
TITULO MODULO 3 AWS ACADEMY FUNDAMENTOS DE MACHINE LEARNING

NOMBRE: JAVIER DAZA , WILLINGTON RIZZO

CARRERA: INGENIERIA INFORMATICA

ASIGNATURA: MACHINE LEARNING

PROFESOR:

FECHA: 22-09-2025

Objetivo del módulo

1.1.1 Objetivo del módulo y del grupo

En primer lugar, el objetivo principal del módulo fue **aprender cómo se entrena un modelo supervisado binario (0 y 1)** que nos ayude a **predecir si un paciente tiene anomalías en la columna o si está sano/normal**.

En segundo lugar, como grupo nos propusimos **seguir familiarizándonos con la plataforma de AWS**, en especial con el servicio **SageMaker**, y también **aprender y explorar las ventajas de usar notebooks** para este tipo de tareas.

Para cumplir con estos objetivos, lo que hicimos fue:

- **Importar el dataset** con las variables biomecánicas y la clase (normal/anormal).
- **Preparar los datos**, separándolos en entrenamiento, validación y prueba.
- **Almacenarlos en S3** en sus respectivos tres archivos.
- **Entrenar el modelo supervisado en SageMaker**.
- Finalmente, **evaluar el modelo** (como parte de los pasos de este y los siguientes labs).

Dataset utilizado

Usamos el **dataset de la columna vertebral** (Vertebral Column Dataset) proveniente del **repositorio UCI Machine Learning**. Este conjunto de datos contiene **310 registros** y **7 columnas** (6 características + 1 variable objetivo).

Las columnas son:

- `pelvic_incidence`
- `pelvic_tilt`
- `lumbar_lordosis_angle`
- `sacral_slope`
- `pelvic_radius`
- `degree_spondylolisthesis`
- `class` (etiqueta binaria: **Normal** o **Abnormal**)

Este dataset se descarga automáticamente al iniciar el laboratorio de AWS Academy en **Amazon SageMaker**, por lo que fue provisto directamente como parte de la práctica (no es creado por nosotros).

Elegimos este dataset porque es **idóneo para los primeros pasos en aprendizaje supervisado binario**. Su tamaño reducido y su claridad didáctica permiten enfocarse en comprender el **flujo de trabajo completo de Machine Learning**: importar datos, prepararlos, dividirlos en entrenamiento/validación/prueba, cargarlos en S3 y entrenar un modelo.

Además, resulta útil para el objetivo del módulo: **predecir si un paciente tiene anomalías en la columna o si es normal**, lo que conecta con un problema de negocio o investigación en el área de la salud.

Es importante decir que cada uno de los 3 data set que salieron del principal los pusimos en S3 para que pueda alimentar el entrenamiento, validación y test.

Entrenamiento en SageMaker

Una de las etapas más importantes en este trabajo es el **entrenamiento**, ya que aquí es donde aplicamos el algoritmo que nos permite obtener el modelo final. El algoritmo utilizado fue **XGBoost**, ejecutado en **Amazon SageMaker**.

Durante el entrenamiento se configuraron algunos **hiperparámetros**. Estos son valores que no se aprenden de los datos, sino que se establecen antes de entrenar para guiar cómo aprende el modelo. Entre los que usamos están:

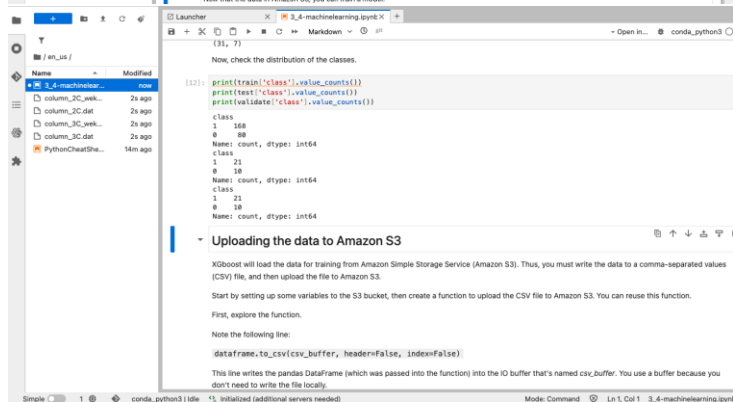
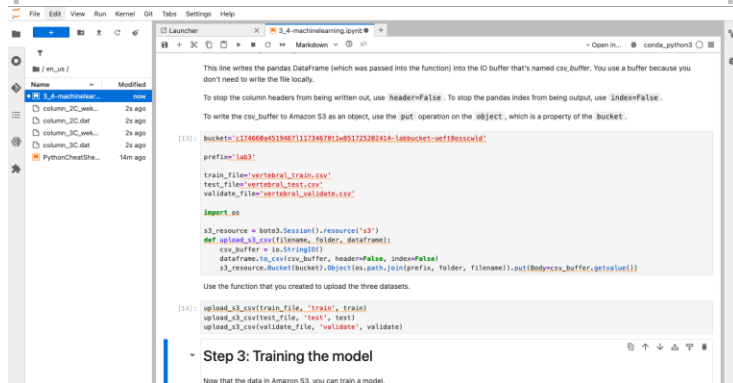
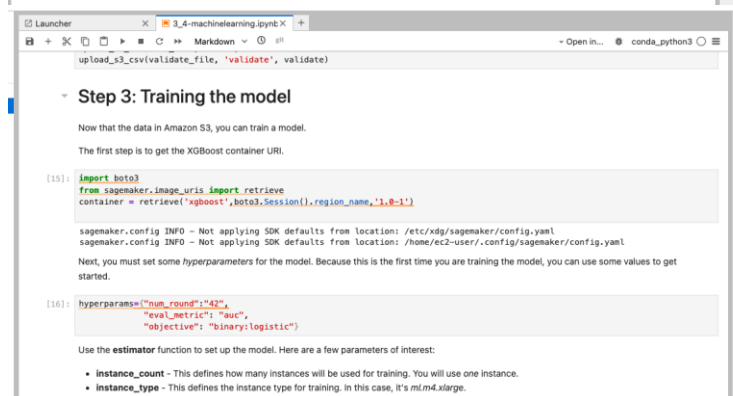
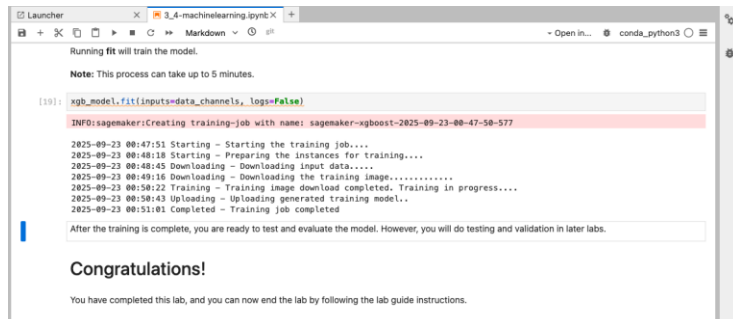
- **objective = binary:logistic** → indica que el problema es de clasificación binaria (0 = normal, 1 = anormal).
- **eval_metric = auc** → define que el rendimiento se mida con AUC, una métrica estándar para clasificación binaria.
- **num_round = 42** → cantidad de iteraciones de entrenamiento.

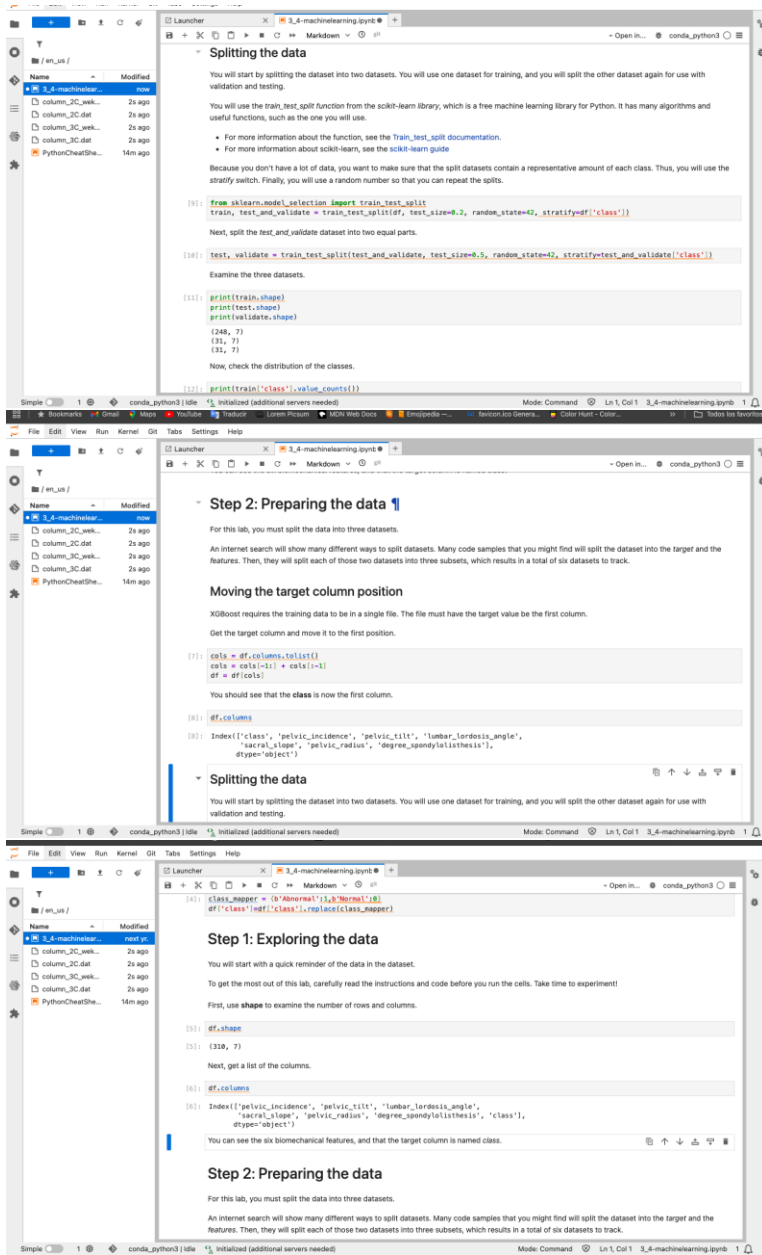
Estos hiperparámetros son importantes porque permiten ajustar el comportamiento del algoritmo, mejorar el rendimiento y evitar problemas como sobreajuste o bajo rendimiento.

El entrenamiento se realizó en una instancia **ml.m4.xlarge**, que es adecuada para este tipo de tareas porque ofrece un equilibrio entre capacidad de cómputo (CPU, memoria) y costo, suficiente para manejar datasets de tamaño mediano como este.

El proceso de entrenamiento tomó aproximadamente **5 minutos**, lo cual es un tiempo normal en SageMaker. Esto ocurre porque el sistema debe preparar los recursos en la nube, ejecutar todas las rondas de entrenamiento y luego guardar el modelo en S3.

Capturas de evidencia





The image displays three sequential screenshots of a Jupyter Notebook titled "3_4-machinelearning.ipynb", showing the process of splitting data into training and testing sets using sklearn.

Top Screenshot: Splitting the data

You will start by splitting the dataset into two datasets. You will use one dataset for training, and you will split the other dataset again for use with validation and testing.

You will use the `train_test_split` function from the `scikit-learn` library, which is a free machine learning library for Python. It has many algorithms and useful functions, such as the one you will use.

- For more information about the function, see the [Train_test_split documentation](#).
- For more information about `scikit-learn`, see the [scikit-learn guide](#).

Because you don't have a lot of data, you want to make sure that the split datasets contain a representative amount of each class. Thus, you will use the `stratify` switch. Finally, you will use a random number so that you can repeat the splits.

```
[9]: from sklearn.model_selection import train_test_split
train, test_and_validate = train_test_split(df, random_state=42, stratify=df['class'])
```

Next, split the `test_and_validate` dataset into two equal parts.

```
[10]: test, validate = train_test_split(test_and_validate, test_size=0.5, random_state=42, stratify=test_and_validate['class'])
```

Examine the three datasets.

```
[11]: print(train.shape)
print(test.shape)
print(validate.shape)

(148, 7)
(31, 7)
(31, 7)
```

Now, check the distribution of the classes.

```
[12]: print(train['class'].value_counts())
```

Middle Screenshot: Step 2: Preparing the data

For this lab, you must split the data into three datasets.

An internet search will show many different ways to split datasets. Many code samples that you might find will split the dataset into the target and the features. Then, they will split each of those two datasets into three subsets, which results in a total of six datasets to track.

Moving the target column position

XGBoost requires the training data to be in a single file. The file must have the target value be the first column.

Get the target column and move it to the first position.

```
[7]: cols = df.columns.tolist()
cols = cols[-1:] + cols[:-1]
df = df[cols]
```

You should see that the `class` is now the first column.

```
[8]: df.columns
```

Bottom Screenshot: Step 1: Exploring the data

You will start with a quick reminder of the data in the dataset.

To get the most out of this lab, carefully read the instructions and code before you run the cells. Take time to experiment!

First, use `shape` to examine the number of rows and columns.

```
[5]: df.shape

(158, 7)
```

Next, get a list of the columns.

```
[6]: df.columns
```

`Index(['class', 'pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pelvic_radius', 'degree_spondylolisthesis'], dtype='object')`

You can see the six biomechanical features, and that the target column is named `class`.

Step 2: Preparing the data

For this lab, you must split the data into three datasets.

An internet search will show many different ways to split datasets. Many code samples that you might find will split the dataset into the target and the features. Then, they will split each of those two datasets into three subsets, which results in a total of six datasets to track.

Launcher

3.4-machinelearning-lybnc

Because this solution is split across several labs in the module, you must run the following cells so that you can load the data.

Importing the data

By running the following cells, the data will be imported and ready for use.

Note: The following cells represent the key steps in the previous labs.

```
[1]: import warnings
warnings.simplefilter('ignore')
import pandas as pd
from sklearn import metrics
import numpy as np

[2]: f_zip = 'https://archive.ics.uci.edu/ml/machine-learning-databases/00212/vertebra_3_column_data.zip'
r = requests.get(f_zip, stream=True)
VertebraZip = zipfile.ZipFile(io.BytesIO(r.content))
VertebraZip.extractall()

[3]: data = arff.loadarff('column_3C_arff')
df = pd.DataFrame(data[0])

[4]: class_mapper = {'Abnormal':1,'Normal':0}
df['class'] = df['class'].replace(class_mapper)
```

Step 1: Exploring the data

You will start with a quick reminder of the data in the dataset.

To get the most out of this lab, carefully read the instructions and code before you run the cells. Take time to experiment!

First, use `shape` to examine the number of rows and columns.

Launcher

3.4-machinelearning-lybnc

Lab 3.4 - Student Notebook

Overview

This lab is a continuation of the guided labs in Module 3.

In this lab, you will split the data into three separate datasets:

- Training Set - This will be used to train the model.
- Validation Set - This will be used during training to validate the model.
- Test Set - This will be held back and used to produce metrics after the model is trained. You will use this dataset in an upcoming lab.

With the split data, you will train a XGBoost model by using Amazon SageMaker.

Introduction to the business scenario

You work for a healthcare provider, and want to improve detection of abnormalities in orthopedic patients.

You are tasked with solving this problem by using machine learning (ML). You have access to a dataset that contains six biomechanical features and a target of normal or abnormal. You can use this dataset to train an ML model to predict if a patient will have an abnormality.

About this dataset

This biomedical dataset was built by Dr. Henrique da Mota during a medical residence period in the Group of Applied Research in Orthopaedics (GARO) of the Centre Médico-Chirurgical de Réadaptation des Massues, Lyon, France. The data has been organized in two different, but related, classification tasks.

The first task consists in classifying patients as belonging to one of three categories:

Launcher

3.5-machinelearning-lybnc

```
Accuracy: 0.886
Precision: 0.857
Recall: 0.857
F1-score: 0.857
ROC-AUC: 0.898
```

```
[17]:
```

	Pred 0 (Normal)	Pred 1 (Anormal)
Real 0 (Normal)	7	3
Real 1 (Anormal)	3	18

```
[18]: def binarize(scores, thr):
    return (scores > thr).astype(int)

for thr in [0.50, 0.60, 0.65, 0.70]:
    y_pred_1 = binarize(scores, thr)
    prec, rec, f1, _ = precision_recall_fscore_support(y_true, y_pred_1, average='binary', pos_label=1)
    print(f"thr={thr:.2f} -> Precision={prec:.3f} Recall={rec:.3f} F1={f1:.3f}")

thr=0.50 -> Precision=0.864 Recall=0.985 F1=0.884
thr=0.60 -> Precision=0.857 Recall=0.857 F1=0.857
thr=0.65 -> Precision=0.857 Recall=0.857 F1=0.857
thr=0.70 -> Precision=0.858 Recall=0.818 F1=0.829
```

```
[17]: # 1) Definir y_true, y_score, y_pred
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support, roc_auc_score, confusion_matrix
import pandas as pd

y_true = test['class'].astype(int).values # etiqueta real
y_score = target_predicted['class'].astype(float).values # probabilidad
y_pred = target_predicted['binary'].astype(int).values # 0/1 tras umbral 0.65

# 2) Métricas principales
acc = accuracy_score(y_true, y_pred)
prec, rec, f1, _ = precision_recall_fscore_support(y_true, y_pred, average='binary', pos_label=1)
auc = roc_auc_score(y_true, y_score)

print(f"Accuracy: {acc:.3f}")
print(f"Precision: {prec:.3f}")
print(f"Recall: {rec:.3f}")
print(f"F1-score: {f1:.3f}")
print(f"ROC-AUC: {auc:.3f}")

# 3) Matriz de confusión
cm = confusion_matrix(y_true, y_pred)
pd.DataFrame(cm, index=['Real 0 (Normal)', 'Real 1 (Anormal)',
                        columns=['Pred 0 (Normal)', 'Pred 1 (Anormal)'])

Accuracy: 0.886
Precision: 0.857
Recall: 0.857
F1-score: 0.857
ROC-AUC: 0.898
```

```
[17]:
```

	Pred 0 (Normal)	Pred 1 (Anormal)
Real 0 (Normal)	7	3
Real 1 (Anormal)	3	18

```
test.head(10)
```

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
0	0	0.386087	1	1	0.777283	1	1
1	0	0.777283	1	1	0.994641	1	1
2	0	0.994641	1	1	0.993608	1	1
3	0	0.993608	1	1	0.939139	1	1
4	0	0.939139	1	1	0.997396	1	1
5	0	0.997396	1	1	0.991977	1	1
6	0	0.991977	1	1	0.987518	1	1
7	0	0.987518	1	1	0.993334	1	1
8	0	0.993334	1	1	0.682776	1	1
9	0	0.682776	1	1			

```
[16]: class pelvic_incidence pelvic_tilt lumbar_lordosis_angle sacral_slope pelvic_radius degree_spondylolisthesis
136 1 88.024499 39.844669 81.774473 48.179830 116.601538 56.766083
230 0 65.611802 23.137919 62.582179 42.473883 124.128001 -4.083298
134 1 52.204693 17.212673 78.094969 34.992020 136.972517 54.939134
130 1 50.066786 9.120340 32.168463 40.946446 99.712453 26.766697
47 1 41.352504 16.577364 30.706191 24.775141 113.266676 -4.497958
135 1 77.121344 30.349874 77.481083 46.771470 110.611148 82.093607
100 1 84.585607 30.361685 65.479486 54.223922 108.010218 25.118478
89 1 71.186811 23.896201 43.696665 47.290610 119.864938 27.283985
297 0 45.575482 18.769135 33.774143 26.816347 116.797007 3.131910
4 1 49.712859 9.652075 28.317406 40.060784 108.168725 7.918501
```

Note: The threshold in the `binary_convert` function is set to .65.

Challenge task: Experiment with changing the value of the threshold. Does it impact the results?

Note: The initial model might not be good. You will generate some metrics in the next lab, before you tune the model in the final lab.

```
python3 | title %s Initialized (additional servers needed) Mode: Command Ln 1, Col 1 3_5-machinelearning.ipynb 1
```

Last, before you perform a transform, configure your transformer with the input file, output location, and instance type.

```
[4]: batch_output = "s3://(I)/(O)/batch-out/".format(bucket_prefix, batch_X_file)
batch_input = "s3://(I)/(O)/batch-in/".format(bucket_prefix, batch_X_file)

xgb_transformer = xgb_model.transformer(instance_count=1,
                                       instance_type="ml.m4.xlarge",
                                       strategy="MultiRecord",
                                       assemble_with="line",
                                       output_path=batch_output)

xgb_transformer.transform(data=batch_input,
                        data_type="SPrefix",
                        content_type="text/csv",
                        split_type="line")

xgb_transformer.wait()
```

INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-09-23-02-31-28-540
 INFO:sagemaker:Creating transform job with name: sagemaker-xgboost-2025-09-23-02-31-29-896

After the transform completes, you can download the results from Amazon S3 and compare them with the input.

First, download the output from Amazon S3 and load it into a pandas DataFrame.

```
[4]: s3 = boto3.client('s3')
obj = s3.get_object(Bucket=bucket, Key="(I)/batch-out/(O)".format(prefix, 'batch-in.csv.out'))
target_predicted = pd.read_csv(io.BytesIO(obj['Body'].read()), sep=',', names=['class'])
target_predicted.head(5)
```

```
[12]: batch_X = test.iloc[:,1:]
batch_X.head()
```

	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	41.352504	16.577364	30.706191	24.775141	113.266676	-4.497958

Next, write your data to a CSV file.

```
[13]: batch_X_files=batch_X.to_csv()
upload_s3_csv(batch_X_file, 'batch-in', batch_X)
```

Last, before you perform a transform, configure your transformer with the input file, output location, and instance type.

Step 3: Terminating the deployed model

To delete the endpoint, use the `delete_endpoint` function on the predictor.

```
[11]: xgb_predictor.delete_endpoint(delete_endpoint_config=True)
```

INFO:sagemaker:Deleting endpoint configuration with name: sagemaker-xgboost-2025-09-23-02-22-01-971
 INFO:sagemaker:Deleting endpoint with name: sagemaker-xgboost-2025-09-23-02-22-01-971

Step 4: Performing a batch transform

When you are in the training-testing-feature engineering cycle, you want to test your holdout or test sets against the model. You can then use those results to calculate metrics. You could deploy an endpoint as you did earlier, but then you must remember to delete the endpoint. However, there is a more efficient way.

You can use the transformer method of the model to get a transformer object. You can then use the transform method of this object to perform a prediction on the entire test dataset. SageMaker will:

- Spin up an instance with the model
- Perform a prediction on all the input values
- Write those values to Amazon Simple Storage Service (Amazon S3)
- Finally, terminate the instance

You will start by turning your data into a CSV file that the transformer object can take as input. This time, you will use `iloc` to get all the rows, and all columns except the first column.

```
[12]: batch_X = test.iloc[:,1:]
batch_X.head()
```

```
[12]: pelvic_incidence pelvic_tilt lumbar_lordosis_angle sacral_slope pelvic_radius degree_spondylolisthesis
```


Launcher

3_5-machinelearning.ipynb

Open in... conda_python3

b"0.9966871844188952"

The result you get isn't a 0 or a 1. Instead, you get a probability score. You can apply some conditional logic to the probability score to determine if the answer should be presented as a 0 or a 1. You will work with this process when you do batch predictions.

For now, compare the result with the test data.

```
[18]: test.head(5)
```

```
[18]:
```

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

Question: Is the prediction accurate?

Challenge task: Update the previous code to send the second row of the dataset. Are those predictions correct? Try this task with a few other rows.

It can be tedious to send these rows one at a time. You could write a function to submit these values in a batch, but SageMaker already has a batch capability. You will examine that feature next. However, before you do, you will terminate the model.

Step 3: Terminating the deployed model

Launcher

3_5-machinelearning.ipynb

Open in... conda_python3

```
[17]:
```

```
serializer = sagemaker.serializers.CSVSerializer(),
instance_type='ml.m4.xlarge')

INFO:sagemaker:Creating model with name: sagemaker-xgboost-2025-09-23-02-22-01-971
INFO:sagemaker:Creating endpoint-config with name: sagemaker-xgboost-2025-09-23-02-22-01-971
INFO:sagemaker:Creating endpoint with name: sagemaker-xgboost-2025-09-23-02-22-01-971
```

Step 2: Performing predictions

Now that you have a deployed model, you will run some predictions.

First, review the test data and re-familiarize yourself with it.

```
[5]: test.shape
```

```
[5]: (31, 7)
```

You have 31 instances, with seven attributes. The first five instances are:

```
[6]: test.head(5)
```

```
[6]:
```

	class	pelvic_incidence	pelvic_tilt	lumbar_lordosis_angle	sacral_slope	pelvic_radius	degree_spondylolisthesis
136	1	88.024499	39.844669	81.774473	48.179830	116.601538	56.766083
230	0	65.611802	23.137919	62.582179	42.473883	124.128001	-4.083298
134	1	52.204693	17.212673	78.094969	34.992020	136.972517	54.939134
130	1	50.066786	9.120340	32.168463	40.946446	99.712453	26.766697
47	1	41.352504	16.577364	30.706191	24.775141	113.266675	-4.497958

You don't need to include the target value (class). This predictor can take data in the comma-separated values (CSV) format. You can thus get the first

Launcher

3_5-machinelearning.ipynb

Open in... conda_python3

```
sagemaker.get_execution_role(),
instance_count=1,
instance_type='ml.m4.xlarge',
method_payloads_output_location,
hyperparameters=hyperparams,
sagemaker_session=sagemaker.Session())

train_channel = sagemaker.inputs.TrainingInput(
"s3://(S3_BUCKET)/train",format(bucket_prefix,train_file),
content_type='text/csv')

validate_channel = sagemaker.inputs.TrainingInput(
"s3://(S3_BUCKET)/val",format(bucket_prefix,validate_file),
content_type='text/csv')

data_channels = {'train': train_channel, 'validation': validate_channel}

xgb_model.fit(inputs=data_channels, logs=False)

print('ready for hosting!')
```

```
INFO:sagemaker:Creating training-job with name: sagemaker-xgboost-2025-09-23-02-15-48-888

2025-09-23 02:15:50 Starting - Starting the training job.....
2025-09-23 02:16:25 Starting - Preparing the instances for training.....
2025-09-23 02:16:54 Downloading - Downloading input data.....
2025-09-23 02:17:06 Downloading - Downloading the training image.....
2025-09-23 02:18:27 Training - Training image download completed. Training in progress....
2025-09-23 02:18:48 Uploading - Uploading generated training model...
2025-09-23 02:19:00 Completed - Training job completed

ready for hosting!
```

Step 1: Hosting the model

Now that you have a trained model, you can host it by using Amazon SageMaker hosting services.



Resultados obtenidos

Como se puede ver en las capturas, el **entrenamiento se completó** después de aproximadamente **5 minutos**. Durante este proceso preparamos los datos, los cargamos en S3 y realizamos el entrenamiento en SageMaker. Al final, el trabajo terminó con éxito, generando un **modelo entrenado listo para ser utilizado** y posteriormente **evaluado mediante métricas de desempeño**.

Tabla de métricas principales (umbral 0.65):

Métrica	Valor
Accuracy	0.806
Precision	0.857
Recall	0.857
F1-score	0.857
ROC-AUC	0.890

Ejemplo de predicciones:

- Probabilidad para una muestra: 0.9966 \rightarrow Predicción = 1 (Anormal)
- Matriz de confusión (umbral 0.65):
 - Normales bien clasificados (True Negatives): 7
 - Normales mal clasificados (False Positives): 3
 - Anormales mal clasificados (False Negatives): 3
 - Anormales bien clasificados (True Positives): 18

Además, se generaron gráficas de desempeño:

- **Curva ROC:** AUC = 0.890
 - **Curva Precision-Recall:** AP = 0.944
 - **Matriz de confusión en heatmap**
-

Discusión

Interpretación de resultados:

- El modelo logró una buena capacidad de discriminación (AUC cercano a 0.9).
- Tanto la precisión como el recall son altos (0.857), lo que indica un buen equilibrio entre evitar falsos positivos y falsos negativos.
- El F1-score confirma este balance, mostrando que el modelo es consistente en ambas métricas.

Limitaciones del modelo:

- El dataset es relativamente pequeño (310 instancias en total, con solo 31 para test), lo que puede limitar la generalización.
- El modelo solo fue entrenado y evaluado con datos ya preparados, sin mayor exploración de hiperparámetros avanzados.
- No se consideraron técnicas de validación cruzada, lo cual podría dar una visión más robusta del desempeño.

Posibles mejoras:

- Ampliar el dataset con más ejemplos clínicos para evitar sobreajuste.
- Ajustar hiperparámetros de XGBoost (como `max_depth`, `learning_rate`, `n_estimators`) con un grid search o random search.
- Probar otros algoritmos supervisados (Random Forest, SVM, Redes Neuronales) y comparar su rendimiento.
- Aplicar validación cruzada y técnicas de balanceo si en versiones futuras se detecta desbalance entre clases.

Conclusión

- En este módulo se aprendió a **entrenar, desplegar y evaluar un modelo supervisado en Amazon SageMaker**, utilizando un dataset biomédico de columna vertebral.
- Se comprobó el flujo completo de ML en la nube: carga de datos en S3, entrenamiento con XGBoost, despliegue en un endpoint, predicciones y evaluación de métricas.
- El modelo mostró un rendimiento adecuado ($AUC = 0.890$, $F1 = 0.857$), siendo una buena base para aplicaciones de diagnóstico automático.
- **Aplicaciones potenciales:** este tipo de modelo puede usarse como apoyo clínico para identificar pacientes con posibles anormalidades en la columna, agilizando diagnósticos preliminares y priorización de casos.

Referencias

Aws

<https://us-east-1.console.aws.amazon.com/sagemaker/home?region=us-east-1#/notebooks-and-git-repos>