
LABORATORIO FINAL

DESPLIEGUE COMPLETO DE UN MODELO DE MACHINE LEARNING EN DOCKER

Objetivo:

El propósito de este laboratorio es desarrollar un flujo completo de implementación de un modelo de Machine Learning, abarcando las etapas de entrenamiento, optimización de hiperparámetros, almacenamiento del modelo y posterior despliegue en producción mediante Docker.

Descripción del Problema:

El proyecto consiste en construir una solución modular que incluya dos componentes principales, cada uno encapsulado en su propio contenedor Docker:

1. **Entrenamiento y Optimización del Modelo (model-train):** contenedor responsable de cargar los datos, entrenar un modelo de Machine Learning, optimizar sus hiperparámetros con **Optuna** y exportar el modelo final en formato **.pkl**. Los hiperparámetros y configuraciones de optimización deben definirse mediante un archivo **config.yaml**, lo que permite ajustar de forma flexible los parámetros del optimizador sin modificar el código base.
2. **Servicio del Modelo en Producción (model-serve):** contenedor encargado de cargar el modelo previamente entrenado y exponerlo para predicciones. Este servicio podrá operar de dos formas:
 - **Modo Batch:** recibe un archivo en formato Parquet, genera predicciones y guarda los resultados en un nuevo archivo.
 - **Modo API:** implementa un servicio **FastAPI** con un endpoint **/predict** para recibir datos en formato JSON y devolver las predicciones en tiempo real. Además, debe incluir un endpoint **/metrics** que muestre métricas relevantes del modelo, como precisión, número de predicciones procesadas, tiempo promedio de respuesta y fecha del último entrenamiento.

El flujo completo debe permitir que un modelo sea entrenado una sola vez y reutilizado para múltiples predicciones, garantizando la reproducibilidad y consistencia del proceso en diferentes entornos.

Dataset:

El conjunto de datos utilizado corresponde al **Hotel Booking Demand Dataset**, disponible en el GES. Este dataset contiene información sobre reservas de dos hoteles (uno urbano y otro resort), incluyendo variables numéricas, categóricas, ordinales y de fecha. El objetivo del modelo es predecir si una reserva será cancelada (**is_canceled = 1**) o no (**is_canceled = 0**).

Entre las variables más relevantes se incluyen:

- **Numéricas:** **lead_time**, **stays_in_weekend_nights**, **adults**, **babies**, **adr**.
- **Categóricas nominales:** **hotel**, **market_segment**, **distribution_channel**.
- **Alta cardinalidad:** **country**.
- **Ordinales:** **reserved_room_type**, **assigned_room_type**.

- **Fechas:** `arrival_date_year`, `arrival_date_month`, `arrival_date_day_of_month`.

El conjunto presenta valores nulos, por lo cual debe diseñar estrategias de imputación adecuadas y aplicar técnicas de codificación eficientes para evitar alta dimensionalidad.

Requerimientos:

1. Estructura del Proyecto

El proyecto debe mantener una organización modular, con la siguiente estructura sugerida:

```
project/
  train/
    train_model.py
    config.yaml
    requirements.txt
    Dockerfile

  serve/
    app.py
    requirements.txt
    Dockerfile

  model/
    model.pkl (generado)

  data/
    hotel_bookings.csv
    input_test.parquet
    output_preds.parquet

  README.md
```

2. Entrenamiento del Modelo

El contenedor `model-train` debe:

- Cargar el dataset de reservas de hotel desde la ruta definida como volumen.
- Entrenar un modelo de clasificación binaria para predecir la variable `is_canceled`.
- Implementar un optimizador de hiperparámetros utilizando `Optuna`, configurado a través del archivo `config.yaml`.
- Guardar el modelo entrenado en la carpeta `/output` dentro del contenedor, que corresponde a la carpeta local `/model`.

Ejemplo de ejecución:

```
docker build -t model-train:latest ./train
docker run --rm -v "$(pwd)/model:/output" model-train:latest
```

3. Configuración de Optuna

El archivo `config.yaml` debe incluir los parámetros básicos de configuración del optimizador, como:

```
optimizer:
  n_trials: 25
  direction: maximize
  metric: accuracy
search_space:
  learning_rate: [0.01, 0.1]
  n_estimators: [50, 200]
  max_depth: [3, 10]
```

Este archivo permitirá modificar fácilmente el comportamiento del optimizador y del proceso de búsqueda sin alterar el código principal.

4. Despliegue del Modelo en Producción

El contenedor `model-serve` debe cargar el modelo entrenado desde `/app/model/model.pkl` y ofrecer las siguientes modalidades:

- **Modo Batch:**

```
docker run --rm \
  -v "$(pwd)/data:/data" \
  -v "$(pwd)/model:/app/model" \
  model-serve:latest \
  --input /data/input_test.parquet \
  --output /data/output_preds.parquet
```

- **Modo API:**

```
docker run --rm -p 8000:8000 \
  -v "$(pwd)/model:/app/model" \
  model-serve:latest
```

En este modo, la API debe exponer los endpoints:

- `/predict`: recibe un JSON con datos y devuelve las predicciones.
- `/metrics`: muestra las métricas del modelo en formato JSON.

5. Preprocesamiento y Generalización

El flujo debe incluir un proceso de preprocesamiento automático que:

- Detecte tipos de variables y aplique transformaciones adecuadas (normalización, codificación, imputación).
- Permita operar sobre cualquier subconjunto del dataset de reservas de hotel, garantizando adaptabilidad ante distintos tamaños y estructuras de datos.

6. Métricas del Servicio

El endpoint `/metrics` debe mostrar información relevante, como:

```
{
  "model": "RandomForest",
  "accuracy": 0.83,
  "predictions_served": 250,
  "avg_response_time_ms": 14.6,
  "last_training": "2025-11-08"
}
```

Estas métricas deben actualizarse dinámicamente según la actividad del servicio o mantenerse registradas mediante logs.

Entregables:

El resultado final debe incluir:

- Dos contenedores Docker funcionales (`model-train` y `model-serve`) capaces de ejecutarse de forma independiente.
- El archivo `config.yaml` con los parámetros de Optuna.
- Un conjunto de instrucciones reproducibles en el archivo `README.md`, detallando los comandos para construir y ejecutar los contenedores en ambos modos de despliegue.
- Todo debe ser enviado mediante el GES.