

Introduction au Machine Learning



Séance 7 — Valider et régler sans tricher : la validation croisée

Dr. El Hadji Bassirou TOURÉ

Département de Mathématiques et Informatique, Faculté des Sciences et Techniques, Université Cheikh Anta Diop de Dakar

<https://ebtoure.sn/cours/introduction-machine-learning/chapitres/07-validation-croisee.html>

Version du 11 octobre 2026



Version en ligne

Introduction au ML — Séance 7

Valider et régler sans tricher : la validation croisée

Dr. El Hadji Bassirou TOURÉ

Département de Mathématiques et Informatique, Faculté des Sciences et Techniques, Université Cheikh Anta Diop de Dakar

Objectifs de la séance

Contenu

La Séance 6 a laissé deux curseurs en suspens — le seuil et `c` — avec une interdiction : ne pas les régler sur le test. Cette séance construit le protocole qui permet de régler *tous* les réglages du cours sans se mentir.

- Les **trois usages** des données : entraîner, régler, juger — et pourquoi ils ne se mélangent pas.
- La **variance du découpage** : pourquoi un score sur un seul split ne prouve presque rien.
- La **validation croisée** (k plis), construite à la main puis vérifiée contre `cross_val_score`.
- Régler un hyperparamètre : la **courbe de validation** (α de Ridge, profondeur d'arbre).
- Plusieurs réglages à la fois : **GridSearchCV**, sa double boucle et son coût.
- Le **seuil** de la Séance 6 enfin réglé proprement, par `cross_val_predict`.

Trois usages, un seul jeu de données

Là où la Séance 6 s'est arrêtée

Le constat du mini-défi

Régler le seuil *en regardant le test* donnait un chiffre flatteur mais sans valeur — la fuite de la Séance 4, version hyperparamètre. La réparation proposée — réserver une tranche de **validation** — a soulevé deux objections immédiates : elle **ampute l'entraînement**, et son verdict **dépend du hasard du découpage**. La séance résout les deux.

L'inventaire des réglages accumulés depuis le début du cours : la profondeur de l'arbre (S3), le k du k -NN (S4), l' α de Ridge (S5), le c et le seuil de la logistique (S6). Tous attendent le même protocole.

Paramètres et hyperparamètres : deux familles de réglages

Définition

Les **paramètres** (b , w , les seuils de coupure d'un arbre...) sont appris *par* l'entraînement.

Les **hyperparamètres** (profondeur, k , α , C , seuil de décision...) sont choisis *avant* ou *autour* de l'entraînement — l'algorithme ne peut pas les apprendre lui-même.

Le test du « qui choisit ? » : si **fit** le calcule, c'est un paramètre ; si c'est un argument du constructeur (ou un curseur appliqué après), c'est un hyperparamètre. Or un hyperparamètre mal choisi peut ruiner le meilleur des algorithmes — la profondeur 12 de l'arbre en S5 divisait son R^2 par deux. Il faut donc des données pour les *départager*.

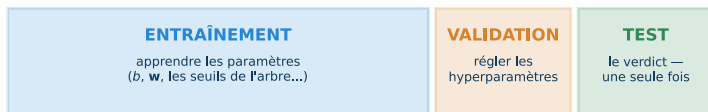
L'idée : trois usages, trois jeux

Analogie

La préparation au baccalauréat : les **annales** pour apprendre (entraînement), le **bac blanc** pour choisir sa stratégie (réglage), le **vrai bac** pour juger — une seule fois. Réviser sur les sujets du vrai bac, c'est falsifier la mesure.

L'idée en une phrase

Toute donnée qui a servi à **choisir** quelque chose — un poids, un hyperparamètre, un seuil — ne peut plus servir à **juger** le résultat de ce choix.



trois usages, trois jeux de données : ce qui sert à régler ne peut plus servir à juger

Le test ne sert qu'une fois

Le bac blanc repassé cinquante fois

Évaluer cinquante variantes sur le test et garder la meilleure, c'est faire du test un jeu de réglage : le chiffre final est *optimiste par construction* — la variante gagnante a en partie gagné par chance, et cette chance ne se reproduira pas en production. Le test se consulte **une seule fois**, à la toute fin, quand tous les choix sont gelés.

D'où la question pratique de toute la séance : avec un budget de données fini, comment fabriquer un *juge de réglage* fiable sans sacrifier ni l'entraînement, ni le test ?

La variance du découpage

Exemple — la même recette, cinq découpages

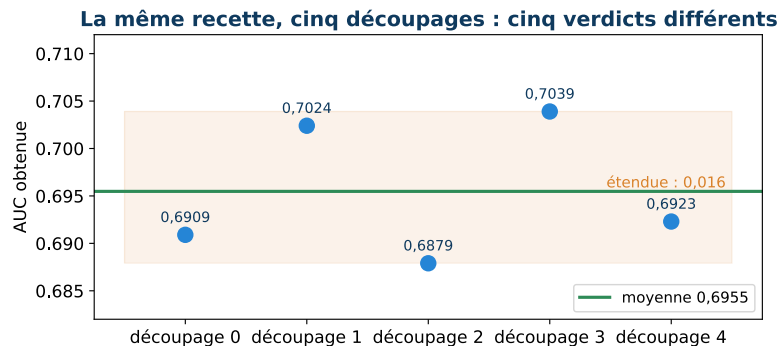
La logistique du paludisme, cinq splits différents

La même recette exactement — Pipeline standardisation + logistique — entraînée et évaluée sur cinq découpages train/test (mêmes proportions, graines différentes) :

découpage	0	1	2	3	4
AUC	0,691	0,702	0,688	0,704	0,692

Moyenne 0,6955, écart-type 0,0064, **étendue** 0,016. Rien n'a changé dans la recette — seul le hasard du découpage a parlé.

Ce que cette variance interdit de conclure



Un score d'évaluation n'est pas *le* niveau du modèle : c'est une **mesure bruitée** de ce niveau, dont le bruit vient du découpage. Comparer deux recettes qui diffèrent de 0,01 sur *un seul* split, c'est comparer deux mesures dont le bruit propre vaut déjà 0,016 : le classement peut s'inverser au prochain tirage.

Exercice — peut-on conclure ?

Énoncé

Sur un unique découpage, la recette A obtient une AUC de 0,72 et la recette B une AUC de 0,73. Au vu de l'étendue mesurée ci-dessus (0,016 entre découpages), peut-on déclarer B meilleure que A ? Que faudrait-il pour trancher ?

Correction

Non : l'écart observé (0,01) est *inférieur* au bruit typique du découpage (0,016) — l'ordre pourrait s'inverser sur un autre split. Pour trancher, il faut **plusieurs** évaluations de chaque recette (plusieurs découpages) et comparer les *moyennes* en regard des écarts-types. C'est exactement ce que la validation croisée va industrialiser.

La validation croisée

L'idée : chacun son tour d'être le juge

Analogie

Une tontine : chaque membre cotise à tous les tours, et reçoit la cagnotte *une fois*, à son tour. La validation croisée organise les données pareil : chaque pli sert à entraîner à tous les tours, sauf au sien — où il devient le juge. À la fin, *toutes* les données ont entraîné, *toutes* ont jugé, et aucune n'a fait les deux en même temps.

L'idée en une phrase

La **validation croisée** à k plis découpe l'entraînement en k morceaux, fait k entraînements (à chaque tour, un pli différent joue la validation), et rend la **moyenne** des k scores — avec son écart-type, qui mesure le bruit.

La validation croisée, formellement

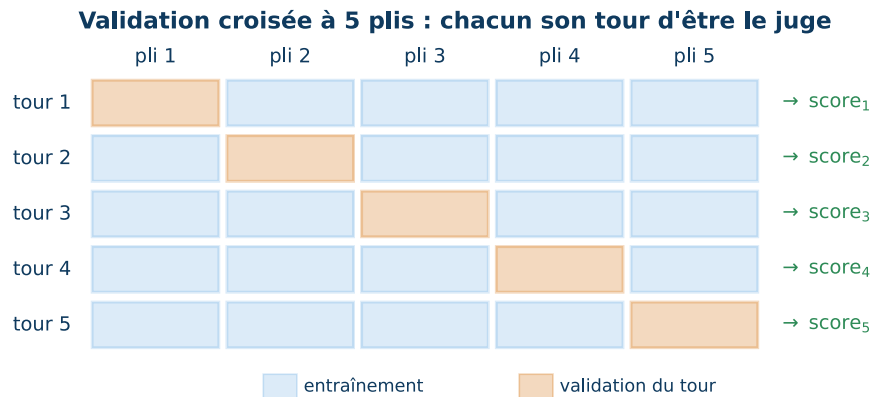
Définition

Soit l'entraînement partitionné en k plis $V_1, \dots, V_k$ de tailles égales. Pour chaque tour j , la recette est entraînée sur tout sauf V_j , puis notée sur V_j : score_j . Le verdict est la moyenne, l'incertitude est l'écart-type.

$$\text{CV}_k = \frac{1}{k} \sum_{j=1}^k \text{score}_j \qquad s = \sqrt{\frac{1}{k} \sum_j (\text{score}_j - \text{CV}_k)^2}$$

Chaque observation est jugée *exactement une fois*, par un modèle qui ne l'a jamais vue — c'est la propriété centrale. Le prix : k entraînements au lieu d'un. Et le test, lui, reste intact dans son coffre.

Le schéma des cinq plis



À chaque tour (ligne), le pli orange juge, les plis bleus entraînent. Cinq tours, cinq scores, une moyenne — et chaque donnée n'a été juge qu'une fois, jamais de sa propre recette.

Le mécanisme — la CV à la main (1/2)

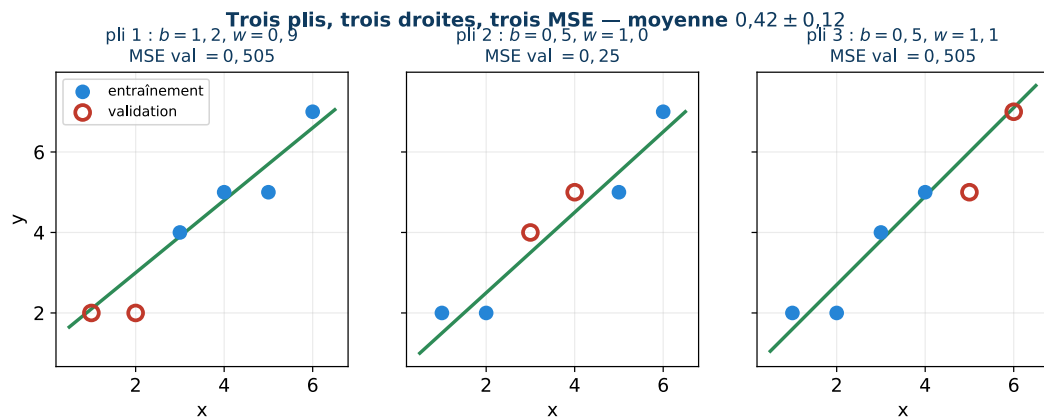
Six points, trois plis, trois équations normales

Le jeu jouet : $x = (1, 2, 3, 4, 5, 6)$, $y = (2, 2, 4, 5, 5, 7)$, découpé en trois plis consécutifs $\{1, 2\}$, $\{3, 4\}$, $\{5, 6\}$. À chaque tour, la droite est ajustée par les **formules fermées de la Séance 5** sur les quatre points restants :

tour	valide sur	entraîne sur	droite apprise	MSE de validation
1	$\{1, 2\}$	$\{3, 4, 5, 6\}$	$b = 1,2, w = 0,9$	0,505
2	$\{3, 4\}$	$\{1, 2, 5, 6\}$	$b = 0,5, w = 1,0$	0,250
3	$\{5, 6\}$	$\{1, 2, 3, 4\}$	$b = 0,5, w = 1,1$	0,505

Le tour 3 entraîne sur $\{1, 2, 3, 4\}$ — et retrouve **exactement** le $b = 0,5, w = 1,1$ du jouet de la Séance 5 : même données, même équation normale, même droite.

Le mécanisme — la CV à la main (2/2)



Trois droites différentes — chaque tour apprend sur des données différentes — et trois MSE de validation. Le verdict : $CV_3 = (0,505 + 0,250 + 0,505)/3 = 0,42$, écart-type $0,12$. `cross_val_score` rend exactement (0,505; 0,250; 0,505) : le mécanisme est celui-là, rien de plus.

Ce que la CV ne fabrique pas

Trois droites, mais lesquelles garder ?

Aucune : la validation croisée n'élit pas un des k modèles — elle évalue une **recette** (un algorithme et ses hyperparamètres). Une fois la recette validée, le modèle final est **réentraîné sur tout l'entraînement** : il profite alors de toutes les données, et la CV a déjà dit ce qu'il vaudra, en moyenne.

Distinguer trois objets : la *recette* (ce que la CV note), les *k modèles intermédiaires* (jetables, ils n'existent que pour noter), et le *modèle final* (réentraîné sur tout, c'est lui qui part en production).

La stratification : la classe rare dans chaque pli

L'idée en une phrase

Sur une cible rare (11,8 % de paludisme), un pli tiré au hasard peut être anormalement pauvre ou riche en positifs — son verdict est faussé. **StratifiedKFold** impose à chaque pli la proportion globale, comme le split stratifié de la Séance 6.

Parts de positifs par pli sur l'entraînement du paludisme

KFold	0,106	0,128	0,124	0,108	0,123
StratifiedKFold	0,1175	0,1175	0,1175	0,1175	0,1181

Sans stratification, le pli le plus pauvre (10,6 %) et le plus riche (12,8 %) jugent des situations sensiblement différentes. Règle : **classification** $\Rightarrow$ **plis stratifiés**, toujours.

Combien de plis ?

Le choix de k est un compromis coût/fiabilité :

- $k = 5$: le standard — chaque tour entraîne sur 80 % des données, cinq entraînements seulement ;
- $k = 10$: verdict un peu plus stable, deux fois plus cher — courant quand les données sont rares ;
- $k = n$ (*leave-one-out*) : chaque point jugé par un modèle entraîné sur tous les autres — n entraînements, réservé aux très petits jeux ;
- $k = 2$ ou 3 : rapide, mais chaque tour n'entraîne que sur la moitié ou les deux tiers — le verdict sous-estime la recette.

Dans tout le cours : $k = 5$, **stratifié en classification**, sauf mention contraire.

En pratique : cross_val_score

```
from sklearn.model_selection import cross_val_score, StratifiedKFold

pipe = make_pipeline(StandardScaler(),
                     LogisticRegression(max_iter=1000, random_state=0))
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=0)
scores = cross_val_score(pipe, X_train, y_train, cv=cv, scoring="roc_auc")
print(scores.mean(), "+/-", scores.std())
```

Trois détails qui comptent : la recette passée est le **Pipeline entier** (la standardisation est refaite *dans* chaque pli — la raison arrive dans les pièges) ; `scoring` nomme la métrique de la Séance 6 ("`roc_auc`", "`recall`", ...) ; et seul `X_train` entre — le test n'approche jamais la CV.

Exercice — dépouiller une validation croisée

Énoncé

Une CV à 5 plis rend les scores (0,70; 0,74; 0,68; 0,73; 0,70) pour la recette A, et un *unique* split rend 0,75 pour la recette B. Calculer la moyenne de A (les cinq valeurs somment à 3,55), puis dire laquelle des deux affirmations est la plus solide : « A vaut environ 0,71 » ou « B vaut 0,75 ».

Correction

Moyenne de A : $3,55/5 = 0,71$, avec une dispersion visible d'environ $\pm 0,02$. L'affirmation solide est celle sur A : cinq mesures indépendantes encadrent le niveau. Le 0,75 de B est *une seule* mesure bruitée — au vu d'un bruit typique de $\pm 0,02$, B pourrait valoir 0,71 comme 0,77. Conclure « B bat A » serait prématuré : il faut la CV de B.

Régler un hyperparamètre

L'idée : une note de CV par valeur du curseur

L'idée en une phrase

Pour régler un hyperparamètre, il suffit de donner à *chaque valeur candidate* sa note de validation croisée, puis de garder la valeur la mieux notée — le test n'a pas été consulté, le réglage est honnête.

Le tracé de la note CV en fonction de la valeur s'appelle la **courbe de validation**. Sa forme typique est déjà connue : c'est la courbe en U de la Séance 5 (sous-ajustement d'un côté, sur-ajustement de l'autre) — mais mesurée proprement, par CV, au lieu d'un unique split.

Exemple — l' α de Ridge, version honnête

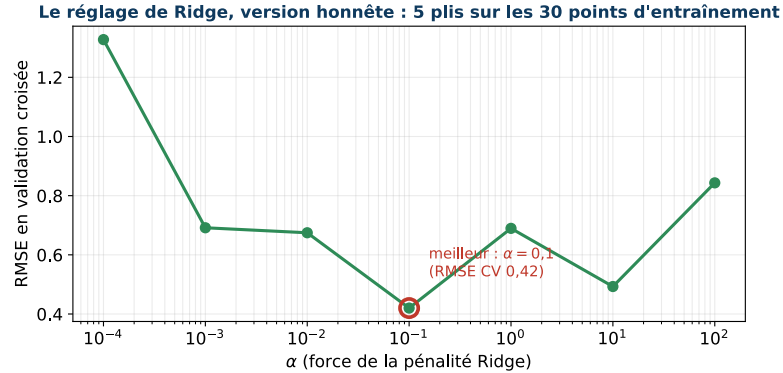
La sinusoïde de la Séance 5, refaite au propre

En Séance 5, l' α du polynôme de degré 15 avait été choisi *en regardant* les 20 points de validation — un réglage sur split unique. Reprise : CV à 5 plis **sur les 30 points d'entraînement seulement** :

α	10^{-4}	10^{-3}	10^{-2}	0,1	1	10	100
RMSE CV	1,33	0,69	0,67	0,42	0,69	0,49	0,84

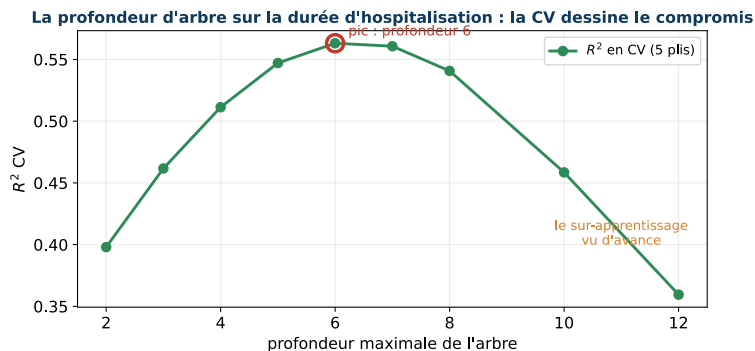
Meilleur : $\alpha = 0,1$. Verdict final sur les 20 points jamais touchés : RMSE = 0,336 — la CV retrouve le bon réglage *sans avoir eu besoin de tricher*.

La courbe de validation de Ridge



La forme générale est le U attendu — trop peu de pénalité : l'explosion du degré 15 ; trop : le sous-ajustement. Le creux local à $\alpha = 1$ rappelle une leçon de la section précédente : avec 30 points seulement, la CV reste une mesure *bruitée* — la tendance se lit, les petites bosses ne se surinterprètent pas.

Exemple — la profondeur d'arbre, vue d'avance



Sur la durée d'hospitalisation : le R^2 de CV monte jusqu'à la profondeur 6 (0,563), plafonne, puis s'effondre (0,36 à 12). La Séance 5 avait montré le sur-apprentissage de la profondeur 10 *après coup*, sur le test (0,726 train contre 0,461 test) ; la CV le voit **à l'avance**, sans dépenser le test.

Le protocole de réglage en trois temps

Régler proprement, toujours pareil

1. Choisir les valeurs candidates et la métrique (Séance 6) → **2.** noter chaque candidate par CV *sur l'entraînement seul* et garder la mieux notée → **3.** réentraîner la recette gagnante sur *tout* l'entraînement, et seulement alors, verdict unique sur le test.

Le score CV de la gagnante et son score test final diffèrent en général légèrement — c'est normal (le premier est un réglage gagnant parmi d'autres, donc un peu optimiste ; le second est la vérité). L'écart Ridge : CV 0,42, test 0,336 ; l'écart arbre : CV 0,569, test 0,554.

Exercice — lire une courbe de validation

Énoncé

Une courbe de validation donne, pour un hyperparamètre de complexité croissante, les scores CV (0,41; 0,52; 0,58; 0,57; 0,44) pour les valeurs (1, 2, 3, 4, 5). Identifier la valeur à retenir, la zone de sous-ajustement et la zone de sur-ajustement.

Correction

Valeur retenue : **3** (pic à 0,58 ; la valeur 4, quasi équivalente à 0,57, serait défendable — à bruit égal, préférer la plus *simple*). Sous-ajustement : valeurs 1-2, le modèle manque de capacité. Sur-ajustement : valeur 5, la chute de 0,57 à 0,44 signe un modèle qui apprend le bruit — le même U que la Séance 5, mesuré sans toucher au test.

GridSearchCV : la grille

L'idée : plusieurs curseurs à la fois

L'idée en une phrase

Quand deux hyperparamètres interagissent (la profondeur *et* la taille minimale des feuilles d'un arbre), on note par CV **chaque combinaison** du produit cartésien — la **grille** — et on garde la meilleure case.

Les régler l'un après l'autre serait hasardeux : le meilleur réglage de l'un dépend souvent de la valeur de l'autre. La grille teste tout, au prix d'un coût qui se compte avant de lancer.

Le mécanisme — la double boucle

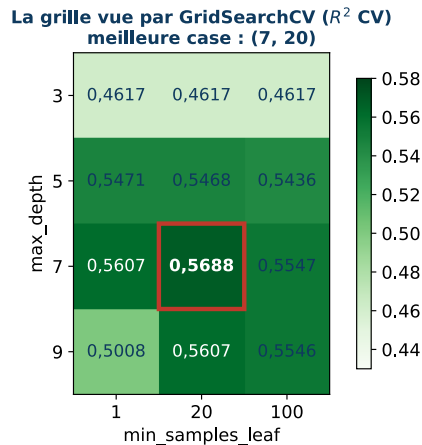
GridSearchCV refait à la main, une case

La grille arbre sur la durée d'hospitalisation : $\text{max_depth} \in \{3, 5, 7, 9\} \times$
 $\text{min_samples_leaf} \in \{1, 20, 100\}$, soit 12 cases. Pour *chaque* case, une CV complète à 5
plis. La case (5, 20), déroulée à la main :

scores des 5 plis = (0,549; 0,532; 0,519; 0,579; 0,554) $\Rightarrow$ moyenne = **0,547**

C'est exactement le `mean_test_score` que `GridSearchCV` inscrit dans cette case. Tout
l'outil tient en deux boucles : *pour chaque combinaison, pour chaque pli* — entraîner, noter,
moyenner.

La grille en chiffres



Meilleure case : `max_depth` = 7, `min_samples_leaf` = 20, R^2 CV = 0,569.

L'interaction se lit dans la colonne 9 : la profondeur 9 n'est bonne *que* si les feuilles sont contraintes (0,501 seule, 0,561 avec 20) — régler les deux séparément aurait pu la manquer.

En pratique : GridSearchCV

```
from sklearn.model_selection import GridSearchCV

grille = {"max_depth": [3, 5, 7, 9], "min_samples_leaf": [1, 20, 100]}
gs = GridSearchCV(DecisionTreeRegressor(random_state=0),
                  grille, cv=5, scoring="r2")
gs.fit(X_train, y_train)      # 12 combinaisons x 5 plis = 60 entraînements
print(gs.best_params_)       # {'max_depth': 7, 'min_samples_leaf': 20}
print(gs.best_score_)        # 0.569 (R2 de CV de la meilleure case)
print(gs.score(X_test, y_test)) # 0.554 : le verdict, une seule fois
```

Après `fit`, l'objet a déjà **réentraîné** la recette gagnante sur tout l'entraînement (`refit=True` par défaut) : `gs` s'utilise directement comme modèle final. Pour une grille sur un Pipeline, préfixer le nom de l'étape : `"logisticregression__C"`.

Le coût se compte avant de lancer

$$\text{nombre d'entraînements} = |\text{grille}| \times k + 1(\text{le refit final})$$

La grille arbre : $12 \times 5 + 1 = 61$ entraînements. Une grille de trois hyperparamètres à dix valeurs chacun en CV 10 plis : $10^3 \times 10 = 10\,001$ — souvent rédhibitoire. Réflexes d'économie : commencer par des grilles *grossières* puis resserrer autour du gagnant ; et garder $k = 5$.

Exercice — budgéter une grille

Énoncé

Une grille croise 4 valeurs de `c`, 2 valeurs de `class_weight` et 5 valeurs de seuil, en CV stratifiée à 5 plis. Compter les combinaisons puis le nombre total d'entraînements (refit compris). Un entraînement prend 2 secondes : la recherche est-elle raisonnable ?

Correction

Combinaisons : $4 \times 2 \times 5 = 40$. Entraînements : $40 \times 5 + 1 = 201$, soit environ 400 secondes — moins de 7 minutes : raisonnable. (Remarque d'expert : le seuil s'applique *après* l'entraînement — les 5 valeurs de seuil peuvent en réalité réutiliser les mêmes modèles, ramenant le vrai coût à $8 \times 5 + 1 = 41$ entraînements. Compter le coût, c'est aussi repérer ces économies.)

Le seuil, réglé proprement

Le fil laissé par la Séance 6

L'idée en une phrase

Le seuil se règle sur des probabilités que le modèle n'a pas apprises par cœur.

`cross_val_predict` fournit exactement cela : pour chaque patient de l'entraînement, la probabilité prédite *par le tour de CV qui ne l'a pas vu* — des probabilités honnêtes, sur tout l'entraînement, sans toucher au test.

Différence avec `cross_val_score` : celui-ci rend k notes ; `cross_val_predict` rend n prédictions (une par observation, chacune venant du tour qui l'excluait). C'est la matière première idéale pour balayer un seuil.

Exemple — le seuil du paludisme par CV

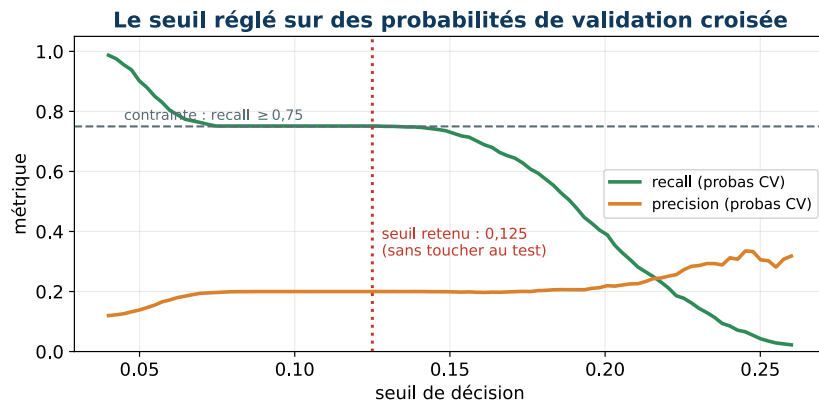
La mission de la Séance 6, protocole complet

Mission : $\text{recall} \geq 0,75$ avec la meilleure precision. **1.** Probabilités CV sur les 8 000 patients d'entraînement (`cross_val_predict`, 5 plis stratifiés). **2.** Balayage des seuils sur ces probabilités : le gagnant est $s = 0,125$ (precision CV 0,200, recall CV 0,751). **3.** Réentraînement sur tout l'entraînement, puis verdict unique sur le test :

$$\text{precision} = 0,205 \quad \text{recall} = \mathbf{0,796}$$

Le seuil a tenu sa promesse sur des données jamais vues — et le test n'a servi qu'à la toute fin, une fois.

La courbe du réglage honnête



Le même graphique qu'en Séance 6 — mais tracé sur les probabilités de *validation croisée*, pas sur le test. La différence est invisible à l'œil et fondamentale en principe : ce balayage-ci avait le droit d'exister *avant* le verdict.

Exercice — trois protocoles, un seul honnête

Énoncé

Trois équipes règlent le seuil du même modèle. **A** : balaye les seuils sur le test, annonce le meilleur. **B** : balaye sur une tranche de validation unique, vérifie sur le test. **C** : balaye sur les probabilités de `cross_val_predict`, vérifie sur le test. Classer les trois protocoles du moins fiable au plus fiable, en justifiant.

Correction

A : invalide — le chiffre annoncé est optimiste par construction (fuite, Séance 6). **B** : valide mais fragile — le seuil dépend du hasard d'une tranche (la variance du début de séance). **C** : le plus fiable — le balayage s'appuie sur n prédictions honnêtes couvrant tout l'entraînement, et le test reste un verdict vierge. Ordre : $A < B < C$.

Les pièges

Piège — standardiser avant de découper

La fuite qui se glisse dans la CV

Standardiser *tout* l'entraînement **puis** lancer la CV est une fuite : la moyenne et l'écart-type utilisés dans chaque tour ont vu les données du pli de validation — le juge a reçu une confiance. La parade est déjà connue : passer le **Pipeline entier** à la CV, qui refait `StandardScaler` à l'intérieur de chaque tour, sur le seul entraînement du tour.

C'est la règle de la Séance 4 — « le préprocesseur s'ajuste sur le train seul » — appliquée récursivement : *chaque tour de CV a son propre train*. Tout ce qui apprend des données (imputation, standardisation, encodage) doit vivre dans le Pipeline, jamais avant.

Piège — la grille optimiste

Le gagnant a aussi eu de la chance

`best_score_` est le maximum de 12 (ou 200) moyennes bruitées : le gagnant gagne en partie par mérite, en partie par chance — son score CV est donc légèrement **optimiste**, et l'optimisme grandit avec la taille de la grille. C'est précisément pour cela que le verdict final se prend sur le test, jamais sur `best_score_`.

Les chiffres du jour l'illustrent sans drame : grille arbre, CV 0,569 contre test 0,554 ; Ridge, CV 0,42 contre test 0,336 (ici l'écart joue dans l'autre sens — le bruit va dans les deux directions, l'optimisme n'est qu'une *tendance*).

Piège — le test consulté « juste pour voir »

Chaque consultation dépense le test

Regarder le test après chaque essai, « pour information », puis continuer à régler, recrée la fuite en silence : les choix suivants sont influencés par ce qui a été vu. Discipline du cours : le test est **scellé** pendant tout le développement ; une seule ouverture, à la fin, et le chiffre obtenu est définitif — bon ou mauvais.

Si le verdict final déçoit, la tentation est de « réessayer un peu » puis de re-tester : à ce moment le test est devenu une validation, et il faudrait de nouvelles données pour juger. Mieux vaut prévenir : ne desceller qu'une fois sûr.

Synthèse

Le bilan : tous les réglages du cours, un seul protocole

Ce que la séance a réglé, chiffres en main

réglage	méthode	choix par CV	verdict test
α de Ridge (sinusoïde S5)	courbe de validation	$\alpha = 0,1$	RMSE 0,336
profondeur d'arbre (durée)	courbe de validation	6	—
arbre : profondeur $\times$ feuilles	GridSearchCV	(7, 20)	R^2 0,554
seuil de la logistique (palu)	cross_val_predict	0,125	recall 0,796

Quatre réglages, quatre outils — un seul principe : noter par CV sur l'entraînement, geler, juger une fois.

Le protocole complet, de bout en bout

Le rituel à dérouler sur tout nouveau problème

1. Split stratifié train/test, le test au coffre → **2.** baseline (Séances 3 et 6) → **3.** la recette dans un **Pipeline** (préprocesseurs compris) → **4.** CV stratifiée à 5 plis pour estimer, courbes de validation ou **GridSearchCV** pour régler, `cross_val_predict` pour le seuil → **5.** réentraîner la recette gagnante sur tout le train → **6.** ouvrir le test, une fois : métriques alignées sur la mission (Séance 6), chiffre définitif.

À retenir — Séance 7

L'essentiel

- Trois usages, trois jeux : ce qui **règle** ne peut plus **juger** ; le test ne sert qu'une fois.
- Un score sur un split unique est une mesure **bruitée** (étendue 0,016 sur le paludisme) — comparer des recettes exige des moyennes.
- La **CV à k plis** : k entraînements, chaque donnée juge une fois, verdict = moyenne $\pm$ écart-type ; elle évalue une *recette*, le modèle final se réentraîne sur tout.
- Classification $\Rightarrow$ **plis stratifiés** ; $k = 5$ par défaut.
- **Courbe de validation** pour un curseur, **GridSearchCV** pour plusieurs (coût = $|\text{grille}| \times k + 1$), **cross_val_predict** pour le seuil.
- Le Pipeline entre *dans* la CV — standardiser avant les plis est une fuite ; `best_score_` est optimiste — le test tranche.

Pièges fréquents

- Régler un hyperparamètre (ou un seuil) sur le test — la fuite originelle.
- Standardiser ou imputer *avant* la CV au lieu de l'inclure dans le Pipeline.
- Oublier la stratification sur une classe rare : des plis qui jugent des problèmes différents.
- Croire que la CV produit un modèle — elle note une recette ; le modèle final se réentraîne.
- Prendre `best_score_` pour le niveau réel : c'est un maximum de mesures bruitées, donc optimiste.
- Surinterpréter les petites bosses d'une courbe de validation bruitée (les 30 points de Ridge).
- Lancer une grille sans compter ses entraînements.
- Rouvrir le test « juste pour voir » après un premier verdict.

Prochaine séance

Séance 8 — La force du nombre : les méthodes d'ensemble

Le mini-défi du TP règle un arbre sur le paludisme par CV : bien réglé, il rivalise avec la logistique — puis plafonne. La Séance 8 dépasse ce plafond par une idée simple et puissante : entraîner *beaucoup* d'arbres et les faire voter — les forêts aléatoires, puis le boosting.

Références

- James, Witten, Hastie, Tibshirani — *An Introduction to Statistical Learning*, 2^e éd., Springer, 2021.
- Géron — *Hands-On Machine Learning*, 3^e éd., O'Reilly, 2022.
- Müller, Guido — *Introduction to Machine Learning with Python*, O'Reilly, 2016.
- Cours : University of Washington CSE 446 ; Stanford CS229.

Ressources de la séance

- **Notebook** · [TP 7 — Validation croisée](#) (81 Ko)
- **TD** · [Fiche d'exercices 7](#) (180 Ko)