MÓDULO 07: EXPERIMENT TRACKING

MLflow a Fondo: Registry, Signatures, y Comparativa vs W&B

Guía MLOps v5.0: Senior Edition | DuqueOM | Noviembre 2025

MÓDULO 07: Experiment Tracking

De "Creo que usé learning_rate=0.01" a Reproducibilidad Total

"Si no puedo ver cómo llegaste a ese resultado, tu experimento no existe."

Duración	Teoría	Práctica
5-6 horas	25%	75%

ADR: MLflow vs Weights & Biases vs Neptune

```
╔═══════════════════════════════════════════════════════════════════════════════╗
║  ADR-007: Selección de Herramienta de Experiment Tracking                     ║
╠═══════════════════════════════════════════════════════════════════════════════╣
║                                                                               ║
║  COMPARATIVA:                                                                 ║
║                                                                               ║
║  │ Criterio         │ MLflow    │ W&B       │ Neptune   │                    ║
║  ├──────────────────┼───────────┼───────────┼───────────┤                    ║
║  │ Open Source      │  100%    │ × SaaS   │ × SaaS   │                   ║
║  │ Self-Hosted      │ □ Fácil   │ ⚠ Caro   │ ⚠ Caro   │                    ║
║  │ Costo (equipos)  │ $0        │ $$$       │ $$$       │                    ║
║  │ Model Registry   │          │          │          │                   ║
║  │ UI/UX            │       │     │      │                 ║
║  │ Colaboración     │       │     │      │                 ║
║  │ Integración DL   │       │     │      │                 ║
║  │ Comunidad        │     │      │      │                 ║
║                                                                               ║
║  DECISIÓN: MLflow para esta guía                                              ║
║                                                                               ║
║  RAZONES:                                                                     ║
║  • 100% open source = sin vendor lock-in                                      ║
║  • Self-hosted sin costo = ideal para aprender y proyectos personales          ║
║  • Estándar de facto en MLOps = skill transferible                            ║
║  • Model Registry incluido = workflow completo                                ║
║                                                                               ║
║  CUÁNDO USAR W&B:                                                             ║
║  • Proyectos de Deep Learning pesados (mejor visualización)                   ║
║  • Equipos grandes que necesitan colaboración avanzada                        ║
║  • Presupuesto disponible para SaaS                                           ║
║                                                                               ║
╚═══════════════════════════════════════════════════════════════════════════════╝
```

7.1 Arquitectura de MLflow

```
╔══════════════════════════════════════════════════════════════════════════════╗
║                          COMPONENTES DE MLFLOW                                ║
╠══════════════════════════════════════════════════════════════════════════════╣
║                                                                              ║
║   ┌──────────────────────────────────────────────────────────────────────┐   ║
║   │                       MLFLOW TRACKING                                │   ║
║   │  • Loguear parámetros, métricas, artefactos                          │   ║
║   │  • Organizar en experiments y runs                                   │   ║
║   │  • UI para visualizar y comparar                                     │   ║
║   └──────────────────────────────────────────────────────────────────────┘   ║
║                                                                              ║
║   ┌──────────────────────────────────────────────────────────────────────┐   ║
║   │                       MLFLOW PROJECTS                                │   ║
║   │  • Empaquetar código de ML reproducible                              │   ║
║   │  • MLproject file + conda/docker environment                         │   ║
║   └──────────────────────────────────────────────────────────────────────┘   ║
║                                                                              ║
║   ┌──────────────────────────────────────────────────────────────────────┐   ║
║   │                       MLFLOW MODELS                                  │   ║
║   │  • Formato estándar para guardar modelos                             │   ║
║   │  • Múltiples "flavors" (sklearn, pytorch, etc.)                      │   ║
║   │  • Fácil deployment (REST, batch, Spark)                             │   ║
║   └──────────────────────────────────────────────────────────────────────┘   ║
║                                                                              ║
║   ┌──────────────────────────────────────────────────────────────────────┐   ║
║   │                       MLFLOW MODEL REGISTRY                          │   ║
║   │  • Ciclo de vida: Staging → Production → Archived                    │   ║
║   │  • Versionado de modelos                                             │   ║
║   │  • Aprobaciones y transiciones                                       │   ║
║   └──────────────────────────────────────────────────────────────────────┘   ║
║                                                                              ║
╚══════════════════════════════════════════════════════════════════════════════╝
```

7.2 Setup y Configuración

Instalación

```
pip install mlflow
```

Modos de Operación

```
# ══════════════════════════════════════════════════════════════════════
# MODO 1: Local (desarrollo personal)
# ══════════════════════════════════════════════════════════════════════
# Los datos se guardan en ./mlruns/
# No necesitas servidor

# En código Python:
import mlflow
mlflow.set_tracking_uri("file:./mlruns")

# ══════════════════════════════════════════════════════════════════════
# MODO 2: Server Local (equipo pequeño)
# ══════════════════════════════════════════════════════════════════════
# Iniciar servidor
mlflow server \
    --host 0.0.0.0 \
    --port 5000 \
    --backend-store-uri sqlite:///mlflow.db \
    --default-artifact-root ./mlartifacts

# En código Python:
mlflow.set_tracking_uri("http://localhost:5000")

# ══════════════════════════════════════════════════════════════════════
# MODO 3: Server con PostgreSQL + S3 (producción)
# ══════════════════════════════════════════════════════════════════════
mlflow server \
    --host 0.0.0.0 \
    --port 5000 \
    --backend-store-uri postgresql://user:pass@host:5432/mlflow \
    --default-artifact-root s3://my-bucket/mlartifacts
```

docker-compose para MLflow Server

```yaml
# docker-compose-mlflow.yml
version: '3.8'

services:
  mlflow:
    image: ghcr.io/mlflow/mlflow:v2.8.0
    ports:
      - "5000:5000"
    environment:
      - MLFLOW_TRACKING_URI=sqlite:///mlflow/mlflow.db
    volumes:
      - mlflow-data:/mlflow
    command: >
      mlflow server
      --host 0.0.0.0
      --port 5000
      --backend-store-uri sqlite:///mlflow/mlflow.db
      --default-artifact-root /mlflow/artifacts
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:5000/health"]
      interval: 30s
      timeout: 10s
      retries: 3

volumes:
  mlflow-data:
```

7.3 Tracking: Loguear Experimentos

Estructura Básica

```python
import mlflow
from mlflow.models import infer_signature
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score, precision_score, recall_score, f1_score

# Configurar tracking URI
mlflow.set_tracking_uri("http://localhost:5000")  # o "file:./mlruns"

# Crear o usar experimento existente
mlflow.set_experiment("bankchurn-experiments")

# ══════════════════════════════════════════════════════════════════════
# OPCIÓN 1: Context Manager (Recomendado)
# ══════════════════════════════════════════════════════════════════════
with mlflow.start_run(run_name="rf_baseline_v1"):

    # 1. Loguear parámetros
    params = {
        "n_estimators": 100,
        "max_depth": 10,
        "random_state": 42,
        "class_weight": "balanced",
    }
    mlflow.log_params(params)

    # 2. Entrenar modelo
    model = RandomForestClassifier(**params)
    model.fit(X_train, y_train)

    # 3. Evaluar
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]

    # 4. Loguear métricas
    metrics = {
        "auc_roc": roc_auc_score(y_test, y_proba),
        "precision": precision_score(y_test, y_pred),
        "recall": recall_score(y_test, y_pred),
        "f1": f1_score(y_test, y_pred),
    }
    mlflow.log_metrics(metrics)

    # 5. Loguear artefactos adicionales
    # - Matriz de confusión
    import matplotlib.pyplot as plt
    from sklearn.metrics import ConfusionMatrixDisplay

    fig, ax = plt.subplots()
    ConfusionMatrixDisplay.from_predictions(y_test, y_pred, ax=ax)
    plt.savefig("confusion_matrix.png")
    mlflow.log_artifact("confusion_matrix.png")

    # - Feature importance
    importance_df = pd.DataFrame({
        'feature': X_train.columns,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=False)
    importance_df.to_csv("feature_importance.csv", index=False)
    mlflow.log_artifact("feature_importance.csv")

    # 6. Loguear modelo con signature
    signature = infer_signature(X_train, model.predict(X_train))
    mlflow.sklearn.log_model(
        model,
        "model",
        signature=signature,
        input_example=X_train.iloc[:3],
    )

    # 7. Tags para organización
    mlflow.set_tags({
        "model_type": "random_forest",
        "dataset_version": "v1.0",
        "author": "tu_nombre",
    })

# ══════════════════════════════════════════════════════════════════════
# OPCIÓN 2: Autolog (Rápido para sklearn)
# ══════════════════════════════════════════════════════════════════════
mlflow.sklearn.autolog()

with mlflow.start_run():
    model = RandomForestClassifier(n_estimators=100)
    model.fit(X_train, y_train)
    # MLflow loguea automáticamente: params, métricas, modelo
```

Loguear Plots y Visualizaciones

```python
import mlflow
import matplotlib.pyplot as plt
from sklearn.metrics import RocCurveDisplay, PrecisionRecallDisplay

with mlflow.start_run():
    # Entrenar modelo...

    # ROC Curve
    fig, ax = plt.subplots(figsize=(8, 6))
    RocCurveDisplay.from_predictions(y_test, y_proba, ax=ax)
    ax.set_title("ROC Curve - BankChurn")
    plt.tight_layout()
    mlflow.log_figure(fig, "roc_curve.png")
    plt.close()

    # Precision-Recall Curve
    fig, ax = plt.subplots(figsize=(8, 6))
    PrecisionRecallDisplay.from_predictions(y_test, y_proba, ax=ax)
    ax.set_title("Precision-Recall Curve")
    plt.tight_layout()
    mlflow.log_figure(fig, "pr_curve.png")
    plt.close()

    # Feature Importance Plot
    fig, ax = plt.subplots(figsize=(10, 8))
    importance = pd.DataFrame({
        'feature': feature_names,
        'importance': model.feature_importances_
    }).sort_values('importance', ascending=True)
    ax.barh(importance['feature'], importance['importance'])
    ax.set_title("Feature Importance")
    plt.tight_layout()
    mlflow.log_figure(fig, "feature_importance.png")
    plt.close()
```

7.4 Model Signatures

¿Qué son y Por Qué Importan?

```
╔══════════════════════════════════════════════════════════════════════════════╗
║                          MODEL SIGNATURES                                     ║
╠══════════════════════════════════════════════════════════════════════════════╣
║                                                                              ║
║  PROBLEMA SIN SIGNATURE:                                                     ║
║  • Cargas modelo de hace 3 meses                                             ║
║  • ¿Qué columnas espera? ¿En qué orden? ¿Qué tipos?                         ║
║  • Error en producción: "Expected 10 features, got 8"                        ║
║                                                                              ║
║  CON SIGNATURE:                                                              ║
║  • El modelo "sabe" qué input espera                                         ║
║  • Validación automática antes de predict                                    ║
║  • Documentación incluida en el artefacto                                    ║
║                                                                              ║
╚══════════════════════════════════════════════════════════════════════════════╝
```

Crear Signatures

```python
from mlflow.models import infer_signature, ModelSignature
from mlflow.types.schema import Schema, ColSpec

# ═══════════════════════════════════════════════════════════════════════
# OPCIÓN 1: Inferir automáticamente (Recomendado)
# ═══════════════════════════════════════════════════════════════════════
signature = infer_signature(
    model_input=X_train,
    model_output=model.predict(X_train),
)

# Para clasificadores, incluir probabilidades
signature = infer_signature(
    model_input=X_train,
    model_output=model.predict_proba(X_train),
)

# ═══════════════════════════════════════════════════════════════════════
# OPCIÓN 2: Definir manualmente (control total)
# ═══════════════════════════════════════════════════════════════════════
from mlflow.types import DataType

input_schema = Schema([
    ColSpec(DataType.double, "CreditScore"),
    ColSpec(DataType.integer, "Age"),
    ColSpec(DataType.integer, "Tenure"),
    ColSpec(DataType.double, "Balance"),
    ColSpec(DataType.integer, "NumOfProducts"),
    ColSpec(DataType.boolean, "HasCrCard"),
    ColSpec(DataType.boolean, "IsActiveMember"),
    ColSpec(DataType.double, "EstimatedSalary"),
    ColSpec(DataType.string, "Geography"),
    ColSpec(DataType.string, "Gender"),
])

output_schema = Schema([
    ColSpec(DataType.double, "churn_probability"),
])

signature = ModelSignature(inputs=input_schema, outputs=output_schema)

# Usar al loguear modelo
mlflow.sklearn.log_model(
    model,
    "model",
    signature=signature,
    input_example=X_train.iloc[:3],
)
```

7.5 Model Registry

Ciclo de Vida de Modelos

```
flowchart LR
    subgraph Development[" Development"]
        A[Experiment Run]
    end

    subgraph Registry[" Model Registry"]
        B[None]
        C[Staging]
        D[Production]
        E[Archived]
    end

    A -->|Register| B
    B -->|Promote| C
    C -->|Approve| D
    D -->|Replace| E
    C -->|Reject| E
```

Registrar Modelos

```python
import mlflow
from mlflow.tracking import MlflowClient

client = MlflowClient()

# ══════════════════════════════════════════════════════════════════════
# OPCIÓN 1: Registrar durante el run
# ══════════════════════════════════════════════════════════════════════
with mlflow.start_run() as run:
    # Entrenar...
    mlflow.sklearn.log_model(
        model,
        "model",
        registered_model_name="bankchurn-predictor",  # Registra automáticamente
    )

# ══════════════════════════════════════════════════════════════════════
# OPCIÓN 2: Registrar run existente
# ══════════════════════════════════════════════════════════════════════
run_id = "abc123..."
model_uri = f"runs:/{run_id}/model"

mlflow.register_model(
    model_uri=model_uri,
    name="bankchurn-predictor",
)

# ══════════════════════════════════════════════════════════════════════
# Gestionar versiones y stages
# ══════════════════════════════════════════════════════════════════════

# Ver todas las versiones
for mv in client.search_model_versions("name='bankchurn-predictor'"):
    print(f"Version: {mv.version}, Stage: {mv.current_stage}")

# Transicionar a Staging
client.transition_model_version_stage(
    name="bankchurn-predictor",
    version=1,
    stage="Staging",
)

# Transicionar a Production
client.transition_model_version_stage(
    name="bankchurn-predictor",
    version=1,
    stage="Production",
    archive_existing_versions=True,  # Archiva la versión anterior en Production
)

# Añadir descripción
client.update_model_version(
    name="bankchurn-predictor",
    version=1,
    description="Random Forest baseline con AUC 0.87. Entrenado con datos Q4 2024.",
)
```

Cargar Modelos desde Registry

```python
import mlflow

# ══════════════════════════════════════════════════════════════════════
# Por stage
# ══════════════════════════════════════════════════════════════════════
model = mlflow.sklearn.load_model("models:/bankchurn-predictor/Production")
predictions = model.predict(X_new)

# ══════════════════════════════════════════════════════════════════════
# Por versión específica
# ══════════════════════════════════════════════════════════════════════
model = mlflow.sklearn.load_model("models:/bankchurn-predictor/3")

# ══════════════════════════════════════════════════════════════════════
# Cargar como PyFunc (genérico)
# ══════════════════════════════════════════════════════════════════════
model = mlflow.pyfunc.load_model("models:/bankchurn-predictor/Production")
predictions = model.predict(X_new)
```

7.6 Integración Completa con Pipeline

```python
# src/bankchurn/training.py
import mlflow
from mlflow.models import infer_signature
from pathlib import Path
from typing import Dict, Any
import pandas as pd
import joblib

from bankchurn.config import TrainingConfig
from bankchurn.pipeline import build_pipeline
from bankchurn.evaluation import evaluate_model


class MLflowTrainer:
    """Trainer con integración completa de MLflow."""

    def __init__(self, config: TrainingConfig):
        self.config = config
        self._setup_mlflow()

    def _setup_mlflow(self):
        """Configura MLflow tracking."""
        mlflow.set_tracking_uri(self.config.mlflow.tracking_uri)
        mlflow.set_experiment(self.config.mlflow.experiment_name)

    def run(
        self,
        X_train: pd.DataFrame,
        y_train: pd.Series,
        X_test: pd.DataFrame,
        y_test: pd.Series,
    ) -> Dict[str, Any]:
        """Ejecuta entrenamiento con tracking completo."""

        with mlflow.start_run(run_name=self.config.mlflow.run_name):
            # 1. Log configuración
            mlflow.log_params(self.config.model.model_dump())
            mlflow.log_params({
                "data_version": self.config.data.version,
                "train_size": len(X_train),
                "test_size": len(X_test),
            })

            # 2. Construir y entrenar pipeline
            pipeline = build_pipeline(
                numerical_features=self.config.features.numerical,
                categorical_features=self.config.features.categorical,
                binary_features=self.config.features.binary,
                model_params=self.config.model.model_dump(),
            )
            pipeline.fit(X_train, y_train)

            # 3. Evaluar
            metrics = evaluate_model(pipeline, X_test, y_test)
            mlflow.log_metrics(metrics)

            # 4. Log modelo con signature
            signature = infer_signature(
                X_train,
                pipeline.predict_proba(X_train)
            )

            mlflow.sklearn.log_model(
                pipeline,
                "pipeline",
                signature=signature,
                input_example=X_train.iloc[:5],
                registered_model_name=self.config.mlflow.model_name,
            )

            # 5. Log artefactos adicionales
            self._log_artifacts(pipeline, X_train, X_test, y_test)

            # 6. Tags
            mlflow.set_tags({
                "model_type": self.config.model.model_type,
                "stage": "development",
            })

            return metrics

    def _log_artifacts(self, pipeline, X_train, X_test, y_test):
        """Loguea artefactos adicionales."""
        import matplotlib.pyplot as plt
        from sklearn.metrics import RocCurveDisplay, ConfusionMatrixDisplay

        y_proba = pipeline.predict_proba(X_test)[:, 1]
        y_pred = pipeline.predict(X_test)

        # ROC Curve
        fig, ax = plt.subplots()
        RocCurveDisplay.from_predictions(y_test, y_proba, ax=ax)
        mlflow.log_figure(fig, "plots/roc_curve.png")
        plt.close()

        # Confusion Matrix
        fig, ax = plt.subplots()
        ConfusionMatrixDisplay.from_predictions(y_test, y_pred, ax=ax)
        mlflow.log_figure(fig, "plots/confusion_matrix.png")
        plt.close()

        # Config YAML
        self.config.to_yaml("config_snapshot.yaml")
        mlflow.log_artifact("config_snapshot.yaml")


# ══════════════════════════════════════════════════════════════════════
# USO
# ══════════════════════════════════════════════════════════════════════
if __name__ == "__main__":
    from bankchurn.config import load_config
    from bankchurn.data import load_and_split_data

    config = load_config("configs/config.yaml")
    X_train, X_test, y_train, y_test = load_and_split_data(config)

    trainer = MLflowTrainer(config)
    metrics = trainer.run(X_train, y_train, X_test, y_test)

    print(f"AUC-ROC: {metrics['auc_roc']:.4f}")
```

7.7 UI y Comparación de Experimentos

Iniciar UI

```
# Si usas file-based tracking
mlflow ui --port 5000

# Si usas server
# Ya está disponible en http://localhost:5000
```

Comparar Runs Programáticamente

```python
import mlflow
from mlflow.tracking import MlflowClient

client = MlflowClient()

# Buscar runs del experimento
experiment = client.get_experiment_by_name("bankchurn-experiments")
runs = client.search_runs(
    experiment_ids=[experiment.experiment_id],
    filter_string="metrics.auc_roc > 0.80",
    order_by=["metrics.auc_roc DESC"],
    max_results=10,
)

# Comparar
for run in runs:
    print(f"Run: {run.info.run_name}")
    print(f"  AUC: {run.data.metrics['auc_roc']:.4f}")
    print(f"  Params: {run.data.params}")
    print()

# Obtener mejor run
best_run = runs[0]
best_model_uri = f"runs:/{best_run.info.run_id}/pipeline"
```

7.8 Ejercicio Integrador

Setup MLflow para Tu Proyecto

1. **Iniciar** MLflow server (o usar file-based)
2. **Crear** experimento para tu proyecto
3. **Loguear** al menos 3 runs con diferentes hiperparámetros
4. **Comparar** resultados en la UI
5. **Registrar** el mejor modelo en el registry

Checklist

```
TRACKING:
[ ] MLflow server corriendo (o file-based configurado)
[ ] Experimento creado
[ ] Runs con params, metrics, y artifacts

MODEL REGISTRY:
[ ] Modelo registrado
[ ] Al menos 1 versión en Staging
[ ] Descripción añadida

ARTIFACTS:
[ ] Modelo con signature
[ ] Plots (ROC, confusion matrix)
[ ] Config snapshot
```

Siguiente Paso

Con experimentos trackeados, es hora de **testear tu código ML** profesionalmente.

Ir a Módulo 08: Testing para ML →

Módulo 07 completado. Tus experimentos ahora son reproducibles y comparables.