Clase 203 — Reentrenamiento programado

Parte: 4 — MLOps · Fuente: Huyen cap. 9 + Airflow docs + Prefect docs. Duración estimada: 75 min.

Clase 203 — Reentrenamiento programado

> *Parte: 4 — MLOps · Fuente: Huyen cap. 9 + Airflow docs + Prefect docs. Duración estimada: 75 min.*

Objetivo

Convertir el reentrenamiento en un proceso programado, auditado y reversible: DAG que cada N horas/días corre pull-data → validate → train → evaluate → promote-if-better → notify, con shadow/canary (Clase 204) antes del rollout pleno. Decidir entre schedule fijo, trigger por drift y trigger por degradación según el caso.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diseñar un DAG (Airflow/Prefect/Dagster) que orqueste el reentrenamiento completo.
- Implementar el patrón champion-challenger: el modelo Production sigue sirviendo hasta que el challenger demuestra ser mejor en métricas de validación + en shadow.
- Configurar tres estrategias de trigger: cron(0 2 MON), on_drift > threshold, on_performance_degraded.
- Garantizar idempotencia (re-runs no duplican datos) y observabilidad (logs estructurados, alertas en fallas).
- Diferenciar online learning (modelo actualiza con cada nueva muestra) de continual training (re-trainings periódicos).

Temas

#	Tema	Por qué importa
1	Triggers: schedule vs drift vs degradation	Sin trigger correcto, retrainings caros si
2	DAG = grafo de tareas con deps	Airflow/Prefect/Dagster son variaciones de
3	Champion-challenger	Cómo evitar regresiones en producción.
4	Idempotencia	Re-runs no deben duplicar/destruir state.
5	Online vs continual training	Trade-off frescura vs estabilidad.
6	Catastrophic forgetting	Retraining puede olvidar lo aprendido si d

Definiciones y características

- DAG (Directed Acyclic Graph): grafo de tareas con dependencias, ejecutable por un scheduler. Airflow inventó la categoría; Prefect/Dagster/Mage son alternativas modernas.
- Champion: modelo activo en producción (alias @champion en MLflow, Clase 195).
- Challenger: candidato nuevo entrenado. Promueve a champion solo si pasa todos los gates (validation metric + shadow + canary).
- Promotion gate: criterio que debe cumplirse para que el challenger reemplace al champion. Ej: f1_test > f1_champion + 0.005 AND no_regression_in_protected_slices.
- Idempotencia: re-ejecutar la misma tarea con los mismos inputs produce el mismo output, sin efectos colaterales acumulativos. Implementación típica: usar execution_date como key, sobrescribir destino, evitar INSERT INTO sin WHERE NOT EXISTS.

- Backfill: re-ejecutar el DAG para fechas pasadas (Airflow soporta nativo). Útil para corregir bugs o llenar gaps.
- Online learning: el modelo actualiza con cada muestra/mini-batch que llega. Funciona bien con SGD-based (LR, RF online, RL). No funciona con XGBoost/RF batch.
- Continual training: re-trainings periódicos (cada hora/día/semana) sobre data acumulada. Más común y más simple que online learning.
- Catastrophic forgetting: cuando retraining sobre data reciente "olvida" patrones que aprendió antes. Mitigación: incluir data histórica en cada retraining, regularización (EWC en deep learning).

Dataset / recursos

- Pipeline target: Airflow local (Docker), o Prefect Cloud free tier, o GitHub Actions schedule.
- Librerías: apache-airflow>=2.9, prefect>=3, mlflow, evidently.

Ejercicios

1. DAG mínimo (Airflow o Prefect): 5 tareas: pull_data → validate_data → train → evaluate → promote. Cada tarea es una función Python. Schedule diario.
2. Champion-challenger: en promote, comparar challenger.f1 vs champion.f1 (leídos de MLflow). Promover solo si challenger.f1 > champion.f1 + 0.005. Sino, logear [skipped] challenger no es significativamente mejor y dejar champion intacto.
3. Idempotencia: ejecutá el DAG dos veces seguidas para la misma fecha. Verificá que no se duplican filas en data/processed/, ni se crean dos registros en MLflow con el mismo execution_date.
4. Trigger por drift: agregá una tarea inicial check_drift que (a) si PSI > 0.2 continúa el DAG, (b) si no, hace raise AirflowSkipException (skipea el resto). Resultado: solo se entrena si hay drift.
5. Backfill: simulá un bug en la tarea validate_data que ya corrió bien por una semana. Fixeá el bug, airflow dags backfill --start-date X --end-date Y. Verificá que se re-procesan los días afectados sin duplicar registros downstream.

Homework verificable

DAG en producción (o local con docker-compose) que:

1. Corre diariamente a las 02:00 UTC.
2. Pull últimos 30 días de data, valida con Great Expectations (Clase 206) — falla con alert si no pasa.
3. Re-entrena, evalúa contra hold-out + slices de fairness, registra en MLflow.
4. Compara contra champion: promueve a @challenger (alias) si métrica supera umbral.
5. Tarea shadow_test que durante 24 h compara predicciones challenger vs champion sobre tráfico real (Clase 204).
6. Tarea promote_champion (manual o auto) que reemplaza alias @champion solo si shadow OK.
7. Alertas Slack en cada falla del DAG + un resumen diario [OK] o [NO TRAIN, no drift].

Criterio de aceptación: en una semana de operación, el DAG corrió 7 veces, ≥1 promoción real ocurrió, ≥1 skip por no-mejora ocurrió, y ningún re-run duplicó datos.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar

DAG corre OK pero modelo no mejora	Probablemente estás re-entrenando solo con
Re-run del DAG duplica filas en la DB de f	Insert sin idempotencia. Fix: INSERT ... O
Catastrophic forgetting visible: f1 sobre	El nuevo retraining sobre data fresca borr
Champion-challenger nunca promueve	Umbral muy estricto, o el modelo nuevo no
DAG falla pero nadie se entera	Sin alerta en on_failure_callback. Fix: co
TaskGroup con N tareas paralelas satura el	Sin paralelismo control. Fix: en Airflow,
MLflow runs sin tag de dag_run_id	Cuando algo falla, no podés trazar qué cor

Preguntas frecuentes

¿Airflow, Prefect, Dagster, Mage, Kubeflow Pipelines?

Airflow: el estándar, máxima madurez, syntax verbosa, multi-cloud. Prefect 3: API Python moderna, hybrid execution (workers locales + cloud). Dagster: orientado a software-defined assets (data-aware). Mage: notebooks-first, UI fuerte. Kubeflow Pipelines: K8s-native, orientado a ML específicamente. Para empezar en ML: Prefect o Dagster suelen ser más rápidos que Airflow.

¿Cada cuánto retrainear?

Depende del drift rate del dominio: NLP general (cambios estacionales): mensual. Fraud (atacantes adaptativos): diario u online. Recommendation (catalog drift): horario. Empezar conservador y acelerar si las métricas justifican.

¿Cuándo NO promover automáticamente?

(1) Decisiones de alto impacto (préstamos, salud): siempre human-in-the-loop. (2) Cuando el A/B test online no es factible: requerir aprobación manual. (3) Cuando el dataset tiene mucho ruido reciente: filtrar antes de retrainar.

¿schedule_interval o trigger por evento?

Para producción crítica: ambos. Schedule fijo como red de seguridad ("al menos 1 vez/día") + trigger por drift/degradación para ser reactivo. Falla del trigger no deja al sistema sin retraining.

¿Cómo manejo training caro (días) en un DAG?

Tarea train no corre en el worker Airflow — manda job a Vertex AI / SageMaker / Ray cluster vía API. La tarea Airflow es un proxy que polea hasta completion (SageMakerTrainingOperator lo hace).

¿Y si el retraining empeora producción?

Por eso existen los gates: shadow → canary → full rollout. Si la métrica online cae después de canary 10%: rollback automático del alias @champion al modelo previo, mantenido en el registry.

Referencias

- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 9 — Continual Learning and Test in Production.
- Airflow docs — DAGs y operators.
- Prefect docs — flows modernos.
- Kirkpatrick et al. Overcoming catastrophic forgetting in neural networks (PNAS, 2017) — EWC.
- SageMaker Pipelines — equivalente managed AWS.

Siguiente clase

Clase 204 — Shadow deployment y canary releases

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

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

import mlflow
mlflow.set_tracking_uri(f'file:{WORK}/mlruns')
mlflow.set_experiment('continual-training')
```

Archivos complementarios

- notebook.ipynb