Clase 196 — Feature stores (Feast)

Parte: 4 — MLOps · Fuente: Huyen cap. 6 + Feast docs 0.40+. Duración estimada: 70 min.

Clase 196 — Feature stores (Feast)

Parte: 4 — MLOps · Fuente: Huyen cap. 6 + Feast docs 0.40+. Duración estimada: 70 min.

Objetivo

Resolver el problema más caro de ML en producción —training/serving skew: que las features que ve el modelo en producción sean distintas a las que vio en training— centralizando definiciones de features en un feature store. Usar Feast para definir entidades + feature views, materializar al online store (Redis/SQLite), y servir features con get_online_features en <10 ms.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Explicar la diferencia entre offline store (parquet/BigQuery, para training) y online store (Redis/DynamoDB, para serving low-latency).
- Definir Entity, FeatureView, FileSource y registrar el repo con feast apply.
- Generar un training dataset point-in-time correct con get_historical_features (evita data leakage temporal).
- Materializar features (feast materialize-incremental) y consumirlas en serving con get_online_features.
- Reconocer cuándo NO usar feature store (ML con <5 features estables, equipo de 1, datos batch puro).

Temas

#	Tema	Por qué importa
1	Training/serving skew — el bug más caro de	Modelo bueno en offline, malo en producció
2	Offline vs online store	Latencia distinta, datos idénticos (en teo
3	Entity + FeatureView + FileSource	Modelo de datos de Feast.
4	Point-in-time joins	Mismo timestamp para feature y label → sin
5	materialize / materialize-incremental	Offline → online en batch programado.
6	Feast vs construir uno propio	Cuándo justifica la dependencia.

Definiciones y características

- Feature store: sistema que (1) define features una vez y las sirve en training + producción con garantía de equivalencia, (2) provee point-in-time correctness para evitar leakage, (3) baja latencia en serving.
- Training/serving skew: la causa #1 de modelos que rinden bien en CV y mal en producción. Surge cuando el pipeline de feature engineering offline y online divergen (lenguajes distintos, librerías distintas, bugs distintos).
- Offline store: dónde viven los datos históricos para training. Feast soporta File (parquet local), BigQuery, Snowflake, Redshift, Spark.
- Online store: dónde viven los valores actuales para serving low-latency. Soporta SQLite (dev), Redis, DynamoDB, Bigtable, Postgres.
- Entity: dimensión de negocio sobre la que se computan features (ej. user_id, driver_id, product_sku).

Tiene un nombre y un tipo (String, Int64).
- FeatureView: agrupa features que (1) vienen de la misma fuente y (2) se computan para la misma entidad. Define ttl (cuánto vive el feature antes de ser stale).
- Point-in-time join: dado (entity_id, event_timestamp) para cada fila del training set, Feast devuelve el valor del feature vigente en ese timestamp — no el más reciente. Evita meter información del futuro en training.
- Materialize: copia features de offline → online. materialize-incremental solo lo que cambió desde la última corrida.

Dataset / recursos

- Dataset: drivers de un servicio tipo Uber — driver_id, event_timestamp, conv_rate, acc_rate, avg_daily_trips. Generado sintéticamente en el notebook.
- Offline store: parquet local en data/.
- Online store: SQLite local en data/online_store.db.
- Librerías: feast>=0.40, pandas, pyarrow.

Ejercicios

1. Setup mínimo: feast init driver_repo. Inspeccioná feature_store.yaml, example_repo.py. Corré feast apply y verificá feast feature-views list.
2. Training dataset histórico: armá un entity_df con driver_id y event_timestamp para 5 momentos distintos del día. Pedile a Feast el feature driver_hourly_stats:conv_rate con get_historical_features. Confirmá manualmente que el valor devuelto es el último anterior al timestamp pedido (no el futuro).
3. Materialización + serving: feast materialize-incremental $(date +%Y-%m-%d). Después: store.get_online_features(features=['driver_hourly_stats:conv_rate'], entity_rows=[{'driver_id': 1001}]).to_dict(). Medí latencia con %timeit (debería ser <2 ms).
4. TTL en acción: configurá ttl=timedelta(days=1). Materializá datos viejos de 3 días atrás. Pedí features online → debería devolver None (porque expiró). Cambiá ttl=timedelta(days=7) y reintentá.
5. Skew check: comparé el feature offline (parquet) y el online (SQLite) para el mismo driver_id. Si difieren después de materialize, hay un bug.

Homework verificable

Repo Feast con:

1. Una Entity customer_id y una segunda merchant_id.
2. Tres FeatureView: customer_stats (avg_purchase, n_purchases_7d), merchant_stats (avg_rating, n_disputes_30d), transaction_features (amount, hour_of_day).
3. Un script build_training.py que arma el training set point-in-time correct para una tarea de fraud detection.
4. Un script serve.py que, dado un (customer_id, merchant_id), devuelve el vector de features listo para el modelo.
5. README del repo Feast con instrucciones para feast apply + materialize + serve.

Criterio de aceptación: el vector de features que devuelve serve.py para un (customer_id, merchant_id, timestamp) coincide exactamente con la fila correspondiente del training set generado por build_training.py (mismo customer/merchant/timestamp).

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
get_historical_features devuelve NaN para	El event_timestamp del entity_df está fuer
get_online_features devuelve None aunque a	TTL expirado, o materializaste hasta now()
feast apply falla con No module named exam	El cwd no es el feature_repo/. Feast carga
Training tarda muchísimo con offline=BigQu	Feast no agrega filtros por timestamp en e
Mi modelo en producción usa features disti	Probablemente tu pipeline de transformació

Preguntas frecuentes

¿Necesito un feature store?

No siempre. Si: ≥3 modelos comparten features, equipo ≥5 personas, requerís serving <50 ms, o tenés bug recurrente de training/serving skew → sí. Si: 1 modelo, 1 persona, batch scoring nocturno → es overkill. Empezá sin feature store y migrá cuando duela.

¿Feast vs Tecton vs Hopsworks?

Feast es open-source, self-hosted, no cobra. No hace compute (vos generás los parquets/tablas). Tecton y Hopsworks son SaaS managed que incluyen compute (definís transformaciones SQL/Python y ellos las corren). Para empezar: Feast. Para empresas grandes con presupuesto y muchas features: Tecton.

¿Feast hace feature engineering?

No por default. Feast sirve features — no las computa. Las OnDemandFeatureView (request-time) y las Stream Feature Views (con Spark) le agregan compute, pero el patrón principal es: vos generás parquet, Feast lo sirve.

¿Por qué point-in-time joins son tan importantes?

Porque la alternativa naive (LEFT JOIN ... ON entity_id) trae el valor más reciente del feature — que pudo ocurrir después del label. Ejemplo: predecís churn del usuario X el 1 de marzo, pero la feature n_logins_7d la tomás de hoy → metés información del futuro en training, y el modelo "predice" perfecto en CV y rompe en prod.

¿SQLite online store sirve para producción?

No: 1 escritor, no es multi-host. Para producción usá Redis (más común), DynamoDB (si ya estás en AWS), o Bigtable (si estás en GCP). SQLite es para desarrollo local y tests.

¿Cómo testear que no hay skew?

Job de comparación periódica: tomar N entities random, computar features offline (mismo código que el training) y online (vía Feast), assertear igualdad con tolerancia. Si falla → alerta. Es el equivalente de "smoke test" para feature stores.

Referencias

- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 6 — sección Feature Engineering y Feature Store.
- Feast docs — empezar por Quickstart.

- Point-in-time joins explained (Feast blog) — el por qué con ejemplos.
- Tecton vs Feast (2024 comparison) — vendor-biased, leer con criterio.
- MLOps at Reasonable Scale (Tagliabue, 2022) — cuándo NO usar feature store.

Siguiente clase

Clase 197 — CI/CD para ML con GitHub Actions

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import os, shutil, tempfile
from pathlib import Path
from datetime import datetime, timedelta
import pandas as pd, numpy as np

WORK = Path(tempfile.gettempdir()) / 'feast_demo'
if WORK.exists(): shutil.rmtree(WORK)
REPO = WORK / 'feature_repo'
(REPO / 'data').mkdir(parents=True)
os.chdir(WORK)
print('cwd:', Path.cwd())
```

Archivos complementarios

- notebook.ipynb