Operaciones de Machine Learning

Proyecto Final
Nivel 4

Pontificia Universidad Javeriana
Cristian Javier Diaz Alvarez

13 de noviembre de 2025

1. Descripción

Este trabajo busca evaluar entendimiento, aplicación y explicación del ciclo de vida de un proyecto de aprendizaje de maquina. Se espera ver aplicados conceptos vistos en el desarrollo del curso, procesamiento de datos mediante pipeline completos que permitan realizar entrenamiento continuo de modelos. Este debe ser un proceso automatizado que permita identificar cambios en los datos que permitan de manera automática realizar nuevos entrenamientos mediante el uso de un orquestador. Cada modelo entrenado debe ser registrado para su posible uso. El modelo que presente mejor desempeño ante las métricas definidas, debe ser usado en un proceso de inferencia mediante una API que estará en un contenedor, el cual su imagen debe crearse y publicarse de manera automática. El objetivo es desplegar todas las herramientas necesarias para ejecutar el procesamiento de datos y entrenamiento de manera programada. Finalmente se espera un análisis del mejor modelo en cada etapa y una explicación de a que se debe el cambio en el modelo, que cambió para que fuera necesario un nuevo entrenamiento.

- Usando AirFlow cree los DAGs que le permitan recolectar, procesar y almacenar los datos.
- Para el registro de experimentos y modelos utilice MLflow
- Usando FastAPI cree un API que consuma el mejor modelo
- Cree una interfaz gráfica usando Streamlit que permita realizar inferencia.
- Utilice Github Actions para construir y publicar las imágenes de contenedores.
- Usando SHAP realice interpretación de los modelos desplegados y cambios de los mismos.
- Integre Argo CD para el despliegue automático en kubernetes.

Todos los servicios/componentes de este proyecto deben estar cada uno en su propio contenedor, no es admitido que las instancias de los contenedores sean construidas en la máquina de despliegue, si requiere construir una imagen de contenedor debe crear un workflow en Github Actions que realice esta tarea, teniendo como fin almacenar la imagen de contenedor en DockerHub, lo que permitirá instanciar la imagen directamente en kubernetes.

La entrega consiste en código fuente en repositorio publico, workflows en Github Actions funcionales, despliegue del sistema sobre la maquina proporcionada mediante Argo-cd, seguimiento de experimentos mediante MLflow con bucket y base de datos. El proceso de inferencia debe tomar siempre el modelo definido en MLflow como producción, sin cambios en código. La recolección, procesamiento, almacenamiento de datos y entrenamiento de modelos debe realizarse usando AirFlow. Cada nuevo entrenamiento después del creado con la linea base de los datos (primera petición) debe estar acompañado de una explicación de por qué se da el entrenamiento más allá de un factor de periodicidad o cantidad de datos nuevos.

Como sustentación del proyecto, se debe entregar un vídeo en un canal de YouTube con una duración no mayor a 10 minutos. En donde explique, organización del proyecto, arquitectura y conexiones entre componentes, procesamiento y experimentación realizada, posteriormente mostrar/usar interfaz gráfica que permite realizar inferencia y cambios entre versiones de modelos acompañados de su respectiva explicación. No olvide mostrar la ejecución de los workflow de Github Actions.

2. Descripción del dataset

Los datos fueron recopilados de realtor un sitio web de listados de bienes raíces operado por Move, Inc., filial de News Corp, y con sede en Santa Clara, California. Es el segundo sitio web de listados de bienes raíces más visitado en los Estados Unidos en 2024, con más de 100 millones de usuarios activos mensuales.

	Variables	**Descripción**
1	brokered_by	agencia/corredor codificado categóricamente
2	status	estado de la vivienda: a. lista para la venta o b. lista para construir
3	price	precio de la vivienda, es el precio de cotización actual o el precio de venta reciente si la casa se vendió recientemente
4	bed	Número de camas
5	bath	Número de baños
6	acre_lot	Tamaño del terreno/Propiedad en acres
7	street	dirección callejera codificada categóricamente
8	city	nombre de la ciudad
9	state	nombre del estado
10	zip_code	código postal de la zona
11	house_size	área de la casa/tamaño/espacio habitable en pies cuadrados
12	prev_sold_date	Fecha de venta anterior

Tabla 1: Descripción de variables.

El objetivo de este dataset es determinar el precio de una propiedad teniendo en cuenta el resto de características de la misma. Este dataset será entregado por partes y debe ser recolectado para ir generando una vista completa del problema. A cada nuevo subconjunto de datos obtenido se debe determinar si es necesario entrenar un nuevo modelo o no.

2.1. Cargar el Dataset

Los datos serán obtenidos a través de una API externa expuesta en la máquina virtual asignada al profesor alojada en la dirección **IP http://10.43.100.103:8000**. Esta API proporcionará un subconjunto de datos diferente en cada petición. Los estudiantes deberán implementar un mecanismo para recolectar estos datos usando Airflow y utilizarlos para entrenar un modelo y registrarlo en el entorno de MLflow. Cada subconjunto de datos nuevos es insumo para realizar un nuevo entrenamiento; para esto, se debe evaluar en términos de datos si se debe entrenar un nuevo modelo y si este modelo resultante debe ser enviado a producción o no.

Debido a la cantidad de datos que entrega la API, se recomienda no usar la interfaz gráfica de la misma, pues no podrá visualizar los datos; úsela como referencia para realizar los llamados a la API. Cada una de las peticiones tarda tiempo diferente, contiene cantidades y orígenes de datos diferentes. Tenga en cuenta esto al momento de realizar peticiones. Es importante resaltar que existe un método para reiniciar el conteo de peticiones según el grupo; este conteo está directamente relacionado con los datos que se entregan; en este proyecto no hay un grado de aleatoriedad en los datos, por lo tanto, la secuencia de datos es la misma en relación al conteo de peticiones.

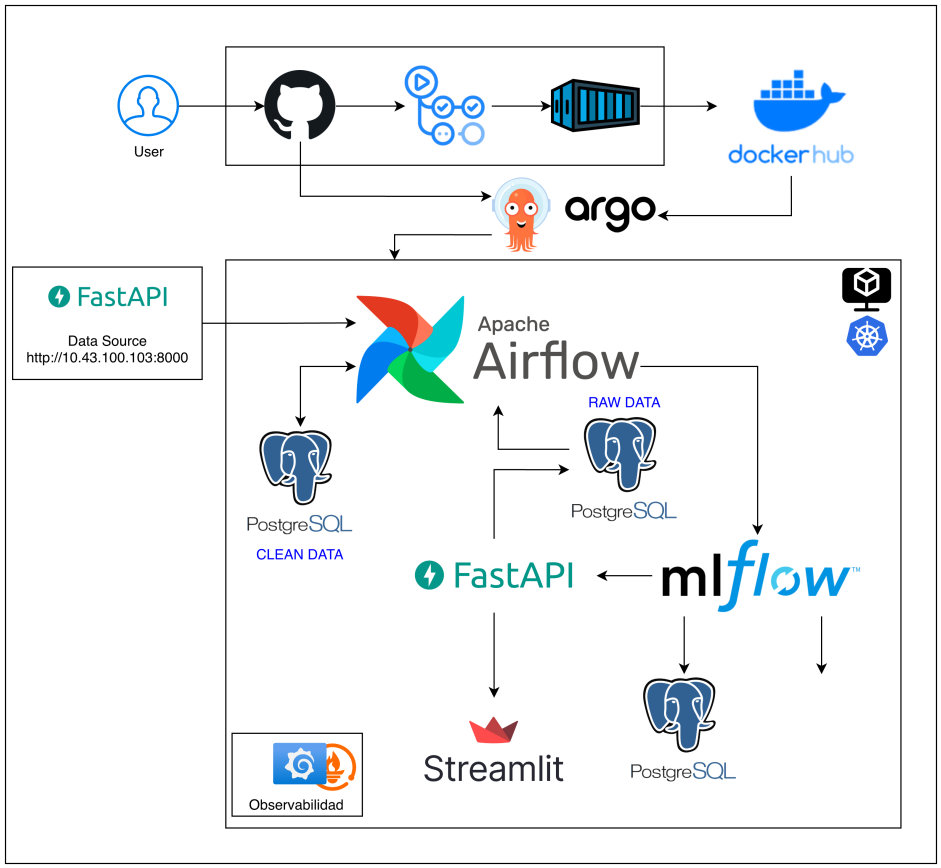

Figura 1: Arquitectura de Referencia

Si presenta inconvenientes con la API, informe al grupo completo en Teams para encontrar la causa y solución lo antes posible. El código de esta API no será publicado, pues este contiene información que se espera sea obtenida mediante análisis.

3. Arquitectura

Este proyecto busca exponer al estudiante a una arquitectura típica en equipos que desarrollan/aplican inteligencia artificial. Estos equipos se enfrentan a tareas relacionadas con la recolección de datos, procesamiento de información, entrenamiento y puesta en producción de modelos de Inteligencia Artificial. Para dar un adecuado manejo de este escenario, se propone una arquitectura que permite orquestar todo este proceso mediante AirFlow, realizar seguimiento de experimentos usando MLflow y distintos componentes de almacenamiento de datos crudos, información procesada, artefactos y meta-datos. Finalmente, se tienen componentes que permiten realizar integración y despliegue continuo. Como puede observar, estos componentes son el acumulado del desarrollo del curso, por lo que su integración no debe suponer un nuevo reto.

El seguimiento de experimentos de Inteligencia Artificial requiere el almacenamiento de dos conjuntos de datos: Artefactos y Metadatos. Los artefactos hacen referencia a los elementos resultantes del proceso de entrenamiento, métricas, código, imágenes, modelo, entre otros. Los metadatos son la información relativa a la ejecución, parámetros, tiempos, conjuntos de ejecución, entre otros. Para almacenar estos datos, se requieren dos artefactos, una base de datos para los metadatos y un sistema de archivos para los artefactos; estos dos elementos dan soporte a MLflow como herramienta de seguimiento. Para la ejecución de experimentos, se requiere otro sistema en donde se van a ejecutar los entrenamientos que se registrarán en MLflow. Por último, el sistema de archivos, también funciona

como registro de modelos, por lo cual, es posible mantener los modelos de producción o desarrollo disponibles para ser usados por las aplicaciones desplegadas.

El proceso de orquestación se realiza mediante AirFlow, esta herramienta ya contiene bases de datos, caché, logs, plugins, todo lo necesario para funcionar, mediante la creación de DAGs se definen los flujos de ejecución. Para el proceso de experimentación se deben presentar los argumentos por los cuales se modifican y separan los datos, se seleccionan los modelos y se definen métricas. Para entrenar el modelo se debe crear el pipeline de procesamiento y entrenamiento que debe ejecutar AirFlow, para los dos escenarios anteriores se debe registrar en MLflow la experimentación realizada, adicionalmente registrar los modelos. Por último, para el proceso de inferencia se debe utilizar los modelos registrados en MLflow, los modelos deben estar publicados y asociados al stage de producción, lo que permite cambiar el modelo seleccionado, sin realizar cambios en el código desplegado.

Todas las imágenes de contenedores que se usen en el proceso de despliegue deben obtenerse de un registro de contenedores (DockerHub), pueden ser versiones oficiales de las herramientas o creadas por el equipo de trabajo. Es importante resaltar que el código fuente de todo el proyecto debe estar en Github, esto incluye los workflow de Github Actions, que se encargarán de tomar los archivos de docker (Dockerfile) correspondientes a cada servicio, los construirán y publicarán en Dockerhub. Así en la máquina virtual destinada para el desarrollo del curso se podrán consumir las imágenes de todo el proyecto en sus correspondientes manifiestos.

Como observará en la descripción de componentes, en la mayoría de casos solo se especifica, su función y relación. Es trabajo del estudiante definir como va a implementar el sistema, recuerde, este ejercicio busca acercarlo a un escenario de producción. Es decir, la experimentación genera modelos y despliegues de clientes consumen esos modelos. Es importante la disponibilidad de cada elemento para garantizar que el equipo de científicos de datos pueda registrar sus modelos y el despliegue de los sistemas de los clientes estén disponibles.

4. Componentes

Para construir este sistema, cada componente debe tener una función clara y definida, así mismo las interacciones entre los componentes deben ser explicitas, es importante prever escenarios de fallo y tomar medidas o elevar los errores necesarios. Cada componente debe existir dentro de un contenedor y estar interconectado en red o archivos según la necesidad, es importante recordar que la comunicación entre servicios debe ser ligera. La elección técnica de cada componente es trabajo del estudiante, basada en la necesidad del microservicio. Cada componente debe ser justificado bajo un criterio técnico o, cuanto menos, lógico. No existe una única respuesta correcta; la argumentación define la calificación.

Esta sección describe los componentes que los estudiantes deben construir/instanciar, es importante resaltar que existen servicios adicionales que no se describen pues no están bajo control de los estudiantes, son herramientas que deben usar y fueron vistas en clase.

4.1. API Data Source

Este componente es una API que expone los datos para ser usados en todo el flujo. Cada una de las peticiones que se hagan estarán asociadas al grupo asignado en el curso y al día de la clase. Cada respuesta que entrega la API pretende simular un nuevo punto de tiempo, un nuevo grupo de información recolectada en un ambiente productivo. Al finalizar la recolección de datos, la respuesta será un error, con un mensaje indicando que se recolectaron todos los datos. No realice peticiones usando la interfaz gráfica de Swagger en el endpoint /docs, pues la cantidad de datos enviados no es posible verlos con esta herramienta.

4.2. MLflow

Este servidor debe estar funcionando constantemente; en caso de caída, debe iniciarse automáticamente. Debe tener conexión entre bucket y base de datos de metadata. Acá debe registrar la experimentación que realice buscando el mejor modelo para el conjunto de datos propuesto. El que considere como mejor versión, deberá estar marcado como modelo de producción. Se deben mantener las versiones entrenadas de los distintos grupos de información, no solo presentar el modelo definitivo.

4.3. AirFlow

Esta herramienta es la encargada de orquestar el procesamiento de datos, debe tomar los datos de la API la cual entregará a cada grupo (indicando su número) una serie de datos para ser analizados y que permitirán realizar entrenamiento si se considera necesario. Tenga en cuenta que no se deben realizar todas las consultas posibles a la API y posteriormente entrenar un modelo, por cada petición se debe realizar el proceso completo de entrenamiento y publicación en caso de ser necesario. Cada DAG en caso de ser más de uno, debe tener múltiples tareas. La información recolectada debe ser almacenada en la base de datos RAW DATA y aplicar el procesamiento definido para utilizar la información en proceso de entrenamiento, almacenando los resultados en CLEAN DATA. Adicionalmente, una vez se define un proceso de entrenamiento, debe ejecutar este proceso, registrando los resultados en MLflow. Asegúrese de generar un registro que permita entender lo sucedido en el DAG. Como sugerencia, registre en MLFLOW los componentes e información relevante para la toma de decisiones.

4.4. Sistema de Archivos - Bucket

Para el registro de artefactos, tal como se vio en clase, propone el uso de un bucket y acceder a el mediante el uso de boto3. Configure el contenedor o si lo desea use otro método de almacenamiento; puede ser con o sin proveedor cloud.

4.5. Base de Datos - Metadata

Esta base de datos contiene lo relativo a ejecuciones registradas en MLflow; a diferencia del escenario presentado en clase, debe crear una base de datos en un contenedor y no puede ser SQLite. Sugerencia: PostgreSQL.

4.6. Base de Datos - CLEAN DATA

Este componente contiene la información que se usará como insumo para la experimentación.

4.7. Base de Datos - RAW DATA

Este componente contiene la información sin modificaciones. En este componente se permite a los estudiantes proponer cómo se dará manejo a los datos. Es importante recordar que en el proceso de experimentación es óptimo poder reproducir los experimentos previamente ejecutados (aunque los modelos no sean deterministas).

4.8. Inferencia UI

Este componente permitirá usar el mejor modelo de la experimentación, es decir, el que esté configurado como 'Production' en MLflow mediante el consumo del componente de FastAPI. Para esto se debe crear un contenedor que permita consumir el modelo registrado en MLflow, solicitar al usuario datos para realizar inferencia y al presentar resultados, indicar cuál es el modelo que está usando. Para la creación de esta interfaz, no se espera un desarrollo web, se propone al estudiante revisar el framework "Streamlit", el cual está diseñado para crear interfaces para aplicaciones de inteligencia artificial o ciencia de datos en minutos. Para este proyecto final se debe generar un apartado de historial y explicabilidad, en donde se debe mostrar un registro de los modelos previamente entrenados, cuáles fueron puestos en producción y cuáles no; adicionalmente, cuál fue el criterio de rechazo (en los casos que aplique) y cuál es el desempeño en métricas de los modelos precios y actual.

4.9. Observabilidad

Este bloque es agnóstico al resto del sistema y sus conexiones serán determinadas por el estudiante; como mínimo, se espera que se recolecte información de la API que permite realizar inferencia al modelo de mejor desempeño. Este bloque está compuesto por dos servicios, Grafana para la visualización de datos y Prometheus para la recolección de métricas.

4.10. FastAPI

Este componente es el encargado de exponer el modelo entrenado, el cual debe estar almacenado en MLflow. Esta API debe consumir el modelo establecido con mejor desempeño, marcado con un TAG específico; no debe requerir cambios en caso de un nuevo entrenamiento. Es importante almacenar los datos nuevos de inferencia en la base de datos RAW DATA, pues estos son insumo vital en el ciclo de vida de un modelo.

4.11. GitHub Actions

Este componente es responsable del ciclo de Integración Continua (CI). Permite definir workflows automáticos que se ejecutan al detectar cambios en el código del repositorio. En el contexto de este proyecto, GitHub Actions debe ser usado para construir las imágenes de contenedores de cada componente (Airflow, FastAPI, Streamlit, MLflow, entre otros), validarlas y publicarlas en un registro de contenedores como DockerHub. Este proceso debe garantizar que toda actualización del código se refleje en una nueva versión de imagen, versionada y lista para ser desplegada posteriormente por Argo CD. La correcta definición de estos flujos asegura trazabilidad, consistencia y automatización del entorno.

4.12. Argo CD

Este componente permite implementar el ciclo de vida de despliegue continuo. Argo CD observa los manifiestos del sistema versionados en el repositorio Git y sincroniza automáticamente los cambios con el entorno de Kubernetes. Esto permite que, ante la generación de una nueva imagen o cambio en configuraciones de despliegue, se actualice automáticamente la API de inferencia u otros servicios.

Apartado opcional - Bono

Aunque el proyecto base no especifica de forma obligatoria que todos los componentes sean desplegados sobre Kubernetes, es aceptable que herramientas como *Airflow* y *MLflow* sean ejecutadas mediante *docker-compose*, siempre que cumplan con los requerimientos funcionales del sistema y se mantenga el principio de contenerización aislada.

No obstante, se otorgará un bono adicional a los estudiantes que desplieguen todo el sistema completamente sobre un entorno **Kubernetes**, utilizando Argo CD como herramienta de gestión de despliegue continuo. Esto implica que todos los servicios, incluyendo Airflow, MLflow, API de inferencia, interfaz gráfica y bases de datos, deben estar definidos mediante manifiestos y gestionados dentro del clúster.

Para el bono se debe usar HELM que es un gestor de paquetes para Kubernetes que permite definir, instalar y actualizar aplicaciones mediante plantillas llamadas *charts*, facilitando el despliegue reproducible y parametrizable de servicios complejos.

- Airflow debe estar sincronizado con un repositorio de git
- minIO debe crear automáticamente el bucket en caso de no existir

- Agregue un apartado de interpretabilidad usando shap en la interfaz gráfica, al menos para el modelo final despues de recibir todas las peticiones de la API

- Agregué los ConfigMap necesarios para tener el dashboard de grafana listo sin necesidad de cargarlo manualmente

Este despliegue completo representa una madurez superior en la adopción de prácticas de MLOps en producción, integrando orquestación, CI/CD y gestión declarativa.