Clase 057 — Fine-tuning: grid search y randomized search

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

Clase 057 — Fine-tuning: grid search y randomized search

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

Objetivo

Que el alumno deje de tunear hiperparámetros "a ojo" y use búsquedas sistemáticas con validación cruzada — GridSearchCV cuando el espacio es chico y discreto, RandomizedSearchCV cuando es grande o continuo — integradas dentro de un Pipeline para evitar data leakage del preprocesamiento.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Distinguir parámetros entrenados (pesos del modelo) de hiperparámetros (los que vos fijás antes del fit).
2. Configurar GridSearchCV con param_grid, cv, scoring, n_jobs=-1 y leer best_params_ / best_estimator_ / cv_results_.
3. Usar RandomizedSearchCV con param_distributions (scipy.stats: randint, uniform, loguniform) y n_iter para presupuestar trials.
4. Integrar HPO dentro de un Pipeline usando claves del tipo 'step__hparam' para tunear preprocesamiento + modelo juntos.
5. Decidir grid vs random vs bayesiano según tamaño del espacio, costo por fit y continuidad de los hiperparámetros.

Temas

#	Tema	Por qué importa
1	Hiperparámetros vs parámetros	El error conceptual #1 al arrancar con skl
2	param_grid y scoring	Definís el espacio y la métrica que querés
3	cv y n_jobs=-1	CV honesto + paralelismo gratis sobre todo
4	GridSearchCV	Producto cartesiano completo. Garantiza en
5	RandomizedSearchCV con distribuciones (ra	Muestrea n_iter puntos; gana cuando pocos
6	Pipelines + HPO con sintaxis step__hparam	Evita leakage del scaler/imputer al hacer
7	Inspeccionar cv_results_	DataFrame con todos los trials; sirve para

Versión profundizada — 2026

El tema moderno que antes vivía como complemento dentro de esta clase ahora tiene su(s) clase(s) propia(s) con patrón completo, ejercicios y homework:

- Clase 052a — Optuna y HPO bayesiano dedicado

Definiciones y características

Hiperparámetro

: Parámetro que vos fijás antes del entrenamiento y que el fit no aprende. Ej.: max_depth, C, n_estimators. Distinto de los parámetros aprendidos (pesos, splits del árbol).

GridSearchCV(estimator, param_grid, cv, scoring, n_jobs)

: Evalúa el producto cartesiano de param_grid con CV. Total de fits = ∏ |valores| × cv. Determinista y exhaustivo dentro del grid.

RandomizedSearchCV(estimator, param_distributions, n_iter, cv, scoring)

: Muestrea n_iter combinaciones de las distribuciones. Acepta tanto listas (como grid) como distribuciones scipy (randint, uniform, loguniform). Gana cuando algunos hparams son irrelevantes.

param_distributions

: Dict {'hparam': distribución_o_lista}. Para escalas tipo learning rate o C de SVM, usá loguniform(1e-4, 1e0) — random uniforme en log-space, no lineal.

refit=True (default)

: Tras encontrar la mejor combinación, re-entrena el estimador sobre todo el train set con esos hparams. Por eso best_estimator_ viene listo para predecir.

best_params_

: Dict con la combinación ganadora.

best_estimator_

: El modelo ya re-entrenado en train completo. Es el que usás para .predict() en test.

scoring

: Métrica a maximizar. Strings como 'accuracy', 'f1', 'neg_root_mean_squared_error' (negados porque sklearn maximiza siempre). También callables custom con make_scorer.

cv_results_

: Dict-of-arrays con todos los trials: parámetros, mean/std de score por fold, tiempos. Lo cargás en un DataFrame para decidir si refinar.

Dataset / recursos

fetch_california_housing() de sklearn (mismo que viene usando Géron en cap. 2). Para los ejercicios con clasificación, load_breast_cancer().

Ejercicios

1. GridSearchCV sobre RandomForestRegressor. California Housing. param_grid con n_estimators {50, 100, 200} y max_features {4, 6, 8}. cv=5, scoring='neg_root_mean_squared_error', n_jobs=-1. Reportá best_params_ y RMSE en test.

2. RandomizedSearchCV con distribuciones. Mismo dataset, ahora con n_estimators=randint(50, 500), max_features=randint(2, 8), min_samples_leaf=randint(1, 20). n_iter=30. Compará tiempo y score vs el grid del ej. 1.

3. Pipeline + HPO. Construí Pipeline([('scaler', StandardScaler()), ('svr', SVR())]). Tuneá svr__C con loguniform(1e-1, 1e3) y svr__gamma con loguniform(1e-4, 1e-1). Mostrá por qué meter el StandardScaler

afuera del CV sería data leakage.

4. Inspección de cv_results_. Cargá search.cv_results_ en un DataFrame, ordenalo por mean_test_score y plotteá score vs n_estimators. ¿Hay meseta? ¿Vale subir más?

5. Optuna sobre el mismo problema. Repetí el ej. 2 pero con Optuna (n_trials=30, TPESampler, MedianPruner). Compará score final y tiempo con RandomizedSearchCV. Mostrá study.best_params y un plot de optuna.visualization.plot_optimization_history.

Homework verificable

Notebook que: (a) baseline con RandomForestRegressor sin tunear; (b) GridSearchCV con grid chico (≤27 combos); (c) RandomizedSearchCV con n_iter=30 y distribuciones log; (d) Optuna con n_trials=30 y MedianPruner; (e) tabla comparativa con RMSE en test, tiempo de búsqueda y mejor combinación de cada método.

Criterio de aceptación: los tres métodos (grid/random/Optuna) mejoran sobre el baseline. La tabla final tiene RMSE_test, segundos y best_params. El Pipeline se usa en todos para evitar leakage del scaler.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
ValueError: Invalid parameter 'C' for esti	Estás pasando 'C' pero dentro de un Pipeli
GridSearchCV tarda eternidades	Producto cartesiano explotó. Fix: pasá a R
RMSE de CV mucho mejor que en test	Data leakage: aplicaste StandardScaler.fit
ImportError: cannot import name 'HalvingGr	Sigue siendo experimental en sklearn. Fix:
scoring='rmse' tira ValueError: 'rmse' is	sklearn no tiene 'rmse' directo; maximiza

Preguntas frecuentes

¿GridSearch, RandomSearch u Optuna?

Regla de bolsillo: espacio chico y discreto (≤50 combos) → Grid, te asegura el óptimo del grid. Espacio mediano o con hparams continuos → Random con loguniform para los que viven en escala log. Cada fit cuesta caro (XGBoost grande, red neuronal) o el espacio es enorme → Optuna con TPE + MedianPruner. Para producción seria hoy, Optuna es default.

¿Cuántos n_iter en RandomizedSearchCV?

Bergstra & Bengio (2012) mostraron que 60 trials random alcanzan el top-5% del espacio con probabilidad >95% si pocos hparams dominan. Empezá con 30-60 y subí si cv_results_ muestra que el top-N todavía mejora.

¿n_jobs=-1 en el search o en el modelo?

Si lo ponés en los dos, se pelean por los cores. Recomendación: n_jobs=-1 en el search (paraleliza folds × candidatos) y n_jobs=1 dentro del estimador. Para modelos que ya paralelizan internamente y son rápidos (RandomForest chico), a veces es al revés — medilo.

¿cv=5 o cv=10?

cv=5 es el sweet spot por costo/varianza. cv=10 baja un poco la varianza pero duplica tiempo. Para datasets chicos (≤1k filas), cv=10 o incluso LOO. Para datasets grandes, cv=3 ya alcanza.

¿Tuneo sobre train y evalúo sobre test, o necesito un set de validación aparte?

Con CV adentro de GridSearchCV ya tenés validación honesta sobre folds del train. Test set se toca una sola vez al final, con el best_estimator_ ya elegido. Si vas a hacer muchas iteraciones de exploración manual encima, dejá un set de validación separado para no contaminar test.

Referencias

- Géron, cap. 2 § Fine-Tune Your Model.
- sklearn — GridSearchCV
- sklearn — RandomizedSearchCV
- sklearn — Successive Halving (experimental)
- Optuna docs — TPE, pruners, dashboard.
- Bergstra & Bengio (2012), Random Search for Hyper-Parameter Optimization, JMLR.

Siguiente clase

Clase 058 — Optuna y HPO bayesiano dedicado

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb