

DML-004 – GUÍA DE LABORATORIO 8

Entrenamiento con Heart Disease Dataset usando Azure ML SDK v2



Objetivos del laboratorio

- Comprender cómo registrar un dataset real en Azure ML como Data Asset.
- Aprender a preparar un script de entrenamiento reproducible con scikit-learn.
- Ejecutar el entrenamiento como un CommandJob en la nube.
- Analizar métricas de desempeño y outputs generados automáticamente.

El dataset

Trabajaremos con el dataset Heart Disease (Cleveland) .


- Contiene variables clínicas: edad, sexo, tipo de dolor en el pecho, presión arterial, colesterol, etc.
- La columna objetivo es target, donde 1 = enfermedad cardiaca y 0 = sano.
- Es un dataset clásico de clasificación binaria.

 Pregúntense: ¿qué tipo de problema de ML es este? (Clasificación binaria .

◆ Paso 1 – Registrar el dataset en Azure ML

En lugar de usar rutas locales, en Azure ML todo parte de registrar los datos.

- El archivo heart.csv se convierte en un Data Asset.
- Eso permite versionar, compartir y reutilizar el dataset.

 Pregunta de reflexión: ¿qué diferencia hay entre cargar un CSV local y registrar un Data Asset?

◆ Paso 2 – Script de entrenamiento (train.py)

El script debe:

1. Leer el dataset desde la ruta que le pase Azure ML (parámetro --data).
2. Separar X (features) e y (target).
3. Armar un pipeline con: imputación, escalado, y modelo Logistic Regression.
4. Hacer un train/test split (80/20).
5. Calcular métricas: accuracy, precision, recall, F1, AUC.
6. Guardar en ./outputs el modelo y las métricas.

 No te doy el código, pero piensa: ¿qué librerías de sklearn necesitarías para este flujo?



◆ Paso 3 – Definir el Job en Azure ML

Ahora necesitas un “puente” entre el script y Azure. Eso lo haces con un CommandJob:

- code="/src" → carpeta donde está tu train.py.
- command="python train.py --data \${{inputs.heart}}" → cómo llamar al script.
- inputs={"heart": ... } → aquí conectas el Data Asset que registraste.
- environment="AzureML-sklearn..." → ambiente con scikit-learn.
- compute="cpu-cluster" → dónde correrlo.

🚩 Pregunta para el grupo: ¿qué diferencia habría si lo corriéramos en un Compute Instance en vez de un Cluster?

◆ Paso 4 – Monitorear la ejecución

- Una vez enviado, el Job aparece en Azure ML Studio → Experiments.
- Desde ahí se pueden ver:
 - Logs en vivo (stdout/stderr).
 - Outputs generados (modelo, métricas, predicciones).
- También se puede revisar el estado desde el SDK (Running, Completed, Failed).

👉 Reflexiona: ¿por qué es importante que cada Job quede registrado con ID único?

◆ Paso 5 – Outputs esperados

Al finalizar, deberías obtener:

- Un archivo model.pkl con el modelo entrenado.
- Un metrics.json con valores de accuracy, precision, recall, F1, AUC.
- Una muestra de predicciones para comparar con la realidad.

📊 Se espera un accuracy entre 80% y 85%, dependiendo de la semilla y el split.



ENTREGABLES

Cada estudiante deberá presentar:

1. **Capturas de pantalla** de:
 - Registro del dataset como Data Asset.
 - Definición y envío del *CommandJob*.
 - Ejecución del Job en Azure ML Studio (o SDK) mostrando estado Completed.
2. **Evidencia de outputs:**



- Carpeta ./outputs con model.pkl y metrics.json.
 - Contenido del metrics.json con al menos accuracy, precision, recall, F1 y AUC.
3. **Explicación corta (máx. 1 página):**
- ¿Qué pipeline implementaron?
 - ¿Qué métricas obtuvieron y cómo interpretan el resultado?
4. **(Bonus opcional):** Ejecución del mismo experimento en un *Compute Instance* y comparación con el *Cluster*.