# Clase 195 — Versionado de modelos y experimentos con MLflow

Parte: 4 — MLOps · Fuente: Huyen, Designing Machine Learning Systems cap. 6 + 11 + docs MLflow 2.x.  Duración estimada: 75 min.

# Clase 195 — Versionado de modelos y experimentos con MLflow

> *Parte: 4 — MLOps · Fuente: Huyen, Designing Machine Learning Systems cap. 6 + 11 + docs MLflow 2.x. Duración estimada: 75 min.*

## Objetivo

Trackear experimentos de ML (parámetros, métricas, artefactos, código) con MLflow Tracking, registrar modelos en el Model Registry con stages (None → Staging → Production → Archived), y entender la diferencia conceptual con DVC: DVC versiona el pipeline; MLflow versiona los runs.

## Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Levantar un servidor MLflow local (mlflow ui o mlflow server) y logear runs con mlflow.start_run() + log_param/log_metric/log_artifact.
- Usar autolog (mlflow.sklearn.autolog()) para que params/metrics se capturen sin código boilerplate.
- Registrar un modelo en el Model Registry (mlflow.register_model) y transicionarlo de Staging a Production con la API o la UI.
- Cargar un modelo registrado en producción con mlflow.pyfunc.load_model("models:/MiModelo/Production").
- Configurar un backend store (PostgreSQL) y un artifact store (S3) para uso en equipo.

## Temas

| # | Tema | Por qué importa |
|---|---|---|
| 1 | Tracking server: backend store + artifact | Dónde van los metadatos vs los blobs (mode |
| 2 | Runs, experiments, tags | Unidad atómica + agrupación + búsqueda. |
| 3 | log_param, log_metric, log_artifact, log_m | Las 4 primitivas. |
| 4 | Autolog por framework (sklearn, PyTorch, X | Cero boilerplate cuando funciona — y sus l |
| 5 | Model Registry + stages | El camino entre "entrené esto" y "esto sir |
| 6 | MLflow Models flavor (pyfunc, sklearn, pyt | Un mismo modelo se puede cargar en N runti |

## Definiciones y características

- MLflow Tracking: API + UI para registrar runs (params, metrics, artifacts, código source). Cada run pertenece a un experiment (agrupación lógica, ej. "fraud-detection-v2").
- Backend store: dónde se guardan los metadatos del run (params, metrics, tags). Opciones: filesystem (default ./mlruns), SQLite, PostgreSQL, MySQL. Para equipo: Postgres.
- Artifact store: dónde se guardan los blobs (modelos, plots, datasets exportados). Opciones: filesystem local, S3, GCS, Azure Blob, HDFS. Para equipo: S3/GCS.
- Run: ejecución atómica de un experimento. Tiene un run_id (UUID), start_time, end_time, status, y N

params/metrics/tags/artifacts.
- Autolog: hook que intercepta llamadas al framework (sklearn, XGBoost, PyTorch Lightning, …) y registra params/metrics sin código. Activar con mlflow.<framework>.autolog() antes del .fit().
- Model Registry: catálogo central de modelos versionados. Cada modelo registrado tiene versiones (v1, v2, …), cada versión tiene un stage (None/Staging/Production/Archived) y aliases (desde MLflow 2.5+: @champion, @challenger).
- MLflow Models: formato estándar para guardar un modelo. Define un MLmodel YAML con uno o más flavors (sklearn, pyfunc, pytorch). pyfunc es el flavor universal — cualquier modelo se sirve como predict(input_df).

## Dataset / recursos

- Dataset: California Housing (sklearn.datasets.fetch_california_housing) — 20 640 filas, regresión, sin problemas de PII.
- Backend local: SQLite (mlflow server --backend-store-uri sqlite:///mlflow.db).
- Artifact local: ./mlruns/artifacts.
- Librerías: mlflow>=2.10, scikit-learn, xgboost.

## Ejercicios

1. Tracking manual: entrená un LinearRegression y un RandomForestRegressor sobre California Housing. Para cada uno, abrí un run y logueá n_estimators (si aplica), max_depth, rmse_train, rmse_test. Comparalos en la UI (mlflow ui --port 5000).
2. Autolog: repetí el ejercicio anterior con mlflow.sklearn.autolog() y verificá que params + metrics + el modelo entero quedaron registrados sin código extra.
3. Sweep de hiperparámetros: corré 10 runs variando max_depth {3, 5, 10, 15, 20} y n_estimators {50, 200}. Usá mlflow.search_runs() para encontrar el mejor por rmse_test.
4. Registry: registrá el mejor modelo (mlflow.register_model(model_uri, "housing-rf")). Transicionalo a Staging, después a Production desde la API (MlflowClient.transition_model_version_stage).
5. Carga en "producción": en una celda nueva, simulá un servicio que carga el modelo Production y predice una fila: model = mlflow.pyfunc.load_model("models:/housing-rf/Production"); model.predict(X_one).

## Homework verificable

Notebook + carpeta mlruns/ que contenga:

1. ≥ 10 runs en un experimento llamado homework-195.
2. Cada run con al menos model_type, rmse_test, r2_test, y el modelo logueado (mlflow.sklearn.log_model).
3. Un modelo registrado como housing-best con al menos una versión en stage Production.
4. Una celda final que carga models:/housing-best/Production y predice sobre 5 filas de test.

Criterio de aceptación: mlflow.search_runs(experiment_names=["homework-195"], order_by=["metrics.rmse_test ASC"]) devuelve la fila top, y su run_id coincide con la versión Production del modelo registrado.

## Errores comunes

| Síntoma / mensaje | Causa y cómo arreglar |
|---|---|
| Los runs van a ./mlruns y no aparecen en l | La UI está leyendo otro --backend-store-ur |
| mlflow.sklearn.autolog() no registra el mo | El modelo se loguea solo si el .fit() ocur |
| ConnectionRefusedError contra el tracking | El MLFLOW_TRACKING_URI apunta a un server |
| Artefactos vacíos en la UI cuando uso S3 | El cliente no tiene credenciales AWS. Fix: |
| RestException: RESOURCE_ALREADY_EXISTS al | Ya existe un modelo con ese nombre. Fix: u |
| El stage Production está deprecated en log | Desde MLflow 2.9+ se recomienda aliases (@ |

## Preguntas frecuentes

¿MLflow vs Weights & Biases vs Neptune?

MLflow es open-source, self-hosted, y el estándar de facto en empresas que no quieren un SaaS externo. W&B y Neptune tienen mejores UIs y features colaborativos (reports, sweeps integrados), pero son SaaS con costo por usuario. Para aprender y para deploys self-hosted: MLflow. Para equipos de research grandes con presupuesto: W&B.

¿MLflow vs DVC?

Resuelven cosas distintas. DVC (Clase 194): versionado de datos + pipeline reproducible (vive en el repo git). MLflow: tracking de runs + model registry (vive en un server). Se usan juntos: el dvc.yaml define cómo correr el pipeline; dentro del script, MLflow logea cada run.

¿Debo usar mlflow server o mlflow ui?

mlflow ui es la UI sobre un backend filesystem (local). mlflow server levanta también una REST API para que otros procesos/máquinas registren runs remotamente. Para equipo: server. Para vos solo: ui alcanza.

¿pyfunc o el flavor nativo (sklearn, pytorch)?

Si vas a cargar el modelo desde Python y usar features específicas (acceso a .coef_, .feature_importances_): flavor nativo. Si vas a servirlo detrás de una API REST o desde otro lenguaje: pyfunc — abstrae todo a predict(DataFrame) → array.

¿Qué pasa si pierdo mlflow.db?

Perdés metadatos (params, metrics, run history). Los artefactos sobreviven si están en S3. Backup: pg_dump si usás Postgres, o copiar mlflow.db si SQLite. En producción: backend Postgres con backups gestionados por la nube.

¿Cómo versionar el código junto al run?

MLflow registra automáticamente el commit git (mlflow.source.git.commit tag) si corrés desde un repo git limpio. Si tenés cambios sin commitear, el tag dice dirty. Política sana: el CI registra runs solo desde commits firmados.

## Referencias

- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 6 (experimentos) y cap. 11 (model serving).
- MLflow Docs — Tracking + Model Registry + Models.
- mlflow.sklearn.autolog — qué se captura automáticamente.

- MLflow Model Registry — stages, aliases (2.5+), webhooks.
- Comparativa MLflow vs W&B vs Neptune (2024) — tomar con grano de sal (parte interesada).

## Siguiente clase

Clase 196 — Feature stores (Feast)

## Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import os, tempfile, shutil
from pathlib import Path
WORK = Path(tempfile.gettempdir()) / 'mlflow_demo'
if WORK.exists(): shutil.rmtree(WORK)
WORK.mkdir(); os.chdir(WORK)

import mlflow
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

mlflow.set_tracking_uri(f'file:{WORK}/mlruns')
mlflow.set_experiment('housing-demo')

X, y = fetch_california_housing(return_X_y=True, as_frame=True)
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.2, random_state=42)
print('train:', Xtr.shape, 'test:', Xte.shape)
```

## Archivos complementarios

- notebook.ipynb