Clase 228 — Reproducibilidad: seeds, lock files, versionado de datasets

Parte: 7 — Ética, Fairness y Privacidad · Fuente: Pineau et al., Improving Reproducibility in ML Research (JMLR 2021) + Gebru et al., Datasheets for Datasets (CACM 2021) + Mitchell et al., Model Cards for Model Reporting (FAT* 2019). Duración estimada: 75 min.

Clase 228 — Reproducibilidad: seeds, lock files, versionado de datasets

> *Parte: 7 — Ética, Fairness y Privacidad · Fuente: Pineau et al., Improving Reproducibility in ML Research (JMLR 2021) + Gebru et al., Datasheets for Datasets (CACM 2021) + Mitchell et al., Model Cards for Model Reporting (FAT* 2019). Duración estimada: 75 min.*

Objetivo

Cerrar la Parte 7 con el problema que atraviesa todo lo anterior: si un experimento no es reproducible, no es auditable, no es comparable y no es ciencia. Aprender a controlar las tres fuentes de no-determinismo (código, datos, ambiente) con seeds, lock files, hashes de datasets, model cards y manifiestos de pipeline. Entender por qué Hutson (Science, 2018) habló de "crisis de reproducibilidad" en ML y qué piden hoy NeurIPS/JMLR como mínimo.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Sembrar correctamente random, numpy, sklearn (y comentar torch) con una función seed_everything() y PYTHONHASHSEED.
- Distinguir requirements.txt (top-level) de un lock file (uv.lock, poetry.lock, conda-lock) que pinea toda la transitive tree.
- Calcular un hash estable de un DataFrame (sha256_of_df) que sobreviva a reorden de columnas e índice y sirva como dataset_id.
- Producir un manifest JSON con {data_hash, code_hash, seed, package_versions} y validar reproducibilidad antes de re-entrenar.
- Redactar una model card mínima (intended use, training data hash, metrics, limitations) según Mitchell et al. 2019.

Temas

#	Tema	Por qué importa
1	Fuentes de no-determinismo	GPU atomicAdd, dict/set ordering, num_work
2	Seeds en stack Python	random.seed, np.random.default_rng, torch.
3	Lock files vs requirements	uv.lock/poetry.lock pinean toda la transit
4	Ambiente reproducible	Docker + base image pineada por digest (py
5	Versionado de datasets	DVC, Git LFS, lakeFS, S3 versioning, hash
6	Documentación: datasheets + model cards	Gebru 2018 (dataset) y Mitchell 2019 (mode

Definiciones y características

- Determinismo: misma entrada + mismo código + mismo ambiente → mismo output bit-a-bit. En ML rara vez se logra al 100% sobre GPU; se aspira a reproducibilidad estadística.

- Seed: entero que inicializa el estado de un PRNG. Cambiar el seed cambia la trayectoria — reportar un solo seed sin promediar varios es una mala práctica (Pineau 2021).
- PYTHONHASHSEED: variable de entorno que controla el hash de strings/objetos. Desde Python 3.3, por default es aleatorio por proceso — afecta orden de iteración de set/dict con keys hashables custom.
- Float associativity: (a + b) + c ≠ a + (b + c) en float32/64 por redondeo. Sumar en otro orden (BLAS multi-threaded, GPU reductions) da resultados que difieren en 1 ULP — suficiente para divergir tras muchas iteraciones.
- Lock file: archivo que pinea la versión exacta de cada dependencia transitiva con hash del wheel. requirements.txt con pandas==2.2.0 no pinea numpy, pytz, etc. — uv.lock sí.
- DVC (Data Version Control): git para datos. Guarda un .dvc pointer en git (hash MD5 + tamaño) y el blob real en storage remoto (S3/GCS). dvc pull re-hidrata.
- Datasheet for Datasets (Gebru et al. 2018): documento que acompaña al dataset con motivación, composición, recolección, uses, distribución. Equivalente al spec sheet de un componente electrónico.
- Model Card (Mitchell et al. 2019): documento que acompaña a un modelo con intended use, factores, métricas desagregadas, training/eval data, ethical considerations, caveats.

Dataset / recursos

- Dataset sintético generado in-notebook (no externo) — la clase es sobre el proceso, no sobre el dato.
- Librerías: stdlib (hashlib, json, importlib.metadata, os, random), numpy, pandas, scikit-learn.
- Opcional para homework: DVC, uv.

Ejercicios

1. Seed everything: implementar seed_everything(seed=42) que cubra random, numpy y PYTHONHASHSEED. Verificar que np.random.rand(5) da el mismo vector en dos corridas consecutivas.
2. Hash de DataFrame: escribir sha256_of_df(df) que ordene columnas alfabéticamente, resetee el índice y serialice a CSV bytes antes de hashear. Mostrar que dos df con columnas en distinto orden dan el mismo hash.
3. Manifest de experimento: dict con {data_hash, code_hash, seed, sklearn_version, numpy_version, pandas_version, python_version} serializado a experiment.json.
4. Validación de reproducibilidad: dado un manifest guardado, re-leer el dataset, recomputar su hash, comparar — abortar con RuntimeError si difiere.
5. Model card mínima: función build_model_card(model, X, y, intended_use, limitations, date_trained) que devuelve dict con campos de Mitchell et al. y lo dumpea a JSON.

Homework verificable

Notebook (o script) que:

1. Construya un pipeline load → train → evaluate con seed_everything(42) al inicio.
2. Calcule dataset_hash con sha256_of_df y code_hash con sha256 sobre el .py (o el source string vía inspect.getsource).
3. Guarde manifest.json con data_hash, code_hash, seed, package_versions y model_card.json con métricas + limitations.
4. Re-corra el mismo pipeline en otra ejecución; verifique que accuracy reportada coincide bit-a-bit y que ambos hashes coinciden.

5. Cree requirements.lock con uv pip compile requirements.in -o requirements.lock (o pip freeze > requirements.lock como fallback) y commitee ambos al repo.

Criterio de aceptación: en dos máquinas distintas (o dos venvs limpios) con uv pip sync requirements.lock, el pipeline produce el mismo accuracy y el mismo dataset_hash. El manifest.json está commiteado.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Resultados cambian entre corridas a pesar	Hay otra fuente: random stdlib sin seedear
Hash del DataFrame cambia y los datos "son	Estás hasheando df.to_csv() directo — incl
requirements.txt con pandas==2.2.0 rompe e	No pinea numpy (transitive). Fix: usar uv.
Torch da resultados distintos en GPU con s	atomicAdd en GPU no es determinista. Fix p
Iteración sobre set da orden distinto cada	PYTHONHASHSEED aleatorio. Fix: setearla an
sum(arr) ≠ np.sum(arr) con float32	Float associativity + orden de reducción.

Preguntas frecuentes

¿Es realmente posible reproducir bit-a-bit un entrenamiento en GPU?

En general no sin esfuerzo. Hay ops no deterministas (atomicAdd, cuDNN convolutions con autotuner). Con torch.use_deterministic_algorithms(True) + CUBLAS_WORKSPACE_CONFIG + num_workers=0 + cudnn.deterministic=True te acercás, a costa de 1.5-3× más slow. En CPU es mucho más fácil.

¿requirements.txt no alcanza si pineo todo con ==?

No del todo. Aunque pinees pandas==2.2.0, pip puede resolver numpy a una versión distinta entre máquinas si el resolver tiene libertad. Un lock file congela el resultado del resolver con hashes de wheels. uv.lock, poetry.lock, Pipfile.lock, conda-lock son equivalentes funcionales.

¿DVC, Git LFS o S3 versioning?

- DVC: si tu workflow es git-céntrico y querés dvc repro (pipelines con cache por inputs). Storage backend a elección.
- Git LFS: si los archivos son binarios chicos (<2GB) y querés transparencia total con git. No tiene pipelines.
- S3 versioning: si ya vivís en AWS y solo necesitás "ver versiones anteriores del objeto". Sin metadata de pipeline.
- lakeFS / Quilt / Pachyderm: para escala enterprise con branching real sobre data lakes.

¿Hashear el .csv original o el DataFrame cargado?

El DataFrame normalizado (columnas ordenadas, sin índice). El .csv puede tener BOM, line endings distintos (\r\n vs \n), encoding distinto y romper el hash sin que el contenido cambie. Hashear post-parsing es semánticamente correcto.

¿Una model card es un trámite burocrático?

No si la usás como gate de release. Auditorías regulatorias (EU AI Act 2024, NIST AI RMF) ya exigen documentación equivalente. Una model card bien hecha es además la entrada del próximo experimento: "el baseline al que tengo que ganarle es éste, entrenado con estos datos, evaluado así".

Referencias

- Pineau, J. et al. Improving Reproducibility in Machine Learning Research (JMLR 2021) — el NeurIPS Reproducibility Checklist.
- Gebru, T. et al. Datasheets for Datasets (CACM 2021).
- Mitchell, M. et al. Model Cards for Model Reporting (FAT* 2019).
- Hutson, M. Artificial intelligence faces reproducibility crisis (Science, 2018).
- PyTorch Reproducibility notes — qué controla cada flag.
- DVC docs · uv (lock files modernos).

Siguiente clase

Clase 229 — Capstone 1 — Problema tabular end-to-end

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
import numpy as np

# Sin seed: distinto cada corrida
print('sin seed (1):', np.random.rand(5).round(4))
print('sin seed (2):', np.random.rand(5).round(4))

# Con seed: idéntico
rng_a = np.random.default_rng(42)
rng_b = np.random.default_rng(42)
print('con seed (a):', rng_a.random(5).round(4))
print('con seed (b):', rng_b.random(5).round(4))
```

Archivos complementarios

- notebook.ipynb