

# Informe:

## Práctica 6: Implementación de una canalización de aprendizaje automático con Amazon SageMaker

Jordi Blasco Lozano  
Infraestructuras y Servicios Cloud  
Universidad de Alicante

13 de diciembre de 2025

### Resumen

En esta práctica hemos implementado un pipeline completo de Machine Learning con Amazon SageMaker para detectar anomalías en la columna vertebral de pacientes. Comenzamos explorando el dataset, luego entrenamos un modelo XGBoost, lo desplegamos y evaluamos con diferentes umbrales de clasificación. Finalmente, experimentamos con ajuste automático de hiperparámetros para intentar mejorar el rendimiento. A través de todos estos pasos vimos cómo funcionan los servicios de AWS para ML desde cero hasta tener un modelo en producción.

## Índice

<b>1. Exploración de Datos (Lab 3.2)</b>	<b>2</b>
1.1. El Dataset . . . . .	2
1.2. Análisis Exploratorio (EDA) . . . . .	2
1.3. Tareas de desafíos . . . . .	3
<b>2. Codificación de Datos Categóricos (Lab 3.3)</b>	<b>4</b>
2.1. Desafío: Añadir Más Variables . . . . .	5
<b>3. División de Datos y Entrenamiento (Lab 3.4)</b>	<b>5</b>
3.1. Preparación del Dataset . . . . .	5
3.2. División Estratificada . . . . .	6
3.3. Entrenamiento con SageMaker . . . . .	6
<b>4. Implementación y Predicciones (Lab 3.5)</b>	<b>6</b>
4.1. Deploy del Modelo . . . . .	6
4.2. Transformación por lotes . . . . .	6
4.3. Tarea desafío . . . . .	6
<b>5. Evaluación del Modelo (Lab 3.6)</b>	<b>7</b>
5.1. La Matriz de Confusión . . . . .	7
5.2. Desafío: Comparar Umbrales . . . . .	8
5.3. Curva ROC y AUC . . . . .	8

<b>6. Ajuste de Hiperparámetros (Lab 3.7)</b>	<b>9</b>
6.1. Métricas del modelo base	9
6.2. Configuración del Tuner	9
6.3. Ejecución y análisis	10
6.4. Resultados obtenidos	10

## 1 Exploración de Datos (Lab 3.2)

### 1.1 El Dataset

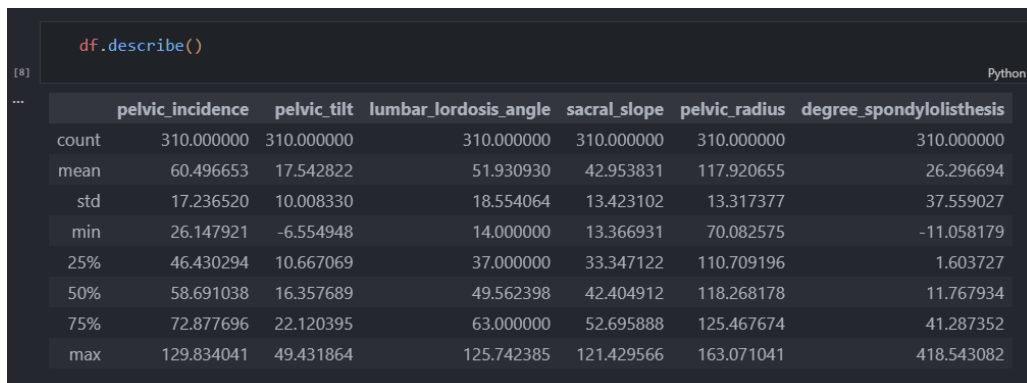
Para esta práctica utilizamos el dataset `column_2C_weka.arff` del repositorio UCI, que contiene información sobre pacientes con columna vertebral. Tiene 310 instancias (100 normales y 210 con anomalías) y 6 atributos biomecánicos:

- Incidencia pélvica
- Inclinación pélvica
- Ángulo de lordosis lumbar
- Inclinación del sacro
- Radio pélvico
- Grado de espondilolistesis

El objetivo era clasificar a los pacientes en dos categorías: **Normal** o **Abnormal**.

### 1.2 Análisis Exploratorio (EDA)

Lo primero que hicimos fue cargar el dataset y explorar su estructura usando `shape`, `describe()` y `info()`. Luego generamos gráficos de densidad (KDE), histogramas y box plots para entender cómo se distribuían los datos.



	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
count	310.000000	310.000000	310.000000	310.000000	310.000000	310.000000
mean	60.496653	17.542822	51.930930	42.953831	117.920655	26.296694
std	17.236520	10.008330	18.554064	13.423102	13.317377	37.559027
min	26.147921	-6.554948	14.000000	13.366931	70.082575	-11.058179
25%	46.430294	10.667069	37.000000	33.347122	110.709196	1.603727
50%	58.691038	16.357689	49.562398	42.404912	118.268178	11.767934
75%	72.877696	22.120395	63.000000	52.695888	125.467674	41.287352
max	129.834041	49.431864	125.742385	121.429566	163.071041	418.543082

Figura 1: Visualización inicial del conjunto de datos (distribuciones y resumen del EDA).

La característica `degree_spondylolisthesis` tenía valores atípicos bastante extremos (alrededor de 400), mientras que el resto de variables estaban más centradas.

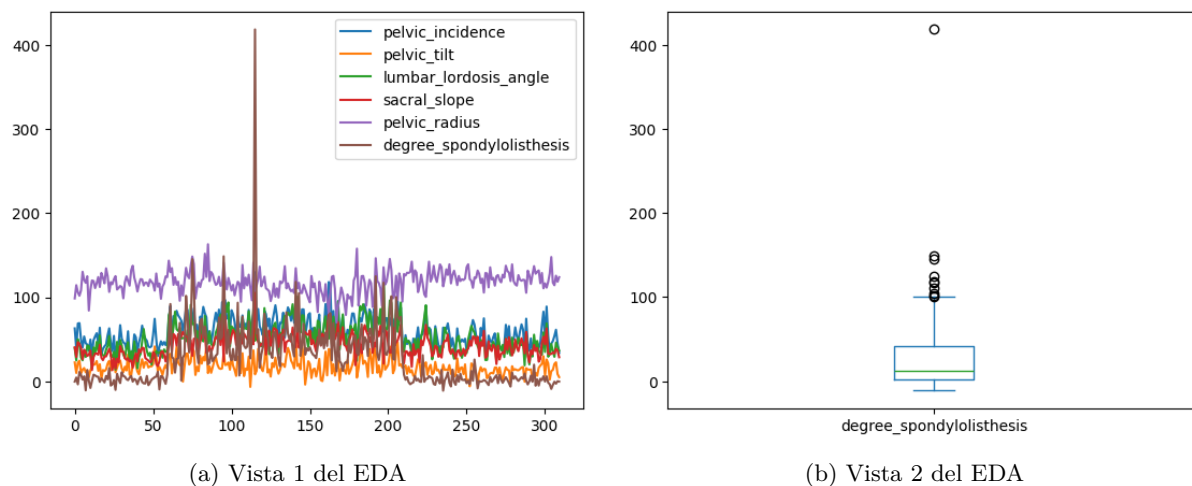


Figura 2: Gráficos comparativos para identificar distribución y valores atípicos.

También vimos que había un desbalanceo moderado: 2/3 Abnormal y 1/3 Normal. Esto es importante tenerlo en cuenta más adelante con las métricas.

```
df['class'].value_counts()

class
b'Abnormal'    210
b'Normal'      100
Name: count, dtype: int64
```

Figura 3: Distribución de clases (Normal vs. Abnormal) en el dataset.

### 1.3 Tareas de desafíos

Las tareas desafío de este lab tenían como objetivo entender como explorar manualmente todas las características e identificar sus valores atípicos mediante pandas. Para hacer la ultima tarea de desafío, para explorar sobre otros datos de UCI, simplemente me descargue el DataSet que estamos usando en nuestra asignatura de Aprendizaje Automático y le pase las mismas celdas de código. Mi data set era del zoo y tube estos resultados para el mapa de calor.

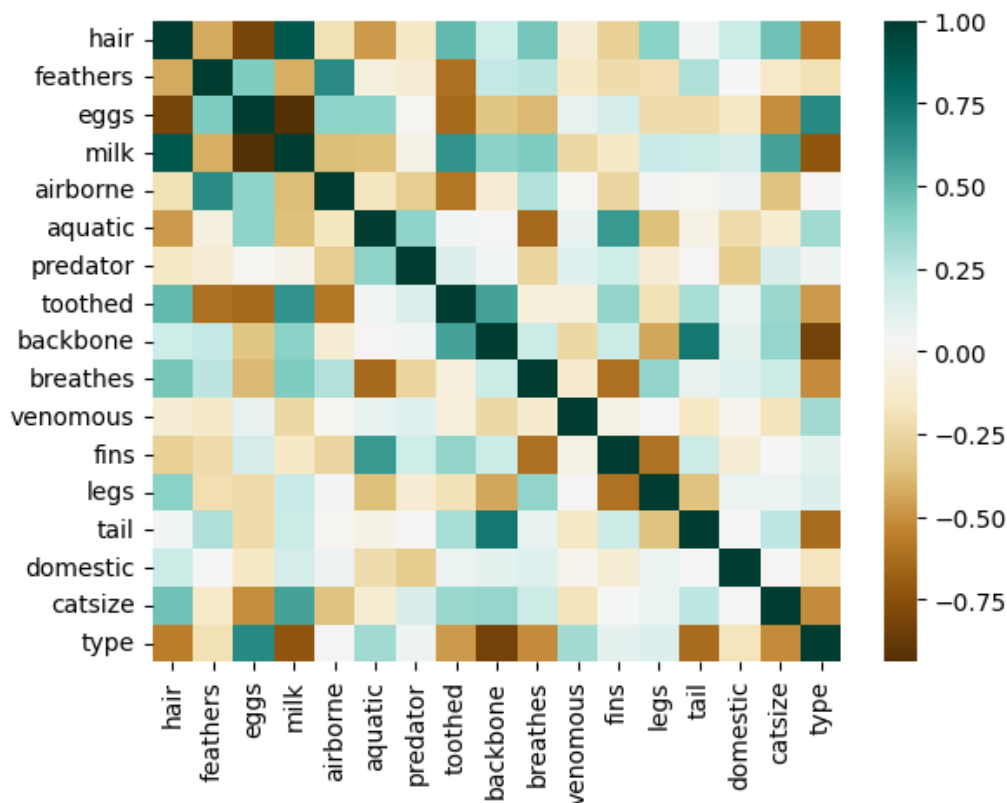


Figura 4: Mapa de calor (dataset alternativo) usado en la tarea de desafío.

## 2 Codificación de Datos Categóricos (Lab 3.3)

En este lab trabajamos con el dataset `imports-85.csv` que tiene información sobre automóviles. El archivo tiene 205 instancias, 25 atributos y no tenía encabezados, así que hay que definirlos manualmente. El objetivo será aprender a codificar variables categóricas de dos formas diferentes.

Para variables numerales o ordinales (con orden), usamos un diccionario que mapeaba valores a números:

	num-of-doors	doors	num-of-cylinders	cylinders
0	two	2.0	four	4
1	two	2.0	four	4
2	two	2.0	six	6
3	four	4.0	four	4
4	four	4.0	five	5

Figura 5: Ejemplo de codificación ordinal mediante un diccionario (mapeo).

Para variables nominales (sin orden natural), usamos `get_dummies()` que creará nuevas columnas binarias:

drive-wheels
rwd
rwd
rwd
fwd
4wd

(a) Antes de `get_dummies()`

drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd
False	False	True
False	False	True
False	False	True
False	True	False
True	False	False

(b) Después de `get_dummies()`

Figura 6: Conversión de variables nominales a variables binarias.

## 2.1 Desafío: Añadir Más Variables

El desafío consistía en codificar variables adicionales: `fuel-type`, `body-style` y `engine-location`. pasamos de esto:

	aspiration	drive-wheels	fuel-type	body-style	engine-location	num-of-doors	num-of-cylinders
0	std	rwd	gas	convertible	front	two	four
1	std	rwd	gas	convertible	front	two	four
2	std	rwd	gas	hatchback	front	two	six
3	std	fwd	gas	sedan	front	four	four
4	std	4wd	gas	sedan	front	four	five

Figura 7: Estado inicial antes de codificar variables adicionales (desafío).

a esto:

	doors	cylinders	aspiration_turbo	drive-wheels_fwd	drive-wheels_rwd	fuel-type_gas	body-style_hardtop	body-style_hatchback	body-style_sedan	body-style_wagon	engine-location_rear
0	2.0	4	False	False	True	True	False	False	False	False	False
1	2.0	4	False	False	True	True	False	False	False	False	False
2	2.0	6	False	False	True	True	False	True	False	False	False
3	4.0	4	False	True	False	True	False	False	True	False	False
4	4.0	5	False	False	False	True	False	False	True	False	False

Figura 8: Resultado tras añadir la codificación de variables adicionales.

## 3 División de Datos y Entrenamiento (Lab 3.4)

### 3.1 Preparación del Dataset

Lo primero fue convertir la clase a números: Abnormal = 1, Normal = 0. XGBoost en SageMaker tiene un requisito importante: la columna objetivo debe estar en la primera posición, así que se reordenaron

todas las columnas.

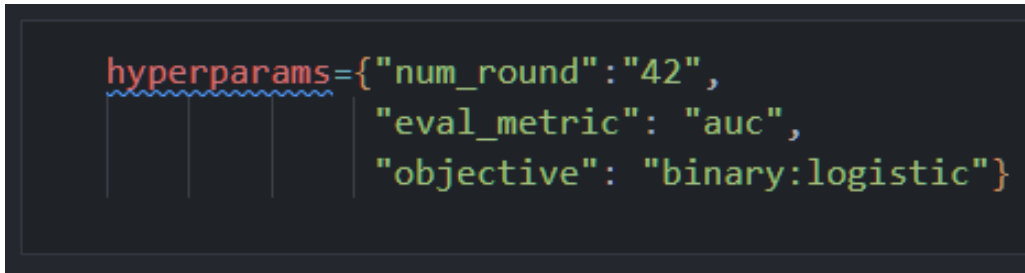
### 3.2 División Estratificada

Dividimos el dataset en tres partes usando `train_test_val` con `stratify` para mantener la proporción de clases: Usamos un 80 por ciento para entrenar, 10 para validación y 10 para test.

Usar `stratify` asegura que cada conjunto tenga la misma proporción de Normal/Abnormal.

### 3.3 Entrenamiento con SageMaker

Exportamos los CSV sin encabezados ni índices y los cargamos a S3. Luego configuramos XGBoost con los siguientes hiperparámetros:



```
hyperparams={"num_round": "42",  
             "eval_metric": "auc",  
             "objective": "binary:logistic"}
```

Figura 9: Hiperparámetros utilizados para el entrenamiento de XGBoost en SageMaker.

SageMaker se encargó de entrenar el modelo en la nube.

## 4 Implementación y Predicciones (Lab 3.5)

### 4.1 Deploy del Modelo

Una vez entrenado, desplegamos el modelo como un *endpoint* en tiempo real con SageMaker. Esto significa que el artefacto entrenado se levanta en una instancia (por ejemplo, `m1.m4.xlarge`) y queda expuesto como un servicio que acepta peticiones y devuelve predicciones al momento. Para inferencia, enviamos las características en formato CSV (sin la columna objetivo), y el modelo devuelve una **probabilidad** (por ejemplo, `0.9966`) de pertenecer a la clase *Abnormal*. A partir de esa probabilidad, aplicamos un **umbral** para convertirla a una decisión binaria (0/1), y vimos que variar el umbral cambia el equilibrio entre falsos positivos y falsos negativos.

### 4.2 Transformación por lotes

Sin embargo, hacer peticiones fila a fila no es eficiente cuando queremos evaluar un conjunto completo. Por eso, siguiendo el Paso 4 del laboratorio, usamos una **transformación por lotes (Batch Transform)**: subimos a S3 un CSV con todas las filas de test (solo features), lanzamos un *job* de transformación que procesa el fichero en bloque, y SageMaker escribe automáticamente las probabilidades de salida en otra ruta de S3. Finalmente descargamos ese resultado y aplicamos la misma conversión binaria por umbral para poder comparar predicción vs. etiqueta real y calcular métricas. Esta alternativa evita mantener un endpoint activo y suele ser más adecuada para evaluación offline.

### 4.3 Tarea desafío

En la tarea de desafío se nos pedía ir cambiando el valor del umbral de decisión. Al bajarlo, el modelo clasifica como *Abnormal* con más facilidad (aumenta la sensibilidad, pero también los falsos positivos). Al subirlo, se vuelve más conservador (reduce falsos positivos, pero aumenta el riesgo de falsos negativos). En las siguientes capturas se ve cómo cambia la salida al probar distintos valores. En nuestro caso tendríamos el mejor rendimiento en un umbral desde 78 hasta 93.

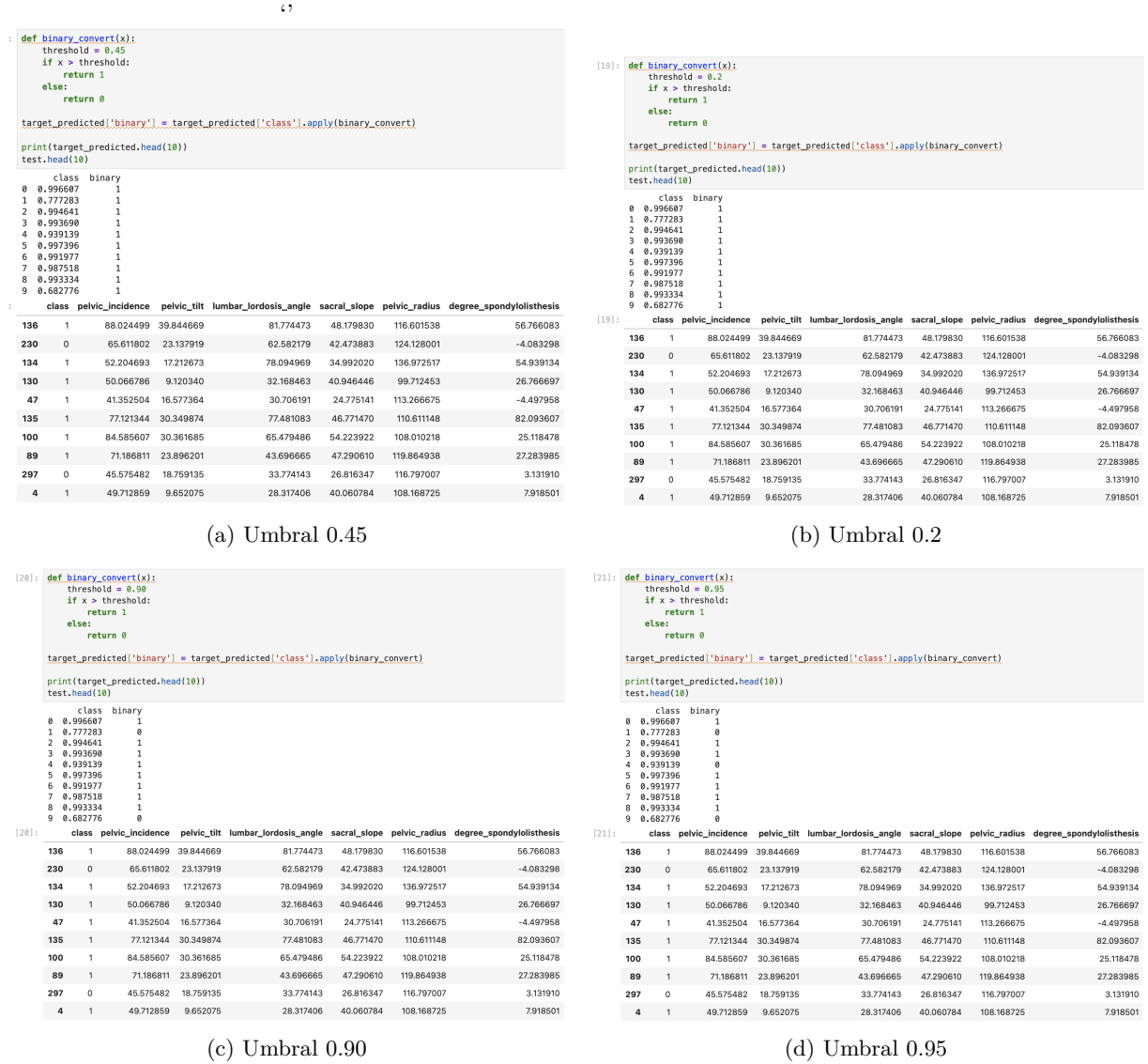


Figura 10: Efecto de distintos umbrales en la conversión de probabilidad a clase binaria.

## 5 Evaluación del Modelo (Lab 3.6)

### 5.1 La Matriz de Confusión

En este laboratorio evaluamos el modelo entrenado en los pasos anteriores sobre el conjunto de test. El modelo devuelve una probabilidad de pertenecer a la clase Abnormal, y para poder comparar contra las etiquetas reales (0/1) aplicamos un umbral  $t$  para convertir esa probabilidad en una decisión binaria.

Con  $t = 0,30$  (umbral usado en el notebook base), obtuvimos la siguiente matriz de confusión:

oprule	Predicho Normal	Predicho Abnormal
extbfReal Normal	7	3
extbfReal Abnormal	2	19

Cuadro 1: Matriz de confusión del modelo con umbral  $t = 0,30$ .

De aquí extraemos:  $TN = 7$ ,  $FP = 3$ ,  $FN = 2$  y  $TP = 19$ . En un contexto médico, el error más crítico suele ser el falso negativo ( $FN$ ), porque implica no detectar una anomalía real. Por otro lado, los

falsos positivos ( $FP$ ) generan alarmas innecesarias (más pruebas, más tiempo clínico), así que también conviene controlarlos.

A partir de la matriz calculamos las métricas principales:

- **Sensibilidad (TPR)** =  $\frac{TP}{TP+FN} = 90,48 \%$ .
- **Especificidad (TNR)** =  $\frac{TN}{TN+FP} = 70,00 \%$ .
- **Precisión (PPV)** =  $\frac{TP}{TP+FP} = 86,36 \%$ .
- **NPV** =  $\frac{TN}{TN+FN} = 77,78 \%$ .
- **Exactitud** =  $\frac{TP+TN}{TP+TN+FP+FN} = 83,87 \%$ .

En resumen, con este umbral el modelo detecta la mayoría de casos anormales (alta sensibilidad), pero aún produce algunas falsas alarmas en pacientes normales (FPR=30 %).

## 5.2 Desafío: Comparar Umbrales

Probamos con umbrales 0.25, 0.30 y 0.75 para ver cómo afectaban a las métricas:

Cambiar el umbral no re-entrena el modelo (por tanto no cambia el sobreajuste del entrenamiento), pero sí modifica la política de decisión y desplaza el equilibrio entre falsos positivos y falsos negativos:

- **Umbral bajo (0.25)**: el clasificador es más *agresivo* marcando *Abnormal*. Normalmente aumenta la sensibilidad (baja  $FN$ ) a costa de aumentar falsos positivos (sube  $FP$  y baja la especificidad).
- **Umbral medio (0.30)**: compromiso intermedio; en nuestro caso dio una sensibilidad del 90.48 % con especificidad del 70.00 %.
- **Umbral alto (0.75)**: el clasificador es más *conservador*. Normalmente reduce falsos positivos (sube la especificidad) pero aumenta el riesgo de falsos negativos.

En un escenario de cribado médico, suele ser preferible reducir  $FN$  (no dejar pasar pacientes con anomalía), aunque eso implique aceptar más  $FP$  y luego confirmar con pruebas adicionales.

## 5.3 Curva ROC y AUC

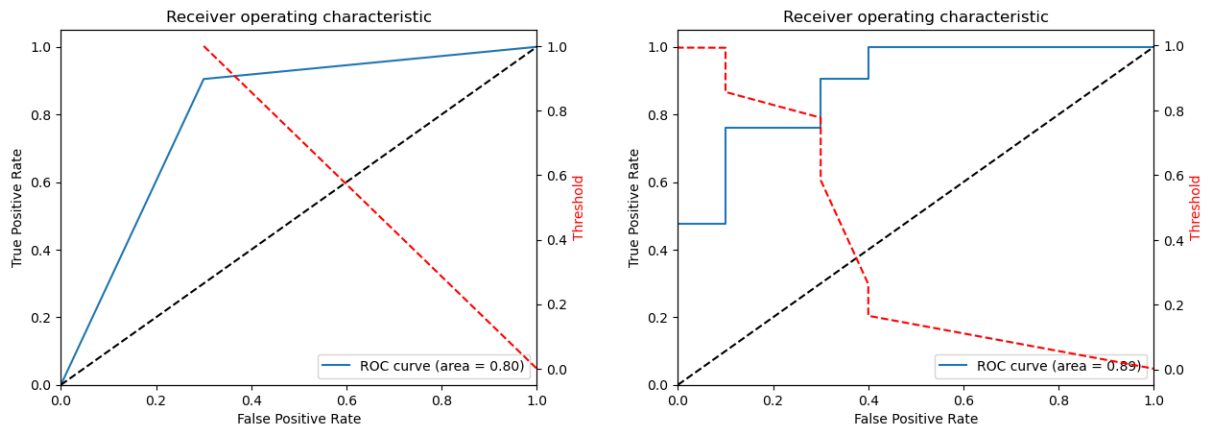
Generamos la curva ROC (Sensibilidad/TPR vs FPR) para visualizar el rendimiento del modelo a **todos** los umbrales posibles. El AUC (*Area Under Curve*) resume la ROC en un único número: si es 1.0 es perfecto, si es 0.5 es equivalente a azar, y valores por encima de 0.8 suelen considerarse buenos.

En nuestros resultados:

- Usando **probabilidades** (*target\_predicted*), obtuvimos **AUC**  $\approx 0,8905$ .
- Usando la salida **binaria** (*target\_predicted\_binary*), el AUC bajó a  $\approx 0,8024$ .

Esto es esperable: la ROC está pensada para construirse con **scores/probabilidades** (que preservan el orden y la confianza de cada predicción). Al binarizar se pierde información y la curva queda menos representativa.





(a) ROC usando probabilidades (AUC  $\approx 0,8905$ ).

(b) ROC usando predicción binaria (AUC  $\approx 0,8024$ ).

Figura 11: Comparativa de curvas ROC: probabilidades vs clases binarizadas.

## 6 Ajuste de Hiperparámetros (Lab 3.7)

En este laboratorio realizamos **ajuste automático de hiperparámetros** con Amazon SageMaker para intentar mejorar el modelo XGBoost entrenado anteriormente. La idea es ejecutar varios entrenamientos variando parámetros críticos y quedarnos con la configuración que optimiza una métrica objetivo.

### 6.1 Métricas del modelo base

Antes de optimizar, calculamos métricas del modelo original para tener una referencia. En este notebook el umbral de decisión se fijó en  $t = 0,50$  para convertir probabilidades en clases (0/1). Con ese umbral, el modelo base obtuvo:

- **Accuracy:** 83.87 %.
- **Sensitivity/TPR:** 90.48 %.
- **Specificity/TNR:** 70.00 %.
- **AUC (con salida binaria):** 0.8024.

### 6.2 Configuración del Tuner

Creamos un estimador de XGBoost y definimos la métrica de evaluación que el tuner debe minimizar. En el lab se usa `error@.40`, que mide la tasa de error binario evaluando como positivo si  $p > 0,40$  (y negativo en caso contrario). La configuración clave fue:

- `objective='binary:logistic'.`
- `eval_metric='error@.40'.`
- `objective_metric_name='validation:error'.`
- `objective_type='Minimize'.`

Después definimos rangos de búsqueda para los hiperparámetros con mayor impacto en XGBoost:

- **alpha** (regularización L1): [0, 100].
- **min\_child\_weight:** [1, 5].
- **subsample:** [0.5, 1].

- **eta** (learning rate): [0.1, 0.3].
- **num\_round**: [1, 50].

Finalmente lanzamos el HyperparameterTuner con `max_jobs=10` y `max_parallel_jobs=1` (para limitar coste y tiempo), habilitando `early_stopping_type='Auto'`.

<input type="radio"/>	<a href="#">sagemaker-xgboost-251211-1547-007-88d6844d</a>	12/11/2025, 4:54:20 PM	a few seconds	✔ Completed	✔ Completed	⊖ Reused	-
<input type="radio"/>	<a href="#">sagemaker-xgboost-251211-1547-006-8c7ca08d</a>	12/11/2025, 4:53:28 PM	a few seconds	✔ Completed	✔ Completed	⊖ Reused	-
<input type="radio"/>	<a href="#">sagemaker-xgboost-251211-1547-005-94300777</a>	12/11/2025, 4:52:38 PM	a few seconds	✔ Completed	✔ Completed	⊖ Reused	-
<input type="radio"/>	<a href="#">sagemaker-xgboost-251211-1547-004-a7b7b1b2</a>	12/11/2025, 4:51:53 PM	a few seconds	✔ Completed	✔ Completed	⊖ Reused	-
<input type="radio"/>	<a href="#">sagemaker-xgboost-251211-1547-003-5b07f0b2</a>	12/11/2025, 4:51:09 PM	a few seconds	✔ Completed	✔ Completed	⊖ Reused	-
<input type="radio"/>	<a href="#">sagemaker-xgboost-251211-1547-002-b6d11902</a>	12/11/2025, 4:50:23 PM	a few seconds	✔ Completed	✔ Completed	⊖ Reused	-
<input type="radio"/>	<a href="#">sagemaker-xgboost-251211-1547-001-a380bae4</a>	12/11/2025, 4:47:24 PM	3 minutes	✔ Completed	✔ Completed	⊖ Reused	-

Figura 12: Jobs generados por el ajuste de hiperparámetros (10 entrenamientos).

### 6.3 Ejecución y análisis

Tras lanzar el tuner, SageMaker ejecutó los 10 entrenamientos y dejó el proceso en estado Completed. Para analizar resultados, cargamos el resumen del tuning en un DataFrame con `HyperparameterTuningJobAnalytics`, que devuelve para cada job los hiperparámetros probados y el valor final de la métrica (`FinalObjectiveValue`). Ordenando por esa columna es fácil identificar la mejor combinación.

A partir del mejor entrenamiento, adjuntamos el tuner y creamos el modelo con `Estimator.attach(best_training_job)`. Después repetimos el flujo de evaluación con `Batch Transform` sobre el conjunto de test y, como en el modelo base, calculamos matriz de confusión y curva ROC.

### 6.4 Resultados obtenidos

Este lab muestra que el tuning no garantiza una mejora grande en todos los casos. En nuestro escenario influyen varios factores: dataset pequeño, espacio de búsqueda recortado y pocas combinaciones (10). En un entorno real, aumentar `max_jobs` (30–50+) y ampliar rangos suele ofrecer una mejora más consistente, aunque a costa de más tiempo y coste.