SORBONNE UNIVERSITÉ

Méthode de Gradient Boosting

Laure Ferraris, Paul Liautaud

20 mai 2022

"Quels que soient les progrès des connaissances humaines,
il y aura toujours place pour l'ignorance et par suite pour le hasard et la probabilité."
Émile Borel

Table des matières

1 Introduction

Le Boosting est une méthode d'apprentissage supervisé consistant à bâtir une prédiction fiable en agrégeant les réponses d'apprenants de base, c'est-à-dire d'estimateurs tout juste meilleurs que le hasard. Cette famille d'algorithmes de machine learning construit séquentiellement des apprenants de base, encore appelés faibles ou *weak learners*. A chaque itération, le nouvel estimateur favorise son apprentissage sur les erreurs du précédent et s'y ajoute pour finalement obtenir un *strong learner*. Cette méthode a été particulièrement reconnue avec l'algorithme Adaboost (FREUND, SHAPIRE, 1996). Aujourd'hui encore de nombreux challenges sont remportés par des méthodes similaires comme XGBoost/LightGBM (FELLOUS, 2019) réputées puissantes tant sur des modèles de régression que de classification. Le Gradient Boosting proposé par FRIEDMAN (1999 et 2002) est une interprétation du Boosting comme une descente de gradient dans un espace fonctionnel appliquée à un problème d'optimisation dont la fonction objectif est l'erreur en espérance. Cette observation permet d'appliquer la méthode pour un large choix de fonctions de perte comme par exemple l'erreur quadratique en régression ou la fonction logit en classification.

Considérons un exemple de *weak learner* qui constituera notre apprenant de base tout le long de ce cours : l'arbre décisionnel. S'il est peu profond, il est particulièrement simple à mettre en oeuvre et interprétable, en revanche sa pertinence est faible. Comment améliorer ses performances ? Augmenter le nombre de noeuds est peu concluant car l'arbre souffre alors d'une variance trop forte. Les méthodes d'agrégation proposent différentes solutions effectives, en générant de multiples versions d'un arbre avant de les combiner. Disposant d'un unique jeu de données, il est alors nécessaire de le perturber pour obtenir des arbres dont les réponses sont différentes mais cohérentes. C'est ce que BREIMAN dénomme *perturbing and combining* [2]. Dans cet esprit, le Bagging construit chaque arbre à partir d'un échantillon obtenu par bootstrapping : tirage uniforme et avec remise de même taille que le jeu d'entraînement. L'algorithme renvoie la moyenne de la collection d'arbres, estimateur plus robuste, sa variance étant diminuée. Les Forêts aléatoires, procèdent selon le même principe mais les phases d'apprentissage sont réalisées sur des sous-échantillons sans remise. La dépendance entre chaque arbre est alors plus faible, la variance de leur moyenne également et la réponse plus fiable encore. Venons en au Boosting connu pour surpasser les performances du Bagging et des Forêts alétoires. Comme chaque nouvel arbre apprend des erreurs du précédent et s'y agrège, on peut voir cela comme une diminution du biais. La variance est également améliorée par cet algorithme. Dans [9] FRIEDMAN introduit l'algorithme Stochastic Gradient Boosting en injectant de l'aléa par sous échantillonage dans le Grandient Boosting. Les résultats de prédictions peuvent gagner en précision et le coût de calcul est allégé.

Nous allons dans un premier temps introduire l'algorithme Gradient Boosting ainsi que sa version stochastique. Dans un deuxième temps nous nous intéresserons à l'algorithme Gradient Tree Boosting où les apprenants faibles sont des arbres CART. Enfin nous discuterons de l'ajustement des paramètres ainsi que de leur interprétation en illustrant nos propos par des simulations.

Ce cours est accompagné du Notebook suivant : NOTEBOOK

2 Gradient Boosting

On dispose d'un jeu de données d'entraînement $\mathcal{D}_n := \{(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), ..., (\mathbf{X}_n, Y_n)\}$ contenant n paires $(\mathbf{X}, Y) \in \mathcal{X} \times \mathcal{Y}$ 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 $\mathcal{D}_n$ indépendantes et identiquement distribuées (i.i.d.) et la v.a. $Y \in \mathbb{R}$ de carré intégrable, i.e. $\mathbb{E}\left[Y^2\right] < \infty$.

But : Dans ce cours on s'intéresse à l'estimation de la fonction $F^* : \mathcal{X} \to \mathcal{Y}$ par l'algorithme Stochastic Gradient Boosting et solution du problème

$$F^* = \arg\min_{F \in \mathcal{F}} \mathbb{E}\left[L(Y, F(\mathbf{X}))\right] \tag{1}$$

où $\mathcal{F}$ est l'espace des fonctions de $\mathcal{X} \to \mathcal{Y}$.

Nous introduisons dans un premier temps l'élément essentiel du Boosting : l'apprenant de base.

2.1 Apprenant de base

Le Boosting transforme un *weak learner* en un *strong learner* par incréments successifs. Chaque agrégat *booste* l'apprenant initial en concentrant son entraînement sur des erreurs passées et s'applique tant à un problème de régression que de classification. Le concept de *weak learner* a émergé du développement de la théorie de l'*apprentissage PAC* (probably approximately correct learning) [14].
La condition pour être un *weak learner* est de prédire une réponse suffisamment précise avec probabilité au moins $\frac{1}{2}$.

Définition 2.1 (Apprenant faible [11])

On appelle *apprenant de base* ou *apprenant faible* un algorithme $\mathcal{W}$ pour lequel il existe $\varepsilon_w < \frac{1}{2}$ et $\delta_w < 1$ tels que, étant données $\mathcal{D}_n$, alors $\mathcal{W}$ génère un résultat avec précision (*accuracy*) au moins $1 - \varepsilon_w$ et probabilité au moins $1 - \delta_w$.

On appellera $\mathcal{H}$ la classe de tels apprenants faibles.

Remarque. Cette définition de la classe $\mathcal{H}$ reste pour l'instant très généraliste étant donnée la variété des faibles apprenants (voir les exemples suivants pour s'en convaincre). On explicitera plus tard $\mathcal{H}$ comme la restriction à une famille d'apprenants de base bien spécifiques pour notre étude.

Exemple 2.1. Bien que les points suivants ne soient pas formellement répertoriés comme apprenants de base, on considère ceux-ci comme de bons candidats pour des modèles de faible apprentissage :

- Multi-Layer Perceptron : un réseau avec peu de couches et un faible nombre de neurones (nœuds) sur chaque couche qui s'applique sur peu de variables d'entrée ;
- Classification naïve bayésienne ;
- Arbres prédicteurs de profondeur faible.

Pour $0 < \varepsilon < \frac{1}{2}$ et $0 < \delta < 1$ donnés, le but d'un algorithme Boosting est alors, en accord avec cette définition, de produire un résultat de haute précision (au moins $1 - \varepsilon$) et ce avec grande confiance $1 - \delta$ en utilisant itérativement le.s apprenant.s faible.s $\mathcal{W}$.

Comment comprendre le *boost* de chaque apprenant ? Considérons un exemple jouet pour en illustrer informellement le principe. Étant donné un échantillon $\mathcal{D}_n = (X_i, Y_i)_{1 \leq i \leq n}$ i.i.d. on souhaite résoudre le problème de régression

$$Y_i = F^*(X_i) + \varepsilon_i,\ 1 \leq i \leq n, \tag{2}$$

avec $F^* : \mathbb{R}^d \to \mathbb{R}$ mesurable et les ε_i i.i.d. tels que $\mathbb{E}[\varepsilon_i | X_i] = 0$. Pour cet exemple, nous simulons un échantillon $\mathbb{X} = (X_1, \ldots, X_n) \in \mathbb{R}^n$, i.i.d. selon la loi $\mathcal{U}([0, 5])$. On pose $F^*(x) = \sin(x)$, $\forall x \in [0, 5]$ et $Y_i = F^*(X_i) + \varepsilon_i$, où ϵ_i est nul avec probabilité $\frac{4}{5}$, soit de loi uniforme sur $[-\frac{1}{2}, \frac{1}{2}]$.

Un unique arbre de régression de profondeur 2 décrira pauvrement l'interdépendance entre X et Y mais mieux que le hasard, c'est notre apprenant faible. La classe $\mathcal{H}$ est ici l'ensemble des arbres de profondeur 2. On évalue l'erreur à l'aide du risque quadratique :

$$\frac{1}{2n} \sum_{i=1}^{n} (Y_i - h(X_i))^2 = \frac{1}{n} \sum_{i=1}^{n} L(Y_i, h(X_i))$$

où $h \in \mathcal{H}$ et $L(y, h) = \frac{1}{2}(y - h)^2$ est la perte quadratique. Notons $\tilde{Y}_{0,i} = Y_i - h_0(X_i)$ l'ensemble des résidus. On peut noter naïvement $Y_i = h_0(X_i) + \tilde{Y}_{0,i}$. Le Boosting améliore les performances de h_0 en lui ajoutant un arbre décisionnel de même profondeur h_1 qui estime $\tilde{Y}_{0,i}$. En pondérant les arbres h_0 et h_1 par des coefficients réels α_0 et α_1 que l'on détaillera plus tard, on obtient un nouvel estimateur $F_1(X_i) = \alpha_0 h_0(X_i) + \alpha_1 h_1(X_i)$. On peut alors définir $\tilde{Y}_{1,i} = Y_i - F_1(X_i)$ et réitérer le processus. La Figure 1 illustre cet exemple pour une famille de 1 à 9 arbres.

A l'itération m, nous disposons d'un estimateur $F_m(X) = \sum_{k=0}^{m} \alpha_k h_k(X)$ où $F_0 = h_0$. On peut voir $F_m(\mathbb{X})$ comme un vecteur de taille de l'échantillon n, $F_m(\mathbb{X}) = (F_m(X_1), \ldots, F_m(X_n))$ dont la perte peut s'écrire $\mathcal{L}(Y_1, \ldots Y_n, F(\mathbb{X})) = \sum_{i=1}^{n} L(Y_i, F(X_i))$. Dans [3] (1997), LEO BREIMAN observe que l'on peut interpréter le Boosting comme une méthode de descente. En effet dans cet exemple $\tilde{Y}_{m,i} = -\left[\frac{\partial L(Y_i, F_m(X_i))}{\partial F_m(X_i)}\right]$ est la direction

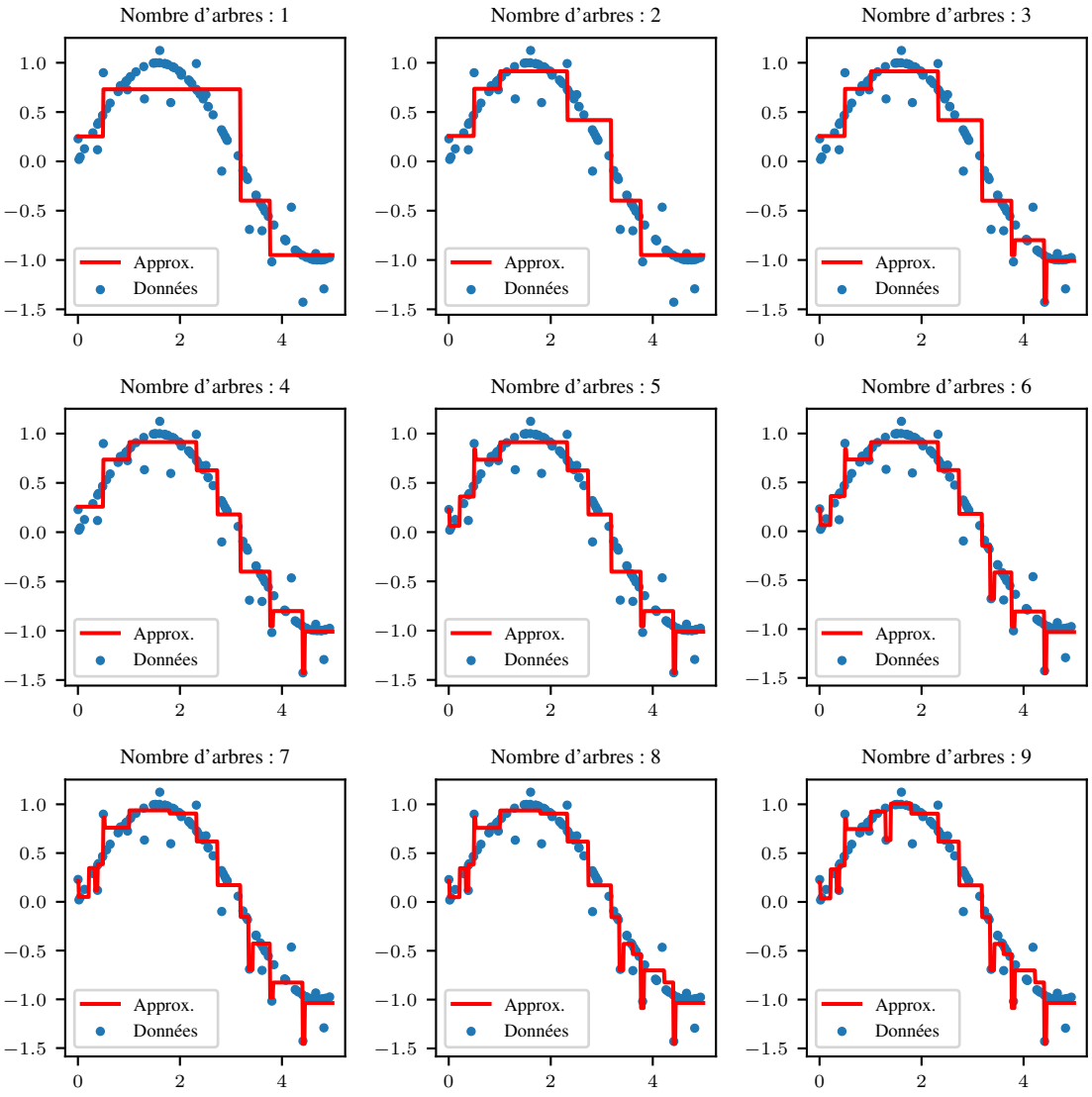

FIGURE 1 – Algorithme Boosting appliqué à des arbres décisionnels de profondeur 2.

de descente maximale en la coordonnée i du vecteur $F_m(\mathbb{X})$. Alors $h_m(X_i)$ approxime cette direction de descente maximale et α_m est peut être interprété comme le pas de l'algorithme à l'étape m, $F_m(\mathbb{X}) = F_{m-1}(\mathbb{X}) + \alpha_m h_m(\mathbb{X})$, avec $h_m(\mathbb{X}) = (h_m(X_1), \ldots, h_m(X_n))$.

Cette remarque permet de définir un cadre général pour le Boosting et de l'appliquer à des fonctions de pertes diverses.

2.2 Caractérisation par la descente de gradient

2.2.1 Heuristique

Dans un problème d'estimation de fonction on dispose d'un ensemble de variables aléatoires réponses ou sorties Y ainsi que de variables aléatoires explicatives ou entrées $\mathbf{X} = (X_1, \ldots, X_d) \in \mathbb{R}^d$. Étant donné le jeu d'entraînement $\mathcal{D}_n = \{(\mathbf{X}_1, Y_1), \ldots, (\mathbf{X}_n, Y_n)\}$ dont les valeurs sont connues, l'objectif est de trouver une fonction $F^*(\mathbf{X})$ qui à $\mathbf{X}$ associe Y. Il n'existe pas forcément de relation déterministe entre $\mathbf{X}$ et Y néanmoins on souhaite trouver le meilleur F^* qui modélise une interdépendance. Pour une classe de fonction $\mathcal{F}$ choisie, un tel F^* est défini par

$$F^* = \arg\min_{F \in \mathcal{F}} \mathbb{E}\left[L(Y, F(\mathbf{X}))\right] \tag{3}$$

où $L : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ est, dans notre cadre, une fonction de perte convexe et intégrable.

En pratique on ne connaît la loi jointe $(\mathbf{X}, Y)$ qu'à travers l'échantillon $\mathcal{D}_n$. On approche (3) par sa version empirique

$$F^* = \arg\min_{F \in \mathcal{F}} \sum_{i=1}^{n} L(Y_i, F(\mathbf{X}_i)). \tag{4}$$

Le Boosting approxime F^* par une forme additive

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

où $\{\beta_m\}_{m=0}^M$ sont les poids attribués à chaque apprenant faible $\{h(\cdot, a_m)\}_{m=0}^M$. Chaque apprenant $h(\cdot, a_m) \in \mathcal{H}$ est caractérisé par un ensemble de paramètres $\{a_m\}_{m=0}^M$. L'estimation dans un espace fonctionnel de dimension infinie est donc ramenée à un problème paramétrique plus simple. On peut alors reformuler (4)

$$F^* = \arg \min_{\{a_m\}_0^M, \{\beta_m\}_0^M} \sum_{i=1}^{n} L\left(Y_i, \sum_{m=0}^{M} \beta_m h(\mathbf{X}_i, a_m)\right). \tag{6}$$

L'optimisation simultanée de nombreux paramètres est complexe. Considérons une version relâchée de (6) en suivant une approche dite *stage wise*, qui scinde le problème (3) en M étapes. On initialise la procédure pour un F_0 puis pour $1 \leq m \leq M$

$$a_m, \beta_m = \arg\min_{a, \beta} \sum_{i=1}^{n} L\left(Y_i, F_{m-1}(\mathbf{X}_i) + \beta h(\mathbf{X}_i, a)\right), \tag{7}$$

où F_{m-1} est définie récursivement par

$$F_m(\mathbf{X}) = F_{m-1}(\mathbf{X}) + \beta_m h(\mathbf{X}, a_m). \tag{8}$$

Pour $1 \leq m \leq M$, la méthode Gradient Boosting optimise les paramètres a_m et β_m un à un

1.
$$a_m = \arg\min_a \sum_{i=1}^{n} \left(\frac{\partial L\left(Y_i, F_{m-1}(\mathbf{X}_i)\right)}{\partial F_{m-1}(\mathbf{X}_i)} - h(\mathbf{X}_i, a) \right)^2 \tag{9}$$

2.
$$\beta_m = \arg\min_\beta \sum_{i=1}^{n} L\left(Y_i, F_{m-1}(\mathbf{X}_i) + \beta h(\mathbf{X}_i, a_m)\right). \tag{10}$$

On peut interpréter les deux étapes ci-dessus comme une itération d'un algorithme de descente de gradient pour le problème d'optimisation sous contrainte dans l'espace fonctionnel $\mathcal{F}$.

Pour $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ de $\mathcal{D}_n$ notons $\mathcal{L}$:

$$\mathcal{L} : \left| \begin{array}{ccc} \mathcal{F} & \longrightarrow & \mathbb{R} \\ F & \longmapsto & \mathcal{L}(F(\mathbf{x}_1), \ldots, F(\mathbf{x}_n)) = \sum_{i=1}^{n} L\left(y_i, F(\mathbf{x}_i)\right). \end{array} \right. \tag{11}$$

On considère le problème d'optimisation

$$\min_{F \in \mathcal{F}} \mathcal{L}(F), \tag{12}$$

sous la contrainte que $F \in \mathcal{H}$.

L'algorithme de descente du gradient est une méthode classique de résolution d'un tel problème. En ignorant la contrainte, les itérés admettent la forme récursive suivante $F_{m+1} = F_m - \beta \nabla \mathcal{L}(F)_{|F=F_m}$, où $\beta > 0$ est un pas d'apprentissage strictement positif. Une version sous contrainte peut être donnée par

$$F_{m+1} = F_m + \beta h_m$$

où h_m est dans l'espace des contraintes et β le pas optimal. Pour $h_m \in \mathcal{H}$ faible apprenant alors F construite de cette façon sera bien définie, comme étant la somme des différents (h_m), par récurrence.
On souhaite alors trouver une fonction $h \in \mathcal{H}$ la plus proche possible de l'opposé du gradient $\nabla \mathcal{L}(F)$ par rapport aux données d'entraînement $\mathcal{D}_n$. Pour ce faire, comme $\nabla \mathcal{L}(F)$ est un vecteur, on peut chercher $h \in \mathcal{H}$ de telle sorte que :

- $\| - \nabla \mathcal{L}(F) - h\|_2^2 = \sum_{i=1}^{n} (-\frac{\partial L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)} - h(\mathbf{x}_i))^2$ soit minimal.

On peut voir cette étape comme la projection de $-\nabla \mathcal{L}(F)$ sur l'espace des contraintes $\mathcal{H}$. Comme h_m est paramétrique par définition on peut chercher les paramètres a_m qui la caractérisent

$$a_m = \arg\min_a \sum_{i=1}^{n} \left(\frac{\partial L\left(y_i, F_{m-1}(\mathbf{x}_i)\right)}{\partial F_{m-1}(\mathbf{x}_i)} - h(\mathbf{x}_i, a) \right)^2.$$

On cherche le pas optimal β_m

$$\beta_m = \arg\min_\beta \sum_{i=1}^{n} L\left(y_i, F_{m-1}(\mathbf{x}_i) + \beta h_m(\mathbf{x}_i)\right).$$

Finalement ces deux étapes correspondent bien à l'itération m décrite par (9) et (10). On note $\forall i \in \{1, \ldots, n\}$ et $\forall m \in \{1, \ldots, M\}$, $\tilde{y}_{i,m} = -\frac{\partial L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)}$, que l'on dénomme les pseudo-résidus.
Pour résumer l'algorithme de Gradient Boosting on pourra se référer au pseudo-code 1.

Dans notre cadre et les simulations suivantes, nous ne considérons que des fonctions de perte convexes. Cependant il faut garder à l'esprit que le Boosting est une approche qui s'applique à des fonctions de perte non convexes. Voir par exemple l'algorithme Brownboost introduit par FREUND en 2001 [8].

Algorithm 1: Gradient Boosting

But : minimisation de (3)
Initialisation : $F_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$; `/* meilleure constante d'approx. */`
for $m = 1$ **to** M **do**
  $\tilde{y}_{i,m} \leftarrow -\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\Big|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \forall 1 \leq i \leq n$; `/* pseudo-résidus */`
  $\mathbf{a}_m \leftarrow \arg\min_{\mathbf{a},\beta} \sum_{i=1}^n (\tilde{y}_{i,m} - \beta h(\mathbf{x}_i; \mathbf{a}))^2$;
  $\beta_m \leftarrow \arg\min_\beta \sum_{i=1}^n L(y_i, F_{m-1}(\mathbf{x}_i) + \beta h(x_i, a))$;
  $F_m \leftarrow F_{m-1} + \beta_m h(\mathbf{x}; \mathbf{a}_m)$; `/* mise à jour */`
end
Sortie : F_M

L'algorithme de Gradient Boosting est paramétré par :

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

Nous introduisons à présent un paramètre d'aléa f.

2.2.2 Gradient Boosting stochastique

Dans [9], FRIEDMAN propose d'injecter de l'aléa dans l'algorithme de Gradient Boosting. À chaque itération, l'apprenant s'entraîne sur $\tilde{n}$ données tirées aléatoirement et sans remise parmi le jeu d'entraînement $\mathcal{D}_n$. L'algorithme 2 résume ce procédé.

Algorithm 2: Gradient Boosting stochastique

But : minimisation de (3)
Entrée : proportion f de données : $\tilde{n} = f \times n$;
Initialisation : $F_0(\mathbf{x}) \leftarrow \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$; `/* meilleure constante d'approx. */`
for $m = 1$ **to** M **do**
  $\mathcal{D}_{\tilde{n}} \leftarrow$ sous échant. aléatoire de taille $\tilde{n} = f \times n$;
  $\tilde{y}_{i,m} \leftarrow -\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\Big|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})}, \forall \mathbf{x}_i \in \mathcal{D}_{\tilde{n}}$; `/* pseudo-résidus */`
  $\mathbf{a}_m \leftarrow \arg\min_{\mathbf{a},\beta} \sum_{\mathbf{x}_i \in \mathcal{D}_{\tilde{n}}} (\tilde{y}_{i,m} - \beta h(\mathbf{x}_i; \mathbf{a}))^2$;
  $\beta_m \leftarrow \arg\min_\beta \sum_{\mathbf{x}_i \in \mathcal{D}_{\tilde{n}}} L(y_i, F_{m-1}(\mathbf{x}_i))$;
  $F_m \leftarrow F_{m-1} + \beta_m h(\mathbf{x}; \mathbf{a}_m)$; `/* mise à jour */`
end
Sortie : F_M

Le paramètre $f \in [0; 1]$, ajuste la part d'aléa.

- $f = 1$ correspond à la version déterministe.
- $0 < f < 1$ représente la fraction de données sous-échantillonnées, i.e. $\tilde{n} = \lfloor fn \rfloor$.

FRIEDMAN remarque que l'ajout d'aléa peut améliorer significativement les performances de l'algorithme. Utiliser $\tilde{n} = \lfloor fn \rfloor$ réduit le coût de calcul d'un facteur f. Cependant choisir f trop petit réduit la taille du set d'entraînement ce qui aura pour effet d'augmenter la variance des apprenants faibles. Gradient Boosting est communément utilisé en entraînant des arbres décisionnels CART. C'est le cas de l'algorithme `GradientBoostingRegressor` du module `sklearn.ensemble` implémenté en langage `Python`. Nous nous intéressons à l'algorithme Gradient Tree Boosting. Avant cela on rappelle brièvement le principe de l'arbre CART.

2.3 Boosting par arbres prédicteurs

On considère le cas $\mathcal{H} = \{\text{arbres prédicteurs à } L \text{ nœuds}\}$. Nous introduisons plus formellement les arbres :

Définition 2.2 (Arbre prédicteur - [4], [7], [10])

Soient $\mathbf{x} \in \mathcal{X}$, et $\mathcal{D}_n$ les données d'entraînement.
Un *arbre* est un *unique* prédicteur $\mathbf{x} \mapsto h(\mathbf{x}; \mathbf{a})$, constant par morceaux et obtenu par partitionnement récursif dyadique aléatoire. Le paramètre $\mathbf{a}$ d'un arbre $h(\cdot, \mathbf{a})$ caractérise les points de découpe de l'espace des données $\mathcal{X}$ en $L \geq 1$ régions $(R_l)_{l=1}^{L}$ correspondant aux feuilles (nœuds terminaux). En particulier, un *arbre de régression* est un prédicteur admettant la forme suivante :

$$h(\mathbf{x}; \mathbf{a}, \mathcal{D}_n) = \sum_{l=1}^{L} \bar{y}_l \mathbb{1}_{\{\mathbf{x} \in R_l\}} \tag{13}$$

avec $\bar{y}_l = \frac{\sum_{\mathbf{x}_i \in R_l} y_i}{|R_l|}$, moyenne empirique sur la région R_l.

Remarque.
- Dans le cadre d'une *classification*, chaque arbre indique la classe la plus vraisemblable à laquelle appartient une donnée $\mathbf{x} \in \mathcal{X}$, par vote majoritaire ;
- Puisque nous serons amenés à construire et utiliser de façon itérative ces arbres, on adoptera la notation suivante : à chaque itération $m \in [\![1; M]\!]$, on construit l'arbre $h_m = h(\cdot, \mathbf{a}_m)$ paramétré par $\mathbf{a}_m$ et nous fournissant un partitionnement en L régions $\{R_{l,m}\}_{l=1}^{L}$ de $\mathcal{X}$.

2.3.1 Arbre de régression CART

Pour une introduction en détail des arbres CART, on se référera à [6].
L'algorithme **CART** (Classification And Regression Trees) a été développé en 1984 par BREIMAN, FRIEDMAN, OLSHEN et STONE. Il permet de concevoir de façon *simple* et *rapide* des *estimateurs* constants par morceaux (par histogramme).

À chaque étape de construction de l'arbre, un nœud (ou autrement dit sa cellule) est sub-divisé en 2 parties (fils) selon les 3 étapes suivantes :

1. **Ré-échantillonage** de notre ensemble $\mathcal{D}_n$ en $\mathcal{D}'_n$ par *boostrap* (avec répétitions possibles) ou *sous-échantillonnage* (sans remises) en effectuant $t \in [\![1; n]\!]$ tirages (hyperparamètre à fixer initialement).
   On a donc $|\mathcal{D}_n^{'}| = t \leq n$;
2. **Splitting rule** en appliquant le *critère CART* la découpe se fait sur une coordonnée parmi les d possibles. On appellera $\mathcal{E}_{\text{dir}}$ l'ensemble des coordonnées de découpes possibles ;
3. **Stopping rule** pour laquelle on arrête l'algorithme lorsque l'on atteint le nombre minimal d'observations par cellule (hyperparamètre fixé préalablement). En général (en particulier lors d'implémentations sous `R` et `Python`), on demande à ce que chaque feuille de l'arbre contienne entre 1 et 5 observations.

Sans perte de généralité et pour faciliter la compréhension du *critère CART*, on considère, dans les lignes qui suivent, que $\mathcal{D}_n^{'} = \mathcal{D}_n$ (pas de ré-échantillonnage).
Considérons $A \subset \mathcal{X}$ une cellule quelconque (à n'importe quelle itération du processus de partitionnement) qui sera caractérisée par $|A|$, le nombre d'observations $\mathbf{X}_i = (X_i^{(1)}, \ldots, X_i^{(d)}) \in \mathcal{X}$ présentes dans A (i.e. son cardinal).
Soit $j \in \{1, \ldots, d\}$ une direction de découpe de $\mathcal{E}_{\text{dir}}$, et $z \in [0; 1]$ une position de découpe sur cette coordonnée j.
On pose $\mathcal{E}_{\text{cut}} := \{(j, z) \in [\![1; d]\!] \times [0; 1] | j \in \mathcal{E}_{\text{dir}}\}$ l'ensemble de tous les couples (j, z) de découpes possibles. Le critère de CART est le suivant :

$$L(j, z) = \frac{1}{|A|} \sum_{i=1}^{n} \left(Y_i - \bar{Y}_A\right)^2 \mathbb{1}_{\{\mathbf{X}_i \in A\}} - \frac{1}{|A|} \sum_{i=1}^{n} \left(Y_i - \bar{Y}_{A_L} \mathbb{1}_{\left\{X_i^{(j)} < z\right\}} - \bar{Y}_{A_R} \mathbb{1}_{\left\{X_i^{(j)} \geq z\right\}}\right)^2 \tag{14}$$

où $A_L = \{\mathbf{x} \in A | \mathbf{x}^{(j)} < z\}$, $A_R = \{\mathbf{x} \in A | \mathbf{x}^{(j)} \geq z\} = A \backslash A_L$ et $\bar{Y}_A$ (respectivement $\bar{Y}_{A_L}$ et $\bar{Y}_{A_R}$) étant les moyennes des (Y_i) appartenant à A (respectivement A_L et A_R).

Pour chaque cellule, la meilleure découpe possible $(j^*, z^*) \in \mathcal{E}_{\text{cut}}$ est alors celle *maximisant* le critère (14), i.e. telle que :

$$(j^*, z^*) \in \underset{(j,z)\in\mathcal{E}_{\text{cut}}}{\arg\max}\, L(j, z).$$

Pour résumer, l'arbre de BREIMAN procède à une construction dépendant à la fois des $(\mathbf{X}_i)$ mais également des (Y_i) [1]. Cet arbre se propose ainsi de partitionner nos données $\mathcal{D}_n$ selon le critère CART (14), qui consiste à choisir itérativement les coupures qui correspondent à l'estimateur dont l'*erreur quadratique* est *minimale*.

Exemple 2.2. On clôture cette introduction des arbres de BREIMAN avec un exemple de partitionnement CART sur $\mathcal{X} = [0; 1]^2$ où l'on considère les 10 données $(\mathbf{X}_i, Y_i)_{1\leq i\leq 10} \subset (\mathcal{X} \times \mathbb{R}_+)$ suivantes :

$$((0.08, 0.25), 310), ((0.2, 0.13), 305), ((0.27, 0.4), 340), ((0.31, 0.62), 500), ((0.15, 0.83), 400),$$
$$((0.4, 0.9), 380), ((0.52, 0.6), 100), ((0.68, 0.35), 70), ((0.875, 0.86), 30), ((0.82, 0.74), 5), ((0.87, 0.1), 20).$$

En appliquant l'algorithme CART avec l'arbre de BREIMAN, on peut alors obtenir le partitionnement représenté en Figure 2, où l'ensemble de découpes $\{(1, 0.45), (2, 0.5), (1, 0.75)\} \subset \mathcal{E}_{\text{cut}} = [\![1; 2]\!] \times [0; 1]$ vérifie bien la maximisation du critère (14).

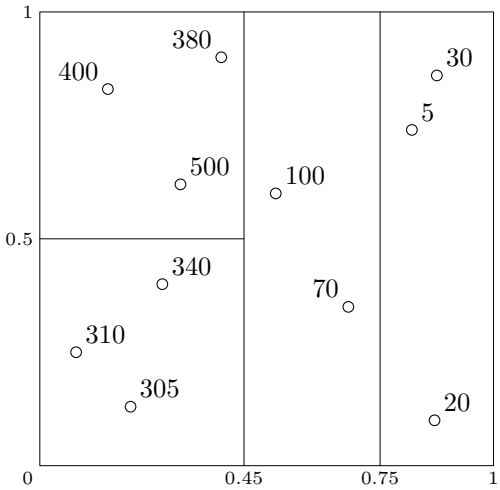

FIGURE 2 – Exemple de représentation d'un arbre de BREIMAN.

2.4 Étapes d'optimisation TreeBoosting

Dans le cas précis où les apprenants de base sont des arbres construisant L partitions de l'espace des observations $\mathcal{X}$, les étapes d'optimisation (8) et (10) peuvent être reformulées :

— Récurrence sur F_m :

$$F_m(\mathbf{x}) = F_{m-1}(\mathbf{x}) + \sum_{l=1}^{L} \gamma_{j,m} \mathbb{1}_{\{\mathbf{x}\in R_{l,m}\}}, \tag{15}$$

avec $\gamma_{l,m} = \beta_m \bar{y}_{l,m}$ le produit entre le coefficient d'expansion et de moyenne locale sur la région donnée, à l'itération m.

— Optimisation sur chaque région $1 \leq l \leq L$, à l'itération m :

$$\gamma_{l,m} = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(Y_i, F_{m-1}(\mathbf{x}_i) + \gamma\right). \tag{16}$$

3 Simulations

Pour une première compréhension du concept de Boosting, on utilisera un jeu de n données $\mathcal{D}_n$ issues d'un modèle très simple basé sur la fonction $x \mapsto \sin(x)$. On se restreint, pour cela à un espace d'observations $\mathcal{X} = [0; 5]$ et

1. Ici l'arbre ne vérifie pas la $\mathbf{X}-$propriété.

$\mathcal{Y} = \mathbb{R}$ de dimension 1. On considère ainsi le modèle de régression introduit en (2) :

$$Y = f(X) + \varepsilon,$$

où $X \sim \mathcal{U}_{[0;5]}, \varepsilon$ un bruit uniforme et centré sur $[-5; 5]$ arrivant avec probabilité $\frac{1}{5}$, nul avec probabilité $\frac{4}{5}$ et $f = \sin$.

Le notebook présente une implémentation de l'algorithme Stochastic Gradient Tree Boost, dont les paramètres sont les suivants :

1. Loss : Choix de la fonction de coût : Least-square, Least absolute deviation, Huber ou Negativ Log likelihood. Nous réalisons des simulations utilisant les deux premières pertes. Une application sur le jeu de données MNIST de classification avec la Negativ Log Likelihood est exposée dans le notebook.
2. learning rate : il s'agit d'un paramètre de shrinkage que l'on applique à chaque apprenant. Il prévient l'overfitting et est typiquement choisit petit (de l'ordre de 10^{-3}).
3. n estimators : Le nombre d'arbres. C'est également un paramètre important. Il est en compromis avec le learning rate.
4. criterion : fixe le critère de découpe pour la construction d'un arbre. Nous choisissons systématiquement 'friedman mse' qui correspond à un arbre CART et nous utilisons l'algorithme `DecisionTreeRegressor` de `sklearn`.
5. max depth : Détermine la profondeur d'un arbre. Le Boosting est réputé performant avec des arbres peu profonds, ce que l'on retrouve dans les simulations. Cependant la nature d'un problème, complexe ou est très grande dimension peu remettre en cause ce principe.
6. max samples : La proportion de données sous-échantillonnée à chaque itération.

Bien entendu ces simulations ont peu de valeur générale mais elles ont pour objectif de faire comprendre l'influence des paramètres et leur sensibilité aux différentes configurations.

3.1 Fonction de perte Least-Squares

Soit la fonction de coût moindres carrés (LS), telle que $L : (x, y) \mapsto \frac{1}{2}\|x - y\|_2^2$.

Dans ce cas précis, le calcul des pseudos-résidus dans l'algorithme 1 s'explicite clairement avec une telle fonction L, de dérivée $\nabla_F L(y, F(\mathbf{x})) = y - F(\mathbf{x})$. Par suite, le Boosting du gradient pour la MSE suit bien l'approche habituelle en ajustant de façon itérative les résidus. En voici le pseudo-code :

Algorithm 3: LS Gradient Boosting

But : minimisation de (3) pour $L = \text{MSE}$
Initialisation : $F_0(\mathbf{x}) = \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$; `/* moyenne empirique */`
for $m = 1$ **to** M **do**
  $\tilde{y}_{i,m} \leftarrow y_i - F_{m-1}(\mathbf{x}_i), \forall\, 1 \leq i \leq n$;
  $(\beta_m, \mathbf{a}_m) \leftarrow \arg\min_{\mathbf{a},\beta} \sum_{i=1}^{n} (\tilde{y}_{i,m} - \beta h(\mathbf{x}_i; \mathbf{a}))^2$;
  $F_m \leftarrow F_{m-1} + \beta_m h(\mathbf{x}; \mathbf{a}_m)$; `/* mise à jour */`
end
Sortie : F_M

Une remarque importante est la suivante : augmenter le nombre de nœuds terminaux pour chaque arbre (apprenant faible) ne permet pas d'accélérer l'apprentissage puisque les courbes en erreur sont similaires à celles obtenues ci-dessus en Figure 4, pour ce modèle particulier.
Le pas d'apprentissage ν est également très important ici, puisqu'en le doublant seulement pour $\nu = 2 \cdot 10^{-3}$, les courbes deviennent instables pour des proportions f de $\mathcal{D}_n$ telles que $f \leq 0.5$. A contrario, le Boosting déterministe $f = 1$ est bien moins sensible aux variations de ν : en augmentant ν jusqu'à 0.08, l'apprentissage reste stable, mais on remarque notamment que l'on atteint rapidement la situation d'*overfitting*, avec la courbe en erreur remontant à partir de la 60e itération (graphe supérieur droit, Figure 3).
En effet, la valeur de f est également responsable de la stabilité de l'entraînement : une petite valeur de f engagera un coût computationnel amoindri puisque moins de données sont utilisées pour entraîner chaque apprenant de base, ceci causant en particulier une augmentation de la *variance* pour chaque arbre ce qui est bien visible pour de petites proportions $f = 0.2$ ou $f = 0.3$.

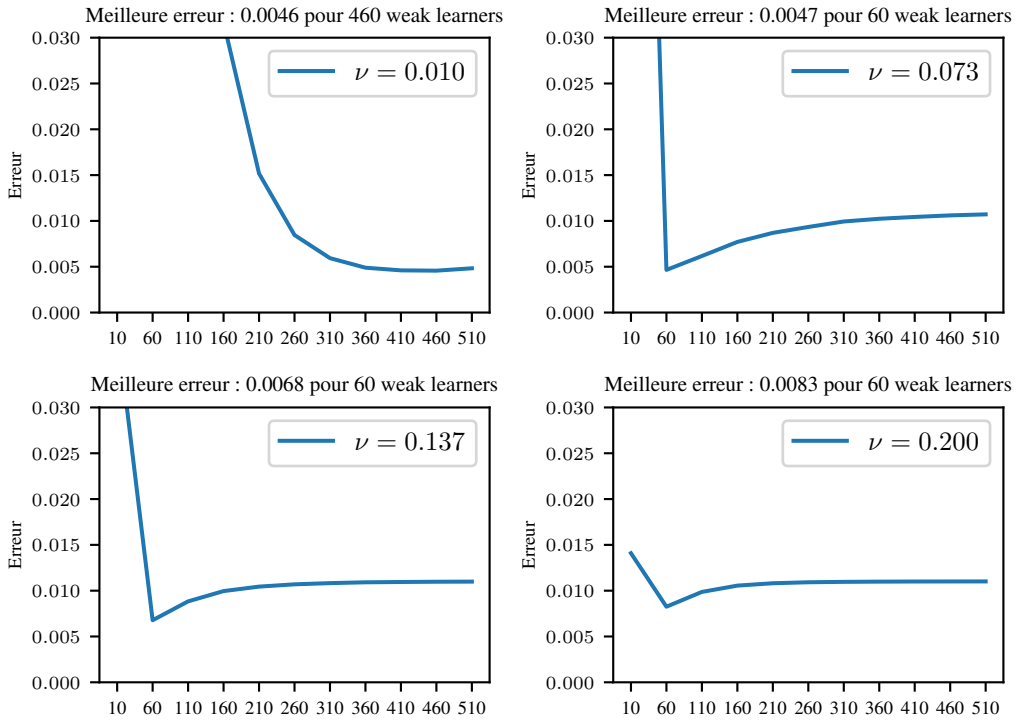

FIGURE 3 – Évaluation du modèle Least Squares Gradient Boosting (LS-TreeBoost) version déterministe ($f = 1$) pour $L = 3$, en fonction du pas d'apprentissage ν.

On peut également remarquer que, assez logiquement, la version déterministe qui prend en considération toutes les données $\mathcal{D}_n$ a besoin de moins d'itérations et donc moins de weak learners pour converger vers une bonne estimation. Le temps d'attente entre chaque construction de weak learner est ainsi très conséquent.

Qualitativement, on représente l'approximation faite par la méthode TreeBoosting pour la fonction de coût moindres carrés, sur la Figure 5.

Comme attendu, le pas d'apprentissage impacte la précision de l'approximation de la fonction cible par les faibles apprenants. La fraction $f = 0.8 < 1$ des données considérées injecte une variance qui se note par une irrégularité dans la prédiction de chaque weak learner (instabilité des courbes en erreur Figure 3). Un pas d'apprentissage trop faible ne permet pas un bon apprentissage comme en atteste le premier graphe dans Figure 5, alors qu'un ν trop grand semble être trop bruité, témoignant alors du phénomène de variance de chaque arbre : un compromis sur le pas d'apprentissage ν est souvent à trouver numériquement pour espérer un bon procédé d'apprentissage. Ce dernier doit être choisi compte tenu d'autres hyper-paramètres tels que la taille de l'arbre $L = 3$ ici, le nombre d'apprenants ainsi que le type de données.
Une analogie peut en particulier être faite avec une descente de gradient usuelle, sur laquelle le choix du pas ν est également crucial : un pas trop faible n'encourage pas une bonne convergence lors de l'apprentissage du modèle alors qu'un ν trop élevé contribue à un phénomène de *gradient explosion*.

3.2 Fonction de perte HUBER

Le but principal des M-régressions est d'être robuste aux faibles déviations, par rapport à la distribution d'un modèle donné, et donc aux outliers. Elle comptabilise aussi l'avantage d'être efficace pour les erreurs "normalement distribuées".
Un bon compromis entre l'efficacité du coût moindre carré (LS) et la robustesse de la distance en valeur absolue (LAD) est la perte de HUBER :

$$L(y, F(\mathbf{x})) = \frac{1}{2}(y - F(\mathbf{x}))^2 \mathbb{1}_{\{|y-F(\mathbf{x})| \leq \delta\}} + \delta \left(|y - F(\mathbf{x})| - \frac{\delta}{2} \right) \mathbb{1}_{\{|y-F(\mathbf{x})| > \delta\}},$$

pour $\delta > 0$ appelé paramètre de *tuning*.

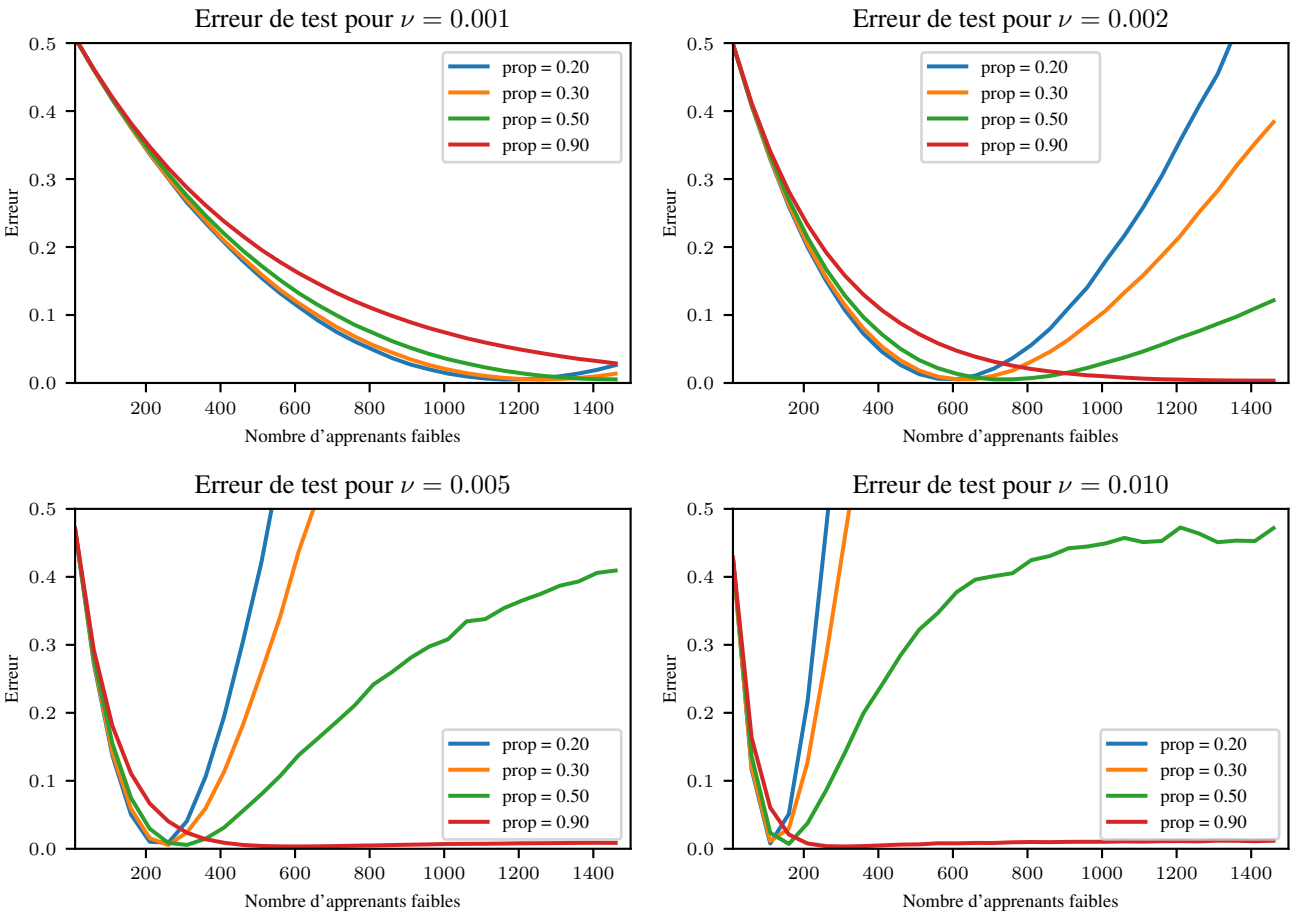

FIGURE 4 – Évaluation du modèle Least Square Gradient Boosting (LS-TreeBoost) version stochastique pour $L = 3$ en fonction du learning rate ν.

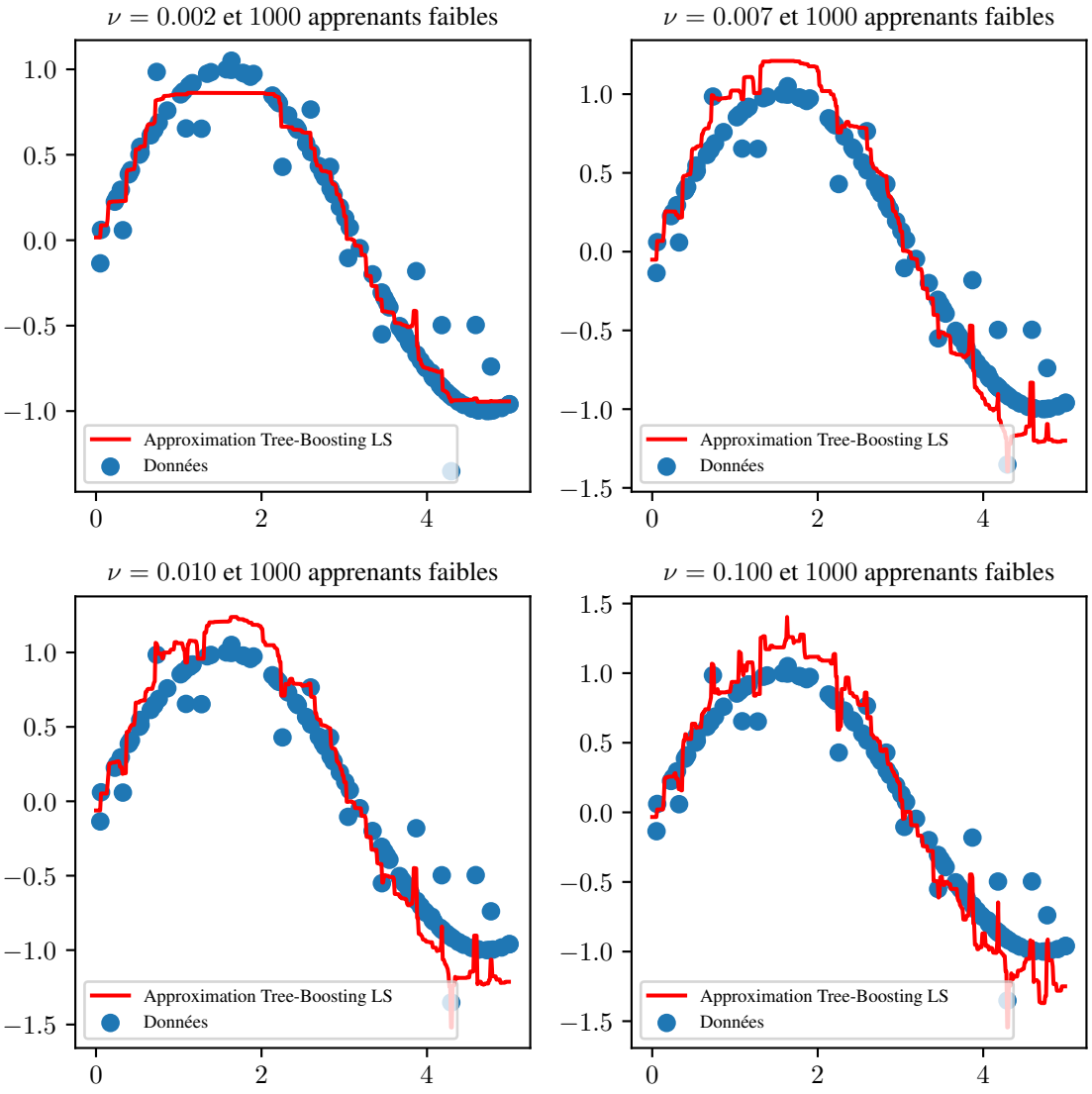

FIGURE 5 – Approximation par LS-TreeBoosting de la fonction régissant $\mathcal{D}_n$ avec 1000 apprenants faiblers sur une proportion $f = 0.8$ de $\mathcal{D}_n$ et $\nu \in \{2 \cdot 10^{-3}; 7 \cdot 10^{-3}; 1 \cdot 10^{-2}; 1 \cdot 10^{-1}\}$.

Remarque. En effet, pour $\delta \to +\infty$, la perte HUBER correspond à la moyenne alors que pour $\delta \to 0$, on retrouve la médiane.

La fonction L est alors *quadratique* pour des valeurs *faibles* des résidus $|y - F(\mathbf{x})|$ et *linéaire* pour de *grandes* valeurs, avec égalité et des pentes égales aux 2 points tels que $|y - F(\mathbf{x})| = \delta$.

Le choix de δ est stratégique : il permet de caractériser la valeur seuil appelée "breakdown point" pour laquelle les résidus $|y - F^*(\mathbf{x})|$ sont considérés comme aberrants ("outliers"). Dans ce dernier cas, la valeur des résidus est alors soumise à une fonction linéaire de type LAD $|y - F^*(\mathbf{x})|$ plutôt que quadratique.
Une valeur optimale pour δ est alors totalement dépendante de la distribution des données $\mathcal{D}_n$ et plus précisément de $y - F^*(\mathbf{x}), \forall \mathbf{x} \in \mathcal{D}_n$, où F^* est notre fonction cible dans (3). Une bonne façon de caractériser ce seuil d'"outliers" pour les valeurs résiduelles est alors d'utiliser les fonctions quantiles.

3.2.1 Rappels sur les quantiles

Les quantiles permettent de caractériser la distribution d'une variable. En effet, ces paramètres contiennent des informations sur la concentration d'une variable. Notons que la médiane est un cas particulier des quantiles.

Définition 3.1 (Fonction quantile)

Pour une variable aléatoire réelle Z de fonction de répartition $z \mapsto \mathbb{P}(Z \leq z)$, le quantile d'ordre α est donné par :

$$q_\alpha(Z) = \inf\{z \in \mathbb{R} | \mathbb{P}(Z \leq z) \geq \alpha\}.$$

En particulier, on peut également définir le quantile α comme la solution du problème de minimisation

$$\arg\min_{\alpha \in \mathbb{R}} \mathbb{E}\left[f_\tau(Z - \alpha)\right],$$

où $f_\tau : x \mapsto \tau|x|\mathbb{1}_{\{x>0\}} + (1-\tau)|x|\mathbb{1}_{\{x<0\}}$ est appelée fonction de perte quantile.

Remarque. L'intervention d'un quantile dans une fonction de perte est particulièrement astucieuse si l'on souhaite créer des seuils de sensibilité ou des domaine de pénalisation plus importante.

3.2.2 Huber-quantile pour la M-régression

Pour plus de détails à propos de cette partie, on pourra se référer à la récente référence [1].
Les quantiles sont donc à la fois de bons témoins de la concentration d'une variable et de ses valeurs aberrantes et donc il fait foi de l'utiliser comme "breakdown point" pour délimiter un potentiel domaine d'aberration.
On choisit ainsi $\delta = q_\alpha$ le quantile d'ordre α de la distribution de $|y - F^*(\mathbf{x})|$, tel que $(1-\alpha)$ contrôle justement la probabilité de franchir le "breakdown point".
Comme F^* est la fonction cible inconnue, il sera alors difficile d'évaluer le quantile q_α tel que défini. On utilisera donc pour cela l'estimation F_{m-1} de F^* à chaque itération $1 \leq m \leq M$, et le quantile q_α peut ainsi être estimer empiriquement de la façon suivante :

$$\delta_m = \hat{q}_{\alpha,m},$$

où $\hat{q}_{\alpha,m}$ est le quantile empirique déterminé sur les données $\{|y_i - F_{m-1}(\mathbf{x}_i),\ \forall\, 1 \leq i \leq n|\}$

En revenant à l'algorithme de Boosting 1, le calcul des pseudos-résidus s'explicite alors de la sorte :

$$\begin{aligned}
\tilde{y}_i &= -\left.\frac{\partial L(y_i, F(\mathbf{x}_i))}{\partial F(\mathbf{x}_i)}\right|_{F(\mathbf{x})=F_{m-1}(\mathbf{x})} \\
&= (y_i - F_{m-1}(\mathbf{x}_i))\mathbb{1}_{\{|y_i - F_{m-1}(\mathbf{x}_i)| \leq \delta\}} + \delta\,\mathrm{sign}(y_i - F_{m-1}(\mathbf{x}_i))\mathbb{1}_{\{|y_i - F_{m-1}(\mathbf{x}_i)| > \delta\}} \\
&= r_{m-1}(\mathbf{x}_i)\mathbb{1}_{\{|r_{m-1}| \leq \delta\}} + \delta\,\mathrm{sign}(r_{m-1})\mathbb{1}_{\{|r_{m-1}| > \delta\}},
\end{aligned}$$

en notant les pseudos résidus $r_{m-1}(\mathbf{x}_i) = y_i - F_{m-1}(\mathbf{x}_i)$.

Pour l'étape de minimisation (16) dans le cas HUBER, la solution peut directement être approchée de la sorte, en partant de $F_0 = \text{médiane}$ et en raisonnant itérativement sur les résidus $|y - F_{m-1}(\mathbf{x})|$, pour chaque itération m :

$$\tilde{r}_{l,m} = \text{médiane}_{\mathbf{x}_i \in R_{l,m}}\{r_{m-1}(\mathbf{x})\}.$$

L'approximation de (16) est alors :

$$\gamma_{l,m} = \tilde{r}_{l,m} + \frac{1}{|R_{l,m}|} \sum_{\mathbf{x}_i \in R_{l,m}} \text{sign}(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{l,m}) \min(\delta_m, |r_{m-1}(\mathbf{x}_i) - \tilde{r}_{l,m}|), \forall 1 \leq l \leq L$$

L'algorithme 4 explicite la méthode de Gradient Boosting dans le cas d'une perte HUBER.

Algorithm 4: Huber M-TreeBoost

But : minimisation de (3) pour $L = \text{Huber}$
Initialisation : $F_0(\mathbf{x}) = \text{médiane}(y_i)_{i=1}^n$;
for $m = 1$ **to** M **do**
  $r_{m-1}(\mathbf{x}_i) \leftarrow y_i - F_{m-1}(\mathbf{x}_i), \forall 1 \leq i \leq n$;
  $\delta_m \leftarrow \text{quantile}_\alpha(|r_{m-1}(\mathbf{x}_i)|)_{i=1}^n$;
  $\tilde{y}_{i,m} \leftarrow \begin{cases} r_{m-1}(\mathbf{x}_i), & |r_{m-1}(\mathbf{x}_i)| \leq \delta_m \\ \delta_m \,\text{sign}(r_{m-1}(\mathbf{x}_i)), & |r_{m-1}(\mathbf{x}_i)| > \delta_m \end{cases}, \forall 1 \leq i \leq n$;
  $\{R_{l,m}\}_{l=1}^L \leftarrow$ nœuds terminaux de l'arbre opérant sur $\{\tilde{y}_i, \mathbf{x}_i\}_{i=1}^n$;
  $\tilde{r}_{l,m} = \text{médiane}_{\mathbf{x}_i \in R_{l,m}}(r_{m-1}(\mathbf{x}_i)), \forall 1 \leq l \leq L$;
  $\gamma_{l,m} \leftarrow \tilde{r}_{l,m} + \frac{1}{|R_{l,m}|} \sum_{\mathbf{x}_i \in R_{l,m}} \text{sign}(r_{m-1}(\mathbf{x}_i) - \tilde{r}_{l,m}) \min(\delta_m, |r_{m-1}(\mathbf{x}_i) - \tilde{r}_{l,m}|), \forall 1 \leq l \leq L$;
  $F_m \leftarrow F_{m-1} + \sum_{l=1}^L \gamma_{l,m} \mathbb{1}_{\{\mathbf{x} \in R_{l,m}\}}$
end
Sortie : F_M

3.2.3 Application au modèle de régression

Reprenons le modèle de régression (2). On obtient alors les résultats en Figure **??** pour la perte HUBER.

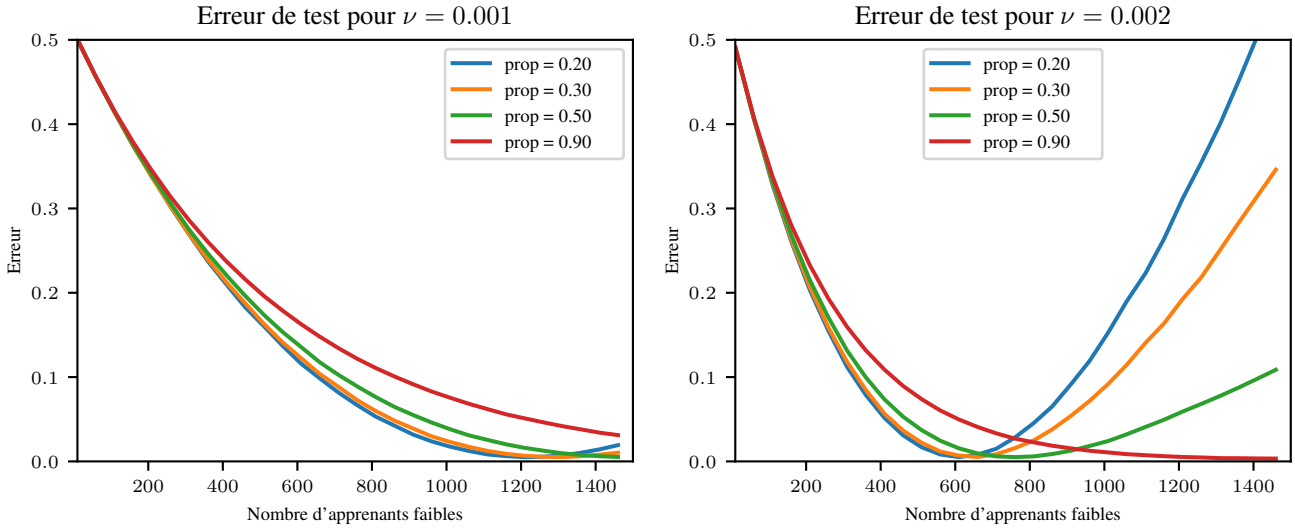

FIGURE 6 – Évaluation du modèle HUBER Gradient Boosting (Huber-TreeBoost) version stochastique pour $L = 3$ en fonction du learning rate ν.

Ici encore, le paramètre de *shrinkage* ν joue un rôle crucial pour le bon apprentissage de notre modèle d'apprentissage par arbres. Pour $\nu = 2 \cdot 10^{-3}$, on remarque que seule la courbe d'erreur associée à une proportion $f = 0.9$ de $\mathcal{D}_n$ semble toujours décroître. Pour le reste, de seuil d'overfitting semble avoir été franchi.
Par ailleurs, notons que pour le cas d'une fonction de coût Huber, l'hyperparamètre α joue un rôle crucial dans la définition itérative des quantiles empiriques, d'autant que la variance des données $\mathcal{D}_n$ est grande. En effet, le modèle de régression considéré ne met pas en avant la capacité de robustesse que présente la perte HUBER-quantile à l'égard des *outliers* : des données à forte variance aurait montré son intérêt.

4 Conclusion

En conclusion l'algorithme Stochastic Gradient Boosting peut s'appliquer sur un très large champ de problèmes tant de classification que de régression. Nous avons étudié en particulier le Stochastic Gradient Tree Boosting (SGTB) pour des fonctions de pertes convexes bien que la méthode soit généralisable. SGTB hérite des bonnes caractéristiques des arbres tels que la sélection de variables ou l'application à des données peu préparées, tout en en améliorant les performances. Les paramètres importants en jeu sont le nombre d'apprenants, leur profondeur, le learning rate et la part d'aléa.

A l'heure des données massives mais aussi de problèmes nécessitant un afflux constant de nouvelles informations, il serait intéressant de considérer une version en ligne de Gradient Boosting. Si nous devions continuer notre étude elle irait dans ce sens.

Références

[1] Aleksandr Y. Aravkin, Anju Kambadur, Aurelie C. Lozano, and Ronny Luss. Sparse quantile huber regression for efficient and robust estimation, 2014.

[2] Leo Breiman. Bias, variance, and arcing classifiers. Technical report, Tech. Rep. 460, Statistics Department, University of California, Berkeley . . ., 1996.

[3] Leo Breiman. Arcing the edge. Technical report, Technical Report 486, Statistics Department, University of California at . . ., 1997.

[4] Leo Breiman. Random forests. *Machine learning*, 45(1) :5–32, 2001.

[5] Leo Breiman. Consistency for a simple model of random forests. 2004.

[6] Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

[7] Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

[8] Yoav Freund. An adaptive version of the boost by majority algorithm. *Machine learning*, 43(3) :293–318, 2001.

[9] Jerome Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38 :367–378, 02 2002.

[10] Robin Genuer and Jean-Michel Poggi. Arbres cart et forêts aléatoires, importance et sélection de variables. 2017.

[11] Max Kuhn, Kjell Johnson, et al. *Applied predictive modeling*, volume 26. Springer, 2013.

[12] Erwan Scornet. Promenade en forêts aléatoires. *MATAPLI*, 111, 2016.

[13] Erwan Scornet. Random forests and kernel methods. *IEEE Transactions on Information Theory*, 62(3) :1485–1500, 2016.

[14] L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11) :1134–1142, nov 1984.