

# INTRODUCTION AU MACHINE LEARNING

## BOOSTING

**Théo Lopès-Quintas**

BPCE Payment Services,  
Université Paris Dauphine

2023

# NOTATIONS

## MODÉLISATION DE L'APPRENTISSAGE SUPERVISÉ

Pour chaque problème de Machine Learning (ML) on dispose d'un dataset  $\mathcal{D}$  :

$$\mathcal{D} = \left\{ (x^{(i)}, y_i) \mid \forall i \leq n, x^{(i)} \in \mathbb{R}^d, y_i \in \mathcal{Y} \right\} \quad (\text{Dataset})$$

Nombre d'observations (pointing to  $n$ )

Nombre d'informations (pointing to  $d$ )

Et on suppose que :

$$\exists f : \mathbb{R}^d \rightarrow \mathcal{Y}, \forall i \leq n, y_i = f(x^{(i)}) \quad (\text{Hypothèse})$$

Autrement dit : il existe une fonction qui *explique* la variable d'intérêt ( $y$ ) à partir des informations du dataset ( $X$ ).

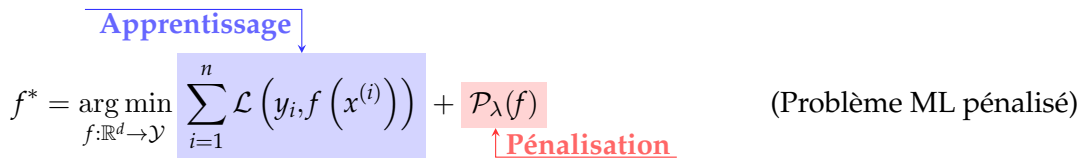
## NOTATIONS

### FONCTION DE PERTE

On appelle la **fonction de perte** :  $\mathcal{L} : \mathcal{Y} \times \mathcal{Y}$  une fonction qui permet de mesurer la performance d'apprentissage d'un modèle ML pour une tâche donnée.

$$f^* = \arg \min_{f: \mathbb{R}^d \rightarrow \mathcal{Y}} \sum_{i=1}^n \mathcal{L} \left( y_i, f \left( x^{(i)} \right) \right) \quad (\text{Problème ML})$$

On peut régulariser le problème en ajoutant une pénalisation :


$$f^* = \arg \min_{f: \mathbb{R}^d \rightarrow \mathcal{Y}} \sum_{i=1}^n \mathcal{L} \left( y_i, f \left( x^{(i)} \right) \right) + \mathcal{P}_\lambda(f) \quad (\text{Problème ML pénalisé})$$

On pense par exemple à la régression Ridge ou LASSO.

# GRADIENT BOOSTING

## BOOSTING

A l'inverse des méthodes ensembliste où chaque weak learners (notons  $f_m$  le  $m$ -ième) apprend indépendamment des autres weak learners pour former le strong learner (notons  $F$ ), dans le Boosting les weak learners apprennent séquentiellement. Chaque weak learners cherchent à améliorer la performance du précédent weak learner. On cherche donc à chaque étape à apprendre le modèle  $h_m$  tel que :

$$\begin{aligned}\forall i \leq n, f_m(x^{(i)}) &= f_{m-1}(x^{(i)}) + h_m(x^{(i)}) = y_i \\ \iff \\ \forall i \leq n, h_m(x^{(i)}) &= y_i - f_{m-1}(x^{(i)})\end{aligned}$$

Autrement dit, on cherche à apprendre à partir des résidus : les erreurs des précédents modèles.

# GRADIENT BOOSTING

## GRADIENT BOOSTING : POURQUOI GRADIENT ?

### Exercice 1 (Descente de gradient et résidus)

Soit la fonction de perte  $\mathcal{L}(y, f(x)) = (y - f(x))^2$  et la fonction de coût  $\mathcal{C}(y, f(x)) = \sum_{i=1}^n \mathcal{L}(y_i, f(x^{(i)}))$ .

Montrer que :

$$-\frac{\partial \mathcal{C}}{\partial f_{m-1}(x^{(i)})} \left( y_i, f_{m-1}(x^{(i)}) \right) = \frac{2}{n} h_m(x^{(i)})$$

Le résultat de cet exercice se généralise, il y a un lien entre l'opposé du gradient de la fonction de coût et les résidus. Ainsi, si l'on compile les différentes équations que l'on a écrites jusqu'à présent on a :

$$\begin{aligned} f_m(x) &= f_{m-1}(x) - \gamma \sum_{i=1}^n \frac{\partial \mathcal{C}}{\partial f_{m-1}(x^{(i)})} \left( y_i, f_{m-1}(x^{(i)}) \right) \\ &= f_{m-1}(x) - \gamma' h_m(x) \end{aligned}$$

Cependant la méthode est longue à l'apprentissage et de nombreuses améliorations sont possibles. Nous parlerons ici spécifiquement d'XGBoost.

# XGBOOST TREES

## PROBLÈME DE DÉPART

La prédiction initiale pour n'importe quel problème XGBoost est 0.5. Prédire par une constante arbitraire n'est pas performant, donc nous devons apprendre un nouvel arbre qui va améliorer cette prédiction. On cherche à minimiser :

$$O_{\text{value}}^* = \arg \min_{O_{\text{value}} \in \mathbb{R}} \sum_{i=1}^n \mathcal{L} \left( y_i, f_{m-1} \left( x^{(i)} \right) + O_{\text{value}} \right) + \frac{1}{2} \lambda O_{\text{value}}^2 + \gamma T \quad (1)$$

Résoudre ce problème est difficile parce que la fonction de perte est générale et peut être coûteuse à évaluer. Il nous faut trouver une manière de contourner le problème.

# XGBOOST TREES

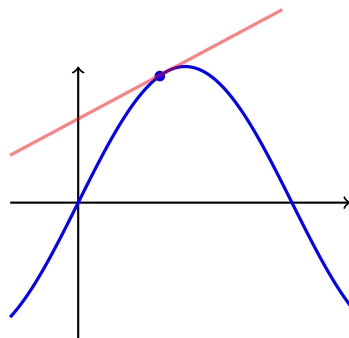
## DÉVELOPPEMENT DE TAYLOR

### Théorème 1 (Taylor)

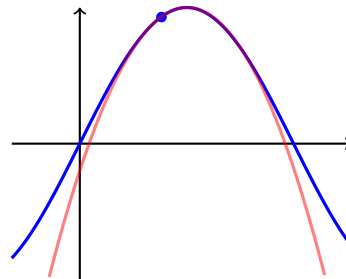
Soit  $I \subset \mathbb{R}$  et  $a \in I$ . Soit  $f : I \mapsto \mathbb{R}$  une fonction  $n$ -fois dérivable en  $a$ . Alors :

$$f(x) = \sum_{k=0}^n \frac{f^{(k)}(a)}{k!} (x - a)^k + R_n(x)$$

Avec le reste  $R_n(x)$  négligeable devant  $(x - a)^n$  au voisinage de  $a$ .



(a) Développement à l'ordre 1



(b) Développement à l'ordre 2

**Figure** – Développement de Taylor pour  $f(x) = 2 \sin(x)$  au point  $x = 1.3$

# XGBOOST TREES

## NOUVELLE FONCTION DE COÛT

### Exercice 2 (Nouvelle fonction de coût)

Nous reprenons l'ensemble des notations définies jusqu'à présent.

1. Soit  $f : \mathcal{I} \mapsto \mathbb{R}$  une fonction  $n$  fois dérivable,  $a \in I$  et  $h \in \mathbb{R}$  tel que  $a + h \in I$ . Justifier :

$$f(a + h) = \sum_{k=0}^n \frac{f^{(k)}(a)}{k!} h^k + R_n(h)$$

Avec  $R_n(x)$  une fonction négligeable devant  $h^n$  au voisinage de 0.

2. A l'aide de l'expression précédente, proposer une approximation à l'ordre 2 de l'expression :

$$\phi \left( f_{m-1} \left( x^{(i)} \right) + O_{value} \right) = \sum_{i=1}^n \mathcal{L} \left( y_i, f_{m-1} \left( x^{(i)} \right) + O_{value} \right)$$

Où  $\mathcal{L}$  est une fonction dérivable deux fois sur  $\mathbb{R}$ .

3. Nous obtenons une approximation du problème du choix de la meilleure valeur  $O_{value}$ . Identifier les termes constants et commenter sur la vitesse de calcul par rapport à la méthode classique.



# XGBOOST TREES

## VALEUR OPTIMALE

Solution pour la question 3 en reprenant l'expression précédente et en écrivant en **rouge** les termes constants pour chacune des itérations, on a :

$$O_{\text{value}}^* = \arg \min_{O_{\text{value}} \in \mathbb{R}} \sum_{i=1}^n \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) + \nabla \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) O_{\text{value}} + \frac{1}{2} \nabla^2 \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) O_{\text{value}}^2 + \frac{1}{2} \lambda O_{\text{value}}^2$$

D'où on a que :

$$O_{\text{value}}^* = - \frac{\sum_{i=1}^n \nabla \mathcal{L}(y_i, \hat{y}_i^{(m-1)})}{\sum_{i=1}^n \nabla^2 \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) + \lambda}$$

## XGBOOST TREES

### SCORE DE SIMILARITÉ

Nous savons à présent correctement identifier la meilleure valeur possible pour une feuille donnée  $O_{\text{value}}$ , nous devons avoir un score qui nous permette de trouver la meilleure coupure pour que nous puissions faire grandir l'arbre.

Appuyons nous sur la forme qu'optimise  $O_{\text{value}}$  pour cela :

$$\Phi(O_{\text{value}}) = \sum_{i=1}^n \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) + \nabla \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) O_{\text{value}} + \frac{1}{2} \nabla^2 \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) O_{\text{value}}^2 + \frac{1}{2} \lambda O_{\text{value}}^2$$

Puisqu'on cherche à maximiser le gain de couper une racine en deux branches, nous devons nous placer dans un cadre de maximisation plutôt que de minimisation. Donc on s'intéresse à  $-\Phi(O_{\text{value}})$  puis en y intégrant la valeur optimale  $O_{\text{value}}^*$  nous sommes capable de construire :

$$\text{Similarity Score} = \frac{\left( \sum_{i=1}^n \nabla \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) \right)^2}{\sum_{i=1}^n \nabla^2 \mathcal{L}(y_i, \hat{y}_i^{(m-1)}) + \lambda}$$

A noter qu'il manque un coefficient  $\frac{1}{2}$  ici : pour éviter les calculs l'implémentation du score de similarité supprime cette constante multiplicative.

# XGBOOST TREES

## RÉGULARISATIONS DE $\lambda$ ET $\gamma$

Si l'on reprend l'expression du score de similarité :

$$\text{Similarity Score} = \frac{\left( \sum_{i=1}^n \nabla \mathcal{L} \left( y_i, \hat{y}_i^{(m-1)} \right) \right)^2}{\sum_{i=1}^n \nabla^2 \mathcal{L} \left( y_i, \hat{y}_i^{(m-1)} \right) + \lambda}$$

On comprend que  $\lambda$  va permettre de régulariser la construction même de l'arbre en jouant directement sur les gains lors des coupures.

Concernant  $\gamma$  son rôle est de contrôler à quel point on va élaguer l'arbre. Dans le problème général que l'on cherche à optimiser :

$$O_{\text{value}}^* = \arg \min_{O_{\text{value}} \in \mathbb{R}} \sum_{i=1}^n \mathcal{L} \left( y_i, f_{m-1} \left( x^{(i)} \right) + O_{\text{value}} \right) + \frac{1}{2} \lambda O_{\text{value}}^2 + \gamma T \quad (2)$$

Ici  $\gamma$  cherche à s'assurer que l'on ne va pas créer des arbres trop profonds. Mais même si on impose que  $\gamma = 0$ , alors si le score de similarité est négatif la branche ne sera pas créée. En pratique, une fois que l'arbre a été appris, une branche ne reste que si la différence entre son score de similarité moins  $\gamma$  est positif. Ainsi,  $\gamma$  intervient pendant la construction de l'arbre en encourageant l'élaguage et le réalise également une fois que l'arbre est construit.

# XGBOOST

## XGBOOST AU GLOBAL

Si l'on résume :

- ▶ L'algorithme prédit 0.5 pour chaque observation au départ
- ▶ Pour autant d'itérations que fixé par l'utilisateur, des arbres boostés vont apprendre des erreurs des précédents pour atteindre la meilleure prédiction possible
- ▶ Nous avons deux manières de contrôler le sur-apprentissage avec  $\lambda$  et  $\gamma$ .

Pour obtenir la prédiction finale, on somme l'ensemble des arbres (à commencer par la valeur 0.5) en multipliant la valeur de chaque arbre par le learning rate  $\eta$  (0.3 par défaut). Cette valeur permet aussi de contrôler un peu le sur-apprentissage.

# XGBOOST

## HYPERPARAMÈTRES

- ▶ Paramétrer les arbres :
  - `max_depth` : limiter la profondeur maximale d'un arbre
  - `min_samples_leaf` : nombre minimal d'observations dans une feuille
  - `max_features` : nombre d'informations à considérer pour chaque coupure
- ▶ Paramétrer le boosting :
  - `n_estimators` : nombre d'arbres à construire dans la forêt
  - `learning_rate` : pas de descente pour réduire le poids des arbres successifs
  - `gamma` : gain minimum pour continuer à construire un arbre
  - `lambda` : régularisation  $L_2$  sur les poids. A noter que le paramètre `alpha` permet de régulariser avec  $L_1$ .
  - `subsample` : fraction des données à utiliser pour apprendre chaque weak learner. Si inférieur à 1, alors on obtient une descente de Gradient Stochastique
  - `objective` : nom ou fonction qui sera la fonction de perte à optimiser
- ▶ Pour arrêter plus tôt le boosting :
  - `validation_fraction` : proportion des données d'entraînement à conserver pour tester l'early-stopping
  - `n_iter_no_change` : nombre minimal d'itérations sans améliorations avant d'arrêter l'apprentissage
  - `tol` : valeur minimale de modification de la loss qui déclenche l'arrêt prématuré