Clase 052 — Testing, validación, hyperparameter tuning, no free lunch theorem

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 2 + sklearn user guide. Duración estimada: 70 min.

Clase 052 — Testing, validación, hyperparameter tuning, no free lunch theorem

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 2 + sklearn user guide. Duración estimada: 70 min.*

Objetivo

Que el alumno entienda cómo medir generalización sin engañarse — separar train/val/test correctamente, usar cross-validation para estimar performance con poca varianza, tunear hiperparámetros con GridSearchCV / RandomizedSearchCV, y aceptar el no free lunch theorem: ningún modelo gana en todos los datasets.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Separar un dataset en train/validation/test y justificar por qué el test no se toca hasta el final.
2. Aplicar KFold y StratifiedKFold con cross_val_score para estimar generalización.
3. Tunear hiperparámetros con GridSearchCV y RandomizedSearchCV, leyendo cv_results_.
4. Reconocer cuándo KFold rompe (datos temporales, grupos) y elegir el splitter correcto.
5. Explicar el no free lunch theorem y por qué siempre conviene comparar varios modelos.

Temas

#	Tema	Por qué importa
1	Train / validation / test split	Test es sagrado: una sola medición al fina
2	KFold y StratifiedKFold	Estima generalización con menos varianza q
3	cross_val_score y cross_validate	Una línea, K métricas, intervalo de confia
4	GridSearchCV vs RandomizedSearchCV	Grid es exhaustivo, random escala mejor en
5	Pipeline + CV (evitar leakage del scaler)	Fitear el scaler dentro del fold, nunca af
6	No free lunch theorem	Ningún algoritmo domina en todo dataset —

Versión profundizada — 2026

El tema moderno que vivía como complemento dentro de esta clase ahora tiene clase propia dedicada con patrón completo, ejercicios y homework:

- Clase 049b — Validación temporal: TimeSeriesSplit, walk-forward, blocking

Definiciones y características

Train / validation / test split

: Train entrena, validation tunea hiperparámetros, test se mira una sola vez al final. Proporciones típicas:

60/20/20 o 70/15/15. En CV el val se reemplaza por los K folds, pero el test queda igualmente intocable.

Cross-validation (CV)

: Estimar generalización promediando K mediciones rotando qué fold es val. Reduce varianza vs un hold-out único. Costo: K veces más entrenamientos.

Hold-out

: Una sola partición train/val. Rápido pero ruidoso — con datasets chicos la métrica depende mucho de qué muestras cayeron en val.

Hyperparameter

: Configuración del modelo que no se aprende del data (ej. max_depth, C, learning_rate). Se tunea en validation, nunca en test.

Generalization gap

: Diferencia entre score en train y score en val/test. Gap grande = overfitting. Gap chico pero score bajo = underfitting.

No free lunch theorem (Wolpert, 1996)

: Promediado sobre todos los problemas posibles, ningún algoritmo de ML es mejor que otro. En la práctica: no hay un "mejor modelo" universal; siempre hay que comparar (linear, RF, GBM, etc.) sobre tu dataset específico.

Leak temporal

: Cuando el train contiene información que en producción no estaría disponible al momento de predecir (típicamente, datos del futuro respecto al target). Mata silenciosamente proyectos de forecasting.

Dataset / recursos

- sklearn.datasets.load_breast_cancer para CV estratificado.
- sklearn.datasets.fetch_california_housing para tuning con GridSearchCV.
- Serie sintética con np.cumsum(np.random.randn(500)) para TimeSeriesSplit.

Ejercicios

1. Hold-out vs CV. Sobre breast_cancer, comparar el score de un único train_test_split (varios random_state) vs cross_val_score(cv=5). Mostrar que el hold-out varía ±0.03 entre seeds, CV mucho menos.

2. StratifiedKFold. Con un target desbalanceado 90/10, comparar KFold vs StratifiedKFold mirando la proporción de la clase minoritaria en cada fold.

3. GridSearchCV en pipeline. Pipeline([('scaler', StandardScaler()), ('svc', SVC())]) + GridSearchCV sobre C y gamma. Verificar que el scaler se fitea dentro de cada fold (no leakage).

4. RandomizedSearchCV. Mismo problema que (3) pero con RandomizedSearchCV(n_iter=30) y distribuciones (scipy.stats.loguniform). Comparar tiempo y best_score_.

5. TimeSeriesSplit. Generar serie con tendencia + ruido. Aplicar TimeSeriesSplit(n_splits=5) y entrenar Ridge con features lag-1, lag-7. Reportar score por fold. Repetir con KFold(shuffle=True) y mostrar que el score se infla artificialmente (leakage temporal).

Homework verificable

Notebook que sobre california_housing: (a) split 80/20 train/test con random_state=42; (b) GridSearchCV(cv=5) con RandomForestRegressor sobre n_estimators {100, 300} y max_depth {None, 10, 20}; (c) reportar best_params_ y best_score_; (d) evaluar el mejor modelo una sola vez sobre test; (e) comparar contra un baseline DummyRegressor(strategy='mean').

Criterio de aceptación: El test se evalúa una única vez al final del notebook. El RF supera al baseline en R^2 al menos por 0.4. cv_results_ queda exportado a CSV.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Score de CV altísimo pero en producción se	Leakage. Fix: scaler/encoder dentro del Pi
Usar KFold con datos temporales y obtener	Estás "viendo el futuro". Fix: TimeSeriesS
GridSearchCV tarda horas	Espacio de búsqueda demasiado grande. Fix:
Mirar el test set varias veces para "ajust	Estás haciendo overfitting del test → la m
Folds con clase minoritaria ausente (Value	KFold random en target desbalanceado. Fix:

Preguntas frecuentes

¿Cuándo TimeSeriesSplit en lugar de KFold?

Siempre que haya orden temporal con sentido predictivo: forecasting, logs, sensores, transacciones, métricas de negocio. Si las observaciones son intercambiables (tabular cross-sectional, ej. clasificar imágenes independientes), KFold/StratifiedKFold está bien. Regla mecánica: si tu target depende del tiempo y predecís hacia adelante, TimeSeriesSplit.

¿GridSearchCV o RandomizedSearchCV?

Grid si tenés ≤ ~50 combinaciones y querés exhaustividad. Random a partir de espacios grandes (>100 combos) o cuando hay hiperparámetros continuos (C, alpha, learning_rate): con 30-60 iteraciones random llegás cerca del óptimo con una fracción del costo (resultado teórico de Bergstra & Bengio 2012).

¿K=5 o K=10 en KFold?

K=5 es el default razonable (buen compromiso varianza/costo). K=10 cuando el dataset es chico y necesitás más muestras por fold. K=N (LOOCV) casi nunca: costo absurdo y varianza alta.

¿Por qué importa el no free lunch theorem en la práctica?

Porque te recuerda no quedarte con un solo modelo "favorito". Para cualquier dataset nuevo, comparar al menos: un baseline tonto (Dummy), un lineal (Ridge/LogReg), un árbol ensamble (RandomForest/GradientBoosting). El que gane depende del problema, no de tu intuición previa.

¿Puedo usar cross_val_score y después GridSearch sobre el mismo set?

Sí, pero entendé que estás haciendo CV anidada implícita. Lo correcto cuando querés estimar la performance del proceso de tuning: cross_val_score(GridSearchCV(...), X_train, y_train, cv=outer_cv). CV externa evalúa, CV interna tunea. Caro pero honesto.

Referencias

- Géron, Hands-On ML, cap. 2 — End-to-End ML Project (sección "Better Evaluation Using Cross-Validation").
- sklearn — Cross-validation user guide
- sklearn — TimeSeriesSplit
- sklearn — GridSearchCV / RandomizedSearchCV
- Bergstra & Bengio (2012), Random Search for Hyper-Parameter Optimization, JMLR.
- Wolpert (1996), The Lack of A Priori Distinctions Between Learning Algorithms.

Siguiente clase

Clase 053 — Validación temporal: TimeSeriesSplit, walk-forward, blocking

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb