RÚBRICA DE EVALUACIÓN - PORTAFOLIO MLOps

Criterios Detallados para Evaluación de Proyectos

Guía MLOps v3.0 | DuqueOM | Noviembre 2025

RÚBRICA DE EVALUACIÓN

Criterios para Evaluar un Portafolio MLOps Profesional

Puntuación Total	Calificación
100 puntos	Escala 0-100

Tabla de Contenidos

1. Resumen de Criterios

Criterio	Peso	Descripción
Calidad del Código	20%	Estructura, estilo, modularidad
Reproducibilidad	20%	Entorno, dependencias, ejecución
Testing y CI/CD	15%	Cobertura, automatización, pipeline
Versionado y Trazabilidad	15%	Git, DVC, MLflow, experimentos
Documentación	15%	README, Model Cards, comentarios
Presentación y Demo	15%	Video, comunicación, profesionalismo
TOTAL	**100%**	

Criterio 1: Calidad del Código (20 puntos)

1.1 Estructura del Proyecto (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Archivos desorganizados, sin estructura clara
2-3	Básico	Estructura básica pero incompleta
4	Competente	Estructura estándar con separación de concerns
5	Destacado	Estructura profesional, modular, escalable

Indicadores de nivel Destacado: - Separación clara: `src/`, `tests/`, `data/`, `notebooks/`, `configs/` - Módulos Python correctamente organizados con `__init__.py` - Configuración externalizada (YAML/JSON) - Makefile o scripts de automatización

1.2 Estilo y Legibilidad (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Código ilegible, sin formato consistente
2-3	Básico	Formato inconsistente, nombres poco claros
4	Competente	PEP8 seguido, nombres descriptivos
5	Destacado	Código limpio, formateado automáticamente (black/ruff)

Indicadores de nivel Destacado: - Formateador configurado (black, ruff) - Linter sin errores (flake8, pylint) - Nombres descriptivos en inglés - Líneas < 10 caracteres

1.3 Modularidad y Reutilización (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Todo en un archivo, código duplicado
2-3	Básico	Algunas funciones, pero código repetido
4	Competente	Funciones modulares, poca duplicación
5	Destacado	Clases y funciones reutilizables, principios SOLID

1.4 Manejo de Errores (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin manejo de errores, crashes frecuentes
2-3	Básico	Try/except genéricos, mensajes poco útiles
4	Competente	Excepciones específicas, mensajes informativos
5	Destacado	Logging estructurado, errores recuperables

Criterio 2: Reproducibilidad (20 puntos)

2.1 Gestión de Dependencias (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin requirements.txt o versiones
2-3	Básico	requirements.txt sin versiones fijadas
4	Competente	Versiones fijadas, separación dev/prod
5	Destacado	Lock file (poetry.lock), reproducible exacto

Ejemplo nivel Destacado:

```
# pyproject.toml con Poetry
# O requirements.txt con hashes
pandas==2.0.3 --hash=sha256:...
```

2.2 Configuración del Entorno (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Requiere instalación manual no documentada
2-3	Básico	Instrucciones incompletas o con errores
4	Competente	Setup funcional con virtualenv
5	Destacado	Docker + devcontainer, setup automático

2.3 Ejecución del Pipeline (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Pipeline no ejecuta o requiere modificaciones
2-3	Básico	Ejecuta con pasos manuales intermedios
4	Competente	Un comando ejecuta todo (make train)
5	Destacado	DVC pipeline completo, reproducible en cualquier entorno

2.4 Semillas y Determinismo (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Resultados diferentes en cada ejecución
2-3	Básico	Algunas semillas, resultados parcialmente reproducibles
4	Competente	Semillas en train/test split y modelo
5	Destacado	100% determinístico, documentado

Criterio 3: Testing y CI/CD (15 puntos)

3.1 Cobertura de Tests (5 puntos)

Puntos	Nivel	Cobertura	Descripción
0-1	Insuficiente	0-20%	Sin tests o mínimos
2-3	Básico	20-50%	Tests básicos para funciones principales
4	Competente	50-80%	Tests unitarios y algunos de integración
5	Destacado	80%+	Tests completos, edge cases, mocking

3.2 Calidad de Tests (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Tests que no verifican comportamiento real
2-3	Básico	Tests simples, sin fixtures ni parametrización
4	Competente	Tests bien estructurados, fixtures, asserts claros
5	Destacado	TDD, tests de regresión, property-based testing

3.3 Pipeline CI/CD (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin CI/CD configurado
2-3	Básico	CI ejecuta pero falla o incompleto
4	Competente	CI funcional: lint + tests + build
5	Destacado	CI/CD completo con deploy staging, matrix de versiones

Ejemplo nivel Destacado:

```yaml
# .github/workflows/ci.yml
jobs:
  test:
    strategy:
      matrix:
        python-version: [3.9, 3.10, 3.11]
    steps:
      - lint
      - test
      - build-docker
      - deploy-staging
```

Criterio 4: Versionado y Trazabilidad (15 puntos)

4.1 Control de Versiones Git (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Commits gigantes, mensajes como “fix” o “update”
2-3	Básico	Commits frecuentes pero sin convención
4	Competente	Conventional commits, branches organizados
5	Destacado	Git flow, PRs con revisión, tags semánticos

Ejemplo nivel Destacado:

```
feat(model): add XGBoost classifier
fix(data): handle missing values in age column
docs(readme): add installation instructions
test(api): add integration tests for /predict
```

4.2 Versionado de Datos (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Datos en Git o sin versionar
2-3	Básico	DVC inicializado pero sin usar activamente
4	Competente	Datos trackeados, remote configurado
5	Destacado	Pipeline DVC, métricas trackeadas, múltiples versiones

4.3 Tracking de Experimentos (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin tracking, resultados en notebooks
2-3	Básico	MLflow básico, pocos experimentos
4	Competente	Experimentos organizados, métricas comparables
5	Destacado	Model Registry, autologging, comparaciones documentadas

---

Criterio 5: Documentación (15 puntos)

5.1 README Principal (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin README o una línea
2-3	Básico	Descripción básica, sin instrucciones completas
4	Competente	Secciones: descripción, instalación, uso, estructura
5	Destacado	Badges, ToC, GIFs, ejemplos ejecutables, contribución

Checklist nivel Destacado: - ☐ Badges (CI, coverage, license, version) - ☐ Descripción clara del problema y solución - ☐ Instalación paso a paso verificada - ☐ Ejemplo de uso con outputs esperados - ☐ Estructura del proyecto explicada - ☐ Guía de contribución - ☐ Licencia y contacto

5.2 Model Cards / Data Cards (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin documentación del modelo
2-3	Básico	Descripción básica sin métricas ni limitaciones
4	Competente	Model Card con métricas, datos de entrenamiento
5	Destacado	Model Card completo: bias, limitaciones, uso ético

5.3 Docstrings y Comentarios (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin docstrings ni comentarios
2-3	Básico	Algunos docstrings, comentarios redundantes
4	Competente	Docstrings en funciones públicas, comentarios útiles
5	Destacado	Google/NumPy style docstrings, API docs generables

Criterio 6: Presentación y Demo (15 puntos)

6.1 Video Demo (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Sin video o video inutilizable
2-3	Básico	Video básico, audio/video de baja calidad
4	Competente	Demo clara de 3-5 min, muestra funcionalidad principal
5	Destacado	Producción profesional, storytelling, edición limpia

Guion sugerido para demo (3-5 min): 1. **Intro** (30s): Problema y solución 2. **Datos** (30s): De dónde vienen, qué representan 3. **Pipeline** (1 min): Mostra ejecución 4. **Resultados** (1 min): Métricas, MLflow UI 5. **API** (1 min): Demo de predicción en vivo 6. **Cierre** (30s): Próximos pasos, contacto

6.2 Comunicación Técnica (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Explicaciones confusas o incorrectas
2-3	Básico	Explica qué hace pero no por qué
4	Competente	Justifica decisiones técnicas claramente
5	Destacado	Comunica trade-offs, limitaciones, alternativas

6.3 Profesionalismo (5 puntos)

Puntos	Nivel	Descripción
0-1	Insuficiente	Presentación descuidada, errores graves
2-3	Básico	Presentable pero con detalles mejorables
4	Competente	Profesional, bien preparado
5	Destacado	Listo para mostrar en entrevista o conferencia

Escala de Calificación

Conversión de Puntos a Calificación

Rango	Calificación	Descripción
90-100	A / Destacado	Portafolio profesional, listo para producción
80-89	B / Competente	Cumple estándares profesionales, mejoras menores
70-79	C / Básico	Funcional pero requiere mejoras significativas
60-69	D / Mínimo	Cumple requisitos mínimos, muchas áreas a mejorar
0-59	F / Insuficiente	No cumple requisitos mínimos

Criterios de Aprobación

```
╔═══════════════════════════════════════════════════════════╗
║                REQUISITOS MÍNIMOS PARA APROBAR            ║
╠═══════════════════════════════════════════════════════════╣
║                                                           ║
║  Para obtener calificación APROBATORIA (≥60) se requiere: ║
║                                                           ║
║  ✓ Repositorio público en GitHub                          ║
║  ✓ README con instalación y uso                           ║
║  ✓ Pipeline que ejecute sin errores                       ║
║  ✓ Al menos 5 tests pasando                               ║
║  ✓ Dockerfile funcional                                   ║
║  ✓ CI/CD ejecutando (puede tener warnings)                ║
║                                                           ║
║  NINGÚN criterio puede tener puntuación 0.                ║
╚═══════════════════════════════════════════════════════════╝
```

Checklist de Autoevaluación

Usa este checklist antes de entregar tu proyecto:

Código

- [ ] Estructura de carpetas profesional
- [ ] Código formateado (black/ruff)
- [ ] Sin errores de linter
- [ ] Funciones documentadas con docstrings
- [ ] Manejo de errores implementado

Reproducibilidad

- [ ] requirements.txt con versiones fijadas
- [ ] Instrucciones de setup verificadas en máquina limpia
- [ ] Seeds configurados para reproducibilidad
- [ ] Un comando ejecuta el pipeline completo

Testing

- [ ] Tests unitarios para funciones principales
- [ ] Coverage > 70%
- [ ] Tests de integración para API
- [ ] Tests pasan localmente y en CI

Versionado

- [ ] Commits con mensajes descriptivos
- [ ] DVC configurado para datos
- [ ] MLflow tracking de experimentos
- [ ] Al menos 3 experimentos comparables

Documentación

- [ ] README completo con todas las secciones
- [ ] Model Card documentando el modelo
- [ ] Docstrings en funciones públicas
- [ ] Comentarios explicando decisiones no obvias

Presentación

- [ ] Video demo de 3-5 minutos
- [ ] Audio claro y entendible
- [ ] Demuestra funcionalidad principal
- [ ] Incluye resultados y métricas

Ejemplo de Evaluación

Proyecto: “Churn Predictor v1.0”

Criterio	Puntos	Comentarios
1. Calidad del Código	16/20	
- Estructura	4/5	Buena, falta separar configs
- Estilo	4/5	Black aplicado, algunas líneas largas
- Modularidad	4/5	Funciones bien separadas
- Errores	4/5	Logging básico implementado
2. Reproducibilidad	17/20	
- Dependencias	4/5	Versiones fijadas, falta lock
- Entorno	5/5	Docker funcional
- Pipeline	4/5	Makefile presente, DVC básico
- Determinismo	4/5	Seeds en modelo, falta en data split
3. Testing/CI	12/15	
- Cobertura	4/5	72% coverage
- Calidad tests	4/5	Buenos fixtures
- CI/CD	4/5	CI funcional, sin CD
4. Versionado	11/15	
- Git	4/5	Conventional commits
- DVC	3/5	Configurado pero sin usar consistentemente
- MLflow	4/5	5 experimentos documentados
5. Documentación	12/15	
- README	5/5	Completo y profesional
- Model Card	3/5	Básico, falta bias analysis
- Docstrings	4/5	Presentes en mayoría de funciones
6. Presentación	11/15	
- Video	4/5	Claro, 4 minutos
- Comunicación	4/5	Explica decisiones
- Profesionalismo	3/5	Algunos errores menores en demo
TOTAL	**79/100**	**Calificación: C+ (Básico-Competente)**

Recomendaciones de mejora: 1. Implementar CD a staging 2. Usar DVC pipeline para reproducibilidad 3. Completar Model Card con análisis de bias 4. Practica demo para evitar errores

---

Esta rúbrica está diseñada para proyectos MLOps profesionales

Índice | Ejercicios | Syllabus