Machine Learning avancé
Ensemble Learning à base d'arbres

Maxime Jumelle

ESLSCA Big Data - MBA 2

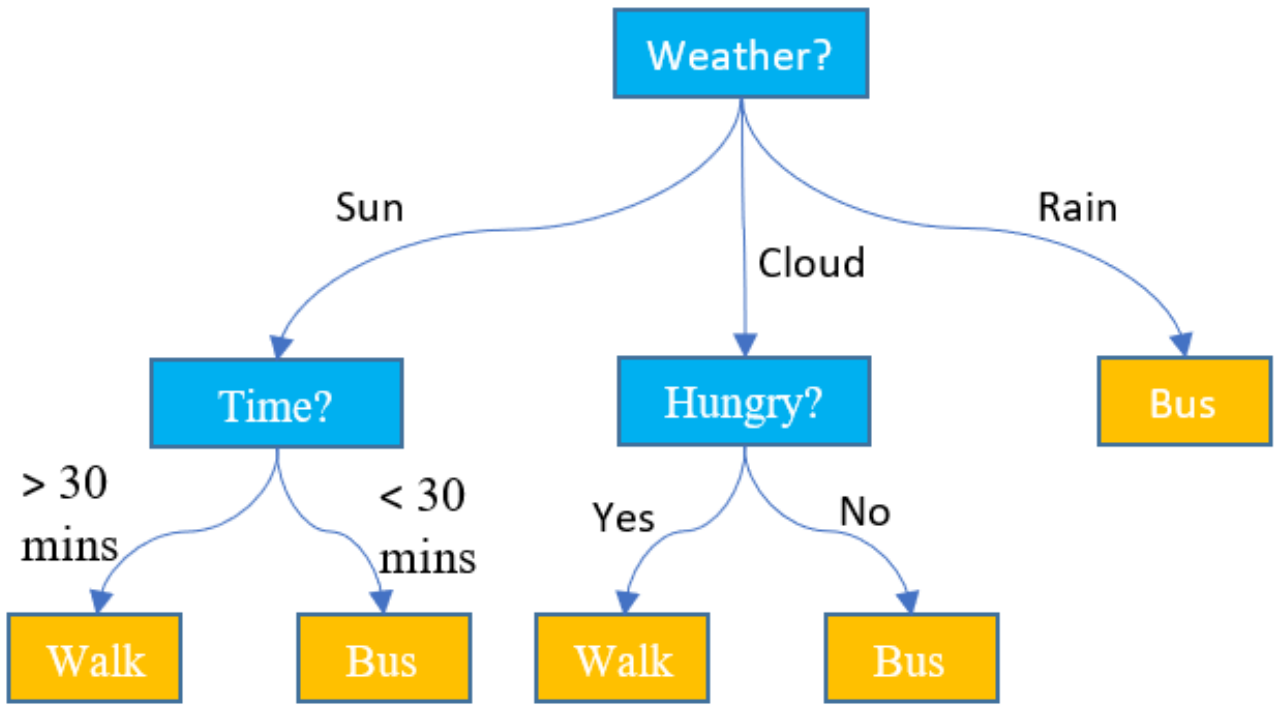
2019 - 2020

Arbres de décision

Parmi les modèles populaires en *Machine Learning*, les arbres de décision forment une classe de modèles non-paramétrique pouvant détecter des relations non-linéaires entre les variables explicatives et la variable réponse. Les travaux des chercheurs sur les deux dernières décennies ont montré qu'empiriquement, les arbres de décision étaient bien adaptés aux méthodes d'*ensemble learning* pour obtenir de bonnes performance sans augmenter significativement la variance.

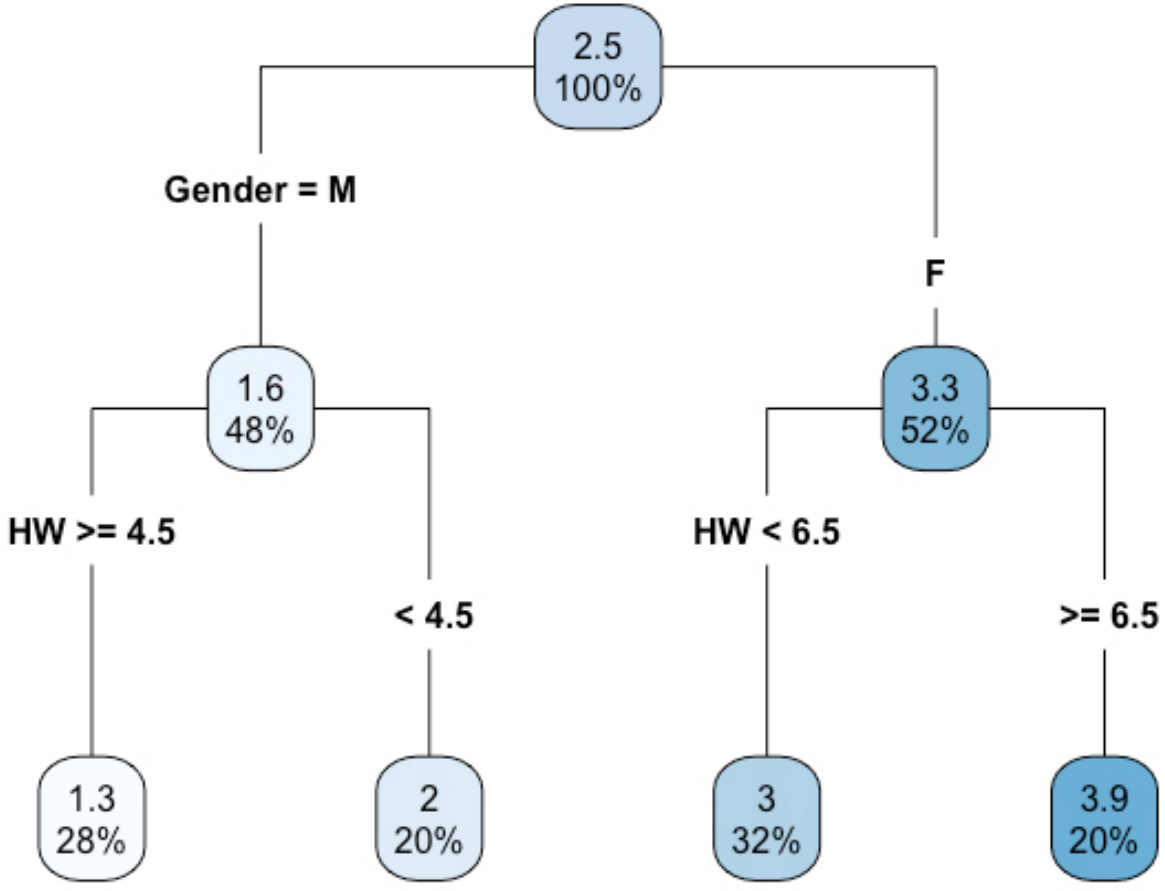
Définition

Un arbre (de décision) est un système qui permet d'aboutir à un résultat par une succession de règles mathématiques. En fonction des possibilités de ces règles mathématiques, les règles ont donc une dépendance récursive. Il est souvent commode de représenter un arbre par un graphe :



Définition

Un arbre (de décision) **binaire** est un arbre qui, à l'issue de chaque règle, ne peut avoir soit 0 possibilité, soit exactement 2 possibilités.



Noeuds

On appelle chaque règle mathématique un **noeud**. Un noeud

- est dit **enfant** s'il est précédé par un noeud ;
- est dit **parent** s'il possède deux enfants ;
- est dit **terminal** s'il ne possède pas d'enfants.

Ainsi, pour chaque noeud parent, en fonction de l'issue de la règle mathématique, "on se dirige" vers un des deux enfants.

Règle

Chaque règle doit être **déterministe** : on doit être capable de connaître exactement l'issue en fonction d'une entrée x fournie. Dans le cas contraire (c'est-à-dire probabiliste), on ne parle plus d'arbre de décision mais d'arbre de probabilité, voir des processus de Markov discrets dans certains cas.

Dans le cas d'un arbre de décision, pour un noeud non terminal, la règle mathématique est une fonction ϕ à valeurs dans $\{0, 1\}$. En fonction de l'issue, le prochain noeud sélectionné est (visuellement) celui de droite ou celui de gauche.
Si le noeud est terminal, alors on renvoie une information, très souvent une valeur numérique.

Règle

Dans le domaine du Machine Learning, on souhaite souvent obtenir des **règles simples**, c'est-à-dire dont le calcul est immédiat pour un algorithme. Le choix s'est naturellement porté sur des **coupures** (ou *split*) selon la valeur d'une entrée **univariée** x. Ainsi, une règle se matérialise généralement par

$$\phi = (x \geq \alpha)$$

avec $\alpha \in \mathbb{R}$ le **split**. Cette condition est ensuite très facilement calculable : si x est effectivement supérieur à α, l'issue de la règle est *Vrai*, sinon, l'issue est *Faux*.

Sommaire

CART

Un **arbre de décision binaire** est une structure qui, par l'intermédiaire d'une succession de règles binaires (à issue positive ou négative) appliquées à une observation, permet d'aboutir à une prédiction. Ces règles formés sont des inégalités $X \leq \alpha, \alpha \in \mathbb{R}$, ce qui permet de segmenter l'espace des variables afin d'approximer la distribution théorique générant les observations. La classe de fonctions des CART est

$$\mathcal{F}_{\text{CART}} = \left\{ \mathbf{x} \mapsto w_{q(\mathbf{x})} | q : \mathbb{R}^d \to T, w \in \mathbb{R}^T \right\},$$

où q représente la structure de l'arbre et T le nombre de feuilles. À chaque nœud, l'arbre de décision sépare l'espace en deux (une partie **gauche** et une partie **droite**) sur une variable seulement.

Apprentissage

L'optimisation des arbres de décisions réside dans trois points :

- Déterminer sur quelle variable effectuer la coupure.
- Trouver la valeur optimale du *split* α.
- Déterminer l'élagage de l'arbre, c'est-à-dire la profondeur optimale.

Cas des arbres de régression

Soient $R_G^j = \{\mathbf{x} : \mathbf{x}^{(j)} \leq \alpha\}$ la région des observations dont la valeur de la j-ème variable est inférieure à α et $R_D^j = \{\mathbf{x} : \mathbf{x}^{(j)} > \alpha\}$ l'autre région.

Pour un arbre de régression, la métrique principalement utilisée est la perte quadratique : dans ce contexte, le critère de minimisation à chaque nœud s'exprime par

$$\min_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} (y_i - \hat{y})^2$$

Cas des arbres de régression

Seulement, α n'apparaît pas dans la fonction de perte. Il faut donc jouer sur les indices en considérant d'une part les points dans R_G^j, et d'autre part les points dans R_D^j.

De plus, puisque l'on souhaite attribuer un poids à chaque noeud, qui ici définira la valeur prédite $\hat{y}$, nous devons introduire c_G et c_D, que l'on interprète comme ceci :

- Tous les points $\mathbf{x}$ qui appartiennent à R_G^j (soit inférieurs à α) auront pour valeur prédite c_G.
- Tous les points $\mathbf{x}$ qui appartiennent à R_D^j (soit strictement supérieurs à α) auront pour valeur prédite c_D.

Cas des arbres de régression

Mais ces poids doivent aussi être optimisés afin d'être calibré selon nos données. Dès lors, nous obtenons une formule beaucoup plus applicable numériquement

$$\min_{\alpha} \left(\min_{c_G \in \mathbb{R}} \frac{1}{|R_G^j|} \sum_{\mathbf{x}_i \in R_G^j} (y_i - c_G)^2 + \min_{c_D \in \mathbb{R}} \frac{1}{|R_D^j|} \sum_{\mathbf{x}_i \in R_D^j} (y_i - c_D)^2 \right)$$

Or, pour une fonction de perte quadratique, c_G et c_D peuvent s'estimer directement :

$$\hat{c_G} = \mathbb{E}[Y|X \in R_G^j] \qquad \hat{c_D} = \mathbb{E}[Y|X \in R_D^j]$$

Et donc le problème se simplifie en

$$\min_{t_1 \in \mathbb{R}} \left(\frac{1}{|R_G^j|} \sum_{x_i \in R_G^j} (y_i - \hat{c_G})^2 + \frac{1}{|R_D^j|} \sum_{x_i \in R_D^j} (y_i - \hat{c_D})^2 \right)$$

Cas des arbres de régression

Preuve

On cherche à résoudre le problème suivant

$$(E_1) : \min_{c \in \mathbb{R}} \sum_{x_i \in R} (y_i - c)^2$$

Considérons tout d'abord le problème (E_2)

$$(E_2) : \min_{c \in \mathbb{R}} \sum_{i=1}^{n} (y_i - c)^2 = \min_{c \in \mathbb{R}} f(c)$$

f est convexe et dérivable sur $\mathbb{R}$ (car somme de fonctions carrés), il existe donc une unique solution c^* au problème (E_2).

$$f'(c^*) = -2 \sum_{i=1}^{n} (y_i - c^*) = 0 \Rightarrow -2 \sum_{i=1}^{n} y_i + 2nc^* = 0$$

Cas des arbres de régression

Preuve

d'où

$$c^* = \frac{1}{n}\sum_{i=1}^{n} y_i = \mathbb{E}(Y)$$

Revenons au problème (E_1). La seule différence est que l'on limite les observations à la région R. La solution $\tilde{c}$ du problème (E_1) s'écrit

$$\tilde{c} = \frac{1}{n}\sum_{x_i \in R} y_i = \frac{1}{n}\sum_{i=1}^{n} y_i \mathbf{1}_{\{x_i \in R\}} = \mathbb{E}(Y|X \in R)$$

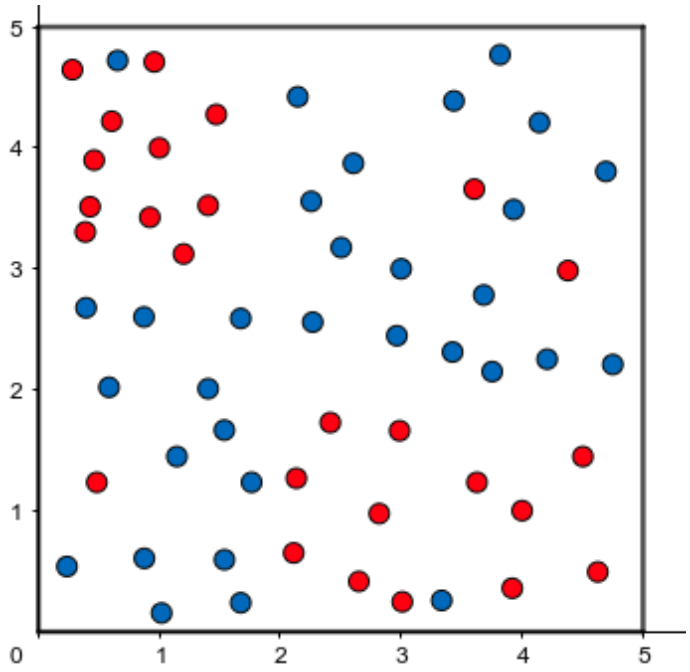
Cas des arbres de régression

Finalement, en minimisant également parmi toutes les variables, le problème d'optimisation d'un CART dans le cas d'une régression s'écrit

$$\min_{1\leq j\leq p} \min_{\alpha} \left(\frac{1}{|R_G^j|} \sum_{\mathbf{x}_i \in R_G^j} (y_i - \hat{c}_G)^2 + \frac{1}{|R_D^j|} \sum_{\mathbf{x}_i \in R_D^j} (y_i - \hat{c}_D)^2 \right)$$

Classification : exemple pas-à-pas

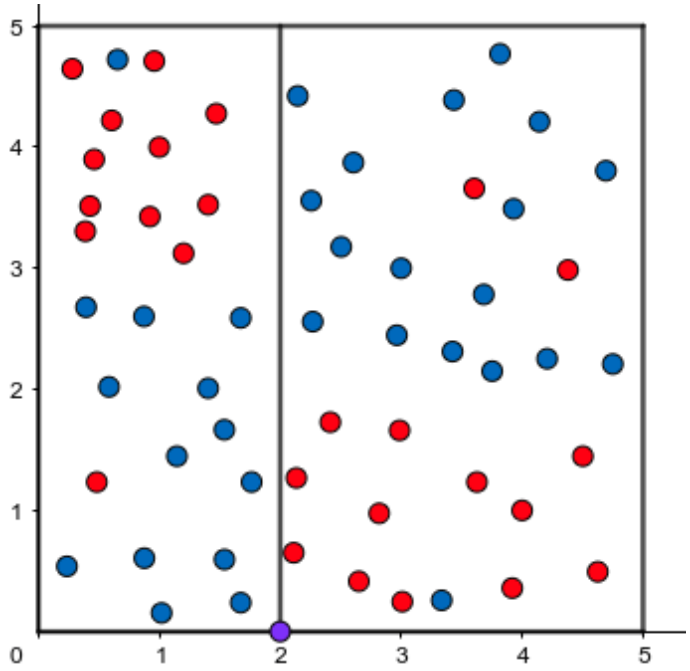
Détaillons un exemple de classification binaire pas-à-pas. On dispose :

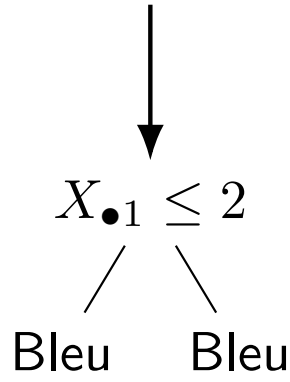


- de n observations $(X_{i1}, X_{i2})_{1 \leq i \leq n} \in \mathbb{R}^2$
- des labels $(Y_i)_{1 \leq i \leq n} \in \{\text{Rouge}, \text{Bleu}\}$

Pour décider de la valeur de sortie du nœud, on utilise un *majority vote* pour les problèmes de classification.

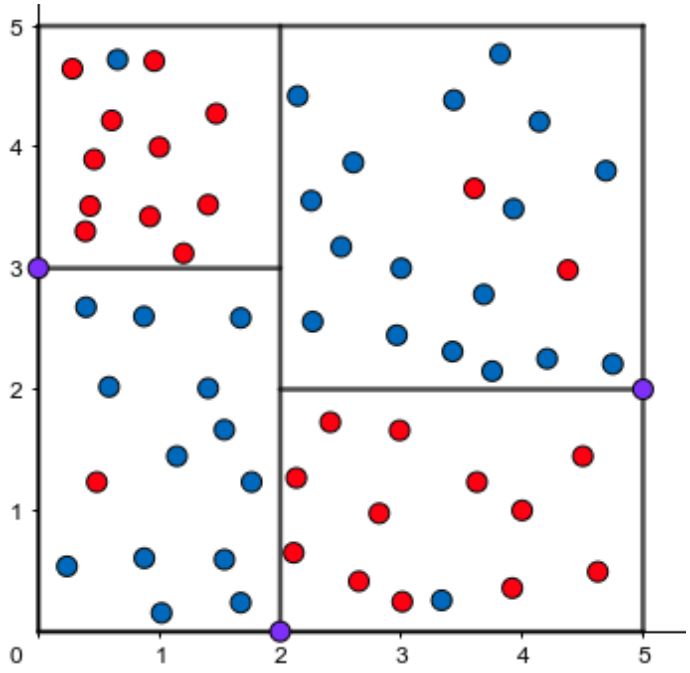
Classification : exemple pas-à-pas

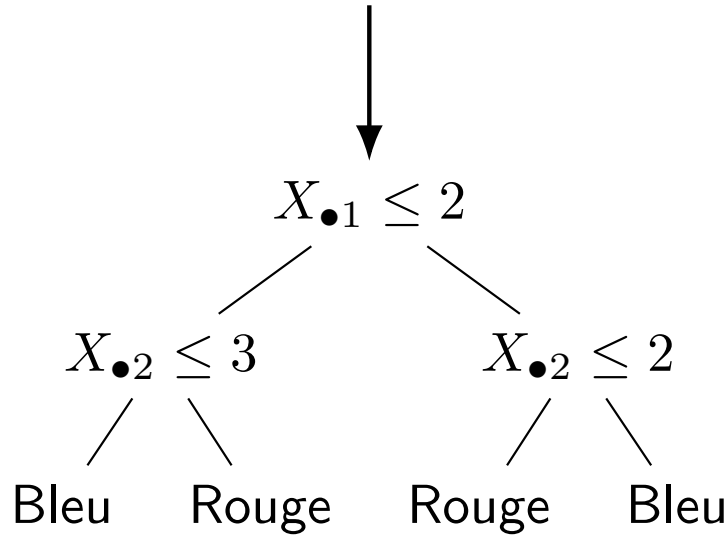




L'arbre n'est pas assez précis avec une profondeur de 1.

Classification : exemple pas-à-pas





La précision devient satisfaisante dès lors que la profondeur vaut 2.

Cas des arbres de classification

Pour un problème de classification, il n'est plus possible d'utiliser la perte quadratique, car celle-ci n'a plus de sens. Considérons un problème de classification multiple à K classes. Deux métriques sont alors principalement utilisées.

- L'impureté de Gini, utilisé par l'algorithme CART.

$$\sum_{k=1}^{K} p_k(1-p_k) \text{ avec } p_k = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{\{y_i=k\}}$$

- Le gain d'information, utilisé par les algorithmes ID3, C4.5 et C5.0.

$$-\sum_{k=1}^{K} p_k \log p_k$$

Cas des arbres de classification

De manière analogue au problème de régression, le problème de minimisation pour la classification suit le même principe.

$$\min_{1\leq j\leq p} \min_{\alpha} \left(\sum_{k=1}^{K} \hat{p}_k^G (1-\hat{p}_k^G) + \sum_{k=1}^{K} \hat{p}_k^D (1-\hat{p}_k^D) \right)$$

$$\text{avec } \hat{p}_k^G = \frac{1}{|R_G^j|} \sum_{\mathbf{x}_i \in R_G^j} \mathbf{1}_{\{y_i=k\}} \text{ et } \hat{p}_k^D = \frac{1}{|R_D^j|} \sum_{\mathbf{x}_i \in R_D^j} \mathbf{1}_{\{y_i=k\}}$$

Le problème de minimisation vise à minimiser conjointement l'impureté de Gini de la région de gauche et de droite. Une fois calibré, il est facile de calculer la prédiction d'une nouvelle observation en évaluant successivement les règles construites automatiquement.

Élagage de l'arbre

Le problème d'un arbre profond est qu'il peut mener à un éventuel **sur-apprentissage**. Ainsi, il est souvent recommandé de pénaliser les arbres profonds qui ne réalisent pas de gains importants par rapport aux arbres moins profonds. Le problème de minimisation voit s'ajouter un terme de pénalisation

$$\min_{1\leq j\leq p} \min_{\alpha} \left(\frac{1}{|R_G^j|} \sum_{\mathbf{x}_i \in R_G^j} (y_i - \hat{c}_G)^2 + \frac{1}{|R_D^j|} \sum_{\mathbf{x}_i \in R_D^j} (y_i - \hat{c}_D)^2 \right) + \gamma |T|$$

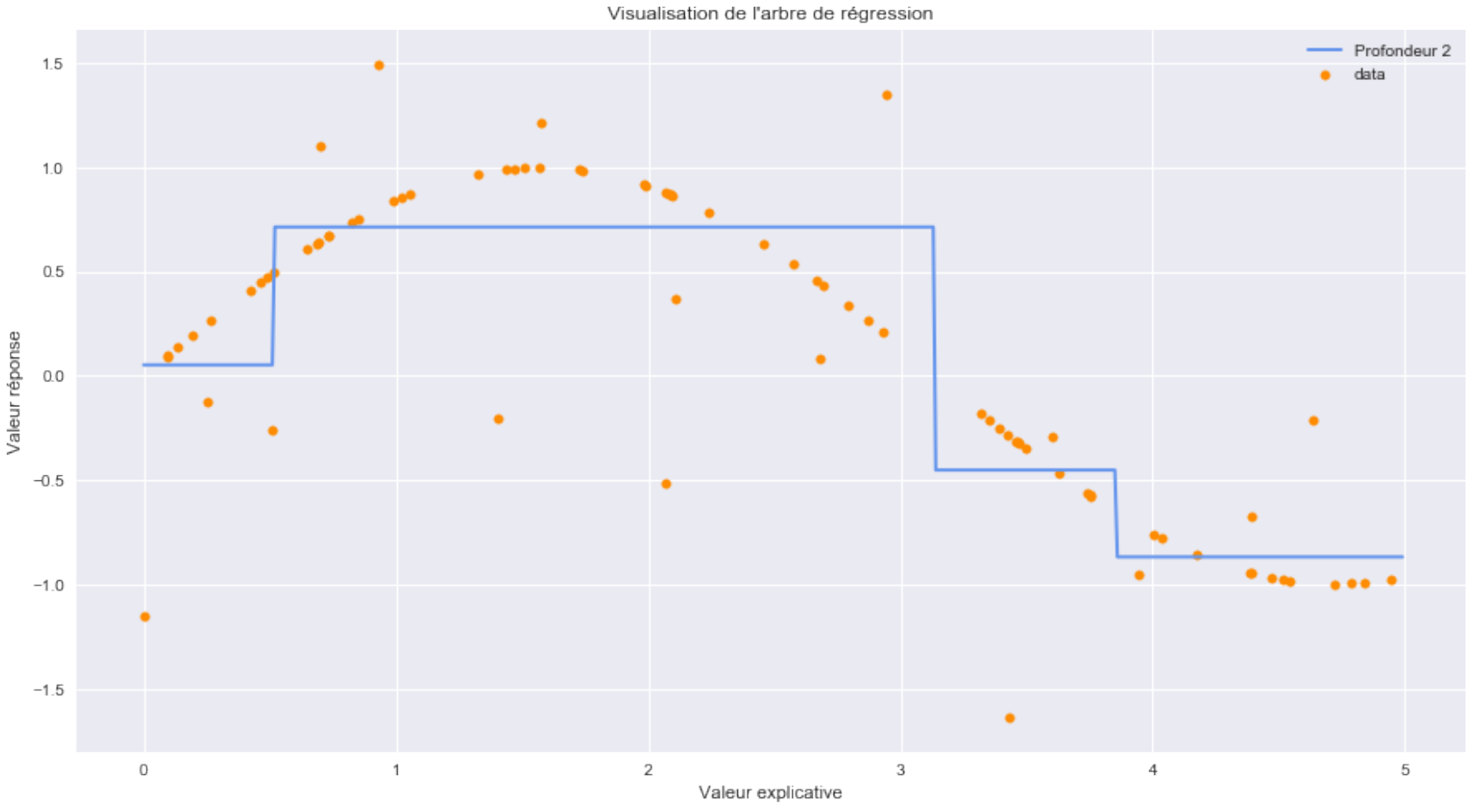
où γ est un paramètre positif et $|T|$ représente le nombre de nœuds terminaux.

Valeur optimale du split

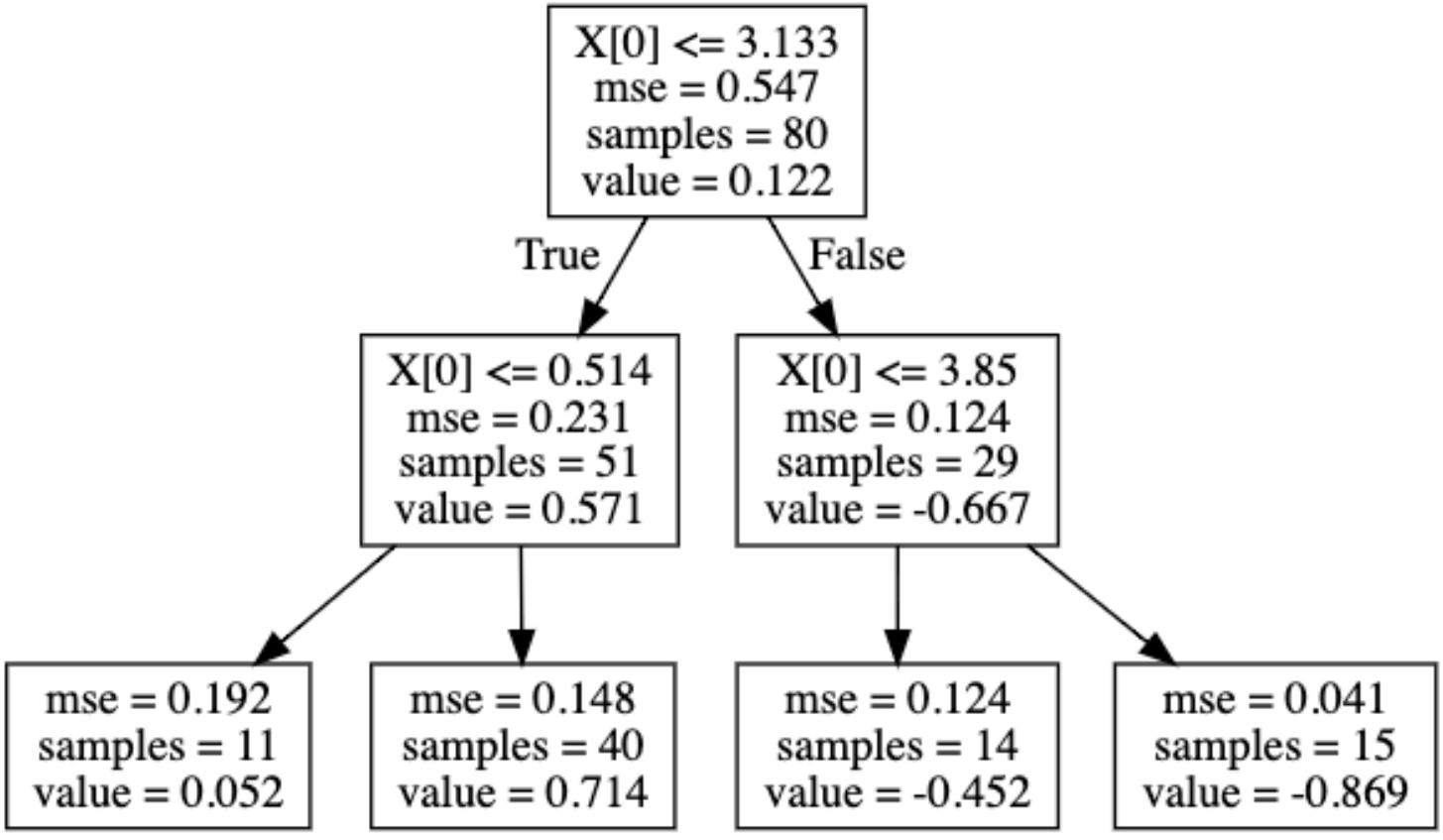
En pratique, trouver la valeur optimale du split n'est pas facile : il conviendrait de tester exactement toutes les valeurs possibles pour split et d'en

Lecture d'un arbre

Considérons l'arbre de régression dont la représentation de la décision est affichée en bleue.



Lecture d'un arbre



Importance des variables

Afin de déterminer quelles sont les variables ayant le plus d'impact sur la décision, il convient de regarder le critère de minimisation employé lors du split, en particulier l'**impureté des noeuds**. Pour tout noeud k non terminal, l'importance de ce noeud correspond au gain d'impureté gagné lors du split pondéré par les populations contenus dans les noeuds enfants.

$$\text{Importance}_k^{\text{Noeud}} = w_k \text{Impureté}_k - w_{\text{left}(k)} \text{Impureté}_{\text{left}(k)} - w_{\text{right}(k)} \text{Impureté}_{\text{right}(k)}$$

où $w_k, w_{\text{left}(k)}$ et $w_{\text{right}(k)}$ représentent la proportion d'observations respectivement contenue dans le noeud parent, et les noeuds enfants gauche et droite.

Importance des variables

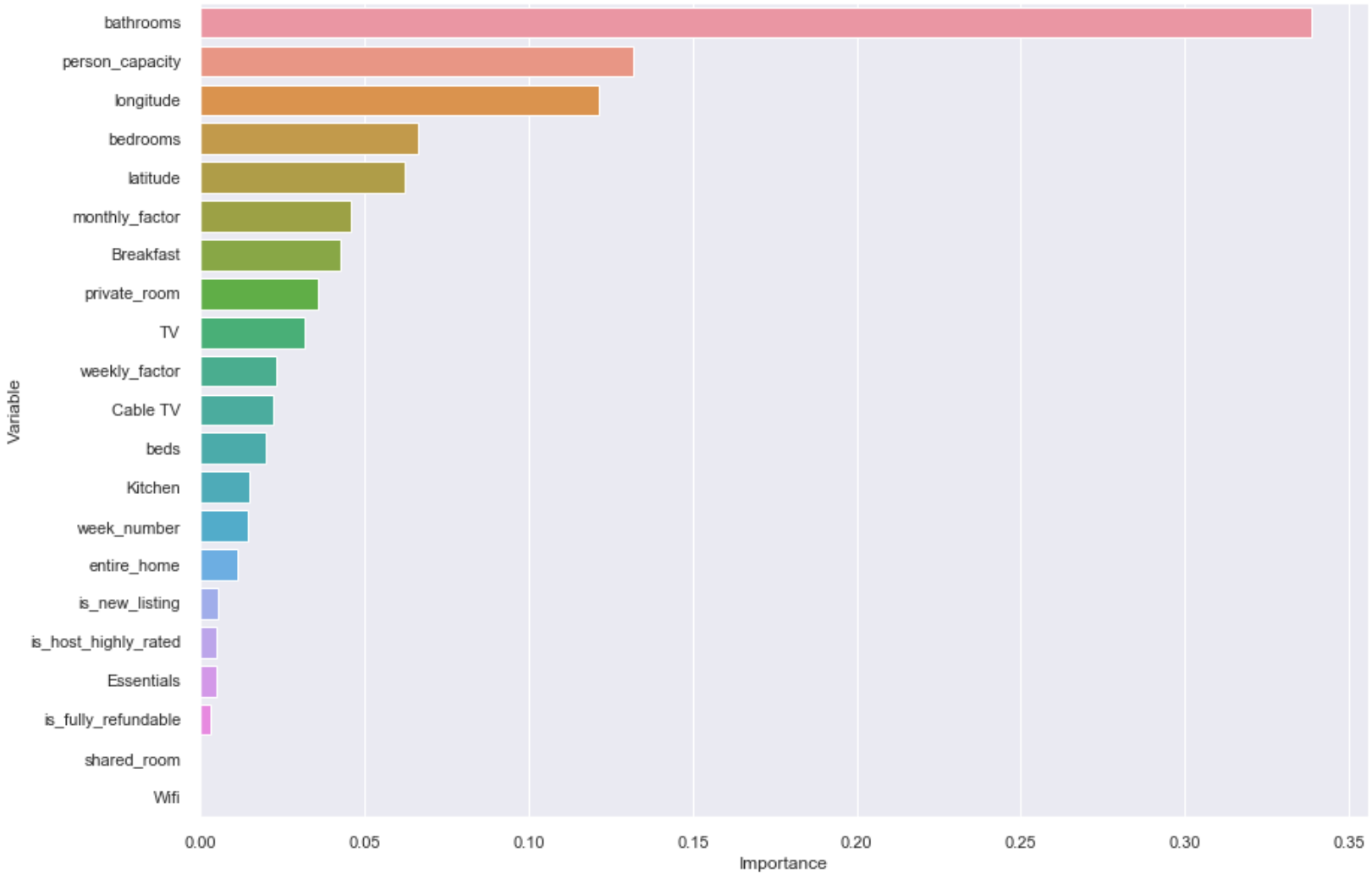
Une fois calculé l'importance pour chaque noeud, l'importance d'une variable j est calculé sur la base de tous les noeuds où le split est effectué sur la j-ème variable.

$$\text{Importance}_j = \frac{\sum_{k:k \text{ split en } j} \text{Importance}_k^{\text{Noeud}}}{\sum_k \text{Importance}_k^{\text{Noeud}}}$$

S'en suit une normalisation fournissant ainsi, pour chaque variable, une importance entre 0 et 1, facilement interprétable.

$$f_j = \frac{\text{Importance}_j}{\sum_m \text{Importance}_m}$$

Importance des variables



Sommaire

Notions de biais et variance

Pour un modèle f, l'erreur commise sur une observation $\mathbf{x}_i$ sous une fonction de perte quadratique peut se décomposer en deux termes :

$$\mathbb{E}[l(y_i, f(\mathbf{x}_i))] = \underbrace{(\mathbb{E}[f(\mathbf{x}_i)] - y_i)^2}_{\text{Biais}(f(\mathbf{x}_i))^2} + \underbrace{\mathbb{E}[f(\mathbf{x}_i) - \mathbb{E}[f(\mathbf{x}_i)]]^2}_{\text{Var}(f(\mathbf{x}_i))}$$

L'équation est appelée **décomposition biais-variance**.

- Le **biais** désigne l'écart moyen constaté entre les prédictions du modèle et la variable réponse. Un biais élevé indique que le modèle n'est pas apte à prédire la variable réponse de manière efficace. Lorsque le biais tend vers 0, le biais minimise les sources d'erreurs **en moyenne**.
- La **variance** indique la dispersion du modèle autour de la moyenne des prédictions. Une variance élevée indique que le modèle est sensible aux petites fluctuations présentes dans l'échantillon d'apprentissage.

Biais et variance

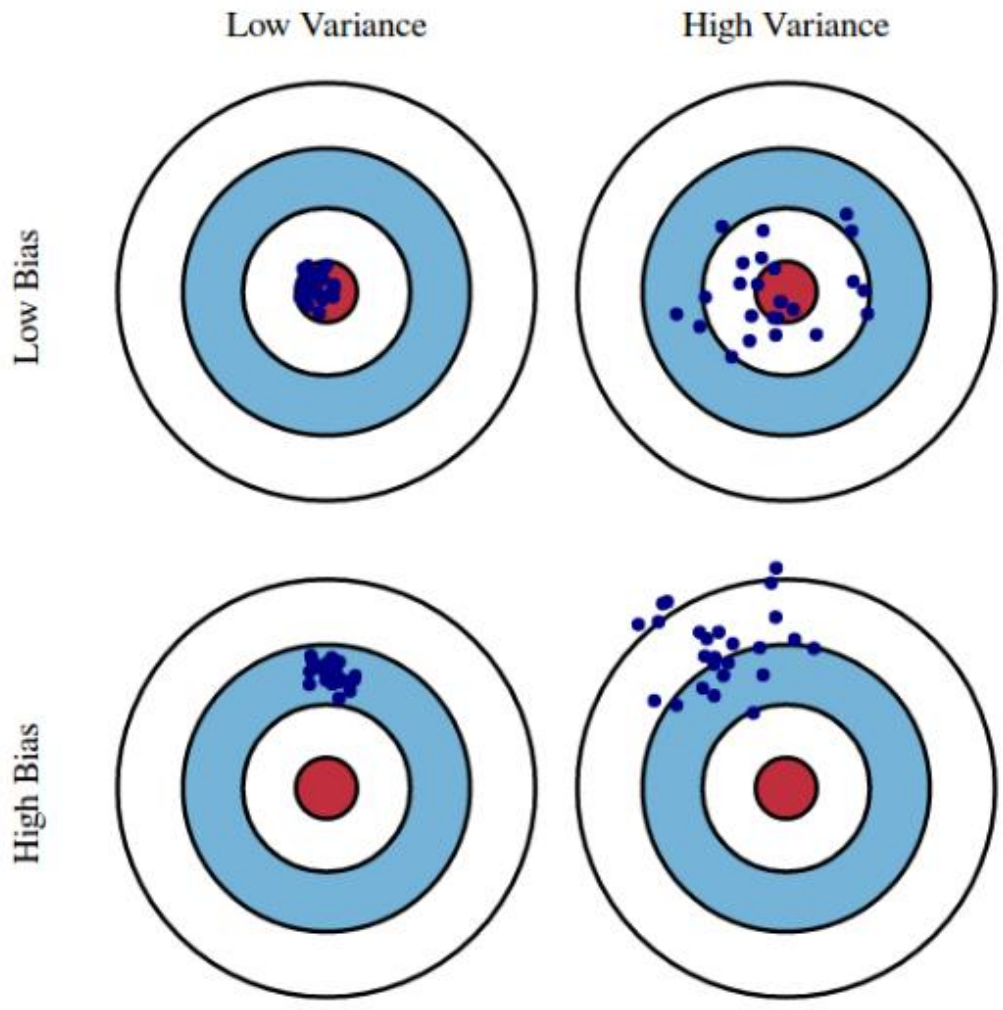
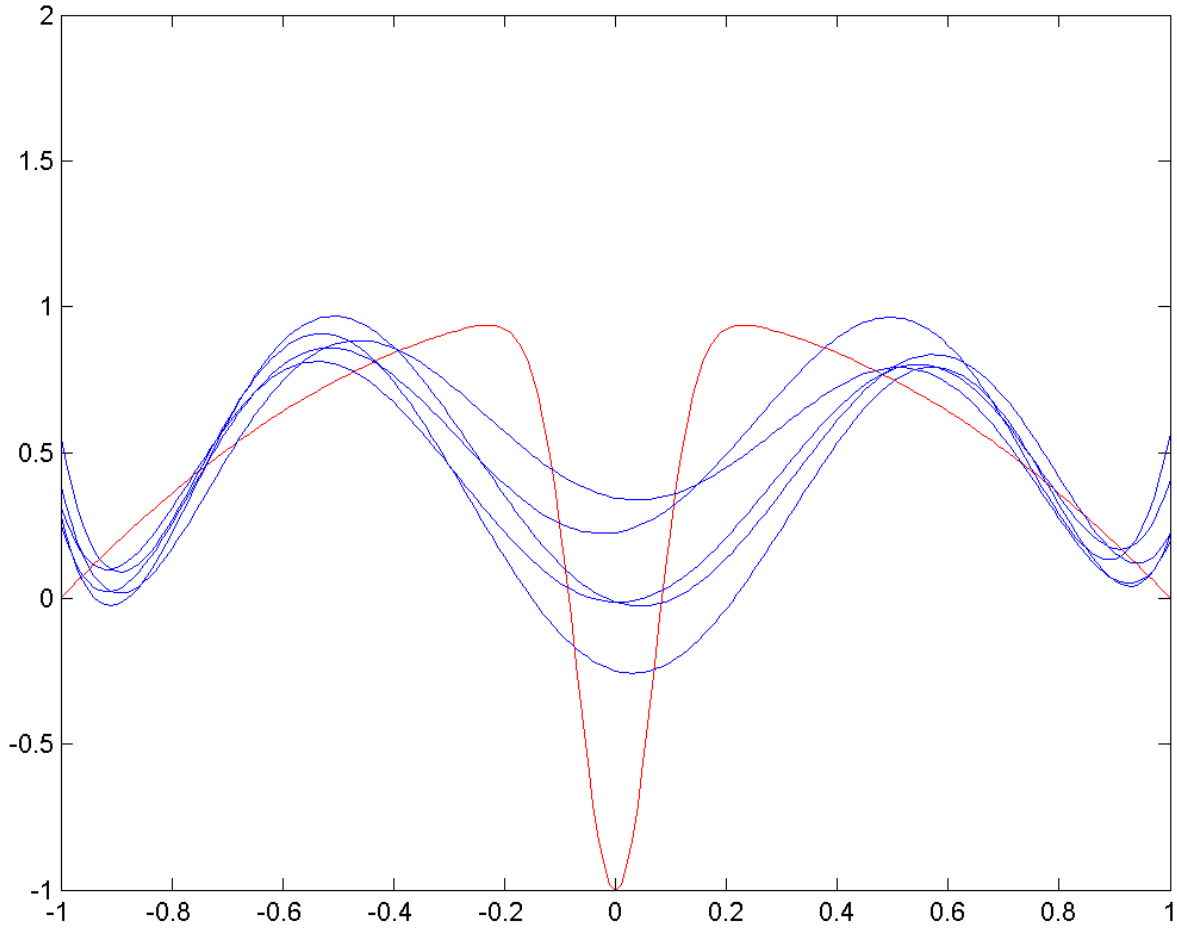


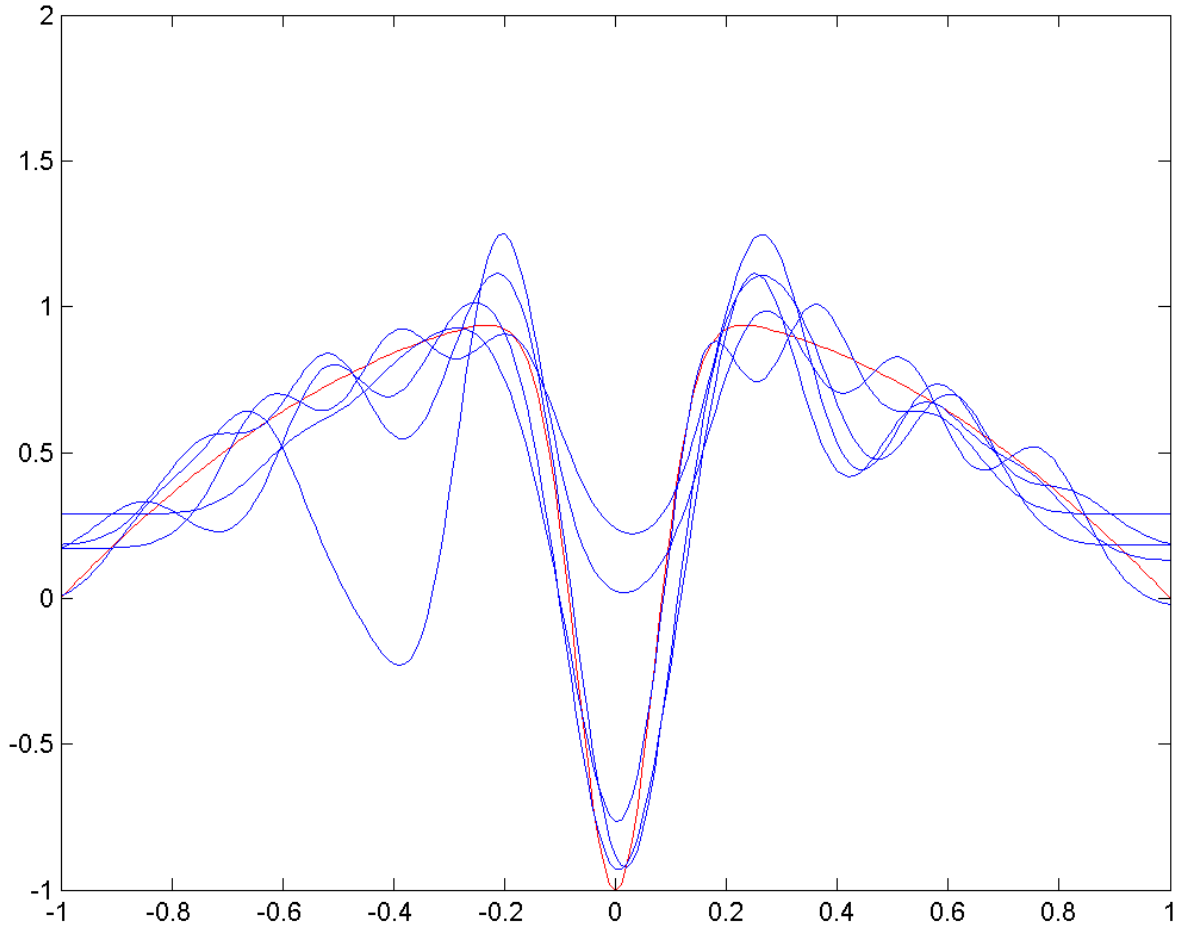
Fig. 1 Graphical illustration of bias and variance.

Dilemme biais-variance



Exemple d'un biais fort et d'une variance faible.

Dilemme biais-variance



Exemple d'un biais faible et d'une variance forte.

Biais et variance

Lorsque le biais est élevé, le modèle est en **sous-apprentissage** : il n'est pas robuste dans sa prédiction puisqu'il ne parvient pas à prédire la variable réponse sur les observations de l'échantillon d'entraînement. Le sous-apprentissage intervient lorsque les hypothèses sur le modèle (linéarité, hyper-paramètres, ...) ne sont pas cohérentes avec la population dont l'échantillon est issu. À l'inverse, une variance forte signifie que le modèle est en **sur-apprentissage** : le modèle cherche à prédire le plus fidèlement possible la variable réponse, au point d'être calibré de manière trop forte sur l'échantillon d'observation en apprenant les bruits. Ainsi, le modèle n'est plus capable de généraliser à de nouvelles observations puisque les bruits ont un comportement par définition aléatoire.

Biais et variance

Face à ces difficultés, les chercheurs ont élaboré de nouvelles techniques d'agrégation afin de pouvoir contrôler simultanément les deux sources d'erreurs, le biais et la variance. Ces modèles forment la famille des algorithmes d'*ensemble learning*, où l'objectif est de construire une modèle agrégé de plusieurs sous-modèles à faibles pouvoir prédictif.

- Le **bagging** (contraction de *bootstrap aggregating*) [**?**, **?**], où plusieurs modèles à faible pouvoir prédictif sont construits de manière **indépendante** sur des sous-échantillons aléatoires de l'échantillon d'entraînement. Les algorithmes de *bagging* permettent de réduire la variance grâce à l'agrégation finale des modèles.
- Le **boosting** [**?**, **?**, **?**] construit récursivement des modèles à faible pouvoir prédictifs, où chacun ont pour objectif de corriger les erreurs commises par les modèles précédents. Le *boosting* est aujourd'hui très populaire et permet de réduire le biais, tout en essayant de maintenir une variance acceptable par l'intermédiaire de termes de régularisation.

Sommaire

Idée générale du boosting

Le **boosting** est une méthode d'apprentissage supervisé qui vise à construire un prédicteur (un méta-modèle) f complexe à partir de prédicteurs f_k dits « faibles » (weak learners). Cette construction est effectuée **par récurrence**, c'est-à-dire que le prédicteur f_k dépends du précédent f_{k-1}. Un modèle de boosting prends la forme d'un **modèle additif généralisé** (GAM), dont la classe de fonctions est représenté par l'ensemble

$$\mathcal{F} = \left\{ f : f(x) = f_0(x) + \sum_{k=1}^{T} \eta_k f_k(x), \begin{pmatrix} \eta_1 \\ \vdots \\ \eta_T \end{pmatrix} \in \mathbb{R}^T, \begin{pmatrix} f_0 \\ \vdots \\ f_T \end{pmatrix} \in \mathcal{F}_{\text{Weak}} \right\}$$

Ici, $\mathcal{F}_{\text{Weak}}$ représente la classe de fonctions des prédicteurs faibles. Lorsque la classe $\mathcal{F}_{\text{Weak}}$ représente les arbres de décisions introduits précédemment, le modèle de boosting est appelé **tree boosting**.

Idée générale du boosting

Une des premières méthodes fonctionnelles de *boosting* est apparue en 1996 avec **AdaBoost** [**?**], contraction de **Adaptive Boosting**. Par la suite, le **Gradient Boosting** [**?**] a été proposé en tant qu'algorithme de boosting qui généralise AdaBoost à des fonctions de perte différentiables quelconques. Enfin, initialement développé en C++ puis formalisé en 2016, XGBoost [**?**] pour eXtreme Gradient Boosting, propose un formalisme légèrement différent au Gradient Boosting. Il est aujourd'hui l'un des modèles les plus populaires et est régulièrement utilisé par la communauté scientifique en *Machine Learning*.

Bien qu'en théorie, il n'y ait aucune restriction sur la nature des classifieurs faibles, les résultats empiriques tendent à montrer que l'utilisation d'arbres de décision à faibles profondeurs s'accorde bien avec le boosting.

Idée générale du boosting

Chaque prédicteur faible f_k possède un biais élevé. Néanmoins, la méthode de construction du méta-modèle agrège les prédicteurs faibles sur des parties de l'espace des caractéristiques de sorte à aboutir à un modèle ayant une performance accrue, tout en réduisant son biais et sa variance.

Initialement, le premier prédicteur faible f_0 est calibré sur les données d'apprentissage $(\mathbf{x}_i, y_i)_{1 \leq i \leq n}$. Ensuite, chaque prédicteur faible f_k a pour objectif de « corriger » les erreurs des prédicteurs faibles précédents : cela permet d'adapter la valeur prédite pour les observations dont l'erreur n'a pas été minimisée (au sens d'une fonction de perte) par les précédents prédicteurs faibles.

Idée générale du boosting

Cette correction s'opère en entraînant les futurs prédicteurs faibles non pas sur la base initiale $(\mathbf{x}_i, y_i)_{1 \leq i \leq n}$, mais en considérant les **résidus** $h(\mathbf{x}_i) = y_i - F_{k-1}(\mathbf{x}_i), \forall 1 \leq i \leq n$ comme variable réponse à prédire. Ainsi, les observations qui ont été correctement prédites jusqu'ici, auront un résidu nul, alors que celles qui n'ont pas été correctement prédites auront un résidu plus ou moins élevé en fonction de l'erreur commise par les précédents prédicteurs faibles. L'algorithme pseudo-code détaille le processus d'entrainement d'un méta-modèle de boosting.

Idée générale du boosting

Algorithme 1 Entraînement générique par boosting

1: **Entrée :** Données d'apprentissage $(\mathbf{x}_i, y_i)_{1 \leq i \leq n}$, nombre d'itérations K
2: Entraînement du prédicteur faible initial $F_0 \stackrel{\Delta}{=} f_0$ sur la base $(\mathbf{x}_i, y_i)_{1 \leq i \leq n}$
3: **for** $k \leftarrow 1$ **to** K **do**
4: Calcul des résidus $h(\mathbf{x}_i) = y_i - F_{k-1}(\mathbf{x}_i)$ pour tout $1 \leq i \leq n$
5: Entraînement du prédicteur faible f_k sur la base $(\mathbf{x}_i, h(\mathbf{x}_i))_{1 \leq i \leq n}$
6: Mise à jour du méta-modèle : $F_k = F_{k-1} + f_k$
7: **end for**
8: **return** F_K

La descente de gradient

Dans ses travaux, Leo Breiman [**?**] mentionna que le *boosting* peut être vu comme un algorithme d'optimisation sur une fonction de perte bien choisie. Par la suite, Jerome H. Friedman [**?**] proposa l'algorithme de Gradient Boosting appliqué à une fonction de perte l différentiable. Le terme *gradient* fait référence au processus d'optimisation utilisé dans le boosting pour le calcul des résidus, la **descente de gradient** (ou *gradient descent* en anglais).

La descente de gradient est un algorithme d'optimisation itératif du premier ordre, particulièrement efficace pour trouver les minimums globaux sur des fonctions convexes et différentiables. Soit f une fonction convexe, différentiable [1]. L'algorithme de la descente de gradient a pour objectif de résoudre le problème d'optimisation suivant

$$(E) : \min_{x \in \mathbb{R}} f(x)$$

1. Il n'est pas obligatoire que f soit convexe, et une modification de l'algorithme permet d'étendre au cas des fonctions sous-différentiables avec les opérateurs proximal. En revanche, si f n'est pas convexe, la descente de gradient peut converger vers un minimum local et non global.

La descente de gradient

Puisque f est convexe et différentiable, il existe une unique solution x^* au problème d'optimisation (E) vérifiant $\nabla f(x^*) = 0$. L'objectif de la descente de gradient est de construire une suite itérative $(x_k)_{k\geq 0}$ tel que (x_k) converge vers x^*.

La descente de gradient

Algorithme 2 Descente de gradient

1: **Entrée :** Fonction convexe différentiable f, point initial x_0, nombre d'itérations maximal $i_{\max}$, seuil de tolérance ε.
2: $i \leftarrow 1$
3: **while** $i < i_{\max}$ **and** $\|\nabla f(x)\| > \varepsilon$ **do**
4: Calcul du pas optimal η_i
5: Mise à jour de la solution approchée

$$x_i = x_{i-1} - \eta_k \nabla f(x_{i-1})$$

6: $i \leftarrow i + 1$
7: **end while**
8: **return** x_{i-1}

Algorithme 3 Entraînement par Gradient Boosting à base d'arbres

1: **Entrée :** Données d'apprentissage $(\mathbf{x}_i, y_i)_{1 \leq i \leq n}$, nombre d'itérations K.
2: Entraînement du prédicteur faible initial $F_0 \overset{\Delta}{=} f_0$ sur la base $(\mathbf{x}_i, y_i)_{1 \leq i \leq n}$
3: **for** $k \leftarrow 1$ **to** K **do**
4: Calcul des pseudo-résidus

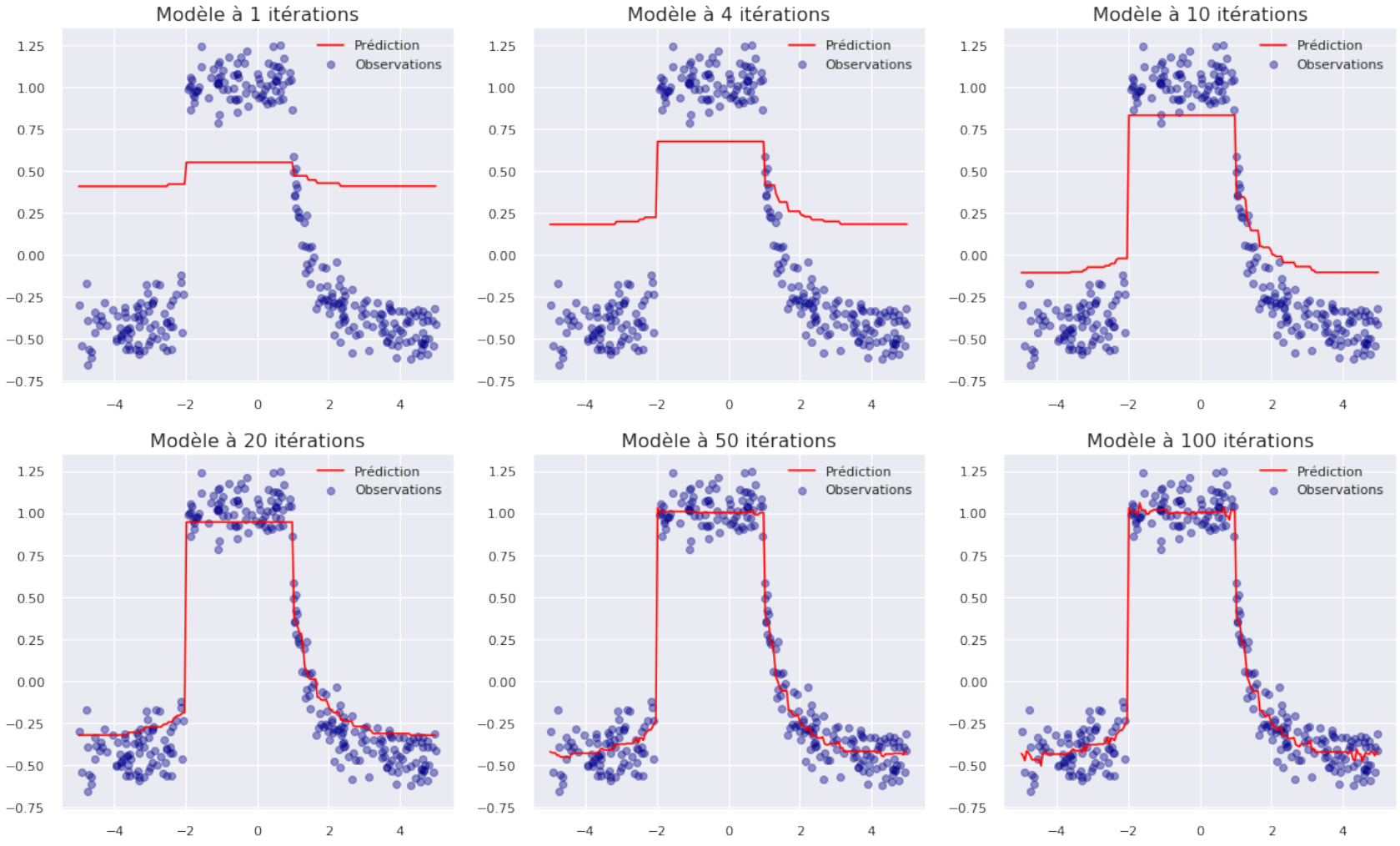
$$h(\mathbf{x}_i) = - \left. \frac{\partial l(y_i, g(\mathbf{x_i}))}{\partial g(\mathbf{x_i})} \right|_{g=F_{k-1}} \qquad 1 \leq i \leq n$$

5: Entraînement d'un CART f_k sur la base $(\mathbf{x}_i, h(\mathbf{x}_i))_{1 \leq i \leq n}$
6: Calcul du pas d'apprentissage

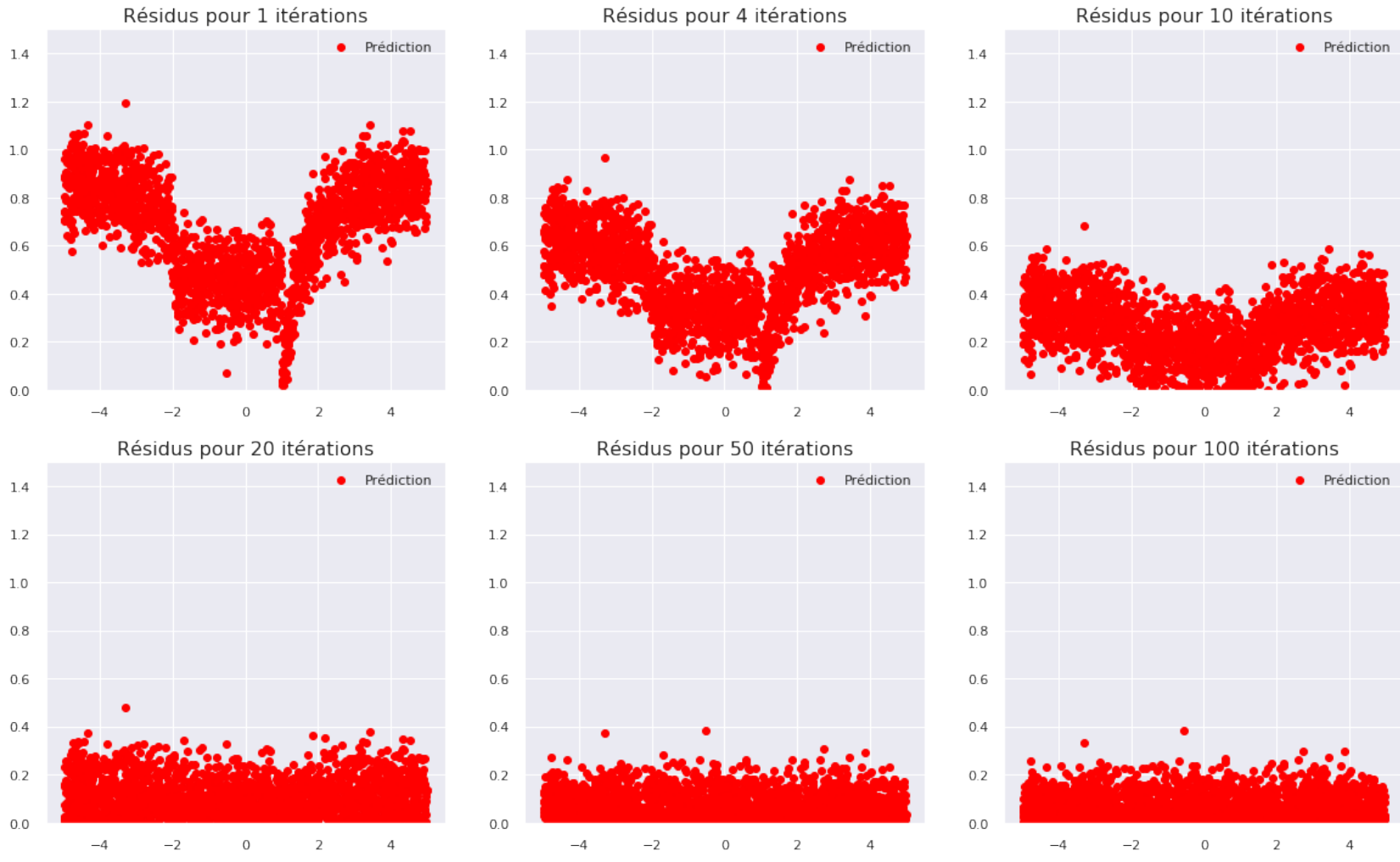
$$\eta_k = \underset{\eta}{\operatorname{argmin}} \sum_{i=1}^{n} l(y_i, F_{k-1}(\mathbf{x}_i) + \eta f_k(\mathbf{x}_i))$$

7: Mise à jour du méta-modèle : $F_k = F_{k-1} + \eta_k f_k$
8: **end for**
9: **return** F_K

Visualisation du modèle additif



Visualisation du modèle additif



Sommaire

XGBoost

Le modèle **XGBoost** (pour *eXtreme Gradient Boosting*) à base d'arbres ressemble au Gradient Boosting dans sa forme, mais la construction des arbres est différente. La classe de fonction candidate devient

$$\mathcal{F} = \left\{ f : f(x) = f_0(x) + \sum_{k=1}^{T} \eta f_k(x), \eta \in]0,1], (f_0, \ldots, f_T) \in \mathcal{F}_{\text{Weak}} \right\}.$$

Chaque arbre construit par XGBoost est une fonction de la classe $\mathcal{F}_{\text{CART}}$. La fonction de perte du modèle de *boosting* est **régularisée**, c'est-à-dire qu'une pénalisation est ajoutée afin de limiter le sur-apprentissage des arbres.

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, F_k(\mathbf{x}_i)) + \sum_{l=1}^{k} \Omega(f_l) \text{ avec } \Omega(f) = \gamma T + \frac{1}{2}\lambda \|w\|^2$$

Apprentissage du XGBoost

Notons que si $\Omega(f) = 0$, le modèle XGBoost est équivalent au Gradient Boosting dans le cadre du processus d'entraînement. L'utilisation du terme de régularisation Ω vise à sélectionner des classifieurs simples et performants. En pratique, il n'est pas possible d'optimiser la fonction objectif puisque le terme Ω nécessite les prédicteurs faibles précédents en paramètre. À l'instar du Gradient Boosting, l'entraînement d'un XGBoost est réalisée de manière **additive**.

Notons $\hat{y}_i^{(t)} \triangleq F_t(\mathbf{x}_i)$ la prédiction de l'observation $\mathbf{x}_i$ à la t-ème itération. La fonction objectif à minimiser $\mathcal{L}^{(t)}$ adaptée de $\mathcal{L}$ s'écrit

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) + \Omega(f_t)$$

Apprentissage du XGBoost

Les termes en $\Omega(f_l)$ pour $l < t$ ne dépendent pas de t et peuvent être retirés de la fonction objectif. Sans perte de généralités, en supposant $y \mapsto l(\cdot, y)$ de classe $\mathcal{C}^2$, par la formule de Taylor-Young, une approximation du second ordre de la fonction de perte permet d'optimiser plus facilement la fonction objectif.

$$\begin{aligned} l(y_i, \hat{y}_i^{(t-1)} + f_t(\mathbf{x}_i)) &= l(y_i, \hat{y}_i^{(t-1)}) + \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}) f_t(\mathbf{x}_i) + \\ & \frac{1}{2} \frac{\partial^2}{\partial (\hat{y}_i^{(t-1)})^2} l(y_i, \hat{y}_i^{(t-1)}) f_t(\mathbf{x}_i)^2 + o((f_t(\mathbf{x}_t)^3) \end{aligned}$$

Apprentissage du XGBoost

En posant $g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$, $h_i = \frac{\partial^2}{\partial (\hat{y}_i^{(t-1)})^2} l(y_i, \hat{y}_i^{(t-1)})$, et en retirant les termes constants, $\mathcal{L}^{(t)}$ se réécrit par

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

Pour optimiser le terme $\Omega(f_l)$, définissons $I_j = \{i : q(\mathbf{x}_i) = j\}$ l'ensemble des observations appartenant à la feuille j. L'équation précédente devient

$$\mathcal{L}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left(\sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T.$$

Apprentissage du XGBoost

Tous les termes de la fonction objectif étant différentiables

$$\nabla \mathcal{L}^{(t)} = \sum_{j=1}^{T} \left[\left(\sum_{i \in I_j} g_i \right) + \left(\sum_{i \in I_j} h_i + \lambda \right) w_j \right] + \gamma T,$$

et pour tout $1 \leq j \leq T$, il existe une unique solution

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

En réinjectant ces solutions dans la fonction objectif,

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

Apprentissage du XGBoost

En théorie, l'équation précédente permettrait d'évaluer la qualité de la structure d'arbre construite. Mais cette méthode nécessite des temps de calcul bien trop élevés : à l'instar de l'algorithme CART, les branches des arbres (les classifieurs faibles) sont construites de manière itérative. En notant R_G et R_D les observations situées respectivement à gauche et à droite du *split* d'un nœud, la fonction objectif permettant de déterminer le *split* optimal est

$$\mathcal{L}_{\text{split}} = \frac{1}{2}\left[\frac{\left(\sum_{i\in R_G} g_i\right)^2}{\sum_{i\in R_G} h_i + \lambda} + \frac{\left(\sum_{i\in R_D} g_i\right)^2}{\sum_{i\in R_D} h_i + \lambda} - \frac{\left(\sum_{i\in R} g_i\right)^2}{\sum_{i\in R} h_i + \lambda}\right] - \gamma,$$

avec $R = R_G \cup R_D$ l'ensemble des observations du nœud où effectuer le *split*.