

**Mémoire présenté le :
Pour l'obtention du diplôme de Statisticien,
Mention Data Science**

Madame / Monsieur : Ibrahima DIALLO

Titre du mémoire :

Gradient Boosting Machine (GBM)

Directeur de mémoire en entreprise :

Nom :

Entreprise :

Signature :

Responsable de la filière

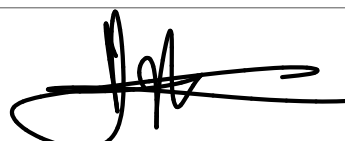
Nom : Olivier Wintenberger

Signature :

Signature du responsable entreprise



Signature du candidat



Sommaire

1	Introduction	2
2	Gradient Boosting	2
2.1	Idée de base : AdaBoost	3
2.2	Apprenant de base ou "weak learner"	3
2.3	Méthodologie	4
2.4	TreeBoost	6
2.5	Régularisation : Shrinkage et Subsampling	8
3	Simulations et Application	8
3.1	Analyse du package "gbm"	9
3.2	Fonctions de perte	10
3.3	Gradient Boosting Stochastique	12
4	Cas pratique sur des données réelles	12
4.1	Exploration et prétraitement des données	13
4.2	Construction, évaluation et analyses des modèles	13
4.3	Optimisation du modèle GBM et comparaison avec le modèle XGBoost	15
5	Conclusion	15

1 Introduction

En apprentissage automatique, les approches traditionnelles reposent souvent sur un seul modèle prédictif, tels que la régression linéaire, la régression logistique, ou les machines à vecteurs de support. Ces modèles, bien qu'efficaces dans certaines situations, peuvent manquer de robustesse et de flexibilité face à des problèmes complexes et des données variées. En réponse à ces limitations, les approches ensemblistes telles que le Bagging (Bootstrap Aggregating), les forêts aléatoires et le Boosting ont gagné du terrain. Le Bagging et les forêts aléatoires reposent sur l'idée de construire un ensemble de modèles, où chaque modèle prédit un résultat individuel. Ces résultats sont ensuite agrégés par une moyenne pondérée. Contrairement à ces méthodes, le Boosting, quant à lui, repose sur une approche séquentielle et adaptative. Il consiste à combiner plusieurs modèles faibles appelés "weak learners" (généralement des arbres de décision), pour obtenir un modèle avec un pouvoir de prédiction puissant. A chaque itération, le nouvel estimateur favorise son apprentissage sur les erreurs du précédent, ce qui permet de réduire l'erreur de biais et de variance. Le boosting est proposé à l'origine pour résoudre des problèmes de classification ensuite adapté à la régression. L'algorithme AdaBoost, développé par FREUND et SCHAPIRE [3] en 1996, a largement contribué à la reconnaissance de cette méthode.

Dans la continuité de ces avancées, FRIEDMAN [1] a proposé une amélioration majeure avec l'introduction du gradient boosting. Une version spécifique de Boosting qui utilise la descente de gradient pour minimiser la fonction de perte. En d'autres termes, chaque nouveau modèle est construit pour réduire les erreurs résiduelles du modèle combiné précédent en suivant la direction du gradient de la fonction de perte. Friedman établit ainsi une connexion entre les expansions additives par étapes et la minimisation par descente de gradient. L'innovation majeure de Friedman réside dans l'application de la descente de gradient dans l'espace des fonctions, plutôt que dans l'espace des paramètres. La démarche à Friedman optimise en ajoutant itérativement des fonctions pour minimiser une fonction de perte donnée. Cette approche permet d'adapter la méthode à une large variété de fonctions de perte, couvrant à la fois les problèmes de régression et de classification.

Pour restituer notre compréhension de l'article de Friedman, nous allons en premier temps introduire l'algorithme de Gradient Boosting structuré autour de trois notions clés que sont la fonction de perte, l'apprenant faible et le modèle additif. Dans un deuxième temps, nous allons introduire le TreeBoost puis, à l'aide de l'article de G. RIDGEWAY [5], nous nous intéresserons à l'implémentation des GBM et la version stochastique de l'algorithme de Gradient Boosting. Enfin, pour illustrer notre travail nous appliquerons ces algorithmes sur des données réelles pour évaluer leurs performances.

Vous retrouverez ce mémoire ainsi que le notebook R qui nous sert d'application pratique sur cette [page Github](#).

2 Gradient Boosting

Dans cette sous-section, nous présentons la méthodologie fondamentale et les algorithmes d'apprentissage du Gradient Boosting Machine (GBM). Ce tutoriel sert d'introduction au GBM, et par conséquent, nous n'aborderons pas forcément les démonstrations mathématiques rigoureuses de l'algorithme et ses propriétés dans cette partie.

2.1 Idée de base : AdaBoost

AdaBoost, ou Adaptive Boosting, est une méthode de classification binaire de base, proposée par Freund et Schapire en 1996 [3]. Il est considéré comme le premier algorithme de boosting. Cette méthode permet de combiner plusieurs classificateurs pour améliorer la précision prédictive. L'algorithme AdaBoost ajuste un classificateur faible aux versions pondérées des données de manière itérative. À chaque itération, les données sont pondérées à nouveau de sorte à donner un poids plus important aux points mal classés. Le modèle qui en résulte s'écrit comme suit :

$$\hat{f}_M(x) = \sum_{i=1}^M \theta_i c_i(x) \quad (1)$$

où M est le nombre de classificateurs final, $c_i(x) \in \{-1, 1\}$ est le i -ème classificateur et θ_i est la i -ème pondération. L'algorithme AdaBoost se présente comme suit :

Algorithm 1 AdaBoost ou (adaptive boosting)

- 1: **Soit** x_0 ;
- 2: **Soit** $z = \{(x_1, y_1), \dots, (x_n, y_n)\}$ un échantillon;
- 3: **Initialiser** les poids $w = \{w_i = \frac{1}{n}; i = 1, \dots, n\}$;
- 4: **for** $i = 1$ to M **do**
- 5: Estimer c_i sur l'échantillon pondéré par w ;
- 6: Calculer le taux d'erreur apparent :

$$\hat{\epsilon}_p = \frac{\sum_{k=1}^n w_k \mathbb{1}\{c_i(x_k) \neq y_k\}}{\sum_{k=1}^n w_k}$$

- 7: Calculer les logits $\theta_i = \log\left(\frac{1-\hat{\epsilon}_p}{\hat{\epsilon}_p}\right)$;
- 8: Calculer les nouvelles pondérations :

$$w_k \leftarrow w_k \cdot \exp[\theta_i \mathbb{1}\{c_i(x_k) \neq y_k\}; k = 1, \dots, n]$$

- 9: **end for**
- 10: **Résultat du vote :**

$$\hat{f}_M(x_0) = \text{sign}\left(\sum_{i=1}^M \hat{\theta}_i \hat{c}_i(x_0)\right)$$

Dans cet algorithme, le paramètre M est le paramètre à optimiser. Par exemple, par validation croisée pour éviter le surajustement.

2.2 Apprenant de base ou "weak learner"

Considérons un espace d'entrée X et un espace de sortie Y . Soit $f : X \rightarrow Y$ la fonction de classification qui attribue des classes à partir des observations $x \in X$. On dispose d'un ensemble de données d'apprentissage composé de paires $\{(x_i, y_i)\}_{i=1}^N$, où $x_i \in X$ et $y_i \in Y$. Le taux d'erreur d'un modèle f est défini comme suit :

$$\epsilon(f) = \frac{1}{N} \sum_{i=1}^N \mathbb{1}\{f(x_i) \neq y_i\} \quad (2)$$

où $\mathbb{1}\{\cdot\}$ est une fonction vaut 1 si la condition est vraie et 0 sinon.

Un *weak learner* est donc un modèle de classification dont le taux d'erreur est légèrement inférieur à celui d'une prédiction aléatoire. Plus précisément, f est considéré comme un weak learner si :

$$\epsilon(f) < \frac{1}{2} \quad (3)$$

Cela signifie qu'il est capable de faire des prédictions qui sont globalement meilleures que le hasard, mais pas suffisamment performantes pour être utilisées seules dans des tâches de classification complexes.

Dans son article, Friedman a considéré les arbres de décisions pour constituer ses weak learners.

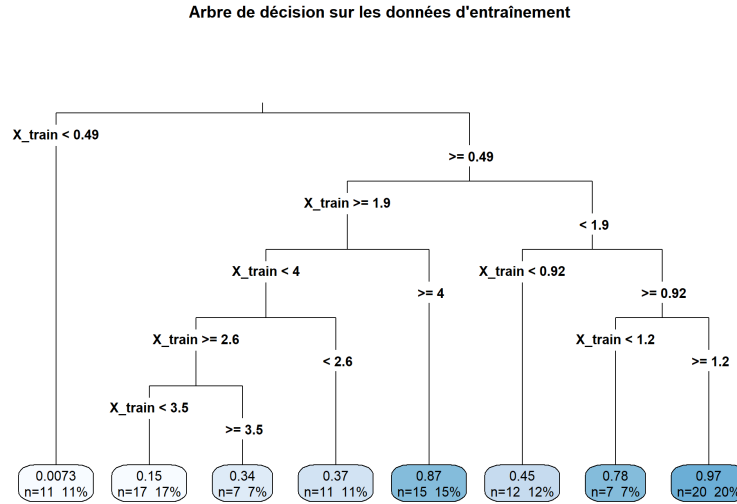


Figure 1: Illustration de l'arbre de décisions sur les données d'entraînement du jeu de données généré à la section 3 de notre mémoire.

2.3 Méthodologie

On dispose d'un jeu de données d'entraînement $D_n := \{(X_1, Y_1), (X_2, Y_2), \dots, (X_n, Y_n)\}$ contenant n paires $(X_i, Y_i)_{1 \leq i \leq n} \in \mathcal{X} \times \mathcal{Y}$ de variables aléatoires (v.a.) où $\mathcal{X} \subset \mathbb{R}^d$ et $\mathcal{Y} = \{-1, 1\}$ pour un problème de classification binaire ou $\mathcal{Y} = \mathbb{R}$ pour un problème de régression.

On suppose les données D_n indépendantes et identiquement distribuées (i.i.d.) et la v.a. $Y \in \mathbb{R}$ de carré intégrable, c'est-à-dire $\mathbb{E}[Y^2] < \infty$.

Objectif : Le but ultime du Gradient Boosting Machine est de trouver une fonction $F(x)$ qui minimise sa fonction de perte $L(y, F(x))$ de la manière suivante :

$$F^* = \arg \min_{F \in \mathcal{F}} \mathbb{E}[L(Y, F(X))] \quad (4)$$

en utilisant un ajustement itératif.

Avec \mathcal{F} est l'espace des fonctions de $\mathcal{X} \rightarrow \mathcal{Y}$, et $L : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ est une fonction de perte convexe et intégrable.

i°) Approche Additive : étant donné que la loi jointe de (X, Y) n'est connue que par l'échantillon D_n , le Boosting par descente de gradient utilise une forme additive pour approximer F^* . Nous exprimons ainsi $F(X)$ comme suit :

$$F(X) = \sum_{m=0}^M \beta_m h(X, a_m) \quad (5)$$

où $h(X, a_m)$ représente des apprenants faibles (typiquement des arbres de décision), et β_m sont les poids appris à chaque étape.

Cette approche ramène le problème d'estimation dans un espace fonctionnel de dimension infinie à un problème paramétrique plus simple. L'optimisation se reformule alors comme suit :

$$F^* = \arg \min_{\{a_m\}_{m=0}^M, \{\beta_m\}_{m=0}^M} \sum_{i=1}^n L \left(Y_i, \sum_{m=0}^M \beta_m h(X_i, a_m) \right). \quad (6)$$

ii°) Optimisation Itérative : l'optimisation simultanée de nombreux paramètres est souvent complexe. Pour simplifier, une approche dite "stage-wise" est utilisée, divisant le problème en M étapes. Cette dernière signifie que le modèle est construit progressivement en ajoutant une nouvelle fonction à chaque itération sans modifier les fonctions précédentes. Voici le processus détaillé :

1. Initialisation :

- On commence avec un modèle initial F_0 .

2. Calcul des Pseudo-Résidus :

- À chaque itération m , on calcule les pseudo-résidus $\tilde{y}_{i,m}$, qui correspondent à l'opposé du gradient de la fonction de perte par rapport aux prédictions actuelles du modèle :

$$\tilde{y}_{i,m} = - \frac{\partial L(Y_i, F_{m-1}(X_i))}{\partial F_{m-1}(X_i)} \quad (7)$$

Ces pseudo-résidus sont utilisés pour ajuster les apprenants faibles à chaque itération.

3. Pour chaque étape m de 1 à M :

(a) Optimisation des paramètres a_m :

- On ajuste les paramètres a_m du modèle faible $h(X, a_m)$ pour minimiser l'écart entre les pseudo-résidus et les prédictions du modèle faible :

$$a_m = \arg \min_a \sum_{i=1}^n \left[\frac{\partial L(Y_i, F_{m-1}(X_i))}{\partial F_{m-1}(X_i)} - h(X_i, a) \right]^2 \quad (8)$$

(b) Optimisation du poids β_m :

- On détermine le poids β_m optimal pour le modèle faible afin de minimiser la fonction de perte totale :

$$\beta_m = \arg \min_{\beta} \sum_{i=1}^n L(Y_i, F_{m-1}(X_i) + \beta h(X_i, a_m)) \quad (9)$$

(c) Mise à jour du modèle :

- On met à jour le modèle en ajoutant le modèle faible pondéré :

$$F_m(X) = F_{m-1}(X) + \beta_m h(X, a_m). \quad (10)$$

Ainsi, l'algorithme de Gradient Boosting se présente comme suit :

Algorithm 2 Gradient Boosting

```
1: Initialisation :  $F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$ 
2: for  $m = 1$  to  $M$  do
3:    $r_{i,m} \leftarrow -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \Big|_{F=F_{m-1}}, \forall 1 \leq i \leq n$ 
4:    $a_m \leftarrow \arg \min_{a, \beta} \sum_{i=1}^n (r_{i,m} - \beta h(x_i; a))^2$ 
5:    $\beta_m \leftarrow \arg \min_{\beta} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \beta h(x_i, a_m))$ 
6:    $F_m \leftarrow F_{m-1} + \beta_m h(x; a_m)$ 
7: end for
8: Sortie :  $F_M$ 
```

L'algorithme de Gradient Boosting est paramétré par :

- le choix de la perte L ;
- le nombre M d'apprenants faibles.

Remarque : Le choix de la fonction de perte ainsi que la technique utilisée pour la régularisation (pour éviter le surapprentissage du modèle) sont développés à aux sections 2.5 et 3.2 de notre mémoire, respectivement.

Cette approche est généralement utilisée avec des arbres de décision de taille fixe en tant que classificateurs faibles (ou ajustement de base dans le cadre d'une régression). Dans ce contexte, on parle alors de **TreeBoost**.

2.4 TreeBoost

Un arbre de décision divise l'espace des prédicteurs X en J régions distinctes, notées $\{R_j\}_{j=1}^J$. Pour chaque région R_j , une constante de prédiction b_j est attribuée. La fonction de prédiction a la forme additive suivante :

$$h(x, \theta) = \sum_{j=1}^J b_j \mathbb{I}(x \in R_j), \quad (11)$$

où $\theta = \{R_j, b_j\}_{j=1}^J$ paramètre l'arbre. Les arbres sont généralement ajustés en utilisant la méthode CART (Classification and Regression Trees).

Dans les modèles de gradient boosting, les arbres de décision sont souvent utilisés comme apprenants faibles. À chaque itération m , la fonction de prédiction est mise à jour en ajoutant la contribution de l'arbre m -ième :

$$f_m(x) = f_{m-1}(x) + \rho_m \sum_{j=1}^{J_m} b_{jm} \mathbb{I}(x \in R_{jm}), \quad (12)$$

où $\{R_{jm}\}_{j=1}^{J_m}$ sont les régions créées par le m -ième arbre pour prédire les pseudo-résidus $\{-g_m(x_i)\}_{i=1}^n$. Les b_{jm} sont les constantes qui minimisent l'erreur quadratique dans chaque région. À l'itération m , b_{jm} est donc la moyenne des pseudo-résidus pour les observations dans la région R_{jm} :

$$b_{jm} = \frac{\sum_{i \in S_{jm}} -g_m(x_i)}{|S_{jm}|}, \quad (13)$$

où S_{jm} est l'ensemble des observations dans la région R_{jm} et $|S_{jm}|$ est le nombre d'observations dans cette région. Le facteur d'échelle ρ_m est déterminé séparément.

Pour améliorer les ajustements, TreeBoost, une version modifiée de l'algorithme de gradient boosting, ajuste chaque région indépendamment en incluant ρ_m dans la somme. On peut réécrire la mise à jour comme :

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x \in R_{jm}), \quad (14)$$

où $\gamma_{jm} = \rho_m b_{jm}$ est le coefficient ajusté pour chaque région.

Les coefficients optimaux γ_{jm} sont déterminés en minimisant l'erreur quadratique des pseudo-résidus, ce qui se formalise comme suit :

$$\{\gamma_{jm}\}_{j=1}^{J_m} = \arg \min_{\{\gamma_{jm}\}_{j=1}^{J_m}} \sum_{i=1}^n L \left(y_i, f_{m-1}(x_i) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x_i \in R_{jm}) \right) \quad (15)$$

Autrement dit, on cherche à minimiser l'écart entre les pseudo-résidus observés et les prédictions fournies par les constantes optimisées pour chaque région.

Le nombre de régions J_m est souvent fixe pour chaque arbre, et ce nombre ainsi que le nombre total d'arbres M sont ajustés comme des hyperparamètres, souvent déterminés par validation croisée pour obtenir les meilleurs résultats.

Remarque : Dans la classification binaire, les apprenants faibles sont des modèles de classification simples, légèrement meilleurs que le hasard, comme des arbres de décision de base. En régression, ce sont généralement de simples arbres de régression, bien que tout modèle de régression simple puisse être utilisé. Utiliser des modèles complexes comme apprenants faibles avec le gradient boosting est possible, mais conduit souvent à de mauvaises performances.

Ci-dessous, la description de l'algorithme TreeBoost :

Algorithm 3 TreeBoost

1: **Initialisation :** $f_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$

2: **for** $m = 1$ **to** M **do**

3: **(i)** Calculer les pseudo-résidus :

$$r_{i,m} = - \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \Big|_{f=f_{m-1}}, \quad \forall 1 \leq i \leq n$$

4: **(ii)** Ajuster un arbre de décision aux données $\{(x_i, r_{i,m})\}_{i=1}^n$, ce qui donne les régions $\{R_{jm}\}_{j=1}^{J_m}$

5: **(iii)** Calculer le coefficient optimal pour chaque région :

$$\gamma_{jm} = \arg \min_{\gamma_{jm}} \sum_{i=1}^n L(y_i, f_{m-1}(x_i) + \gamma_{jm} \mathbb{I}(x_i \in R_{jm}))^2, \quad j = 1, \dots, J_m$$

6: **(iv)** Mettre à jour la fonction de prédiction :

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} \mathbb{I}(x \in R_{jm})$$

7: **end for**

8: **Fonction de prédiction finale :** $f_M(x)$

2.5 Régularisation : Shrinkage et Subsampling

L'un des problèmes qu'on rencontre en machine learning est le surajustement (ou "overfitting"). Si l'algorithme n'est pas bien réglé, le modèle peut facilement surajuster les données, c'est-à-dire qu'il prédit les données d'entraînement elles-mêmes plutôt que la relation entre les variables d'entrée et les variables de réponse. Un paramètre de régularisation clé pour le Gradient Boosting est le nombre d'itérations M . Augmenter M réduit l'erreur de prédiction sur les données d'apprentissage, mais peut entraîner un surajustement. La valeur optimale de M est souvent déterminée par validation croisée. D'autres techniques ont aussi été introduites afin de mieux équilibrer la performance prédictive du modèle. Parmi elles, nous pouvons citer :

i°) Shrinkage : c'est une technique de régularisation clé utilisée dans les algorithmes de gradient boosting qui consiste à modifier la règle de mise à jour. Elle permet de réduire la vitesse à laquelle l'algorithme apprend à partir des données d'entraînement à chaque itération. Cela se fait en introduisant un paramètre de retrécissement δ , communément appelé taux d'apprentissage (learning rate). Ainsi, à chaque itération, le nouvel apprenant faible est simplement ajusté en fonction de ce taux.

$$\hat{F}_m(x) = \hat{F}_{m-1}(x) + \delta \gamma_m c_m(x), \quad 0 < \delta \leq 1 \quad (16)$$

où :

- δ : le taux d'apprentissage contrôlant la vitesse de convergence.
- γ_m : le poids associé à l'apprenant faible à l'itération m .
- $c_m(x)$: la contribution de l'apprenant faible pour l'observation x

Plus δ est faible, plus l'algorithme apprend lentement. Par conséquent, diminuer la valeur de δ augmente la valeur optimale pour M . Ce qui signifie que ces deux paramètres doivent être optimisés conjointement, par exemple avec la validation croisée. Un δ petit augmente le nombre d'ajustements nécessaires, améliorant généralement la qualité des prédictions, tandis qu'un grand δ peut entraîner un surajustement.

ii°) Subsampling : la méthode la plus simple introduite pour les GBM est le sous-échantillonnage. Il consiste à introduire de l'aléa, à chaque itération, seulement une portion aléatoire des données d'entraînement au lieu de prendre l'ensemble complet. Cela peut se faire sans ou avec remplacement. Le paramètre principal est la `bag.fraction`, qui indique la proportion des données à utiliser à chaque itération, généralement entre 0 et 1. Par exemple, `bag.fraction = 0.25` signifie que, pour chaque arbre construit, seulement 25% des données sont utilisées, sélectionnées aléatoirement.

Remarque : Pour les arbres, la profondeur maximale est un autre paramètre important. Une plus grande profondeur augmente la complexité du modèle et sa capacité à capturer des interactions complexes entre les variables, mais peut aussi entraîner un surajustement.

3 Simulations et Application

Pour illustrer le Boosting et les propriétés des fonctions de perte, nous avons généré un jeu de données D_n avec $n = 100$ observations. Le jeu de données est échantillonné à partir d'une fonction $\sin^2(X)$ avec deux sources de bruit simulé artificiellement : une composante de bruit gaussien $\epsilon_1 \sim \mathcal{N}(0, 0.1^2)$ et une composante de bruit impulsif

$$\epsilon_2 = 1 \times (0.5 - U(0, 1)).$$

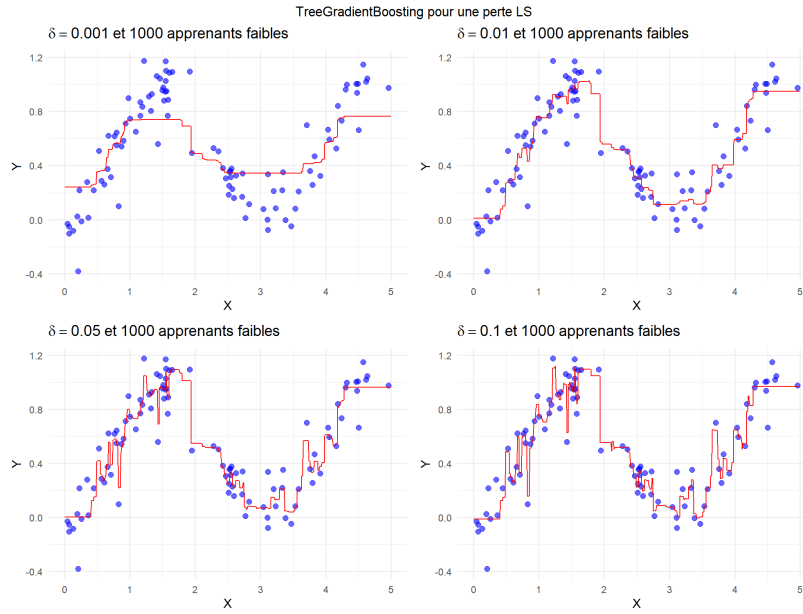


Figure 2: Entrainement du modèle avec la perte L2 (LS) où on fait varier le taux d'apprentissage δ

Le modèle de régression utilisé est :

$$Y = \sin^2(X) + \epsilon,$$

où $X \sim U(0,5)$ et ϵ le bruit défini par : $\epsilon = \epsilon_1 + \epsilon_2$

Ce jeu de données, avec $Y_i = \sin^2(X_i) + \epsilon_{1i} + \epsilon_{2i}$, permet d'analyser la robustesse des algorithmes de Boosting face à des données bruitées.

3.1 Analyse du package "gbm"

Le package `gbm` implémente l'algorithme de **Gradient Boosting Machine (GBM)** développé par Friedman, largement utilisé pour des problèmes de régression et classification. Il utilise la perte quadratique

$$L(y, f(x)) \sim (y - f(x))^2$$

par défaut, mais il peut être adapté avec d'autres fonctions de perte que nous développerons à la section suivante. Bien que `gbm` soit plus lent que des outils plus récents comme `XGBoost` (TIANQI CHEN, [7], 2016) par exemple, il reste un choix solide pour les utilisateurs cherchant une implémentation fidèle de l'algorithme original. Il permet une grande flexibilité dans le réglage des hyperparamètres, et son intégration avec des packages comme `caret` facilite l'entraînement et la validation croisée des modèles. Parmi ses paramètres clés à spécifier, on trouve :

- **Fonction de perte** (distribution) : gaussian (perte L2), huberized (perte Huber) et laplacian (perte L1) pour la régression ou encore bernoulli (perte Logistique) et adaboost pour la classification. Nous utiliserons quelques-unes d'entre elles dans la partie application de notre mémoire.
- **Nombre d'itérations** T (`n.trees`) : Entier spécifiant le nombre total d'arbres à ajuster. Cela équivaut au nombre d'itérations et au nombre de fonctions de base dans l'expansion additive. La valeur par défaut est 100. Si le nombre d'arbres de décision à construire est élevé cela peut améliorer la performance, mais augmente aussi le risque de surapprentissage (overfitting).
- **Profondeur des arbres** K (`interaction.depth`) : permet de déterminer la profondeur d'un arbre. Plus les arbres sont moins profonds plus le modèle est robuste et performant.

- **Taux d'apprentissage** λ (shrinkage) : connu sous le nom shrinkage; il s'agit du paramètre que l'on applique à chaque apprenant. Il prévient le surajustement (ou overfitting) et est typiquement choisis très petit. Des valeurs comprises entre 0.001 et 0.1 fonctionnent généralement, mais un taux d'apprentissage plus petit nécessite généralement plus d'arbres.
- **Taux de sous-échantillonnage** p , (bag.fraction) : à chacune des itérations, ce paramètre permet de définir la proportion de données à utiliser pour construire chaque arbre. Une façon d'introduire de l'aléa dans l'algorithme afin d'améliorer la robustesse du modèle.

3.2 Fonctions de perte

La première question qu'on se pose est : si on devait résoudre un problème de régression au lieu d'une classification, qu'est-ce qui changerait ? La réponse à cette question détermine exactement comment nous allons optimiser et quelles caractéristiques nous pouvons attendre du modèle final. Le choix de la bonne fonction de perte reste donc une étape cruciale à ne pas négliger. Nous allons ainsi explorer ces quelques fonctions pour les deux cas les plus courants en apprentissage automatique : la classification $\mathcal{Y} = \{-1, 1\}$ et la régression $\mathcal{Y} = \mathbb{R}$.

3.2.1. La régression : $\mathcal{Y} = \mathbb{R}$

Lorsque la variable de réponse y est continue, une tâche de régression est résolue.

- **Perte L2** (distribution = "gaussian") : une fonction de perte classique, couramment utilisée en pratique, est la perte en erreur quadratique L2. Elle est associée à la moyenne conditionnelle des données :

$$\mathcal{L}_{L2}(y, f) = \frac{1}{2}(y - f)^2 \quad (17)$$

Dans le cas de la fonction de perte L2, sa dérivée est le résidu $y - f$, ce qui implique que l'algorithme GBM effectue simplement un réajustement des résidus. L'idée derrière cette fonction de perte est de pénaliser les grandes déviations par rapport aux valeurs cibles tout en négligeant les petits résidus.

- **Perte L1** (distribution = "laplace") : appelée aussi la fonction de perte "Laplacienne". La perte L1 correspond à la médiane de la distribution conditionnelle, elle est donc considérée comme une perte robuste pour la régression :

$$\mathcal{L}_{L1}(y, f) = |y - f| \quad (18)$$

Elle peut être particulièrement intéressante dans les tâches où la variable de réponse présente une distribution d'erreurs à longue traîne. Bien qu'elle ne soit pas différentiable en certains points, elle est souvent préférée car elle réduit l'influence des grandes erreurs.

- **Perte Huber** (distribution = "huberized") : une alternative robuste à la perte L1 est la fonction de perte de Huber. Elle se compose de deux parties, correspondant aux pertes L2 et L1. La perte de Huber est définie comme suit :

$$\mathcal{L}_{\text{Huber}}(y, f, \delta) = \begin{cases} \frac{1}{2}(y - f)^2 & \text{si } |y - f| \leq \delta \\ \delta(|y - f| - \frac{\delta}{2}) & \text{si } |y - f| > \delta \end{cases} \quad (19)$$

Le paramètre de découpage δ est utilisé pour spécifier l'effet de robustesse de la fonction de perte. L'intuition derrière ce paramètre est de spécifier la valeur maximale de l'erreur, après laquelle la perte L1 doit être appliquée.

Une approche plus générale consiste à prédire un quantile conditionnel de la variable de réponse (Koenker et Hallock, 2001). Cette approche est libre de toute distribution et se

révèle en général offrir une bonne robustesse aux valeurs aberrantes. La perte quantile est organisée comme suit :

$$\mathcal{L}_\alpha(y, f) = \begin{cases} (1 - \alpha)|y - f| & \text{si } y - f \leq 0 \\ \alpha|y - f| & \text{si } y - f > 0 \end{cases} \quad (20)$$

Le paramètre α dans ce cas spécifie le quantile souhaité de la distribution conditionnelle. On peut noter que lorsque $\alpha = 0,5$, cela coïncide avec la perte $L1$, donnant ainsi la médiane conditionnelle.

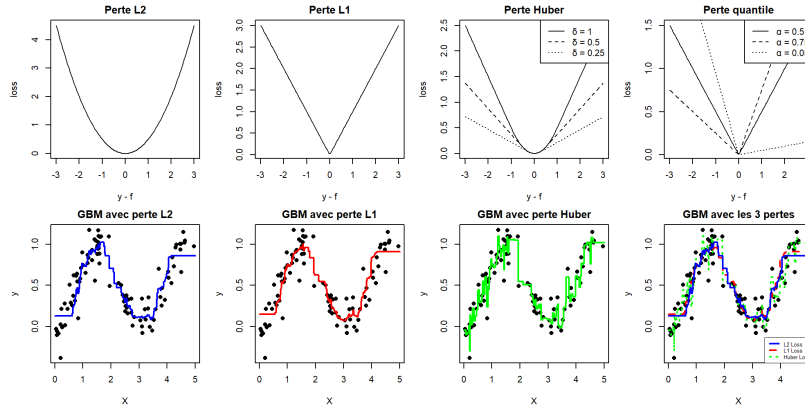


Figure 3: Fonction de pertes et GBM pour chacune des 3 fonctions de perte

Comme on peut le voir sur cette figure ci-dessous, on constate que le modèle GBM qui a pour fonction de perte la fonction de Huber a tendance à mieux traiter les outliers (valeurs aberrantes) que les deux autres fonctions que sont la $L1$ et la $L2$.

3.2.1. La classification : $\mathcal{Y} = \{-1, 1\}$

Dans le cas d'une réponse catégorielle, la variable de réponse y prend généralement des valeurs binaires $y \in \{0, 1\}$, un problème de classification est alors résolu.

- **Perte Logistique** (distribution = "bernoulli") : supposons les étiquettes transformées \bar{y} , en posant $\bar{y} = 2y - 1$, ce qui donne $\bar{y} \in \{-1, 1\}$. Dans ce cas, la probabilité de réponse par classe peut être estimée en minimisant la log-vraisemblance négative, associée aux nouvelles étiquettes de classe :

$$\mathcal{L}_{\text{Bern}}(y, f) = \log(1 + \exp(-2y\bar{f})) \quad (21)$$

Cette fonction de perte est communément appelée la perte de Bernoulli.

- **Perte AdaBoost** : un autre choix courant de fonction de perte catégorielle est la simple perte exponentielle, telle qu'elle est utilisée dans l'algorithme Adaboost (Schapire, 2002). En suivant la même notation que celle de la perte de Bernoulli, la fonction de perte Adaboost est donc définie comme suit :

$$\mathcal{L}_{\text{Ada}}(y, f) = \exp(-y\bar{f}) \quad (22)$$

Il est possible d'établir un lien entre la réduction d'influence des GBMs avec la fonction de perte Adaboost et l'algorithme Adaboost de réduction de poids (Friedman, 2001).

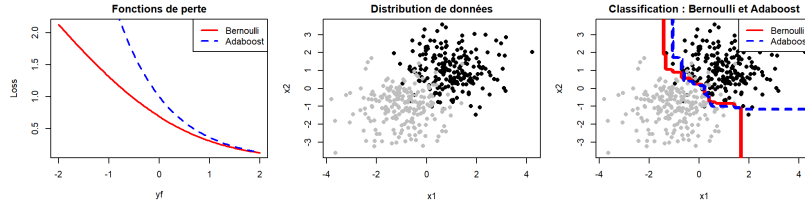


Figure 4: Fonctions de perte Logistique et Adaboost, et leur application dans un problème de classification

3.3 Gradient Boosting Stochastique

Dans [4], Friedman propose d'ajouter de l'aléa dans l'algorithme de Gradient Boosting par le biais du subsampling. À chaque itération, un sous-échantillon de \tilde{n} données, tirées aléatoirement et sans remise parmi le jeu d'entraînement D_n , est utilisé pour entraîner l'apprenant faible. Cette méthode, connue sous le nom de Gradient Boosting Stochastique, permet de réduire le biais et la variance tout en augmentant la robustesse du modèle et en évitant l'overfitting. L'algorithme de gradient boosting stochastique se présente comme suit :

Algorithm 4 Gradient Boosting stochastique

But : minimisation de (3)

Entrée : proportion f de données : $\tilde{n} = f \times n$

Initialisation : $F_0(x) \leftarrow \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$

for $m = 1$ **to** M **do**

$\mathcal{D}_{\tilde{n}} \leftarrow$ sous échant. aléatoire de taille $\tilde{n} = f \times n$

$\tilde{y}_{i,m} \leftarrow -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} \Big|_{F(x)=F_{m-1}(x)}, \forall x_i \in \mathcal{D}_{\tilde{n}}$

$a_m \leftarrow \arg \min_{a, \beta} \sum_{x_i \in \mathcal{D}_{\tilde{n}}} (\tilde{y}_{i,m} - \beta h(x_i; a))^2$

$\beta_m \leftarrow \arg \min_{\beta} \sum_{x_i \in \mathcal{D}_{\tilde{n}}} L(y_i, F_{m-1}(x_i) + \beta h(x_i; a_m))$

$F_m \leftarrow F_{m-1} + \beta_m h(x; a_m)$

end

Sortie : F_M

Le paramètre $f \in [0, 1]$ ajuste le niveau d'aléa dans l'algorithme. Pour $f = 1$, l'algorithme est déterministe, tandis que pour $0 < f < 1$, une fraction \tilde{n} des données est utilisée à chaque itération. Cela peut améliorer les performances et réduire les coûts de calcul, mais un f trop faible peut augmenter la variance des modèles faibles.

Le Gradient Boosting est souvent utilisé avec des arbres de décision CART, comme le montre l'implémentation `gbm` dans le package `gbm` de R.

Remarque : Dans le package `gbm` de R, le gradient boosting stochastique est spécifié par le paramètre `bag.fraction`. Par exemple, un `bag.fraction = 0.2` signifie que chaque arbre est construit sur un sous-échantillonnage de 20% des données d'entraînement.

4 Cas pratique sur des données réelles

Dans cette section, nous allons plonger dans une étude de cas détaillée à partir d'un ensemble de données obtenu sur [Kaggle](#). Tout d'abord, nous donnerons un aperçu de ce jeu de données. Ensuite, nous passerons en revue notre variable cible et les variables explicatives. Puis, nous partagerons notre démarche pour construire, entraîner et comparer ces modèles. Enfin, nous dévoilerons nos résultats et analyses.

4.1 Exploration et prétraitement des données

Le jeu de données que nous utilisons pour notre application pratique du modèle de gradient boosting machine (GBM) est un ensemble de données qui porte sur l'étude du cancer du sein. Ce genre de jeu de données est couramment utilisé pour tester des algorithmes de classification binaire. Il contient des mesures issues de biopsies réalisées sur des patients. Notre objectif sera donc de mettre en place un modèle de classification pour prédire si une tumeur est bénigne ("B") ou maligne ("M") sur la base des mesures (les variables explicatives). Notre variable cible "*diagnosis*" est qualitative (categorical) tandis que le reste des variables (explicatives) sont quantitatives (numeric). La table de la variable cible de notre jeu de données montre qu'on a 357

```
# Conversion de la variable diagnosis : on convertit le "M" en "1" et le "B" en "0"
data$diagnosis <- as.factor(ifelse(data$diagnosis == "M", 1, 0))

# Vérifier les changements
table(data$diagnosis)
```

0	1
357	212

Figure 5:

patients avec une tumeur bénigne ("0") et 212 avec une tumeur maligne ("1"). Ce qui nous donne un ensemble de données légèrement déséquilibré avec 62,742% pour la classe "0" et 32,258% pour la classe "1". Notons que les variables explicatives ce sont des mesures qui peuvent inclure des caractéristiques comme la taille, le diamètre, la symétrie, la texture, la surface, les points concaves et d'autres attributs des cellules du corps. Le graphe ci-dessous nous illustre les corrélations entre la variable cible et les variables explicatives :

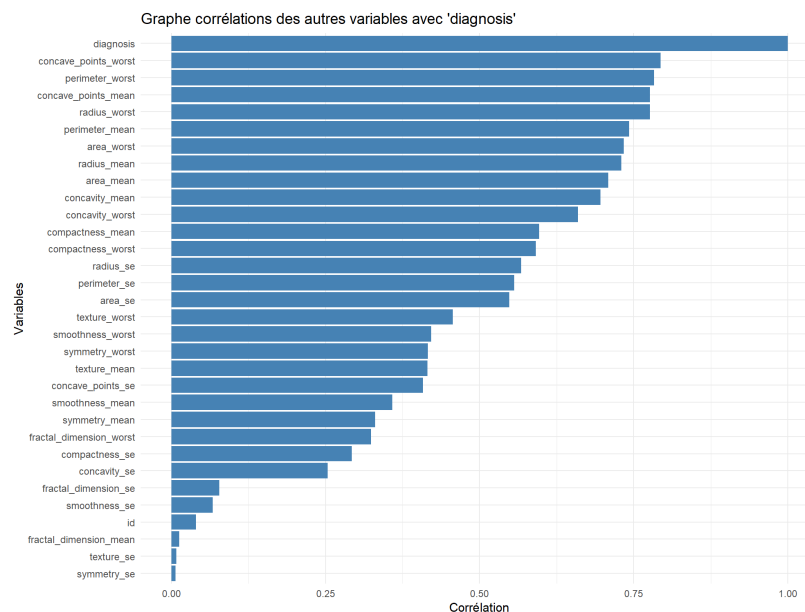


Figure 6: Valeur absolue de la corrélation entre la variable *diagnosis* et les variables explicatives

Remarque : Nous avons utilisé la valeur absolue pour la comparaison des corrélations. On s'est concentré uniquement sur la force d'influence des variables explicatives sur la variable cible "*diagnosis*", sans se soucier de la direction (négative ou positive).

4.2 Construction, évaluation et analyses des modèles

On va consacrer cette partie à la mise en place de modèles d'agrégation (par la moyenne) comme le randomForest (forêts aléatoires) et le Bagging (bootstrap), et le modèle Support Vector Machine

(SVM). On va ensuite essayer de les comparer à notre modèle de gradient boosting (stochastic).

Tout d'abord on a divisé le jeu de données en données d'entraînement (80%) et en données de test (20%) qu'on va normaliser par la suite afin d'améliorer la performance et la qualité des modèles.

Et pour améliorer leurs performances et robustesses on a utilisé la validation croisée sur les données d'entraînement pour tous les modèles. Cela nous permet d'atteindre de meilleures performances et de réduire au maximum les erreurs de prédiction. En plus des arbres de décision comme apprenant de base, le gradient boosting machine utilise des techniques de sous-échantillonnage (`bag.fraction`) et des hyperparamètres tels que le `shrinkage` pour cerner les problèmes d'overfitting et gérer encore mieux les erreurs de prédictions. Le graphe ci-dessous nous montre la précision (**accuracy**) de chacun des différents modèles qu'on a entraîné à partir de notre jeu de données :

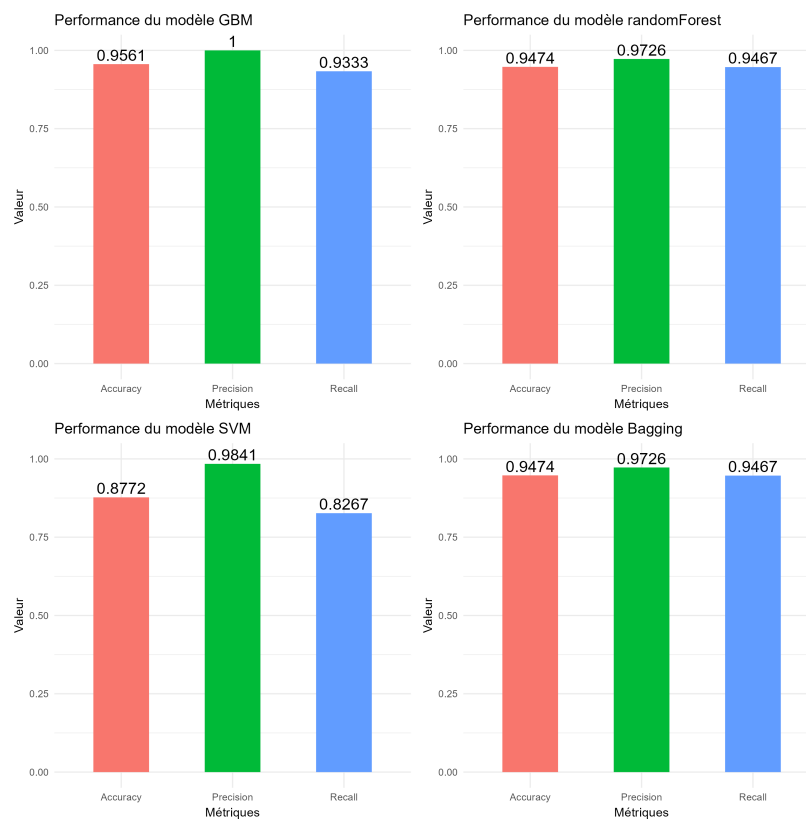


Figure 7: Comparaison des métriques (accuracy, precision, recall) des modèles : GBM, Bagging, randomForest et SVM

On constate que le modèle de gradient boosting stochastic est meilleur que les trois autres (un "peu" meilleur que les modèles Bagging et randomForest). Le modèle GBM se distingue par sa précision globale (**accuracy**) élevée (95,61%) et sa précision de 100%, ce qui veut dire qu'il fait aucune erreur lorsqu'il s'agit d'identifier une tumeur Maligne ("1"). Par contre, son recall de 93,33% prouve qu'il en manque quelques unes. Les modèles randomForest et Bagging, avec des métriques très similaires, sont presque aussi précis (accuracy = 94,74%), et détectent un peu mieux les tumeurs Malignes (recall = 94,67%) que le modèle GBM. Ce peut les rendre plus fiables lorsqu'il s'agit de détecter de faux négatifs. Le modèle SVM, quant à lui, est le moins performant des quatre modèles. Même avec une bonne précision (98,41%), le modèle a plus de cas manqués. Ce qui fait de lui le modèle qui occupe la dernière place de notre classement (sur ce jeu de données bien sûr).

Pour conclure, le modèle de GBM reste le meilleur pour des prédictions fiables. Mais si le but était de détecter plus de cas positifs les modèles randomForest et Bagging seraient de meilleurs

candidats.

Ainsi, pour nous assurer d'avoir le meilleur modèle nous allons essayer de trouver des paramètres optimaux par la technique de la grille de recherche afin d'améliorer sa robustesse ainsi que sa fiabilité. Nous allons essayer de le comparer à l'un des modèles de boosting les plus récents, le modèle xgboost (TIANQI CHEN , 2016, [7]).

4.3 Optimisation du modèle GBM et comparaison avec le modèle XGBoost

Nous avons essayé d'améliorer notre modèle de gradient boosting en cherchant les hyperparamètres optimaux qui nous donneraient une performance encore meilleure que celle-ci, bien vrai que 95,614% reste quand-meme un bon score de prédiction. Puis on a constaté que son accuracy et son recall ne changent pas, bien qu'on ait pris des paramètres optimaux. Cependant, sa précision décroît, ce qui nous pousse à garder le modèle idéaln dans ce cas pour la comparaison avec le modèle xgboost. Ce dernier montre un accuracy de 96,49% contre 95,61%. Ce qui veut dire il a classé correctement un peu plus de données que le modèle GBM. De plus, avec 97,33% contre 93,33%, ces deux métriques nous permet de conclure que le modèle XGBoost est légèrement plus précis et complet que GBM pour la détection de tumeurs Malignes. Pour une précision de 97,33% contre 100%, le modèle GBM n'a commis aucune erreur de prédiction contrairement à XGBoost. Mais il manque un peu plus de tumeurs Malignes dans sa détection, contrairement à XGBoost.

Globalement, on peut en déduire que XGBoost semble etre le meilleur modèle pour ce cas, surtout si le but de l'étude est de classer toutes les tumeurs Malignes.

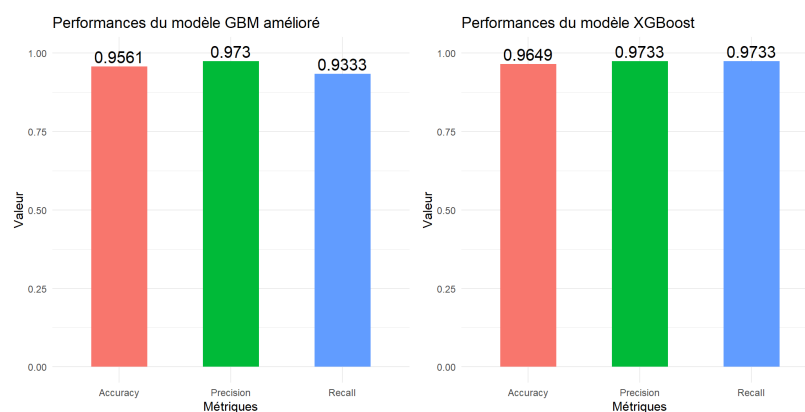


Figure 8: Comparaison des métriques (accuracy, precision, recall) des modèles : GBM et XGBoost

5 Conclusion

A travers ce mémoire nous avons exploré en profondeur la théorie de Fiedman sur le Gradient Boosting Machine (GBM) avant de l'appliquer avec le package `gbm` de R. Les algorithmes de GBM sont à l'origine de modèles robustes et très puissants qu'on peut appliquer à différents problèmes de classification ou de régression. Nous avons su souligner l'importance de la fonction de perte, des arbres de décisions comme apprenants faibles et surtout les techniques de régularisation. L'applicationn pratique sur des données réelles nous a démontré la robustesse et la flexibilité du gradient boosting, en mettant en lumière l'ajustement des hyperparamètres qui peuvent parfois lui faire défaut. Toutefois, le gradient boosting reste limité car pouvant parfois etre un peu trop lent et moins efficace comparé à certains modèles comme XGBoost par exemple.

A l'ère de la Big Data (données massives) les problèmes sont de plus en plus complexes et la rapidité est au coeur des meilleurs modèles, ce qui nécessite des pistes d'améliorations. Ainsi, si on devrait pousser notre étude ce serait dans ce sens.

Bibliography

- [1] J. H. Friedman. *Greedy function approximation: A gradient boosting machine*. Annals of Statistics, 29(5):1189–1232, 2001.
- [2] C. Burges. *From ranknet to lambdarank to lambdamart: An overview*. Microsoft Research Technical Report MSR-TR-2010-82, 2010.
- [3] Y. Freund and R. E. Schapire. *A decision-theoretic generalization of on-line learning and an application to boosting*. Journal of Computer and System Sciences, 55(1):119–139, 1997.
- [4] J. H. Friedman. *Stochastic Gradient Boosting*. Computational Statistics & Data Analysis 38, 367–378, 2002.
- [5] G. Ridgeway. *Generalized Boosted Models: A guide to the gbm package*. Technical Report, June 26, 2024. Available online: <https://cran.r-project.org/web/packages/gbm/vignettes/gbm.pdf>
- [6] Robin Genuer and Jean-Michel Poggi. *Arbres cart et forêts aléatoires, importance et sélection de variables.*, 2017.
- [7] T. Chen and C. Guestrin. *XGBoost: A Scalable Tree Boosting System*. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, pp. 785–794, 2016. Available at: <https://arxiv.org/abs/1603.02754>
- [8] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*, Routledge, 2017.