Clase 194 — Versionado de datos con DVC

Parte: 4 — MLOps · Fuente: Huyen, Designing Machine Learning Systems cap. 6 + docs oficiales DVC 3.x. Duración estimada: 75 min.

Clase 194 — Versionado de datos con DVC

> *Parte: 4 — MLOps · Fuente: Huyen, Designing Machine Learning Systems cap. 6 + docs oficiales DVC 3.x. Duración estimada: 75 min.*

Objetivo

Versionar datasets pesados (>100 MB, que git rechaza) con DVC 3.x, separando qué dato se usó (puntero en git, ~200 bytes) de dónde vive el blob real (S3, GCS, Azure, disco local). Reproducir un entrenamiento de hace 3 meses con git checkout <sha> && dvc pull y entender por qué dvc.lock es a los datos lo que package-lock.json es a npm.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Inicializar un repo con dvc init y rastrear un dataset con dvc add data/raw.csv (entiende que git solo guarda el .dvc pointer).
- Configurar un remote (dvc remote add -d origin s3://bucket/path) y sincronizar con dvc push / dvc pull.
- Construir un pipeline reproducible declarando stages en dvc.yaml (deps, outs, params, metrics) y ejecutarlo con dvc repro.
- Comparar experimentos con dvc exp run, dvc exp show, dvc exp diff — alternativa ligera a MLflow para casos simples.
- Diagnosticar ERROR: output 'X' is already tracked by SCM y otros choques DVC ↔ git.

Temas

#	Tema	Por qué importa
1	El problema: git LFS no escala a TB	Costo por GB, throughput, y bloqueo en git
2	Modelo mental DVC: pointer en git + blob e	Git versiona el hash MD5/SHA-256; el dato
3	dvc add vs dvc.yaml stages	Trackeo manual vs pipeline declarativo.
4	Remotes (S3, GCS, Azure, SSH, local)	Donde realmente está la data; dvc remote m
5	dvc.lock — el "lockfile" de tu pipeline	Hashes congelados de inputs/outputs por st
6	dvc exp — branching-less experiments	Ejecutar 20 corridas sin ensuciar git log.

Definiciones y características

- DVC (Data Version Control): herramienta open-source que extiende git para datos/modelos. Versión actual: 3.x (rompió compat con 2.x — dvc.yaml mismo, pero comandos exp cambiaron).
- .dvc file: archivo YAML pequeño (~200 B) con md5, size, path del blob. Va a git. El blob va al remote.
- Cache local (.dvc/cache/): copia local de los blobs. dvc add mueve el archivo al cache y deja un link (reflink/hardlink/symlink) en el working tree. Por eso dvc add no duplica espacio en disco (en filesystems que soportan reflinks: APFS, XFS, btrfs, ReFS).
- Remote: backend remoto (S3/GCS/Azure/SSH/HTTP/local-path). dvc push sube cache local → remote; dvc pull baja remote → cache → working tree.

- Stage (en dvc.yaml): unidad reproducible. Tiene cmd, deps (inputs), outs (outputs), opcionalmente params (de params.yaml) y metrics. DVC re-ejecuta solo los stages cuyos hashes de deps cambiaron — como make, pero por contenido en vez de timestamp.
- dvc.lock: hashes congelados de deps/outs después de la última dvc repro exitosa. Va a git. Es lo que hace reproducible el pipeline.
- dvc exp run: ejecuta el pipeline con (opcionalmente) parámetros distintos y guarda el resultado como "experimento" — un commit interno que no contamina git log hasta que hagás dvc exp apply o dvc exp branch.

Dataset / recursos

- Dataset: seaborn.load_dataset('titanic') exportado a data/raw/titanic.csv (~60 KB — pequeño a propósito para que la clase corra sin S3 real).
- Remote demo: directorio local /tmp/dvc-remote-demo (simula S3 sin credenciales). El comando para S3 real se muestra pero no se ejecuta.
- Librerías: dvc[s3] (o dvc pelado si remote local), pandas, scikit-learn.

Ejercicios

1. Setup mínimo: inicializá un repo git + DVC (git init && dvc init). Generá data/raw/titanic.csv, hacelo trackear con dvc add data/raw/titanic.csv. Inspeccioná el .dvc resultante con cat data/raw/titanic.csv.dvc y entendé los campos md5, size, path.
2. Remote local: configurá un remote en /tmp/dvc-remote-demo con dvc remote add -d local /tmp/dvc-remote-demo. Hacé dvc push y verificá con ls /tmp/dvc-remote-demo/files/md5/ que el blob aparece como <2-char-prefix>/<resto-del-hash>.
3. Pipeline declarativo: creá dvc.yaml con dos stages — prepare (lee raw/titanic.csv, elimina nulos, escribe data/processed/clean.csv) y train (entrena un LogisticRegression, escribe model.pkl y metrics.json). Corré dvc repro y observá dvc.lock.
4. Reproducción: tocá un parámetro en params.yaml (test_size: 0.2 → 0.3). Volvé a correr dvc repro y verificá que ambos stages se re-ejecutan (porque prepare no depende de params, pero train sí — y el output de train cambió). Compará con dvc repro --dry.
5. Experimentos sin branching: dvc exp run -S 'train.C=0.1' tres veces con valores distintos de C. Listalos con dvc exp show. Aplicá el mejor con dvc exp apply <hash>.

Homework verificable

Pipeline DVC en un repo nuevo con:

1. dvc.yaml con stages prepare → train → evaluate.
2. params.yaml con al menos test_size, random_state, model.C.
3. metrics.json con accuracy y f1 que se reporten con dvc metrics show.
4. Tres corridas dvc exp run variando model.C {0.01, 1, 100}.
5. Output de dvc exp show --no-pager adjunto como experiments.txt.

Criterio de aceptación: dvc repro corre limpio desde cero (rm -rf .dvc/cache data/processed model.pkl metrics.json && dvc pull && dvc repro) y produce los mismos hashes en dvc.lock.

Errores comunes

Síntoma / mensaje	Causa y cómo arregl				
ERROR: output 'data/r	El archivo ya está en g				
dvc push no hace nad	El cache local y el rem				
dvc repro re-ejecuta to	Modificaste un archivo				
Pierdo el cache y dvc	Probablemente el rem				
Git push lentísimo des	Alguien hizo git add da	sort -u \	xargs -I{} du -h {} 2>/d	sort -h \	tail. Después git filter-repo` o BFG pa

Preguntas frecuentes

¿DVC vs git LFS?

LFS es más simple (un binario, git lfs track "*.csv") pero (1) cobra por GB en GitHub Enterprise/GitLab, (2) bloquea git push hasta que el blob suba, (3) no tiene noción de pipeline reproducible ni de experimentos. DVC desacopla storage del provider git y agrega dvc.yaml/dvc.lock/dvc exp. Para datasets <1 GB y casos triviales, LFS alcanza; para ML serio, DVC.

¿DVC vs MLflow?

Son complementarios, no competidores. DVC versiona datos + pipeline (lado git). MLflow trackea runs (params, metrics, artifacts) y registra modelos en un model registry (Clase 195). En la práctica: DVC para reproducibilidad determinística del pipeline; MLflow para comparar 500 experimentos en un dashboard.

¿Tengo que usar S3? ¿Funciona en una notebook personal?

No, funciona con remote local (dvc remote add -d local /path/a/carpeta). Útil para aprender, o para equipos chicos con un NAS compartido. Para producción real con varios colaboradores: S3/GCS/Azure.

¿Qué pasa con datos sensibles (PII)?

DVC no encripta por sí mismo. Si el remote es S3 con SSE-KMS, ya está encriptado en reposo. Para PII fuerte: bucket privado + IAM rol por equipo + auditoría con CloudTrail. No subas datos con PII a un remote público (S3 público, GCS público) — DVC no te avisa.

¿dvc.lock lo edito a mano?

No. Lo regenera dvc repro. Editarlo manualmente rompe la garantía de reproducibilidad. Si necesitás "forzar" un hash, usá dvc commit (con cuidado — saltea la ejecución).

¿Funciona con Jupyter notebooks?

Sí, pero el notebook en sí sigue versionado por git. DVC entra cuando un cell genera/lee artefactos pesados. Patrón común: notebook orquesta, pero las funciones heavy van a un src/, y los datasets están bajo DVC. (Para diff limpio de notebooks: nbdime o jupytext — fuera del scope de esta clase.)

Referencias

- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 6 — Model Development and Offline Evaluation, sección sobre data versioning.
- DVC 3.x documentation — empezar por Get Started.
- DVC dvc.yaml reference — todos los campos de stage.
- Iterative Studio — UI hosted opcional para visualizar pipelines DVC (no requerida, gratis para repos públicos).

- Comparativa práctica DVC vs LFS vs LakeFS: Data versioning landscape 2024.

Siguiente clase

Clase 195 — Versionado de modelos y experimentos con MLflow

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import os, shutil, subprocess, tempfile, json
from pathlib import Path
import pandas as pd
import seaborn as sns

WORK = Path(tempfile.gettempdir()) / 'dvc_demo'
REMOTE = Path(tempfile.gettempdir()) / 'dvc_remote_demo'
for p in (WORK, REMOTE):
    if p.exists(): shutil.rmtree(p)
    p.mkdir(parents=True)
os.chdir(WORK)
print('cwd:', Path.cwd())
```

Archivos complementarios

- notebook.ipynb