Clase 207 — Testing de modelos: invariance + behavioral tests

Parte: 4 — MLOps · Fuente: Ribeiro et al., Beyond Accuracy: Behavioral Testing of NLP Models with CheckList (ACL 2020, best paper) + Huyen cap. 9 + Deepchecks docs. Duración estimada: 80 min.

Clase 207 — Testing de modelos: invariance + behavioral tests

> *Parte: 4 — MLOps · Fuente: Ribeiro et al., Beyond Accuracy: Behavioral Testing of NLP Models with CheckList (ACL 2020, best paper) + Huyen cap. 9 + Deepchecks docs. Duración estimada: 80 min.*

Objetivo

Ir más allá de "accuracy en hold-out": tests que verifican que el modelo se comporta como debería en casos específicos. Tres familias: invariance tests ("misma predicción si reemplazo Juan por María"), directional tests ("si subo el ingreso, la proba de aprobar el préstamo no debe bajar"), minimum functionality tests ("predicción correcta sobre casos canónicos hand-crafted").

Cierra la Parte 4: con esto, el modelo en producción tiene 6 capas de protección (data tests, model tests, monitoring, shadow, canary, rollback).

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Implementar invariance tests (perturbaciones que NO deben cambiar la predicción): swap de nombres protegidos, sinónimos, ruido pequeño.
- Implementar directional tests (perturbaciones que deben cambiar la predicción en una dirección esperada): subir ingreso → menos riesgo crediticio.
- Crear minimum functionality test sets (MFT): casos hand-crafted que cubren cada feature/segmento crítico.
- Integrar tests de modelo en pytest y correrlos en CI antes de promover (gate de Clase 197).
- Usar Deepchecks o CheckList para suites pre-armados (especialmente NLP).

Temas

#	Tema	Por qué importa
1	Accuracy ≠ corrección — los 3 tipos de tes	Modelo "bueno" puede tener bugs sistemátic
2	Invariance tests	Detecta sesgos (mismo CV con nombre cambia
3	Directional tests	Detecta inconsistencias (más experiencia →
4	MFT (Minimum Functionality)	Casos triviales que un humano resuelve sin
5	Slice-based testing	Performance por subgrupo, no solo overall.
6	Integración con CI	Tests verdes ≠ gate, requerir explícitamen

Definiciones y características

- Invariance test (INV): assert model(x) == model(perturb(x)). La perturbación es una transformación que no debería cambiar la predicción. Ej: cambiar nombre, agregar puntuación, sinónimos.
- Directional Expectation test (DIR): assert sign(model(x_modified) - model(x)) == expected_direction. Ej: si aumento income, P(default) debe bajar o quedar igual — nunca subir.
- Minimum Functionality Test (MFT): un set chico de ejemplos hand-crafted etiquetados manualmente. El

modelo DEBE acertar todos. Ej: "Pésimo servicio" → negativo en un sentiment classifier.
- Slice-based testing: medir métrica por sub-grupo (gender, age, country) y reportar el peor caso, no el promedio. El promedio esconde regresiones en minorías.
- CheckList (Ribeiro et al.): framework + paper que estandarizó los 3 tipos de test (INV/DIR/MFT) para NLP. Acabó ganando best paper en ACL 2020.
- Deepchecks: librería Python con suites pre-armados para tabular, vision, NLP. Detecta data leakage, train-test drift, performance bias.
- Adversarial testing: variante de invariance — perturbaciones pequeñas adversarialmente diseñadas que rompen el modelo. cleverhans, foolbox para deep learning.

Dataset / recursos

- Tabular: Adult Income (UCI) con gender como atributo sensible.
- NLP: subset de IMDb reviews para sentiment.
- Librerías: deepchecks>=0.18, checklist, pytest, pandas.

Ejercicios

1. MFT tabular: para un modelo de crédito sobre Adult, hand-craft 20 casos: 10 obvios "should approve" (alto ingreso, educación alta, sin debts) y 10 obvios "should reject" (ingreso bajo, edad joven, sin historial). Asseré con pytest que el modelo acierta los 20.
2. Invariance — gender swap: tomá 500 registros, cambiá gender M ↔ F, dejá el resto igual. Mediá accuracy(predictions_original == predictions_swapped). Si <99%: el modelo está usando gender (probablemente vía proxy).
3. Directional — income up: para los mismos 500, multiplicá income × 1.5. La proba predicha de >50K debería subir (o quedar igual) en >95% de los casos. Si baja en muchos: bug.
4. Slice-based: calculá accuracy por slice (gender × race × age_bucket). Reportá top 5 worst slices. Si el peor slice tiene accuracy 0.55 y el promedio 0.85: tenés un problema de fairness invisible en métricas agregadas.
5. Pytest gate: empaquetá los 4 tests anteriores en tests/test_model_behavior.py. Hacelo correr en GitHub Actions (Clase 197) como required check. Si tests rojos: PR no se mergea.

Homework verificable

Repo con:

1. tests/test_model_behavior.py con ≥4 tests: 1 MFT, 1 INV, 1 DIR, 1 slice-based.
2. CI workflow que corre pytest tests/test_model_behavior.py después de cada training.
3. deepchecks full suite ejecutado, reporte HTML guardado.
4. Reporte por slice (markdown) con métricas en gender, race, age_bucket y bottom-5 worst slices identificados.
5. Documentación de 3 bugs reales encontrados por estos tests durante el desarrollo (puede ser bug fixed; lo importante es demostrar que los tests sirven).

Criterio de aceptación: si en una iteración futuro alguien cambia el modelo y un test rompe (ej. gender swap accuracy cae de 0.99 a 0.85), el CI falla y el PR no se mergea sin discusión explícita.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Gender swap test "falla" pero el modelo no	Está usando un proxy (ocupación, código po
MFT pasa pero el modelo rinde mal en produ	El MFT está sesgado al subset que vos imag
Slice-based: el worst slice tiene n=12 y m	Sample size insuficiente para sacar conclu
Tests behavioral pasan todos siempre	Probablemente las perturbaciones son trivi
Directional test falla "raro"	Quizás la relación NO es monótona — ej: in
CI lento por tests	Si pytest tarda 20 min, nadie los corre lo

Preguntas frecuentes

¿Esto reemplaza al test set normal?

No, lo complementa. Hold-out test set mide performance estadística sobre distribución similar a training. Behavioral tests miden correctness sobre casos importantes hand-crafted. Necesitás ambos.

¿Cuántos behavioral tests son suficientes?

CheckList paper sugiere 1 MFT + 1 INV + 1 DIR por capability del modelo. En tabular: por cada feature crítica. En NLP: por cada fenómeno linguístico. Empezá con 10-20 tests bien diseñados, no con 1000 mediocres.

¿Esto es lo mismo que fairness testing?

Hay overlap. Fairness suele ser un subset de invariance tests (perturbar atributos protegidos) + slice-based (medir disparidad). Pero behavioral tests también cubren directional/MFT que no son sobre fairness directamente.

¿Deepchecks o escribir tests propios?

Deepchecks ofrece checks pre-armados (data leakage, integrity, performance bias) que cubren casos genéricos. Para tu dominio específico (correctness de negocio), escribís tests custom. Usar ambos.

¿Cómo manejo tests probabilísticos (no determinísticos)?

(1) Fijar seeds. (2) Para tests aproximados: assert metric > 0.99 (no == 1.0). (3) Bootstrap CI para asseré lower_bound > threshold.

¿Y modelos LLM?

Aplican igual los 3 tipos:

- MFT: prompts canónicos con respuestas esperadas exactas/reguladas.
- INV: paráfrasis del mismo prompt no debería cambiar el output dramáticamente.
- DIR: agregar "be brief" al prompt debería acortar output.

Frameworks: promptfoo, LangSmith, Phoenix (Arize).

Referencias

- Ribeiro, M. et al. Beyond Accuracy: Behavioral Testing of NLP Models with CheckList (ACL 2020) — el paper que estableció el vocabulario.
- Huyen, Chip. Designing Machine Learning Systems (O'Reilly, 2022), cap. 9 — Continual Learning and Test in Production.
- Deepchecks docs — suites para tabular/vision/NLP.

- CheckList GitHub — framework de Ribeiro.
- promptfoo / LangSmith — equivalente para LLMs.

Siguiente clase

Clase 208 — Bases de datos relacionales: PostgreSQL para ML

> *Fin de la Parte 4 — MLOps. Tenés ahora 6 capas de protección para modelos en producción: data tests (206), model tests (207), monitoring (202), shadow (204), canary (204), rollback (204). El próximo módulo (Parte 5) entra a ingeniería de datos — qué pasa antes de que la data llegue al modelo.*

Apéndice: notebook (primer bloque)

Implementamos los 4 tipos sobre un modelo de income prediction (Adult dataset proxy con California Housing modificado para el ejemplo).

```
import numpy as np, pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

data = fetch_california_housing(as_frame=True)
df = data.data.copy()
# Convertimos a clasificación binaria: 'high income block' = target > mediana
df['high'] = (data.target > data.target.median()).astype(int)
# Sintetizamos 'gender' aleatorio (sin info real → modelo NO debería usarlo)
rng = np.random.default_rng(42)
df['gender'] = rng.choice([0, 1], size=len(df))

FEATURES = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude', 'gender']
Xtr, Xte, ytr, yte = train_test_split(df[FEATURES], df['high'], test_size=0.3, random_state=42)
model = RandomForestClassifier(n_estimators=200, random_state=42, n_jobs=-1).fit(Xtr, ytr)
print('test acc:', model.score(Xte, yte))
```

Archivos complementarios

- notebook.ipynb