2022 v1

# XGBoost – Notes sur cet algorithme indispensable

www.objectifdatascience.com

# Table des matières

# Contexte – Pourquoi ce document ?

XGBoost (eXtreme Gradient Boosting) est un des algorithmes star de ces dernières années. Il suffit de regarder les résultats Kaggle et de constater que XGBoost est l'algorithme qui occupe les premières places de manière systématique.

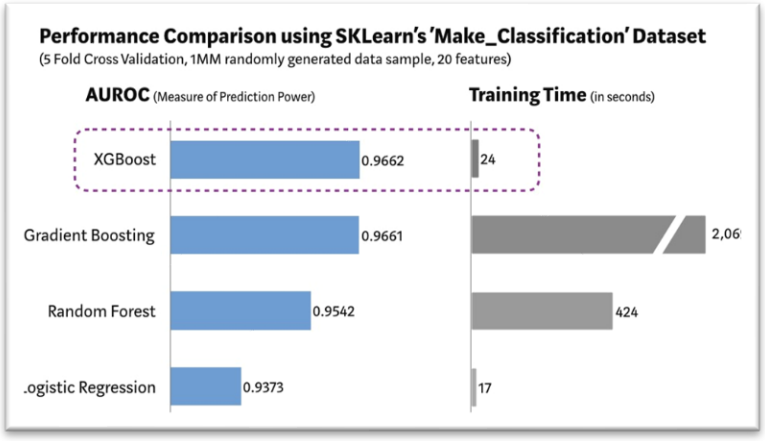



*Figure 1- Benchmark [Référence à trouver]*

Quand j'ai commencé à m'intéresser à cet algorithme et à l'utiliser, je voulais comprendre ses principes algorithmiques vus ces résultats assez insolents.

Les résultats que j'obtenais surpassaient largement ceux obtenus par les algorithmes plus classiques (régression linéaire, régression logistique, random Forest …).

Cela m'a poussé à vouloir comprendre ce qu'il y avait sous le capot de cet algorithme.

Il y a beaucoup de ressources qui existent sur le sujet, mais rien qui me convenait complètement. Les explications étaient insuffisantes et les mathématiques sous jacentes pas forcément explicites.

J'ai donc écrit cette note en regroupant toutes les idées importantes pour bien appréhender le fonctionnement de cet algorithme.

Vous trouvez à la fin de ce document une liste des ressources qui m'ont aidé à écrire cette note.

En espérant que vous trouverez ce document utile dans la compréhension de cet algorithme !

Enfin, si vous vous intéressez à la Datascience et au Machine Learning, je vous invite à ce que vous passiez du temps à comprendre ce qu'il y a derrière les algos. A pousser votre compréhension au maximum. Ne faites pas l'erreur de ne savoir que d'aligner les lignes de codes sans compréhension.

Si vous avez des remarques ou des questions, vous pouvez me joindre via Twitter ou je suis le plus actif en ce moment (@ObjectifDataSci)

## Ce que vous trouverez dans ce document

Ce document est issu de mes notes que j'ai prises lorsque j'ai étudié cet algorithme, en m'appuyant sur :

- Lecture de la document
- Lecture des papiers qui en parlent
- Blogs et vidéos YT qui en parlent
- Forums qui en parlent (reddit, stackoverflow) …

Avant de dire ce que vous y lirez, je vais d'abord vous dire ce que vous ne trouverez pas. Vous n'aurez pas par exemple de code. Je considère que le web est plein de sites qui donnent des exemples de codes (quoiqu'on trouve pas mal de chose à redire sur les exemples qui sont employés).

Dans ce document, vous trouverez :

- Le rappel des grands principes qui régissent cet algorithme
- Les fondements mathématiques qui le gouvernent. Vous verrez entre autre comment passer de l'expression d'une fonction objective assez générique à l'expression précise des paramètres nécessaires au modèle
- J'y ai mis toutes les démonstrations mathématiques nécessaires pour comprendre l'algorithme. Une personne avec un niveau de mathématiques du Bac doit pouvoir suivre les raisonnements mathématiques (les dérivées sont les concepts les plus complexes dans ce document)
- La concrétisation de cet algorithme dans les cas de régression et de classification, à partir de fonctions de perte bien précises
- L'illustration de ces deux cas à travers deux exemples, où on va ensemble calculer à la main les éléments du modèle, suivant les règles de l'algorithme
- Quelques éléments intéressants à connaître qui permettent de comprendre la force et la puissance de cet algo.

## Premiers pas

XGBoost est apparu en 2015 (c'est pas si vieux) et il s'est imposé rapidement comme un des algorithmes les plus efficaces en Machine Learning. Ses principales caractéristiques :

- Utilisé pour de régression (prédiction de valeurs numériques continues) ou la classification / segmentation
- Algorithme d'apprentissage supervisé, qui a besoin d'un jeu de données d'entraînement qui construira un modèle qui pourra être ensuite généralisé
- Fait partie des algorithmes « Ensemble Learning » qui impliquent l'utilisation de multiples arbres de décision pour construire la prédiction

Il est particulièrement performant :

- Dans sa capacité à généraliser car intégrant dans sa construction des mécanismes de régularisation assez puissants et astucieux
- Sa capacité à traiter les données manquantes, sans pour autant dégrader sa performance
- Sa rapidité de calcul sur des gros volumes en faisant des approximations élégantes lors de la construction des arbres de décision.

Nous verrons dans ce document les principes qui gouvernent cet algorithme et qui le rendent si performant.

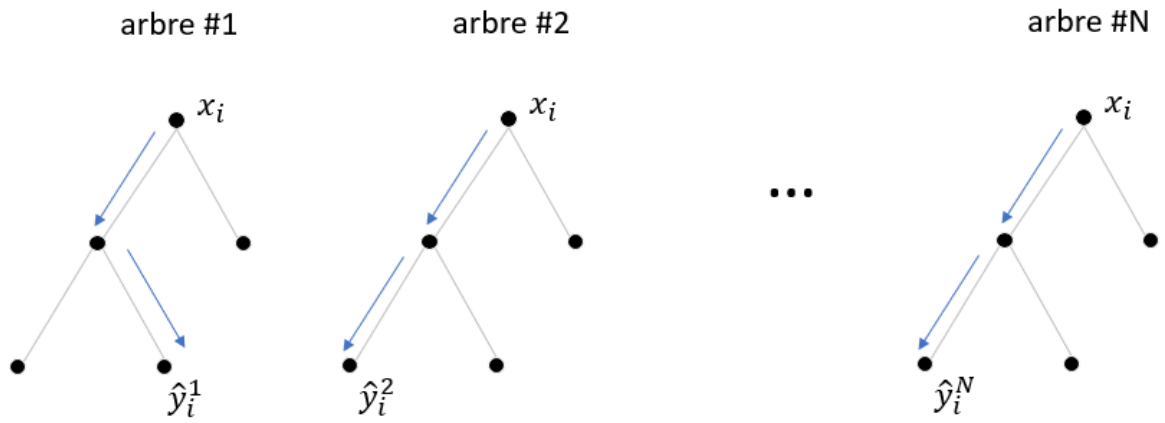
A noter également que XGBoost ne se sert pas des principes classiques de construction d'arbres de décisions (Critères d'entropie, de GINI par exemple). Ce sont des critères bien spécifiques qui sont détaillés et expliqués dans ce document.

## Pourquoi cela s'appelle Gradient Boosting ?

Le modèle algorithmique Gradient boosting fait appel à de multiples arbres de décision, à utiliser dans un ordre déterminé. Pour une observation, chaque arbre donne un résultat, et la prédiction finale est obtenue en additionnant chacune des valeurs obtenues données par les arbres.

Voici un schéma avec N arbres. L'observation est passée en entrée de chaque arbre et on récupère la prédiction que donne chaque arbre.

Dans la construction des arbres, XGBoost intègre des mécanismes de régularisation permettant d'avoir des arbres relativement simples (« weak learners ») pour éviter l'overfitting et permettre une généralisation sur des données inconnues du modèle avec un bon niveau de performance.




A chaque arbre #t : $x_i \rightarrow \widehat{y_i^t}$

Au final, la prédiction sera égale à l'addition de toutes les prédictions :

$$\hat{y}_i = \Phi(x_i) = \sum_{t=1}^{N} \varphi_t(x_i) = \sum_{t=1}^{N} \hat{y}_i^{(t)}$$

$\Phi : fonction\ globale\ pour\ la\ prédiction\ (régression\ ou\ classification)$

$\varphi_t : fonction\ de\ prédiction\ de\ l'arbre\ \#t$

Pour en revenir au nom de « Gradient boosting », on se rend compte qu'au fur et à mesure des différentes prédictions, quand on somme les valeurs obtenues, on s'approche de plus en plus de la prédiction finale. C'est exactement comme lorsque l'on fait la descente de gradient de la fonction coût lors de la phase d'apprentissage d'un algorithme d'apprentissage supervisé.

Les principes sont identiques, que l'on soit dans le cas d'une régression ou d'une classification. Pour une classification binaire, la valeur va se rapprocher de la valeur 0 (classe 1) ou 1 (classe 0).

## Phase d'apprentissage et Objective function

Lors de la phase d'apprentissage, XGBoost force la fonction globale objectif à contenir ces deux caractèristiques :

i) Une Loss fonction, mesurant l'écart entre valeurs prédites et cibles. Cette fonction doit être différentiable et convexe (afin de pouvoir être optimisée facilement et atteindre ainsi son minimum)
Exemples : squared-loss pour la régression et binary-loss pour la classification. On reviendra un peu plus loin sur ces deux fonctions

ii) Une fonction pour pénaliser la complexité du modèle, afin d'éviter l'overfitting (sur-apprentissage)

### Loss Function

Grand classique du Machine Learning pour l'apprentissage supervisé.

Sa forme générique est :

$$L = \sum_{i}^{\#instances} l(\hat{y}_i, y_i)$$

$\hat{y}_i : prédiction\ globale$

$y_i : valeur\ cible$

$\#instances : nombre\ de\ données\ utilisées\ lors\ de\ la\ phase\ d'apprentissage$

Exemple classique pour la régression – moindre carré (mean-square)

$$L = \frac{1}{2} \times \sum_{i}^{\#instances} (\hat{y}_i - y_i)^2$$

Exemple pour la classification – Log loss (cross-entropy)

$$L = -\sum_{i}^{\#instances} (y_i . \log(\hat{y}_i) + (1 - y_i) . \log(1 - \hat{y}_i))$$

Avec $y_i = 0\ ou\ 1$

Il existe quantités d'autres moyens d'exprimer la fonction L. Les implémentations de XGBoost permettent justement cette liberté aux utilisateurs.

## Pénalisation (Régularisation)

Sans régularisation, le risque est grand de faire du sur-apprentissage. Le modèle sera alors parfait sur les données d'apprentissage mais médiocre lors de la phase d'inférence ou il devra généraliser sur des données inconnues.

Le risque est d'autant plus grand que les arbres de décision ont cette capacité d'être « parfaits » sur les données de training, si on les développe jusqu'à ce qu'il n'y ait qu'une instance par feuille.

On définit ici une fonction qui traduit la complexité globale du système. Pour cela, on somme la complexité de chaque arbre composant le modèle

$$\sum_{k}^{\#arbres} \Omega(arbre_{\#k})$$

$$ou\ \Omega(arbre_{\#k}) = \gamma.T + \frac{1}{2}.\lambda.\|w\|^2$$

La complexité de chaque arbre dépend de deux variables :

- $T$ : $nombre\ de\ feuilles\ de\ l'arbre$
- $\|w\|^2$ : $norme\ L2\ des\ valeurs\ de\ chaque\ feuille\ des\ arbres$

$\gamma\ et\ \lambda$ sont des hyper-paramètres du modéle global. Plus ils sont grands,moins le modèle comportera de la complexité. On verra par la suite leur influence dans la construction du modéle

Avec cette fonction de pénalisation, l'algorithme fera en sorte de limiter les arbres qui ont un grand nombre de feuilles ou des valeurs de prédiction importantes.

L'intégration by design de cette régularisation fait une des forces de XGBoost qui sera ainsi en capacité de généraliser sur les instances inconnues lors de l'apprentissage.

C'est une des forces majeures de cet algorithme qui en fait sa performance.

## Fonction objectif : Loss + Pénalisation

D'après ce qui a été vu plus haut :

$$\mathcal{L} = \underbrace{\sum_{i}^{\#instances} l(\hat{y}_i, y_i)}_{Loss} + \underbrace{\sum_{k}^{\#arbres} (\gamma.T_k + \frac{1}{2}.\lambda.\|w_k\|^2)}_{Pénalisation\ modèle}$$

La forme de cette fonction est ce qui rend cet algorithme mystérieux à saisir.

On note plusieurs choses qui nous empêche de faire une optimisation facile de cette fonction :

- La fonction loss fait intervenir la valeur $\hat{y}_i$ qui est obtenue en utilisant tous les arbres. Les deux sommes sont donc intriquées, ce qui rend l'optimisation compliquée
- En effet, classiquement, on cherche à calculer le gradient de la fonction pour faire évoluer les paramètres vers sa minimisation. Ici, ce n'est pas une chose évidente vue la forme que prend la fonction
- Cela voudrait dire que l'on pourrait optimiser cette fonction une fois que les arbres sont tous connus. Or comment construire ces arbres si on a pas un moyen simple d'utiliser cette fonction pour chacun de ces arbres ?
- L'optimisation devra donc comporter une simplification permettant de résoudre ce probléme. Cette simplification devra « démixer » le fait que l'on doive sommer sur toutes les instances et en parallèle que l'on doive sommer sur tous les arbres.

A mon sens, c'est le principal défi dans la compréhesion de l'algorithme de XGBoost.

C'est ce que j'ai eu du mal à comprendre lorsque j'ai voulu saisir les subtilités de l'algorithme. Beaucoup de commentaires sur cet algorithme passent sous silence cette difficulté (soit parcequ'elle n'a pas été vue, soit pas comprise).

Pour arriver à aller plus loin, l'astuce utilisée est l'approximation donnée par l'expression de Taylor (que nous verrons en détail par la suite)

## Ré-écriture de la fonction objectif $\mathcal{L}$

Pour opérer cette ré-écriture, on va utiliser le principe premier de la conception de ce modèle : celui qui dit qu'en ajoutant un nouvel arbre à chaque itération, on se rapproche pas à pas de la prédiction finale en additionnant les valeurs obtenues.

A la première itération (t=0), on choisit une simple constante comme valeur de prédiction. Prenons par exemple 0.

$$\hat{y}_i^{(0)} = 0$$

A l'itération suivante, on ajoute le résultat obtenu grâce au premier arbre

$$\hat{y}_i^{(1)} = \hat{y}_i^{(0)} + \varphi_1(x_i)$$

A l'itération t, on a :

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} \varphi_k\,(x_i) = \hat{y}_i^{(t-1)} + \varphi_t(x_i)$$

Et la forme de la fonction objectif est alors, à cette itération t :

$$\mathcal{L}_t = \sum_{i}^{\#instances} l(\hat{y}_i^{(t-1)} + \varphi_t(x_i), y_i) + \Omega(arbre_{\#t})$$

Ainsi, si on veut minimiser la fonction objectif à l'itération t, il suffit de trouver $\varphi_t(x_i)$ qui minimise $\mathcal{L}_t$. Cela simplifie grandement les opérations.

Ici, on se repose sur un raisonnement par récurrence, qui suppose qu'à l'itération t, le modèle est déjà optimisé pour l'itération t-1 et donc qu'on a plus à y revenir.

Autrement dit :

$$\varphi_t = \underset{\varphi_t}{argmin} \sum_{i}^{\#instances} l(\hat{y}_i^{(t-1)} + \varphi_t(x_i)\,, y_i) + \Omega(arbre_{\#t})$$

Cette ré-écriture de la fonction permettra de faire une optimisation par étape, ce qui sera beaucoup plus simple à implémenter.

# Application de la formule de Taylor à la fonction objectif $\mathcal{L}$

Allons plus loin dans l'écriture de la fonction $\mathcal{L}$.

On va prendre les mêmes principes que dans la descente de gradient, qui fait appel au gradient de la fonction. En fait, cela revient à faire appel à l'expression de Taylor au 1er degré.

Pour XGBoost, on va pousser le developpement de Taylor à l'ordre 2, ce qui permettra une simplification des écritures ainsi qu'une convergence plus rapide des résultats [Référence à trouver pour prouver cela]

## Rappels sur l'expression de Taylor

Pour toute fonction f, il est possible de l'approximer au voisinage d'une valeur a grâce aux valeurs de ses dérivées de différents ordres au point a

$$f(x) \approx f(a) + (x-a).f'(a) + \cdots + \frac{(x-a)^n}{n!}.f^n(a)$$

Pour x proche de a.

La forme que prend cette approximation est donc une forme polymoniale de degré n, plus simple à manipuler (cf formes de Newton et de Lagrange par exemple)

Autre écriture possible, avec a = $x + dx$ et dx petit :

$$f(x+dx) \approx f(x) + dx.f'(x) + \frac{dx^2}{2}.f''(x) + \cdots + \frac{dx^n}{n!}.f^n(x)$$

Rappellons la forme de la fonction objectif est l'itération t :

$$\mathcal{L}_t = \sum_{i}^{\#instances} l(\hat{y}_i^{(t-1)} + \varphi_t(x_i), y_i) + \Omega(arbre_{\#t})$$

Et là, c'est ici que le coup de génie (plus une certaine élégance mathématique) s'opère pour avoir une simplification de l'écriture.

On va supposer que la quantité $\varphi_t(x_i)$ est suffisamment petite pour pouvoir appliquer la formule de Taylor au degré 2

$$\boxed{\mathcal{L}_t \approx \sum_{i}^{\#instances} \left[ l(\hat{y}_i^{(t-1)}, y_i) + g_i.\varphi_t(x_i) + \frac{1}{2}.h_i.\varphi_t^2(x_i) \right] + \Omega(arbre_{\#t})}$$

Ou

- $g_i = \frac{\partial}{\partial \hat{y}_i^{(t-1)}} . l(\hat{y}_i^{(t-1)}, y_i)$ est le gradient (dérivée première)
- $h_i = \frac{\partial^2}{\partial^2 \hat{y}_i^{(t-1)}} . l(\hat{y}_i^{(t-1)}, y_i)$ est le hessien (dérivée seconde)

Il faut bien comprendre la portée de ce qui a été fait ici. C'est la partie un peu compliquée du raisonnement.

En passant de :

$$\sum_{i}^{\#instances} l(\hat{y}_i^{(t-1)} + \varphi_t(x_i), y_i)$$

à l'équation suivante

$$\sum_{i}^{\#instances} l(\hat{y}_i^{(t-1)}, y_i) + g_i . \varphi_t(x_i) + \frac{1}{2} . h_i . \varphi_t^2(x_i)$$

on obtient une propriété très intéressante.

La fonction L ne dépend plus de l'itération t, mais uniquement de l'itération t-1. Cette approximation permet de désintriquer les équations, de séparer les étapes, et simplifie ainsi grandement les calculs en prenant désormais une démarche itérative, arbre après arbre.

Reprenons l'expression que l'on souhaite minimiser à l'étape t :

$$\varphi_t = argmin_{\varphi_t} \sum_{i}^{\#instances} l(\hat{y}_i^{(t-1)} + \varphi_t(x_i), y_i) + \Omega(arbre_{\#t})$$

$$\varphi_t = argmin_{\varphi_t} \sum_{i}^{\#instances} \left\{ l(\hat{y}_i^{(t-1)}, y_i) + g_i . \varphi_t(x_i) + \frac{1}{2} . h_i . \varphi_t^2(x_i) \right\} + \Omega(arbre_{\#t})$$

Or à l'étape t : $l(\hat{y}_i^{(t-1)}, y_i)$ est une constante ! (Raisonnement par récurrence – l'optimisation faite à l'itération t-1 rend cette valeur optimale, donc constante …)

Et donc on peut l'enlever de l'expression à minimaliser :

$$\varphi_t = argmin_{\varphi_t} \sum_{i}^{\#instances} \left\{ g_i . \varphi_t(x_i) + \frac{1}{2} . h_i . \varphi_t^2(x_i) \right\} + \Omega(arbre_{\#t})$$

Avec cette simplification, revenons maintenant au calcul de la fonction objectif, en précisant le terme de régularisation Ω pour l'arbre t

$$\mathcal{L}_t = \sum_{i}^{\#instances} \left\{ g_i . \varphi_t(x_i) + \frac{1}{2} . h_i . \varphi_t^2(x_i) \right\} + \gamma . T + \frac{1}{2} . \lambda . \sum_{j=1}^{T} w_j^2$$

Et développons les sommes

$$\mathcal{L}_t = \sum_{i}^{\#instances} g_i . \varphi_t(x_i) + \frac{1}{2} . \sum_{i}^{\#instances} h_i . \varphi_t^2(x_i) + \gamma . T + \frac{1}{2} . \lambda . \sum_{j=1}^{T} w_j^2$$

Ici, on est bien coincés car les sommes ne portent pas sur les mêmes objets :

- On a des sommes sur les instances d'apprentissage (pour la fonction L)
- Et des sommes sur les arbres (pour les fonctions de régularisation

Mais il y a un moyen astucieux de s'en sortir …

On va maintenant exprimer les sommes sur toutes les instances, par des sommes qui portent sur les feuilles de l'arbre. Cela revient à inverser les signes sommes …

$$\sum_{i}^{\#instances} \left[ \sum^{Arbres} g_i . \varphi_t(x_i) \right] \rightarrow \text{à } transformer\ en \sum_{T}^{Arbres} \left[ \sum^{\#instances} g_i . \varphi_t(x_i) \right]$$

Donc plutôt que de boucler chaque instance et de chercher la feuille concernée pour avoir le terme $g_i . \varphi_t(x_i)$, on parcourt toutes les feuilles et on identifie quelle instance tombe sur cette feuille. Cela permet de factoriser la valeur de la feuille $w_j$ pour chacune des feuilles et « d'inverser » les sommes

Autrement dit :

$$\sum_{i}^{\#instances} g_i . \varphi_t(x_i) = g_1 . \varphi_t(x_1) + g_2 . \varphi_t(x_2) + \cdots + g_n . \varphi_t(x_n)$$

avec

$$\varphi_t(x_i) = w_j\ si\ on\ tombe\ sur\ la\ feuille\ j\ de\ l'arbre, dont\ la\ valeur\ est\ w_j$$

On peut donc factoriser par rapport à chacune des feuilles de l'arbre et sommer tous les gradients $g_i$ qui tombent sur la même feuille j de l'arbre

Soit

$$\sum_{i}^{\#instances} g_i.\varphi_t(x_i) = \sum_{j=1}^{T}\left[\left(\sum_{i\in j} g_i\right).w_j\right]$$

Car $\varphi_t(x_i) = w_j$ par définition.

Et on a gagné !

$$\sum_{i}^{\#instances} g_i.\varphi_t(x_i) = \sum_{j=1}^{T}\left[\left(\sum_{i\in j} g_i\right).w_j\right]$$

De la même facon :

$$\frac{1}{2}.\sum_{i}^{\#instances} h_i.\varphi_t^2(x_i) = \frac{1}{2}.\sum_{j=1}^{T}\left[\left(\sum_{i\in j} h_i\right).w_j^2\right]$$

Soit, en utilisant ces deux nouvelles égalités :

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

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

Enfin

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

Grace à cette nouvelle forme d'équation, on peut facilement trouver la valeur $w_j$ optimale pour l'arbre T.

Pour trouver ces valeurs, on se sert de la relation classique disant qu'on obtient un optimum si la dérivée première (ou le gradient) s'annule. Soit :

$$w_j = argmin_{w_j}\,\mathcal{L}_t \implies \frac{\partial \mathcal{L}_t}{\partial w_j} = 0$$

$$\frac{\partial \mathcal{L}_t}{\partial w_j} = 0 \Rightarrow \frac{\partial}{\partial w_j} \sum_{j=1}^{T} \left[ \left( \sum_{i \in j} g_i \right) . w_j + \frac{1}{2} . \left( \sum_{i \in j} h_i + \lambda \right) . w_j^2 \right] + \frac{\partial}{\partial w_j} \gamma . T = 0$$

Mais $\frac{\partial}{\partial w_j} \left\{ \left( \sum_{i \in j} g_i \right) . w_j + \frac{1}{2} . \left( \sum_{i \in j} (h_i + \lambda) \right) . w_j^2 + \gamma . T \right\} = 0$ *pour toutes les feuilles* $\neq j$ *car ne dépendant pas de* $w_j$

Donc :

$$\frac{\partial}{\partial w_j} \left[ \left( \sum_{i \in j} g_i \right) . w_j + \frac{1}{2} . \left( \sum_{i \in j} h_i + \lambda \right) . w_j^2 + \gamma . T \right] = 0$$

Soit :

$$\left( \sum_{i \in j} g_i \right) + \left( \sum_{i \in j} h_i + \lambda \right) . w_j = 0$$

Car $\frac{\partial}{\partial w_j} \gamma . T = 0$ ne dépendant pas de $w_j$

Donc :

$$w_j = - \frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i + \lambda}$$

Cette formule permet de calculer la valeur optimale d'une feuille quand l'arbre est construit

Maintenant, calculons la valeur de la fonction Loss pour un arbre. On a :

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

Remplaçons $w_j$ par sa valeur que nous venons de trouver

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

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

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

Soit

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

Cette formule donne la perte pour l'arbre construit à l'étape t. Plus cette valeur est petite, meilleure sera la contribution de cet arbre.

Cette formule nous servira plus tard à déterminer le split optimal pour l'arbre – c'est-à-dire quel est le critère à utiliser pour construire une nouvelle branche décisionnelle.

## Récapitulatif

On a vu beaucoup d'équations. Récapitulons ici les plus importantes.

Comment prédire la valeur pour $x_i$, étant donné le modèle déjà construit

$$\hat{y}_i = \Phi(x_i) = \sum_{t=1}^{N} \varphi_t(x_i) = \sum_{t=1}^{N} \tilde{y}_i^{(t)}$$

$$\Phi : fonction\ globale\ pour\ la\ prédiction\ (régression\ ou\ classification)$$

$$\varphi_t : fonction\ de\ prédiction\ de\ l'arbre\ \#t$$

Comment prédire la valeur pour $x_i$, à l'étape t

$$\tilde{y}_i^{(t)} = \sum_{k=1}^{t} \varphi_k(x_i) = \tilde{y}_i^{(t-1)} + \varphi_t(x_i)$$

La fonction objectif

$$\mathcal{L} = \sum_{i}^{\#instances} l(\hat{y}_i, y_i) + \sum_{k}^{\#arbres} (\gamma . T_k + \frac{1}{2} . \lambda . \|w_k\|^2)$$

Le poids optimal pour une feuille

$$w_j = -\frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i + \lambda}$$

$g_i$ : gradient de la fonction Loss

$h_i$ : dérivée seconde de la fonction Loss

J : index de la feuille de l'arbre

La perte pour un arbre

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

## Construction des arbres – détermination des splits

Avec la formule de perte vue précédemment pour un arbre donné, il sera facile de déterminer les critères permettant de construire une nouvelle branche de l'arbre.

Pour expliquer l'algorithme, nous allons prendre l'hypothèse que l'ensemble des features et des valeurs seront testées, pour évaluer la pertinence d'un nouveau split.

On verra par la suite que XGBoost a un fonctionnement beaucoup plus subtil que celui-ci (même si une recherche exhaustive est faite si le jeu de données d'apprentissage est petit). En effet, le test systématique de toutes les valeurs peut-être extrêmement lourd en terme de calcul.

Construisons l'arbre de manière iterative.

A l'étape 0, nous avons un arbre avec une seule feuille et une seule valeur.

2 questions se posent à nous :

- Devons nous effectuer un nouveau split, c'est-à-dire créer deux nouvelles feuilles ?
- Si oui, comment faire ce split ? Quelle feature et quelle valeur prendre pour le split ?

Ces 2 questions se poseront systématiquement à chaque étape de la construction de l'arbre.

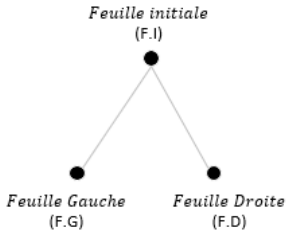
Devons nous faire un nouveau split ?

Pour répondre à cette question, on dois se reposer sur le Loss. La réponse sera positive si en faisant un nouveau split, on diminue la perte ! Les deux nouvelles branches doivent diminuer la perte globale.

Pour cela on calcule :

- Loss de la feuille de Gauche + Loss de la feuille de droite
- Loss de la feuille initiale

On rappelle que la régularisation est intégrée dans les loss de chacune des feuilles et qu'elle sert à empêcher un arbre de trop se complexifier.




Pour que le split soit intéressant, on vérifie que la somme pénalisée des Loss des deux nouvelles feuilles est inférieure au Loss de la feuille initiale. En effet, si on a plus de perte avec le nouveau split, celui-ci ne sert à rien.

C'est-à-dire :

$$Loss_{Feuille\ Gauche} + Loss_{Feuille\ Droite} < Loss_{Feuille\ Initiale}$$

On connait l'expression de la perte pour un arbre (somme de la perte sur toutes ses feuilles)

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

On en déduit que la perte calculée sur une seule feuille est la suivante :

$$-\frac{1}{2}.\frac{(\sum_{i\in Feuille} g_i)^2}{\sum_{i\in Feuille} h_i + \lambda} + \gamma$$

On remarque que dans cette expression que les éléments de pénalisation $\lambda$ et $\gamma$ sont prises en compte

L'équation devient alors :

$$-\frac{1}{2}.\frac{(\sum_{i\in FG} g_i)^2}{\sum_{i\in FG} h_i + \lambda} + \gamma - \frac{1}{2}.\frac{(\sum_{i\in FD} g_i)^2}{\sum_{i\in FD} h_i + \lambda} + \gamma < -\frac{1}{2}.\frac{(\sum_{i\in FI} g_i)^2}{\sum_{i\in FI} h_i + \lambda} + \gamma$$

avec FG : Feuille Gauche, FD : Feuille Droite et FI : Feuille Initiale

Alors :

$$-\frac{1}{2}.\frac{(\sum_{i\in FG} g_i)^2}{\sum_{i\in FG} h_i + \lambda} - \frac{1}{2}.\frac{(\sum_{i\in FD} g_i)^2}{\sum_{i\in FD} h_i + \lambda} + \gamma < -\frac{1}{2}.\frac{(\sum_{i\in FI} g_i)^2}{\sum_{i\in FI} h_i + \lambda}$$

Et :

$$-\frac{(\sum_{i\in FG} g_i)^2}{\sum_{i\in FG} h_i + \lambda} - \frac{(\sum_{i\in FD} g_i)^2}{\sum_{i\in FD} h_i + \lambda} + 2.\gamma < -\frac{(\sum_{i\in FI} g_i)^2}{\sum_{i\in FI} h_i + \lambda}$$

Donc :

$$\frac{(\sum_{i\in FG} g_i)^2}{\sum_{i\in FG} h_i + \lambda} + \frac{(\sum_{i\in FD} g_i)^2}{\sum_{i\in FD} h_i + \lambda} - \frac{(\sum_{i\in FI} g_i)^2}{\sum_{i\in FI} h_i + \lambda} - 2.\gamma > 0$$

Un split, transformant une feuille de l'arbre en 2 nouvelles feuilles, peut donc être réalisé si cette relation est satisfaite. Dans ce cas, le nouveau split va faire converger la prédiction vers la solution, tout en respectant les contraintes de régularisation !

## Précisions sur les critères d'un split (création de nouvelles feuilles)

L'expression sur la validité d'un split en fonction du Loss avant/après split peut aider pour déterminer quels sont les critères du split optimal.

En effet, appelons Score cette valeur :

$$Score = \frac{(\sum_{i \in FG} g_i)^2}{\sum_{i \in FG} h_i + \lambda} + \frac{(\sum_{i \in FD} g_i)^2}{\sum_{i \in FD} h_i + \lambda} - \frac{(\sum_{i \in FI} g_i)^2}{\sum_{i \in FI} h_i + \lambda} - 2.\gamma$$

La manière la plus triviale et brutale de trouver les critères d'un split est de scanner l'ensemble des couples (feature, Valeur de feature), de tester le gain du split sur ces couples, et d'identifier la combinaison qui donne le score positif maximal.

Le pseudo-code est le suivant :

```
Max_score = 0
Best_feature = None
Best_feature_value = None
Pour toutes les features k
        Pour toutes les instances i de la feuille initiale avant split, on split sur la valeur x_ik de i
                Calcul du score (FD + FG – FI – Penalty)
                Si score > Max_score
                        Max_score = score
                        Best_feature = k
                        Best_feature_value = x_ik

Nota : x_ik est la valeur de la feature k de l'instnace i
```

Ici $x_{ik}$ est la valeur de l'instance i pour la feature k, et la feuille gauche sera définie par les instances qui auront leur valeur sur la feature k < $x_{ik}$, et la feuille droite > $x_{ik}$

Cet algorithme est très gourmand en calcul et XGBoost propose une optimisation qui réduit drastiquement les temps de calcul. Nous verrons en quoi consiste cette optimisation un peu plus loin.

## Application pour la régression

Rappelons les résultats obtenus jusqu'à présent.

Pour un arbre donné :

- la valeur optimale des feuilles est la suivante : $w_j = -\frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i + \lambda}$,
- la perte sur une seule feuille est : $-\frac{1}{2} . \frac{(\sum_{i \in Feuille} g_i)^2}{\sum_{i \in Feuille} h_i + \lambda} + \gamma$
- la condition de split est la suivante : $\frac{(\sum_{i \in FG} g_i)^2}{\sum_{i \in FG} h_i + \lambda} + \frac{(\sum_{i \in FD} g_i)^2}{\sum_{i \in FD} h_i + \lambda} - \frac{(\sum_{i \in FI} g_i)^2}{\sum_{i \in FI} h_i + \lambda} - 2.\gamma > 0$

avec gi gradient (dérivée première) de la fonction Loss et hi dérivée seconde de la fonction Loss

On se propose de rendre cela plus concrêt pour la régression.

Définissons la fonction Loss pour la régression, comme étant la fonction classique suivante :

$$L(\hat{y}_i, y_i) = \frac{1}{2} \times \sum_{i}^{\#instances} (\hat{y}_i - y_i)^2$$

Calculons le gradient pour cette fonction :

$$g_i = \frac{\partial}{\partial \hat{y}_i} . L(\hat{y}_i, y_i)$$

$$g_i = \frac{\partial}{\partial \hat{y}_i} . \frac{1}{2} \times \sum_{i}^{\#instances} (\hat{y}_i - y_i)^2$$

$$g_i = \sum_{i}^{\#instances} (\hat{y}_i - y_i) : \textit{opposé de la somme des résidus } (Résidu = y_i - \hat{y}_i)$$

Calculons le hessien pour cette fonction :

$$h_i = \frac{\partial^2}{\partial {\hat{y}_i}^2} . L(\hat{y}_i, y_i) = \frac{\partial}{\partial \hat{y}_i} . g(\hat{y}_i, y_i)$$

soit

$$h_i = \frac{\partial}{\partial \hat{y}_i} \sum_{i}^{\#instances} (\hat{y}_i - y_i) = \sum_{i}^{\#instances} 1 = \textit{Nombre d'instances}$$

Avec ces valeurs on obtient :

la valeur optimale des feuilles :

$$w_j = -\frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i + \lambda} = \frac{\sum_{i \in j} y_i - \hat{y}_i}{Nb\ Instances + \lambda}$$

la perte sur une seule feuille :

$$-\frac{1}{2} . \frac{(\sum_{i \in j} y_i - \hat{y}_i)^2}{Nb\ Instances + \lambda} + \gamma$$

Et la notion de score permettant de trouver le split optimal :

$$Score = \underbrace{\frac{(\sum y_i - \hat{y}_i)^2}{Nb\ Instances + \lambda}}_{Feuille\ Gauche} + \underbrace{\frac{(\sum y_i - \hat{y}_i)^2}{Nb\ Instances + \lambda}}_{Feuille\ droite} - \underbrace{\frac{(\sum y_i - \hat{y}_i)^2}{Nb\ Instances + \lambda}}_{Feuille\ initiale} - 2.\gamma$$

## Application pour la classification

Définissons la fonction Loss pour la classification, comme étant la fonction classique logloss :

$$L = -\sum_{i}^{\#instances} (y_i.\log(p_i) + (1 - y_i).\log(1 - p_i)) \; avec \; p_i = \frac{1}{1 + e^{-\hat{y}_i}}$$

Calculons le gradient et le hessian pour cette fonction, en faisant appel à la composition des dérivations (chaînage)

$$g_i = \frac{\partial}{\partial \hat{y}_i} . L(\hat{y}_i, y_i) = \frac{\partial L(\hat{y}_i, y_i)}{\partial p_i} \times \frac{\partial p_i}{\partial \hat{y}_i}$$

Calculons le premier terme de ce produit :

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial p_i} = -\frac{\partial}{\partial p_i} . \sum_{i}^{\#instances} (y_i.\log(p_i) + (1 - y_i).\log(1 - p_i))$$

On a :

$$\frac{\partial}{\partial p_i} . y_i.\log(p_i) = \frac{y_i}{p_i}$$

Et :

$$\frac{\partial}{\partial p_i} . (1 - y_i).\log(1 - p_i) = -\frac{1 - y_i}{1 - p_i}$$

Soit :

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial p_i} = -\sum_{i}^{\#instances} \left[\frac{y_i}{p_i} - \frac{1 - y_i}{1 - p_i}\right] = -\sum_{i}^{\#instances} \left[\frac{y_i.(1 - p_i)}{p_i.(1 - p_i)} - \frac{(1 - y_i).p_i}{(1 - p_i).p_i}\right]$$

$$\frac{\partial L(\hat{y}_i, y_i)}{\partial p_i} = -\sum_{i}^{\#instances} \left[\frac{y_i.(1 - p_i) - (1 - y_i).p_i}{p_i.(1 - p_i)}\right] = -\sum_{i}^{\#instances} \left[\frac{y_i - p_i}{p_i.(1 - p_i)}\right]$$

Calcul du deuxième terme du produit :

$$\frac{\partial p_i}{\partial \hat{y}_i} = -\left(\frac{1}{1 + e^{-\hat{y}_i}}\right)^2 \times (-1) \times e^{-\hat{y}_i} = \frac{e^{-\hat{y}_i}}{(1 + e^{-\hat{y}_i})^2} = \frac{1}{1 + e^{-\hat{y}_i}} \times \frac{e^{-\hat{y}_i}}{1 + e^{-\hat{y}_i}}$$

$$\frac{\partial p_i}{\partial \hat{y}_i} = \frac{1}{1+e^{-\hat{y}_i}} \times \frac{1+e^{-\hat{y}_i}-1}{1+e^{-\hat{y}_i}} = \frac{1}{1+e^{-\hat{y}_i}} \times \left[\frac{1+e^{-\hat{y}_i}}{1+e^{-\hat{y}_i}} - \frac{1}{1+e^{-\hat{y}_i}}\right]$$
$$= \frac{1}{1+e^{-\hat{y}_i}} \times \left[1 - \frac{1}{1+e^{-\hat{y}_i}}\right]$$

Soit :

$$\frac{\partial p_i}{\partial \hat{y}_i} = p_i.(1 - p_i)$$

Enfin

$$g_i = \frac{\partial}{\partial \hat{y}_i}.L(\hat{y}_i, y_i) = \frac{\partial L(\hat{y}_i, y_i)}{\partial p_i}\frac{\partial p_i}{\partial \hat{y}_i} = -\sum_{i}^{\#instances}\left[\frac{y_i - p_i}{p_i.(1-p_i)}\right] \times p_i.(1-p_i)$$
$$= -\sum_{i}^{\#instances}(y_i - p_i)$$

Et

$$h_i = \frac{\partial^2}{\partial {\hat{y}_i}^2}.L(\hat{y}_i, y_i) = -\frac{\partial}{\partial \hat{y}_i}\sum_{i}^{\#instances}(y_i - p_i) = \sum_{i}^{\#instances} p_i.(1-p_i)$$

Soit

$$g_i = -\sum_{i}^{\#instances}(y_i - p_i) \; et \; h_i = \sum_{i}^{\#instances} p_i.(1-p_i)$$

Avec ces valeurs on obtient :

la valeur optimale des feuilles :

$$w_j = -\frac{\sum_{i\in j} g_i}{\sum_{i\in j} h_i + \lambda} = \frac{\sum_{i\in j}(y_i - p_i)}{\sum_{i\in j} p_i.(1-p_i) + \lambda}$$

la perte sur une seule feuille :

$$-\frac{1}{2}.\frac{(\sum_{i\in j} y_i - p_i)^2}{\sum_{i\in j} p_i.(1-p_i) + \lambda} + \gamma$$

## Application numérique - Régression

Pour l'application numérique j'ai repris l'illustration utilisée dans cette vidéo : https://www.youtube.com/watch?v=OtD8wVaFm6E

Soit le jeu de donnée suivant, qui mesure l'efficacité d'un médicament en fonction de son dosage. Soit x le dosage et y la mesure de l'efficacité.

L'objectif est de construire une suite d'arbre pour prédire en fonction de x la valeur de y.

| X (dosage) | 10 | 20 | 25 | 35 |
|---|---|---|---|---|
| Y (efficacité) | -10 | 7 | 8 | -7 |

Ce jeu de données est simpliste : 4 observations et 1 feature. Malgré sa simplicité, cela permettra d'illustrer parfaitement l'algorithme d'apprentissage de XGBoost pour la régression.

Pour rappel, les étapes qui seront suivies :

- étape 0 : 1er estimateur qui est une constante $\hat{y}^{(0)}$
- étape 1 : calcul de résidus (différence entre $y_i$ et la dernière estimation $\hat{y}^{(0)}$)
- étape 2 : construction de l'arbre #1 permettant de calculer les estimations $\hat{y}_i{}^{(0)}$
    - A chaque niveau de l'arbre, on identifie le split optimal défini par le gain maximum
    - Lorsque l'arbre est construit, on calcule les estimations de chacune des feuilles
    - On itère jusqu'à ce qu'il n'y ait plus de split possible pour l'arbre qui est en cours de construction
- Etape 3 : on calcule les résidus permettant de construire un nouvel arbre, l'arbre #2
- Et ainsi de suite

Etape 0 : on définit le premier estimateur $\hat{y}^{(0)}$= 0.5 (par exemple)

Etape 1 : on calcule les résidus

| X (dosage) | 10 | 20 | 25 | 35 |
|---|---|---|---|---|
| Y (efficacité) | -10 | 7 | 8 | -7 |
| 1er estimation : $\hat{y}^{(0)}$ | 0.5 | 0.5 | 0.5 | 0.5 |
| Résidus : $y_i - \hat{y}^{(0)}$ | -10.5 | 6.5 | 7.5 | -7.5 |

Etape 2 : Construction du 1er arbre à partir des résidus

On cherche quel split en x va donner le score maximum, défini par :

$$Score = \frac{(\sum_{i \in FG} g_i)^2}{\sum_{i \in FG} h_i + \lambda} + \frac{(\sum_{i \in FD} g_i)^2}{\sum_{i \in FD} h_i + \lambda} - \frac{(\sum_{i \in FI} g_i)^2}{\sum_{i \in FI} h_i + \lambda} - 2.\gamma$$

Ou, en remplaçant par les expressions des gradients et hessiens dans le cas de la régressions

$$Score = \frac{(\sum y_i - \hat{y}_i)^2}{\underbrace{Nb\ Instances + \lambda}_{Feuille\ Gauche}} + \frac{(\sum y_i - \hat{y}_i)^2}{\underbrace{Nb\ Instances + \lambda}_{Feuille\ droite}} - \frac{(\sum y_i - \hat{y}_i)^2}{\underbrace{Nb\ Instances + \lambda}_{Feuille\ initiale}} - 2.\gamma$$

Pour calculer ces scores, il nous faut donc calculer les scores de la feuille initiale (F.I), et des feuilles gauche (F.G) et droite (F.D) après split. On modulera également avec les hyper-paramètres $\lambda$ et $\gamma$.

Cas 1 : on prend $\lambda$ et $\gamma$ = 0.

Split du 1er niveau de l'arbre – comparons le score de chacun des splits. On a 4 valeurs dans cet exemple, donc 3 splits possibles

| | Split 1.1 (x < 15) | Split 1.2 (x < 22.5) | Split 1.3 (x < 30) |
|---|---|---|---|
| Feuille initiale - Résidus | [-10.5, 6.5, 7.5, -7.5] | | |
| Feuille initiale – Score ($\lambda = 0$) | $\frac{(-10.5 + 6.5 + 7.5 - 7.5)^2}{4} = 4$ | | |
| Feuille gauche - Résidus | [-10.5] | [-10.5, 6.5] | [-10.5, 6.5, 7.5] |
| Feuille gauche – Score ($\lambda = 0$) | $\frac{(-10.5)^2}{1} = 110.25$ | $\frac{(-10.5 + 6.5)^2}{2} = 8$ | $\frac{(-10.5 + 6.5 + 7.5)^2}{3} = 4{,}08$ |
| Feuille droite - Résidus | [6.5, 7.5, -7.5] | [7.5, -7.5] | [-7.5] |
| Feuille droite – Score ($\lambda = 0$) | $\frac{(6.5 + 7.5 - 7.5)^2}{3} = 14{,}08$ | $\frac{(7.5 - 7.5)^2}{2} = 0$ | $\frac{(-7.5)^2}{1} = 56.25$ |
| Total Score ($\lambda = 0, \gamma = 0$) | 110.25 + 14.08 – 4 = 120.33 | 8 + 0 – 4 = 4 | 4.08 + 56.25 – 4 = 56.33 |

Le split 1.1 (x<15) a le plus fort score, il est donc conservé dans la construction de l'arbre au premier niveau

Split du 2éme  niveau de l'arbre – split de la feuille de droite qu'on vient d'obtenir

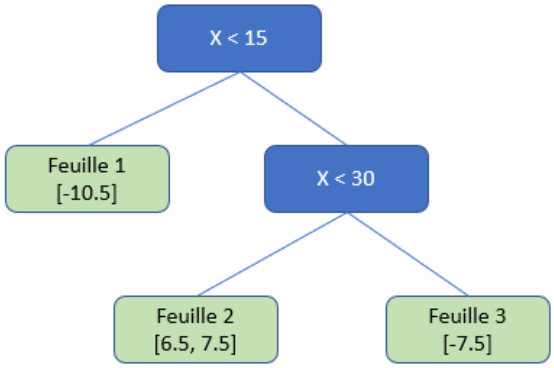
| | Split 2.1 (x < 22.5) | Split 2.2 (x < 30) |
|---|---|---|
| Feuille initiale – Résidus | [6.5, 7.5, -7.5] | |
| Feuille initiale – Score ($\lambda = 0$) | $\frac{(6.5 + 7.5 - 7.5)^2}{3} = 14.08$ | |
| Feuille gauche - Résidus | [6.5] | [-6.5, 7.5] |
| Feuille gauche – Score ($\lambda = 0$) | $\frac{(6.5)^2}{1} = 42.25$ | $\frac{(6.5 + 7.5)^2}{2} = 98$ |
| Feuille droite - Résidus | [7.5, -7.5] | [-7.5] |
| Feuille droite – Score ($\lambda = 0$) | $\frac{(7.5 - 7.5)^2}{2} = 0$ | $\frac{(-7.5)^2}{1} = 56.25$ |
| Total Score ($\lambda = 0, \gamma = 0$) | 42.25 + 0 – 14,08 = 28.17 | 98 + 56.25 – 14,08 = 140.17 |

Le split 2.2 (x<30) a le plus fort gain, il est donc conservé dans la construction de l'arbre

On suppose maintenant que notre arbre est complétement construit, on calcule maintenant les estimations optimales par feuille. On se sert des résidus restant par feuille

Formule à utiliser :

$$w_j = -\frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i + \lambda} = \frac{\sum_{i \in j} y_i - \hat{y}_i}{Nb\ Instances + \lambda}$$




| Feuille | Résidus | Output ($\lambda = 0$) |
|---|---|---|
| Feuille 1 | [-10.5] | $\frac{-10.5}{1} = -10.5$ |
| Feuille 2 | [6.5, 7.5] | $\frac{6.5 + 7.5}{2} = 7$ |
| Feuille 3 | [-7.5] | $\frac{-7.5}{1} = -7.5$ |

Maintenant qu'on a les prédictions du premier arbre, on peut calculer les nouveaux résidus afin de construire un nouvel arbre

Etape 3 : on calcule les nouveaux résidus suite à la construction du nouvel arbre

| X (dosage) | 10 | 20 | 25 | 35 |
|---|---|---|---|---|
| Y (efficacité) | -10 | 7 | 8 | -7 |
| 1er estimation : $\hat{y}^{(0)}$ | 0.5 | 0.5 | 0.5 | 0.5 |
| Résidus : $y_i - \hat{y}^{(0)}$ | -10.5 | 6.5 | 7.5 | -7.5 |
| $\varphi_1(x_i)$ $(\lambda = 0)$ | -10.5 | 7 | 7 | -7.5 |
| $\hat{y}^{(1)} = \hat{y}^{(0)} + \varphi_1(x_i)$ $(\lambda = 0)$ | 0.5-10.5 | 0.5+7 | 0.5+7 | 0.5-7.5 |

Cas 2 : on prend $\lambda$ = 1 et $\gamma$ = 0.

Reprenons le raisonnement complet du cas 1)

Split du 1er niveau de l'arbre

| | Split 1.1 (x < 15) | Split 1.2 (x < 22.5) | Split 1.3 (x < 30) |
|---|---|---|---|
| Feuille initiale - Résidus | [-10.5, 6.5, 7.5, -7.5] | | |
| Feuille initiale – Score $(\lambda = 1)$ | $\frac{(-10.5 + 6.5 + 7.5 - 7.5)^2}{4 + 1} = 3.2$ | | |
| Feuille gauche - Résidus | [-10.5] | [-10.5, 6.5] | [-10.5, 6.5, 7.5] |
| Feuille gauche – Score $(\lambda = 1)$ | $\frac{(-10.5)^2}{2} = 55.12$ | $\frac{(-10.5 + 6.5)^2}{3} = 5{,}33$ | $\frac{(-10.5 + 6.5 + 7.5)^2}{4} = 3{,}06$ |
| Feuille droite - Résidus | [6.5, 7.5, -7.5] | [7.5, -7.5] | [-7.5] |
| Feuille droite – Score $(\lambda = 1)$ | $\frac{(6.5 + 7.5 - 7.5)^2}{4} = 10.56$ | $\frac{(7.5 - 7.5)^2}{3} = 0$ | $\frac{(-7.5)^2}{2} = 28{,}1$ |
| Total Score $(\lambda = 1, \gamma = 0)$ | 55.12 + 10.56 -3.2 = 62.48 | 5.38 -3,2 = 2,18 | 3,06 + 28,1 – 3,2 = 27,96 |

On voit que c'est toujours le split 1.1 (x<15) qui a le plus fort score, il est donc conservé dans la construction de l'arbre au premier niveau.

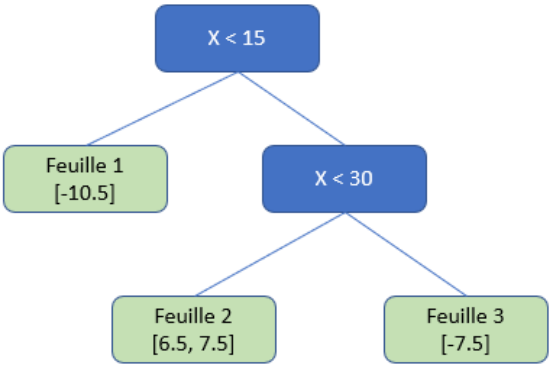
Par contre, l'ajout d'une pénalisation $\lambda$ = 1 abaisse fortement les scores, ce qui amène à des arbres beaucoup moins profonds que sans pénalisation.

C'est bien le but recherché : avoir des arbres moins profonds permettant de limiter les risques de sur apprentissage.

On calcule maintenant les estimations optimales par feuille, avec $\lambda$ = 1.

Formule à utiliser :

$$w_j = -\frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i + \lambda} = \frac{\sum_{i \in j} y_i - \hat{y}_i}{Nb\ Instances + \lambda}$$




| Feuille | Résidus | Output ($\lambda = 0$) | Output ($\lambda = 1$) |
|---|---|---|---|
| Feuille 1 | [-10.5] | $\frac{-10.5}{1} = -10.5$ | $\frac{-10.5}{2} = -5.25$ |
| Feuille 2 | [6.5, 7.5] | $\frac{6.5 + 7.5}{2} = 7$ | $\frac{6.5 + 7.5}{3} = 4.6$ |
| Feuille 3 | [-7.5] | $\frac{-7.5}{1} = -7.5$ | $\frac{-7.5}{2} = -3.75$ |

On voit que l'ajout d'une pénalité décroit systématiquement la valeur de la prédiction. La convergence vers la valeur cible est plus lente, et c'est aussi le but recherché de la pénalisation.

Cas 3 : $\gamma > 0$

On rappelle que $\gamma$ intervient dans la formule du score pour le choix du scoring.

$$Score = \underbrace{\frac{(\sum y_i - \hat{y}_i)^2}{Nb\ Instances + \lambda}}_{Feuille\ Gauche} + \underbrace{\frac{(\sum y_i - \hat{y}_i)^2}{Nb\ Instances + \lambda}}_{Feuille\ droite} - \underbrace{\frac{(\sum y_i - \hat{y}_i)^2}{Nb\ Instances + \lambda}}_{Feuille\ initiale} - 2.\gamma$$

Avec $\gamma > 0$, cela réduit les possibilités qu'un split soit valide (il faut que le score soit positif).

Et donc par conséquent, cela limite la profondeur des arbres, et donc les risques de sur-apprentissage.

## Application numérique - Classification

Pour illustrer la construction d'un modèle dans le cas d'une classification, nous partirons du jeu de données suivant.

L'objectif est de construire une suite d'arbre pour prédire en fonction de x la valeur de y.

| X (dosage) | 2 | 8 | 11 | 17 |
|---|---|---|---|---|
| Y (efficacité) | 0 | 1 | 1 | 0 |

Ces données nous disent que le médicament étudié est efficace (Y = 0) seulement si le dosage est dans une plage bornée.

Pour construire le modèle, nous effectuons exactement les mêmes étapes que dans le cas de la régression.

- étape 0 : 1er estimateur qui est une constante $\hat{y}^{(0)}$
- étape 1 : calcul de résidus (différence entre $y_i$ et la dernière estimation $\hat{y}^{(0)}$)
- étape 2 : construction de l'arbre #1 permettant de calculer les estimations $\hat{y}_i^{(0)}$
    - A chaque niveau de l'arbre, on identifie le split optimal défini par le gain maximum
    - Lorsque l'arbre est construit, on calcule les estimations de chacune des feuilles
- Etape 3 : on calcule les résidus permettant de construire un nouvel arbre, l'arbre #2
- Et ainsi de suite …

Etape 0 : on définit le premier estimateur $\hat{y}^{(0)}$= 0.5 (par exemple)

Etape 1 : on calcule les résidus

| X (dosage) | 2 | 8 | 11 | 17 |
|---|---|---|---|---|
| Y (efficacité) | 0 | 1 | 1 | 0 |
| 1er estimation : $\hat{y}^{(0)}$ | 0.5 | 0.5 | 0.5 | 0.5 |
| Résidus : $y_i - \hat{y}^{(0)}$ | -0.5 | 0.5 | 0.5 | -0.5 |

Etape 2 : Construction du 1er arbre à partir des résidus

On cherche quel split en x va donner le score maximum, défini par :

$$Score = \frac{(\sum_{i \in FG} g_i)^2}{\sum_{i \in FG} h_i + \lambda} + \frac{(\sum_{i \in FD} g_i)^2}{\sum_{i \in FD} h_i + \lambda} - \frac{(\sum_{i \in FI} g_i)^2}{\sum_{i \in FI} h_i + \lambda} - 2.\gamma$$

Ou, en remplaçant par les expressions des gradients et hessiens dans le cas de la régressions

$$Score = \frac{(\sum y_i - \hat{y}_i)^2}{\underbrace{Nb\ Instances + \lambda}_{Feuille\ Gauche}} + \frac{(\sum y_i - \hat{y}_i)^2}{\underbrace{Nb\ Instances + \lambda}_{Feuille\ droite}} - \frac{(\sum y_i - \hat{y}_i)^2}{\underbrace{Nb\ Instances + \lambda}_{Feuille\ initiale}} - 2.\gamma$$

Pour calculer ces scores, il nous faut donc calculer les scores de la feuille initiale (F.I), et des feuilles gauche (F.G) et droite (F.D) après split. On modulera également avec les hyper-paramètres $\lambda$ et $\gamma$.

On remarquera que la formule diffère de celle de la régression par les termes aux dénominateurs des scores. Plutôt que d'avoir la somme des instances, on a la somme du produit des probabilités 1 et 0.

Split du 1er niveau de l'arbre

| | Split (x < 15) |
|---|---|
| Feuille initiale - Résidus | [-0.5, 0.5, 0.5,-0.5] |
| Feuille initiale – Score ($\lambda = 0$) | $\frac{(-0.5 + 0.5 + 0.5 - 0.5)^2}{0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5)} = 0$ |
| Feuille gauche - Résidus | [-0.5, 0.5, 0.5] |
| Feuille gauche – Score ($\lambda = 0$) | $\frac{(-0.5 + 0.5 + 0.5)^2}{0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5)} = 0.33$ |
| Feuille droite - Résidus | [-0.5] |
| Feuille droite – Score ($\lambda = 0$) | $\frac{(-0.5)^2}{0.5 \times (1 - 0.5)} = 1$ |
| Total Score ($\lambda = 0, \gamma = 0$) | 0.33 + 1 – 0 = 1.33 |

On peut montrer facilement en suivant les mêmes calculs que c'est ce split qui donne le score maximum

Split du 2er niveau de l'arbre

Ici la feuille initiale est la feuille gauche du 1er niveau.

| | Split (x < 5) & (x < 15) |
|---|---|
| Feuille initiale - Résidus | [-0.5, 0.5, 0.5] |
| Feuille initiale – Score ($\lambda = 0$) | $\frac{(-0.5 + 0.5 + 0.5)^2}{0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5) + 0.5 \times (1 - 0.5)} = 0.33$ |
| Feuille gauche - Résidus | [-0.5] |

| | |
|---|---|
| Feuille gauche – Score ($\lambda = 0$) | $\frac{(-0.5)^2}{0.5 \times (1-0.5)} = 1$ |
| Feuille droite - Résidus | [0.5, 0.5] |
| Feuille droite – Score ($\lambda = 0$) | $\frac{(0.5+0.5)^2}{0.5 \times (1-0.5) + 0.5 \times (1-0.5)} = 2$ |
| Total Score ($\lambda = 0, \gamma = 0$) | 1 + 2 – 0.33 = 0.66 |

Apparemment, l'algorithme fait intervenir la notion de « cover » qui est un hyper-paramètre. Si jamais les dénominateurs des scores des feuilles droite ou gauche sont inférieurs à cette valeur de « Cover », le split ne se fait pas. Or par défaut, la valeur de « cover » vaut 1. Cette valeur est génante pour le cas de la classification. Il faut la mettre à 0.

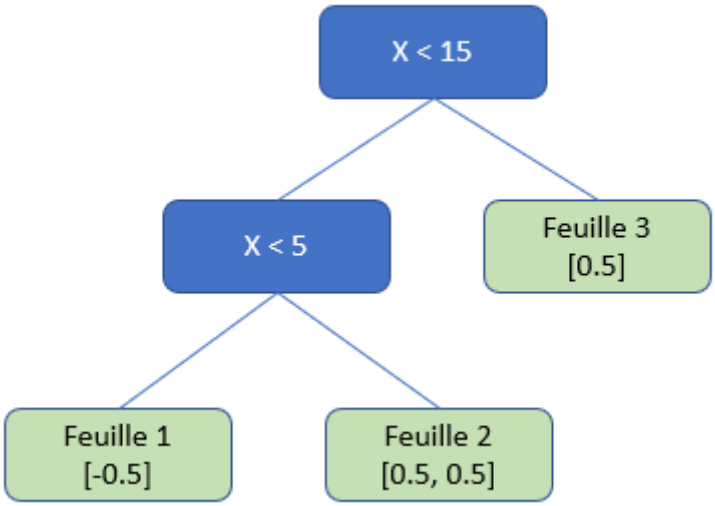
[Rechercher la référence de cette notion]

Note : pour la régression cela n'est pas impactant (car cela revient à avoir au moins une instance, ce qui est toujours vrai)

On calcule maintenant les estimations optimales par feuille, avec $\lambda$ = 1.

Formule à utiliser :

$$w_j = -\frac{\sum_{i \in j} g_i}{\sum_{i \in j} h_i + \lambda} = \frac{\sum_{i \in j} y_i - \hat{y}_i}{\sum p_i.(1 - p_i) + \lambda}$$




| Feuille | Résidus | Output ($\lambda = 0$) | Output ($\lambda = 1$) |
|---|---|---|---|
| Feuille 1 | [-0.5] | $\frac{-0.5}{0.5 \times (1-0.5)} = -2$ | $\frac{-0.5}{0.5 \times (1-0.5) + 1} = -0.4$ |
| Feuille 2 | [0.5, 0.5] | $\frac{0.5+0.5}{0.5 \times (1-0.5) + 0.5 \times (1-0.5)} = 2$ | $\frac{0.5+0.5}{0.5 \times (1-0.5) + 0.5 \times (1-0.5) + 1} = 0.67$ |
| Feuille 3 | [-0.5] | $\frac{-0.5}{0.5 \times (1-0.5)} = -2$ | $\frac{-0.5}{0.5 \times (1-0.5) + 1} = -0.4$ |

Maintenant qu'on a les prédictions du premier arbre, on peut calculer les nouveaux résidus afin de construire un nouvel arbre

Etape 3 : on calcule les nouveaux résidus suite à la construction du nouvel arbre

| X (dosage) | 2 | 8 | 11 | 17 |
|---|---|---|---|---|
| Y (efficacité) | 0 | 1 | 1 | 0 |
| 1er estimation : $\hat{y}^{(0)}$ | 0.5 | 0.5 | 0.5 | 0.5 |
| Résidus : $y_i - \hat{y}^{(0)}$ | -0.5 | 0.5 | 0.5 | -0.5 |
| $\varphi_1(x_i)$ $(\lambda = 0)$ | -2 | 2 | 2 | -2 |

Attention, ici on a une subtilité pour calculer les estimations à la première iteration.

Il ne faut pas oublier que dans la construction du modèle, on a choisit cette fonction de mesure du Loss

$$L = -\sum_{i}^{\#instances} (y_i.\log(p_i) + (1 - y_i).\log(1 - p_i))$$

Et que

$$p_i = \frac{1}{1 + e^{-\hat{y}_i}}$$

Soit

$$\hat{y}_i = log\frac{p_i}{1 - p_i} = \log\_odds(p_i)$$

On reprend alors le tableau :

| X (dosage) | 2 | 8 | 11 | 17 |
|---|---|---|---|---|
| Y (efficacité) | 0 | 1 | 1 | 0 |
| 1er estimation : $\hat{y}^{(0)}$ | 0.5 | 0.5 | 0.5 | 0.5 |
| Résidus : $y_i - \hat{y}^{(0)}$ | -0.5 | 0.5 | 0.5 | -0.5 |
| $\varphi_1(x_i)$ $(\lambda = 0)$ | -2 | 2 | 2 | -2 |
| Log_odds($\hat{y}^{(0)}$) | $log\frac{0.5}{1-0.5} = 0$ | $log\frac{0.5}{1-0.5} = 0$ | $log\frac{0.5}{1-0.5} = 0$ | $log\frac{0.5}{1-0.5} = 0$ |
| $p_i = \frac{1}{1 + e^{-\varphi_1(x_i)}}$ | $\frac{1}{1+e^2} = 0.11$ | $\frac{1}{1+e^{-2}} = 0.88$ | $\frac{1}{1+e^{-2}} = 0.88$ | $\frac{1}{1+e^2} = 0.11$ |

## Learning rate

Classiquement, la descente de gradient sur d'autres types d'algo fait appel à un paramètre $\alpha$ qui module la modification des paramètres en train d'être appris, de sorte que l'algorithme converge tranquillement.

XGBoost intègre également ce type de paramètre lors de la phase d'apprentissage, mais aussi lors de la phase d'inférence (à vérifier). Le paramètre s'appelle Shrinkage ou Eta.

Donc une fois que l'arbre est construit dans la phase d'apprentissage, on module la valeur trouvée $w_j$ par le learning rate pour déterminer les résidus utilisés dans la construction du nouvel arbre.

## Optimisation pour la recherche des splits

Nous l'avons vu, la recherche de split peut être longue et couteuse en ressources. Il s'agit effectivement de tester pour l'ensemble des features quel split donne le meilleur score (ou gain).

Ainsi imaginons que nous ayons n features, chacune comportant m valeurs uniques en moyenne, il faudrait faire n x (m-1) calculs de score pour tous les splits à un niveau donné de l'arbre => algorithme gourmand (« greedy »).

Au lieu de cela, les auteurs proposent de faire les splits à partir de quantiles représentant la distribution statistique des données.

Imaginons que nous ayons une feature qui comporte 500.000 valeurs différentes. L'algorithme qui teste toutes le valeurs (algo greedy) ferait 499.999 calculs. Maintant imaginons que nous regroupions les données suivant des quantiles. Par exemple, si on prend 10 quantiles, il y aurait 10 groupes de données suivant leur position dans la distribution

- Quantile 1 : les valeurs appartenant aux 10% des valeurs les plus basses
- Quantile 2 : les 10% suivantes
- Etc …

Ainsi, les splits seront testés aux frontières de ces quantiles, ce qui réduit drastiquement les temps de calcul – 9 tests pour 10 groupes.

Et même si le split idéal a de très faible chance d'être trouvé, l'approximation faite sera suffisante.

Une valeur que l'on retrouve souvent est de 33 quantiles [Référence à trouver]

## Algorithmes pour les valeurs manquantes

Maintenant imaginons que la feature sur laquelle on veut tester le split comporte de nombreuses valeurs manquantes (valeurs nulles). L'algorithme a un fonctionnement assez original avec ces données qui donne de bons résultats.

L'idée est que si une instance est testée et que la feature est nulle, l'algorithme aura une branche par défaut pour cette instance. Toutes les instances avec une feature nulle passeront par cette branche (gauche ou droite).

Maintenant, comment déterminer quelle est la branche par défaut ?

Pendant la phase d'apprentissage, lors de la détermination d'un nouveau split sur l'arbre en cours, l'algorithme se comportera comme cela :

- Il construira son split sur cette feature en prenant en compte que les valeurs non nulles
- Une valeur sera déterminée pour le split (comme vu prédemment)
- Ensuite l'algorithme fera deux nouveaux calculs – un premier calcul de score en mettant toutes les features nulles sur la branche de gauche, et un calcul en les mettant sur la branche de droite. La branche ayant le plus fort score sera la branche par défaut pour toutes les instances qui auront cette instance manquante.

# Références


1. https://arxiv.org/abs/1603.02754
2. https://xgboost.readthedocs.io/en/latest/index.html
3. https://github.com/dmlc/xgboost/tree/master/demo
4. XGBoost Regressor: https://www.youtube.com/watch?v=OtD8wVaFm6E
5. XGBoost Part 1: XGBoost Trees for Regression: https://www.youtube.com/watch?v=OtD8wVaFm6E
6. XGBoost Part 2: XGBoost Trees For Classification: https://www.youtube.com/watch?v=8b1JEDvenQU
7. XGBoost Part 3: Mathematical Details: https://www.youtube.com/watch?v=ZVFeW798-2I
8. XGBoost Part 4: Crazy Cool Optimizations: https://www.youtube.com/watch?v=oRrKeUCEbq8