Chapitre 20 : Validation Croisée et Hyperparamètres

22 février 2026

Plan du chapitre

Le découpage classique (Train / Test)

Jusqu'ici, nous avons évalué nos modèles en séparant l'ensemble de données $\mathscr{S}$ en deux : un ensemble d'entraînement ($\mathscr{S}_{\text{train}}$) et un ensemble de test ($\mathscr{S}_{\text{test}}$).

La limite de cette approche

- **Variance de l'évaluation :** Si le jeu de données est petit, la performance mesurée dépendra fortement de *comment* le découpage aléatoire a été fait. Un "coup de chance" (ou de malchance) biaise notre jugement.
- **Gaspillage de données :** Les données gardées pour le test ne sont jamais utilisées pour entraîner le modèle final, ce qui est critique quand la donnée est rare.

Le piège de la Fuite de Données (Data Leakage)

Les modèles possèdent des **hyperparamètres** : la profondeur d'un arbre, le λ du Lasso, le C ou le noyau d'un SVM.

Si l'on entraîne plusieurs modèles avec différents hyperparamètres et que l'on choisit celui qui a le meilleur score sur l'ensemble de **Test... on vient d'incorporer l'information du Test dans le modèle !**

Règle d'Or en Machine Learning

L'ensemble de Test ne doit servir **qu'une seule et unique fois**, tout à la fin du projet. L'optimisation des hyperparamètres nécessite donc un troisième sous-ensemble : l'**ensemble de Validation**.

La Validation Croisée à K blocs (K-Fold CV)

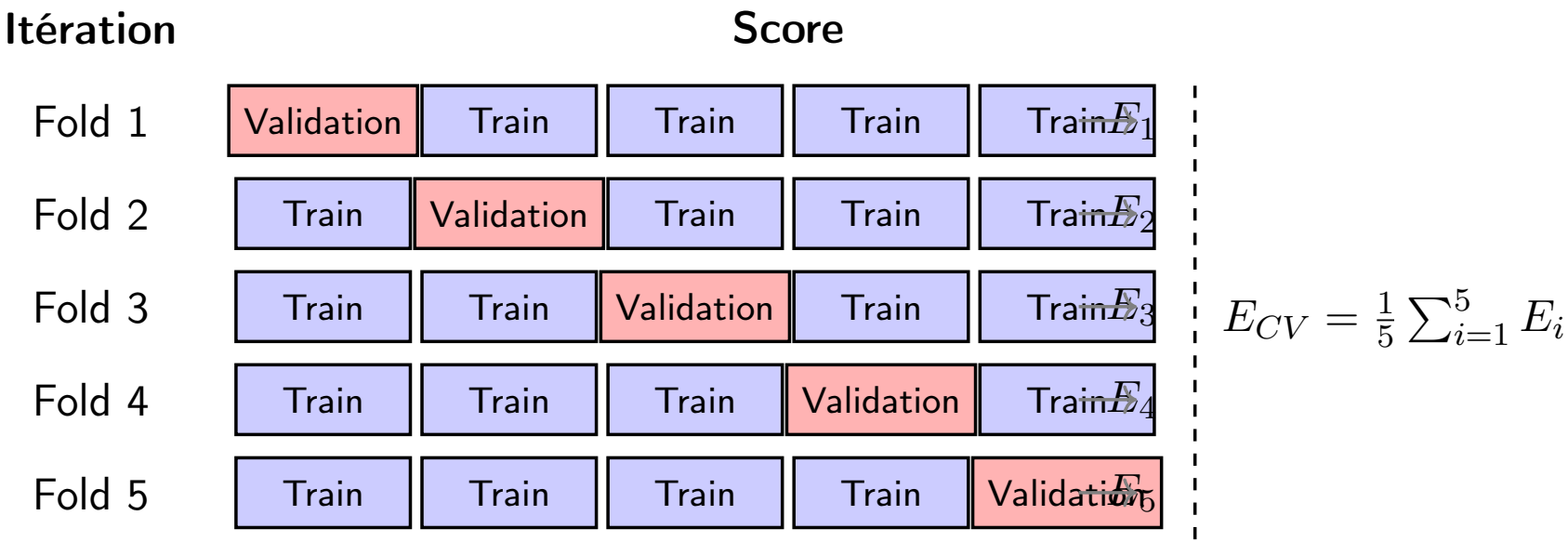
Pour pallier la rareté des données et la variance de l'évaluation, on utilise la **Validation Croisée K-Fold**.

Algorithme :

1. Diviser les données d'entraînement aléatoirement en K blocs (folds) de tailles égales.
2. Pour k allant de 1 à K :
   - Entraîner le modèle sur $K-1$ blocs (fusionnés).
   - Évaluer le modèle sur le bloc k restant (le pli de validation).
3. La performance finale estimée est la **moyenne** des K scores obtenus.

Valeurs typiques : $K = 5$ *ou* $K = 10$.

Illustration : K-Fold (K=5)



Avantages : Chaque donnée sert exactement une fois à valider et $K-1$ fois à entraîner. L'estimation de l'erreur est beaucoup plus robuste.

Inconvénient : Le temps de calcul est multiplié par K.

Le cas extrême : Leave-One-Out (LOOCV)

La validation **Leave-One-Out** (Laisser un de côté) est le cas limite où $K = n$ (le nombre total de données).
À chaque itération, on entraîne le modèle sur $n - 1$ données, et on le teste sur la **seule donnée restante**.

Avantages et Inconvénients

- **Avantage :** Le biais de l'estimateur de l'erreur est quasi nul (le modèle s'entraîne sur presque toutes les données).
- **Inconvénient 1 :** Temps de calcul catastrophique (nécessite d'entraîner n modèles).
- **Inconvénient 2 :** Les n ensembles d'entraînement sont presque identiques. Les modèles sont donc fortement corrélés, ce qui entraîne une **très forte variance** statistique sur l'erreur estimée.

K-Fold Stratifié

Le problème du déséquilibre

Si un jeu de données contient 90% de classe A et 10% de classe B, un découpage K-Fold purement aléatoire pourrait, par malchance, créer un pli de validation qui ne contient **aucun** exemple de la classe B !

Solution : Stratified K-Fold

L'algorithme de découpage force chaque pli (fold) à respecter rigoureusement la distribution d'origine des classes.

Si le ratio global est 90/10, **chaque** bloc d'entraînement et de validation aura exactement 90% de classe A et 10% de classe B.

C'est une étape absolument obligatoire en classification médicale, bancaire, etc.

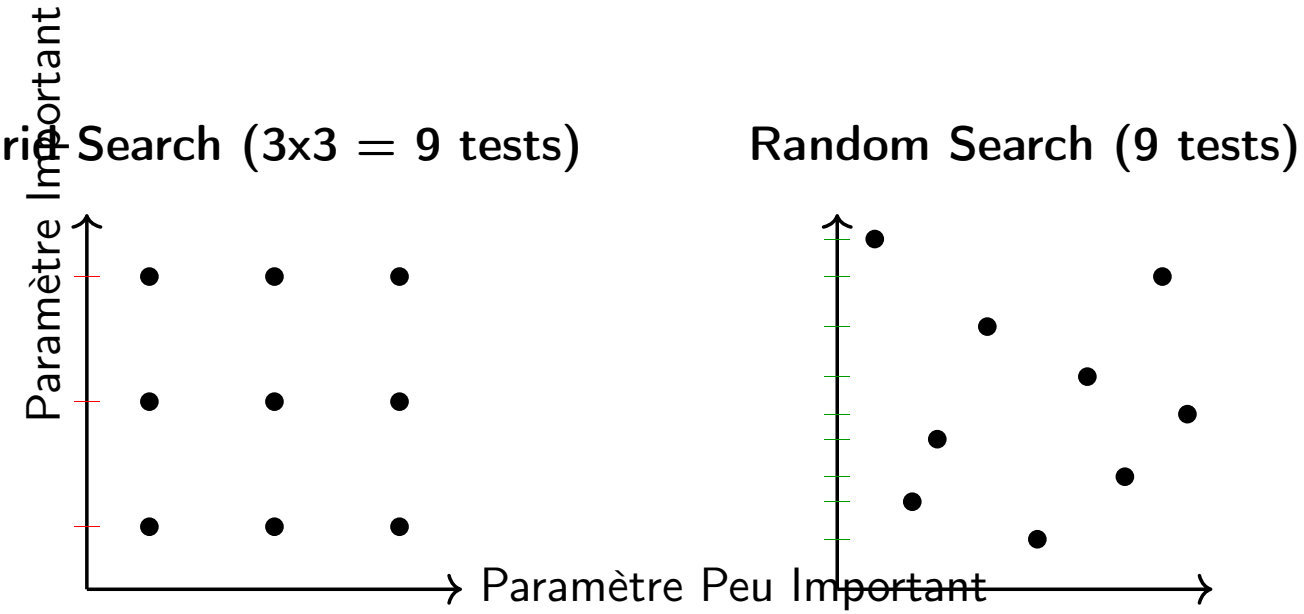
La recherche de la configuration optimale

La Validation Croisée nous permet d'évaluer de manière robuste *une* configuration d'hyperparamètres (ex : SVM avec Noyau RBF, $C = 10$, $\gamma = 0.1$).
Comment trouver la **meilleure** configuration possible ?

- **Recherche sur grille (Grid Search) :** On définit une liste finie de valeurs pour chaque paramètre, et on teste (par K-Fold) *toutes les combinaisons possibles*.
- **Recherche aléatoire (Random Search) :** On tire des valeurs au hasard dans des distributions continues.

Grid Search vs Random Search (Bergstra & Bengio)

Pourquoi Random Search est mathématiquement plus efficace en grande dimension ?



Le théorème : Souvent, seuls quelques hyperparamètres (axe vertical) contrôlent vraiment l'erreur. Avec le même budget de calcul (9 essais), la grille n'a exploré que **3 valeurs uniques** pour le paramètre important (marques rouges). Le tirage aléatoire a exploré **9 valeurs uniques** (marques vertes), maximisant les chances de trouver l'optimum !

Au-delà de l'aléatoire : L'Optimisation Bayésienne

Grid et Random Search sont "aveugles" : ils ne tirent aucune leçon de leurs échecs précédents.

L'Optimisation Bayésienne (ex : Hyperopt, Optuna)

Elle modélise la fonction objectif (inconnue et coûteuse à calculer) par un **Processus Gaussien**. À chaque itération, elle choisit le prochain hyperparamètre à tester en équilibrant :

- **Exploitation** : Tester autour des valeurs qui ont déjà donné de bons résultats.
- **Exploration** : Tester dans les zones d'incertitude (où le modèle n'a pas encore cherché).

C'est la méthode de pointe absolue pour régler les hyperparamètres des grands modèles (XGBoost, Deep Learning).