UT5

Machine Learning Operations

Fundamentos del Aprendizaje Automático

Profesor: Ing. Juan Francisco Kurucz

juan.kuruczsosa@ucu.edu.uy

Production ML Systems, MLOps y deuda técnica

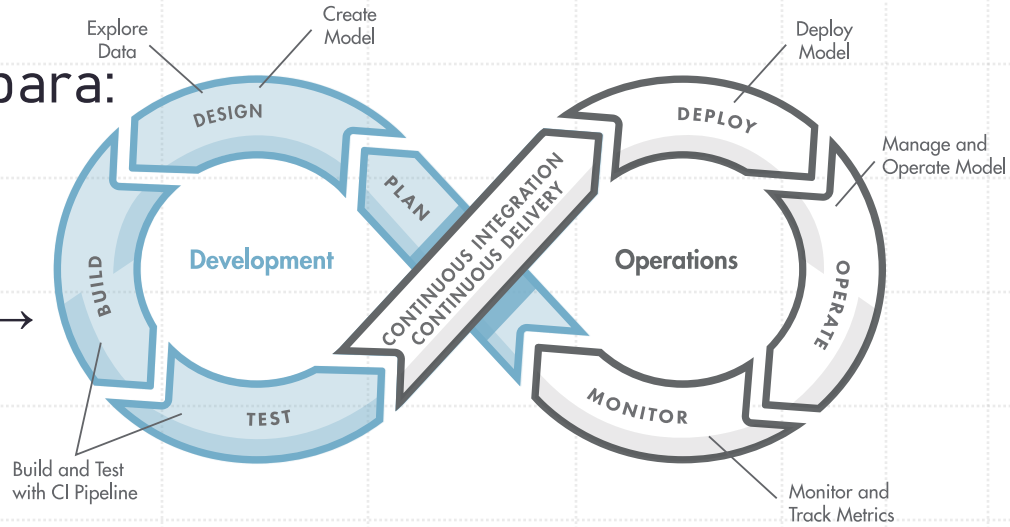
- Hoy: ¿qué pasa después de entrenar un modelo en el notebook?
- Introducción a **MLOps**: procesos, personas y herramientas.
- Veremos:
  - Por qué el modelo es solo una pequeña parte del sistema.
  - Conceptos de **CI/CD**, entrenamiento continuo y monitoreo de modelos.
  - Riesgos de **deuda técnica** en sistemas de ML.

¿Por qué no alcanza con tener “0.95 de accuracy”?

- En el notebook:
  - Datos limpios, bien armados.
  - Un solo dataset, sin cambios en el tiempo.
  - Métrica bonita en el test set 😄.
- En producción:
  - Los **datos cambian** (nuevos usuarios, nuevos formatos, nuevas reglas de negocio).
  - Nuevas versiones de la app / backend cambian el flujo de información.
  - Hay **latencia**, **costos de cómputo**, fallas de red, permisos, etc.
- Idea clave:
  - El modelo es una **pieza más** dentro de un sistema mucho más grande.
  - Sin procesos y automatización, el modelo “se pudre” rápido.

¿Qué es MLOps?

- Inspirado en **DevOps**, pero aplicado a sistemas de ML.
- Conjunto de **prácticas, procesos y herramientas** para:
  - Desarrollar, entrenar y desplegar modelos de forma **repetible y confiable**.
  - Automatizar el ciclo de vida completo de ML: datos → entrenamiento → deploy → monitoreo → retraining.
- Objetivo central:
  - Reducir fricción entre **investigación (data science)** y **producción (ingeniería)**.
  - Evitar “modelos cajón” que nunca llegan a producción o se rompen al primer cambio.



El modelo es solo el 5% del sistema

- En un sistema real de ML también tenemos:
  - **Ingesta de datos** (logs, eventos, bases).
  - **Validación y limpieza** de datos.
  - **Feature engineering** y pipelines de features.
  - **Infraestructura de entrenamiento** (jobs, GPUs, colas).
  - **Infraestructura de servicio** (APIs, endpoints, escalado).
  - **Monitoreo y alertas**.
- En la práctica:
  - El código del modelo suele ser una **fracción pequeña** del código total.
- Mensaje para estudiantes:
  - MLOps es aprender a diseñar y cuidar **todo el ecosistema alrededor del modelo**, no solo la red neuronal / algoritmo.

Ciclo de vida de un modelo en producción

- Fases típicas:
  1. Entender el problema y definir métricas.
  2. Recolectar y preparar datos.
  3. Entrenar y tunear modelos.
  4. Evaluar vs. métricas de negocio y técnicas.
  5. Desplegar modelo a producción.
  6. Monitorear datos, métricas, errores.
  7. **Reentrenar** y volver a desplegar (ciclo continuo).
- Lo importante:
  1. Pasar de un ciclo **manual y ad-hoc** a uno **automatizado y reproducible** (pipelines).
  2. Cada paso idealmente versionado y testeado.

Componentes clave de MLOps

- **Pipelines de datos (ETL/ELT):**
  - Extraer, transformar, cargar datos de forma confiable.
- **Pipelines de entrenamiento:**
  - Jobs que entrenan modelos usando datos actualizados.
- **Pipelines de validación:**
  - Chequear calidad de datos y performance del modelo antes de promoverlo.
- **Pipelines de deployment (CI/CD):**
  - Empaquetar y desplegar modelos / servicios de inferencia.
- **Monitoreo y alertas:**
  - Controlar el comportamiento en producción (latencia, errores, drift).
  - MLOps integra todo esto en un **flujo automático**, no en scripts sueltos.

CI/CD adaptado a ML

- **CI (Continuous Integration):**
  - Tests de código: unidades, integración.
  - Tests de datos: schemas, rangos, valores faltantes.
  - Tests del modelo: sanidad básica (métricas mínimas, outputs válidos).
- **CD (Continuous Delivery/Deployment):**
  - Automatizar despliegue de:
    - Código de pipelines (orquestadores).
    - Servicios de inferencia (APIs / endpoints).
  - Estrategias de rollout:
    - Canary, A/B tests, blue/green.
- Diferencia importante vs DevOps clásico:
  - Además de código, hay que gestionar **versionado de datos y modelos**.

Entrenamiento continuo (Continuous Training)

- Idea:
  1. El modelo se **reentrena de forma regular** cuando:
     - Llegan datos nuevos.
     - Cambian las distribuciones.
     - Cambia una regla de negocio.
- Pipeline típico de CT:
  1. Detectar evento (nuevos datos / tiempo).
  2. Ejecutar pipeline de entrenamiento.
  3. Evaluar contra modelo actual.
  4. Solo promover si **mejora** (gate de métricas).
- Beneficio:
  - El modelo se mantiene alineado con la realidad sin hacerlo “a mano”.
- Riesgo:
  - Si no hay buenos tests/monitoreo, podés **romper producción más rápido**.

Monitoreo de modelos en producción

- ¿Qué monitoreamos?
  - **Métricas de negocio**: conversión, engagement, churn, etc.
  - **Métricas del modelo**: accuracy, recall, calibration (donde se pueda medir).
  - **Métricas de datos**:
    - *Input drift*: cambian las características de los datos de entrada.
    - *Prediction drift*: cambian las distribuciones de las predicciones.
  - **Latencia y errores** del servicio.
- Ejemplo (Vertex AI / plataformas similares):
  - Configurar "model monitoring" para:
    - Ver si cambian las distribuciones de features.
    - Disparar alertas cuando algo se sale de lo esperado.
- Sin monitoreo:
  - Tenés un modelo "a ciegas" que puede estar dañando el negocio sin que nadie se entere.

“Rules of ML”: priorizar el sistema sobre el modelo

- **Regla 1:** Empezá simple.
  - Primero un modelo básico → validar pipeline y datos.
- **Regla 2:** Invertí en el **pipeline**, no en hiper-optimizar el modelo al principio.
  - La mayoría de las mejoras vienen de mejores datos / features.
- **Regla 3:** Evitá *training-serving skew*.
  - Usar mismas transformaciones y features en entrenamiento y producción.
  - Loguear lo que ve el modelo en producción.
- **Regla 4:** No tires datos “viejos” sin pensar.
  - Podés usar muestreo en vez de borrar, o al menos entender qué se pierde.
- **Regla 5:** Medí todo desde el día 1.
  - Métricas y logs pensados desde el diseño.

“Hidden Technical Debt” en sistemas de ML

- Idea central (Sculley et al., NIPS 2015):
  - Los sistemas de ML acumulan **deuda técnica oculta** más rápido que el software tradicional.
- Fuentes típicas de deuda:
  - **Glue code** y “pipeline jungle”:
    - Muchos scripts, parches y conexiones frágiles.
  - **Dependencias de datos**:
    - Cambia una columna y rompe todo el pipeline.
  - **Feedback loops no controlados**:
    - El modelo afecta los datos futuros (ej: sistemas de recomendación).
  - **Configuraciones y parámetros escondidos**:
    - Es difícil reproducir resultados o entender qué versión está en producción.
- Rol de MLOps:
  - Hacer explícito lo que antes era “magia”.
  - Estandarizar pipelines, versionar modelos/datos, poner tests y monitoreo.