

Introduction aux méthodes ensemblistes*

On va vous booster le gradient

Mélina Hillion

Unité SSP-Lab

Insee

melina.hillion@insee.fr

Olivier Meslin

Unité SSP-Lab

Insee

olivier.meslin@insee.fr

November 27, 2024

Abstract

A compléter

Keywords: machine learning • méthodes ensemblistes • formation

*Nous remercions Daffy Duck et Mickey Mouse pour leur contribution.

Sommaire

1. Introduction	3
2. Aperçu des méthodes ensemblistes	4
2.1. Que sont les méthodes ensemblistes?	4
2.2. Pourquoi utiliser des méthodes ensemblistes?	4
2.3. Comment fonctionnent les méthodes ensemblistes?	6
2.3.1. Le modèle de base: l'arbre de classification et de régression	6
2.3.2. Le <i>bagging</i> (Bootstrap Aggregating) et les forêts aléatoires	7
2.3.3. Le <i>gradient boosting</i>	11
2.4. Comparaison entre forêts aléatoires et <i>gradient boosting</i>	13
2.4.1. Quelle approche choisir?	14
2.5. Le point de départ recommandé est de commencer par entraîner une forêt aléatoire avec les hyperparamètres par défaut.	14
3. La forêt aléatoire	15
3.1. Principe de la forêt aléatoire	15
3.2. Comment construit-on une forêt aléatoire?	15
3.3. Pourquoi les forêts aléatoires sont-elles performantes?	16
3.3.1. Réduction de la variance par agrégation	16
3.3.2. Convergence et limite théorique au surapprentissage	16
3.3.3. Facteurs influençant l'erreur de généralisation	17
3.4. Evaluation des performances par l'erreur <i>Out-of-Bag</i> (OOB)	18
3.5. Interprétation et importance des variables	19
3.5.1. Mesures d'importance classiques (et leurs biais)	19
3.5.2. Méthodes d'importance avancées	20
3.6. Préparation des données	21
3.6.1. Préparation des données (Feature Engineering)	21
3.6.2. Process: utiliser les pipelines scikit, pour expliciter la structure du modèle complet et réduire les risques d'erreur	21
3.6.3. Train-test	21
3.7. Evaluation des performances du modèle et optimisation des hyper-paramètres	21

3.7.1. Estimation de l'erreur par validation croisée	21
3.7.2. Choix des hyper-paramètres du modèle	23
4. Guide d'usage des forêts aléatoires	25
4.1. Quelles implémentations utiliser?	25
4.2. Les hyperparamètres clés des forêts aléatoires	25
4.3. Comment entraîner une forêt aléatoire?	28
4.3.1. Approche simple	29
4.3.2. Approches plus avancées	30
4.4. Mesurer l'importance des variables	30
References	32

1. Introduction

Une bien belle introduction pour le site et le DT.

2. Aperçu des méthodes ensemblistes

Principe: cette partie propose une présentation intuitive des méthodes ensemblistes à destination des lecteurs souhaitant un aperçu du fonctionnement et des cas d'utilisation de ces méthodes. Elle ne contient aucun formalisme mathématique.

2.1. Que sont les méthodes ensemblistes?

Les approches ensemblistes désignent un ensemble d'algorithmes de *machine learning* supervisé développés depuis le début des années 1990, c'est-à-dire des méthodes statistiques permettant de prédire une variable-cible y (appelée *target*) à partir d'un ensemble de variables \mathbf{X} (appelées *features*). Elles peuvent par exemple être utilisées pour prédire le salaire d'un salarié, la probabilité de réponse dans une enquête, le niveau de diplôme... Au-delà de leur diversité, ces approches se définissent par un point commun: plutôt que de tenter de construire d'emblée un unique modèle très complexe et très performant, elles visent à obtenir un modèle très performant en combinant intelligemment un ensemble de modèles peu performants, appelés "apprenants faibles" (*weak learner* ou *base learner*). Le choix de ces modèles de base (des arbres de décision dans la plupart des cas) et la manière dont leurs prédictions sont combinées sont des facteurs déterminants pour la performance de ces approches. Le présent document se concentre sur les méthodes ensemblistes à base d'arbres de décisions qui sont parmi les plus utilisées en pratique.

On distingue **deux grandes familles de méthodes ensemblistes** à base d'arbres de décisions, selon qu'elles s'appuient sur des modèles de base entraînés en parallèle indépendamment les uns des autres, ou au contraire entraînés de façon séquentielle. Lorsque les modèles sont *entraînés en parallèle, indépendamment les uns des autres*, on parle de *bagging* ou de forêt aléatoire (*random forest*). Les implémentations les plus courantes des forêts aléatoires sont les *packages* **ranger** en R et **scikit-learn** en Python. Lorsque les modèles de base sont *entraînés de manière séquentielle*, chaque modèle de base visant à améliorer la prédiction proposée par l'ensemble des modèles de base précédents, on parle de *boosting*. Ce document aborde essentiellement le *gradient boosting*, qui est l'approche de *boosting* la plus utilisée actuellement. Les implémentations les plus courantes du *gradient boosting* sont actuellement **XGBoost**, **CatBoost** et **LightGBM**.

2.2. Pourquoi utiliser des méthodes ensemblistes?

Les méthodes ensemblistes sont particulièrement bien adaptées à de nombreux cas d'usage de la statistique publique, pour deux raisons. D'une, elles sont conçues pour s'appliquer à des *données tabulaires* (enregistrements en lignes, variables en colonnes), structure de données om-

niprésente dans la statistique publique. D'autre part, elles peuvent être mobilisées dans toutes les situations où le statisticien mobilise une régression linéaire ou une régression logistique (imputation, repondération...).

Les méthodes ensemblistes présentent trois avantages par rapport aux méthodes économétriques traditionnelles (régression linéaire et régression logistique):

- Elles ont une **puissance prédictive supérieure**: alors que les méthodes traditionnelles supposent fréquemment l'existence d'une relation linéaire ou log-linéaire entre y et \mathbf{X} , les méthodes ensemblistes ne font quasiment aucune hypothèse sur la relation entre y et \mathbf{X} , et se contentent d'approximer le mieux possible cette relation à partir des données disponibles. En particulier, les modèles ensemblistes peuvent facilement modéliser des **non-linéarités** de la relation entre y et \mathbf{X} et des **interactions** entre variables explicatives *sans avoir à les spécifier explicitement* au préalable, alors que les méthodes traditionnelles supposent fréquemment l'existence d'une relation linéaire ou log-linéaire entre y et \mathbf{X} .
- Elles nécessitent **moins de préparation des données**: elles ne requièrent pas de normalisation des variables explicatives et peuvent s'accommoder des valeurs manquantes (selon des techniques variables selon les algorithmes).
- Elles sont généralement **moins sensibles aux valeurs extrêmes et à l'hétéroscédasticité** des variables explicatives que les approches traditionnelles.

Elles présentent par ailleurs deux inconvénients rapport aux méthodes économétriques traditionnelles. Premièrement, bien qu'il existe désormais de multiples approches permettent d'interpréter partiellement les modèles ensemblistes, leur interprétabilité reste moindre que celle d'une régression linéaire ou logistique. Deuxièmement, les modèles ensemblistes sont plus complexes que les approches traditionnelles, et leurs hyperparamètres doivent faire l'objet d'une optimisation, par exemple au travers d'une validation croisée. Ce processus d'optimisation est généralement plus complexe et plus long que l'estimation d'une régression linéaire ou logistique. En revanche, les méthodes ensemblistes sont relativement simples à prendre en main, et ne requièrent pas nécessairement une puissance de calcul importante.

i Et par rapport au *deep learning*?

Si les approches de *deep learning* sont sans conteste très performantes pour le traitement du langage naturel, des images et du son, leur supériorité n'est pas établie pour les applications reposant sur des données tabulaires. Les comparaisons disponibles dans la littérature concluent en effet que les méthodes ensemblistes à base d'arbres sont soit plus performantes que les approches de *deep learning* (L. Grinsztajn, E. Oyallon, and G. Varoquaux [1], R. Schwartz-Ziv and A. Armon [2]), soit font jeu égal avec elles (D. McElfresh *et al.* [3]). Ces études ont identifié trois avantages des méthodes ensemblistes: elles sont peu sensibles aux variables explicatives non pertinentes, robustes aux valeurs extrêmes des variables explicatives, et capables d'approximer des fonctions très irrégulières. De plus, dans la pratique les méthodes ensemblistes sont souvent plus rapides à entraîner et moins gourmandes en ressources informatiques, et l'optimisation des hyperparamètres s'avère souvent moins complexe (R. Schwartz-Ziv and A. Armon [2]).

2.3. Comment fonctionnent les méthodes ensemblistes?

Ce paragraphe présente d'abord le modèle de base sur lesquelles sont construites les méthodes ensemblistes à base d'arbres: l'arbre de classification et de régression (Section 2.3.1). Elle introduit ensuite les deux grandes familles de méthodes ensemblistes couvertes dans ce document: le *bagging* et les forêts aléatoires (Section 2.3.2), puis le *gradient boosting* (Section 2.3.3).

2.3.1. Le modèle de base: l'arbre de classification et de régression

2.3.1.1. Qu'est-ce qu'un arbre CART?

Le modèle de base des méthodes ensemblistes est le plus souvent un arbre de classification et de régression (CART, L. Breiman, J. Friedman, R. Olshen, and C. Stone [4]). Un arbre CART est un algorithme prédictif assez simple avec trois caractéristiques essentielles:

- L'arbre partitionne l'espace des variables explicatives X en régions (appelées feuilles ou *leaves*) les plus homogènes possible, au sens d'une certaine mesure de l'hétérogénéité;
- Chaque région est définie par un ensemble de conditions, appelées règles de décision (*splitting rules*), qui portent sur les valeurs des variables explicatives (par exemple, une région peut être définie par la condition: *age* > 40 et *statut* = 'Cadre');
- Une fois l'arbre construit, les prédictions de l'arbre pour chaque région se déduisent des données d'entraînement de façon intuitive: il s'agira de la classe la plus fréquente parmi les observations situées dans cette région dans le cas d'une classification, et de la moyenne des observations situées dans cette région dans le cas d'une régression.

La structure de cet algorithme a deux conséquences importantes:

- L'algorithme CART ne fait **aucune hypothèse *a priori* sur la relation entre X et y** et se laisse au contraire guider par les données. Par exemple, on ne suppose pas qu'il existe une relation linéaire de type $y = \mathbf{X}\beta$.
- **L'arbre final est une fonction constante par morceaux**: la prédiction est identique pour toutes les observations situées dans la même région, et ne varie que d'une région à l'autre.

Illustration, et représentation graphique (sous forme d'arbre et de graphique).

2.3.1.2. Avantages et limites des arbres CART

Les arbres CART présentent plusieurs avantages: leur principe est simple, ils sont aisément interprétables et peuvent faire l'objet de représentations graphiques intuitives. Par ailleurs, la flexibilité offerte par le partitionnement récursif assure que les arbres obtenus reflètent les corrélations observées dans les données d'entraînement.

Ils souffrent néanmoins de deux limites. D'une part, les arbres CART ont souvent un **pouvoir prédictif faible** qui en limite l'usage. D'autre part, ils sont **peu robustes et instables**: on dit qu'ils présentent une **variance élevée**. Ainsi, un léger changement dans les données (par exemple l'ajout ou la suppression de quelques observations) peut entraîner des modifications significatives dans la structure de l'arbre et dans la définition des feuilles. Les arbres CART sont notamment sensibles aux valeurs extrêmes, aux points aberrants et au bruit statistique. De plus, les prédictions des arbres CART sont sensibles à de petites fluctuations des données: celles-ci peuvent aboutir à ce qu'une partie des observations change brutalement de feuille et donc de valeur prédite.

Les deux familles de méthodes ensemblistes présentées ci-dessous (*bagging*, *random forests* et *gradient boosting*) combinent un grand nombre d'arbres de décision pour en surmonter les deux limites: il s'agit d'obtenir un modèle dont le pouvoir prédictif est élevé et dont les prédictions sont stables. La différence essentielle entre ces deux familles portent sur la façon dont les arbres sont entraînés.

2.3.2. Le *bagging* (Bootstrap Aggregating) et les forêts aléatoires

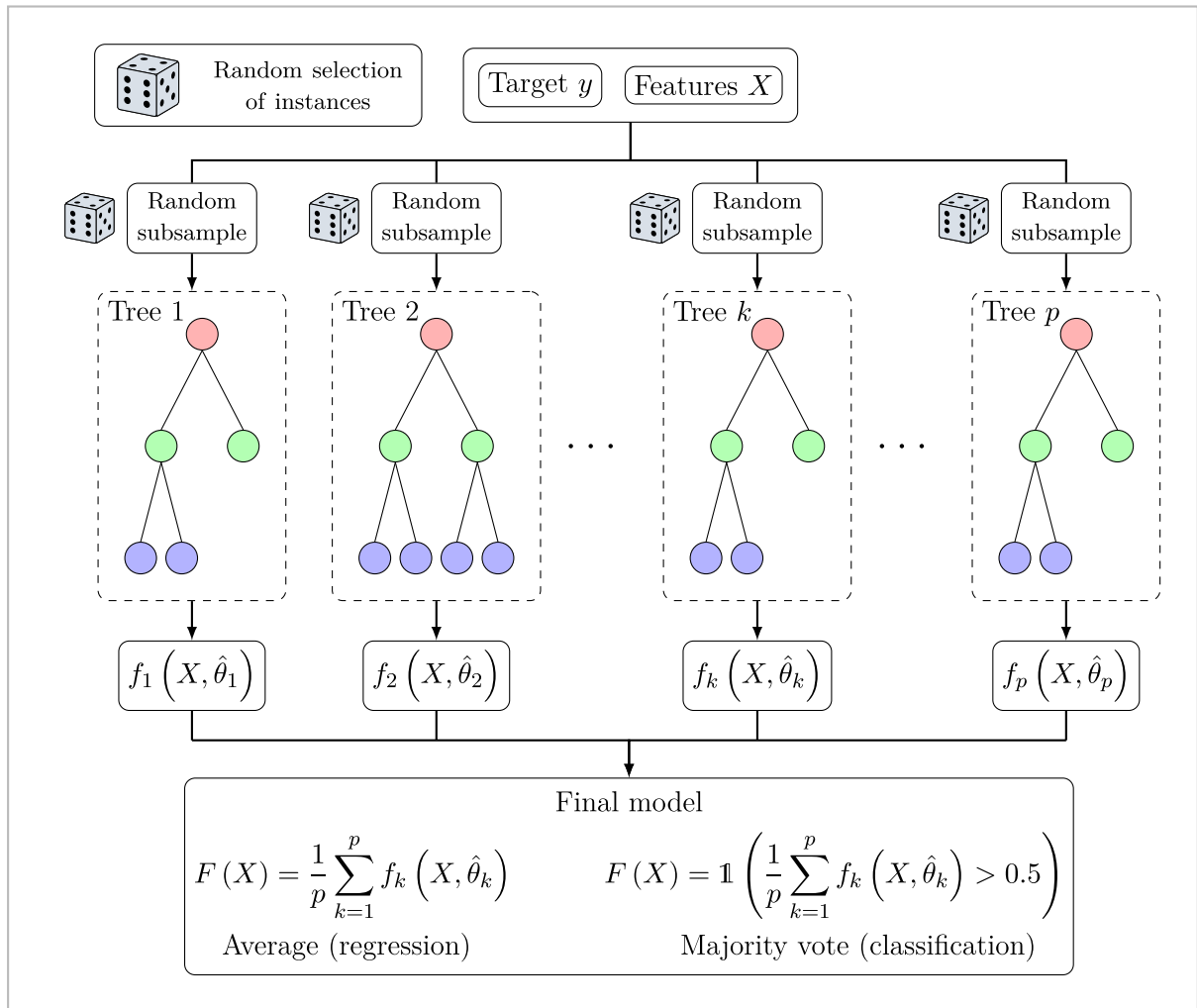
Le *bagging* (Bootstrap Aggregating) et les forêts aléatoires constituent une famille de méthodes ensemblistes dont le point commun est de combiner des modèles de bases qui ont été entraînés indépendamment les uns des autres.

2.3.2.1. Le *bagging*

Le *bagging*, ou *Bootstrap Aggregating* (L. Breiman [5]), est une méthode ensembliste qui comporte trois étapes principales:

- **Tirage de sous-échantillons aléatoires:** À partir du jeu de données initial, plusieurs sous-échantillons sont générés par échantillonnage aléatoire avec remise (*bootstrapping*). Chaque sous-échantillon a la même taille que le jeu de données original, mais peut contenir des observations répétées, tandis que d'autres peuvent être omises.
- **Entraînement parallèle:** Un arbre est entraîné sur chaque sous-échantillon de manière indépendante. Ces arbres sont habituellement assez complexes et profonds.
- **Agrégation des prédictions:** Les prédictions des modèles sont combinées pour produire le résultat final. En classification, la prédiction finale est souvent déterminée par un vote majoritaire, tandis qu'en régression, elle correspond généralement à la moyenne des prédictions.

Figure 1 : Représentation schématique d'un algorithme de *bagging*



La figure [Figure 1](#) propose une représentation schématique du *bagging*: tout d'abord, on tire des sous-échantillons aléatoires des données d'entraînement. Ensuite, un arbre est entraîné sur chaque sous-échantillon. Enfin, les arbres sont combinés de façon à obtenir la prédiction finale.

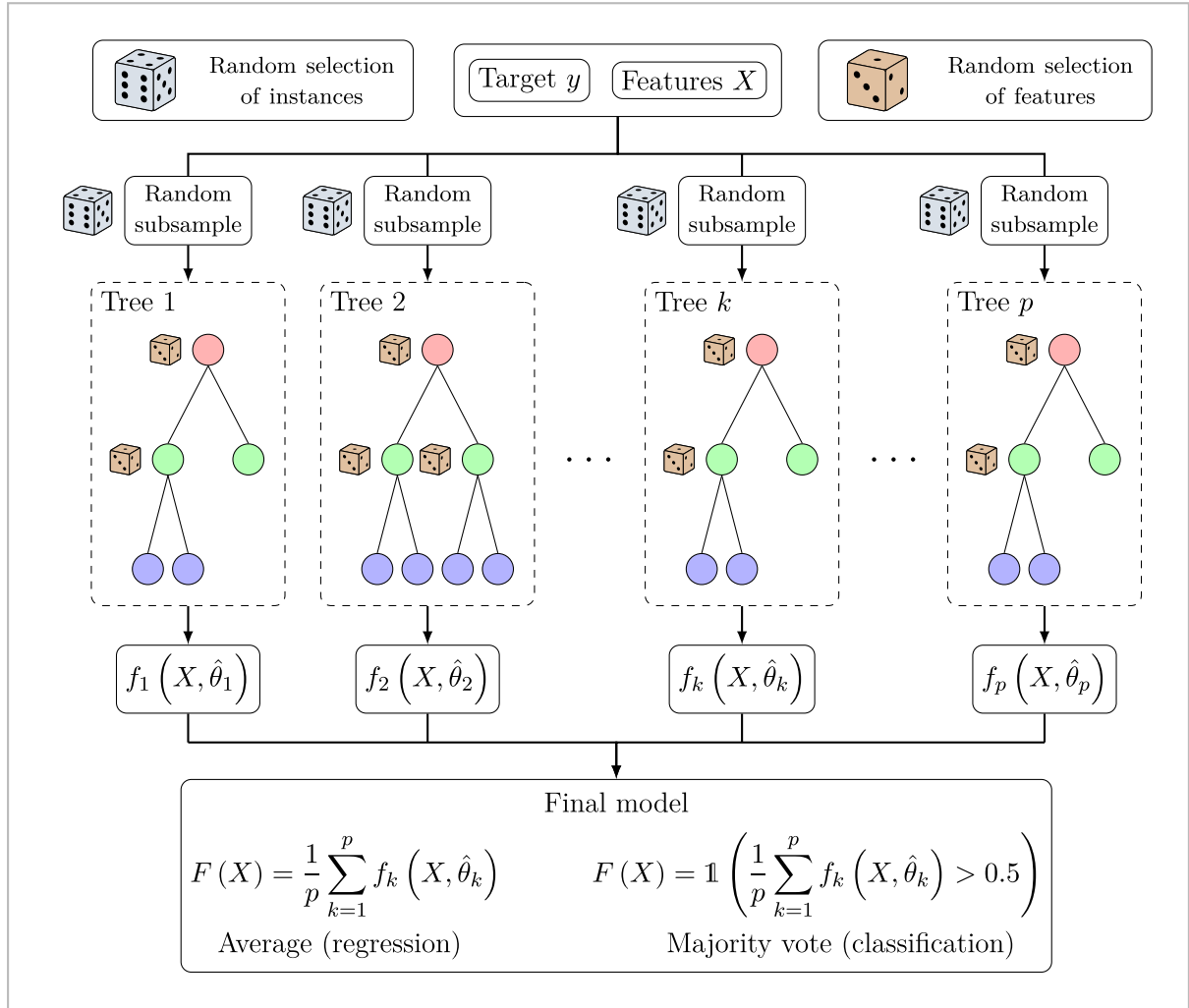
- Illustration avec un cas d’usage de classification en deux dimensions.

L’intuition qui explique l’efficacité du *bagging* est la suivante: en diversifiant les données d’entraînement par le tirage d’échantillons aléatoires, on obtient un grand nombre d’arbres différents les uns des autres et qui, pris dans leur ensemble, constituent un modèle plus prédictif et plus stable que chaque arbre pris isolément. Une image fréquemment employée pour décrire le *bagging* est celle d’un collège de juges. Chaque juge a sa propre façon de juger, qui est imparfaite et qui dépend des cas qu’il a déjà rencontrés. Il peut donc rendre une décision complètement erronée dans telle ou telle situation, rendant son verdict instable et peu fiable. Mais si le verdict repose sur l’opinion majoritaire d’un ensemble de juges différents les uns des autres, il est probable que le jugement sera plus robuste et plus fiable.

Le *bagging* présente donc deux avantages par rapport aux arbres CART: un pouvoir prédictif plus élevé et des prédictions plus stables. L’inconvénient du *bagging* réside dans la corrélation des arbres entre eux: malgré l’échantillonnage des données, les arbres ont souvent une structure similaire car les relations entre variables restent à peu près les mêmes dans les différents sous-échantillons. Ce phénomène de corrélation entre arbres est le principal frein à la puissance prédictive du *bagging*, et c’est pour surmonter (ou au moins minimiser) ce problème que les forêts aléatoires ont été mises au point. Le pouvoir prédictif plus élevé des forêts aléatoires explique pourquoi le *bagging* est très peu utilisé en pratique aujourd’hui.

2.3.2.2. Les *random forests*

Les forêts aléatoires (*random forests*, L. Breiman [6]) sont une variante du *bagging* qui vise à produire des modèles très performants en conciliant deux objectifs: maximiser le pouvoir prédictif des arbres pris isolément, et minimiser la corrélation entre ces arbres (le problème inhérent au *bagging*). Pour atteindre ce second objectif, la forêt aléatoire introduit une nouvelle source de variation aléatoire dans la construction des arbres: au moment de choisir une règle de décision pour diviser une région en deux sous-régions, la procédure d’entraînement ne considère qu’un **sous-ensemble de variables sélectionnées aléatoirement**, et non toutes les variables. Cette randomisation supplémentaire a pour effet mécanique d’aboutir à des arbres plus diversifiés (parce que des arbres différents ne peuvent pas mobiliser les mêmes variables au même moment) et donc de **réduire la corrélation entre arbres**, ce qui permet d’améliorer la performance et la stabilité du modèle agrégé. Un enjeu important de l’entraînement d’une forêt aléatoire est l’arbitrage entre puissance prédictive des arbres et corrélation entre arbres.

Figure 2 : Représentation schématique d'un algorithme de forêt aléatoire


La figure [Figure 2](#) propose une représentation schématique d'une forêt aléatoire. La logique d'ensemble est identique à celle du *bagging*: combinés de façon à obtenir la prédiction finale. La seule différence est que la liste des variables utilisables pour construire des règles de décision varie à chaque étape de l'entraînement. Cette restriction de la liste des variables considérées permet de réduire l'utilisation des variables les plus prédictives et de mieux mobiliser l'information disponible dans les variables peu corrélées avec y .

Contrairement au *bagging*, les forêts aléatoires sont un algorithme qui est très largement employé pour plusieurs raisons: les forêts aléatoires ont un faible nombre d'hyperparamètres, sont généralement peu sensibles aux valeurs de ces hyperparamètres et proposent de bonnes performances avec les valeurs par défaut. Les forêts aléatoires sont toutefois sujettes au problème de surapprentissage (voir encadré), bien que dans une mesure moindre que le *gradient boosting*.

Les forêts aléatoires présentent également un avantage de taille: **il est possible d'évaluer la qualité d'une forêt aléatoire en utilisant les données sur lesquelles elle a été entraînée**, sans avoir besoin d'un jeu de test séparé. En effet, lors de la construction de chaque arbre, l'échantillonnage aléatoire implique que certaines observations ne sont pas utilisées pour entraîner cet arbre; ces observations sont dites *out-of-bag*. On peut donc construire pour chaque observation une prédiction qui agrège uniquement les arbres pour lesquels cette observation est *out-of-bag*; cette prédiction n'est pas affectée par le surapprentissage. De cette façon, il est possible d'évaluer correctement la performance de la forêt aléatoire.

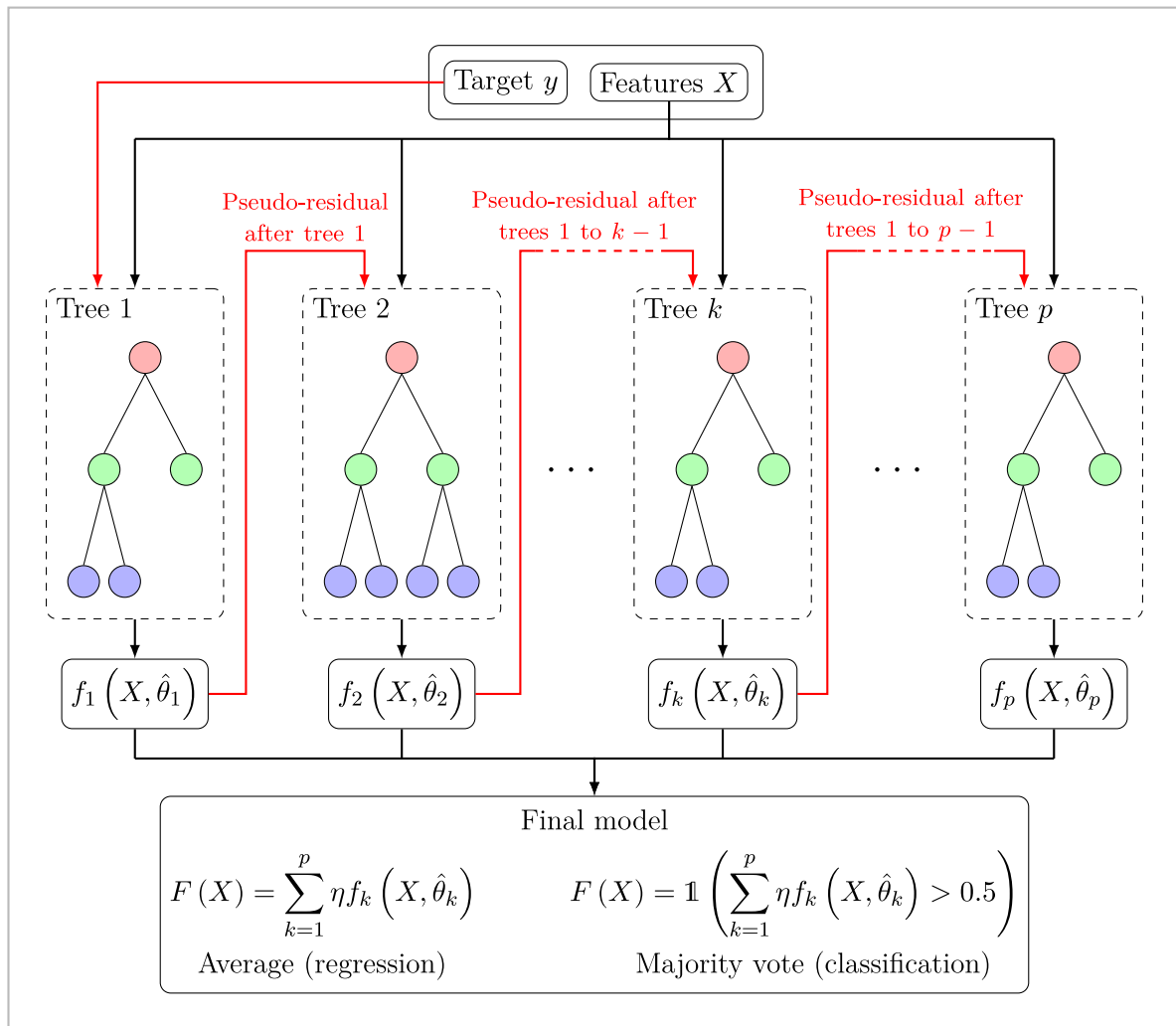
i Qu'est-ce que le surapprentissage?

Le surapprentissage (*overfitting*) est un phénomène fréquent en *machine learning* où un modèle apprend non seulement les relations sous-jacentes entre la variable cible et les variables explicatives, mais également le bruit présent dans les données d'entraînement. En capturant ces fluctuations aléatoires plutôt que les tendances générales, le modèle affiche une performance excellente mais trompeuse sur les données d'entraînement, et s'avère médiocre sur des données nouvelles ou de test, car il ne parvient pas à généraliser efficacement.

2.3.3. Le *gradient boosting*

Alors que les forêts aléatoires construisent un ensemble d'arbres complexes et indépendants les uns des autres, le *gradient boosting* adopte une autre approche, dans laquelle les arbres sont peu complexes et entraînés de façon séquentielle, chaque arbre essayant d'améliorer la prédiction proposée par l'ensemble des arbres précédents. Bien qu'elles ressemblent fortement aux forêts aléatoires en apparence, il est important de noter que les approches de *boosting* reposent sur des fondements théoriques très différents. La logique du *gradient boosting* est illustrée par la figure [Figure 3](#):

Figure 3 : Représentation schématique d'un algorithme de *gradient boosting*



- Un premier modèle simple et peu performant est entraîné sur les données.
- Un deuxième modèle est entraîné de façon à corriger les erreurs du premier modèle (par exemple en pondérant davantage les observations mal prédites);
- Ce processus est répété en ajoutant des modèles simples, chaque modèle corrigeant les erreurs commises par l'ensemble des modèles précédents;
- Tous ces modèles sont finalement combinés (souvent par une somme pondérée) pour obtenir un modèle complexe et performant.

Il s'avère que le *gradient boosting* offre des performances prédictives particulièrement élevées. Toutefois, cet avantage incontestable ne doit pas masquer les sérieux inconvénients de cette approche: les algorithmes de *gradient boosting* comprennent un nombre élevé d'hyperparamètres et sont plus sensibles que les forêts aléatoires aux valeurs de ces hyperparamètres. Par ailleurs, ces algorithmes se caractérisent par un risque élevé de surapprentissage, et sont assez sensibles

au bruit statistique et aux éventuelles erreurs sur la variable-cible. Par conséquent, l’usage de ces algorithmes est plus délicat, et l’optimisation de leurs hyperparamètres est une étape importante qui peut prendre un certain temps et demande une bonne connaissance des algorithmes. Enfin, il est indispensable de disposer d’un jeu de données de test (non utilisées pendant l’entraînement) pour évaluer la qualité d’un modèle de *boosting*.

2.4. Comparaison entre forêts aléatoires et *gradient boosting*

Les forêts aléatoires et le *gradient boosting* paraissent très similaires au premier abord: il s’agit de deux approches ensemblistes, qui construisent des modèles très prédictifs performants en combinant un grand nombre d’arbres de décision. Mais en réalité, ces deux approches présentent plusieurs différences fondamentales:

- Les deux approches reposent sur des **fondements théoriques différents**: la loi des grands nombres pour les forêts aléatoires, la théorie de l’apprentissage statistique pour le *boosting*.
- **Les arbres n’ont pas le même statut dans les deux approches**. Dans une forêt aléatoire, les arbres sont entraînés indépendamment les uns des autres et constituent chacun un modèle à part entière, qui peut être utilisé, représenté et interprété isolément. Dans un modèle de *boosting*, les arbres sont entraînés séquentiellement, ce qui implique que chaque arbre n’a pas de sens indépendamment de l’ensemble des arbres qui l’ont précédé dans l’entraînement. Par ailleurs, les arbres d’une forêt aléatoire sont relativement complexes et profonds (car ce sont des modèles à part entière), alors que dans le *boosting* les arbres sont plus souvent simples et peu profonds.
- Les **points d’attention lors de l’entraînement** des algorithmes sont différents: l’enjeu principal de l’entraînement d’une forêt aléatoire est trouver le bon arbitrage entre puissance prédictive des arbres et corrélation entre arbres, tandis que l’entraînement d’un algorithme de *gradient boosting* porte davantage sur la lutte contre le surapprentissage.
- **Complexité d’usage**: les forêts aléatoires s’avèrent plus faciles à prendre en main que le *gradient boosting*, car elles comprennent moins d’hyperparamètres dont l’optimisation est moins complexe.
- **Conditions d’utilisation**: il est possible d’évaluer la qualité d’une forêt aléatoire en utilisant les données sur lesquelles elle a été entraînée grâce à l’approche *out-of-bag*, alors que c’est impossible avec le *gradient boosting*, pour lequel il faut impérativement conserver un ensemble de test. Cette différence peut sembler purement technique en

apparence, mais elle s'avère importante en pratique dans de nombreuses situations, par exemple lorsque les données disponibles sont de taille restreinte ou lorsque les ressources informatiques disponibles ne sont pas suffisantes pour mener un exercice de validation croisée.

2.4.1. Quelle approche choisir?

2.5. Le point de départ recommandé est de commencer par entraîner une forêt aléatoire avec les hyperparamètres par défaut.

3. La forêt aléatoire

La forêt aléatoire (*random forests*) est une méthode ensembliste puissante, largement utilisée pour les tâches de classification et de régression. Elle combine la simplicité des arbres de décision et l'échantillonnage des observations et des variables avec la puissance de l'agrégation pour améliorer les performances prédictives et réduire le risque de surapprentissage (*overfitting*).

3.1. Principe de la forêt aléatoire

La forêt aléatoire est une extension du *bagging*, présenté dans la section [?@sec-bagging-detail](#). Elle introduit un niveau supplémentaire de randomisation dans la construction des arbres, puisqu'à chaque nouvelle division (*noeud*), le critère de séparation est choisi en considérant uniquement un sous-ensemble de variables **sélectionné aléatoirement**. Cette randomisation supplémentaire **réduit la corrélation** entre les arbres, ce qui permet de diminuer la variance des prédictions du modèle agrégé.

Les forêts aléatoires reposent sur quatre éléments essentiels:

- **Les arbres CART**: Les modèles élémentaires sont des arbres CART non élagués, c'est-à-dire autorisés à pousser jusqu'à l'atteinte d'un critère d'arrêt défini en amont.
- **L'échantillonnage *bootstrap***: Chaque arbre est construit à partir d'un échantillon aléatoire du jeu de données d'entraînement tiré avec remise (ou parfois sans remise).
- **La sélection aléatoire de variables** : Lors de la construction d'un arbre, à chaque nœud de celui-ci, un sous-ensemble aléatoire de variables est sélectionné. La meilleure division est ensuite choisie parmi ces caractéristiques aléatoires.
- **L'agrégation des prédictions** : Comme pour le *bagging*, les prédictions de tous les arbres sont combinées. On procède généralement à la moyenne (ou à la médiane) des prédictions dans le cas de la régression, et au vote majoritaire (ou à la moyenne des probabilités prédites pour chaque classe) dans le cas de la classification.

3.2. Comment construit-on une forêt aléatoire?

L'entraînement d'une forêt aléatoire est très similaire à celui du *bagging* et se résume comme suit:

- Le nombre d'arbres à construire est défini *a priori*.
- Pour chaque arbre, on effectue les étapes suivantes:
 - Générer un échantillon *bootstrap* de taille fixe à partir des données d'entraînement.
 - Construire récursivement un arbre de décision à partir de cet échantillon:

- À chaque nœud de l’arbre, un sous-ensemble de *features* est sélectionné aléatoirement.
- Déterminer quel couple (variable, valeur) définit la règle de décision qui divise la population du nœud en deux sous-groupes les plus homogènes possibles.
- Créer les deux nœuds-enfants à partir de cette règle de décision.
- Arrêter la croissance de l’arbre selon des critères d’arrêt fixés *a priori*.

Pour construire la prédiction de la forêt aléatoire une fois celle-ci entraînée, on agrège les arbres selon une méthode qui dépend du problème modélisé:

- Régression: la prédiction finale est la moyenne des prédictions de tous les arbres.
- Classification: chaque arbre vote pour une classe, et la classe majoritaire est retenue.

Les principaux hyper-paramètres des forêts aléatoires (détaillés dans la section [Section 4](#)) sont les suivants: le nombre d’arbres, la méthode et le taux d’échantillonnage, le nombre (ou la proportion) de variables considérées à chaque nœud, le critère de division des nœuds (ou mesure d’hétérogénéité), et les critères d’arrêt (notamment la profondeur de l’arbre, le nombre minimal d’observations dans une feuille terminale, et le nombre minimal d’observations qu’un nœud doit comprendre pour être divisé en deux).

3.3. Pourquoi les forêts aléatoires sont-elles performantes?

Les propriétés théoriques des forêts aléatoires permettent de comprendre pourquoi (et dans quelles situations) elles sont particulièrement robustes et performantes.

3.3.1. Réduction de la variance par agrégation

L’agrégation de plusieurs arbres permet de réduire la variance globale du modèle, ce qui améliore la stabilité des prédictions. Lorsque les estimateurs sont (faiblement) biaisés mais caractérisés par une variance élevée, l’agrégation permet d’obtenir un estimateur avec un biais similaire mais une variance réduite. La démonstration est identique à celle présentée dans la section [?@sec-bagging-detail](#).

3.3.2. Convergence et limite théorique au surapprentissage

Bien qu’elle s’avèrent très performantes en pratique, **il n’est pas prouvé à ce stade que les forêts aléatoires convergent vers une solution optimale** lorsque la taille de l’échantillon tend vers l’infini ([G. Louppe \[7\]](#)). Plusieurs travaux théoriques ont toutefois fourni des preuves de convergence pour des versions simplifiées de l’algorithme (par exemple, [G. Biau \[8\]](#)).

Par ailleurs, une propriété importante des forêts aléatoires démontrée par [L. Breiman \[6\]](#) est que leur erreur de généralisation, c’est-à-dire l’écart entre les prédictions du modèle et les

résultats attendus sur des données jamais vues (donc hors de l'échantillon d'entraînement), diminue à mesure que le nombre d'arbres augmente et converge vers une valeur constante. Autrement dit, **la forêt aléatoire ne souffre pas d'un surapprentissage croissant avec le nombre d'arbres**. La conséquence pratique de ce résultat est qu'inclure un (trop) grand nombre d'arbres dans le modèle n'en dégrade pas la qualité, ce qui contribue à la rendre particulièrement robuste. En revanche, une forêt aléatoire peut souffrir de surapprentissage si ses autres hyperparamètres sont mal choisis (des arbres trop profonds par exemple).

3.3.3. Facteurs influençant l'erreur de généralisation

L'erreur de généralisation des forêts aléatoires est influencée par deux facteurs principaux :

- **La puissance prédictrice des arbres individuels** : Les arbres doivent être suffisamment prédictifs pour contribuer positivement à l'ensemble, et idéalement sans biais.
- **La corrélation entre les arbres** : Moins les arbres sont corrélés, plus la variance de l'ensemble est réduite, car leurs erreurs tendront à se compenser. Inversement, des arbres fortement corrélés auront tendance à faire des erreurs similaires, donc agréger un grand nombre d'arbres n'apportera pas grand chose.

On peut mettre en évidence ces deux facteurs dans le cas d'une forêt aléatoire utilisée pour une tâche de régression (où l'objectif est de minimiser l'erreur quadratique moyenne). Dans ce cas, la variance de la prédiction du modèle peut être décomposée de la façon suivante:

$$\text{Var}(\hat{f}(x)) = \rho(x)\sigma(x)^2 + \frac{1-\rho(x)}{M}\sigma(x)^2 \quad (1)$$

où $\rho(x)$ est le coefficient de corrélation moyen entre les arbres individuels, $\sigma(x)^2$ est la variance d'un arbre individuel, M est le nombre d'arbres dans la forêt. Cette décomposition fait apparaître l'influence de la corrélation entre les arbres sur les performance de la forêt aléatoire:

- **Si $\rho(x)$ est proche de 1** (forte corrélation entre les arbres) : la première composante $\rho\sigma^2$ domine et la réduction de variance est moindre lorsque le nombre d'arbres augmente.
- **Si $\rho(x)$ est proche de 0** (faible corrélation entre les arbres) : la seconde composante $\frac{1-\rho}{M}\sigma^2$ et la variance est davantage réduite avec l'augmentation du nombre d'arbres M .

L'objectif de l'entraînement des forêts aléatoires est donc de minimiser la corrélation entre les arbres tout en maximisant leur capacité à prédire correctement, ce qui permet de réduire la variance globale sans augmenter excessivement le biais. La sélection aléatoires des caractéristiques (*features*) à chaque nœud joue un rôle majeur dans cet arbitrage entre puissance prédictive des arbres et corrélation entre arbres.

3.4. Evaluation des performances par l'erreur *Out-of-Bag* (OOB)

La forêt aléatoire présente une particularité intéressante et très utile en pratique: **il est possible d'évaluer les performances d'une forêt aléatoire directement à partir des données d'entraînement**, grâce à l'estimation de l'erreur *Out-of-Bag* (OOB). Cette technique repose sur le fait que chaque arbre est construit à partir d'un échantillon *bootstrap*, c'est-à-dire un échantillon tiré avec remise. Cela implique qu'une part conséquente des observations ne sont pas utilisées pour entraîner un arbre donné. Ces observations laissées de côté forment un **échantillon dit *out-of-bag***, que l'on peut utiliser pour évaluer la performance de chaque arbre. On peut donc construire pour chaque observation du jeu d'entraînement une prédiction qui agrège uniquement les prédictions des arbres pour lesquels cette observation est *out-of-bag*; cette prédiction n'est pas affectée par le surapprentissage (puisque cette observation n'a jamais été utilisée pour entraîner ces arbres). De cette façon, il est possible d'évaluer correctement la performance de la forêt aléatoire en comparant ces prédictions avec la variable-cible à l'aide d'une métrique bien choisie.

La procédure d'estimation de l'erreur OOB se déroule comme ceci:

1. **Entraînement de la forêt aléatoire:** la forêt aléatoire est entraînée sur les données d'entraînement selon la procédure détaillée ci-dessus.
2. **Prédiction *out-of-bag* :** Pour chaque observation (x_i, y_i) des données d'entraînement, on calcule la prédiction de tous les arbres pour lesquels elle fait partie de l'échantillon *out-of-bag*.
3. **Agrégation des prédictions :** La prédiction finale est obtenue en agrégeant les prédictions selon la procédure standard détaillée ci-dessus (moyenne pour la régression, vote majoritaire pour la classification).
4. **Calcul de l'erreur OOB :** L'erreur OOB est ensuite calculée en comparant les prédictions avec la variable-cible y sur toutes les observations, à l'aide d'une métrique (précision, rappel, AUC, erreur quadratique moyenne, score de Brier...).

L'utilisation de l'erreur OOB présente de multiples avantages:

- **Approximation de l'erreur de généralisation:** L'erreur OOB est en général considérée comme une bonne approximation de l'erreur de généralisation, comparable à celle obtenue par une validation croisée.
- **Pas besoin de jeu de validation séparé :** L'un des principaux avantages de l'erreur OOB est qu'elle ne nécessite pas de réserver une partie des données pour la validation. Cela est particulièrement utile lorsque la taille du jeu de données est limitée, car toutes les données peuvent être utilisées pour l'entraînement tout en ayant une estimation

fiable de la performance. Ceci dit, il est malgré tout recommandé de conserver un ensemble de test si la taille des données le permet, car il arrive que l'erreur OOB sous

- **Gain de temps** : Contrairement à la validation croisée qui requiert de réentraîner plusieurs fois le modèle pour un jeu donné d'hyperparamètres, l'erreur OOB ne nécessite qu'un seul entraînement du modèle. Cela induit un gain de temps appréciable lors de l'optimisation des hyperparamètres.

3.5. Interprétation et importance des variables

Les forêts aléatoires sont des modèles d'apprentissage performants, mais leur complexité interne les rend difficiles à interpréter, ce qui leur vaut souvent le qualificatif de “boîtes noires”. Comprendre l'influence des variables explicatives sur les prédictions est crucial pour interpréter les résultats et être en mesure d'extraire des connaissances.

L'objectif des **méthodes d'interprétabilité** (ou d'importance des variables) est d'identifier les variables les plus influentes sur la variable cible, de comprendre les mécanismes prédictifs sous-jacents, et potentiellement d'extraire des règles de décision simples et transparentes. Plusieurs méthodes d'importance des variables existent, mais il est important de comprendre leurs forces et faiblesses.

3.5.1. Mesures d'importance classiques (et leurs biais)

- **Réduction moyenne de l'impureté** (*Mean Decrease in Impurity - MDI*) : Cette méthode quantifie l'importance d'une variable par la somme des réductions d'impureté qu'elle induit dans tous les arbres de la forêt. Plus spécifiquement, pour chaque variable, on s'intéresse à la moyenne des réductions d'impureté qu'elle a engendrées dans tous les nœuds de tous les arbres où elle est impliquée. Les variables présentant la réduction moyenne d'impureté la plus élevée sont considérées comme les prédicteurs les plus importants.

La MDI présente des biais importants. Elle est notamment sensible aux variables catégorielles avec de nombreuses modalités, qui peuvent apparaître artificiellement importantes (même si leur influence réelle est faible), ainsi qu'aux variables avec une échelle de valeurs plus étendues, qui obtiennent des scores plus élevés, indépendamment de leur importance réelle. Elle est également fortement biaisée en présence de variables explicatives corrélées, ce qui conduit à surestimer l'importance de variables redondantes. Les interactions entre variables ne sont pas non plus prises en compte de manière adéquate.

- **Importance par permutation** (*Mean Decrease Accuracy - MDA*) : Cette méthode évalue l'importance d'une variable en mesurant la diminution de précision du modèle après permutation aléatoire de ses valeurs. Plus spécifiquement, pour chaque variable,

les performances du modèle sont comparées avant et après la permutation de ses valeurs. La différence moyenne de performance correspond à la MDA. L'idée est que si l'on permute aléatoirement les valeurs d'une variable (cassant ainsi sa relation avec la cible), une variable importante entraînera une hausse significative de l'erreur de généralisation.

Comme la MDI, la MDA présente des biais lorsque les variables sont corrélées. En particulier, la MDA peut surévaluer l'importance de variables qui sont corrélées à d'autres variables importantes, même si elles n'ont pas d'influence directe sur la cible (C. Bénard, S. Da Veiga, and E. Scornet [9]).

Plusieurs stratégies peuvent aider à réduire les biais d'interprétation :

- **Prétraitement des variables:** Standardisation des variables, regroupement des modalités rares des variables catégorielles, réduction de la cardinalité des variables catégorielles.
- **Analyse des corrélations:** Identification et gestion des variables fortement corrélées, qui peuvent fausser les mesures d'importance.
- **Choix de méthodes robustes:** Privilégier les méthodes moins sensibles aux biais, comme les CIF ou la Sobol-MDA, et, le cas échéant, SHAFF pour les valeurs de Shapley. Ces méthodes sont présentées dans la section suivante.

3.5.2. Méthodes d'importance avancées

Pour pallier les limites des méthodes traditionnelles, des approches plus sophistiquées ont été développées.

- **Valeurs de Shapley:** Les valeurs de Shapley permettent de quantifier la contribution de chaque variable explicative à la variance expliquée de la variable cible, en tenant compte des interactions entre les variables. Elles attribuent à chaque variable une contribution marginale moyenne à la performance du modèle, en considérant toutes les combinaisons possibles de sous-ensembles de variables. Cependant, l'estimation des valeurs de Shapley est computationnellement coûteuse (complexité exponentielle avec le nombre de variables). Des méthodes approximatives existent, mais peuvent introduire des biais. L'algorithme SHAFF (C. Bénard, G. Biau, S. Da Veiga, and E. Scornet [10]) propose une solution rapide et précise à ce problème, en tirant parti des propriétés des forêts aléatoires.
- **Conditional Inference Forests (CIF):** Les CIF (C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn [11]), implémentées dans le package party de R (cforest), corrigent certains biais de la MDI en utilisant des tests statistiques conditionnels pour sélectionner les variables et les seuils de coupure dans les arbres. Elles sont particulièrement robustes face aux variables hétérogènes et aux corrélations entre variables. Couplées

à un échantillonnage sans remise, les CIF fournissent des mesures d'importance plus fiables.

- **Sobol-MDA:** La Sobol-MDA combine l'idée de la MDA avec une approche basée sur les indices de Sobol, permettant de gérer efficacement les variables dépendantes. Au lieu de permuter les valeurs, elle projette la partition des arbres sur le sous-espace excluant la variable dont on souhaite mesurer l'importance, simulant ainsi son absence. Elle est plus efficace en calcul que les méthodes MDA classiques tout en fournissant une mesure d'importance cohérente, convergeant vers l'indice de Sobol total (la mesure appropriée pour identifier les covariables les plus influentes, même avec des dépendances) (C. Bernard, S. Da Veiga, and E. Scornet [9]).

3.6. Préparation des données

3.6.1. Préparation des données (Feature Engineering)

- **Valeurs manquantes :** Les forêts aléatoires peuvent gérer les données manquantes, mais une imputation préalable peut améliorer les performances.
- **Variables catégorielles :** Utiliser un encodage adapté (one-hot encoding, ordinal encoding) en fonction de la nature des données. Convertir en variables continues dès que c'est possible.
- **Échelle des Variables :** Pas nécessaire de normaliser, les arbres sont invariants aux transformations monotones.

3.6.2. Process: utiliser les pipelines scikit, pour expliciter la structure du modèle complet et réduire les risques d'erreur

3.6.3. Train-test

Pas indispensable pour RF, mais souhaitable. Indispensable pour GB.

3.7. Evaluation des performances du modèle et optimisation des hyper-paramètres

3.7.1. Estimation de l'erreur par validation croisée

La validation croisée est une méthode d'évaluation couramment utilisée en apprentissage automatique pour estimer la capacité d'un modèle à généraliser les prédictions à de nouvelles données. Bien que l'évaluation par l'erreur *Out-of-Bag* (OOB) soit généralement suffisante pour les forêts aléatoires, la validation croisée permet d'obtenir une évaluation plus robuste, car

moins sensible à l'échantillon d'entraînement, notamment sur des jeux de données de petite taille.

Concrètement, le jeu de données est divisé en k sous-ensembles, un modèle est entraîné sur $k - 1$ sous-ensembles et testé sur le sous-ensemble restant. L'opération est répétée k fois de manière à ce que chaque observation apparaisse au moins une fois dans l'échantillon test. L'erreur est ensuite moyennée sur l'ensemble des échantillons test.

Procédure de validation croisée:

La validation croisée la plus courante est la validation croisée en k sous-échantillons (*k-fold cross-validation*):

- **Division des données** : Le jeu de données est divisé en k sous-échantillons égaux, appelés folds. Typiquement, k est choisi entre 5 et 10, mais il peut être ajusté en fonction de la taille des données.
- **Entraînement et test** : Le modèle est entraîné sur $k - 1$ sous-échantillons et testé sur le sous-échantillon restant. Cette opération est répétée k fois, chaque sous-échantillon jouant à tour de rôle le rôle de jeu de test.
- **Calcul de la performance** : Les k performances obtenues (par exemple, l'erreur quadratique moyenne pour une régression, ou l'accuracy (*exactitude*) pour une classification) sont moyennées pour obtenir une estimation finale de la performance du modèle.

Avantages de la validation croisée:

- **Utilisation optimale des données** : En particulier lorsque les données sont limitées, la validation croisée maximise l'utilisation de l'ensemble des données en permettant à chaque échantillon de contribuer à la fois à l'entraînement et au test.
- **Réduction de la variance** : En utilisant plusieurs divisions des données, on obtient une estimation de la performance moins sensible aux particularités d'une seule division.

Bien que plus coûteuse en termes de calcul, la validation croisée est souvent préférée lorsque les données sont limitées ou lorsque l'on souhaite évaluer différents modèles ou hyperparamètres avec précision.

Leave-One-Out Cross-Validation (LOOCV) : Il s'agit d'un cas particulier où le nombre de sous-échantillons est égal à la taille du jeu de données. En d'autres termes, chaque échantillon est utilisé une fois comme jeu de test, et tous les autres échantillons pour l'entraînement. LOOCV fournit une estimation très précise de la performance, mais est très coûteuse en temps de calcul, surtout pour de grands jeux de données.

3.7.2. Choix des hyper-paramètres du modèle

L'estimation Out-of-Bag (OOB) et la validation croisée sont deux méthodes clés pour optimiser les hyper-paramètres d'une forêt aléatoire. Les deux approches permettent de comparer les performances obtenues pour différentes combinaisons d'hyper-paramètres et de sélectionner celles qui maximisent les performances prédictives, l'OOB étant souvent plus rapide et moins coûteuse, tandis que la validation croisée est plus fiable dans des situations où le surapprentissage est un risque important (P. Probst, M. N. Wright, and A.-L. Boulesteix [12]).

Il convient de définir une stratégie d'optimisation des hyperparamètres pour ne pas perdre de temps à tester trop de jeux d'hyperparamètres. Plusieurs stratégies existent pour y parvenir, les principales sont exposées dans la section [Section 4](#). Les implémentations des forêts aléatoires disponibles en R et en Python permettent d'optimiser aisément les principaux hyper-paramètres des forêts aléatoires.

3.7.2.1. Méthodes de recherche exhaustives

- **Recherche sur grille** (Grid Search): Cette approche simple explore toutes les combinaisons possibles d'hyperparamètres définis sur une grille. Les paramètres continus doivent être discrétisés au préalable. La méthode est exhaustive mais coûteuse en calcul, surtout pour un grand nombre d'hyperparamètres.
- **Recherche aléatoire** (Random Search): Plus efficace que la recherche sur grille, cette méthode échantillonne aléatoirement les valeurs des hyperparamètres dans un espace défini. Bergstra et Bengio (2012) ont démontré sa supériorité pour les réseaux neuronaux, et elle est également pertinente pour les forêts aléatoires. La distribution d'échantillonnage est souvent uniforme.

3.7.2.2. Optimisation séquentielle/itérative basée sur un modèle (SMBO)

La méthode SMBO (Sequential model-based optimization) est une approche plus efficace que les précédentes car elle s'appuie sur les résultats des évaluations déjà effectuées pour guider la recherche des prochains hyper-paramètres à tester (P. Probst, M. N. Wright, and A.-L. Boulesteix [12]).

Voici les étapes clés de cette méthode:

- **Définition du problème:** On spécifie une mesure d'évaluation (ex: AUC pour la classification, MSE pour la régression), une stratégie d'évaluation (ex: validation croisée k-fold), et l'espace des hyperparamètres à explorer.
- **Initialisation:** échantillonner aléatoirement des points dans l'espace des hyper-paramètres et évaluer leurs performances.

- Boucle itérative :
 - Construction d'un modèle de substitution (surrogate model): un modèle de régression (ex: krigeage ou une forêt aléatoire) est ajusté aux données déjà observées. Ce modèle prédit la performance en fonction des hyperparamètres.
 - Sélection d'un nouvel hyperparamètre: un critère basé sur le modèle de substitution sélectionne le prochain ensemble d'hyperparamètres à évaluer. Ce critère vise à explorer des régions prometteuses de l'espace des hyperparamètres qui n'ont pas encore été suffisamment explorées.
 - Évaluer les points proposés et les ajouter à l'ensemble déjà exploré: la performance du nouvel ensemble d'hyperparamètres est évaluée et ajoutée à l'ensemble des données d'apprentissage du modèle de substitution afin d'orienter les recherches vers de nouveaux hyper-paramètres prometteurs.

4. Guide d’usage des forêts aléatoires

Ce guide d’entraînement des forêts aléatoires rassemble et synthétise des recommandations sur l’entraînement des forêts aléatoires disponibles dans la littérature, en particulier dans [P. Probst, M. N. Wright, and A.-L. Boulesteix \[12\]](#). Ce guide comporte un certain nombre de choix méthodologiques forts, comme les implémentations recommandées ou la procédure d’entraînement proposée, et d’autres choix pertinents sont évidemment possibles. C’est pourquoi les recommandations de ce guide doivent être considérées comme un point de départ raisonnable, pas comme un ensemble de règles devant être respectées à tout prix.



4.1. Quelles implémentations utiliser?

Il existe de multiples implémentations des forêts aléatoires. Le présent document présente et recommande l’usage de deux implémentations de référence: le *package* **R** `ranger` et le *package* **Python** `scikit-learn` pour leur rigueur, leur efficacité et leur simplicité d’utilisation. Il est à noter qu’il est possible d’entraîner des forêts aléatoires avec les algorithmes **XGBoost** et **LightGBM**, mais il s’agit d’un usage avancé qui n’est pas recommandé en première approche. Cette approche est présentée dans la partie **REFERENCE A LA PARTIE USAGE AVANCE**.

4.2. Les hyperparamètres clés des forêts aléatoires

Cette section décrit en détail les principaux hyperparamètres des forêts aléatoires listés dans le tableau [Table 1](#). Les noms des hyperparamètres utilisés sont ceux figurant dans le *package* **R** `ranger`, et dans le *package* **Python** `scikit-learn`. Il arrive qu’ils portent un nom différent dans d’autres implémentations des forêts aléatoires, mais il est généralement facile de s’y retrouver en lisant attentivement la documentation.

Table 1 : Les principaux hyperparamètres des forêts aléatoires

Hyperparamètre		Description
 ranger	 scikit-learn	
num.trees	n_estimators	Le nombre d'arbres
mtry	max_features	Le nombre de variables candidates à chaque noeud
sample.fraction	max_samples	Le taux d'échantillonnage des données
replacement		L'échantillonnage des données se fait-il avec ou sans remise?
min.node.size	min_samples_leaf	Nombre minimal d'observations nécessaire pour qu'un noeud puisse être partagé
min.bucket	min_samples_split	Nombre minimal d'observations dans les noeuds terminaux
max.depth	max_depth	Profondeur maximale des arbres
splitrule	criterion	Le critère de choix de la règle de division des noeuds intermédiaires
oob.error	oob_score	Calculer la performance de la forêt par l'erreur OOB (et choix de la métrique pour scikit)

- Le **nombre d'arbres** par défaut varie selon les implémentations (500 dans **ranger**, 100 dans **scikit-learn**). Il s'agit d'un hyperparamètre particulier car il n'est associé à aucun arbitrage en matière de performance: la performance de la forêt aléatoire croît avec le nombre d'arbres, puis se stabilise. Le nombre optimal d'arbres est celui à partir duquel la performance de la forêt ne croît plus (ce point est détaillé plus bas) où à partir duquel l'ajout d'arbres supplémentaires génère des gains marginaux. Il est important de noter que ce nombre optimal dépend des autres hyperparamètres. Par exemple, un taux d'échantillonnage faible et un nombre faible de variables candidates à chaque noeud aboutissent à des arbres peu corrélés, mais peu performants, ce qui requiert probablement un plus grand nombre d'arbres. Dans le cas d'une classification,

l'utilisation de mesures comme le score de Brier ou la fonction de perte logarithmique est recommandée pour évaluer la convergence plutôt que la précision (métrique par défaut de `ranger` et `scikit-learn`).

- Le **nombre (ou la part) de variables candidates à chaque nœud** (souvent appelé `mtry`) est un hyperparamètre essentiel qui détermine le nombre de variables prédictives sélectionnées aléatoirement à chaque nœud lors de la construction des arbres. Ce paramètre exerce la plus forte influence sur les performances du modèle, et un compromis doit être trouvé entre puissance prédictive des arbres et corrélation entre arbres. Une faible valeur de `mtry` conduit à des arbres moins performants mais plus diversifiés et donc moins corrélés entre eux. Inversement, une valeur plus élevée améliore la précision des arbres individuels mais accroît leur corrélation (les mêmes variables ayant tendance à être sélectionnées dans tous les arbres). La valeur optimale de `mtry` dépend du nombre de variables réellement pertinentes dans les données: elle est plus faible lorsque la plupart des variables sont pertinentes, et plus élevée lorsqu'il y a peu de variables pertinentes. Par ailleurs, une valeur élevée de `mtry` est préférable si les données comprennent un grand nombre de variables binaires issues du *one-hot-encoding* des variables catégorielles (LIEN AVEC LA PARTIE PREPROCESSING). Par défaut, cette valeur est fréquemment fixée à \sqrt{p} pour les problèmes de classification et à $p/3$ pour les problèmes de régression, où p représente le nombre total de variables prédictives disponibles.
- Le **taux d'échantillonnage** et le **mode de tirage** contrôlent le plan d'échantillonnage des données d'entraînement. Les valeurs par défaut varient d'une implémentation à l'autre; dans le cas de `ranger`, le taux d'échantillonnage est de 63,2% sans remise, et de 100% avec remise. L'implémentation `scikit-learn` ne propose pas le tirage sans remise. Ces hyperparamètres ont des effets sur la performance similaires à ceux du nombre de variables candidates, mais d'une moindre ampleur. Un taux d'échantillonnage plus faible aboutit à des arbres plus diversifiés et donc moins corrélés (car ils sont entraînés sur des échantillons très différents), mais ces arbres peuvent être peu performants car ils sont entraînés sur des échantillons de petite taille. Inversement, un taux d'échantillonnage élevé aboutit à des arbres plus performants mais plus corrélés. Les effets de l'échantillonnage avec ou sans remise sur la performance de la forêt aléatoire sont moins clairs et ne font pas consensus. Les travaux les plus récents semblent toutefois suggérer qu'il est préférable d'échantillonner sans remise (P. Probst, M. N. Wright, and A.-L. Boulesteix [12]).

- Le **nombre minimal d’observations dans les noeuds terminaux** contrôle la taille des noeuds terminaux. La valeur par défaut est faible dans la plupart des implémentations (entre 1 et 5). Il n’y a pas vraiment de consensus sur l’effet de cet hyperparamètre sur les performances, bien qu’une valeur plus faible augmente le risque de sur-apprentissage. En revanche, il est certain que le temps d’entraînement décroît fortement avec cet hyperparamètre: une valeur faible implique des arbres très profonds, avec un grand nombre de noeuds. Il peut donc être utile de fixer ce nombre à une valeur plus élevée pour accélérer l’entraînement, en particulier si les données sont volumineuses et si on utilise une méthode de validation croisée pour le choix des autres hyperparamètres. Cela se fait généralement sans perte significative de performance.
- Le **critère de choix de la règle de division des noeuds intermédiaires**: la plupart des implémentations des forêts aléatoires retiennent par défaut l’impureté de Gini pour la classification et la variance pour la régression, même si d’autres critères de choix ont été proposés dans la littérature (p-value dans les forêts d’inférence conditionnelle, arbres extrêmement randomisés, etc.). Chaque règle présente des avantages et des inconvénients, notamment en termes de biais de sélection des variables et de vitesse de calcul. A ce stade, aucun critère de choix ne paraît systématiquement supérieur aux autres en matière de performance. Modifier cet hyperparamètre relève d’un usage avancé des forêts aléatoires. Le lecteur intéressé pourra se référer à la discussion détaillée dans [P. Probst, M. N. Wright, and A.-L. Boulesteix \[12\]](#).

4.3. Comment entraîner une forêt aléatoire?

Les forêts aléatoires nécessitent généralement moins d’optimisation que d’autres modèles de *machine learning*, car leurs performances varient relativement peu en fonction des hyperparamètres. Les valeurs par défaut fournissent souvent des résultats satisfaisants, ce qui réduit le besoin d’optimisation intensive. Cependant, un ajustement précis des hyperparamètres peut apporter des gains de performance, notamment sur des jeux de données complexes.

Comme indiqué dans la partie [Section 3.3.3](#), la performance prédictive d’une forêt aléatoire varie en fonction de deux critères essentiels: elle croît avec le pouvoir prédictif des arbres, et décroît avec la corrélation des arbres entre eux. L’optimisation des hyperparamètres d’une forêt aléatoire vise donc à trouver un équilibre optimal où les arbres sont suffisamment puissants pour être prédictifs, tout en étant suffisamment diversifiés pour que leurs erreurs ne soient pas trop corrélées.

La littérature propose de multiples approches pour optimiser simultanément plusieurs hyperparamètres: la recherche par grille (*grid search*), la recherche aléatoire (*random search*) et

l'optimisation basée sur modèle séquentiel (SMBO), et il peut être difficile de savoir quelle approche adopter. Ce guide propose donc une première approche délibérément simple, avant de présenter les approches plus avancées.

4.3.1. Approche simple

Voici une procédure simple pour entraîner une forêt aléatoire. Elle ne garantit pas l'obtention d'un modèle optimal, mais elle est lisible et permet d'obtenir rapidement un modèle raisonnablement performant.

- **Entraîner une forêt aléatoire avec les valeurs des hyperparamètres par défaut.** Ce premier modèle servira de point de comparaison pour la suite.
- **Ajuster le nombre d'arbres:** entraîner une forêt aléatoire avec les hyperparamètres par défaut en augmentant progressivement le nombre d'arbres, puis déterminer à partir de quel nombre d'arbres la performance se stabilise (en mesurant la performance avec l'erreur OOB avec pour métrique le [score de Brier](#)). Fixer le nombre d'arbres à cette valeur par la suite.
- **Ajuster le nombre de variables candidates et le taux d'échantillonnage:** optimiser ces deux hyperparamètres grâce à une méthode de *grid search* évaluée par une approche de validation-croisée, ou par une approche reposant sur l'erreur OOB.
- **Ajuster le nombre minimal d'observations dans les noeuds terminaux:** optimiser cet hyperparamètre grâce à une méthode de *grid search* évaluée par une approche de validation-croisée, ou par une approche reposant sur l'erreur OOB. Ce n'est pas l'hyperparamètre le plus important, mais s'il est possible de le fixer à une valeur plus élevée que la valeur par défaut sans perte de performance, cela permet d'accélérer le reste de la procédure.
- **Entraîner du modèle final:** entraîner une forêt aléatoire avec les hyperparamètres optimisés déduits des étapes précédentes.
- **Évaluer du modèle final:** mesurer la performance du modèle final soit avec l'approche *out-of-bag* (OOB), soit avec un ensemble de test. Il est souvent instructif de comparer les performances du modèle final et du modèle entraîné avec les valeurs des hyperparamètres par défaut (parfois pour se rendre compte que ce dernier était déjà suffisamment performant...).

4.3.2. Approches plus avancées

Lorsque l'espace des hyperparamètres est large ou que les performances initiales sont insuffisantes, adopter des méthodes avancées comme l'optimisation basée sur un modèle séquentiel (SMBO). En R, il existe plusieurs implémentations d'appuyant sur cette méthode: `tuneRF` (limité à l'optimisation de `mtry`), `tuneRanger` (optimise simultanément `mtry`, node size, et sample size). La méthode SMBO est généralement la plus performante, mais demande un temps de calcul plus important.

Il est également possible de remplacer les critères classiques (le taux d'erreur pour une classification par exemple) par d'autres critères de performance, comme le score de Brier ou la fonction de perte logarithmique (P. Probst and A.-L. Boulesteix [13]).

Pour gérer la contrainte computationnelle, il est possible de commencer par utiliser des échantillons réduits pour les étapes exploratoires, puis d'augmenter la taille de l'échantillon pour les tests finaux.

4.4. Mesurer l'importance des variables

Les méthodes classiques d'évaluation de l'importance des variables, telles que l'indice de Gini (Mean Decrease in Impurity - MDI) et l'importance par permutation (Mean Decrease Accuracy - MDA), peuvent produire des résultats biaisés dans certaines situations (C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn [11], C. Bérnard, S. Da Veiga, and E. Scornet [9], C. Bérnard, G. Biau, S. Da Veiga, and E. Scornet [10]). Notamment, lorsque les variables prédictives sont fortement corrélées, présentent des échelles de mesure différentes ou possèdent un nombre variable de catégories, ces méthodes peuvent surestimer l'importance de certaines variables. Par exemple, les variables avec un grand nombre de catégories ou des échelles continues étendues peuvent être artificiellement privilégiées, même si leur contribution réelle à la prédiction est limitée.

En pratique, il est recommandé d'utiliser des méthodes d'importance des variables moins sensibles aux biais, comme les CIF ou la Sobol-MDA. Les valeurs de Shapley, issues de la théorie des jeux, sont également une alternative intéressante. Elles attribuent à chaque variable une contribution proportionnelle à son impact sur la prédiction. Cependant, leur calcul est souvent complexe et coûteux en ressources computationnelles, surtout en présence de nombreuses variables. Des méthodes comme SHAFF (SHApley eFFects via random Forests) ont été développées pour estimer efficacement ces valeurs, même en présence de dépendances entre variables.

On conseille l'utilisation de trois implémentations pour comparer l'importances des variables d'une forêt aléatoire:

- Pour la MDI: l'algorithme CIF proposé par C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn [11] et implémenté en R

- Pour la MDA: l'algorithme Sobol-MDA proposé par C. Bénard, S. Da Veiga, and E. Scornet [9] et implémenté en R
- Pour les valeurs de Shapley : l'algorithme SHAFF proposé par C. Bénard, G. Biau, S. Da Veiga, and E. Scornet [10] et implémenté en R

Enfin, nous recommandons de combiner plusieurs méthodes pour une analyse plus robuste et de tenir compte des prétraitements des données afin de minimiser les biais potentiels.

References

- [1] L. Grinsztajn, E. Oyallon, and G. Varoquaux, “Why do tree-based models still outperform deep learning on typical tabular data?,” *Advances in neural information processing systems*, vol. 35, pp. 507–520, 2022.
- [2] R. Schwartz-Ziv and A. Armon, “Tabular data: Deep learning is not all you need,” *Information Fusion*, vol. 81, pp. 84–90, 2022.
- [3] D. McElfresh *et al.*, “When do neural nets outperform boosted trees on tabular data?,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.
- [4] L. Breiman, J. Friedman, R. Olshen, and C. Stone, “Cart,” *Classification and regression trees*, 1984.
- [5] L. Breiman, “Bagging predictors,” *Machine learning*, vol. 24, pp. 123–140, 1996.
- [6] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [7] G. Louppe, “Understanding random forests: From theory to practice,” *arXiv preprint arXiv:1407.7502*, 2014.
- [8] G. Biau, “Analysis of a random forests model,” *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 1063–1095, 2012.
- [9] C. Bénard, S. Da Veiga, and E. Scornet, “Mean decrease accuracy for random forests: inconsistency, and a practical solution via the Sobol-MDA,” *Biometrika*, vol. 109, no. 4, pp. 881–900, 2022, doi: [10.1093/biomet/asac017](https://doi.org/10.1093/biomet/asac017).
- [10] C. Bénard, G. Biau, S. Da Veiga, and E. Scornet, “SHAFF: Fast and consistent SHAPley eEffect estimates via random Forests,” in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2022, pp. 5563–5582.
- [11] C. Strobl, A.-L. Boulesteix, A. Zeileis, and T. Hothorn, “Bias in random forest variable importance measures: Illustrations, sources and a solution,” *BMC bioinformatics*, vol. 8, pp. 1–21, 2007.
- [12] P. Probst, M. N. Wright, and A.-L. Boulesteix, “Hyperparameters and tuning strategies for random forest,” *Wiley Interdisciplinary Reviews: data mining and knowledge discovery*, vol. 9, no. 3, p. e1301, 2019.
- [13] P. Probst and A.-L. Boulesteix, “To tune or not to tune the number of trees in random forest,” *Journal of Machine Learning Research*, vol. 18, no. 181, pp. 1–18, 2018.