MÓDULO 20: PLAN DE ESTUDIOS

Roadmap de 10 Semanas

Guía MLOps v2.0 | DuqueOM | Noviembre 2025

MÓDULO 20: Plan de Estudios

Roadmap de 10 Semanas

"Un camino claro hacia la maestría en MLOps."

Nivel	Duración
Referencia	10 semanas

Objetivo

Proveer un plan estructurado de 10 semanas para construir un portafolio MLOps completo, con lecturas, ejercicios prácticos y entregables evaluables.

Vista General

```
┌─────────────────────────────────────────────────────────────┐
│                  PLAN DE 10 SEMANAS                         │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  Semana 1-2:  FUNDAMENTOS                                   │
│               Git, estructura de repos, Python packaging    │
│                                                             │
│  Semana 3-4:  DATOS Y PIPELINE                              │
│               DVC, pipeline de entrenamiento, sklearn       │
│                                                             │
│  Semana 5-6:  TRACKING Y TESTING                            │
│               MLflow, pytest, coverage                      │
│                                                             │
│  Semana 7-8:  PRODUCTIZACIÓN                                │
│               Docker, FastAPI, CI/CD                        │
│                                                             │
│  Semana 9:    OPERACIONES                                   │
│               Kubernetes básico, Terraform, monitoreo       │
│                                                             │
│  Semana 10:   DOCUMENTACIÓN Y DEMO                          │
│               Model Cards, README, video demo               │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

Semana 1: Fundamentos de Git y Estructuración

Objetivos

- Dominar comandos esenciales de Git
- Entender branching y merge
- Crear estructura de repositorio profesional

Lecturas (2-3 horas)

- [ ] Pro Git Book - Capítulos 1-3
- [ ] Conventional Commits
- [ ] Módulo 04: GIT_GITHUB.md de esta guía

Práctica (4-5 horas)

```
# Ejercicio 1: Crear repo con estructura ML
mkdir mi-proyecto-ml
cd mi-proyecto-ml
git init

# Crear estructura
mkdir -p src/myproject data/raw models tests configs docs

# Crear archivos base
touch src/myproject/__init__.py
touch tests/__init__.py
touch README.md requirements.txt .gitignore

# Primer commit
git add .
git commit -m "chore: initial project structure"
```

```
# Ejercicio 2: Branching workflow
git checkout -b develop
git checkout -b feature/add-training

# Hacer cambios...
git add .
git commit -m "feat: add training script skeleton"

# Merge
git checkout develop
git merge feature/add-training
```

Entregable

- [ ] Repositorio en GitHub con estructura profesional
- [ ] Al menos 5 commits con mensajes convencionales
- [ ] README.md básico con descripción del proyecto
- [ ] .gitignore apropiado para Python/ML

Criterios de Evaluación

Criterio	Puntos
Estructura de carpetas correcta	25
Commits con formato convencional	25
README con secciones básicas	25
.gitignore completo	25

Semana 2: Python Packaging y Configuración

Objetivos

- Crear paquete Python instalable
- Usar Pydantic para configuración
- Implementar logging profesional

Lecturas (2-3 horas)

- [ ] Python Packaging Guide
- [ ] Pydantic Documentation
- [ ] Módulo 03: ESTRUCTURA_REPO.md

Práctica (5-6 horas)

```toml
# Ejercicio 1: pyproject.toml
# pyproject.toml
[build-system]
requires = ["setuptools>=65.0"]
build-backend = "setuptools.build_meta"

[project]
name = "mi-proyecto-ml"
version = "0.1.0"
dependencies = [
    "pandas>=1.3.0",
    "scikit-learn>=1.0.0",
    "pydantic>=2.0.0",
]

[project.optional-dependencies]
dev = ["pytest", "black", "flake8"]
```

```python
# Ejercicio 2: Configuración con Pydantic
# src/myproject/config.py
from pydantic import BaseModel, Field
from pathlib import Path
import yaml

class DataConfig(BaseModel):
    target_column: str = "target"
    test_size: float = Field(default=0.2, ge=0.1, le=0.5)
    random_state: int = 42

class ModelConfig(BaseModel):
    n_estimators: int = Field(default=100, ge=10, le=1000)
    max_depth: int = Field(default=10, ge=1, le=50)

class ProjectConfig(BaseModel):
    data: DataConfig = DataConfig()
    model: ModelConfig = ModelConfig()

    @classmethod
    def from_yaml(cls, path: str) -> "ProjectConfig":
        with open(path) as f:
            return cls(**yaml.safe_load(f))
```

```python
# Ejercicio 3: Logging
# src/myproject/logger.py
import logging
import sys

def get_logger(name: str) -> logging.Logger:
    logger = logging.getLogger(name)
    logger.setLevel(logging.INFO)

    handler = logging.StreamHandler(sys.stdout)
    formatter = logging.Formatter(
        '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
    )
    handler.setFormatter(formatter)
    logger.addHandler(handler)

    return logger
```

Entregable

- [ ] `pyproject.toml` completo
- [ ] Módulo `config.py` con Pydantic
- [ ] `configs/config.yaml` de ejemplo
- [ ] Sistema de logging configurado
- [ ] Paquete instalable con `pip install -e .`

Semana 3: DVC y Versionado de Datos

Objetivos

- Instalar y configurar DVC
- Versionar dataset
- Crear pipeline DVC básico

Lecturas (2 horas)

- [ ] DVC Getting Started
- [ ] Módulo 05: DVC.md

Práctica (4-5 horas)

```bash
# Ejercicio 1: Setup DVC
pip install dvc

cd mi-proyecto-ml
dvc init

# Configurar remote local (para demo)
mkdir -p ../dvc-storage
dvc remote add -d myremote ../dvc-storage
```

```
# Ejercicio 2: Versionar datos
# Añadir dataset
dvc add data/raw/dataset.csv

# Commit metadatos
git add data/raw/dataset.csv.dvc data/raw/.gitignore
git commit -m "data: add initial dataset v1"

# Subir datos
dvc push
```

```
# Ejercicio 3: Pipeline DVC
# dvc.yaml
stages:
  preprocess:
    cmd: python src/myproject/preprocess.py
    deps:
      - data/raw/dataset.csv
      - src/myproject/preprocess.py
    outs:
      - data/processed/train.csv
      - data/processed/test.csv

  train:
    cmd: python src/myproject/train.py
    deps:
      - data/processed/train.csv
      - src/myproject/train.py
    outs:
      - models/model.pkl
```

```
# Ejecutar pipeline
dvc repro

# Ver grafo
dvc dag
```

Entregable

- [ ] DVC inicializado en el repo
- [ ] Dataset versionado con DVC
- [ ] `dvc.yaml` con al menos 2 stages
- [ ] Pipeline ejecutable con `dvc repro`

Semana 4: Pipeline de Entrenamiento

Objetivos

- Implementar Trainer modular
- Usar sklearn Pipeline
- Guardar modelos correctamente

Lecturas (2-3 horas)

- [ ] sklearn Pipeline
- [ ] Módulo 06: PIPELINE_ML.md

Práctica (6-7 horas)

```python
# Ejercicio: Implementar ChurnTrainer completo
# src/myproject/training.py

import pandas as pd
import joblib
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, cross_val_score

from .config import ProjectConfig
from .logger import get_logger

logger = get_logger(__name__)

class Trainer:
    def __init__(self, config: ProjectConfig):
        self.config = config
        self.pipeline = None

    def _build_preprocessor(self, X: pd.DataFrame):
        numeric_features = X.select_dtypes(include=['int64', 'float64']).columns
        categorical_features = X.select_dtypes(include=['object']).columns

        numeric_transformer = Pipeline([
            ('imputer', SimpleImputer(strategy='median')),
            ('scaler', StandardScaler())
        ])

        categorical_transformer = Pipeline([
            ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
            ('encoder', OneHotEncoder(drop='first', handle_unknown='ignore'))
        ])

        return ColumnTransformer([
            ('num', numeric_transformer, numeric_features),
            ('cat', categorical_transformer, categorical_features)
        ])

    def train(self, df: pd.DataFrame) -> dict:
        logger.info(f"Training with {len(df)} samples")

        X = df.drop(columns=[self.config.data.target_column])
        y = df[self.config.data.target_column]

        X_train, X_test, y_train, y_test = train_test_split(
            X, y,
            test_size=self.config.data.test_size,
            random_state=self.config.data.random_state,
            stratify=y
        )

        preprocessor = self._build_preprocessor(X_train)
        model = RandomForestClassifier(
            n_estimators=self.config.model.n_estimators,
            max_depth=self.config.model.max_depth,
            random_state=self.config.data.random_state
        )

        self.pipeline = Pipeline([
            ('preprocessor', preprocessor),
            ('classifier', model)
        ])

        self.pipeline.fit(X_train, y_train)

        # Métricas
        train_score = self.pipeline.score(X_train, y_train)
        test_score = self.pipeline.score(X_test, y_test)

        cv_scores = cross_val_score(self.pipeline, X, y, cv=5)

        metrics = {
            'train_accuracy': train_score,
            'test_accuracy': test_score,
            'cv_mean': cv_scores.mean(),
            'cv_std': cv_scores.std()
        }

        logger.info(f"Test accuracy: {test_score:.3f}")
        return metrics

    def save(self, path: str):
        joblib.dump(self.pipeline, path)
        logger.info(f"Model saved to {path}")
```

Entregable

- ☐ `training.py` con clase Trainer completa
- ☐ `predict.py` con clase Predictor
- ☐ Pipeline sklearn unificado
- ☐ Script `main.py` que orquesta todo
- ☐ Modelo entrenado en `models/`

Semana 5: MLflow Tracking

Objetivos

- Configurar MLflow
- Trackear experimentos
- Comparar runs

Lecturas (2 horas)

- ☐ MLflow Quickstart

- [ ] Módulo 07: MLFLOW.md

Práctica (4-5 horas)

```python
# Integrar MLflow en Trainer
import mlflow
import mlflow.sklearn

class Trainer:
    def train_with_tracking(self, df: pd.DataFrame, experiment_name: str):
        mlflow.set_tracking_uri("file:./mlruns")
        mlflow.set_experiment(experiment_name)

        with mlflow.start_run():
            # Log params
            mlflow.log_params({
                "n_estimators": self.config.model.n_estimators,
                "max_depth": self.config.model.max_depth,
                "test_size": self.config.data.test_size
            })

            # Train
            metrics = self.train(df)

            # Log metrics
            mlflow.log_metrics(metrics)

            # Log model
            mlflow.sklearn.log_model(self.pipeline, "model")

            # Log config
            mlflow.log_artifact("configs/config.yaml")

        return metrics
```

```bash
# Ejecutar experimentos
python main.py --experiment baseline

# Cambiar config y re-ejecutar
python main.py --experiment tuned_v1

# Ver UI
mlflow ui --port 5000
```

Entregable

- [ ] MLflow integrado en training
- [ ] Al menos 3 experimentos logueados
- [ ] Comparación de runs en UI
- [ ] Screenshot de MLflow UI en docs/

Semana 6: Testing con pytest

Objetivos

- Escribir tests unitarios
- Crear tests de integración
- Alcanzar 70% coverage

Lecturas (2-3 horas)

- [ ] pytest Documentation
- [ ] Módulo 08: TESTING.md

Práctica (6-7 horas)

```python
# tests/conftest.py
import pytest
import pandas as pd
import numpy as np

@pytest.fixture
def sample_data():
    np.random.seed(42)
    n = 100
    return pd.DataFrame({
        'feature1': np.random.randn(n),
        'feature2': np.random.choice(['A', 'B', 'C'], n),
        'target': np.random.randint(0, 2, n)
    })

@pytest.fixture
def config():
    from src.myproject.config import ProjectConfig
    return ProjectConfig()
```

```python
# tests/test_training.py
import pytest
from src.myproject.training import Trainer

class TestTrainer:
    def test_train_returns_metrics(self, sample_data, config):
        trainer = Trainer(config)
        metrics = trainer.train(sample_data)

        assert 'test_accuracy' in metrics
        assert 0 <= metrics['test_accuracy'] <= 1

    def test_pipeline_created(self, sample_data, config):
        trainer = Trainer(config)
        trainer.train(sample_data)

        assert trainer.pipeline is not None

    def test_handles_missing_values(self, sample_data, config):
        # Introducir NaN
        sample_data.loc[0:5, 'feature1'] = np.nan

        trainer = Trainer(config)
        metrics = trainer.train(sample_data)

        assert metrics is not None  # No crash
```

```bash
# Ejecutar tests
pytest tests/ -v --cov=src --cov-report=html
```

Entregable

- [ ] `tests/conftest.py` con fixtures
- [ ] `tests/test_training.py` (≥5 tests)
- [ ] `tests/test_config.py` (≥3 tests)
- [ ] Coverage ≥70%
- [ ] Reporte HTML en `htmlcov/`

Semana 7: Docker

Objetivos

- Crear Dockerfile multi-stage
- Usar docker-compose
- Optimizar tamaño de imagen

Lecturas (2-3 horas)

- [ ] Dockerfile Best Practices
- [ ] Módulo 10: DOCKER.md

Práctica (5-6 horas)

```dockerfile
# Dockerfile
# Stage 1: Builder
FROM python:3.11-slim AS builder
WORKDIR /build
COPY requirements.txt .
RUN pip wheel --no-cache-dir --wheel-dir /wheels -r requirements.txt

# Stage 2: Runtime
FROM python:3.11-slim AS runtime
WORKDIR /app

# Usuario no-root
RUN useradd -r appuser

# Copiar wheels y instalar
COPY --from=builder /wheels /wheels
RUN pip install --no-cache /wheels/*

# Copiar código
COPY . .

USER appuser
EXPOSE 8000
CMD ["python", "-m", "uvicorn", "app.main:app", "--host", "0.0.0.0"]
```

```yaml
# docker-compose.yml
version: '3.8'
services:
  api:
    build: .
    ports:
      - "8000:8000"
    volumes:
      - ./models:/app/models:ro
    environment:
      - LOG_LEVEL=INFO
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost:8000/health"]
      interval: 30s
```

```
# Build y run
docker build -t mi-proyecto-ml:latest .
docker run -p 8000:8000 mi-proyecto-ml:latest

# Con compose
docker-compose up -d
```

Entregable

- [ ] Dockerfile multi-stage funcional
- [ ] docker-compose.yml
- [ ] Imagen <500MB
- [ ] Healthcheck configurado
- [ ] `.dockerignore` apropiado

Semana 8: FastAPI y CI/CD

Objetivos

- Crear API REST con FastAPI
- Configurar GitHub Actions
- Automatizar tests y build

Lecturas (3-4 horas)

- [ ] FastAPI Tutorial
- [ ] GitHub Actions Docs
- [ ] Módulos 11 y 09 de esta guía

Práctica (7-8 horas)

```python
# app/main.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import joblib

app = FastAPI(title="ML API", version="1.0.0")

# Cargar modelo al inicio
model = joblib.load("models/model.pkl")

class PredictRequest(BaseModel):
    feature1: float
    feature2: str

class PredictResponse(BaseModel):
    prediction: int
    probability: float

@app.get("/health")
def health():
    return {"status": "healthy"}

@app.post("/predict", response_model=PredictResponse)
def predict(request: PredictRequest):
    import pandas as pd
    df = pd.DataFrame([request.dict()])
    pred = model.predict(df)[0]
    proba = model.predict_proba(df)[0, 1]
    return PredictResponse(prediction=int(pred), probability=float(proba))
```

```yaml
# .github/workflows/ci.yml
name: CI
on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - run: pip install -e ".[dev]"
      - run: pytest --cov=src --cov-fail-under=70

  build:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - run: docker build -t mi-proyecto:${{ github.sha }} .
```

Entregable

- [ ] API FastAPI funcional con /health y /predict
- [ ] Documentación automática en /docs
- [ ] GitHub Actions workflow
- [ ] Badge de CI en README
- [ ] Tests de API

Semana 9: Kubernetes y Monitoreo Básico

Objetivos

- Crear manifiestos K8s básicos
- Entender Deployment, Service, Ingress
- Configurar métricas con Prometheus

Lecturas (3 horas)

- [ ] Kubernetes Basics
- [ ] Módulos 12 y 14

Práctica (5-6 horas)

```yaml
# k8s/deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-api
spec:
  replicas: 2
  selector:
    matchLabels:
      app: ml-api
  template:
    metadata:
      labels:
        app: ml-api
    spec:
      containers:
      - name: ml-api
        image: mi-proyecto:latest
        ports:
        - containerPort: 8000
        resources:
          requests:
            memory: "256Mi"
            cpu: "250m"
          limits:
            memory: "512Mi"
            cpu: "500m"
        readinessProbe:
          httpGet:
            path: /health
            port: 8000
---
apiVersion: v1
kind: Service
metadata:
  name: ml-api-service
spec:
  selector:
    app: ml-api
  ports:
  - port: 80
    targetPort: 8000
```

Entregable

- [ ] `k8s/deployment.yaml`
- [ ] `k8s/service.yaml`
- [ ] Prometheus config básico
- [ ] Documentación de deploy en K8s

Semana 10: Documentación y Demo

Objetivos

- Crear Model Card completo
- Escribir README profesional
- Grabar video demo

Lecturas (2 horas)

- [ ] Model Cards Paper
- [ ] Módulos 15, 16, 17

Práctica (6-7 horas)

```
# Model Card Template

## Model Details
- **Name**: Customer Churn Predictor
- **Version**: 1.0.0
- **Type**: Binary Classification
- **Framework**: scikit-learn

## Intended Use
- **Primary use**: Identificar clientes en riesgo de abandono
- **Users**: Equipo de retención
- **Out of scope**: Decisiones automáticas sin supervisión humana

## Training Data
- **Source**: Synthetic dataset
- **Size**: 10,000 samples
- **Features**: 14
- **Target distribution**: 20% churn, 80% retained

## Performance
| Metric   | Value   |
|----------|---------|
| Accuracy | 0.85    |
| F1 Score | 0.72    |
| AUC-ROC  | 0.87    |

## Limitations
- Entrenado en datos sintéticos
- Puede tener sesgo por geografía
- Requiere reentrenamiento periódico

## Ethical Considerations
- No usar para discriminar clientes
- Supervisión humana requerida
- Monitorear fairness por grupos
```

Guión para Video Demo (3-5 minutos)

```
0:00-0:30  Introducción
           "Hola, soy [nombre] y voy a mostrar mi portafolio MLOps..."

0:30-1:00  El problema
           "Este proyecto predice abandono de clientes bancarios..."

1:00-2:00  Estructura del código
           Mostrar árbol de archivos, explicar organización

2:00-3:00  Demo en vivo
           - git clone
           - docker-compose up
           - Hacer predicción via curl/Swagger

3:00-4:00  CI/CD
           Mostrar GitHub Actions, tests pasando

4:00-4:30  Cierre
           "Gracias por ver, el código está en..."
```

Entregable Final

- [ ] Model Card completo
- [ ] README profesional con badges
- [ ] Video demo (3-5 min)
- [ ] Portafolio publicado en GitHub

Rúbrica de Evaluación Final

Componente	Peso	Criterios
Código	30%	Modular, documentado, sigue PEP8
Testing	20%	Coverage ≥70%, tests significativos
CI/CD	15%	Pipeline completo, pasa en verde
Docker	15%	Multi-stage, optimizado, funcional
Documentación	10%	README, Model Card, comments
Demo	10%	Video claro, profesional

Certificado de Completitud

Al completar todos los módulos con los entregables aprobados, habrás demostrado competencia en:

- Arquitectura de proyectos ML
- Control de versiones (código y datos)

- Pipelines de entrenamiento reproducibles
- Tracking de experimentos
- Testing profesional
- CI/CD automatizado
- Contenerización y APIs
- Documentación técnica

Navegación

◀ Anterior	Índice	▶ Siguiente
19_DECISIONES_TECH.md	Índice	21_PLANTILLAS.md

Módulo 20 Completado