Clase 205 — Interpretabilidad: SHAP, LIME, PDP, ICE

Parte: 4 — MLOps · Fuente: Molnar, Interpretable ML (2ª ed.) + Lundberg & Lee (2017, NIPS, SHAP paper) + Ribeiro et al. (2016, KDD, LIME paper). Duración estimada: 80 min. · Esta clase es complementaria a la Clase 079 (SHAP profundo, Parte 1). Acá el foco es producción: cómo integrar interpretabilidad al servicio (/explain endpoint), cómo escalar SHAP a grandes datasets, y cómo presentar explicaciones a stakeholders no técnicos.

Clase 205 — Interpretabilidad: SHAP, LIME, PDP, ICE

> *Parte: 4 — MLOps · Fuente: Molnar, Interpretable ML (2ª ed.) + Lundberg & Lee (2017, NIPS, SHAP paper) + Ribeiro et al. (2016, KDD, LIME paper). Duración estimada: 80 min. · Esta clase es complementaria a la Clase 079 (SHAP profundo, Parte 1). Acá el foco es producción: cómo integrar interpretabilidad al servicio (/explain endpoint), cómo escalar SHAP a grandes datasets, y cómo presentar explicaciones a stakeholders no técnicos.*

Objetivo

Hacer interpretabilidad deployable: exponer un endpoint /explain que devuelva la atribución por feature de una predicción individual (SHAP/LIME), generar reportes globales (PDP/ICE) al promover un modelo, y entender los tres trade-offs: fidelidad vs simplicidad, global vs local, exacto vs aproximado.

Resultados de aprendizaje

Al finalizar, el estudiante podrá:

- Diferenciar local (esta predicción) vs global (modelo en general) y model-agnostic vs model-specific.
- Usar SHAP con TreeExplainer (rápido, exacto, árboles), KernelExplainer (lento, model-agnostic), DeepExplainer (redes neuronales) y Partition (recientes).
- Usar LIME para explicaciones model-agnostic en texto/imágenes/tabular.
- Generar PDP (Partial Dependence Plot) e ICE (Individual Conditional Expectation) con sklearn.inspection.
- Decidir cuándo SHAP es lo correcto y cuándo es overkill (ej. modelo lineal: usar coeficientes directamente).

Temas

#	Tema	Por qué importa
1	Local vs global, model-agnostic vs specifi	Vocabulario antes de elegir herramienta.
2	SHAP: TreeExplainer vs KernelExplainer	Exacto y rápido vs aproximado y lento.
3	LIME para texto/imágenes	Cuando SHAP no aplica.
4	PDP + ICE: efecto marginal vs heterogeneid	Lo que cambia con cada feature.
5	Endpoint /explain en producción	Latencia, caching, on-demand vs pre-comput
6	Comunicar a stakeholders no técnicos	Bar chart con top 5 features ≠ paper acade

Definiciones y características

- Interpretabilidad local: explica una predicción individual. "¿Por qué a este cliente lo categorizaste como riesgo?".
- Interpretabilidad global: explica el modelo en agregado. "¿Qué features pesan más en general?".
- Model-agnostic: funciona sin importar el modelo (LIME, KernelSHAP, PDP). Pro: portable. Con: más lento, aproximaciones.
- Model-specific: usa estructura del modelo (TreeSHAP usa árboles). Pro: rápido, exacto. Con: solo para

esa familia.
- SHAP values: contribución de cada feature al cambio respecto a la predicción base, basado en teoría de juegos (valores de Shapley). Propiedades garantizadas: efficiency (suman al output), symmetry, dummy, additivity.
- TreeExplainer: para tree-based (XGBoost, LightGBM, CatBoost, sklearn RF). Polynomial time, exacto.
- KernelExplainer: agnostic. Aproxima Shapley values con regresión lineal local con kernel. O(2^n) para exacto, K samples para aprox.
- PDP (Partial Dependence Plot): efecto promedio de una feature j sobre la predicción, marginalizando las demás. Asume independencia entre features (riesgo: si correladas, PDP miente).
- ICE (Individual Conditional Expectation): un PDP por instancia, sin promediar. Permite ver heterogeneidad (si el efecto de la feature varía por sub-grupo).
- LIME: aprende un modelo lineal/árbol simple alrededor de una instancia perturbando los inputs y observando outputs.

Dataset / recursos

- Dataset: California Housing.
- Modelos: XGBRegressor (TreeSHAP) y MLPRegressor (KernelSHAP/DeepSHAP).
- Librerías: shap>=0.45, lime, sklearn>=1.5, matplotlib.

Ejercicios

1. TreeSHAP: entrená XGB sobre California. explainer = shap.TreeExplainer(model). shap_values = explainer(X_test[:100]). shap.plots.waterfall(shap_values[0]) (local) y shap.plots.beeswarm(shap_values) (global).
2. KernelSHAP vs TreeSHAP: corré KernelExplainer con 100 background samples sobre el mismo XGB. Compará SHAP values con TreeSHAP. ¿Son iguales? ¿Cuánto tardó cada uno?
3. LIME tabular: LimeTabularExplainer sobre el mismo modelo. Explicá la misma instancia que en SHAP. Compará las features importantes.
4. PDP + ICE: sklearn.inspection.PartialDependenceDisplay.from_estimator(model, X, features=['MedInc', 'HouseAge'], kind='both'). Identificá si hay no-linealidad y/o heterogeneidad.
5. Endpoint /explain: extendé el FastAPI (Clase 199) con POST /explain que devuelve top-5 features + SHAP values + base value, para una instancia. Medí latencia — TreeSHAP debería ser <50 ms.

Homework verificable

Servicio FastAPI con:

1. POST /predict (de Clase 199).
2. POST /explain que devuelve {"prediction": float, "base_value": float, "top_features": [{"name": "MedInc", "shap": 0.23}, ...]}.
3. Endpoint GET /global-explanation que devuelve PDP + global SHAP precomputados (cacheados al startup).
4. UI HTML básica que muestra waterfall plot (SHAP) para inputs interactivos.
5. README que justifica: por qué TreeSHAP y no KernelSHAP, por qué top-5 y no top-20, por qué pre-computar global.

Criterio de aceptación: curl /explain -d '{"features":[...]}' devuelve top-5 features con SHAP, latencia <100 ms p99, y la UI muestra el waterfall correctamente.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
KernelExplainer tarda 30 s por instancia	Esperado — es O(N samples × N features). F
SHAP values no suman a la predicción	Probablemente comparás shap_values + base_
PDP miente: la curva sugiere feature j no	feature j está correlada con otra feature;
Explicaciones distintas en cada corrida	LIME usa sampling random. Fix: fijar seed;
Top features no coinciden entre SHAP y LIM	Es esperado — son métodos distintos con as
Bar chart con SHAP global muestra "MedInc	Comunicación. Fix: traducir nombres ("MedI

Preguntas frecuentes

¿SHAP, LIME, Permutation Importance — cuál uso?

- Modelo árbol (XGBoost, LightGBM, RF): TreeSHAP sin pensar.
- Modelo agnóstico, dataset grande: Permutation Importance (sklearn) para global; LIME para local si SHAP es muy lento.
- Red neuronal: DeepSHAP o Integrated Gradients (Captum para PyTorch).
- Modelo lineal: coeficientes × valor de feature.

¿Por qué SHAP es "el estándar"?

Por las propiedades teóricas (axiomas de Shapley: efficiency, symmetry, dummy, additivity) y porque tiene implementaciones rápidas (TreeSHAP) para modelos populares. LIME es más viejo y más heurístico.

¿On-demand /explain o pre-computado?

- Pocas inferencias/día: on-demand está bien.
- Muchas inferencias, mismo input recurrente: cachear.
- Análisis batch (auditoría mensual): pre-computar en bulk con TreeSHAP — sklearn pipelines + Spark/Dask.

¿Cómo manejo SHAP para LLMs?

shap.Explainer(model) con Partition masker funciona para text. Para LLMs grandes: caro y lento. Alternativas: attention attribution, gradient-based (Captum), o prompt-level ("¿qué part del prompt fue importante?"). Área activa de research.

¿Y si el regulador me pide explicabilidad pero el modelo es Black Box?

Tres opciones: (1) Cambiá a un modelo intrínsecamente interpretable (Logistic Regression, GAM, EBM de interpret); (2) Generá SHAP por cada decisión + archivá; (3) Para fairness/compliance, también probá aequitas o fairlearn (Clase 161).

¿PDP o ALE?

PDP es más popular y simple, pero mentira con features correladas. ALE corrige eso, ligeramente más complejo. Si tus features son correladas (lo usual): ALE. Si independientes (raro): PDP alcanza.

Referencias

- Molnar, C. Interpretable Machine Learning (2ª ed., libro online gratis):

https://christophm.github.io/interpretable-ml-book/
- Lundberg, S. & Lee, S. A Unified Approach to Interpreting Model Predictions (NIPS 2017) — SHAP.
- Ribeiro, M. et al. "Why Should I Trust You?": Explaining the Predictions of Any Classifier (KDD 2016) — LIME.
- SHAP docs — TreeExplainer, KernelExplainer, plots.
- InterpretML / EBM — modelos intrínsecamente interpretables (Microsoft Research).

Siguiente clase

Clase 206 — Testing de datos: Great Expectations, Deequ

Apéndice: notebook (primer bloque)

TreeSHAP vs KernelSHAP, PDP+ICE, comparación con LIME, y diseño de un endpoint /explain. Requiere: pip install shap lime xgboost matplotlib.

```
import numpy as np, pandas as pd, time
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
import xgboost as xgb

data = fetch_california_housing(as_frame=True)
X, y = data.data, data.target
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=0.2, random_state=42)

model = xgb.XGBRegressor(n_estimators=200, max_depth=5, random_state=42, n_jobs=-1).fit(Xtr, ytr)
print('test R2:', model.score(Xte, yte))
```

Archivos complementarios

- notebook.ipynb