INTRODUCTION À LA SCIENCE DES DONNÉES

Apprentissage Supervisé

Méthodes d'ensembles et Forêts aléatoires

Victor Bertret

Docteur en Mathématiques

Ingénieur IA chez **Purecontrol**

2 mars 2026

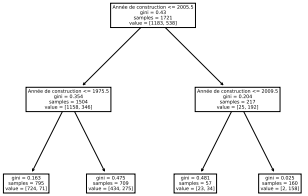
Sommaire de la séance

Flashback S2 : L'Arbre de Décision (Le bon côté)

Rappel : Nous avons vu comment construire un Arbre de Décision via l'algorithme CART.

Modèle "Boîte Blanche"

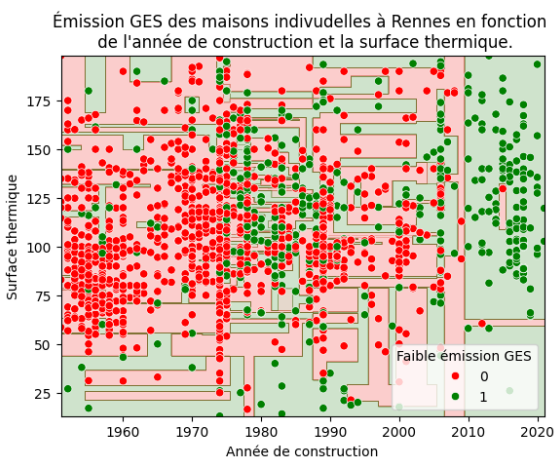
- **Intuitif :** Mimétisme avec le raisonnement humain.
- **Robuste :** Insensible aux échelles (pas de normalisation).
- **Non-Linéaire :** Découpe l'espace en hyper-rectangles.



Des règles simples : "Si $X_1 > 24.5$ et $X_2 < 52$..."

Le Défaut Mortel : La Dentelle

Problème : Si on laisse l'arbre grandir sans contrainte (Profondeur = ∞).

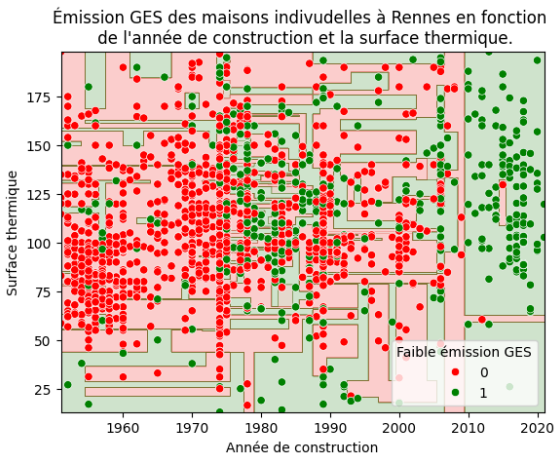


OVERFITTING !

- L'arbre a isolé chaque point individuellement.
- Il a appris le **bruit** par cœur.
- La frontière de décision n'a plus de **sens physique**.

Le Défaut Mortel : La Dentelle

Problème : Si on laisse l'arbre grandir sans contrainte (Profondeur = ∞).



OVERFITTING !

- L'arbre a isolé chaque point individuellement.
- Il a appris le **bruit** par cœur.
- La frontière de décision n'a plus de **sens physique**.

Capacité de généralisation $\approx$ Nulle.

Diagnostic Théorique : Biais vs Variance

Souvenez-vous du compromis vu en Séance 1... Où se situe un Arbre profond ?

+

✔ Biais Très Faible

L'arbre est **extrêmement flexible**.

Il peut "se tordre" pour s'adapter à n'importe quelle forme de donnée. L'erreur d'entraînement $\hat{R}_S$ peut atteindre 0.

💣 Variance Explosive

L'arbre est **hyper-sensible**.

Si je modifie un seul point dans mon dataset S, la première coupure à la racine peut changer, modifiant **toute la structure** en cascade.

Diagnostic Théorique : Biais vs Variance

Souvenez-vous du compromis vu en Séance 1... Où se situe un Arbre profond ?

+

✔ Biais Très Faible

L'arbre est **extrêmement flexible**.

Il peut "se tordre" pour s'adapter à n'importe quelle forme de donnée. L'erreur d'entraînement $\hat{R}_S$ peut atteindre 0.

💣 Variance Explosive

L'arbre est **hyper-sensible**.

Si je modifie un seul point dans mon dataset S, la première coupure à la racine peut changer, modifiant **toute la structure** en cascade.

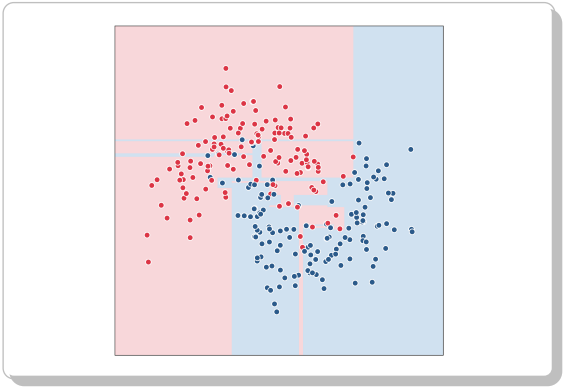
🔍 La Question Centrale d'Aujourd'hui :

Comment casser cette variance sans perdre le bénéfice du biais faible ?

Preuve Visuelle : L'instabilité de l'Arbre

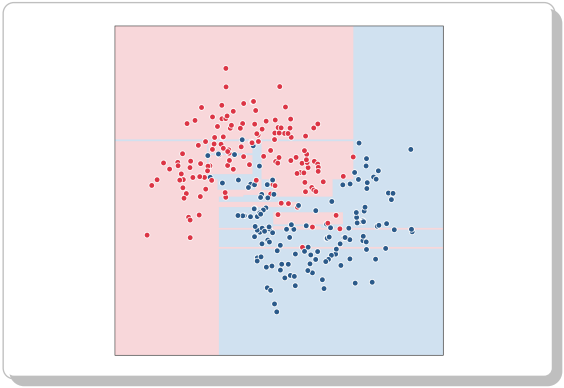
🔀 **Expérience :** On prend le même jeu de données. On retire **seulement 10%** des points au hasard. Voici les arbres générés.

Arbre A (Tirage 1)



≠

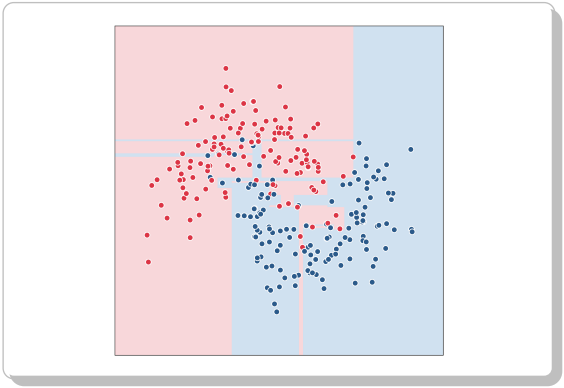
Arbre B (Tirage 2)



Preuve Visuelle : L'instabilité de l'Arbre

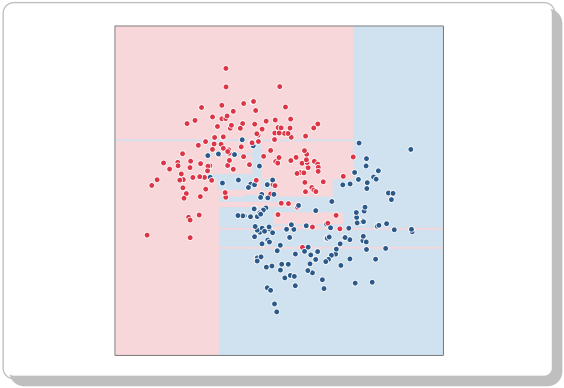
🔀 **Expérience :** On prend le même jeu de données. On retire **seulement 10%** des points au hasard. Voici les arbres générés.

Arbre A (Tirage 1)



≠

Arbre B (Tirage 2)



⚠ Un algorithme sur lequel on ne peut pas compter d'un tirage à l'autre n'est pas fiable en production.

Le Dilemme : Comment exploiter cette instabilité ?

Question : Face à un problème complexe, vous avez généré 100 arbres de décision légèrement différents. Comment obtenir la prédiction finale ?

Option A

Sélectionner le **MEILLEUR** arbre.

(Celui qui a la plus petite erreur sur les données de validation).

OU

Option B

Faire la **MOYENNE** de tous les arbres.

(Combiner les prédictions des 100 modèles).

Que choisissez-vous et pourquoi ?

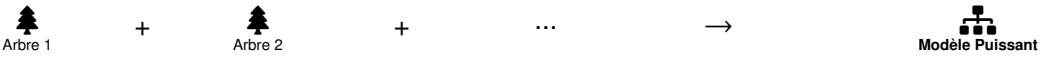
La Solution : Les Méthodes d'Ensemble

✗ Pourquoi l'Option A échoue :
Chercher "le meilleur", c'est chercher celui qui a le plus **sur-appris**. Vous allez juste sélectionner l'arbre qui a eu le plus de chance sur son tirage.

✓ Pourquoi l'Option B gagne :
L'arbre a une forte variance. En faisant la **moyenne**, les erreurs aléatoires (le bruit) des différents arbres **s'annulent**.

"L'Union fait la Force"

Le Principe des Méthodes d'Ensemble (Ensembling) :
Au lieu d'essayer de construire un **unique** modèle parfait (qui va fatalement sur-apprendre), on va construire **des dizaines ou centaines** de "mauvais" modèles (instables), et les faire voter.



Analogie : La "Sagesse des Foules"

Le Théorème du Jury

Nicolas de Condorcet, 1785

Imaginez un tribunal. Chaque juré a une probabilité p de prendre la bonne décision.

- Si $p > 50\%$ (ils sont juste un peu meilleurs que le hasard)...
- ... alors la probabilité que la **majorité** ait raison tend vers **100%** avec le nombre de jurés !

L'Avis du Public

"Qui veut gagner des millions ?"

Un individu seul, même expert, peut se tromper lourdement.

→ La moyenne de la foule trouve presque toujours la vérité.

Analogie : La "Sagesse des Foules"

Le Théorème du Jury

Nicolas de Condorcet, 1785

Imaginez un tribunal. Chaque juré a une probabilité p de prendre la bonne décision.

- Si $p > 50\%$ (ils sont juste un peu meilleurs que le hasard)...
- ... alors la probabilité que la **majorité** ait raison tend vers **100%** avec le nombre de jurés !

L'Avis du Public

"Qui veut gagner des millions ?"

Un individu seul, même expert, peut se tromper lourdement.

→ **La moyenne de la foule trouve presque toujours la vérité.**

Le secret : L'indépendance des erreurs.

Mieux vaut 100 modèles "moyens" qui font des erreurs **différentes**, qu'un seul modèle "expert" très biaisé.

Le mur de la réalité : Comment construire cette "Foule" ?

Protocole proposé

On donne notre jeu de données d'entraînement S à l'algorithme CART 100 fois de suite, puis on fait la moyenne des 100 prédictions.

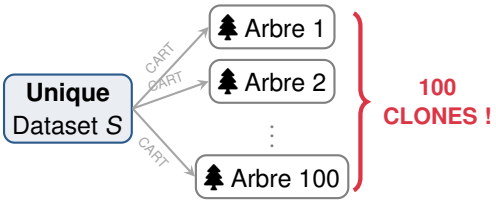
À votre avis, que va-t-il se passer ? Cela va-t-il fonctionner ?

Le mur de la réalité : Comment construire cette "Foule" ?

Protocole proposé

On donne notre jeu de données d'entraînement S à l'algorithme CART 100 fois de suite, puis on fait la moyenne des 100 prédictions.

À votre avis, que va-t-il se passer ? Cela va-t-il fonctionner ?



✗ Échec total !

L'algorithme CART est **déterministe**. À données égales, il fera toujours rigoureusement les mêmes choix de coupure (Gini/Variance).

→ *Faire voter 100 clones n'apporte strictement aucune information nouvelle.*

Le Défi : Créer de la diversité artificielle

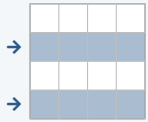
Pour que le vote soit utile, il faut que nos modèles soient **différents** et fassent des erreurs **indépendantes**.

Le Défi : Créer de la diversité artificielle

Pour que le vote soit utile, il faut que nos modèles soient **différents** et fassent des erreurs **indépendantes**.

Diversité n°1 : Les Lignes

Modifier les **Données** d'entraînement pour chaque arbre.



Le Bootstrap

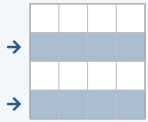
(Tirer un sous-ensemble des **données** *n*)

Le Défi : Créer de la diversité artificielle

Pour que le vote soit utile, il faut que nos modèles soient **différents** et fassent des erreurs **indépendantes**.

Diversité n°1 : Les Lignes

Modifier les **Données** d'entraînement pour chaque arbre.

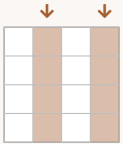


Le Bootstrap

(Tirer un sous-ensemble des **données** *n*)

Diversité n°2 : Les Colonnes

Modifier les **Variables** observées par chaque arbre.



Le Feature Bagging

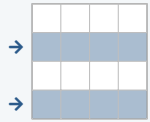
(Tirer un sous-ensemble des **variables** *p*)

Le Défi : Créer de la diversité artificielle

Pour que le vote soit utile, il faut que nos modèles soient **différents** et fassent des erreurs **indépendantes**.

Diversité n°1 : Les Lignes

Modifier les **Données** d'entraînement pour chaque arbre.

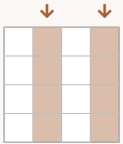


Le Bootstrap

(Tirer un sous-ensemble des **données** n)

Diversité n°2 : Les Colonnes

Modifier les **Variables** observées par chaque arbre.



Le Feature Bagging

(Tirer un sous-ensemble des **variables** p)

La combinaison de ces deux astuces crée la Forêt Aléatoire.

Sommaire

Créer de la diversité : Le Bootstrap

Le Bootstrap : Technique statistique consistant à créer de nouveaux échantillons à partir d'un échantillon unique, par **tirage aléatoire avec remise**.

La Règle du jeu

À partir de votre jeu de données S de taille n :

- Vous tirez **au hasard** un élément.
- Vous le copiez dans le nouveau dataset S'.
- **Vous le remettez** dans S (il peut être tiré à nouveau !).
- Vous répétez l'opération n fois.

Créer de la diversité : Le Bootstrap

Le Bootstrap : Technique statistique consistant à créer de nouveaux échantillons à partir d'un échantillon unique, par **tirage aléatoire avec remise**.

La Règle du jeu

À partir de votre jeu de données S de taille n :

- Vous tirez **au hasard** un élément.
- Vous le copiez dans le nouveau dataset S'.
- **Vous le remettez** dans S (il peut être tiré à nouveau !).
- Vous répétez l'opération n fois.

Conséquences directes

Le nouveau dataset S' aura exactement la même taille n que S, **mais** :

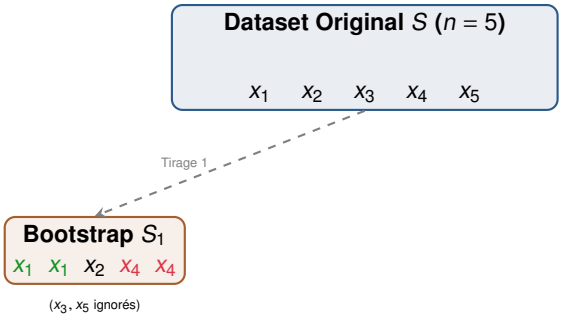
- Certains éléments apparaîtront **plusieurs fois** (Doublons).
- Certains éléments de S ne seront **jamais tirés**.

Mécanique du Bootstrap (Illustration)

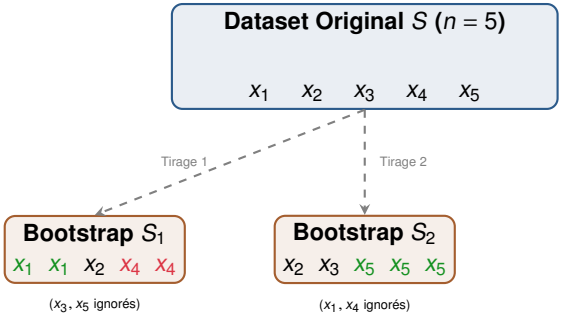
Dataset Original S ($n = 5$)

x_1 x_2 x_3 x_4 x_5

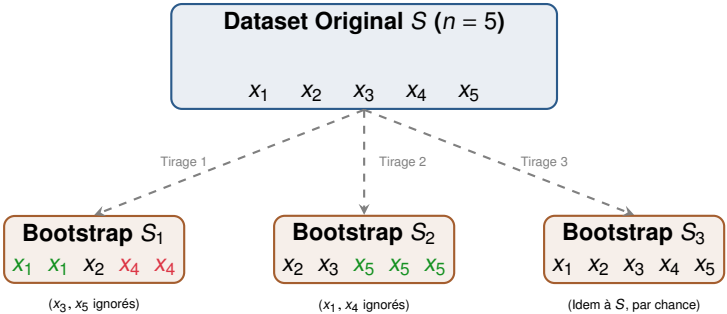
Mécanique du Bootstrap (Illustration)



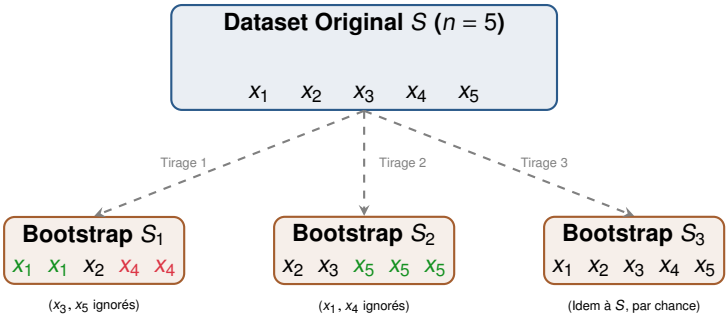
Mécanique du Bootstrap (Illustration)



Mécanique du Bootstrap (Illustration)



Mécanique du Bootstrap (Illustration)



🌲 **Conséquence :** Si on entraîne un algorithme sur S_1, S_2 et S_3, on obtiendra 3 modèles structurellement **différents** !

L'instant Mathématique : L'Échantillon Oublié

La Question Théorique

Soit un jeu de données S contenant n observations.

Si l'on crée un échantillon Bootstrap (tirage de n éléments avec remise), **quelle est la probabilité formelle qu'une observation spécifique x_i ne soit JAMAIS tirée ?**

L'instant Mathématique : L'Échantillon Oublié

La Question Théorique

Soit un jeu de données S contenant n observations.

Si l'on crée un échantillon Bootstrap (tirage de n éléments avec remise), **quelle est la probabilité formelle qu'une observation spécifique x_i ne soit JAMAIS tirée ?**

Sortez vos cahiers, on calcule la limite quand $n \to \infty$...

Démonstration : La limite asymptotique

Calculons la probabilité qu'un point x_i soit totalement ignoré.

1. Un seul tirage

Le jeu de données S contient n individus uniques.

Lors d'un tirage aléatoire uniforme, on a exactement 1 chance sur n de piocher l'individu x_i.

Probabilité de rater x_i :

$$p = 1 - \frac{1}{n}$$

Démonstration : La limite asymptotique

Calculons la probabilité qu'un point x_i soit totalement ignoré.

1. Un seul tirage

Le jeu de données S contient n individus uniques.

Lors d'un tirage aléatoire uniforme, on a exactement 1 chance sur n de piocher l'individu x_i.

Probabilité de rater x_i :

$$p = 1 - \frac{1}{n}$$

2. Sur n tirages

Le Bootstrap effectue n tirages **avec remise**.

Les tirages sont donc strictement **indépendants**. On multiplie les probabilités à chaque itération.

Probabilité de toujours rater x_i :

$$P = \left(1 - \frac{1}{n}\right)^n$$

Démonstration : La limite asymptotique

Calculons la probabilité qu'un point x_i soit totalement ignoré.

1. Un seul tirage

Le jeu de données S contient n individus uniques.

Lors d'un tirage aléatoire uniforme, on a exactement 1 chance sur n de piocher l'individu x_i.

Probabilité de rater x_i :

$$p = 1 - \frac{1}{n}$$

2. Sur n tirages

Le Bootstrap effectue n tirages **avec remise**.

Les tirages sont donc strictement **indépendants**. On multiplie les probabilités à chaque itération.

Probabilité de toujours rater x_i :

$$P = \left(1 - \frac{1}{n}\right)^n$$

3. Passage à la limite

En Machine Learning, n est souvent très grand (des milliers de lignes).

La définition classique de la fonction exponentielle nous donne :

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = e^{-1}$$

Bilan : $\frac{1}{e} \approx$ **0.368**

soit **36.8%**

Démonstration : La limite asymptotique

Calculons la probabilité qu'un point x_i soit totalement ignoré.

1. Un seul tirage

Le jeu de données S contient n individus uniques.

Lors d'un tirage aléatoire uniforme, on a exactement 1 chance sur n de piocher l'individu x_i.

Probabilité de rater x_i :

$$p = 1 - \frac{1}{n}$$

2. Sur n tirages

Le Bootstrap effectue n tirages **avec remise**.

Les tirages sont donc strictement **indépendants**. On multiplie les probabilités à chaque itération.

Probabilité de toujours rater x_i :

$$P = \left(1 - \frac{1}{n}\right)^n$$

3. Passage à la limite

En Machine Learning, n est souvent très grand (des milliers de lignes).

La définition classique de la fonction exponentielle nous donne :

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = e^{-1}$$

Bilan : $\frac{1}{e} \approx$ **0.368**
soit **36.8%**

Conclusion : Lors d'un bootstrap, environ **36.8% du dataset n'est jamais vu** par le modèle !

Bilan du Bootstrap : La Règle des 63%

Statistiquement, lors d'un tirage Bootstrap sur un grand jeu de données :

"In-Bag"

63.2 %

Des données originales uniques.

"Out-Of-Bag" (OOB)

36.8 %

Oubliées par le tirage.

Bilan du Bootstrap : La Règle des 63%

Statistiquement, lors d'un tirage Bootstrap sur un grand jeu de données :

"In-Bag"

63.2 %

Des données originales uniques.

"Out-Of-Bag" (OOB)

36.8 %

Oubliées par le tirage.

Notez bien ce concept d'OOB (Out-Of-Bag).
Il va nous offrir un **Set de Validation gratuit** plus tard !

À quoi sert vraiment le Bootstrap en statistiques ?

Avant le Machine Learning, le Bootstrap (B. Efron, 1979) a révolutionné les statistiques.

Le Problème Classique

J'ai récolté **un seul** jeu de données S (ex : 100 patients) et j'ai calculé une statistique (ex : la moyenne).

Question : Quelle est la marge d'erreur (variance) de cette estimation ?

✗ Impasse : Il est impossible (ou trop cher) de retourner chercher 1000 autres patients.

À quoi sert vraiment le Bootstrap en statistiques ?

Avant le Machine Learning, le Bootstrap (B. Efron, 1979) a révolutionné les statistiques.

Le Problème Classique

J'ai récolté **un seul** jeu de données S (ex : 100 patients) et j'ai calculé une statistique (ex : la moyenne).

Question : Quelle est la marge d'erreur (variance) de cette estimation ?

✗ Impasse : Il est impossible (ou trop cher) de retourner chercher 1000 autres patients.

L'Astuce du Bootstrap

On fait **comme si** S était la population entière ($\mathcal{D}$).

- Générer 1000 échantillons Bootstrap.
- Calculer la moyenne pour **chacun**.
- Observer la distribution finale !

✓ On obtient un **Intervalle de Confiance** sans nouvelles données.

À quoi sert vraiment le Bootstrap en statistiques ?

Avant le Machine Learning, le Bootstrap (B. Efron, 1979) a révolutionné les statistiques.

Le Problème Classique

J'ai récolté **un seul** jeu de données S (ex : 100 patients) et j'ai calculé une statistique (ex : la moyenne).

Question : Quelle est la marge d'erreur (variance) de cette estimation ?

✗ Impasse : Il est impossible (ou trop cher) de retourner chercher 1000 autres patients.

L'Astuce du Bootstrap

On fait **comme si** S était la population entière ($\mathcal{D}$).

- Générer 1000 échantillons Bootstrap.
- Calculer la moyenne pour **chacun**.
- Observer la distribution finale !

✓ On obtient un **Intervalle de Confiance** sans nouvelles données.

→ Le pont vers le Machine Learning :
Au lieu de calculer un *chiffre* (moyenne) sur chaque échantillon...
on va **entraîner un modèle** (Arbre) sur chaque échantillon !

L'Algorithme Bagging (Bootstrap Aggregating)

Bootstrap **AGG**regat**ING** → **BAGGING** (Leo Breiman, 1996)

</> L'Algorithme en 2 phases

Soit $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ l'ensemble d'apprentissage.

Phase 1 : BOOTSTRAP

Pour $b = 1, \ldots, B$ **faire :**

1. Tirer aléatoirement **avec remise** un échantillon $S^{(b)}$ de taille n.
2. Entraîner un modèle "instable" $h^{(b)}$ (ex : Arbre de Décision profond) sur $S^{(b)}$.

Fin Pour

Phase 2 : AGGREGATING (L'Agrégation)

Régression

Moyenne des B prédictions.

$$\hat{y}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} h^{(b)}(x)$$

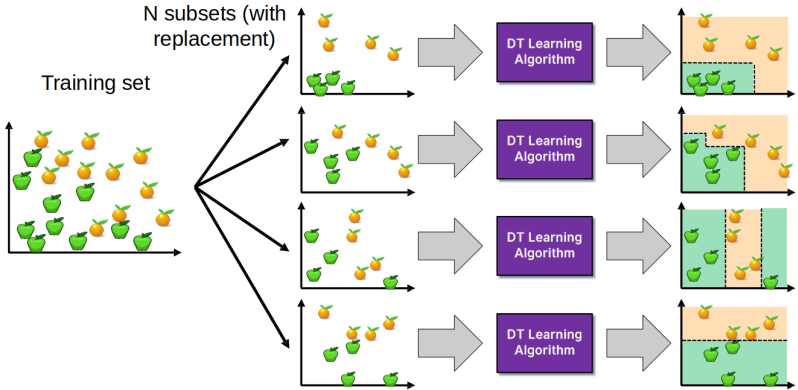
Classification

Vote majoritaire.

$$\hat{y}_{bag}(x) = \text{argmax}_k \sum_{b=1}^{B} \mathbb{1}_{\{h^{(b)}(x)=k\}}$$

Le Bagging en image : Phase d'Entraînement

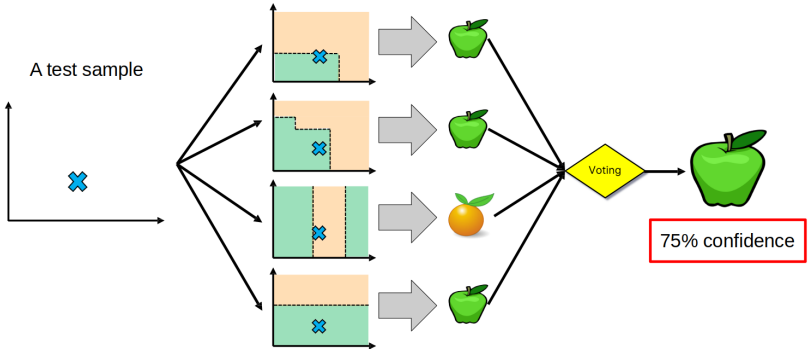
⚙ Chaque sous-échantillon va forcer l'algorithme CART à créer des frontières de décisions différentes pour le même problème global.



Génération de N arbres différents à partir de N tirages Bootstrap.

Le Bagging en image : Phase d'Inférence (La Prédiction)

▶ Le nouvel objet à classer (test sample) est envoyé simultanément dans tous les arbres. Le vote majoritaire l'emporte, et donne un score de "confiance".



Ici, 3 arbres disent "Pomme Verte", 1 arbre dit "Orange". Résultat : Pomme Verte (Confiance 75%).

Pourquoi agréger réduit la variance ? (La Théorie)

Hypothèse Idéale : Supposons que nous avons B estimateurs (arbres) $\hat{h}_1, \ldots, \hat{h}_B$ qui sont **parfaitement indépendants** et identiquement distribués (i.i.d), avec chacun une variance σ^2.

Règle de l'Espérance (Biais)

Règle de la Variance

Pourquoi agréger réduit la variance ? (La Théorie)

Hypothèse Idéale : Supposons que nous avons B estimateurs (arbres) $\hat{h}_1, \ldots, \hat{h}_B$ qui sont **parfaitement indépendants** et identiquement distribués (i.i.d), avec chacun une variance σ^2.

Règle de l'Espérance (Biais)

L'espérance d'une somme/moyenne est égale à la moyenne des espérances :

$$\mathbb{E}[\hat{h}_{bag}] = \mathbb{E}\left[\frac{1}{B}\sum_{b=1}^{B}\hat{h}_b\right] = \mathbb{E}[\hat{h}_1]$$

→ Le Bagging **ne change pas** le Biais moyen.

Règle de la Variance

Pourquoi agréger réduit la variance ? (La Théorie)

Hypothèse Idéale : Supposons que nous avons B estimateurs (arbres) $\hat{h}_1, \ldots, \hat{h}_B$ qui sont **parfaitement indépendants** et identiquement distribués (i.i.d), avec chacun une variance σ^2.

Règle de l'Espérance (Biais)

L'espérance d'une somme/moyenne est égale à la moyenne des espérances :

$$\mathbb{E}[\hat{h}_{bag}] = \mathbb{E}\left[\frac{1}{B}\sum_{b=1}^{B}\hat{h}_b\right] = \mathbb{E}[\hat{h}_1]$$

→ Le Bagging **ne change pas** le Biais moyen.

Règle de la Variance

Si les variables sont indépendantes :

$$\begin{aligned}\mathrm{Var}(\hat{h}_{bag}) &= \mathrm{Var}\left(\frac{1}{B}\sum_{b=1}^{B}\hat{h}_b\right) \\ &= \frac{1}{B^2}\sum_{b=1}^{B}\mathrm{Var}(\hat{h}_b) \\ &= \mathbf{\frac{1}{B}\sigma^2}\end{aligned}$$

Pourquoi agréger réduit la variance ? (La Théorie)

Hypothèse Idéale : Supposons que nous avons B estimateurs (arbres) $\hat{h}_1, \ldots, \hat{h}_B$ qui sont **parfaitement indépendants** et identiquement distribués (i.i.d), avec chacun une variance σ^2.

Règle de l'Espérance (Biais)

L'espérance d'une somme/moyenne est égale à la moyenne des espérances :

$$\mathbb{E}[\hat{h}_{bag}] = \mathbb{E}\left[\frac{1}{B}\sum_{b=1}^{B}\hat{h}_b\right] = \mathbb{E}[\hat{h}_1]$$

→ Le Bagging **ne change pas** le Biais moyen.

Règle de la Variance

Si les variables sont indépendantes :

$$\begin{aligned}\mathrm{Var}(\hat{h}_{bag}) &= \mathrm{Var}\left(\frac{1}{B}\sum_{b=1}^{B}\hat{h}_b\right) \\ &= \frac{1}{B^2}\sum_{b=1}^{B}\mathrm{Var}(\hat{h}_b) \\ &= \mathbf{\frac{1}{B}\sigma^2}\end{aligned}$$

✔ **Victoire mathématique :** Si on met $B = 100$ arbres, on **divise la variance par 100** !

Le Piège : L'illusion de l'indépendance

Malheureusement, la diapositive précédente est un mensonge.

Pourquoi ?

- Pour que Var $= \sigma^2/B$, il faut que les arbres soient **strictement indépendants**.
- Or, ils sont tous entraînés sur des échantillons tirés du **même jeu de données original** S.
- Pire : On a vu avec la "Règle des 63%" que deux échantillons Bootstraps ont environ **63% de données en commun** !

$$\rho = \text{Cov}(\hat{h}_i, \hat{h}_j) \neq 0$$

Nos arbres sont fortement **corrélés** (ρ).

La vraie formule de la variance du Bagging

La Variance d'estimateurs corrélés

Si les B arbres ont une corrélation ρ entre eux, la véritable variance de la moyenne est :

$$\text{Var}(\hat{h}_{bag}) = \underbrace{\rho\sigma^2}_{\text{Limite infranchissable}} + \underbrace{\frac{1-\rho}{B}\sigma^2}_{\text{Terme qui s'annule}}$$

La vraie formule de la variance du Bagging

La Variance d'estimateurs corrélés

Si les B arbres ont une corrélation ρ entre eux, la véritable variance de la moyenne est :

$$\text{Var}(\hat{h}_{bag}) = \underbrace{\rho\sigma^2}_{\text{Limite infranchissable}} + \underbrace{\frac{1-\rho}{B}\sigma^2}_{\text{Terme qui s'annule}}$$

↓ Le terme de droite :
Quand le nombre d'arbres B tend vers l'infini, cette partie tend vers zéro. (*Faire tourner 500 arbres est toujours utile*).

La vraie formule de la variance du Bagging

La Variance d'estimateurs corrélés

Si les B arbres ont une corrélation ρ entre eux, la véritable variance de la moyenne est :

$$\text{Var}(\hat{h}_{bag}) = \underbrace{\rho\sigma^2}_{\text{Limite infranchissable}} + \underbrace{\frac{1-\rho}{B}\sigma^2}_{\text{Terme qui s'annule}}$$

↓ Le terme de droite :
Quand le nombre d'arbres B tend vers l'infini, cette partie tend vers zéro. (*Faire tourner 500 arbres est toujours utile*).

🔒 Le terme de gauche :
Peu importe si vous mettez $B = 10\,000$ arbres, la variance **ne descendra jamais en dessous de** $\rho\sigma^2$.

Le Paradoxe : Quel type de modèle faut-il "Bagger" ?

Rappel Mathématique : Le Bagging **ne réduit pas le Biais** ($\mathbb{E}[\hat{h}_{bag}] = \mathbb{E}[\hat{h}]$).
Il ne sert **qu'à réduire la Variance**.

✕ Bagger un Modèle Stable

Exemples : Régression Linéaire, Arbre de prof. 1.

- Modèles à **fort biais** et **faible variance**.
- Le bootstrap ne les perturbe pas (trouvent la même frontière).
- **Conséquence :** $\rho \approx 1$.

Le Bagging est INUTILE.

Le Paradoxe : Quel type de modèle faut-il "Bagger" ?

Rappel Mathématique : Le Bagging **ne réduit pas le Biais** ($\mathbb{E}[\hat{h}_{bag}] = \mathbb{E}[\hat{h}]$).
Il ne sert **qu'à réduire la Variance**.

✗ Bagger un Modèle Stable

Exemples : Régression Linéaire, Arbre de prof. 1.

- Modèles à **fort biais** et **faible variance**.
- Le bootstrap ne les perturbe pas (trouvent la même frontière).
- **Conséquence :** $\rho \approx 1$.

Le Bagging est INUTILE.

✓ Bagger un Modèle Instable

Exemple : Arbre de Décision très profond.

- Modèles à **biais faible** mais **variance explosive**.
- Le moindre changement modifie complètement l'arbre.
- **Conséquence :** Arbres différents (ρ baisse).

Le Bagging est SURPUISSANT.

Le Paradoxe : Quel type de modèle faut-il "Bagger" ?

Rappel Mathématique : Le Bagging **ne réduit pas le Biais** ($\mathbb{E}[\hat{h}_{bag}] = \mathbb{E}[\hat{h}]$).
Il ne sert **qu'à réduire la Variance**.

✗ Bagger un Modèle Stable

Exemples : Régression Linéaire, Arbre de prof. 1.

- Modèles à **fort biais** et **faible variance**.
- Le bootstrap ne les perturbe pas (trouvent la même frontière).
- **Conséquence :** $\rho \approx 1$.

Le Bagging est INUTILE.

✓ Bagger un Modèle Instable

Exemple : Arbre de Décision très profond.

- Modèles à **biais faible** mais **variance explosive**.
- Le moindre changement modifie complètement l'arbre.
- **Conséquence :** Arbres différents (ρ baisse).

Le Bagging est SURPUISSANT.

⚠ **Attention au vocabulaire :** Pour le Bagging, on ne veut pas un "Prédicteur Faible" (qui se trompe beaucoup), on veut un prédicteur **"Instable"** (qui sur-apprend) !

Attention au vote : Bagger un "mauvais" modèle

Le Théorème du Jury (Suite) : En classification, le vote majoritaire n'est magique **que si** l'erreur individuelle e est strictement inférieure à 0.5.

Cas 1 : Le "Bon" modèle ($e < 0.5$)

Chaque arbre a un taux d'erreur $e = 0.4$ (40%).

La proportion de votes corrects suit une loi Binomiale $\mathcal{B}(B, 1-e)$.

Si on augmente le nombre d'arbres $B \to \infty$:

$$\mathbb{P}(\text{Majorité a raison}) \to \mathbf{100}\%$$

Le Bagging annule l'erreur !

Attention au vote : Bagger un "mauvais" modèle

Le Théorème du Jury (Suite) : En classification, le vote majoritaire n'est magique **que si** l'erreur individuelle e est strictement inférieure à 0.5.

✔ Cas 1 : Le "Bon" modèle ($e < 0.5$)

Chaque arbre a un taux d'erreur $e = 0.4$ (40%).

La proportion de votes corrects suit une loi Binomiale $\mathcal{B}(B, 1-e)$.

Si on augmente le nombre d'arbres $B \to \infty$:

$$\mathbb{P}(\text{Majorité a raison}) \to \mathbf{100\%}$$

Le Bagging annule l'erreur !

✘ Cas 2 : Le "Mauvais" modèle ($e > 0.5$)

Supposons la vraie classe $Y = 1$.
Notre arbre prédit :

- $Y = 1$ avec probabilité 0.4
- $Y = 0$ avec probabilité 0.6

Si on fait voter 1000 arbres, la classe 0 gagnera **toujours** (loi des grands nombres).

Erreur de l'arbre = 0.6 $\implies$ Erreur du Bagging = **1.0**

Le Bagging empire les choses !

Analyse : Le problème du "Prédicteur Fort"

🔍 **Cas pratique :** Que se passe-t-il si, dans mon dataset, une variable (ex : "Le Salaire") est extrêmement corrélée avec la cible Y par rapport aux 99 autres variables ?

Analyse : Le problème du "Prédicteur Fort"

Cas pratique : Que se passe-t-il si, dans mon dataset, une variable (ex : "Le Salaire") est extrêmement corrélée avec la cible Y par rapport aux 99 autres variables ?

Comportement de l'Algorithme CART :

- L'arbre est **Glouton** (Greedy).
- À la racine, il va chercher la meilleure coupure globale.
- Il va **toujours** choisir de couper sur "Le Salaire" en premier.

L'armée de Clones

Résultat :
Tous les arbres du Bagging se ressembleront (même sommet).

Conséquence Mathématique :
La corrélation ρ est proche de 1. La variance ne baisse pas.

Analyse : Le problème du "Prédicteur Fort"

Cas pratique : Que se passe-t-il si, dans mon dataset, une variable (ex : "Le Salaire") est extrêmement corrélée avec la cible Y par rapport aux 99 autres variables ?

Comportement de l'Algorithme CART :

- L'arbre est **Glouton** (Greedy).
- À la racine, il va chercher la meilleure coupure globale.
- Il va **toujours** choisir de couper sur "Le Salaire" en premier.

L'armée de Clones

Résultat :
Tous les arbres du Bagging se ressembleront (même sommet).

Conséquence Mathématique :
La corrélation ρ est proche de 1. La variance ne baisse pas.

Comment éviter ce phénomène ?
Comment forcer les arbres à regarder les autres variables pour devenir indépendants ?

Sommaire

Comment forcer la décorrélation des arbres ?

L'Intuition de Leo Breiman (2001)

"Si tous les arbres utilisent la même variable forte à la racine, ils seront tous pareils.
Interdisons-leur de la voir !"

✕ En Bagging classique

À chaque nœud, l'algorithme CART regarde **toutes les** p **variables** disponibles pour trouver la meilleure coupure.

→ Le prédicteur fort gagne toujours. ρ reste élevé.

→

Comment forcer la décorrélation des arbres ?

L'Intuition de Leo Breiman (2001)

"Si tous les arbres utilisent la même variable forte à la racine, ils seront tous pareils.
Interdisons-leur de la voir !"

✕ En Bagging classique

À chaque nœud, l'algorithme CART regarde **toutes les** p **variables** disponibles pour trouver la meilleure coupure.

→ Le prédicteur fort gagne toujours. ρ reste élevé.

→

La Random Forest

On ajoute une contrainte : le **Feature Bagging**.

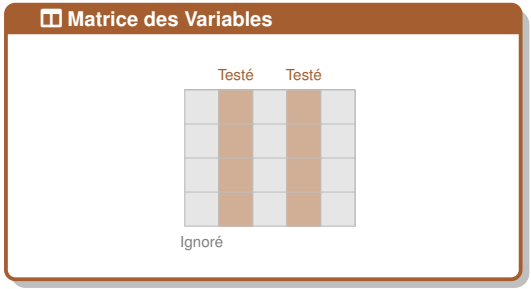
✓ L'arbre est obligé d'explorer d'autres pistes, ce qui garantit l'indépendance !

La Mécanique : Le "Feature Bagging" (Random Subspace)

⚙ **Règle d'or de la Forêt Aléatoire :** L'aléatoire n'intervient pas qu'au début (Bootstrap), il intervient à **chaque nœud de chaque arbre**.

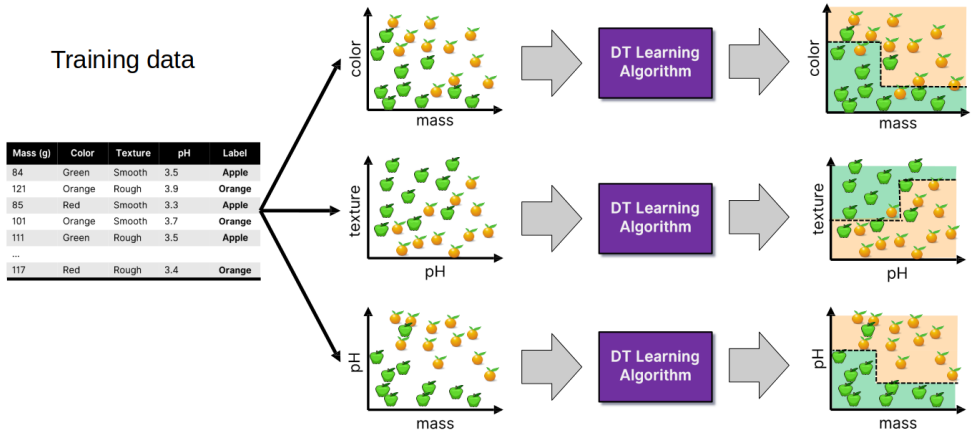
À chaque fois qu'un arbre veut se diviser :

- On ne lui donne pas accès aux p variables.
- On tire au hasard un sous-ensemble de m **variables** ($m < p$).
- L'algorithme doit trouver la meilleure coupure **uniquement** parmi ces m variables.
- ↻ On recommence ce tirage au nœud suivant !



Le Feature Bagging : Phase d'Entraînement

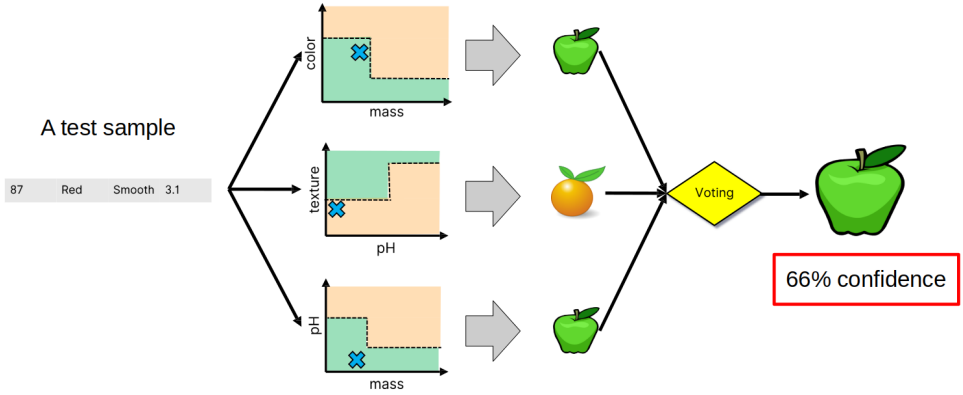
🔀 Privés de la variable évidente "Couleur", certains arbres vont devenir des experts de la "Texture" ou du "pH". Cela crée des spécialistes indépendants !



La diversité des variables force des découpages d'espace radicalement différents.

Le Feature Bagging : Phase d'Inférence

Le vote final est de bien meilleure qualité. Si un arbre s'est fait "piéger" par le bruit sur la couleur, les autres corrigeront grâce au pH et à la texture.



La corrélation ρ a chuté, la variance globale de la forêt s'effondre.

La Recette Complète : Algorithme Random Forest

</> Algorithme formel de la Forêt Aléatoire

Entrée : Un ensemble d'apprentissage $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ avec p variables.

Pour $b = 1, \ldots, B$ **faire :**

(a) Tirer un échantillon bootstrap $\mathcal{S}^{(b)}$ de taille n avec remise parmi $\mathcal{S}$.

(b) Faire croître un arbre de décision sur $\mathcal{S}^{(b)}$ en répétant récursivement les étapes suivantes pour chaque nœud, jusqu'à atteindre la taille minimale de nœud n_{min} :

i. Sélectionner aléatoirement m **variables** parmi les p.

ii. Trouver la meilleure variable et le meilleur seuil de coupure (split) **uniquement parmi ces** m **variables**.

iii. Diviser le nœud en deux nœuds enfants.

(c) Estimer l'erreur de généralisation en utilisant les échantillons *out-of-bag* (OOB).

Fin Pour

Inférence : Pour une nouvelle observation $\mathbf{x}$, la prédiction $\hat{y}$ est obtenue par :

- Vote majoritaire des B arbres (Classification).
- Moyenne des prédictions des B arbres (Régression).

Réf : Leo Breiman (2001), "Random Forests", Machine Learning, vol. 45, p. 5-32.

Le paramètre clé : Combien de variables *m* tirer ?

Le paramètre *m* (souvent appelé `max_features` dans Scikit-Learn) contrôle la quantité d'aléatoire injectée et **pilote le compromis de corrélation** entre les arbres.

Les Standards

Recommandations de Breiman :

En Classification

$$m = \lfloor \sqrt{p} \rfloor$$

En Régression

$$m = \lfloor \frac{p}{3} \rfloor$$

Pourquoi ces valeurs ? (Les Extrêmes)

Le paramètre clé : Combien de variables m tirer ?

Le paramètre m (souvent appelé `max_features` dans Scikit-Learn) contrôle la quantité d'aléatoire injectée et **pilote le compromis de corrélation** entre les arbres.

Les Standards

Recommandations de Breiman :

En Classification

$$m = \lfloor \sqrt{p} \rfloor$$

En Régression

$$m = \lfloor \frac{p}{3} \rfloor$$

Pourquoi ces valeurs ? (Les Extrêmes)

Si m est très grand ($m \to p$)

$\approx$ On retombe sur du Bagging classique.

✗ Les arbres redeviennent très corrélés ($\rho \nearrow$). La variance de la forêt ne baisse plus.

Le paramètre clé : Combien de variables m tirer ?

Le paramètre m (souvent appelé `max_features` dans Scikit-Learn) contrôle la quantité d'aléatoire injectée et **pilote le compromis de corrélation** entre les arbres.

Les Standards

Recommandations de Breiman :

En Classification

$$m = \lfloor \sqrt{p} \rfloor$$

En Régression

$$m = \lfloor \frac{p}{3} \rfloor$$

Pourquoi ces valeurs ? (Les Extrêmes)

Si m est très grand ($m \to p$)

$\approx$ On retombe sur du Bagging classique.

✗ Les arbres redeviennent très corrélés ($\rho \nearrow$). La variance de la forêt ne baisse plus.

Si m est très petit ($m \to 1$)

$\approx$ Arbres "aveugles" et totalement aléatoires.

✗ Arbres décorrélés ($\rho \searrow$), mais individuellement trop faibles (Le Biais global explose).

Le paramètre clé : Combien de variables m tirer ?

Le paramètre m (souvent appelé `max_features` dans Scikit-Learn) contrôle la quantité d'aléatoire injectée et **pilote le compromis de corrélation** entre les arbres.

Les Standards

Recommandations de Breiman :

En Classification

$$m = \lfloor \sqrt{p} \rfloor$$

En Régression

$$m = \lfloor \frac{p}{3} \rfloor$$

Pourquoi ces valeurs ? (Les Extrêmes)

Si m est très grand ($m \to p$)

$\approx$ On retombe sur du Bagging classique.

✗ Les arbres redeviennent très corrélés ($\rho \nearrow$). La variance de la forêt ne baisse plus.

Si m est très petit ($m \to 1$)

$\approx$ Arbres "aveugles" et totalement aléatoires.

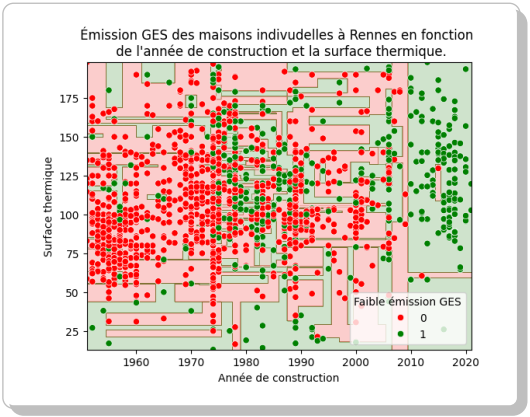
✗ Arbres décorrélés ($\rho \searrow$), mais individuellement trop faibles (Le Biais global explose).

Bilan : Choisir m, c'est piloter le **Compromis Biais-Variance** de la Forêt !

L’impact visuel : De l’escalier à la courbe

La magie de la moyenne : des milliers de frontières "carrées" créent une surface de décision lisse.

1 seul Arbre Profond



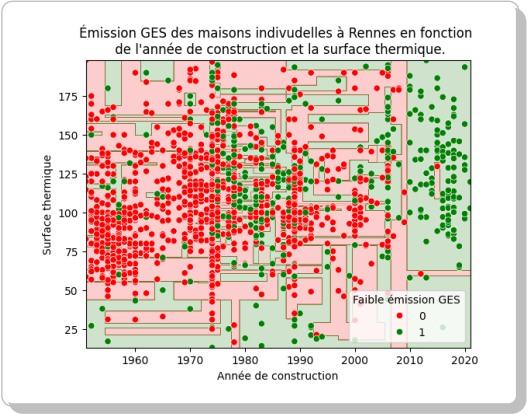
Des frontières strictes, orthogonales et bruitées (sur-apprentissage).

VS

L'impact visuel : De l'escalier à la courbe

La magie de la moyenne : des milliers de frontières "carrées" créent une surface de décision lisse.

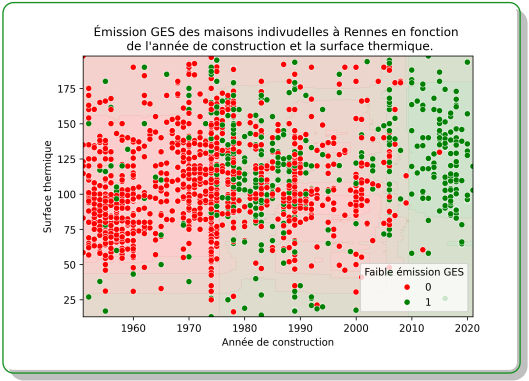
1 seul Arbre Profond



Des frontières strictes, orthogonales et bruitées (sur-apprentissage).

VS

Forêt Aléatoire (500 arbres)



Des zones de probabilités lisses issues du vote majoritaire (l'agrégation).

Compromis biais-variance lié au nombre d'arbres B ?

La Question

"Si j'augmente énormément le nombre d'arbres ($B = 1000, B = 5000 \ldots$), ma forêt ne va-t-elle pas finir par sur-apprendre le dataset ?"

Compromis biais-variance lié au nombre d'arbres B ?

La Question

"Si j'augmente énormément le nombre d'arbres ($B = 1000, B = 5000 \ldots$), ma forêt ne va-t-elle pas finir par sur-apprendre le dataset ?"

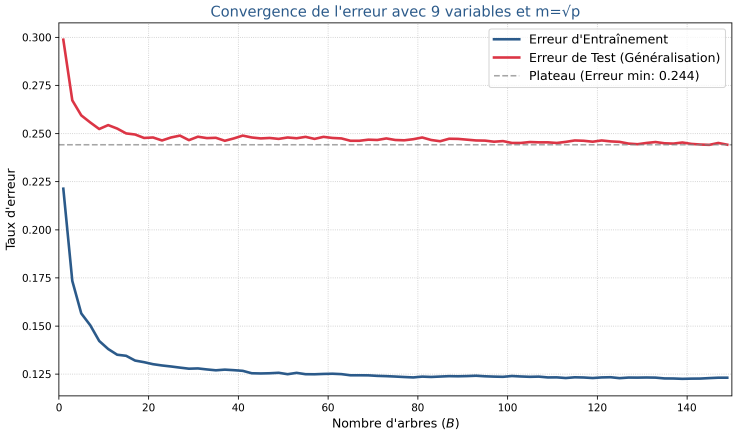
NON !

L'ajout d'arbres n'augmente **pas** la complexité mathématique du modèle. Grâce à la **Loi des Grands Nombres**, la variance de la Forêt va s'écraser asymptotiquement vers sa limite indépassable ($\rho\sigma^2$) puis stagner.

✔ **Plus on a d'arbres, plus la prédiction est stable.**

Preuve Visuelle : Convergence de l'erreur

Application DPE : Évolution du Taux d'Erreur de classification en fonction du nombre d'arbres (`n_estimators`).



Observations :

- L'erreur de Test chute drastiquement au début.
- Vers $B = 50$, la courbe rouge se **stabilise** et forme un plateau.
- Elle ne remonte **jamais** (pas de courbe en "U").
- Ajouter 500 arbres ne dégrade pas le score, mais gaspille de la puissance de calcul !

Mais alors, comment une Forêt peut-elle sur-apprendre ?

Le Piège : L'arbre infini

Par défaut (`max_depth=None`), les arbres poussent sans limite :

- Ils mémorisent parfaitement le **bruit**.
- La variance individuelle σ^2 devient si énorme que l'agrégation ($\frac{1}{B}$) ne suffit plus à la compenser.

Mais alors, comment une Forêt peut-elle sur-apprendre ?

Le Piège : L'arbre infini

Par défaut (`max_depth=None`), les arbres poussent sans limite :

- Ils mémorisent parfaitement le **bruit**.
- La variance individuelle σ^2 devient si énorme que l'agrégation ($\frac{1}{B}$) ne suffit plus à la compenser.

Tolérance naturelle :
Néanmoins, grâce à la moyenne, une Forêt supportera toujours des arbres **bien plus profonds** et instables qu'un arbre de décision simple !

Mais alors, comment une Forêt peut-elle sur-apprendre ?

Le Piège : L'arbre infini

Par défaut (`max_depth=None`), les arbres poussent sans limite :

- Ils mémorisent parfaitement le **bruit**.
- La variance individuelle σ^2 devient si énorme que l'agrégation ($\frac{1}{B}$) ne suffit plus à la compenser.

La Pratique : Le Pré-élagage

On bride légèrement la croissance pour lisser les frontières :

- **`max_depth`** : Limite la hauteur.
- **`min_samples_split`** : Nb min diviser nœud.
- **`min_samples_leaf`** : Nb min valider feuille.

Tolérance naturelle :
Néanmoins, grâce à la moyenne, une Forêt supportera toujours des arbres **bien plus profonds** et instables qu'un arbre de décision simple !

Mais alors, comment une Forêt peut-elle sur-apprendre ?

Le Piège : L'arbre infini

Par défaut (`max_depth=None`), les arbres poussent sans limite :

- Ils mémorisent parfaitement le **bruit**.
- La variance individuelle σ^2 devient si énorme que l'agrégation ($\frac{1}{B}$) ne suffit plus à la compenser.

Tolérance naturelle :
Néanmoins, grâce à la moyenne, une Forêt supportera toujours des arbres **bien plus profonds** et instables qu'un arbre de décision simple !

La Pratique : Le Pré-élagage

On bride légèrement la croissance pour lisser les frontières :

- **`max_depth`** : Limite la hauteur.
- **`min_samples_split`** : Nb min diviser nœud.
- **`min_samples_leaf`** : Nb min valider feuille.

Et l'élagage a posteriori (α) ?

Le *Cost-Complexity Pruning* (`ccp_alpha`) est très peu utilisé ici :

- On **veut** des arbres instables. L'élagage tue la diversité du Bagging.
- Post-élaguer 500 arbres géants est un gouffre absolu en temps de calcul !

Sommaire

Le défi informatique : Évaluer une Forêt

⏳ La **Validation Croisée** (K-Fold) est la méthode reine pour évaluer un modèle. Mais avec des Forêts Aléatoires sur de grands jeux de données, le temps de calcul explose.

Le calcul (Exemple 5-Fold)

- Une Forêt contient $B = 500$ arbres.
- La CV 5-Fold l'entraîne 5 fois.
- **Total : 2 500 arbres à entraîner !**

L'Échelle Industrielle

Sur notre dataset de la Bretagne (> 400 000 maisons et 9 variables), une simple CV peut prendre plusieurs dizaines de minutes.

⚠ **Et tester 10 grilles d'hyperparamètres prendra des heures !**

Le défi informatique : Évaluer une Forêt

⏳ La **Validation Croisée** (K-Fold) est la méthode reine pour évaluer un modèle. Mais avec des Forêts Aléatoires sur de grands jeux de données, le temps de calcul explose.

Le calcul (Exemple 5-Fold)

- Une Forêt contient $B = 500$ arbres.
- La CV 5-Fold l'entraîne 5 fois.
- **Total : 2 500 arbres à entraîner !**

L'Échelle Industrielle

Sur notre dataset de la Bretagne (> 400 000 maisons et 9 variables), une simple CV peut prendre plusieurs dizaines de minutes.

⚠ **Et tester 10 grilles d'hyperparamètres prendra des heures !**

❓ D'après vous : comment pourrait-on évaluer notre modèle "gratuitement", sans avoir à entraîner le moindre arbre supplémentaire ?

L'Astuce : Retour sur les mathématiques du Bootstrap

Probabilité d'être ignoré par un arbre

Lorsqu'on tire l'échantillon Bootstrap $\mathcal{S}^{(b)}$ de taille n (avec remise), quelle est la probabilité qu'une observation spécifique $\mathbf{x}_i$ ne soit **jamais** tirée ?

$$\mathbb{P}(\mathbf{x}_i \notin \mathcal{S}^{(b)}) = \left(1 - \frac{1}{n}\right)^n$$

Quand n est grand, cette probabilité converge vers une limite célèbre :

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \mathbf{e^{-1}} \approx \mathbf{0.368}$$

L'Astuce : Retour sur les mathématiques du Bootstrap

Probabilité d'être ignoré par un arbre

Lorsqu'on tire l'échantillon Bootstrap $\mathcal{S}^{(b)}$ de taille n (avec remise), quelle est la probabilité qu'une observation spécifique $\mathbf{x}_i$ ne soit **jamais** tirée ?

$$\mathbb{P}(\mathbf{x}_i \notin \mathcal{S}^{(b)}) = \left(1 - \frac{1}{n}\right)^n$$

Quand n est grand, cette probabilité converge vers une limite célèbre :

$$\lim_{n\to\infty} \left(1 - \frac{1}{n}\right)^n = \mathbf{e^{-1}} \approx \mathbf{0.368}$$

! Bilan : Environ **36.8%** des données originales sont exclues de l'entraînement de **chaque arbre**. On les appelle les données **Out-Of-Bag (OOB)**.

La Mécanique de l'Out-Of-Bag : Un "Test Set" par arbre

♻ **L'intuition de Leo Breiman :** Les 36.8% de données ignorées par le Bootstrap de l'arbre *b* sont **strictement inconnues** de cet arbre. Elles forment un Validation Set naturel et gratuit !

La Matrice de Vote OOB

Comment évaluer un point x_i ? Seuls les arbres "ignorants" ont le droit de voter.

Point	Arbre 1	Arbre 2	Arbre 3	La prédiction OOB ($\hat{y}_{i,OOB}$) est issue de...
$\mathbf{x}_1$	*In-Bag*	**OOB**	*In-Bag*	→ L'Arbre 2 (Seul à ne pas l'avoir vu)
$\mathbf{x}_2$	**OOB**	*In-Bag*	**OOB**	→ Vote majoritaire : Arbres 1 et 3
$\mathbf{x}_3$	**OOB**	**OOB**	*In-Bag*	→ Vote majoritaire : Arbres 1 et 2

ℹ *En moyenne, chaque point du dataset sera évalué par ~ 36.8% des B arbres de la forêt.*

L'OOB Score : La Note Finale de la Forêt

Formule de l'Erreur de Généralisation

En répétant ce processus matriciel, on obtient une prédiction "vierge" $\hat{y}_{i,OOB}$ pour **chaque point** $i \in \{1, \ldots, n\}$ de notre dataset.

Il suffit alors de moyenner les erreurs :

$$\text{Erreur}_{OOB} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{\hat{y}_{i,OOB} \neq y_i\}}$$

L'OOB Score : La Note Finale de la Forêt

Formule de l'Erreur de Généralisation

En répétant ce processus matriciel, on obtient une prédiction "vierge" $\hat{y}_{i,OOB}$ pour **chaque point** $i \in \{1, \ldots, n\}$ de notre dataset.

Il suffit alors de moyenner les erreurs :

$$\text{Erreur}_{OOB} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{\hat{y}_{i,OOB} \neq y_i\}}$$

La Magie de l'OOB :
Cette erreur est calculée **en même temps** que l'entraînement !

À la fin de la fonction `fit()`, le modèle connaît déjà sa capacité à généraliser (son OOB Score), **sans avoir entraîné un seul arbre supplémentaire**.

L'OOB Score : La Note Finale de la Forêt

Formule de l'Erreur de Généralisation

En répétant ce processus matriciel, on obtient une prédiction "vierge" $\hat{y}_{i,OOB}$ pour **chaque point** $i \in \{1, \ldots, n\}$ de notre dataset.

Il suffit alors de moyenner les erreurs :

$$\text{Erreur}_{OOB} = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{\{\hat{y}_{i,OOB} \neq y_i\}}$$

La Magie de l'OOB :
Cette erreur est calculée **en même temps** que l'entraînement !

À la fin de la fonction `fit()`, le modèle connaît déjà sa capacité à généraliser (son OOB Score), **sans avoir entraîné un seul arbre supplémentaire**.

Conséquence : On peut régler nos hyperparamètres sur des millions de lignes de manière très rapide !

Match au sommet : OOB Score vs Validation Croisée

⚖️ L'OOB est une astuce algorithmique brillante, mais **la Validation Croisée (*K*-Fold) reste la méthode la plus rigoureuse** en Machine Learning.

⏱ L'OOB Score

Avantages :

- ✔ Totalement **gratuit** (calculé au vol).
- ✔ Parfait pour monitorer l'apprentissage.

Limites :

- ✘ Exclusif aux modèles basés sur le Bootstrap (Bagging, RF).
- ✘ Impossible pour comparer avec un autre algorithme (ex : SVM).

La Validation Croisée

Avantages :

- ✔ Le **Gold Standard**. Rigoureux et universel.
- ✔ Permet d'évaluer la variance du score (stabilité) via les *folds*.

Limites :

- ✘ Extrêmement **coûteux** en temps de calcul sur les grands jeux de données.

Le Workflow Pratique & Le problème de l'Interprétabilité

1. Phase de Prototypage ⚙

Utilisez l'OOB (`oob_score=True`) pour explorer rapidement vos hyperparamètres (*m*, *max_depth*) sur le jeu complet sans faire exploser les temps de calcul.

↓

2. Phase de Validation Finale 🏆

Figez le modèle, et lancez une véritable **Validation Croisée** pour comparer sa performance officielle face aux autres algorithmes du marché.

Le Workflow Pratique & Le problème de l'Interprétabilité

1. Phase de Prototypage ⚙

Utilisez l'OOB (`oob_score=True`) pour explorer rapidement vos hyperparamètres (*m*, *max_depth*) sur le jeu complet sans faire exploser les temps de calcul.

↓

2. Phase de Validation Finale 🏆

Figez le modèle, et lancez une véritable **Validation Croisée** pour comparer sa performance officielle face aux autres algorithmes du marché.

Le revers de la médaille : La Boîte Noire

Un Arbre de Décision unique était une "Boîte Blanche" (des règles 'SI... ALORS' lisibles par un humain).
Mais comment interpréter un vote issu de 500 arbres décorrélés ? L'algorithme est devenu opaque.

→ *Comment savoir quelles variables ont réellement compté dans la décision ?*

Sommaire

Le Prix de la Performance : La Boîte Noire

Le Paradoxe : Nous avons drastiquement réduit la variance et amélioré la précision. Mais en moyennant 500 arbres aléatoires, **nous avons perdu la lisibilité des règles de décision**.

L'Arbre Unique (Boîte Blanche)

- Les règles sont explicites.
- `SI (Année > 1974) ET (Murs < 50m²)` → `Classe C`.
- Parfait pour expliquer la décision à un client ou au métier.

→

La Forêt (Boîte Noire)

- C'est un vote de 500 modèles discordants.
- Impossible d'imprimer ou de lire les règles !
- **Quelles variables ont vraiment compté ?**

À vous de jouer : Comment mesurer l'importance ?

❓ Si vous deviez concevoir cet algorithme...

"Comment feriez-vous pour demander à une Forêt Aléatoire de vous dire quelles variables (ex : Année, Surface) ont été les plus cruciales dans ses décisions ?"

À vous de jouer : Comment mesurer l'importance ?

Si vous deviez concevoir cet algorithme...

"Comment feriez-vous pour demander à une Forêt Aléatoire de vous dire quelles variables (ex : Année, Surface) ont été les plus cruciales dans ses décisions ?"

Indices pour la salle :

- Piste 1 : Regarder à l'intérieur de la mécanique des arbres (Gini).
- Piste 2 : Cacher une variable et voir si le modèle s'effondre.

Méthode 1 : L'approche historique (MDI)

Mean Decrease in Impurity (MDI) : On va comptabiliser l'effort fourni par chaque variable pour "nettoyer" le dataset lors de la construction des arbres.

Le Principe Algorithmique

À chaque fois qu'un nœud utilise la variable X_j pour se diviser, l'impureté de Gini (ou la Variance en régression) diminue.

On somme toutes ces diminutions d'impureté (ΔGini) attribuables à X_j sur **tous les nœuds** de **tous les arbres** :

$$\text{Importance}(X_j) = \frac{1}{B} \sum_{b=1}^{B} \sum_{t \in \text{nœuds}(X_j)} \Delta\text{Gini}(t)$$

Méthode 1 : L'approche historique (MDI)

Mean Decrease in Impurity (MDI) : On va comptabiliser l'effort fourni par chaque variable pour "nettoyer" le dataset lors de la construction des arbres.

Le Principe Algorithmique

À chaque fois qu'un nœud utilise la variable X_j pour se diviser, l'impureté de Gini (ou la Variance en régression) diminue.

On somme toutes ces diminutions d'impureté (ΔGini) attribuables à X_j sur **tous les nœuds** de **tous les arbres** :

$$\text{Importance}(X_j) = \frac{1}{B} \sum_{b=1}^{B} \sum_{t \in \text{nœuds}(X_j)} \Delta\text{Gini}(t)$$

En pratique dans Scikit-Learn :

C'est l'approche utilisée par défaut. Après un `rf.fit(X, y)`, ces valeurs (qui sont automatiquement normalisées pour que leur somme vaille 100%) sont disponibles dans l'attribut : `rf.feature_importances_`

Le Piège du MDI : Un peu de réflexion

💣 Le Biais de Cardinalité

Imaginez deux variables dans votre dataset :

- **Variable A :** `Présence_Balcon` (2 modalités : Oui/Non)
- **Variable B :** `Code_Postal` (Des milliers de modalités)

D'après vous, laquelle sera artificiellement favorisée par le calcul du MDI (Gini) et pourquoi ?

Le Piège du MDI : Un peu de réflexion

Le Biais de Cardinalité

Imaginez deux variables dans votre dataset :

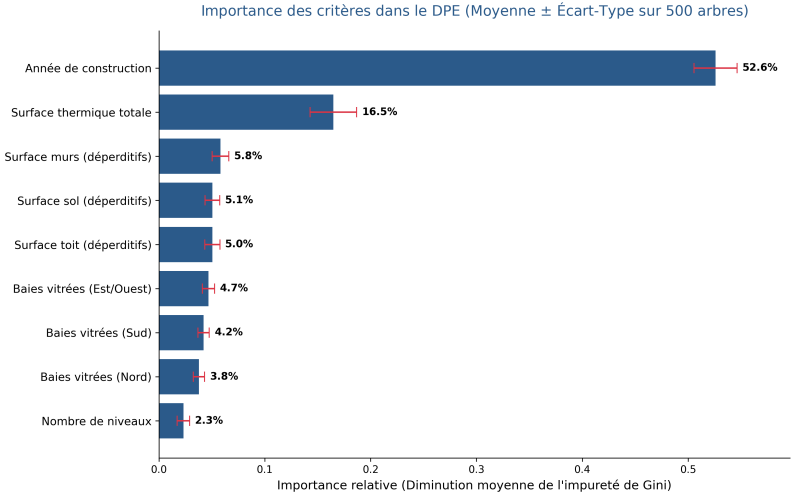
- **Variable A :** `Présence_Balcon` (2 modalités : Oui/Non)
- **Variable B :** `Code_Postal` (Des milliers de modalités)

D'après vous, laquelle sera artificiellement favorisée par le calcul du MDI (Gini) et pourquoi ?

⚠ **Réponse :** La variable B ! Avec des milliers de modalités, l'algorithme a une infinité de façons de couper. Il va la surexploiter par opportunisme mathématique, même si elle n'a aucun sens physique.

Résultat MDI : Ce que pense notre Forêt du DPE

Approche MDI : Calcul basé sur la réduction de l'impureté de Gini. Les barres d'erreur montrent la variance entre les 500 arbres.



Méthode 2 : L'approche scientifique (Permutation)

Permutation Importance : Au lieu de regarder *comment* l'arbre est construit, on regarde **comment il réagit si on lui sabote volontairement une information**.

L'algorithme de "Shuffle"

1. On calcule l'erreur de référence sur un *Test Set* propre.
2. On choisit une colonne (ex : Surface) et on **mélange ses valeurs** aléatoirement. Les autres restent intactes.
3. On refait prédire le modèle.
4. On mesure la **chute de performance**.

Illustration du Sabotage

Dataset original

Année	**Surface**	GES
1980	**90**	C
2015	**50**	A
1970	**120**	E

↓ *Permutation de la Surface* ↓

Dataset "Bruité"

Année	**Surface**	GES
1980	**120**	?
2015	**90**	?
1970	**50**	?

L'interprétation : Détecter les variables clés

Le Score d'Importance

$$\text{Importance}(X_j) = \text{Erreur}_{\text{Après Mélange}} - \text{Erreur}_{\text{Initiale}}$$

↑ Cas 1 : L'erreur explose

Que signifie ce résultat ?

↔ Cas 2 : L'erreur stagne

Que signifie ce résultat ?

L'interprétation : Détecter les variables clés

Le Score d'Importance

$$\text{Importance}(X_j) = \text{Erreur}_{\text{Après Mélange}} - \text{Erreur}_{\text{Initiale}}$$

↑ Cas 1 : L'erreur explose

Que signifie ce résultat ?

✔ La variable était cruciale. Sans elle, le modèle s'effondre.

↔ Cas 2 : L'erreur stagne

Que signifie ce résultat ?

L'interprétation : Détecter les variables clés

Le Score d'Importance

$$\text{Importance}(X_j) = \text{Erreur}_{\text{Après Mélange}} - \text{Erreur}_{\text{Initiale}}$$

↑ Cas 1 : L'erreur explose

Que signifie ce résultat ?

✔ La variable était cruciale. Sans elle, le modèle s'effondre.

↔ Cas 2 : L'erreur stagne

Que signifie ce résultat ?

✖ La variable ne servait à rien. La remplacer par du bruit ne change rien.

L'interprétation : Détecter les variables clés

Le Score d'Importance

$$\text{Importance}(X_j) = \text{Erreur}_{\text{Après Mélange}} - \text{Erreur}_{\text{Initiale}}$$

↑ Cas 1 : L'erreur explose

Que signifie ce résultat ?

✔ La variable était cruciale. Sans elle, le modèle s'effondre.

↔ Cas 2 : L'erreur stagne

Que signifie ce résultat ?

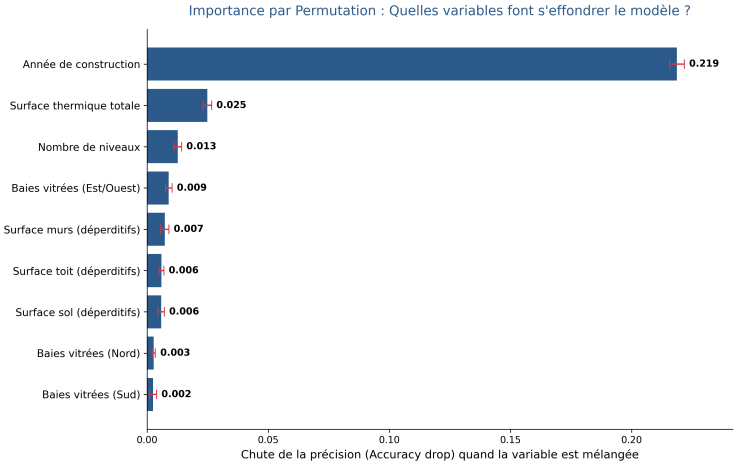
✖ La variable ne servait à rien. La remplacer par du bruit ne change rien.

</> En pratique dans Scikit-Learn :

Cette méthode s'applique sur le jeu de **Test** (pour évaluer la vraie capacité de généralisation), via une fonction dédiée : `result = permutation_importance(rf, X_test, y_test, n_repeats=10)`

Résultat Permutation : Le vrai juge de paix

Approche Permutation : Chute de la précision lors du sabotage d'une variable sur le jeu de test. Cette méthode révèle l'utilité réelle pour généraliser.



Décryptage Métier :

- **Écrasante domination :** L'*Année de construction* fait tout le travail. La saboter fait chuter la précision de 21.9%.
- **Le reste s'effondre :** Les surfaces déperditives et l'orientation des baies vitrées ont un impact quasi nul (< 1%).
- **Conclusion :**
  Inutile de mesurer les fenêtres, le modèle trouve le DPE presque uniquement grâce à l'âge du bâtiment !

Le Bilan des Importances

⚖ Que retenir pour vos projets Data professionnels ?

L'attribut MDI (Gini)

- ✓ **Immédiat :** Calculé gratuitement pendant l'entraînement (`feature_importances_`).
- ✗ **Biaisé :** Favorise les variables avec beaucoup de modalités.
- › **Usage :** Exploration initiale rapide.

La Permutation Importance

- ✓ **Scientifique :** Corrige le biais de cardinalité et s'applique à tout type de modèle ML.
- ✗ **Coûteux :** Nécessite des dizaines de prédictions supplémentaires.
- › **Usage :** Reporting final métier.

Le Bilan des Importances

⚖ Que retenir pour vos projets Data professionnels ?

L'attribut MDI (Gini)

- ✓ **Immédiat :** Calculé gratuitement pendant l'entraînement (`feature_importances_`).
- ✗ **Biaisé :** Favorise les variables avec beaucoup de modalités.
- › **Usage :** Exploration initiale rapide.

La Permutation Importance

- ✓ **Scientifique :** Corrige le biais de cardinalité et s'applique à tout type de modèle ML.
- ✗ **Coûteux :** Nécessite des dizaines de prédictions supplémentaires.
- › **Usage :** Reporting final métier.

🎓 Le secret d'expert :

Ne calculez **jamais** la Permutation sur le jeu d'entraînement (cela mesure la mémorisation). Calculez-la sur le jeu de **Test** ou sur les données **OOB** pour mesurer la vraie capacité de généralisation !

Sommaire

La Boîte à Outils : Que faut-il optimiser ?

🔧 **Le Workflow du Data Scientist :** Face à une nouvelle Forêt Aléatoire, voici l'ordre de priorité pour régler vos hyperparamètres (le *Tuning*).

1. **`n_estimators` (Le nombre d'arbres B)** :
   Règle : Mettez-en le plus possible en fonction de votre temps/RAM (ex : 100 à 500). Plus il y en a, plus la variance baisse. **Ce paramètre ne fait jamais sur-apprendre.**

2. **`max_features` (La taille du sous-espace m)** :
   Règle : C'est le cœur de la décorrélation ! Testez `sqrt` (racine carrée, le défaut en classification), $\log_2$, ou `None` (Bagging classique).

3. **`min_samples_leaf` (Le pré-élagage)** :
   Règle : Pour lisser les frontières de décision et calmer un modèle qui sur-apprend. Testez des valeurs comme 1 (défaut), 5, 10 ou 20.

Le Point d'Attention : L'Extrapolation en Régression

La faille mathématique des Arbres

Une Forêt Aléatoire ne sait pas extrapoler !

Le Problème :
Une prédiction de forêt est une **moyenne** des observations des feuilles (les données d'entraînement).

→ **Conséquence :**
Le modèle ne pourra **jamais** prédire une valeur plus grande que le maximum absolu vu à l'entraînement, ni plus petite que le minimum.

Exemple Métier :
Si vous prédisez le Chiffre d'Affaires d'un magasin, et qu'il n'a jamais dépassé 10 000€ dans votre jeu de données...

Même avec les meilleures conditions météo et promotions du monde, **la forêt ne prédira jamais** 11 000**€**. Une régression linéaire, si !

Bilan : Quand utiliser une Random Forest ?

Les Forces (Pourquoi on l'adore)

- ✔ **Aucune normalisation** requise (les arbres s'en fichent des échelles).
- ✔ **Robuste** aux valeurs aberrantes (Outliers) géométriques.
- ✔ Algorithme **hautement parallélisable** sur plusieurs cœurs CPU (le paramètre `n_jobs=-1`).
- ✔ Fonctionne très bien "out-of-the-box" (avec les paramètres par défaut).

Les Faiblesses (Les limites)

- ✘ Modèle **très lourd en mémoire RAM** (il faut stocker 500 arbres complets !).
- ✘ L'inférence (prédiction en production) peut être **lente** face à une régression logistique.
- ✘ C'est une **Boîte Noire** (nécessite l'OOB et la Permutation Importance).

Le mot de la fin : La simplicité Scikit-Learn

</> Beaucoup de théorie... pour 3 lignes de code !

```
from sklearn.ensemble import RandomForestClassifier

# 1. Instanciation
rf = RandomForestClassifier(n_estimators=500, max_depth=15,
min_samples_leaf=5)

# 2. L'entraînement (La magie opère ici)
rf.fit(X_train, y_train)

# 3. La prédiction en production
y_pred = rf.predict(X_test)
```

</> Voir le Code Complet du Cours

Synthèse du Module : L'Évolution des Algorithmes

Prenez du recul : Vous maîtrisez désormais la logique d'évolution des algorithmes de Machine Learning classiques.

Les fondations (La Recette)

Pour tout algorithme, il faut : des données (X, y), une famille de modèles, et une **fonction de coût** à minimiser.

L'Arbre de Décision

Découpage de l'espace par des règles. **Boîte blanche** très interprétable, mais mathématiquement instable.

K-Nearest Neighbors

"Dis-moi qui sont tes voisins..." Intuitif et local, mais souffre de la malédiction de la dimension.

La Forêt Aléatoire

On détruit l'instabilité par la sagesse des foules (Bagging). **Boîte noire** redoutable et robuste.

Le Fil Rouge : Le Compromis Biais-Variance

La bataille universelle du Machine Learning

Chaque algorithme vu cette année possède son propre curseur pour naviguer entre le **Sous-apprentissage** (Biais) et le **Sur-apprentissage** (Variance).

Le Danger : La Variance

Le modèle mémorise le bruit.

- **KNN :** Quand k est trop petit (ex : $k = 1$).
- **Arbre :** Quand on le laisse pousser à l'infini (`max_depth=None`).
- **Conséquence :** Frontières de décision hachées, mauvaise généralisation.

Le Remède : La Régularisation

On bride la complexité ou on moyenne.

- **KNN :** On augmente le nombre de voisins k.
- **Arbre :** On pré-élague (`min_samples_leaf`).
- **Forêt :** On utilise le **Bagging** (moyenne de B arbres décorrélés).

Le Mot de la Fin : La vraie valeur du Data Scientist

⚙️ **Prenez conscience d'une chose :** Aujourd'hui, entraîner un algorithme tient en 3 lignes de code Python. Votre valeur ajoutée est ailleurs.

🖩 Paramètres internes

Ils sont appris **automatiquement** par l'algorithme pendant le `fit()`.

- Le seuil s et la variable j d'une coupe.
- Les poids θ d'une régression.

🔧 Hyperparamètres externes

Ils doivent être fixés **par vous**, avant même de lancer l'entraînement.

- Le nombre d'arbres B (`n_estimators`).
- La taille du sous-espace m (`max_features`).

Le Mot de la Fin : La vraie valeur du Data Scientist

⚙ **Prenez conscience d'une chose :** Aujourd'hui, entraîner un algorithme tient en 3 lignes de code Python. Votre valeur ajoutée est ailleurs.

Paramètres internes

Ils sont appris **automatiquement** par l'algorithme pendant le `fit()`.

- Le seuil s et la variable j d'une coupe.
- Les poids θ d'une régression.

Hyperparamètres externes

Ils doivent être fixés **par vous**, avant même de lancer l'entraînement.

- Le nombre d'arbres B (`n_estimators`).
- La taille du sous-espace m (`max_features`).

La Règle d'Or en Entreprise :
Ne choisissez jamais vos hyperparamètres au hasard. La rigueur scientifique du Data Scientist réside dans l'utilisation stricte de la **Validation Croisée** ou de l'**OOB Score** pour construire sa pipeline !

ANNEXES & BONUS

Feature Engineering :

Préparer le terrain pour l'IA

Le Problème : L'ordinateur ne parle que "Maths"

Le mur de la réalité : Dans notre dataset breton, tout va bien avec la "Surface" ou "l'Année". Mais comment gérer une colonne "Type de Chauffage" (Gaz, Électrique, Bois) ?

A La barrière du texte

Les algorithmes de Machine Learning (arbres, distances, régressions) ne sont que de gigantesques équations mathématiques.

- ✗ Ils ne peuvent pas additionner du "Gaz" et du "Bois".
- ✗ L'erreur classique en Python : `ValueError: could not convert string to float`.

→ **Il faut encoder (traduire) ces mots en nombres.**

La Mauvaise Idée : Le Label Encoding

⚠ L'idée naïve : Remplacer chaque mot par un chiffre

On pourrait se dire : *"Remplaçons Pomme par 1, Banane par 2, et Orange par 3"*. C'est ce qu'on appelle le **Label Encoding**.

💣 Le Piège Mathématique

En faisant cela, vous venez de dire secrètement à votre algorithme que :

$$1 < 2 < 3$$

donc

Pomme $<$ **Banane** $<$ **Orange**

Pire encore : $1 + 2 = 3 \implies$ **Pomme + Banane = Orange** !

➔ Vous avez créé **un ordre et une distance artificiels** qui vont complètement fausser les prédictions (surtout pour les distances géométriques !).

La Solution : Le One-Hot Encoding

Le One-Hot Encoding (Variables Muettes)

Pour éviter de créer un faux ordre, on va éclater notre colonne texte en plusieurs colonnes binaires (0 ou 1).

- Chaque modalité (chaque fruit) devient **sa propre colonne**.
- On met un 1 si la ligne possède cette caractéristique, sinon on met un 0.
- En Python (Pandas) : `pd.get_dummies(df)` ou `OneHotEncoder` (Scikit-Learn).

En Pratique : L'éclatement des colonnes

Avant

Fruit
Pomme
Orange
Banane
Pomme

→

Après (One-Hot)

est_Pomme	est_Orange	est_Banane
1	0	0
0	**1**	0
0	0	**1**
1	0	0

Remarque : Toutes les distances mathématiques entre ces catégories sont désormais égales !

Standardisation : Indispensable ou Inutile ?

↔ **Mise à l'échelle (StandardScaler) :** Faut-il écraser toutes nos variables pour qu'elles aient une moyenne de 0 et une variance de 1 ? **Ça dépend du modèle !**

K-NN (Distances)

VITAL.

- Le KNN calcule des distances euclidiennes.
- Si le Salaire varie de 10 000 et l'Âge varie de 50, le Salaire va complètement **écraser** l'Âge dans le calcul.
- Il **faut** tout mettre à la même échelle.

Forêts Aléatoires

TOTALEMENT INUTILE.

- Un arbre ne calcule aucune distance, il cherche juste des **points de coupure** (seuils).
- Couper "Âge < 40" ou "Âge normé < 0.1" donne **exactement la même séparation** des données.

Références Bibliographiques I

T. Hastie, R. Tibshirani, J. Friedman (2009).
The Elements of Statistical Learning : Data Mining, Inference, and Prediction.
Springer Series in Statistics. 2nd Edition.

S. Shalev-Shwartz, S. Ben-David (2014).
Understanding Machine Learning : From Theory to Algorithms.
Cambridge University Press.

Scikit-Learn Developers.
User Guide : Machine Learning in Python.
Documentation officielle : `https://scikit-learn.org`

L. Breiman, J. Friedman, R. Olshen, C. Stone (1984).
Classification and Regression Trees.
Wadsworth International Group.