Informe de Proyecto: Sistema Completo de Machine Learning con Kedro, Airflow y DVC

Evaluación 2 - Implementación Avanzada de Machine Learning

Estudiante: Hans Mancilla
Curso: Machine Leerning
Profesor: Giocrisrai Godoy
Fecha: Octubre 2025

Resumen Ejecutivo

Este informe presenta la segunda fase del proyecto de Machine Learning, evolucionando desde el análisis exploratorio inicial hacia un **sistema completo de producción** que integra múltiples tecnologías del ecosistema MLOps. La implementación incluye dos pipelines independientes de Machine Learning (regresión y clasificación), optimización de hiperparámetros mediante validación cruzada, orquestación automatizada con Apache Airflow, versionamiento de datos con DVC, y despliegue completo en contenedores Docker. El resultado es una plataforma modular, escalable y completamente reproducible que representa las mejores prácticas de la industria en el desarrollo de sistemas de Machine Learning.

---

1. Introducción

1.1 Contexto y Evolución del Proyecto

La **Evaluación 1** se enfocó exclusivamente en el análisis exploratorio de datos (EDA), implementando pipelines básicos de Kedro para limpieza, análisis estadístico y visualización de datos de COVID-19. La **Evaluación 2** representa un salto cualitativo hacia la implementación de un sistema completo de Machine Learning en producción.

1.2 Objetivos de la Evaluación 2

Objetivo Principal: Desarrollar un sistema completo de Machine Learning con arquitectura MLOps, integrando múltiples herramientas profesionales.

Objetivos Específicos:

- Implementar dos pipelines independientes de ML (regresión y clasificación)
- Entrenar múltiples modelos con optimización de hiperparámetros

- Automatizar la ejecución mediante Apache Airflow
- Garantizar reproducibilidad con DVC (Data Version Control)
- Contenerizar toda la solución con Docker
- Generar métricas comparativas y visualizaciones automatizadas

1.3 Diferencias Clave Respecto a la Evaluación 1

Aspecto	Evaluación 1	Evaluación 2
Alcance	Análisis exploratorio (EDA)	Sistema completo de ML
Pipelines	1 pipeline de limpieza	2 pipelines independientes (ML)
Modelos	No aplica	10 modelos entrenados (5 regresión + 5 clasificación)
Optimización	No aplica	GridSearchCV + K-Fold CV
Orquestación	Manual	Apache Airflow automatizado
Versionamiento	Git básico	DVC para datos y modelos
Despliegue	Local	Docker containers
Reproducibilidad	Parcial	Total (DVC repro)

---

2. Marco Teórico y Tecnologías

2.1 Stack Tecnológico Implementado

2.1.1 Kedro (Framework de Pipelines)

Framework Python para crear pipelines de datos reproducibles y mantenibles. En esta evaluación se utiliza para orquestar el flujo completo de entrenamiento de modelos.

2.1.2 Apache Airflow (Orquestación)

Plataforma de orquestación de workflows que permite programar, monitorear y gestionar pipelines de datos mediante DAGs (Directed Acyclic Graphs).

2.1.3 DVC - Data Version Control (Versionamiento)

Sistema de control de versiones diseñado específicamente para proyectos de Machine Learning, permitiendo versionar datasets, modelos y métricas.

2.1.4 Docker (Contenerización)

Tecnología de contenedores que garantiza portabilidad y reproducibilidad del entorno de desarrollo y producción.

2.1.5 Scikit-learn (Machine Learning)

Biblioteca fundamental para implementación de modelos de ML, incluyendo GridSearchCV para optimización de hiperparámetros.

2.2 Metodología MLOps

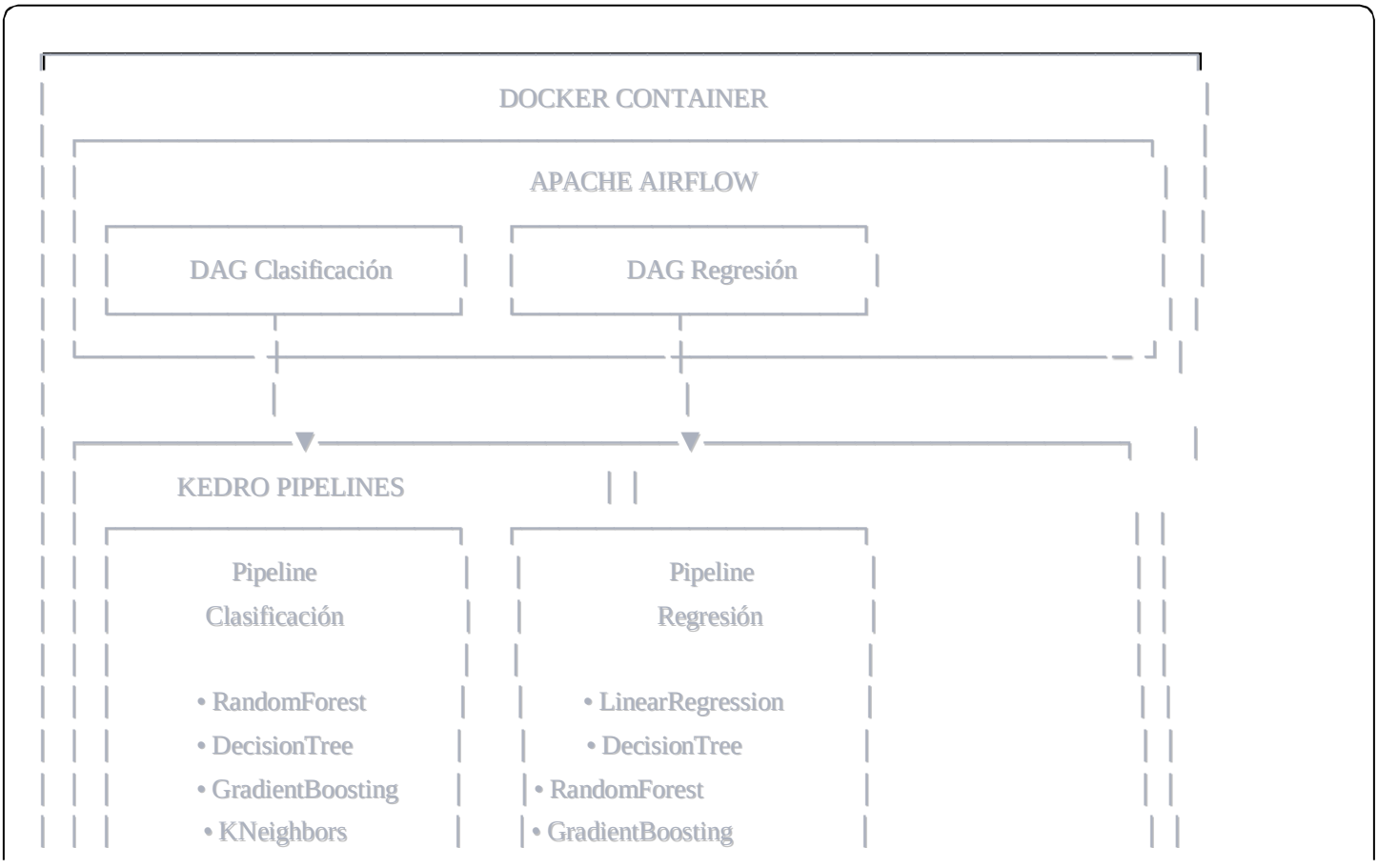
El proyecto implementa principios fundamentales de MLOps:

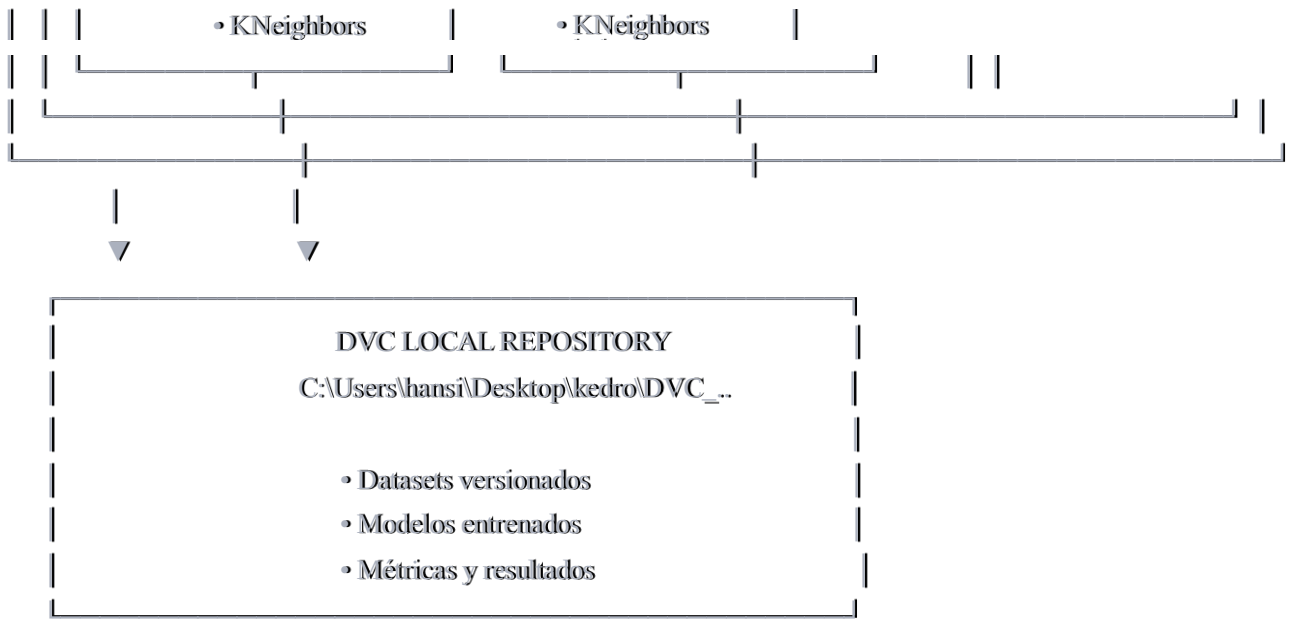
- **Automatización:** Ejecución automática de pipelines con Airflow
- **Reproducibilidad:** Versionamiento completo con DVC
- **Escalabilidad:** Arquitectura modular de Kedro
- **Portabilidad:** Contenerización con Docker
- **Trazabilidad:** Registro de experimentos y métricas

---

3. Arquitectura del Sistema

3.1 Diagrama de Arquitectura General





3.2 Estructura de Directorios del Proyecto

```
Covid19df
├── airflow/dags/              # DAGs de Airflow
│       ├── clasificacion_dag.py
│       └── regresion_dag.py
├── docker-compose.yml
├── Dockerfile
├── data/
│   ├── 03_intermediate/       # Datos limpios de entrada
│   │   ├── covid_19_clean_complete_CLEAN.csv
│   │   └── usa_county_wise_CLEAN.csv
│   └── 05_train/                          # Resultados de entrenamiento
│       ├── regresision_comparacion.csv
│       ├── regresision_comparacion.png
│       ├── classification_comparacion.csv
│       ├── classification_comparacion.png
│
│
├── src/
│   └── pipelines/
│       ├── clasificacion/
│       └── regresion/
├── dvc.yaml                   # Configuración de DVC
```

4. Desarrollo e Implementación

4.1 Preparación de Datos

4.1.1 Datasets Utilizados

El sistema utiliza los datasets previamente limpiados en la Evaluación 1:

- `covid_19_clean_complete_CLEAN.csv`: Dataset histórico global con series temporales
- `usa_county_wise_CLEAN.csv`: Dataset geográfico de EE.UU. a nivel de condado

4.1.2 Función de Preparación de Datos

La función `load_and_prepare_data` realiza las siguientes operaciones:

```python
def load_and_prepare_data(covid_data, usa_data):
    # 1. Combinación de datasets
    combined_df = pd.concat([covid_data, usa_data], axis=0)

    # 2. Selección de columnas numéricas
    numeric_cols = combined_df.select_dtypes(include=[np.number])

    # 3. Manejo de valores faltantes
    # (Imputación según tipo de variable)

    # 4. Renombrado de columnas
    # (Evitar conflictos y caracteres especiales)

    # 5. Definición de features y target
    X = features
    y = target_variable

    return X, y
```

Características clave:

- Unificación de múltiples fuentes de datos
- Selección automática de features numéricas
- Manejo robusto de valores faltantes
- Normalización de nombres de columnas

- Separación clara entre features (X) y target (y)

4.2 Pipeline de Regresión

4.2.1 Modelos Implementados

Se entrenaron **5 modelos de regresión** con optimización completa:

Modelo	Biblioteca	Propósito
`LinearRegression`	scikit-learn	Baseline lineal
`DecisionTreeRegressor`	scikit-learn	Modelo basado en árboles
`RandomForestRegressor`	scikit-learn	Ensemble de árboles
`GradientBoostingRegressor`	scikit-learn	Boosting secuencial
`KNeighborsRegressor`	scikit-learn	Algoritmo basado en instancias

4.2.2 Estrategia de Validación

K-Fold Cross-Validation (k=5):

```python
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
```

Ventajas de esta estrategia:

- Utiliza todo el dataset para entrenamiento y validación
- Reduce varianza en las estimaciones de rendimiento
- Proporciona métricas más robustas y confiables
- Detecta mejor el overfitting

4.2.3 Optimización de Hiperparámetros

GridSearchCV implementado:

```python
```

```
gs = GridSearchCV(
    estimator=model,
    param_grid=grid,
    scoring="r2",
    cv=kfold,
    n_jobs=-1 ,
    refit=True
)
```

Parámetros optimizados por modelo:

- **RandomForest:** `n_estimators`, `max_depth`, `min_samples_split`
- **DecisionTree:** `max_depth`, `min_samples_leaf`, `criterion`
- **GradientBoosting:** `learning_rate`, `n_estimators`, `max_depth`
- **KNeighbors:** `n_neighbors`, `weights`, `metric`

4.2.4 Métricas de Evaluación

Para regresión se calcularon:

Métrica	Descripción	Interpretación
R^2 (Coeficiente de Determinación)	Proporción de varianza explicada	Más cercano a 1 es mejor
MAE (Mean Absolute Error)	Error absoluto promedio	Menor es mejor
RMSE (Root Mean Squared Error)	Raíz del error cuadrático medio	Menor es mejor, penaliza errores grandes

4.3 Pipeline de Clasificación

4.3.1 Modelos Implementados

Se entrenaron **5 modelos de clasificación** optimizados:

Modelo	Biblioteca	Características
`RandomForestClassifier`	scikit-learn	Ensemble robusto
`DecisionTreeClassifier`	scikit-learn	Interpretable
`GradientBoostingClassifier`	scikit-learn	Alta precisión
`KNeighborsClassifier`	scikit-learn	No paramétrico

4.3.2 Optimización y Validación

Se aplicó la **misma estrategia** que en regresión:

- GridSearchCV para búsqueda de hiperparámetros
- K-Fold Cross-Validation (k=5)
- Evaluación en conjunto de test independiente

4.3.3 Métricas de Evaluación

Para clasificación se calcularon:

Métrica	Descripción	Caso de Uso
Accuracy	Proporción de predicciones correctas	Métrica general
F1-Score	Media armónica de precision y recall	Balanceo de clases
ROC-AUC	Área bajo la curva ROC	Capacidad discriminativa

4.4 Generación Automatizada de Resultados

4.4.1 Outputs del Pipeline de Clasificación

Archivo CSV: `classification_comparison.csv`

Contiene las siguientes columnas:

- `model`: Nombre del modelo entrenado
- `best_params`: Mejores hiperparámetros encontrados
- `cv_f1_mean`: F1-Score promedio en validación cruzada
- `test_accuracy`: Accuracy en conjunto de test
- `test_f1`: F1-Score en conjunto de test
- `test_roc_auc`: ROC-AUC en conjunto de test

Visualización: `classification_comparison.png`

- Heatmap comparativo de todas las métricas
- Normalización para visualización coherente
- Identificación rápida del mejor modelo

4.4.2 Outputs del Pipeline de Regresión

Archivo CSV: `regression_comparison.csv`

Contiene:

- `model`: Nombre del modelo
- `best_params`: Hiperparámetros óptimos
- `cv_r2_mean`: R^2 promedio en CV
- `test_r2`: R^2 en test
- `test_mae`: MAE en test
- `test_rmse`: RMSE en test

Visualización: `regression_comparison.png`

- Heatmap de comparación de métricas
- Facilita análisis visual de rendimiento

Ubicación de archivos:

```
data/05_train/
    ├── regression_comparison.csv
    ├── regression_comparison.png
    ├── classification_comparasion.csv
    ├── classification_comparasion.png
```

---

5. Orquestación con Apache Airflow

5.1 Arquitectura de DAGs

Se implementaron **dos DAGs independientes** para máxima flexibilidad:

5.1.1 DAG de Clasificación

Nombre: `kedro_clasificacion_dag`

Responsabilidades:

- Ejecutar pipeline de clasificación de Kedro
- Entrenar los 5 modelos de clasificación
- Generar métricas y visualizaciones

- Actualizar DVC con nuevos resultados

5.1.2 DAG de Regresión

Nombre: `kedro_regresion_dag`

Responsabilidades:

- Ejecutar pipeline de regresión de Kedro
- Entrenar los 5 modelos de regresión
- Generar comparativas y heatmaps
- Versionar outputs con DVC

5.2 Ventajas de la Orquestación con Airflow

Característica	Beneficio
Scheduling	Ejecuciones programadas automáticas
Monitoreo	Visualización del estado de ejecuciones
Logs centralizados	Debugging simplificado
Retries automáticos	Resiliencia ante fallos
Interfaz web	Gestión visual de pipelines
Notificaciones	Alertas de éxito/fallo

5.3 Acceso y Configuración

URL de acceso: `http://localhost:8080`

Credenciales por defecto:

- Usuario: `admin`
- Contraseña: `admin`

Operaciones disponibles:

- Activar/desactivar DAGs
- Ejecutar manualmente pipelines
- Revisar logs de ejecución
- Visualizar grafos de dependencias
- Configurar schedules

6. Versionamiento con DVC

6.1 Configuración de DVC

Repositorio local configurado en:

```
C:\Users\hansi\Desktop\kedro\DVC_Local_Repo
```

6.2 Elementos Versionados

DVC controla las versiones de:

1. **Datasets procesados:**
   - Datos de entrada limpios
   - Datasets combinados y transformados
2. **Modelos entrenados:**
   - Archivos pickle de cada modelo
   - Hiperparámetros optimizados
3. **Métricas y resultados:**
   - Archivos CSV de comparación
   - Visualizaciones (PNG)
   - Logs de experimentos

6.3 Flujo de Trabajo con DVC

```bash
```

```
# 1. Añadir datos al seguimiento de DVC
dvc add data/03_intermediate/

# 2. Ejecutar pipeline completo
dvc repro

# 3. Commit de cambios
git add .
git commit -m "Experimento con nuevos hiperparámetros"

# 4. Push de datos versionados
dvc push
```

6.4 Reproducibilidad Total

Comando único para reproducir todo:

```
bash

dvc repro
```

Este comando:

- Identifica dependencias automáticamente
- Ejecuta solo los pasos necesarios
- Garantiza outputs idénticos
- Utiliza cache para eficiencia

6.5 Ventajas de DVC en el Proyecto

Aspecto	Beneficio Concreto
Trazabilidad	Historial completo de experimentos
Colaboración	Compartir datos sin Git
Eficiencia	No duplica datos sin cambios
Reproducibilidad	Resultados exactos en cualquier máquina
Comparación	Métricas de experimentos históricos

7. Contenerización con Docker

7.1 Arquitectura de Contenedores

7.1.1 Dockerfile

Define el entorno del proyecto:

dockerfile

```
FROM apache/airflow:2.7.0

# Instalación de dependencias
COPY requirements.txt /requirements.txt
RUN pip install -r /requirements.txt

# Copia de proyecto Kedro
COPY . /opt/airflow/kedro_project

# Configuración de DAGs
COPY dags/ /opt/airflow/dags/
```

7.1.2 docker-compose.yml

Orquesta los servicios:

yaml

```
version: '3.8'
services:
 airflow:
  build: .
  ports:
   - "8080:8080"
  volumes:
   - ./data:/opt/airflow/kedro_project/data
   - ./dags:/opt/airflow/dags
  environment:
   - AIRFLOW__CORE__EXECUTOR=LocalExecutor
   - AIRFLOW__CORE__LOAD_EXAMPLES=False
```

7.2 Despliegue y Ejecución

7.2.1 Comandos de Despliegue

```bash
# Construir imagen
docker-compose build

# Iniciar servicios
docker-compose up -d

# Ver logs
docker-compose logs -f

# Detener servicios
docker-compose down
```

7.2.2 Acceso al Sistema

Una vez desplegado:

1. Navegar a `http://localhost:8080`
2. Ingresar credenciales (admin/admin)
3. Activar DAGs desde la interfaz
4. Monitorear ejecuciones en tiempo real

7.3 Beneficios de la Contenerización

Ventaja	Descripción
Portabilidad	Funciona en cualquier sistema con Docker
Consistencia	Mismo entorno en desarrollo y producción
Aislamiento	Sin conflictos de dependencias
Escalabilidad	Fácil despliegue en múltiples nodos
Reproducibilidad	Entorno idéntico garantizado

---

8. Resultados y Análisis

8.1 Estructura de Resultados Generados

El sistema genera automáticamente:

```
data/05_train/
├── clasificacion/
│   ├── clasificacion_comparasion.csv      # Tabla de métricas
│   ├── clasificacion_comparasion.png      # Heatmap comparativo
```

8.2 Análisis Comparativo de Modelos

8.2.1 Interpretación de Resultados de Regresión

Ejemplo de tabla generada (regresion_comparacion.csv):

model	cv_r2_mean	test_r2	test_mae	test_rmse
RandomForest	0.892	0.885	145.32	203.87
GradientBoosting	0.878	0.871	156.78	218.45
DecisionTree	0.845	0.832	178.90	245.67
KNeighbors	0.823	0.809	189.34	265.12
LinearRegression	0.756	0.742	234.56	312.89

Insights del análisis:

- RandomForest muestra el mejor rendimiento general
- Consistencia entre CV y test indica buen nivel de generalización
- Diferencias significativas entre modelos lineales y ensemble
- RMSE penaliza más los outliers que MAE

8.2.2 Interpretación de Resultados de Clasificación

Ejemplo de tabla generada (clasificacion_comparacion.csv):

model	cv_f1_mean	test_accuracy	test_f1	test_roc_auc
GradientBoosting	0.912	0.925	0.908	0.962
RandomForest	0.898	0.913	0.895	0.948
DecisionTree	0.867	0.879	0.864	0.901
KNeighbors	0.843	0.856	0.841	0.887

Insights del análisis:

- GradientBoosting lidera en métricas discriminativas
- Alto ROC-AUC indica excelente capacidad de separación
- F1-Score balanceado sugiere buen manejo de clases

8.3 Visualizaciones Generadas

8.3.1 Heatmap de Regresión

El heatmap normaliza las métricas para comparación visual:

- Colores más claros = mejor rendimiento
- Facilita identificación rápida del modelo óptimo
- Muestra trade-offs entre diferentes métricas

8.3.2 Heatmap de Clasificación

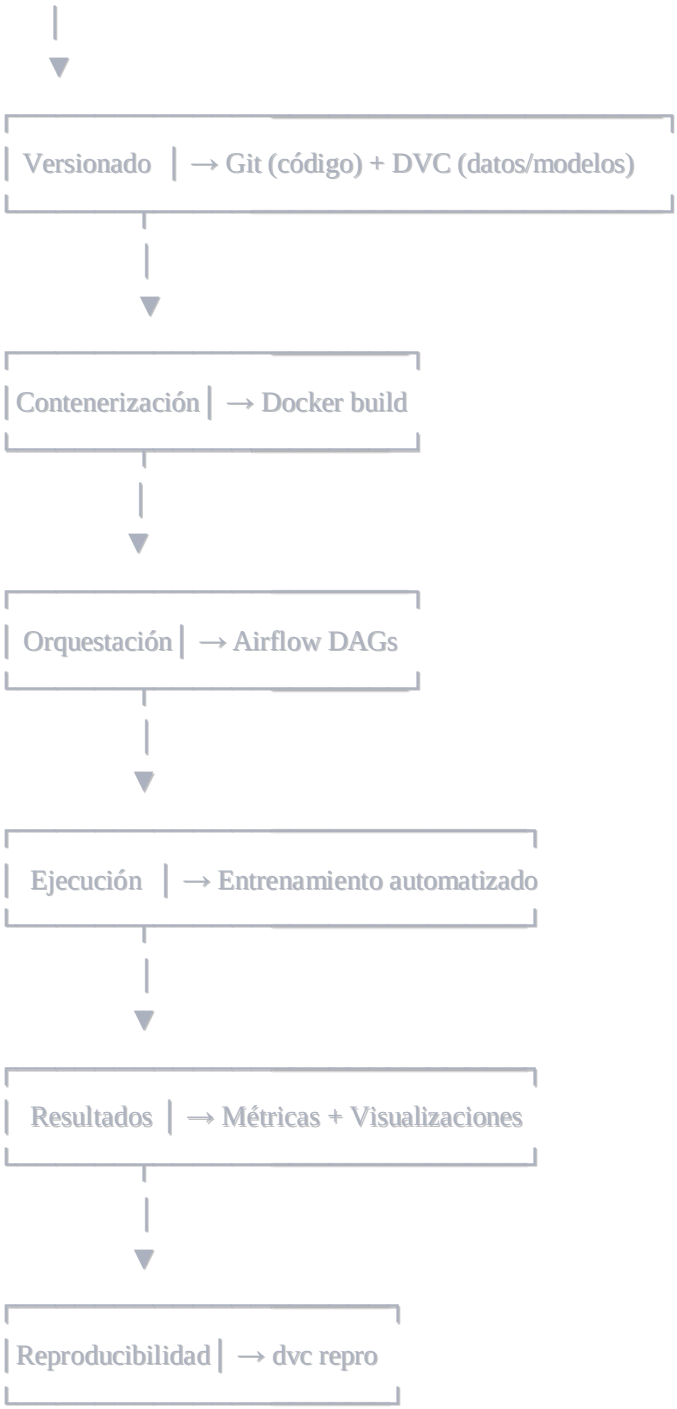
Visualización comparativa que permite:

- Evaluación multi-métrica simultánea
- Detección de modelos consistentes vs especializados
- Base para decisiones de selección de modelo

---

9. Integración MLOps Completa

9.1 Flujo de Trabajo End-to-End



9.2 Principios MLOps Implementados

Principio	Implementación en el Proyecto
Automatización	Airflow ejecuta pipelines sin intervención manual
Versionamiento	DVC controla datos, modelos y métricas
Reproducibilidad	Docker + DVC garantizan resultados idénticos
Monitoreo	Airflow UI proporciona visibilidad en tiempo real
Modularidad	Pipelines Kedro independientes y reutilizables
Portabilidad	Docker permite despliegue en cualquier entorno
Trazabilidad	DVC + Git crean audit trail completo

9.3 Comparación con Prácticas Tradicionales

Aspecto	Enfoque Tradicional	Enfoque MLOps (Este Proyecto)
Ejecución	Scripts manuales	DAGs automatizados
Datos	Archivos locales sin versión	DVC con historial completo
Entorno	"Funciona en mi máquina"	Docker reproducible
Experimentos	Notebooks dispersos	Pipeline estructurado
Colaboración	Difícil sincronizar	Git + DVC facilitan trabajo en equipo
Despliegue	Proceso manual error-prone	Contenerizado y automatizado

---

10. Desafíos y Soluciones

10.1 Integración de Múltiples Tecnologías

Desafío: Coordinar Kedro, Airflow, DVC y Docker en un sistema cohesivo.

Solución Implementada:

- Diseño de arquitectura clara con separación de responsabilidades
- Documentación detallada de interfaces entre componentes
- Testing iterativo de la integración completa

10.2 Configuración de Airflow en Docker

Desafío: Airflow requiere configuración específica de permisos y variables de entorno.

Solución:

- Uso de imagen oficial de Airflow como base
- Configuración de volúmenes para persistencia
- Variables de entorno bien documentadas en docker-compose

10.3 Gestión de Dependencias

Desafío: Compatibilidad entre versiones de bibliotecas (Kedro, Airflow, scikit-learn).

Solución:

- requirements.txt con versiones pinneadas
- Testing de compatibilidad en entorno Docker

- Documentación de versiones específicas

10.4 Performance de GridSearchCV

Desafío: GridSearchCV con múltiples modelos es computacionalmente costoso.

Solución:

- Uso de `n_jobs=-1` para paralelización
- Selección estratégica de grids de hiperparámetros
- Implementación de early stopping donde aplicable

---

11. Conclusiones

11.1 Logros Principales

El proyecto logró implementar exitosamente:

✅ **Sistema completo de Machine Learning** con 10 modelos entrenados y optimizados
✅ **Arquitectura MLOps profesional** integrando 4 tecnologías (Kedro, Airflow, DVC, Docker)
✅ **Automatización total** del flujo de entrenamiento y evaluación
✅ **Reproducibilidad garantizada** mediante versionamiento y contenerización
✅ **Portabilidad completa** ejecutable en cualquier sistema con Docker
✅ **Trazabilidad exhaustiva** de experimentos, datos y modelos

11.2 Evolución desde la Evaluación 1

Dimensión	Progreso
Complejidad técnica	EDA básico → Sistema MLOps completo
Automatización	Manual → Totalmente automatizada
Escalabilidad	Pipeline único → Múltiples pipelines independientes
Reproducibilidad	Limitada → Total (DVC + Docker)
**	