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.

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).

Siguiente clase

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

Apéndice: notebook (primer bloque)

Este capstone tiene dos ramas opt-in: - Rama A — NLP: clasificación de texto / NER / RAG mini con transformers + FastAPI. Descrita en el README; requiere descargar pesos de Hugging Face. - Rama B — Series de tiempo: forecasting con baselines + ML + intervalos. Implementada aquí porque corre 100% offline con numpy/pandas/sklearn (+ statsmodels opcional).

```
import numpy as np, pandas as pd
from sklearn.linear_model import Ridge, QuantileRegressor
from sklearn.metrics import mean_absolute_error
from scipy import stats

rng = np.random.default_rng(42)

# Serie sintetica: 3 anios diarios = 1095 puntos
n_days = 365 * 3
t = np.arange(n_days)
trend   = 100 + 0.05 * t
weekly  = 8  np.sin(2  np.pi * t / 7)
yearly  = 15  np.sin(2  np.pi * t / 365.25 - np.pi / 2)
noise   = rng.normal(0, 3, n_days)
y = trend + weekly + yearly + noise

dates = pd.date_range('2023-01-01', periods=n_days, freq='D')
df = pd.DataFrame({'date': dates, 'y': y}).set_index('date')
print(f'serie: {df.shape[0]} dias | rango {df.index.min().date()} -> {df.index.max().date()}')
print(df.head())
```

Archivos complementarios

- notebook.ipynb