar otros algoritmos de modelado para encontrar una solución más efectiva.

4.3 Métodos de Ensamblado

4.3.1 Compara los siguientes modelos en cuanto a precisión, recall, F1-score y AUC-ROC:

- Random Forest
- Gradient Boosting
- AdaBoost
- Extra Trees

[ ]:
```python
# ================================
# RANDOM FOREST
# ================================

print("\n RANDOM FOREST")
rf = RandomForestClassifier(
    n_estimators=200, max_depth=10, min_samples_split=5, min_samples_leaf=2,
    max_features='sqrt', bootstrap=True, n_jobs=-1, random_state=42
)
rf.fit(X_train, y_train)  # Entrenar modelo
y_pred_rf = rf.predict(X_test)  # Predecir
acc_rf = accuracy_score(y_test, y_pred_rf)  # Calcular precisión
```

```
auc_rf = roc_auc_score(y_test, rf.predict_proba(X_test)[:, 1])  # Calcular AUC
print(f"Precisión: {acc_rf:.4f}")
print(f"AUC-ROC: {auc_rf:.4f}")

# ================================
# GRADIENT BOOSTING
# ================================

print("\n GRADIENT BOOSTING")
gb = GradientBoostingClassifier(
    n_estimators=150, learning_rate=0.1, max_depth=4,
    min_samples_split=10, min_samples_leaf=5,
    subsample=0.8, random_state=42
)
gb.fit(X_train, y_train)
y_pred_gb = gb.predict(X_test)
acc_gb = accuracy_score(y_test, y_pred_gb)
auc_gb = roc_auc_score(y_test, gb.predict_proba(X_test)[:, 1])
print(f"Precisión: {acc_gb:.4f}")
print(f"AUC-ROC: {auc_gb:.4f}")

# ================================
# ADABOOST
# ================================

print("\n ADABOOST")
ada = AdaBoostClassifier(
    estimator=DecisionTreeClassifier(max_depth=1),
    n_estimators=100, learning_rate=1.0,
    # Cambiar 'SAMME.R' a 'SAMME' ya que 'SAMME.R' no es un valor válido en␣
 ↪esta versión de scikit-learn
    algorithm='SAMME',
    random_state=42
)
ada.fit(X_train, y_train)
y_pred_ada = ada.predict(X_test)
acc_ada = accuracy_score(y_test, y_pred_ada)
auc_ada = roc_auc_score(y_test, ada.predict_proba(X_test)[:, 1])
print(f"Precisión: {acc_ada:.4f}")
print(f"AUC-ROC: {auc_ada:.4f}")

# ================================
# EXTRA TREES
# ================================

print("\n EXTRA TREES")
et = ExtraTreesClassifier(
```

```
    n_estimators=200, max_depth=None, min_samples_split=5, min_samples_leaf=2,
    max_features='sqrt', bootstrap=False, n_jobs=-1, random_state=42
)
et.fit(X_train, y_train)
y_pred_et = et.predict(X_test)
acc_et = accuracy_score(y_test, y_pred_et)
auc_et = roc_auc_score(y_test, et.predict_proba(X_test)[:, 1])
print(f"Precisión: {acc_et:.4f}")
print(f"AUC-ROC: {auc_et:.4f}")

# ================================
# COMPARACIÓN DE TODOS LOS MODELOS
# ================================

# Crear diccionario de resultados
resultados = {
    'Random Forest': {'Precisión': acc_rf, 'AUC-ROC': auc_rf},
    'Gradient Boosting': {'Precisión': acc_gb, 'AUC-ROC': auc_gb},
    'AdaBoost': {'Precisión': acc_ada, 'AUC-ROC': auc_ada},
    'Extra Trees': {'Precisión': acc_et, 'AUC-ROC': auc_et}
}

# Mostrar resultados como DataFrame
resultados_df = pd.DataFrame(resultados).T
print("\n COMPARACIÓN FINAL DE TODOS LOS MÉTODOS:")
print(resultados_df.round(4))
```

```
 RANDOM FOREST
Precisión: 0.8938
AUC-ROC: 0.7880

 GRADIENT BOOSTING
Precisión: 0.8932
AUC-ROC: 0.7949

 ADABOOST
Precisión: 0.8909
AUC-ROC: 0.7744

 EXTRA TREES
Precisión: 0.8941
AUC-ROC: 0.7842

 COMPARACIÓN FINAL DE TODOS LOS MÉTODOS:
                  Precisión  AUC-ROC
Random Forest        0.8938   0.7880
Gradient Boosting    0.8932   0.7949
```

```
AdaBoost              0.8909   0.7744
Extra Trees           0.8941   0.7842
```

4.3.2 ¿Qué modelo sería más adecuado para una campaña bancaria real que busca minimizar falsos negativos? Justifica tu elección.

Dado que la campaña bancaria busca minimizar falsos negativos, es crucial elegir un modelo que tenga un buen recall, ya que los falsos negativos se relacionan directamente con la capacidad de detectar correctamente las suscripciones positivas. En este caso, aunque los modelos tienen precisiones similares, el Gradient Boosting tiene el mejor AUC-ROC (0.7949), lo que indica una mejor capacidad para distinguir entre las clases. Además, un buen AUC-ROC sugiere que el modelo puede manejar mejor el trade-off entre recall y precisión. Por lo tanto, el Gradient Boosting sería el más adecuado para esta campaña, ya que puede ofrecer un mejor equilibrio entre minimizar falsos negativos y mantener un buen rendimiento general.

4.4 Interpretación de variables

4.4.1 Extrae las variables más importantes de al menos dos modelos ensamblados. ¿Qué patrones o perfiles de cliente parecen más propensos a contratar el depósito?

[ ]:
```python
# Primero optemos las varibales de Gradient Boosting
importancia_gb = pd.DataFrame({
    'Variable': X.columns,
    'Importancia': gb.feature_importances_
}).sort_values(by='Importancia', ascending=False).head(10)

# Luego , obtengo la importancia de variables de Random forest
importancia_ada = pd.DataFrame({
    'Variable': X.columns,  # Nombre de las variables (columnas del dataset)
    'Importancia': rf.feature_importances_  # Importancia calculada por el␣
 ↪modelo
}).sort_values(by='Importancia', ascending=False).head(10)  # Ordeno de mayor a␣
 ↪menor y me quedo con las 10 principales

# Finalmente, imprimo los resultados de forma textual (sin gráficos) para poder␣
 ↪analizarlos con facilidad
print("\n  Top 10 variables más importantes - Gradient Boosting")
print(importancia_ada.to_string(index=False))

print("\n  Top 10 variables más importantes - Random forest")
print(importancia_gb.to_string(index=False))
```

```
 Top 10 variables más importantes - Gradient Boosting
Variable  Importancia
```

```
poutcome     0.203162
   month     0.154752
     age     0.129830
   pdays     0.124817
 balance     0.086611
 housing     0.078984
previous     0.049374
 contact     0.048357
     job     0.036352
campaign     0.032749

  Top 10 variables más importantes - Random forest
Variable  Importancia
   month     0.213941
   pdays     0.186964
poutcome     0.166115
     age     0.140360
 balance     0.081331
 contact     0.062937
 housing     0.056797
previous     0.023031
campaign     0.020020
     job     0.015420
```

Repuesta Con nuestra respuesta anterior elegimos nuestra mejor opcion Gradient Boosting y Random forest

- poutcome: Ambos modelos coinciden en que el resultado de campañas anteriores es muy importante. Clientes que han respondido positivamente en el pasado son más propensos a contratar nuevos productos.
- month: El mes de la campaña es crucial, sugiriendo que ciertos meses pueden ser más propicios para las campañas de marketing.
- age: La edad es un factor importante, indicando que ciertas etapas de la vida pueden estar más dispuestas a contratar productos financieros.
- pdays: La recurrencia y la reciente interacción son relevantes. Clientes contactados recientemente pueden estar más dispuestos a considerar una oferta.
- balance: Un saldo anual más alto puede indicar una capacidad financiera mayor, lo que puede hacer que los clientes sean más propensos a contratar productos de ahorro.
- housing: Tener un préstamo hipotecario puede indicar una situación financiera estable, lo que puede hacer que los clientes sean más propensos a considerar productos adicionales.

Los clientes más propensos a contratar el depósito son aquellos que han respondido positivamente a campañas anteriores, han sido contactados recientemente, tienen una edad avanzada, y tienen una situación financiera estable (alto saldo y préstamos hipotecarios).

---

5 Interpretaciones finales

5.1 Para concluir el laboratorio, reflexiona y responde:

- ¿Qué fortalezas y debilidades identificas en los modelos basados en árboles para este tipo de tarea?
- ¿Qué modelo aplicarías si tuvieras que entregar una solución a un equipo de marketing no técnico? ¿Por qué?
- ¿Qué problemas éticos o de sesgo podrían surgir al usar estos modelos para tomar decisiones comerciales automatizadas?
- ¿Cómo cambiaría tu estrategia si el dataset estuviera aún más desbalanceado o si faltaran datos en variables clave?

Los modelos basados en árboles tienen fortalezas como su capacidad para manejar variables categóricas y numéricas sin normalización, y son fáciles de interpretar, lo que es útil para equipos no técnicos. Sin embargo, pueden sufrir de sobreajuste y ser menos robustos a datos ruidosos. Para un equipo de marketing no técnico, aplicaría un modelo de Random Forest debido a su equilibrio entre precisión y facilidad de interpretación, y porque puede manejar bien datos desbalanceados. Al usar estos modelos, podrían surgir problemas éticos como sesgos en la toma de decisiones si los datos de entrenamiento no son representativos de toda la población. Si el dataset estuviera más desbalanceado, consideraría técnicas de re-muestreo o ajuste de pesos de clase. Si faltaran datos en variables clave, exploraría imputación de datos o selección de características para reducir la dependencia de esas variables.
