INTRODUCTION AUX MÉTHODES ENSEMBLISTES

1. Introduction

Une bien belle introduction

Principe: cette partie propose une présentation intuitive des méthodes ensemblistes, à destination notamment des *managers* sans bagage en *machine learning*. Elle ne contient aucune formalisation mathématique.

1.1. Que sont les méthodes ensemblistes?.

1.1.1. *L'union fait la force.*

Plutôt que de chercher à construire d'emblée un unique modèle très complexe, les approches ensemblistes vise à obtenir un modèle très performant en combinant un grand nombre de modèles simples.

Il existe quatre grandes approches ensemblistes:

- le *bagging*;
- la *random forest*;
- le *stacking*;
- le *boosting*.

Le présent document se concentre sur deux approches: la *random forest* et le *boosting*.

Les méthodes ensemblistes désignent un ensemble d'algorithmes d'apprentissage supervisé (notamment les forêts aléatoires et le *boosting*) développés depuis le début des années 2000. Ces méthodes consistent à entraîner plusieurs modèles de base, puis à combiner les résultats obtenus afin de produire une prédiction consolidée. Les modèles de base, dits "apprenants faibles" ("weak learners"), sont généralement peu complexes. Le choix de ces modèles et la manière dont leurs prédictions sont combinées sont des facteurs clés pour la performance de ces approches.

Les méthodes ensemblistes peuvent être divisées en deux grandes familles selon qu'elles s'appuient sur des modèles entrainés en parallèle ou de manière imbriquée ou séquentielle. Lorsque les modèles sont *entrainés en parallèle*, chaque modèle de base est

entraîné en utilisant soit un échantillon aléatoire des données d'entraînement, soit un sous-ensemble des variables disponibles, et le plus souvent une combinaison des deux, auquel cas on parle de forêt aléatoire. 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 *entrainés de manière séquentielle*, chaque modèle de base vise à minimiser l'erreur de prédiction de l'ensemble des modèles de base précédents. Les implémentations les plus courantes du *boosting* sont actuellement XGBoost, CatBoost et LightGBM.

1.2. **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 part, elles sont conçues pour s'appliquer à des données tabulaires (enregistrements en lignes, variables en colonnes), structure de données omniprésente dans la statistique publique. D'autre part, elles peuvent être mobilisées dans toutes les situations où on utilise une régression linéaire ou une régression logistisque (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'interpétrer partiellement les modèles ensemblistes, leur interprétabilité reste globalement 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, utiliser des méthodes ensemblistes ne requiert pas de connaissances avancées en informatique ou de 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 et le traitement d'image, 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* (Grinsztajn et al. (2022), Shwartz-Ziv & Armon (2022)), soit font jeu égal avec elles (McElfresh et al. (2024)). 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 (Shwartz-Ziv & Armon (2022)).

1.3. **Comment fonctionnent les méthodes ensemblistes?.**

Quatre temps:

- les arbres de décision et de régression (CART);
- les forêts aléatoires;
- le boosting.

1.3.1. *Le point de départ: les arbres de décision et de régression.*

Présenter *decision tree* et *regression tree*. Reprendre des éléments du chapitre 9 de https://bradleyboehmke.github.io/HOML/

Principes d'un arbre:

- fonction constante par morceaux;
- partition de l'espace;
- interactions entre variables.

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

1.3.2. *Critères de performance et sélection d'un modèle.*

La performance d'un modèle augmente généralement avec sa complexité, jusqu'à atteindre un maximum, puis diminue. L'objectif est d'obtenir un modèle qui minimise à la fois le sous-apprentissage (biais) et le sur-apprentissage (variance). C'est ce qu'on appelle le compromis biais/variance. Cette section présente très brièvement les critères utilisés pour évaluer et comparer les performances des modèles.

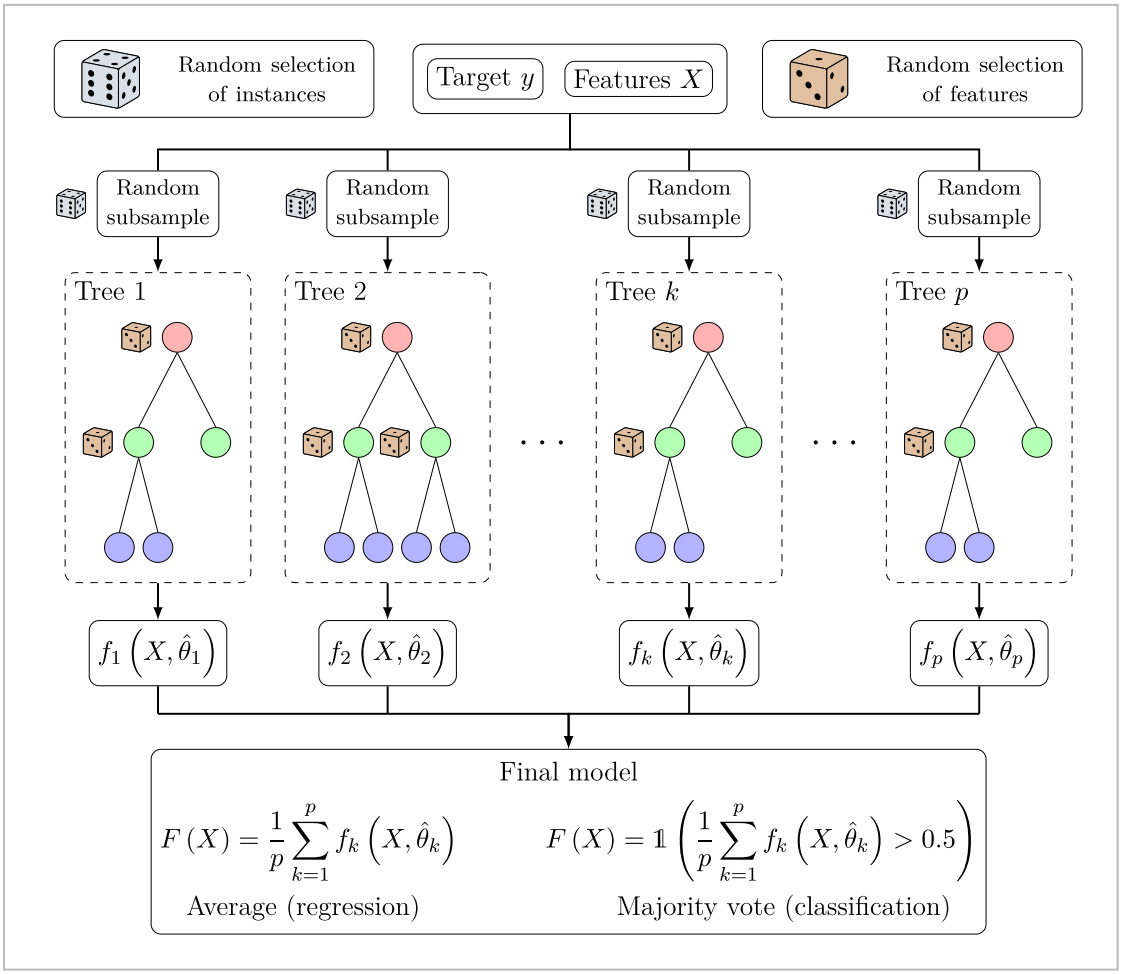

Figure 1: Représentation schématique d'un algorithme de forêt aléatoire

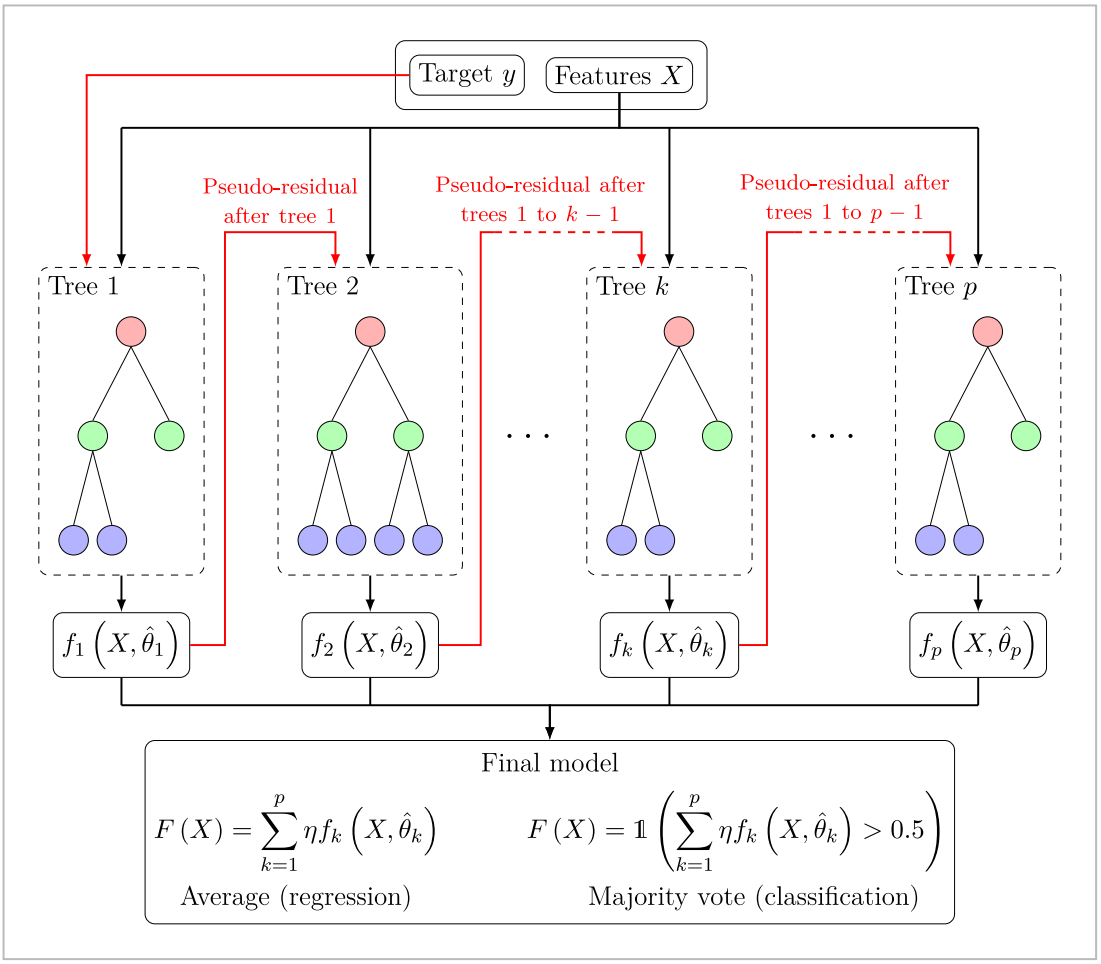

Figure 2: Représentation schématique d'un algorithme de *boosting*

1.4. Le *bagging*, les *random forests* et le *boosting*.

Il existe plusieurs types de méthodes ensemblistes, toutes ayant en commun la combinaison de modèles élémentaires. Le présent document présente les 3 principales méthodes : le Bagging, la Random Forests et le Boosting.

1.4.1. *Le* bagging *(Bootstrap Aggregating).*

Présenter le *bagging* en reprenant des éléments du chapitre 10 de https://bradleyboehmke.github.io/HOML. Mettre une description de l'algorithme en pseudo-code?

- Présentation avec la figure en SVG;
- Illustration avec un cas d'usage de classification en deux dimensions.

Le bagging, ou Bootstrap Aggregating, est une méthode ensembliste qui comporte trois étapes principales :

```
Création de sous-échantillons : 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. Cette technique permet de diversifier les données d'entraînement en créant des échantillons variés, ce qui aide à réduire la variance et à améliorer la robustesse du modèle.

Entraînement parallèle : Un modèle distinct est entraîné sur chaque sous-échantillon de manière indépendante. Cette technique permet un gain d'efficacité et un meilleur contrôle du surapprentissage (overfitting).

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. En combinant les prédictions de plusieurs modèles, le bagging renforce la stabilité et la performance globale de l'algorithme, notamment en réduisant la variance des prédictions.
```

Le bagging appliqué aux arbres de décision est la forme la plus courante de cette technique.

Le bagging est particulièrement efficace pour réduire la variance des modèles, ce qui les rend moins vulnérables au surapprentissage. Cette caractéristique est particulièrement utile dans les situations où la robustesse et la capacité de généralisation des modèles sont cruciales. De plus, comme le bagging repose sur des processus indépendants, l'exécution est plus plus rapide dans des environnements distribués.

Cependant, bien que chaque modèle de base soit construit indépendamment sur des sous-échantillons distincts, les variables utilisées pour générer ces modèles ne sont pas forcément indépendantes d'un modèle à l'autre. Dans le cas du bagging appliqué aux arbres de décision, cela conduit souvent à des arbres ayant une structure similaire.

Les forêts aléatoires apportent une amélioration à cette approche en réduisant cette corrélation entre les arbres, ce qui permet d'augmenter la précision de l'ensemble du modèle.

1.4.2. *Les* random forests.

Expliquer que les *random forests* sont une amélioration du *bagging*, en reprenant des éléments du chapitre 11 de https://bradleyboehmke.github.io/HOML/

- Présentation avec la figure en SVG;
- Difficile d'illustrer avec un exemple (car on ne peut pas vraiment représenter le *feature sampling*);
- Bien insister sur les avantages des RF: 1/ faible nombre d'hyperparamètres; 2/ faible sensibilité aux hyperparamètres; 3/ limite intrinsèque à l'overfitting.

1.4.3. *Le* boosting.

Reprendre des éléments du chapitre 12 de https://bradleyboehmke.github.io/HOML/ et des éléments de la formation boosting.

Le *boosting* combine l'**approche ensembliste** avec une **modélisation additive par étapes** (*forward stagewise additive modeling*).

- Présentation;
- Avantage du boosting: performances particulièrement élevées.
- Inconvénients: 1/ nombre élevé d'hyperparamètres; 2/ sensibilité des performances aux hyperparamètres; 3/ risque élevé d'overfitting.
- Préciser qu'il est possible d'utiliser du subsampling par lignes et colonnes pour un algoithme de boosting. Ce point est abordé plus en détail dans la partie sur les hyperparamètres.### Comparaison RF-GBDT

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 séquen-

tiellement, ce qui implique que chaque arbre n'a pas de sens indépendamment de l'ensemble des arbres qui l'ont précédé dans l'entraînement.

- Les points d'attention dans l'entraînement ne sont pas les mêmes: arbitrage puissance-corrélation dans la RF, arbitrage puissance-overfitting dans le *boosting*.
- *overfitting*: borne théorique à l'*overfitting* dans les RF, contre pas de borne dans le *boosting*. Deux conséquences: 1/ lutter contre l'overfitting est essentiel dans l'usage du *boosting*; 2/ le *boosting* est plus sensible au bruit et aux erreurs sur y que la RF.
- Conditions d'utilisation: la RF peut être utilisée en OOB, pas le *boosting*.
- Complexité d'usage: peu d'hyperparamètres dans les RF, contre un grand nombre dans le *boosting*. # Le bagging

Le bagging, ou "bootstrap aggregating", est une méthode ensembliste qui vise à améliorer la stabilité et la précision des algorithmes d'apprentissage automatique en agrégeant plusieurs modèles (Breiman (1996)). Chaque modèle est entraîné sur un échantillon distinct généré par une technique de rééchantillonnage (*bootstrap*). Ces modèles sont ensuite combinés pour produire une prédiction agrégée, souvent plus robuste et généralisable que celle obtenue par un modèle unique.

1.5. **Principe du bagging.**

Le bagging comporte trois étapes principales:

- **L'échantillonnage bootstrap** : L'échantillonnage bootstrap consiste à créer des échantillons distincts en tirant aléatoirement avec remise des observations du jeu de données initial. Chaque échantillon *bootstrap* contient le même nombre d'observations que le jeu de données initial, mais certaines observations sont répétées (car sélectionnées plusieurs fois), tandis que d'autres sont omises.
- **L'entraînement de plusieurs modèles** : Un modèle (aussi appelé *apprenant de base* ou *weak learner*) est entraîné sur chaque échantillon bootstrap. Les modèles peuvent être des arbres de décision, des régressions ou tout autre algorithme d'apprentissage. Le bagging est particulièrement efficace avec des modèles instables, tels que les arbres de décision non élagués.
- **L'agrégation des prédictions** : Les prédictions de tous les modèles sont ensuite agrégées, en procédant 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, afin d'obtenir des prédictions plus précises et généralisables.

1.6. **Pourquoi (et dans quelles situations) le bagging fonctionne.**

Certains modèles sont très sensibles aux données d'entraînement, et leurs prédictions sont très instables d'un échantillon à l'autre. L'objectif du bagging est de construire un prédicteur plus précis en agrégeant les prédictions de plusieurs modèles entraînés sur des échantillons (légèrement) différents les uns des autres.

Breiman (1996) montre que cette méthode est particulièrement efficace lorsqu'elle est appliquée à des modèles très instables, dont les performances sont particulièrement sensibles aux variations du jeu de données d'entraînement, et peu biaisés.

Cette section vise à mieux comprendre comment (et sous quelles conditions) l'agrégation par bagging permet de construire un prédicteur plus performant.

Dans la suite, nous notons $\varphi(x, L)$ un prédicteur (d'une valeur numérique dans le cas de la *régression* ou d'un label dans le cas de la *classification*), entraîné sur un ensemble d'apprentissage L, et prenant en entrée un vecteur de caractéristiques x.

1.6.1. *La régression: réduction de l'erreur quadratique moyenne par agrégation.*

Dans le contexte de la **régression**, l'objectif est de prédire une valeur numérique Y à partir d'un vecteur de caractéristiques x. Un modèle de régression $\phi(x, L)$ est construit à partir d'un ensemble d'apprentissage L, et produit une estimation de Y pour chaque observation x.

1.6.1.1. *Définition du prédicteur agrégé.*

Dans le cas de la régression, le **prédicteur agrégé** est défini comme suit :

$ _A(x) = E_L[(x, L)] $

où $\phi_A(x)$ représente la prédiction agrégée, $E_L[.]$ correspond à l'espérance prise sur tous les échantillons d'apprentissage possibles L, chacun étant tiré selon la même distribution que le jeu de données initial, et $\phi(x, L)$ correspond à la prédiction du modèle construit sur l'échantillon d'apprentissage L.

1.6.1.2. *La décomposition biais-variance.*

Pour mieux comprendre comment l'agrégation améliore la performance globale d'un modèle individuel $\phi(x, L)$, revenons à la **décomposition biais-variance** de l'erreur quadratique moyenne (il s'agit de la mesure de performance classiquement considérée dans un problème de régression):

$$E_L\left[(Y - \phi(x, L))^2\right] = \underbrace{(E_L[\phi(x, L) - Y])^2}_{\text{Biais}^2} + \underbrace{E_L\left[(\phi(x, L) - E_L[\phi(x, L)])^2\right]}_{\text{Variance}} \quad (1)$$

- Le **biais** est la différence entre la valeur observée Y que l'on souhaite prédire et la prédiction moyenne $E_L[\phi(x, L)]$. Si le modèle est sous-ajusté, le biais sera élevé.
- La **variance** est la variabilité des prédictions ($\phi(x, L)$) autour de leur moyenne ($E_L[\phi(x, L)]$). Un modèle avec une variance élevée est très sensible aux fluctuations au sein des données d'entraînement: ses prédictions varient beaucoup lorsque les données d'entraînement se modifient.

L'équation Equation 1 illustre l'**arbitrage biais-variance** qui est omniprésent en *machine learning*: plus la complexité d'un modèle s'accroît (exemple: la profondeur d'un arbre), plus son biais sera plus faible (car ses prédictions seront de plus en plus proches des données d'entraînement), et plus sa variance sera élevée (car ses prédictions, étant très proches des données d'entraînement, auront tendance à varier fortement d'un jeu d'entraînement à l'autre).

1.6.1.3. *L'inégalité de Breiman (1996).*

Breiman (1996) compare l'erreur quadratique moyenne d'un modèle individuel avec celle du modèle agrégé et démontre l'inégalité suivante :

$$(Y - \phi_A(x))^2 \leq E_L\left[(Y - \phi(x, L))^2\right] \quad (2)$$

- Le terme $(Y - \phi_A(x))^2$ représente l'erreur quadratique du **prédicteur agrégé** $\phi_A(x)$;
- Le terme $E_L\left[(Y - \phi(x, L))^2\right]$ est l'erreur quadratique moyenne d'un **prédicteur individuel** $\phi(x, L)$ entraîné sur un échantillon aléatoire L. Cette erreur varie en fonction des données d'entraînement.

Cette inégalité montre que **l'erreur quadratique moyenne du prédicteur agrégé est toujours inférieure ou égale à la moyenne des erreurs des prédicteurs individuels**. Puisque le biais du prédicteur agrégé est identique au biais du prédicteur

individuel, alors l'inégalité précédente implique que la **variance du modèle agrégé** $\phi_A(x)$ est **toujours inférieure ou égale** à la variance moyenne d'un modèle individuel :

$$\text{Var } (\phi_A(x)) = \text{Var } (E_L[\phi(x, L)]) \leq E_L[\text{Var } (\phi(x, L))]$$

Autrement dit, le processus d'agrégation réduit l'erreur de prédiction globale en réduisant la **variance** des prédictions, tout en conservant un biais constant.

Ce résultat ouvre la voie à des considérations pratiques immédiates. Lorsque le modèle individuel est instable et présente une variance élevée, l'inégalité $Var(\phi_A(x)) \leq E_L[Var(\phi(x, L))]$ est forte, ce qui signifie que l'agrégation peut améliorer significativement la performance globale du modèle. En revanche, si $\phi(x, L)$ varie peu d'un ensemble d'entraînement à un autre (modèle stable avec variance faible), alors $Var(\phi_A(x))$ est proche de $E_L[Var(\phi(x, L))]$, et la réduction de variance apportée par l'agrégation est faible. Ainsi, **le bagging est particulièrement efficace pour les modèles instables**, tels que les arbres de décision, mais moins efficace pour les modèles stables tels que les méthodes des k plus proches voisins.

1.6.2. *La classification: vers un classificateur presque optimal par agrégation.*

Dans le cas de la classification, le mécanisme de réduction de la variance par le bagging permet, sous une certaine condition, d'atteindre un **classificateur presque optimal** (*nearly optimal classifier*). Ce concept a été introduit par Breiman (1996) pour décrire un modèle qui tend à classer une observation dans la classe la plus probable, avec une performance approchant celle du classificateur Bayésien optimal (la meilleure performance théorique qu'un modèle de classification puisse atteindre).

Pour comprendre ce résutlat, introduisons $Q(j \mid x) = E_L\left(1_{\varphi(x,L)=j}\right) = P(\varphi(x, L) = j)$, la probabilité qu'un modèle $\varphi(x, L)$ prédise la classe j pour l'observation x, et $P(j \mid x)$, la probabilité réelle (conditionnelle) que x appartienne à la classe j.

1.6.2.1. *Définition : classificateur order-correct.*

Un classificateur $\varphi(x, L)$ est dit **order-correct** pour une observation x si, en espérance, il identifie **correctement la classe la plus probable**, même s'il ne prédit pas toujours avec exactitude les probabilités associées à chaque classe $Q(j \mid x)$.

Cela signifie que si l'on considérait tous les ensemble de données possibles, et que l'on évaluait les prédictions du modèle en x, la majorité des prédictions correspondraient à la classe à laquelle il a la plus grande probabilité vraie d'appartenir $P(j \mid x)$.

Formellement, un prédicteur est dit “order-correct” pour une entrée x si :

$argmax_j Q(j \mid x) = argmax_j P(j \mid x)$

où $P(j \mid x)$ est la vraie probabilité que l'observation x appartienne à la classe j, et $Q(j \mid x)$ est la probabilité que x appartienne à la classe j prédite par le modèle $\varphi(x, L)$.

Un classificateur est **order-correct** si, pour **chaque** observation x, la classe qu'il prédit correspond à celle qui a la probabilité maximale $P(j \mid x)$ dans la distribution vraie.

1.6.2.2. *Prédicteur agrégé en classification: le vote majoritaire.*

Dans le cas de la classification, le prédicteur agrégé est défini par le **vote majoritaire**. Cela signifie que si K classificateurs sont entraînés sur K échantillons distincts, la classe prédite pour x est celle qui reçoit **le plus de votes** de la part des modèles individuels.

Formellement, le classificateur agrégé $\varphi A(x)$ est défini par :

$\varphi A(x) = \text{argmax}_j \sum_L I(\phi(x, L) = j) = argmax_j Q(j \mid x)$

1.6.2.3. *Performance globale: convergence vers un classificateur presque optimal.*

Breiman (1996) montre que si chaque prédicteur individuel $\varphi(x, L)$ est order-correct pour une observation x, alors le prédicteur agrégé $\varphi A(x)$, obtenu par **vote majoritaire**, atteint la performance optimale pour cette observation, c'est-à-dire qu'il converge vers la classe ayant la probabilité maximale $P(j \mid x)$ pour l'observation x lorsque le nombre de prédicteurs individuels augmente. Le vote majoritaire permet ainsi de **réduire les erreurs aléatoires** des classificateurs individuels.

Le classificateur agrégé ϕA est optimal s'il prédit systématiquement la classe la plus probable pour l'observation x dans toutes les régions de l'espace.

Cependant, dans les régions de l'espace où les classificateurs individuels ne sont pas order-corrects (c'est-à-dire qu'ils se trompent majoritairement sur la classe d'appartenance), l'agrégation par vote majoritaire n'améliore pas les performances. Elles peuvent même se détériorer par rapport aux modèles individuels si l'agrégation conduit à amplifier des erreurs systématiques (biais).

1.7. **L'échantillage par bootstrap peut détériorer les performances théoriques du modèle agrégé.**

En pratique, au lieu d'utiliser tous les ensembles d'entraînement possibles L, le bagging repose sur un nombre limité d'échantillons bootstrap tirés avec remise à partir d'un même jeu de données initial, ce qui peut introduire des biais par rapport au prédicteur agrégé théorique.

Les échantillons bootstrap présentent les limites suivantes :

- Une **taille effective réduite par rapport au jeu de données initial**: Bien que chaque échantillon bootstrap présente le même nombre d'observations que le jeu de données initial, environ 1/3 des observations (uniques) du jeu initial sont absentes de chaque échantillon bootstrap (du fait du tirage avec remise). Cela peut limiter la capacité des modèles à capturer des relations complexes au sein des données (et aboutir à des modèles individuels sous-ajustés par rapport à ce qui serait attendu théoriquement), en particulier lorsque l'échantillon initial est de taille modeste.
- Une **dépendance entre échantillons** : Les échantillons bootstrap sont tirés dans le même jeu de données, ce qui génère une dépendance entre eux, qui réduit la diversité des modèles. Cela peut limiter l'efficacité de la réduction de variance dans le cas de la régression, voire acroître le biais dans le cas de la classification.
- Une **couverture incomplète de l'ensemble des échantillons possibles**: Les échantillons bootstrap ne couvrent pas l'ensemble des échantillons d'entraînement possibles, ce qui peut introduire un biais supplémentaire par rapport au prédicteur agrégé théorique.

1.8. **Le bagging en pratique.**

1.8.1. *Quand utiliser le bagging en pratique.*

Le bagging est particulièrement utile lorsque les modèles individuels présentent une variance élevée et sont instables. Dans de tels cas, l'agrégation des prédictions peut réduire significativement la variance globale, améliorant ainsi la performance du modèle agrégé. Les situations où le bagging est recommandé incluent typiquement:

- Les modèles instables : Les modèles tels que les arbres de décision non élagués, qui sont sensibles aux variations des données d'entraînement, bénéficient grande-

ment du bagging. L'agrégation atténue les fluctuations des prédictions dues aux différents échantillons.

- Les modèles avec biais faibles: En classification, si les modèles individuels sont order-corrects pour la majorité des observations, le bagging peut améliorer la précision en renforçant les prédictions correctes et en réduisant les erreurs aléatoires.

Inversement, le bagging peut être moins efficace ou même néfaste dans certaines situations :

- Les modèles stables avec variance faible : Si les modèles individuels sont déjà stables et présentent une faible variance (par exemple, la régression linéaire), le bagging n'apporte que peu d'amélioration, car la réduction de variance supplémentaire est minimale.
- La présence de biais élevée : Si les modèles individuels sont biaisés, entraînant des erreurs systématiques, le bagging peut amplifier ces erreurs plutôt que de les corriger. Dans de tels cas, il est préférable de s'attaquer d'abord au biais des modèles avant de considérer l'agrégation.
- Les échantillons de petite taille : Avec des ensembles de données limités, les échantillons bootstrap peuvent ne pas être suffisamment diversifiés ou représentatifs, ce qui réduit l'efficacité du bagging et peut augmenter le biais des modèles.

Ce qui qu'il faut retenir: le bagging peut améliorer substantiellement la performance des modèles d'apprentissage automatique lorsqu'il est appliqué dans des conditions appropriées. Il est essentiel d'évaluer la variance et le biais des modèles individuels, ainsi que la taille et la représentativité du jeu de données, pour déterminer si le bagging est une stratégie adaptée. Lorsqu'il est utilisé judicieusement, le bagging peut conduire à des modèles plus robustes et précis, exploitant efficacement la puissance de l'agrégation pour améliorer la performance des modèles individuels.

1.8.2. *Comment utiliser le bagging en pratique.*

1.8.2.1. *Combien de modèles agréger?.*

"Optimal performance is often found by bagging 50–500 trees. Data sets that have a few strong predictors typically require less trees; whereas data sets with lots of noise or multiple strong predictors may need more. Using too many trees will not lead to

overfitting. However, it's important to realize that since multiple models are being run, the more iterations you perform the more computational and time requirements you will have. As these demands increase, performing k-fold CV can become computationally burdensome."

1.8.2.2. *Evaluation du modèle: cross validation et échantillon Out-of-bag (OOB).*

"A benefit to creating ensembles via bagging, which is based on resampling with replacement, is that it can provide its own internal estimate of predictive performance with the out-of-bag (OOB) sample (see Section 2.4.2). The OOB sample can be used to test predictive performance and the results usually compare well compared to k-fold CV assuming your data set is sufficiently large (say n≥1,000). Consequently, as your data sets become larger and your bagging iterations increase, it is common to use the OOB error estimate as a proxy for predictive performance."

1.9. **Mise en pratique (exemple avec code).**

Ou bien ne commencer les mises en pratique qu'avec les random forest ?

1.10. **Interprétation.**

2. 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 avec la puissance de l'agrégation pour améliorer les performances prédictives et réduire le risque de surapprentissage (overfitting).

La méthode s'appuie sur la technique du bagging, qui consiste à entraîner chaque arbre sur un échantillon (*bootstrap*) tiré au hasard à partir du jeu de données initial. Elle introduit un degré supplémentaire de randomisation au moment de la construction d'un arbre, puisqu'à chaque nouvelle division (*noeud*), un sous-ensemble de variables sur lequel sera fondé le critère de séparation est **sélectionné aléatoirement**. Cette randomisation supplémentaire **réduit la corrélation** entre les arbres, ce qui permet de diminuer la variance des prédiction du modèle agrégé.

Objectifs:

- comprendre les propriétés fondamentales des forêts aléatoires afin de comprendre comment elles améliorent les performances des modèles
- comprendre les étapes permettant de construire une forêt aléatoire : échantillonnage bootstrap, sélection de variables, partitions, prédiction, évaluation, interprétation
- permettre l'optimisation des performances du modèle: fournir une présentation formelle pour guider le choix des hyperparamètres, la préparation des données, etc.

2.1. **Principe de la forêt aléatoire.**

Les forêts aléatoires reposent sur plusieurs 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 tiré avec remise du jeu de données d'entraînement.
- **La sélection aléatoire de caractéristiques (*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.

2.2. **Pourquoi (et dans quelles situations) la random forest fonctionne: fondements théoriques des forêts aléatoires.**

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

2.2.1. *Réduction de la variance par agrégation.*

L'agrégation de plusieurs arbres dans une forêt aléatoire permet de réduire la variance globale du modèle, ce qui améliore la stabilité et la précision des prédictions lorsque les biais initiaux sont "faibles". La démonstration est présentée dans la section *bagging*.

2.2.2. *Convergence et absence de surapprentissage.*

Les forêts aléatoires sont très performantes en pratique, mais la question de leur convergence vers une solution optimale lorsque la taille de l'échantillon tend vers l'infini reste ouverte (Loupes 2014). Plusieurs travaux théoriques ont toutefois fourni des preuves de consistance pour des versions simplifiées de l'algorithme (par exemple, Biau 2012).

Même si la convergence vers le modèle optimal n'est pas encore formellement établie, il est possible de montrer, grâce à la Loi des Grands Nombres, que l'erreur de généralisation de l'ensemble diminue lorsque le **nombre d'arbres** augmente. Cela implique que la forêt aléatoire ne souffre pas d'un surapprentissage (également appelé *overfitting*) croissant avec le nombre d'arbres inclus dans le modèle, ce qui la rend particulièrement robuste.

2.2.3. *Facteurs déterminants la diminution de l'erreur de généralisation.*

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

- **La force (*puissance prédictrice*) des arbres individuels** : La force d'un arbre dépend de sa capacité à faire des prédictions correctes (sans biais). Pour que l'ensemble soit performant, chaque arbre doit être suffisamment prédictif.
- **La corrélation entre les arbres** : Une corrélation faible entre les arbres améliore les performances globales. En effet, des arbres fortement corrélés auront tendance à faire des erreurs similaires. La randomisation des caractéristiques à chaque nœud contribue à réduire cette corrélation, ce qui améliore la précision globale.

Dans le cas de la régression, où l'objectif est de minimiser l'erreur quadratique moyenne, la décomposition de la variance de l'ensemble (la forêt aléatoire) permet de faire apparaître le rôle de la corrélation entre les arbres:

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

Où $\rho(x)$ est le coefficient de corrélation entre les arbres individuels, $\sigma(x)^2$ est la variance d'un arbre individuel, M est le nombre d'arbres dans la forêt.

Conséquences:

- **Si $\rho(x)$ est faible** : La variance est significativement réduite avec l'augmentation du nombre d'arbres M.
- **Si $\rho(x)$ est élevée** : La réduction de variance est moindre, car les arbres sont plus corrélés entre eux.

L'objectif 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.

2.3. **Les hyper-paramètres clés.**

- **Nombre d'arbres** : plus le nombre d'arbes est élevé, plus la variance est réduite, jusqu'à un certain point de saturation. Souvent, quelques centaines d'arbres suffisent à stabiliser les performances des modèles.
- **Nombre de variables à sélectionner à chaque noeud**: cet hyperparamètre permet de contrôler l'interdépendance entre les arbres. Avec K le nombre total de variables, une règle empirique usuelle est de considérer $\sqrt{K}$ pour une classification et $K/3$ pour une régression.
- **Profondeur des arbres** : laisser les arbres se développer pleinement (sans élagage) pour profiter de la réduction de variance par agrégation.

2.4. **Evaluation des performances du modèle et choix des hyper-paramètres.**

2.4.1. *Estimation de l'erreur Out-of-Bag (OOB).*

L'estimation **Out-of-Bag (OOB)** est une méthode particulièrement efficace pour évaluer les performances des forêts aléatoires sans nécessité une **validation croisée** ou de réserver une partie des données pour l'étape du test. Cette technique repose sur le fait que chaque arbre dans une forêt aléatoire est construit à partir d'un échantillon bootstrap du jeu de données d'origine, c'est-à-dire un échantillon tiré avec remise. Or, en moyenne, environ **36 %** des observations ne sont pas inclus dans chaque échantillon bootstrap, ce qui signifie qu'elles ne sont pas utilisées pour entraîner l'arbre correspondant. Ces observations laissées de côté forment un **échantillon out-of-bag**. Chaque

arbre peut donc être évalué sur son **échantillon out-of-bag** plutôt que sur un échantillon test.

Procédure d'Estimation OOB:

1. **Construction des arbres** : Chaque arbre de la forêt est construit à partir d'un échantillon bootstrap tiré avec remise à partir du jeu de données d'origine. Cela signifie que certaines observations seront sélectionnées plusieurs fois, tandis que d'autres ne seront pas sélectionnées du tout.
2. **Prédiction OOB** : Pour chaque observation (x_i, y_i) qui n'a pas été inclus dans l'échantillon bootstrap qui a permi de construire un arbre donné, l'arbre est utilisé pour prédire la valeur de y_i. Ainsi, chaque observation est prédite par tous les arbres pour lesquels elle fait partie de l'échantillon out-of-bag.
3. **Agrégation des prédictions** : La prédiction finale pour chaque échantillon out-of-bag est obtenue en moyennant les prédictions de tous les arbres pour lesquels cet échantillon était OOB (pour la régression) ou par un vote majoritaire (pour la classification).
4. **Calcul de l'erreur OOB** : L'erreur OOB est ensuite calculée en comparant les prédictions agrégées avec les valeurs réelles des observations y_i. Cette erreur est une bonne approximation de l'erreur de généralisation du modèle.

Avantages de l'Estimation OOB:

- **Pas besoin de jeu de validation séparé** : L'un des principaux avantages de l'estimation 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.
- **Estimation directe et éfficace** : Contrairement à la validation croisée qui peut être coûteuse en temps de calcul, l'estimation OOB est disponible "gratuitement" pendant la construction des arbres. Cela permet d'évaluer la performance du modèle sans avoir besoin de réentraîner plusieurs fois le modèle.
- **Approximation de l'erreur de généralisation** : L'erreur OOB est considérée comme une bonne approximation de l'erreur de généralisation, comparable à celle obtenue par une validation croisée 10-fold.

2.4.2. *Estimation de l'erreur par cross-validation.*

La validation croisée est une technique d'évaluation couramment utilisée en apprentissage automatique pour estimer la capacité d'un modèle à généraliser à de nouvelles données. Bien que l'estimation Out-of-Bag (OOB) soit généralement suffisante pour les forêts aléatoires, la validation croisée permet d'obtenir une évaluation plus robuste, notamment sur des jeux de données de petite taille.

L'idée derrière la validation croisée est de maximiser l'utilisation des données disponibles en réutilisant chaque observation à la fois pour l'entraînement et pour le test. Cela permet d'obtenir une estimation de la performance du modèle qui est moins sensible aux fluctuations dues à la division des données en ensembles d'entraînement et de test. La validation croisée répète cette division plusieurs fois, puis moyenne les résultats pour obtenir une estimation plus fiable.

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.

2.4.3. *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 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.

2.5. **Interprétation et importance des variables.**

Objectif: Identifier les variables qui contribuent le plus à la prédiction/influencent le plus la variable cible, extraire des caractéristiques pertinentes pour comprendre les mécanismes de prédiction sous-jacent, établir des règles de décision simplifiées etc.

Méthodes ususelles (mais biaisées):

- **Réduction moyenne de l'impureté** (*Mean Decrease in Impurity*) : pour chaque variable, on calcule 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.
- **Permutation importance** (*Mean Decrease Accuracy*) : 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.

Il est essentiel de noter que ces deux mesures peuvent présenter des **biais** importants. Elles sont notamment sensibles aux variables catégorielles avec de nombreuses modalités, qui peuvent apparaître artificiellement importantes. Elles sont également fortement biaisée en présence de variables explicatives corrélées ou d'interactions complexes avec la cible.

2.6. **Préparation des données (Feature Engineering).**

- **Variables catégorielles** : Encoder correctement (one-hot encoding, ordinal encoding).
- **Valeurs manquantes** : Les forêts aléatoires peuvent gérer les données manquantes, mais une imputation préalable peut améliorer les performances.
- **Échelle des Variables** : Pas nécessaire de normaliser, les arbres sont invariants aux transformations monotones.

2.7. **Le *boosting*.**

2.7.1. *Introduction.*

Le fondement théorique du *boosting* est un article de de 1990 (Shapire (1990)) qui a démontré théoriquement que, sous certaines conditions, il est possible de transformer un modèle prédictif peu performant en un modèle prédictif très performant. Plus précisément, un modèle ayant un pouvoir prédictif arbitrairement élevé (appelé *strong learner*) peut être construit en combinant des modèles simples dont les prédictions ne sont que légèrement meilleures que le hasard (appelé *weak learners*). Le *boosting* est donc une méthode qui combine une approche ensembliste reposant sur un grand nombre de modèles simples avec un entraînement séquentiel: chaque modèle simple (souvent des arbres de décision peu profonds) tâche d'améliorer la prédiction globale en corrigeant les erreurs des prédictions précédentes à chaque étape. Bien qu'une approche de *boosting* puisse en théorie mobiliser différentes classes de *weak learners*, en pratique les *weak learners* utilisés par les algorithmes de *boosting* sont presque toujours des arbres de décision.

S'il existe plusieurs variantes, les algorithmes de *boosting* suivent la même logique :

- 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.

En termes plus techniques, les algorithmes de *boosting* partagent trois caractéristiques communes:

- Ils visent à **trouver une approximation** $\hat{F}$“)`{=typst} d'une fonction inconnue $F^* : \mathbf{x} \mapsto y$ à partir d'un ensemble d'entraînement $(y_i, \mathbf{x_i})_{i=1,\dots,n}$;
- Ils supposent que la fonction F^* peut être approchée par une **somme pondérée de modèles simples** f de paramètres θ:

$$F(\mathbf{x}) = \sum_{m=1}^{M} \beta_m f(\mathbf{x}, \boldsymbol{\theta}_m)$$

- ils reposent sur une **modélisation additive par étapes**, qui décompose l'entraînement de ce modèle complexe en une **séquence d'entraînements de petits modèles**. Chaque étape de l'entraînement cherche le modèle simple f qui améliore la puissance prédictive du modèle complet, sans modifier les modèles précédents, puis l'ajoute de façon incrémentale à ces derniers:

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \hat{\beta}_m f\left(\mathbf{x}_i, \hat{\boldsymbol{\theta}}_{\mathbf{m}}\right)$$

METTRE ICI UNE FIGURE EN UNE DIMENSION, avec des points et des modèles en escalier qui s'affinent.

2.7.2. *Les premières approches du* boosting.

2.7.2.1. *Le* boosting *par repondération: Adaboost.*

Dans les années 1990, de nombreux travaux ont tâché de proposer des mise en application du *boosting* (Breiman (1998), Grove & Schuurmans (1998)) et ont comparé les mérites des différentes approches. Deux approches ressortent particulièrement de cette littérature: Adaboost (Adaptive Boosting, Freund & Schapire (1997)) et la *Gradient Boosting Machine* (Friedman (2001)). Ces deux approches reposent sur des principes très différents.

Le principe d'Adaboost consiste à pondérer les erreurs commises à chaque itération en donnant plus d'importance aux observations mal prédites, de façon à obliger les modèles

simples à se concentrer sur les observations les plus difficiles à prédire. Voici une esquisse du fonctionnement d'AdaBoost:

- Un premier modèle simple est entraîné sur un jeu d'entraînement dans lequel toutes les observations ont le même poids.
- A l'issue de cette première itération, les observations mal prédites reçoivent une pondération plus élevé que les observations bien prédites, et un deuxième modèle est entraîné sur ce jeu d'entraînement pondéré.
- Ce deuxième modèle est ajouté au premier, puis on repondère à nouveau les observations en fonction de la qualité de prédiction de ce nouveau modèle.
- Cette procédure est répétée en ajoutant de nouveaux modèles et en ajustant les pondérations.

L'algorithme Adaboost a été au coeur de la littérature sur le *boosting* à la fin des années 1990 et dans les années 2000, en raison de ses performances sur les problèmes de classification binaire. Il a toutefois été progressivement remplacé par les algorithmes de *gradient boosting* inventé quelques années plus tard.

2.7.2.2. *L'invention du* boosting boosting *: la* Gradient Boosting Machine.

La *Gradient Boosting Machine* (GBM) propose une approche assez différente: elle introduit le *gradient boosting* en reformulant le *boosting* sous la forme d'un problème de descente de gradient. Voici une esquisse du fonctionnement de la *Gradient Boosting Machine*:

- Un premier modèle simple est entraîné sur un jeu d'entraînement, de façon à minimiser une fonction de perte qui mesure l'écart entre la variable à prédire et la prédiction du modèle.
- A l'issue de cette première itération, on calcule la dérivée partielle (*gradient*) de la fonction de perte par rapport à la prédiction en chaque point de l'ensemble d'entraînement. Ce gradient indique dans quelle direction et dans quelle ampleur la prédiction devrait être modifiée afin de réduire la perte.
- A la deuxième itération, on ajoute un deuxième modèle qui va tâcher d'améliorer le modèle complet en prédisant le mieux possible l'opposé de ce gradient.
- Ce deuxième modèle est ajouté au premier, puis on recalcule la dérivée partielle de la fonction de perte par rapport à la prédiction de ce nouveau modèle.
- Cette procédure est répétée en ajoutant de nouveaux modèles et en recalculant le gradient à chaque étape.

L'approche de *gradient boosting* proposée par Friedman (2001) présente deux grands avantages. D'une part, elle peut être utilisée avec n'importe quelle fonction de perte différentiable, ce qui permet d'appliquer le gradient boosting à de multiples problèmes (régression, classification binaire ou multiclasse, *learning-to-rank*...). D'autre part, elle offre souvent des performances comparables ou supérieures aux autres approches de *boosting*. Le *gradient boosting* d'arbres de décision (*Gradient boosted Decision Trees* - GBDT) est donc devenue l'approche de référence en matière de *boosting*. En particulier, les implémentations modernes du *gradient boosting* comme XGBoost, LightGBM, et CatBoost sont des extensions et améliorations de la *Gradient Boosting Machine*.

2.8. **La mécanique du *gradient boosting*.**

Depuis la publication de Friedman (2001), la méthode de *gradient boosting* a connu de multiples développements et raffinements, parmi lesquels XGBoost (Chen & Guestrin (2016)), LightGBM (Ke et al. (2017)) et CatBoost (Prokhorenkova et al. (2018)). S'il existe quelques différences entre ces implémentations, elles partagent néanmoins la même mécanique d'ensemble, que la section qui suit va présenter en détail en s'appuyant sur l'implémentation proposée par XBGoost.Chen & Guestrin (2016, .)

Choses importantes à mettre en avant:

- Le boosting est fondamentalement différent des forêts aléatoires. See ESL, chapitre 10.
- Toute la mécanique est indépendante de la fonction de perte choisie. En particulier, elle est applicable indifféremment à des problèmes de classification et de régression.
- Les poids sont calculés par une formule explicite, ce qui rend les calculs extrêmement rapides.
- Comment on interprète le gradient et la hessienne: cas avec une fonction de perte quadratique.
- Le boosting est fait pour overfitter; contrairement aux RF, il n'y a pas de limite à l'overfitting. Donc lutter contre le surapprentissage est un élément particulièrement important de l'usage des algorithmes de boosting.

2.8.1. *Le modèle à entraîner.*

On veut entraîner un modèle comprenant K arbres de régression ou de classification:

$$\hat{y}_i = \phi(\mathbf{x}_i) = \sum_{k=1}^{K} f_k(\mathbf{x}_i)$$

Chaque arbre f est défini par trois paramètres:

- sa structure qui est une fonction $q : \mathbb{R}^m \to \{1, ..., T\}$ qui à un vecteur d'inputs $\mathbf{x}$ de dimension m associe une feuille terminale de l'arbre);
- son nombre de feuilles terminales T;
- les valeurs figurant sur ses feuilles terminales $\mathbf{w} \in \mathbb{R}^T$ (appelées poids ou *weights*).

Le modèle est entraîné avec une **fonction-objectif** constituée d'une **fonction de perte** l et d'une **fonction de régularisation** Ω. La fonction de perte mesure la distance entre la prédiction $hat(y)$ et la vraie valeur y et présente généralement les propriétés suivantes: elle est convexe et dérivable deux fois, et atteint son minimum lorsque $hat(y) = y$. La fonction de régularisation pénalise la complexité du modèle. Dans le cas présent, elle pénalise les arbres avec un grand nombre de feuilles (T élevé) et les arbres avec des poids élevés (w_t élevés en valeur absolue).

$$\mathcal{L}(\phi) = \underbrace{\sum_i l(\hat{y}_i, y_i)}_{\substack{\text{Perte sur les} \\ \text{observations}}} + \underbrace{\sum_k \Omega(f_k)}_{\substack{\text{Fonction de} \\ \text{régularisation}}} \text{ avec } \Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{t=1}^{T} w_j^2 \tag{3}$$

2.8.2. *Isoler le k-ième arbre.*

La fonction-objectif introduite précédemment est très complexe et ne peut être utilisée directement pour entraîner le modèle, car il faudrait entraîner tous les arbres en même temps. On va donc reformuler donc cette fonction objectif de façon à isoler le k-ième arbre, qui pourra ensuite être entraîné seul, une fois que les $k-1$ arbres précédents auront été entraînés. Pour cela, on note $\hat{y}_i^{(k)}$ la prédiction à l'issue de l'étape k: $\hat{y}_i^{(k)} = \sum_{j=1}^{t} f_j(\mathbf{x}_i)$, et on \$1 la fonction-objectif $\mathcal{L}^{(k)}$ au moment de l'entraînement du k-ième arbre:

$$\begin{aligned}
\mathcal{L}^{(t)} &= \sum_{i=1}^{n} l\Big(y_i, \hat{y}_i^{(t)}\Big) + \sum_{k=1}^{t} \Omega(f_k) \\
&= \sum_{i=1}^{n} l\Big(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)\Big) + \Omega(f_t) + constant
\end{aligned}$$

2.8.3. *Faire apparaître le gradient.*

Une fois isolé le k-ième arbre, on fait un développement limité d'ordre 2 de $l\Big(y_i, \hat{y}_i^{(k-1)} + f_k(\mathbf{x}_i)\Big)$ au voisinage de $\hat{y}_i^{(k-1)}$, en considérant que la prédiction du k-ième arbre $f_k(\mathbf{x}_i)$ est

$$\mathcal{L}^{(t)} \approx \sum_{i=1}^{n}\left[l\Big(y_i, \hat{y}_i^{(t-1)}\Big) + g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)\right] + \Omega(f_t)$$

avec

$$g_i = \frac{\partial l\Big(y_i, \hat{y}_i^{(t-1)}\Big)}{\partial \hat{y}_i^{(t-1)}} \text{ et } h_i = \frac{\partial^2 l\Big(y_i, \hat{y}_i^{(t-1)}\Big)}{\partial \hat{y}_i^{((t-1))^2}}$$

Les termes g_i et h_i désignent respectivement la dérivée première (le gradient) et la dérivée seconde (la hessienne) de la fonction de perte par rapport à la variable prédite. Il est important de noter que les termes (A) et (B) sont constants car les $k-1$ arbres précédents ont déjà été entraînés et ne sont pas modifiés par l'entraînement du k-ième arbre. On peut donc retirer ces termes pour obtenir la fonction-objectif simplifiée qui sera utilisée pour l'entraînement du k-ième arbre.

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}\left[g_i f_t(\mathbf{x}_i) + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)\right] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_i^2$$

Cette expression est importante car elle montre qu'on est passé d'un problème complexe où il fallait entraîner un grand nombre d'arbres simultanément (équation Equation 3) à un problème beaucoup plus simple dans lequel il n'y a qu'un seul arbre à entraîner.

2.8.4. *Calculer les poids optimaux.*

A partir de l'expression précédente, il est possible de faire apparaître les poids w_j du k-ième arbre. Pour une structure d'arbre donnée ($q : \mathbb{R}^m \to \{1, ..., T\}$), on définit $I_j = \{i \mid q(\mathbf{x}_i) = j\}$ l'ensemble des observations situées sur la feuille j puis on réorganise $\tilde{\mathcal{L}}^{(k)}$:

$$\begin{aligned}
\tilde{\mathcal{L}}^{(t)} &= \sum_{j=1}^{T}\sum_{i\in I_j}\Big[\, g_i f_t(\mathbf{x}_i) \qquad\qquad + \frac{1}{2}h_i f_t^2(\mathbf{x}_i)\,\Big] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_i^2 \\
&= \sum_{j=1}^{T}\sum_{i\in I_j}\Big[\, g_i w_j \qquad\qquad + \frac{1}{2}h_i w_j^2\,\Big] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_i^2 \\
&= \sum_{j=1}^{T}\Big[\, w_j\sum_{i\in I_j} g_i \qquad + \frac{1}{2}w_j^2\sum_{i\in I_j} h_i + \lambda\,\Big] + \gamma T
\end{aligned}$$

Dans la dernière expression, on voit que la fonction de perte simplifiée se reformule comme une combinaison quadratique des poids w_j, dans laquelle les dérivées première et

seconde de la fonction de perte interviennent sous forme de pondérations. Tout l'enjeu de l'entraînement devient donc de trouver les poids optimaux w_j qui minimiseront cette fonction de perte, compte tenu de ces opérations.

Il se trouve que le calculs de ces poids optimaux est très simple: pour une structure d'arbre donnée ($q : \mathbb{R}^m \to \{1, ..., T\}$), le poids optimal w_j^{} de la feuille j est donné par l'équation:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda}$$

Par conséquent, la valeur optimale de la fonction objectif pour l'arbre q est égale à

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i + \lambda} + \gamma T \tag{4}$$

Cette équation est utile car elle permet de comparer simplement la qualité de deux arbres, et de déterminer lequel est le meilleur.

2.8.5. *Construire le k-ième arbre.*

Dans la mesure où elle permet de comparer des arbres, on pourrait penser que l'équation Equation 4 est suffisante pour choisir directement le k-ième arbre: il suffirait d'énumérer les arbres possibles, de calculer la qualité de chacun d'entre eux, et de retenir le meilleur. Bien que cette approche soit possible théoriquement, elle est inemployable en pratique car le nombre d'arbres possibles est extrêmement élevé. Par conséquent, le k-ième arbre n'est pas défini en une fois, mais construit de façon gloutonne:

REFERENCE A LA PARTIE CART/RF?

- on commence par le noeud racine et on cherche le *split* qui réduit au maximum la perte en séparant les données d'entraînement entre les deux noeuds-enfants.
- pour chaque noeud enfant, on cherche le *split* qui réduit au maximum la perte en séparant en deux la population de chacun de ces noeuds.
- Cette procédure recommence jusqu'à que l'arbre ait atteint sa taille maximale (définie par une combinaison d'hyperparamètres d$1s dans la partie **référence à ajouter**).

2.8.6. *Choisir les* splits.

Traduire split par critère de partition?

Reste à comprendre comment le critère de partition optimal est choisi à chaque étape de la construction de l'arbre. Imaginons qu'on envisage de décomposer la feuille I en deux nouvelles feuilles I_L et I_R (avec $I = I_L \cup I_R$), selon une condition logique reposant sur une variable et une valeur de cette variable (exemple: $x_6 > 11$). Par application de l'équation Equation 4, le gain potentiel induit par ce critère de partition est égal à:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma \tag{5}$$

Cette dernière équation est au coeur de la mécanique du *gradient boosting* car elle permet de comparer les critères de partition possibles. Plus précisément, l'algorithme de détermination des critère de partition (*split finding algorithm*) consiste en une double boucle sur les variables et les valeurs prises par ces variables, qui énumère un grand nombre de critères de partition et mesure le gain associé à chacun d'entre eux avec l'équation Equation 5. Le critère de partition retenu est simplement celui dont le gain est le plus élevé.

L'algorithme qui détermine les critère de partition est un enjeu de performance essentiel dans le *gradient boosting*. En effet, utiliser l'algorithme le plus simple (énumérer tous les critères de partition possibles, en balayant toutes les valeurs de toutes les variables) s'avère très coûteux dès lors que les données contiennent soit un grand nombre de variables, soit des variables continues prenant un grand nombre de valeurs. C'est pourquoi les algorithmes de détermination des critère de partition ont fait l'objet de multiples améliorations et optimisations visant à réduire leur coût computationnel sans dégrader la qualité des critères de partition.

2.8.7. *La suite.*

2.8.7.1. *Les moyens de lutter contre l'*overfitting*:.*

- le *shrinkage*;
- le subsampling des lignes et des colonnes;
- les différentes pénalisations.

2.8.7.2. *Les hyperparamètres.*

Hyperparamètre	Description	Valeur par défaut
`booster`	Le type de *weak learner* utilisé	`'gbtree'`
`learning_rate`	Le taux d'apprentissage	0.3
`max_depth`	La profondeur maximale des arbres	6
`max_leaves`	Le nombre maximal de feuilles des arbres	0
`min_child_weight`	Le poids minimal qu'une feuille doit contenir	1
`n_estimators`	Le nombre d'arbres	100
`lambda` ou `reg_lambda`	La pénalisation L2	1
`alpha` ou `reg_alpha`	La pénalisation L1	0
`gamma`	Le gain minimal nécessaire pour ajouter un noeud supplémentaire	0
`tree_method`	La méthode utilisée pour rechercher les splits	`'hist'`
`max_bin`	Le nombre utilisés pour discrétiser les variables continues	0
`subsample`	Le taux d'échantillonnage des données d'entraînement	1
`sampling_method`	La méthode utilisée pour échantillonner les données d'entraînement	`'uniform'`
`colsample_bytree` `colsample_bylevel` `colsample_bynode`	Taux d'échantillonnage des colonnes par arbre, par niveau et par noeud	1, 1 et 1
`scale_pos_weight`	Le poids des observations de la classe positive (classification uniquement)	1
`sample_weight`	La pondération des données d'entraînement	1
`enable_categorical`	Activer le support des variables catégorielles	`False`
`max_cat_to_onehot`	Nombre de modalités en-deça duquel XGBoost utilise le *one-hot-encoding*	A COMPLETER
`max_cat_threshold`	Nombre maximal de catégories considérées dans le partitionnement optimal des variables catégorielles	A COMPLETER

Table 1: Les principaux hyperparamètres d'XGBoost

2.8.7.3. *La préparation des données.*

- les variables catégorielles:
    - ordonnées: passer en integer;
    - non-ordonnées: OHE ou approche de Fisher.
- les variables continues:
    - inutile de faire des transformations monotones.
    - Utile d'ajouter des transformations non monotones.

2.8.7.4. *Les fonctions de perte.*

2.8.8. *Liste des hyperparamètres d'une RF.*

Source: Probst et al. (2019)

- structure of each individual tree:
    - dudu
    - dudu
    - dudu
- structure and size of the forest:
- The level of randomness (je dirais plutôt :)

References

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*, 123–140.

Breiman, L. (1998). Rejoinder: arcing classifiers. *The Annals of Statistics*, *26*(3), 841–849.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 785–794.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*(1), 119–139.

Friedman, J. H. (2001). Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 1189–1232.

Grinsztajn, L., Oyallon, E., & Varoquaux, G. (2022). Why do tree-based models still outperform deep learning on typical tabular data?. *Advances in Neural Information Processing Systems*, *35*, 507–520.

Grove, A. J., & Schuurmans, D. (1998). Boosting in the limit: Maximizing the margin of learned ensembles. *AAAI/IAAI*, 692–699.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, *30*.

McElfresh, D., Khandagale, S., Valverde, J., Prasad C, V., Ramakrishnan, G., Goldblum, M., & White, C. (2024). When do neural nets outperform boosted trees on tabular data?. *Advances in Neural Information Processing Systems*, *36*.

Probst, P., Wright, M. N., & Boulesteix, A.-L. (2019). Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *9*(3), e1301.

Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, *31*.

Shapire, R. (1990). The strength of weak learning. *Machine Learning*, *5*(2).

Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, *81*, 84–90.

Email address:
URL: