Parte 8 — Capstones — Proyectos Integradores

4 clases · Parte 8 del programa

Parte 8 — Capstones — Proyectos Integradores

> *4 clases · bundle consolidado del currículo v3.*

Índice de clases

Clase 229 — Clase 229 — Capstone 1: problema tabular end-to-end (EDA, modelo, API, dashboard)

> *Parte: 8 — Capstones · Fuente: integrador de Partes 0-7 + Huyen, C. Designing Machine Learning Systems (O'Reilly, 2022) caps. 4-7 + Géron, A. Hands-On ML, 3ª ed. caps. 2-3. Duración estimada: 180 min.*

Objetivo

Integrar en un único proyecto entregable todo lo aprendido en Partes 0-7: cargar un dataset tabular real, hacer EDA, construir un pipeline ColumnTransformer reproducible, entrenar y tunear un modelo gradient-boosting con MLflow tracking, serializar el modelo, exponerlo vía FastAPI con Pydantic v2, construir un dashboard Streamlit con SHAP, y dejar todo bajo CI con GitHub Actions. La entrega debe poder reproducirse desde un clon limpio con uv sync y un docker compose up.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diseñar un pipeline ML end-to-end siguiendo el flujo de Huyen (Designing ML Systems): data → features → modelo → tracking → serving → monitoring.
- Tunear un gradient-boosting con Optuna (≥50 trials) y loguear cada run en MLflow con parámetros, métricas y artefactos.
- Exponer el modelo en FastAPI con schemas Pydantic v2 validados y latencia P95 < 200 ms.
- Construir un dashboard Streamlit con explicaciones SHAP por predicción + drift monitor (Evidently).
- Publicar una Model Card (Mitchell et al. 2019) con uso intencionado, métricas por subgrupo y limitaciones conocidas.

Fases del capstone

#	Fase	Duración	Entregable
1	EDA	25 min	notebooks/01_eda.ipynb + reporte ydata-pro
2	Feature engineering	25 min	src/features.py con ColumnTransformer repr
3	Modelo + tuning	40 min	src/train.py con Optuna 50 trials + MLflow

4	Tracking + Model Card	20 min	mlruns/ + MODEL_CARD.md
5	API FastAPI	35 min	src/api/main.py con /predict, /health, Ope
6	Dashboard + CI	35 min	app.py Streamlit con SHAP + .github/workfl

Definiciones y características

- Stack 2026 recomendado: uv (lock + venv) + Polars o pandas 2.x + scikit-learn 1.5+ + xgboost/lightgbm + optuna 3.x + mlflow 2.x + fastapi 0.115+ + pydantic 2.x + streamlit 1.x + shap + evidently 0.4+ + docker compose.
- Métricas clasificación: ROC-AUC (ranking), F1 al threshold elegido, log-loss (calibración), Brier score, PR-AUC si hay desbalance >10:1.
- Métricas regresión: RMSE, MAE, R^2, MAPE (cuidado con y≈0).
- Threshold selection: el modelo devuelve probabilidad; el threshold se elige por F1, Youden's J, o coste-beneficio. No usar 0.5 por default.
- MLflow run: una ejecución de entrenamiento con params, metrics, tags y artifacts (modelo + plots). Optuna integra vía MLflowCallback.
- FastAPI schema: BaseModel Pydantic v2 con Field(..., ge=0, le=100) para validar rangos; OpenAPI auto-generado en /docs.
- Streamlit dashboard: app reactiva en Python puro; cachear cargas pesadas con @st.cache_resource.
- Model Card (Mitchell 2019): documento corto con uso intencionado, datos de entrenamiento, métricas por subgrupo, consideraciones éticas, limitaciones.
- CI smoke test: GitHub Actions corre pytest + levanta la API en background + hace curl /health → debe responder 200.

Dataset / recursos

- Dataset sugerido (elegí uno): UCI Adult (clasificación binaria, 48K filas, mix num+cat), Telco Customer Churn (Kaggle, 7K filas, churn binario), o Ames House Prices (Kaggle, 1.5K filas, regresión).
- Librerías: pandas o polars, scikit-learn, xgboost/lightgbm, optuna, mlflow, fastapi, uvicorn, pydantic, streamlit, shap, evidently, joblib, pytest, httpx.
- Infra local: Docker + compose.yml con 3 servicios (mlflow, api, streamlit).

Ejercicios

1. EDA: cargar dataset, generar ydata-profiling HTML, identificar missing, outliers, balance de clases. Documentar 5 hallazgos en notebooks/01_eda.ipynb.
2. Pipeline FE: armar ColumnTransformer con OneHotEncoder(handle_unknown='ignore') para cat, StandardScaler para num, SimpleImputer para missing. Persistir el preprocessor.
3. Modelo + Optuna: entrenar baseline (LogisticRegression) y challenger (XGBClassifier/LGBMClassifier). Tunear con Optuna 50 trials maximizando ROC-AUC sobre validación. Loguear cada trial en MLflow.
4. API: en src/api/main.py, definir PredictRequest(BaseModel) con todas las features tipadas y PredictResponse(BaseModel) con probability y prediction. Cargar modelo en lifespan. Endpoint /predict + /health.
5. Dashboard: app.py Streamlit con (a) form input → llama a la API y muestra predicción, (b) plot SHAP waterfall de la última predicción, (c) Evidently report comparando producción vs training data.

Homework verificable

Entregar un repo público en GitHub con:

1. Estructura src/, notebooks/, tests/, MODEL_CARD.md, README.md, pyproject.toml lockeado con uv.

2. MLflow tracking con ≥3 runs registradas (baseline + 2 challengers) y artefactos (modelo, plots de ROC, calibration curve).
3. FastAPI con /predict y /health, schemas Pydantic v2 validados, OpenAPI en /docs, latencia P95 < 200 ms medida con locust o httpx.
4. Dashboard Streamlit con SHAP plot por predicción y screenshot en el README.
5. Model Card completa (uso intencionado, métricas por subgrupo si aplica, limitaciones).
6. CI GitHub Actions corriendo pytest + smoke test contra /health con la API levantada en el runner. Badge verde en el README.

Criterio de aceptación: clonar el repo desde cero → uv sync → docker compose up → API responde a /predict con un request válido → dashboard Streamlit accesible en localhost:8501 → CI verde en GitHub.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Modelo tunea AUC=0.99 en validación, 0.65	Data leakage — features computadas con inf
ROC-AUC perfecto pero el modelo es inútil	Target leakage — alguna feature es proxy d
422 Unprocessable Entity al llamar /predic	FastAPI sin schema Pydantic o tipos mal de
Latencia /predict > 1 segundo	Cargás el modelo en cada request. Fix: car
Streamlit re-entrena modelo en cada intera	No cacheaste. Fix: @st.cache_resource para
MLflow run sin artefactos, solo métricas	Olvidaste mlflow.sklearn.log_model(model,

Preguntas frecuentes

¿XGBoost o LightGBM?

Cualquiera. LightGBM suele ser 2-5× más rápido en entrenamiento sobre datasets >100K filas; XGBoost tiene mejor soporte en producción legacy. Para el capstone: elegí uno y justificalo en el README.

¿Necesito GPU?

No. Datasets tabulares <1M filas entrenan en CPU en minutos con XGBoost/LGBM. Reservá GPU para deep learning (Parte 7).

¿FastAPI o Flask?

FastAPI. Pydantic v2 da validación automática + OpenAPI gratis + async nativo. Flask sigue siendo válido pero pedís el doble de boilerplate.

¿Cómo pruebo la API en CI sin levantar Docker?

pytest + httpx.AsyncClient(app=app) — testea la app FastAPI in-process, sin red. Smoke test real con Docker corre en otro job del workflow.

¿La Model Card es opcional?

No. Es entregable obligatorio. 1-2 páginas Markdown con: propósito, datos, métricas por subgrupo (género/edad/región si aplica), limitaciones conocidas, contacto. Mitchell et al. 2019 tiene el template.

Referencias

- Huyen, C. Designing Machine Learning Systems (O'Reilly, 2022) — caps. 4 (Training Data), 5 (Feature Engineering), 7 (Model Deployment).
- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow, 3ª ed. (O'Reilly, 2022) — caps. 2-3, el end-to-end project canónico.

- MLflow Tracking — runs, params, metrics, artifacts.
- FastAPI tutorial — Pydantic v2 + OpenAPI auto.
- Streamlit docs — cache_resource, cache_data, layout.
- Mitchell, M. et al. Model Cards for Model Reporting (FAT* 2019) — el template oficial.

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 230 — Clase 230 — Capstone 2: NLP o series de tiempo end-to-end

> *Parte: 8 — Capstones · Fuente: Hyndman & Athanasopoulos, Forecasting: Principles and Practice (3ª ed.) + Hugging Face NLP course. Duración estimada: 180 min.*

Objetivo

Entregar un segundo capstone end-to-end eligiendo una de dos ramas: (A) NLP — clasificación de texto / NER / RAG con transformers y endpoint FastAPI, o (B) series de tiempo — forecasting con baselines + modelos modernos, backtesting honesto e intervalos de predicción. En ambas ramas: tracking con MLflow, contenedorización con Docker, reproducibilidad con uv lock + seeds, Model Card, y discusión de drift (texto o forecast).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Elegir entre NLP o series de tiempo según interés/dominio y justificar la decisión en el README del proyecto.
- Construir un pipeline reproducible con split honesto (estratificado en NLP, temporal sin shuffle en series).
- Reportar métricas del dominio: F1/accuracy + slice analysis (NLP) o sMAPE/MASE + pinball loss (series).
- Servir el modelo en un endpoint FastAPI dockerizado, con MLflow tracking del entrenamiento.
- Detectar drift (Evidently text drift en NLP; KS sobre residuos en series) y documentarlo en la Model Card.

Fases del capstone

#	Fase	Entregable
1	Definición + dataset + Model Card v0	README.md del proyecto con problema, métri
2	EDA específico del dominio	NLP: distribución de longitudes, balance d
3	Split honesto + baselines	NLP: TF-IDF + Logistic / clase mayoritaria
4	Modelo moderno + MLflow tracking	NLP: fine-tune DistilBERT o sentence-trans
5	Backtesting + intervalos	NLP: slice analysis (longitud, idioma, cla
6	Empaquetado + endpoint + drift	Docker + FastAPI (/predict o /forecast) +

Definiciones y características

- sMAPE (symmetric MAPE): mean(2·|y−ŷ| / (|y|+|ŷ|)). Acotada en [0, 2], no estalla cuando y=0 (a

diferencia de MAPE).
- MASE (Mean Absolute Scaled Error): error escalado por el error del naive estacional in-sample. MASE<1 mejor que naive; comparable entre series de distinta escala.
- Pinball loss (quantile loss): max(τ·(y−ŷ), (τ−1)·(y−ŷ)). Loss correcta para predicción de cuantiles (intervalos P10/P90).
- Backtesting expanding window: train[0:T] → predict[T:T+h], luego train[0:T+h] → predict[...], etc. No re-mezcla; respeta el orden temporal.
- Slice analysis (NLP): medir métricas por subconjunto (longitud del texto, idioma, clase) — un F1 global de 0.92 puede esconder F1=0.40 en textos largos.
- Fine-tuning ligero: entrenar solo la cabeza (head) de un transformer pre-entrenado o aplicar LoRA (adapters de bajo rango) → 10–100× menos parámetros entrenables.
- RAG (Retrieval-Augmented Generation): embeddings → vector index (FAISS) → recuperar top-k → LLM responde con contexto. Reduce alucinaciones, no requiere fine-tuning.
- Drift de forecast: residuos y−ŷ cuya distribución cambia en el tiempo — señal de que el modelo necesita re-entrenamiento.

Dataset / recursos

- Rama A — NLP: IMDB reviews (50K, binario), AG News (120K, 4 clases), o reseñas de Yelp en español. Para RAG: Wikipedia dump pequeño o docs internos.
- Rama B — Series: M5 (Walmart, jerárquica), electricity load (UCI), o clima diario (NOAA). Mínimo 2 años con estacionalidad clara.
- Stack común: uv (Parte 7), Docker, MLflow, FastAPI, Evidently. NLP extra: transformers, datasets, sentence-transformers, faiss-cpu. Series extra: statsmodels, statsforecast, darts o neuralprophet.

Ejercicios

1. Definición: escribir el problema en 5 líneas (qué se predice, para qué, métrica primaria, baseline aceptable, costo de errar). Elegir rama A o B.
2. EDA del dominio: rama A → distribuciones de longitud y balance de clases por idioma; rama B → STL + ACF/PACF + test de estacionariedad ADF.
3. Baselines: rama A → TF-IDF + LogisticRegression; rama B → naive + seasonal naive + ETS. Loggear todo a MLflow.
4. Modelo moderno: rama A → fine-tune DistilBERT 2 epochs con transformers.Trainer; rama B → SARIMA o NeuralProphet con backtesting expanding window 5 folds.
5. Endpoint + drift: FastAPI con /predict (NLP) o /forecast?horizon=14 (series), dockerizado. Reporte de drift entre primera y segunda mitad del test (Evidently o KS test manual).

Homework verificable

Repositorio con:

1. README.md del proyecto con problema, rama elegida (A o B), métrica primaria y resultado.
2. Notebook de EDA + entrenamiento con seeds fijas (42) y MLflow tracking visible (screenshot o mlruns/ versionado).
3. Dockerfile + compose.yml que levanten FastAPI + MLflow UI con un solo docker compose up.
4. Endpoint funcional: curl -X POST localhost:8000/predict (NLP) o curl localhost:8000/forecast?horizon=14 (series) devuelve JSON válido.
5. Model Card (1 página) con métricas globales y por slice/horizonte, datos de entrenamiento, sesgos conocidos, y plan de monitoreo de drift.

Criterio de aceptación: el modelo moderno supera al mejor baseline en la métrica primaria (NLP: ≥ +3 puntos

de F1; series: MASE < 1.0 y sMAPE menor que seasonal naive), el endpoint responde en <500 ms, y la Model Card identifica al menos un slice/horizonte donde el modelo es débil.

Errores comunes

Síntoma / mens	Causa y cómo a						
sMAPE = 200%	y=0 con ŷ>0. Fix	y−ŷ	/(	y	+	ŷ	+ε)`, o cambiar a MASE.
Modelo de forec	Hiciste train_test						
transformers no	Falta caché o no						
F1 global alto pe	No hiciste slice a						
Intervalos P10/P	Modelo subestim						
OOM al fine-tune	Batch size dema						

Preguntas frecuentes

¿NLP o series? ¿Cuál es más fácil?

Series suele ser más rápido de prototipar (statsforecast entrena SARIMA sobre miles de series en minutos, sin GPU). NLP con transformers requiere GPU para fine-tuning serio, o aceptar fine-tuning lento en CPU con DistilBERT y max_length=128.

¿Puedo usar GPT-4/Claude vía API en vez de fine-tunear?

Sí — es válido como baseline en RAG o clasificación zero-shot, pero el capstone pide al menos un modelo propio entrenado y versionado en MLflow. La llamada a API puede ser el comparativo.

¿Por qué MASE además de sMAPE?

Porque MASE es comparable entre series (escala-invariante vs el naive estacional). Si reportás solo sMAPE, no podés saber si 12% es bueno o malo sin contexto. MASE < 1 mejor que naive, en cualquier serie.

¿FastAPI o BentoML?

FastAPI por default (ya cubierto en Parte 4). BentoML si necesitás batch inference automática, model registry integrado, o despliegue a SageMaker/Vertex con menos código.

¿La Model Card es realmente necesaria?

Sí — Mitchell et al. (Google, 2019) la propusieron por una razón: sin ella, nadie sabe qué slices son débiles ni cuándo el modelo se rompe. Es lo primero que un evaluador externo (o auditor) pide.

Referencias

- Hyndman, R., Athanasopoulos, G. Forecasting: Principles and Practice (3ª ed., 2021) — https://otexts.com/fpp3/
- Hugging Face NLP course — https://huggingface.co/learn/nlp-course
- Hvitfeldt, E., Silge, J. Supervised ML for Text Analysis (2021) — https://smltar.com/ (R, conceptos aplican)
- Statsforecast docs — https://nixtlaverse.nixtla.io/statsforecast/index.html
- Darts (forecasting deep + clásico) — https://unit8co.github.io/darts/
- Mitchell, M. et al. Model Cards for Model Reporting (FAT* 2019).

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.

- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 231 — Clase 231 — Capstone 3: visión por computadora con transfer learning

> *Parte: 8 — Capstones · Fuente: Géron, Hands-On ML 3ª ed. cap. 14-15 + timm + PyTorch Lightning docs. Duración estimada: 180 min.*

Objetivo

Construir un clasificador de imágenes de calidad producción usando transfer learning sobre un backbone moderno (ConvNeXt-tiny / EfficientNetV2-S / ViT-Base/16). Entrenar en dos fases (feature extraction → fine-tuning progresivo), aplicar augmentation moderna (RandAugment, MixUp, CutMix), evaluar con métricas per-clase + slice analysis, y servir vía ONNX + FastAPI con endpoint /predict que recibe imagen en base64. Cerrar el capstone con un fairness check si aplica.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diseñar un pipeline de transfer learning completo: backbone preentrenado en ImageNet → head custom → fine-tuning progresivo con LR diferencial por grupo.
- Aplicar augmentation moderna con Albumentations (RandAugment, MixUp, CutMix, RandomErasing) y verificar invariancia de label.
- Entrenar con PyTorch Lightning + AMP (mixed precision) + torch.compile + grad accumulation, con seed_everything y deterministic algorithms.
- Reportar accuracy + per-class F1 + confusion matrix + slice analysis (Clase 169) y un fairness check (Clase 224) si el dataset tiene atributos sensibles.
- Exportar el modelo a ONNX o TorchScript y servirlo en un endpoint FastAPI /predict que reciba imagen base64.

Fases del capstone

#	Fase	Por qué importa
1	Dataset + EDA	Class imbalance, resolución variable, leak
2	Augmentation	RandAugment + MixUp suben 2-4 puntos de ac
3	Backbone preentrenado	ConvNeXt-tiny / EffNetV2-S / ViT-B/16 vía
4	Fine-tuning progresivo	Feature extraction (head only) → unfreeze
5	Evaluación	Accuracy global engaña con imbalance; per-
6	Serving	Export a ONNX / TorchScript + FastAPI /pre

Definiciones y características

- Transfer learning: reutilizar pesos de un modelo entrenado en una tarea grande (ImageNet, 1.2M imágenes, 1000 clases) como punto de partida para tu tarea chica. Dos modos: feature extraction (congelar backbone, entrenar solo head) y fine-tuning (descongelar progresivamente).
- Backbone moderno (2026): ConvNeXt-tiny (28M params, CNN moderna), EfficientNetV2-S (21M, balance speed/accuracy), ViT-Base/16 (86M, attention puro). Todos disponibles en timm con pesos ImageNet-21k.
- LR diferencial por grupo: el backbone preentrenado necesita LR ~100× menor que el head random.

torch.optim.AdamW([{'params': backbone, 'lr': 1e-5}, {'params': head, 'lr': 1e-3}]).

- RandAugment: política automática que aplica N=2 transformaciones random con magnitud M=9 (rotación, color jitter, equalize, shear...). Plug-and-play en torchvision.transforms.v2.
- MixUp / CutMix: combinan 2 imágenes (MixUp = blend pixel-wise α≈0.2; CutMix = pega un parche de B sobre A). Labels se mezclan en la misma proporción. Regularizador potente para datasets chicos.
- AMP (Automatic Mixed Precision): usar float16 para forward/backward, float32 para pesos. 2× speedup y 2× memoria libre en GPUs ≥ Volta. torch.amp.autocast() + GradScaler.
- torch.compile: en torch 2.x, JIT-compila el grafo del modelo. +20-40% throughput sin tocar código. model = torch.compile(model, mode="reduce-overhead").
- Slice analysis: accuracy/F1 desagregado por sub-grupo (tamaño del objeto, iluminación, demografía). Revela disparidades que la métrica global esconde (Clase 169 + 224).

Dataset / recursos

- Opciones de dataset (elegí una):
- Cats vs Dogs (Kaggle, ~800 MB, 25K imágenes, binario) — el clásico para empezar.
- Food-101 subset (10 clases de 101, ~500 MB) — multiclase realista, fondos heterogéneos.
- Plant Disease (PlantVillage, 38 clases, ~1.5 GB) — class imbalance natural, alto impacto agro.
- Casting Defect (Kaggle industrial, binario, ~100 MB) — defect detection, dataset chico (~7K) ideal para mostrar el valor del transfer learning.
- Stack: torch 2.x · torchvision · timm · pytorch-lightning · albumentations · onnx · onnxruntime · fastapi · uvicorn · wandb o mlflow.

Ejercicios

1. Baseline tonto: entrenar una CNN de 2 capas conv from scratch (sin transfer). Reportar accuracy de validation. Esperá <60% en multiclase — establece el piso.
2. Feature extraction: cargar timm.create_model('convnext_tiny', pretrained=True, num_classes=N). Congelar backbone (for p in model.parameters(): p.requires_grad = False), entrenar solo head 5 epochs. Esperá +20-30 puntos sobre el baseline.
3. Fine-tuning progresivo: unfreeze último bloque → 5 epochs con LR 1e-4 → unfreeze full → 10 epochs con LR diferencial (1e-5 backbone, 1e-3 head). Loggear con W&B/MLflow.
4. Augmentation ablation: comparar (a) sin aug, (b) flip + crop, (c) RandAugment, (d) RandAugment + MixUp + CutMix. Reportar curva val_acc.
5. Serving end-to-end: exportar a ONNX (torch.onnx.export), validar con onnxruntime que la inferencia da la misma probabilidad ±1e-4, levantar FastAPI con endpoint POST /predict que reciba {"image_b64": "..."} y devuelva {"class": "...", "prob": 0.94}. Probar con curl.

Homework verificable

Notebook + repo con:

1. Dataset elegido + EDA (distribución de clases, resolución, ejemplos por clase).
2. Pipeline Lightning con 3 fases de entrenamiento (feature extraction → unfreeze parcial → unfreeze full).
3. Augmentation con Albumentations + verificación de label invariance.
4. Reporte de métricas: accuracy global, per-class F1, confusion matrix, slice analysis sobre al menos 1 dimensión.
5. Si el dataset tiene atributos sensibles (rostros, demografía): fairness check con disparate impact + equal opportunity (Clase 224).
6. Export a ONNX + script serve.py con FastAPI /predict funcionando localmente.
7. README del repo con resultados, decisiones de diseño y limitaciones.

Criterio de aceptación: accuracy de test > 90% en binario o > 75% en multiclase (10+ clases), el endpoint /predict responde en <500 ms en CPU, y el reporte identifica al menos 1 slice donde el modelo subperforma.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Accuracy de train sube a 99%, val se queda	Overfitting clásico. Fix: subir augmentati
CUDA out of memory al hacer fine-tune full	Batch demasiado grande para AMP off. Fix:
Val accuracy es altísima pero el modelo en	Leakage train/test (mismas imágenes duplic
RuntimeError: shape mismatch al exportar a	El head custom espera shape que el dummy_i
Predicciones ONNX difieren de PyTorch en 5	Esperable (precisión float). Fix: validar
Una clase tiene F1=0.0 y nadie lo notó	Class imbalance (esa clase es 2% del datas

Preguntas frecuentes

¿ConvNeXt, EfficientNetV2 o ViT?

Para datasets chicos (<10K imágenes), ConvNeXt-tiny o EfficientNetV2-S ganan: las CNN tienen inductive bias (locality, translation equivariance) que ViT no tiene y por eso ViT necesita 10× más datos para igualar. Para datasets grandes (>100K) o si tenés pretraining en ImageNet-21k, ViT-Base/16 suele ser top. Probá los 3 vía timm y comparalos — son 3 líneas de código cada uno.

¿Feature extraction o fine-tuning?

Feature extraction primero (más rápido, menos riesgo de catastrophic forgetting). Si la accuracy se estanca, hacer fine-tuning progresivo: descongelar el último bloque, después dos, después full, siempre con LR diferencial. Nunca descongelés todo en epoch 1 — el gradiente random del head corrompe los pesos preentrenados.

¿Por qué Lightning y no PyTorch puro?

Lightning te da gratis: AMP, multi-GPU, gradient accumulation, checkpointing, EarlyStopping, logging a W&B/MLflow, y seed_everything. En un capstone querés invertir el tiempo en el modelo, no en el training loop.

¿ONNX o TorchScript para serving?

ONNX si querés portabilidad (servir desde C++, Java, Node, navegador con onnxruntime-web). TorchScript si te quedás en Python/C++ con torch instalado y querés el path más simple. Para FastAPI en producción, ONNX + onnxruntime suele ser 2-3× más rápido que torch eager.

¿Fairness check siempre?

Solo si el dataset tiene atributos sensibles (rostros con edad/género/etnia, datos médicos con demografía). Si clasificás defectos industriales o platos de comida, no aplica. Cuando aplica, es obligatorio — releé Clase 224 antes de publicar el modelo.

Referencias

- Géron, A. Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow 3ª ed. (O'Reilly, 2022), cap. 14 (CNN) y 15 (transfer learning).
- timm — PyTorch Image Models (Ross Wightman) — 1000+ backbones preentrenados con API única.
- PyTorch Lightning docs — el training loop estándar de facto.
- Albumentations docs — augmentation rápida y composable.

- He, K. et al. Deep Residual Learning for Image Recognition (CVPR 2016). <https://arxiv.org/abs/1512.03385>
- Dosovitskiy, A. et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (ICLR 2021, ViT). <https://arxiv.org/abs/2010.11929>

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Clase 232 — Clase 232 — Portafolio público en GitHub Pages y presentación

> *Parte: 8 — Capstones · Fuentes: Chip Huyen, How to build a data science portfolio + Eugene Yan, What I learned from looking at 200 ML portfolios + Mitchell et al., Model Cards for Model Reporting (FAT* 2019). Duración estimada: 150 min.*

Objetivo

Empaquetar los tres capstones (229, 230, 231) en un portafolio público que un reclutador entienda en 30 segundos: sitio en GitHub Pages, demos hosted, blog técnico, deck de presentación y CV. La regla es calidad > cantidad: 3 proyectos terminados con métricas honestas valen más que 10 a medias.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Estructurar un sitio de portafolio con MkDocs Material o Quarto desplegado en GitHub Pages.
- Escribir un README de proyecto que comunica problema, approach, métricas y limitaciones sin marketing vacío.
- Publicar demos hosted (Streamlit Community Cloud, HuggingFace Spaces) y un blog técnico por capstone.
- Armar un deck de 10-15 slides que cuenta cada proyecto en formato problema → datos → enfoque → resultado → trade-offs.
- Detectar y evitar los antipatterns clásicos (10 proyectos a medias, README sin números, demo rota).

Componentes del portafolio

#	Componente	Para qué sirve
1	README pulido por proyecto	Es la primera impresión. Badge de CI, hero
2	Demos hosted (Streamlit/HF Spaces)	El reclutador clickea y prueba en 10 seg.
3	Blog técnico (1 post por capstone)	Demuestra que sabés comunicar además de co
4	Presentación (deck 10-15 slides)	Para entrevistas técnicas y meetups. Reusa
5	Video demo (2-3 min)	Loom o screencast. Único formato consumibl
6	CV técnico (1 página)	Verbo de impacto + número medible. Linkea

Definiciones y características

- Portafolio: sitio público (no PDF) que centraliza proyectos, blog, CV y links de contacto. Vive en username.github.io o subdominio propio.
- GitHub Pages: hosting estático gratis de GitHub. Build con Jekyll por defecto, pero podés deployar

cualquier sitio estático (MkDocs, Quarto, Docusaurus, Hugo).
- MkDocs Material: generador estático en Python; búsqueda built-in, dark mode, navegación lateral. Lo más rápido de levantar.
- Quarto: generador científico (Posit/RStudio); ejecuta notebooks .qmd/.ipynb y renderiza con bibliografía BibTeX y matemáticas LaTeX. Ideal si tu blog tiene mucho código + plots.
- Model Card (Mitchell et al., 2019): ficha estándar de un modelo — datos de entrenamiento, métricas por subgrupo, casos de uso previstos y NO previstos, riesgos éticos.
- Hero image / GIF: imagen o screencast al inicio del README que muestra el output en 1 vistazo. Sin hero, nadie scrollea.
- Quick start de 30 segundos: bloque bash con 3-5 líneas para correr el proyecto local. Si no entra en 30 seg, perdiste al reviewer.
- Reproducibilidad (Parte 7): lock file + Dockerfile + CI verde + dataset documentado. Es el diferenciador entre "proyecto de tutorial" y "proyecto serio".

Recursos

- Herramientas: MkDocs Material, Quarto, Streamlit Community Cloud, HuggingFace Spaces, Render, Vercel.
- Templates: mkdocs-material starter, Quarto blog template.
- Inspiración: portfolios de Eugene Yan, Chip Huyen, Vicki Boykis, Hamel Husain — todos en eugeneyan.com, huyenchip.com, etc.

Ejercicios

1. MkDocs Material setup: pip install mkdocs-material, mkdocs new portfolio, editar mkdocs.yml con tema material y deployar a gh-pages con mkdocs gh-deploy. Verificar que el sitio carga en https://<user>.github.io/portfolio.
2. README de capstone: tomar el capstone 229 (NLP) y reescribir el README con hero image, badges (CI, license, Python version), quick start de 30 seg, sección "decisiones técnicas" (3 bullets) y "limitaciones" (3 bullets).
3. Demo hosted: subir el capstone 230 (CV) a HuggingFace Spaces con Gradio. URL pública compartible.
4. Blog post técnico: escribir 1 post de 800-1500 palabras sobre el capstone 231 (tabular) siguiendo el outline problema → datos → approach → resultado con un plot → trade-offs → próximos pasos.
5. Deck: armar 10-15 slides (Google Slides, Marp o reveal.js) presentando los 3 capstones con la regla de "1 idea por slide".

Homework verificable

Sitio en GitHub Pages con:

1. Index con foto, bio de 3 líneas, links a 3 proyectos, link a blog, link a CV PDF, link a LinkedIn/GitHub.
2. Una página por proyecto (3 capstones) con README + Model Card + link a demo + link a blog post + link a repo.
3. Blog con al menos 1 post técnico publicado.
4. CV PDF de 1 página linkeado desde el index.
5. Demos: al menos 1 de los 3 proyectos con demo hosted funcionando (Streamlit Cloud o HF Spaces).

Criterio de aceptación: un reclutador no técnico abre el sitio, en 30 segundos sabe (a) tu nombre, (b) qué hacés, (c) ve un proyecto con métricas, (d) puede clickear una demo que funciona. Si falla algún paso → el portafolio no aprueba.

Errores comunes

Síntoma	Causa y cómo arreglar
README dice "implementé un modelo de ML pa	Cero métricas, cero credibilidad. Fix: pon
10 proyectos en el portafolio, todos a 30%	Cantidad sin profundidad. Fix: borrar 7, d
Demo rota (link a Streamlit que duerme y n	Streamlit free duerme tras 7 días sin uso,
Hero image inexistente — README es solo te	Sin imagen no scrollean. Fix: GIF de 5 seg
requirements.txt con pandas sin versión	Imposible reproducir. Fix: lock file (uv.l
CV de 3 páginas con responsabilidades gené	Nadie lee la pág 2. Fix: 1 página, verbos

Preguntas frecuentes

¿MkDocs Material o Quarto?

MkDocs Material si el sitio es documentación + landing. Quarto si el sitio tiene mucho notebook ejecutable y math (papers, posts técnicos largos con BibTeX). Ambos deployan a GitHub Pages igual.

¿Necesito dominio propio?

No para empezar. username.github.io funciona perfecto. Comprá dominio (tunombre.dev, ~12 USD/año) cuando el portafolio esté en versión estable y quieras una URL más memorable.

¿Y si no tengo experiencia laboral todavía?

El portafolio reemplaza la falta de experiencia laboral. 3 capstones con métricas, demo y blog post pesan más en una entrevista junior que "tomé 5 cursos en Coursera".

¿Subo el código del homework de las clases?

No al portafolio principal. Mantenelos en un repo aparte (python-data-science-program-homework). El portafolio es solo para proyectos pulidos.

¿Cuánto tiempo dedicar al portafolio antes de aplicar a trabajos?

Regla práctica: 1 semana de pulido full-time tras terminar el capstone 231. Pasada esa semana, aplicás aunque no sea perfecto — el portafolio se itera con feedback real de entrevistas.

Referencias

- Chip Huyen, How to build a data science portfolio — huyenchip.com.
- Eugene Yan, What I learned from looking at 200 ML portfolios — eugeneyan.com.
- MkDocs Material — documentación oficial.
- Quarto — sitio oficial, especialmente Websites y Blogs.
- HuggingFace Spaces — hosting gratis para demos.
- Mitchell, M. et al. Model Cards for Model Reporting (FAT* 2019) — arxiv:1810.03993.

Cierre del programa

Llegaste a la clase 232 de 232. Empezaste con print("hola mundo") y terminás con un portafolio público, tres capstones con métricas honestas, demos hosted y un blog que explica cómo pensás. El programa termina acá; tu trabajo como practicante de data science recién arranca. Lo que sigue no es otro curso — es elegir una vertical (research, applied science, MLE, product DS), buscar a la gente que está haciendo el trabajo que querés hacer, y empezar a contribuir: issues en open source, posts en tu blog, charlas en meetups, un proyecto nuevo cada trimestre. La ventaja competitiva no es saber más algoritmos: es enviar cosas terminadas, medibles y reproducibles. Eso ya lo sabés hacer.

Gracias por llegar hasta acá. Ahora andá a romperla.

Volver al índice general · Índice de Parte 8

Material descargable

- Guía explicativa (PDF) — versión imprimible con todo el contenido de la clase.
- Presentación (PPTX) — deck PowerPoint listo para proyectar en clase.
- Notebook ejecutable (.ipynb) — abrilo desde el laboratorio del programa o desde Jupyter.

Cierre de la parte

Fin del bundle consolidado de Parte 8 — Capstones — Proyectos Integradores · 4 clases.