Clase 081 — Regularización de árboles

Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 6. Duración estimada: 50 min.

Clase 081 — Regularización de árboles

> *Parte: 1 — Machine Learning Clásico · Fuente: Géron, cap. 6. Duración estimada: 50 min.*

Objetivo

Que el alumno controle el overfitting de un DecisionTreeClassifier/Regressor usando hiperparámetros de regularización (pre-pruning) y cost-complexity pruning (post-pruning), y sepa elegir entre ambas estrategias en función del problema.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Identificar overfitting en un árbol sin regularizar (train accuracy ≈ 1, test bajo).
2. Tunear max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes, max_features con GridSearchCV.
3. Aplicar cost-complexity pruning vía ccp_alpha y leer la curva cost_complexity_pruning_path.
4. Diferenciar pre-pruning (frenar el crecimiento) de post-pruning (podar después).
5. Justificar la elección de hiperparámetros con curvas de validación, no a ojo.

Temas

#	Tema	Por qué importa
1	Árboles sin regularizar = overfitting gara	Un árbol crece hasta hojas puras.
2	max_depth y max_leaf_nodes	Los dos frenos más efectivos y baratos.
3	min_samples_split / min_samples_leaf	Frenan splits sobre muestras chicas (ruido
4	max_features	Aleatoriza el split; antesala de Random Fo
5	ccp_alpha (cost-complexity pruning)	Post-pruning principled, basado en α.
6	Pre-pruning vs post-pruning	Trade-off: rapidez vs óptimo bias-variance

Definiciones y características

max_depth

: Profundidad máxima del árbol. Default None (crece hasta hojas puras → overfit). Valor típico: 3–10. El hiperparámetro más impactante y barato de tunear.

min_samples_split

: Mínimo de muestras que un nodo necesita para ser candidato a split. Default 2 (cualquier nodo se parte). Subirlo (ej. 20) bloquea splits sobre muestras chicas, que suelen ser ruido.

min_samples_leaf

: Mínimo de muestras que debe quedar en cada hoja resultante del split. Default 1 (hojas con 1 sample → memorización). Subirlo a 5–20 suaviza la frontera.

max_leaf_nodes

: Tope absoluto de hojas. El árbol crece en best-first (mejor reducción de impureza primero) hasta alcanzarlo. Equivalente funcional a max_depth pero más fino — el árbol no tiene que ser balanceado.

max_features

: Cantidad de features consideradas en cada split. Default = todas. Bajarlo (sqrt, log2) introduce aleatoriedad, reduce varianza y acelera el fit. Es la idea base de Random Forest.

ccp_alpha (cost-complexity pruning)

: Hiperparámetro $\alpha \geq 0$ que penaliza la complejidad del árbol al podar. Se minimiza $R(T) + \alpha \cdot |T|$ donde $|T|$ = nº de hojas. $\alpha=0$ no poda; α grande poda agresivo. Se elige con cost_complexity_pruning_path() + CV.

Pre-pruning

: Frenar el crecimiento durante el fit con max_depth, min_samples_*, max_leaf_nodes. Rápido y simple — es lo que más se usa.

Post-pruning

: Dejar crecer el árbol completo y después podarlo con ccp_alpha. Más fundamentado teóricamente, pero requiere doble pasada (fit completo + path + CV).

Dataset / recursos

moons de sklearn.datasets.make_moons(n_samples=1000, noise=0.4) — clásico para visualizar fronteras de decisión y el efecto de regularizar.

Ejercicios

1. Overfit baseline. Entrená un DecisionTreeClassifier() sin tocar nada sobre moons. Reportá train vs test accuracy. ¿Cuánto gap hay?

2. max_depth sweep. Para max_depth ∈ {1, 2, 3, 5, 10, None} graficá train y test accuracy. Identificá el punto donde empieza el overfit.

3. GridSearch. Buscá con GridSearchCV(cv=5) la mejor combinación de max_depth, min_samples_leaf y max_leaf_nodes. Reportá mejores params y test score.

4. Cost-complexity path. Llamá tree.cost_complexity_pruning_path(X_train, y_train) para obtener ccp_alphas. Entrená un árbol por cada α y graficá test accuracy vs α. Elegí el óptimo.

5. Visualización de fronteras. Plotteá la decision boundary de (a) árbol sin regularizar y (b) árbol con max_depth=4. Compará overfit visual.

Homework verificable

Notebook con moons (noise=0.4, 1000 samples, split 70/30): (a) baseline sin regularizar — reportar gap train/test; (b) GridSearchCV sobre max_depth y min_samples_leaf con 5-fold; (c) cost-complexity pruning — graficar ccp_alphas vs test accuracy y elegir el mejor α; (d) comparar test accuracy de los 3 modelos (baseline, grid, pruned) en tabla.

Criterio de aceptación: El modelo regularizado tiene gap train-test ≤ 5 pp y test accuracy ≥ baseline. La elección de α está justificada con la curva, no a ojo.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
Train accuracy = 1.0, test = 0.75	Árbol sin regularizar memorizó train. Fix:
GridSearchCV tarda eternamente	Grilla enorme sobre 4+ hiperparámetros. Fi
ccp_alpha no hace nada	Default es 0.0 (sin pruning). Fix: corré c
min_samples_split=2 "no es default" — sí l	Confusión típica: 2 es el default y signif
Tuneás max_features=sqrt y los resultados	max_features < n_features introduce aleato

Preguntas frecuentes

¿Pre-pruning o post-pruning?

Pre-pruning (max_depth, min_samples_leaf) en el 90% de los casos: rápido, simple, suficiente. Post-pruning (ccp_alpha) cuando querés un árbol único, interpretable, óptimo bias-variance — típico en problemas tabulares chicos donde el árbol es el modelo final, no un building block. Para ensembles (RF, GBM), pre-pruning y listo.

¿max_depth o max_leaf_nodes?

max_depth es más fácil de interpretar ("profundidad ≤ 5"). max_leaf_nodes es más flexible — deja que el árbol sea asimétrico y crezca solo donde realmente reduce impureza. Si te importa interpretabilidad: max_depth. Si te importa performance: max_leaf_nodes.

¿Cómo elijo min_samples_leaf?

Regla de dedo: 1–5% de n_samples. Con 1000 filas, probá 10–50. Cuanto más ruidoso el dataset, más alto.

¿max_features regulariza?

Sí, indirectamente: al limitar features candidatas por split, fuerza al árbol a usar features sub-óptimas y reduce varianza. Es el mecanismo central de Random Forest, no tanto de árbol único.

¿Por qué ccp_alpha se usa poco en la práctica?

Porque (a) requiere doble pasada (path + CV), (b) en ensembles ya tenés regularización implícita por agregación, (c) max_depth + min_samples_leaf suele alcanzar. Pero es la podada con mejor fundamento teórico (Breiman et al., CART).

Referencias

- Géron, cap. 6 § "Regularization Hyperparameters".
- scikit-learn — Decision Trees user guide
- scikit-learn — Post pruning decision trees with cost complexity pruning
- Breiman, Friedman, Olshen & Stone (1984), Classification and Regression Trees — el paper original de CART y cost-complexity pruning.

Siguiente clase

Clase 082 — Regresión con árboles

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
# Imports y configuración inicial
```

Archivos complementarios

- notebook.ipynb