

Supervised Machine Learning

Master 1 - MIASHS

Guillaume Metzler

Université Lumière Lyon 2
Laboratoire ERIC, UR 3083, Lyon

guillaume.metzler@univ-lyon2.fr

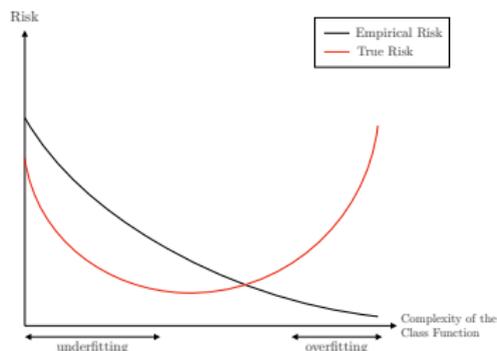
Automne 2022

Retour sur l'Apprentissage

Décomposition de l'erreur I

Retour en arrière

Dans la première partie du cours, on beaucoup parlé de compromis *biais - variance* de notre modèle, l'objectif étant de trouver un modèle performant sur les données d'apprentissage (un biais faible) mais présentant peu de variance afin que celui-ci reste performant sur de nouvelles de données.



Décomposition de l'erreur II

Retour sur la régression

On va tenter d'expliquer d'où vient ce compromis biais-variance si important en apprentissage mais qui sera aussi à l'origine de méthodes d'apprentissage plus avancées.

On se place dans le cadre de la régression et on suppose sur les données sont générées selon le modèle suivant :

$$y = f(\mathbf{x}) + \varepsilon,$$

où la fonction f est bien sûr inconnue, c'est celle que l'on va chercher à estimer à l'aide de nos données, potentiellement bruitées avec cet ajout d'un bruit gaussien ε .

On supposons que l'on dispose d'un jeu de données qui nous permis d'apprendre un modèle h qui approxime f et tentons d'évaluer ses

Décomposition de l'erreur III

performances en généralisation, en se basant sur l'*erreur quadratique moyenne*, *i.e.*

$$\mathbb{E}[(y - h(\mathbf{x}))^2].$$

Décomposition de l'erreur IV

Proposition 1.1: Error Decomposition

We consider the following data generation model

$$y = f(\mathbf{x}) + \varepsilon,$$

Using a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ generated by the previous, we learn a hypothesis h which is an estimator of f . The generalization error of h according to mean squared error can be written :

$$\mathbb{E}[(y - h(\mathbf{x}))^2] = (\mathbb{E}[h(\mathbf{x})] - f(\mathbf{x}))^2 + \mathbb{E} \left[(\mathbb{E}[h(\mathbf{x})] - h(\mathbf{x}))^2 \right] + \mathbb{E}[(y - f(\mathbf{x}))^2].$$

Décomposition de l'erreur V

Démonstration

On utilisera la formule de *König-Huygens* qui dit que pour toute variable aléatoire admettant une variance, on a :

$$\mathbb{E} \left[(X - \mathbb{E}[X])^2 \right] = \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

On cherche à développer le membre de gauche :

$$\begin{aligned} \mathbb{E}[(y - h(\mathbf{x}))^2] &= \mathbb{E}[y^2 - 2yh(\mathbf{x}) + h(\mathbf{x})^2], \\ &\downarrow \text{Linéarité de l'espérance} \\ &= \underbrace{\mathbb{E}[y^2]} - \mathbb{E}[2yh(\mathbf{x})] + \underbrace{\mathbb{E}[h(\mathbf{x})^2]}, \\ &\downarrow \text{formule de } \textit{König-Huygens} \end{aligned}$$

Décomposition de l'erreur VI

$$= \mathbb{E}[y]^2 + \mathbb{E} \left[(y - \mathbb{E}[y])^2 \right] - 2 \mathbb{E}[y] \mathbb{E}[h(\mathbf{x})] \\ + \mathbb{E}[h(\mathbf{x})]^2 + \mathbb{E} \left[(h(\mathbf{x}) - \mathbb{E}[h(\mathbf{x})])^2 \right],$$

↓ on utilise le modèle de génération des données

$$= f(\mathbf{x})^2 + \mathbb{E} \left[(y - f(\mathbf{x}))^2 \right] - 2f(\mathbf{x}) \mathbb{E}[h(\mathbf{x})] \\ + \mathbb{E}[h(\mathbf{x})]^2 + \mathbb{E} \left[(h(\mathbf{x}) - \mathbb{E}[h(\mathbf{x})])^2 \right],$$

↓ linéarité de l'espérance

$$= (\mathbb{E}[h(\mathbf{x})] - f(\mathbf{x}))^2 + \mathbb{E} \left[(\mathbb{E}[h(\mathbf{x})] - h(\mathbf{x}))^2 \right] \\ + \mathbb{E}[(y - f(\mathbf{x}))^2].$$

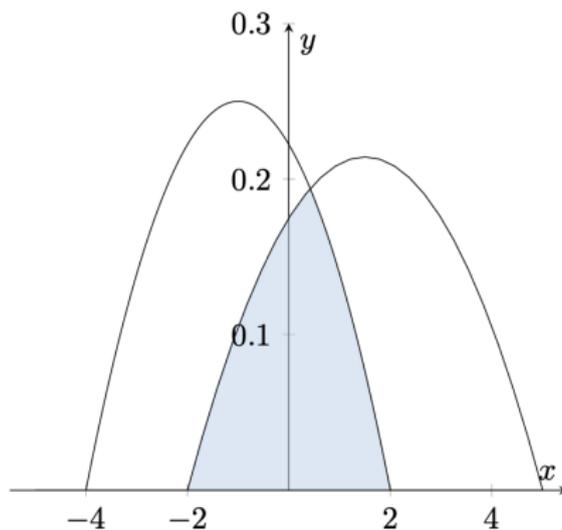
Décomposition de l'erreur VII

Plus que le compromis biais-variance, on voit en fait que notre erreur se décompose en trois termes :

- le carré du **biais** de l'hypothèse h : la différence entre la moyenne de h sur la distribution et la valeur de f .
On fera attention
- le deuxième terme représente la **variance** de l'estimateur h , comment h évolue autour de la moyenne lorsque S , i.e l'échantillon, varie. Cela représente la sensibilité du moodèle aux données.
- le troisième terme représente l'erreur de bayes. Elle ne dépend ni des données, ni du modèle appris.

Décomposition de l'erreur VIII

Erreur de Bayes



Elle correspond à la surface sous l'intersection des deux distributions, *i.e.* l'aire représentée en bleu. Difficile à estimer en pratique.

Décomposition de l'erreur IX

Exercice

On considère deux densités d_1 et d_2 définies ci-dessous et dont la représentation graphique se trouve au slide précédent :

$$d_1 = \begin{cases} \frac{3}{2}x^2 + x & \text{when } 0 \leq x \leq 1, \\ 0 & \text{otherwise.} \end{cases} \quad \text{and} \quad d_2 = \begin{cases} 1 & \text{when } \frac{3}{4} \leq x \leq \frac{7}{4}, \\ 0 & \text{otherwise.} \end{cases}$$

1. Montrer qu'il s'agit bien de deux densités
2. Déterminer l'erreur de bates associés, *i.e.* l'aire sous l'intersection des deux courbes

Décomposition de l'erreur X

L'erreur de Bayes n'est donc pas optimisable et dépend entièrement de la distribution des données, donc ce que l'on cherche à minimiser en général a fin de définir un bon algorithme, c'est ce que l'on appelle **l'excès de risque** :

$$\mathbb{E}[(y - h(\mathbf{x}))^2] - \mathbb{E}[(y - f(\mathbf{x}))^2].$$

Concentrons un peu sur cet excès de risque pour faire le lien avec les bornes en généralisation évoquées plus tôt.

Pour cela, on se rappelle que le rsique d'une hypothèse h , par rapport à une loss ℓ , est défini par

$$\mathcal{R}(h) = \mathbb{E}_{(\mathbf{x}, y) \sim \mathcal{D}} [\ell(y, h(\mathbf{x}))].$$

Décomposition de l'erreur XI

On garde à l'esprit que cette hypothèse h est, en pratique, apprise à partir d'un ensemble S de données.

On suppose également que l'on se fixe un espace d'hypothèses \mathcal{H} (on peut l'analogie avec la régularisation dans nos problèmes d'apprentissage).

Notre précédent **excès de risque**, aussi appelé **regret de Bayes**, peut également s'écrire :

$$\mathcal{R}(h) - \mathcal{R}^* = \left(\mathcal{R}(h) - \inf_{g \in \mathcal{H}} \mathcal{R}(g) \right) + \inf_{g \in \mathcal{H}} \mathcal{R}(g) - \mathcal{R}^*,$$

où \mathcal{R}^* est la plus petite erreur que l'on puisse atteindre, *i.e.* notre risque de Bayes.

Le premier terme est un **excès de risque** vis à vis évalué au sein de l'espace d'hypothèses. Il se peut que l'hypothèse apprise ne soit pas

Décomposition de l'erreur XII

optimale, cela arrive souvent quand les problèmes d'optimisation sont non convexes.

Le dernier terme est un **biais** qui dépend uniquement de l'espace d'hypothèse que l'on considère. On l'appelle aussi **erreur d'approximation**.

Décomposition de l'erreur XIII

Lien avec la généralisation

On va se concentrer sur le premier terme et voir l'information apportée par les bornes en généralisation. Plus précisément, on va voir qu'elle nous permet de borner notre **excès de risque**. Pour cela :

$$h \in \arg \min_{g \in \mathcal{H}} \mathcal{R}_S(g).$$

On appellera $h_{\mathcal{H}}$ le minimiseur du risque \mathcal{R} dans l'espace d'hypothèses \mathcal{H}

$$h_{\mathcal{H}} = \arg \min_{g \in \mathcal{H}} \mathcal{R}(g).$$

Décomposition de l'erreur XIV

L' **excès de risque** : $\mathcal{R}(h) - \inf_{g \in \mathcal{H}} \mathcal{R}(g)$ peut s'écrire comme la somme de trois termes :

$$\begin{aligned} \mathcal{R}(h) - \mathcal{R}(h_{\mathcal{H}}) &= (\mathcal{R}(h) - \mathcal{R}_S(h)) + (\mathcal{R}_S(h) - \mathcal{R}_S(h_{\mathcal{H}})) \\ &\quad + (\mathcal{R}_S(h_{\mathcal{H}}) - \mathcal{R}(h_{\mathcal{H}})), \end{aligned}$$

où $(\mathcal{R}(h) - \mathcal{R}_S(h))$ est la différence entre le risque réel et le risque empirique de l'hypothèse h . On peut facilement borner cette quantité en déterminant une borne sur les performances en généralisation.

Décomposition de l'erreur XV

$(\mathcal{R}_S(h) - \mathcal{R}_S(h_{\mathcal{H}}))$ est négatif par définition. Il suffit de se rappeler de la définition de $h_{\mathcal{H}}$.

$(\mathcal{R}_S(h_{\mathcal{H}}) - \mathcal{R}(h_{\mathcal{H}}))$ est un terme que l'on peut facilement contrôler, car il dépend d'une hypothèse $h_{\mathcal{H}}$ "connue" et de la loi forte des grands nombres.

$(\mathcal{R}(h) - \mathcal{R}_S(h))$ et $(\mathcal{R}_S(h_{\mathcal{H}}) - \mathcal{R}(h_{\mathcal{H}}))$ dépendent donc de la richesse de notre espace d'hypothèses que l'on va noter $\mathfrak{R}(\mathcal{H})$.

Ainsi notre regret de Bayes, peut s'écrire

$$\mathcal{R}(h) - \mathcal{R}^* \leq \inf_{g \in \mathcal{H}} \mathcal{R}(g) - \mathcal{R}^* + \mathfrak{R}(\mathcal{H}).$$

Illustre le compromis **biais-variance** sur la minimisation du risque.

Combinaison de modèles

Approches naïves I

Jusqu'à présent, nous avons présenté des modèles simples, plus ou moins expressifs ou complexes pour nos tâches de régression ou encore de classification. Mais du coup :

- Pourquoi n'utiliser qu'un seul modèle ?
- Est-ce qu'il ne serait pas plus intéressant de combiner plusieurs modèles entre eux.

Les modèles linéaires sont en général beaucoup trop simples pour résoudre des problèmes complexes. Un espoir subsiste si l'on cherche à combiner ces modèles là.

Approches naïves II

Approche naïve

On pourrait imaginer créer T modèles à partir du même jeu de données $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$. Disposant ainsi de plusieurs hypothèses h_t , $t = 1, \dots, T$, nous pourrions les combiner pour former un seul et unique modèle H_T dont le but serait de moyenner les décisions prises par chaque h_t individuellement.

- **En régression** : cela revient à prédire la valeur moyenne de l'ensemble des modèles

$$H_T(\mathbf{x}) = \frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x}).$$

Approches naïves III

- En classification : on va effectuer un **vote de majorité**, *i.e.* l'étiquette prédite sera celle ayant obtenu le plus de voix

$$H_T(\mathbf{x}) = \left(\frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x}) \right).$$

Cela est le cas lorsque $h_t(\mathbf{x}) \in \{-1, +1\}$, mais dans le cas où $h_t(\mathbf{x}) \in \mathbb{R}$, alors on peut voir cela comme **un vote de majorité pondéré**.

Si cette approche est intéressante, elle n'est pas satisfaisante quand nos problèmes sont convexes. En effet, on risque d'apprendre plusieurs fois le même modèle.

Approches naïves IV

Basée sur les hyper-paramètres

On pourrait chercher à apprendre des modèles h_t différents en faisant varier les hyper-paramètres du modèles, *i.e.* apprendre T modèles en utilisant à chaque fois des valeurs différentes des hyper-paramètres.

- **Avantage** : on va créer de la diversité au sein des hypothèses h_t , elles vont chacune avoir des comportements différents (potentiellement)
- **Inconvénients** : le choix des hyper-paramètres est aussi important pour la performance des modèles. Un mauvais choix peut entraîner l'apprentissage d'un modèle au très faible pouvoir prédictif et cette dernière serait alors inutile.

En outre, cela remettrait en cause le processus de cross-validation.

On fait le choix de se focaliser sur un bon usage de l'ensemble d'entraînement S .

Approches naïves V

Plusieurs façons de manipuler notre jeu de données afin d'apprendre des hypothèses avec des expressivités différentes

- Créer des sous-échantillons S_t obtenus par tirage aléatoire (avec ou sans remises) à partir de l'échantillon S .
- Attribuer des poids différents aux exemples lors de l'apprentissage successif des différents modèles.

La première méthode donne naissance à une méthode de combinaisons de modèles nommée le **bagging** et la deuxième approche est une approche dite de **boosting**.

Bagging I

Illustration

Il s'agit d'une méthode qui consiste à combiner des modèles qui ont des bonnes performances sur l'ensemble d'entraînement (**biais faible**) afin de **réduire la variance** de ces derniers.

$$\mathcal{R}(h) - \mathcal{R}^* \leq \text{tr} \left\{ \inf_{g \in \mathcal{H}} \mathcal{R}(g) - \mathcal{R}^* \right\} + \mathfrak{R}(\mathcal{H}).$$

On va donc se focaliser sur le terme de variance, *i.e.* sur la sensibilité du modèle à des variations de données pendant la phase d'apprentissage.

On va prendre un exemple basé sur la régression pour illustrer cela

Bagging II

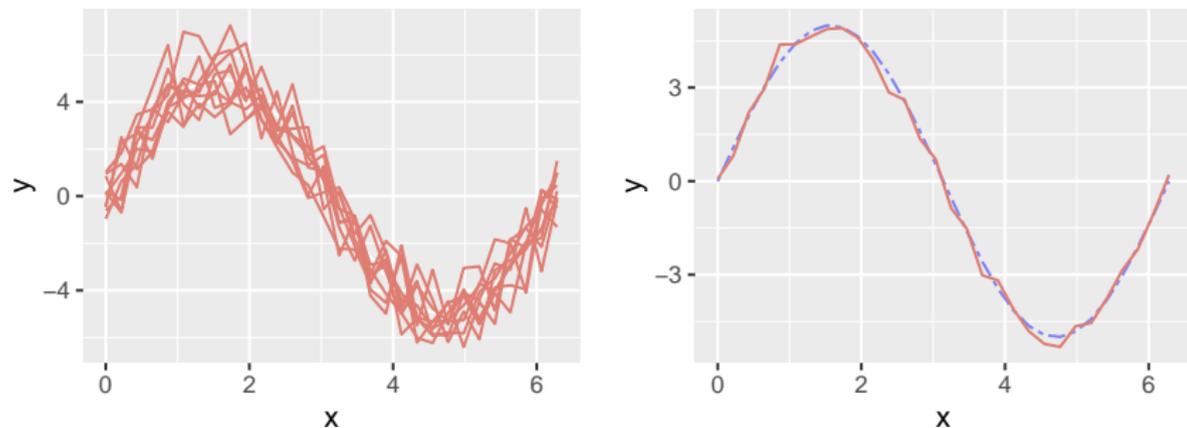


Figure – Illustration du phénomène de réduction de variance. Gauche : 10 modèles appris sur des échantillons de 30 données issues d'une distribution. Droite : combinaison (moyenne) des 10 modèles précédents (en rouge) ainsi que la vraie fonction de régression (en bleu)

Bagging III

- Chaque modèle, individuellement, présente une très grande variabilité bien que le biais soit faible.
- En combinant ces modèles, on remarque que l'on approche grandement la vraie fonction qui a permis de générer les données.

Cela illustre bien le phénomène de réduction de la variance, tout en conservant de bonnes performances.

Mais comment faire cela en pratique ? On va utiliser une méthode de Bootstrap (c'est d'ailleurs de là que vient le nom de Bagging).

Bagging IV

Bootstrap

Soit un échantillon S de taille m , *i.e.* un échantillon d'apprentissage avec m exemples.

Pour créer un échantillon bootstrap S_t de la même taille m , nous devons effectuer m *tirages aléatoires indépendants et avec remise* dans l'ensemble S .

Ainsi chaque exemple a la même probabilité d'être tiré au cours du processus. En outre, un même exemple peut apparaître plusieurs fois dans un échantillon bootstrap S_t , ce qui va contraindre l'algorithme appris à se focaliser sur les exemples qui apparaissent plusieurs fois.

C'est une méthode intéressante pour créer de la diversité dans les modèles appris car chaque modèle se base sur des données différentes.

Bagging V

Exercices

Soit $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, un ensemble d'apprentissage avec m exemples et considérons la méthode bootstrap présentée précédemment :

1. Quelle est la probabilité de tirer un exemple \mathbf{x}_i au cours de la création d'un échantillon S_t ?
2. Quelle est la probabilité qu'un exemple ne soit jamais tiré au cours de m tirages ?
3. Approximer cette valeur lorsque m tend vers l'infini.
4. D'après ce qui précède, quelle est la proportion d'exemples différents qui se trouvent dans un échantillon bootstrap ?
5. Combien d'échantillons bootstrap différents est-ce que l'on peut créer (sans tenir compte de l'ordre des éléments) ?

Bagging VI

Algorithme Bagging

Input: Notre jeu de données $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ ainsi que le nombre de modèles que l'on souhaite apprendre T

Output: Un modèle $H_T = \sum_{t=0}^T \alpha_t h_t$

begin

for $t = 1, \dots, T$ **do**

 créer un échantillon bootstrap S_t de taille m en utilisant S .

 apprendre une hypothèse h_t à partir de S_t

 Poser $H_T = \sum_{t=0}^T \alpha_t h_t$

return H_T

Bagging VII

Une analyse

On va regarder pourquoi cette approche basée sur le bagging fonction bien et permet de réduire notre erreur.

Supposons que nos données soient générées par une vraie fonction inconnue r (ici de régression) que l'on cherchera à estimer, *i.e.*

$$y = r(\mathbf{x})$$

Dans un contexte de bagging, on va donc apprendre une suite d'hypothèses h_t qui vont chercher à approcher la fonction r , *i.e.*

$$h_t(\mathbf{x}) = r(\mathbf{x}) + \varepsilon_t(\mathbf{x}).$$

Comme nous sommes dans un contexte de régression on va donc s'intéresser à l'erreur quadratique moyenne

Bagging VIII

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[(h(\mathbf{x}) - r(\mathbf{x}))^2 \right] = \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\varepsilon(\mathbf{x})^2 \right].$$

Bagging IX

On peut ainsi déterminer l'erreur en généralisation de notre classifieur

$$H_T = \frac{1}{T} \sum_{t=1}^T h_t \text{ par}$$

$$\begin{aligned} \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[(H_T(\mathbf{x}) - r(\mathbf{x}))^2 \right] &= \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\left(\frac{1}{T} \sum_{t=1}^T h_t(\mathbf{x}) - r(\mathbf{x}) \right)^2 \right], \\ &\downarrow \text{ par définition} \\ &= \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\left(\frac{1}{T} \sum_{t=1}^T \varepsilon_t(\mathbf{x}) \right)^2 \right]. \end{aligned}$$

On va maintenant supposer que nos erreurs ε_t sont centrées et non corrélées (cette dernière hypothèse n'est très réaliste ...).

Ainsi on peut écrire

Bagging X

$$\mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[(H_T(\mathbf{x}) - r(\mathbf{x}))^2 \right] = \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\left(\frac{1}{T} \sum_{t=1}^T \varepsilon_t(\mathbf{x}) \right)^2 \right],$$

↓ on développe le carré

$$= \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[\left(\frac{1}{T} \sum_{t_1=1}^T \varepsilon_{t_1}(\mathbf{x}) \right) \left(\frac{1}{T} \sum_{t_2=1}^T \varepsilon_{t_2}(\mathbf{x}) \right) \right],$$

↓ $\mathbb{E}_{\mathbf{x} \sim \mathcal{X}} [\varepsilon_{t_1}(\mathbf{x}) \varepsilon_{t_2}(\mathbf{x})] = 0$ pour tout $t_1 \neq t_2$.

$$= \frac{1}{T^2} \sum_{t=1}^T \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} [\varepsilon_t(\mathbf{x})^2] \cdot \mathbb{E}_{\mathbf{x} \sim \mathcal{X}} \left[(H_T(\mathbf{x}) - r(\mathbf{x}))^2 \right],$$

$$= \frac{1}{T} \left(\frac{1}{T} \sum_{t=1}^T \varepsilon_t(\mathbf{x})^2 \right).$$

Bagging XI

Dit autrement, l'erreur en généralisation de notre "moyenne" de classifieurs H_T , n'est rien d'autre que l'erreur moyenne de l'ensemble des classifieurs h_t , divisée par T .

Ainsi, si notre hypothèse selon laquelle nos erreurs sont bien indépendantes était vraie, nous pourrions techniquement atteindre une erreur en généralisation proche de 0!

En revanche, nous sommes quand même sûr que l'erreur du classifieur H_T ne va jamais excéder l'erreur en moyenne de l'ensemble des classifieurs h_t .

On va maintenant regarder un algorithme basé sur le bagging, les *forêts aléatoires*.

Forêts Aléatoires I

A propos

Lorsque l'on a parlé des arbres de décisions, nous avons vu que la profondeur de l'arbre dépendait fortement des données et que plus cet arbre était profond (entre autre), plus il était sujet au sur-apprentissage.

Nos arbres de décisions présentent donc un faible **biais** mais une très forte **variance**, il apparaît donc naturel de lui appliquer une procédure de bagging, pour les combiner entre eux afin de réduire cette variance [Breiman, 2001].

On va donc chercher à créer plusieurs échantillons bootstrap S_t sur lesquelles on va apprendre nos arbres (de classification ou de régression) que l'on va ensuite agréger pour obtenir une hypothèse plus puissante.

Forêts Aléatoires II

Algorithme

Inputs : Ensemble d'entraînement $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, nombre d'arbres T , une taille m' et un nombre de variable p'

Output : Hypothèse H_T

- 1: **for** $t = 1$ à T **do**
- 2: créer un échantillon bootstrap S_t de taille $m' < m$ à partir de S .
- 3: Construire un arbre h_t où à chaque noeud, on teste uniquement p' variables pour le critère de séparation.
- 4: **end for**
- 5: poser $H_T = \frac{1}{T} \sum_{t=1}^T h_t = 0$

Forêts Aléatoires III

- un algorithme basé sur un double échantillonnage
- un échantillonnage sur les données avec la méthode de bootstrap en réduisant la taille du jeu de données initial
- un échantillonnage sur les variables à chaque noeud
- permet de gagner un diversité dans les arbres appris
- permet aussi de réduire le temps d'apprentissage d'un arbre.

Quel est l'autre intérêt de faire du bootstrap avec $m < m'$ selon vous ? Si on se replace dans un contexte de cross-validation ?

Forêts Aléatoires IV

Les exemples non utilisés pour la construction de l'arbre serviront pour la validation ! Lorsque l'on apprend une forêt aléatoires, les exemples qui ne sont pas utilisés pour apprendre un arbre sont utilisées pour évaluer le modèle.

Vous pouvez regarder ce que l'on appelle le *Our of Bag Error* pour plus d'informations.

Sur plan pratique, les arbres (donc les forêts) ont l'avantage de pouvoir s'apprendre rapidement et son donc adaptées à de très grands jeux de données. Ils sont également facile à exploiter et très utilisés lorsque l'on cherche à obtenir des modèles interprétables (explainable AI).

Forêts Aléatoires V

Pour finir

- Le bagging est une méthode reposant sur l'échantillonnage, éventuellement à double niveau : données et variables.
- Il permet de combiner des modèles appris avec des données et des informations différentes pour obtenir un modèle combiné plus performant
- Il préserve le biais des sous-modèles mais permet de réduire la variance du modèle combiné.

Le bagging s'applique donc aux modèles "puissants" ou "expressifs", mais comment faire pour combiner des modèles qui ne sont pas du tout performant à la base, comme de simples modèles linéaires ?

C'est ce que nous allons voir avec le boosting.

Boosting I

Définition 2.1: Strong Learnability [Mohri et al., 2012]

A concept class \mathcal{C} is said to be (strongly) PAC learnable if there exists an algorithm \mathcal{A} and polynomial function $poly$ such that for any $\varepsilon > 0$ and $\delta > 0$, for all distributions \mathcal{X} and for any target concept $c \in \mathcal{C}$, the following holds for any sample size $m \geq poly(1/\varepsilon, 1/\delta, d, size(c))$:

$$\mathbb{P}_{S \sim \mathcal{D}^m} [\mathcal{R}(h_S) \leq \varepsilon] \geq 1 - \delta,$$

where h_S is the hypothesis returned by \mathcal{A} when trained on S . If \mathcal{A} further runs in $poly(1/\varepsilon, 1/\delta, d, size(c))$, then \mathcal{C} is said to be efficiently PAC-learnable.

Boosting II

Définition 2.2: Weak Learnability [Mohri et al., 2012]

A concept class \mathcal{C} is said to be weakly PAC learnable if there exists an algorithm \mathcal{A} , $\gamma > 0$ and polynomial function $poly$ such that for any $\delta > 0$, for all distributions \mathcal{X} and for any target concept $c \in \mathcal{C}$, the following holds for any sample size $m \geq poly(1/\delta, d, size(c))$:

$$\mathbb{P}_{S \sim \mathcal{D}^m} \left[\mathcal{R}(h_S) \leq \frac{1}{2} - \gamma \right] \geq 1 - \delta,$$

where h_S is the hypothesis returned by \mathcal{A} when trained on S . when such an algorithm exists, it is called a weak learning algorithm, a weak learner or a base classifier.

Boosting III

Pour résumer

- Un apprenant **fort** est un apprenant qui peut potentiellement atteindre la plus petite erreur souhaitée à condition de supposer de suffisamment d'exemples.
C'est par exemple le cas de nos arbres de décisions, de nos méthodes à noyaux ou de la plupart des réseaux de neurones que l'on connaît.
- Un apprenant **faible** voit ses performances plus limitées. On exige de lui qu'il fasse simplement un petit peu mieux que l'aléatoire dans ses prédictions, *i.e.* on exige que le modèle ait appris un minimum des données dont il dispose, sinon ce dernier ne servirait à rien.

Boosting IV

Principe du boosting

- On va apprendre une suite d'hypothèses $(h_t)_{t \in \mathbb{N}}$ comme ce que nous avons fait avec le bagging, c'est à dire sur des échantillons "différents".
- Les hypothèses apprises ne seront pas indépendantes, *i.e.* pas de possibilité d'apprentissage des modèles en parallèle. Nous devons les apprendre itérativement.
- Le but est d'obtenir un apprenant fort à partir des apprenants faibles, cela peut ne peut se faire que si le modèle h_{t+1} est capable de corriger des erreurs effectuées par le modèle h_t .
- On va faire cela en pondérant, de façon itérative le poids de chaque exemple.

On va regarder un premier algorithme de boosting que l'on appelle **Adaboost** [Freund and Schapire, 1999].

Boosting V

Principe du boosting

On dispose initialement de notre échantillon S avec nos m exemples (\mathbf{x}_i, y_i) et toutes les données ont **le même poids**, *i.e.* la même importance.

On va maintenant regarder comment une hypothèse h_{t+1} est apprise en fonction des performances du classifieur h_t .

Pour cela, plaçons nous à une étape t de notre algorithme où les exemples ont un poids égal à $w_i^{(t)}$. Une hypothèse h_t est alors apprise et nous pouvons évaluer son taux d'erreur en classification ε_t

$$\varepsilon_t = \sum_{i=1}^m w_i^{(t)} \mathbb{1}_{\{h_t(\mathbf{x}_i)y_i < 0\}}$$

A partir de cette erreur, nous allons déterminer une quantité α_t définie par :

Boosting VI

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

et qui va permettre de quantifier l'importance de l'hypothèse h_t dans la décision finale, *i.e.* on va définir un poids sur le classifieur appris. Par définition, si le classifieur appris est trop peu performant, son poids sera très faible (proche de 0) et ce poids sera d'autant plus élevé que le classifieur est bon.

Boosting VII

Le reste de la procédure consiste ensuite à trouver **une bonne repondération des exemples** de façon à ce que l'hypothèse qui sera apprise à l'itération suivante, puis se focaliser sur les erreurs commises par l'hypothèse actuelle, cela se fait par la mise à jour suivante :

$$w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t},$$

où Z_t est un facteur de normalisation permettant d'avoir une distribution sur les poids des exemples. Nous verrons plus tard que ce facteur de normalisation est donné par $Z_t = 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}$. La fonction de repondration des exemples, va augmenter le poids des exemples mal classés et diminuer celui des exemples bien classés.

Boosting VIII

Input: Echantillon d'apprentissage S de taille m ,
un nombre T de modèles

Output: Un modèle $H_T = \sum_{t=0}^T \alpha_t h_t$

begin

Distribution uniforme $w_i^{(0)} = \frac{1}{m}$

for $t = 1, \dots, T$ **do**

 Apprendre un classifieur h_t à partir d'un algorithme \mathcal{A}

 Calculer l'erreur ε_t de l'algorithme.

if $\varepsilon_t > 1/2$ **then**

 | Stop

else

 Calculer $\alpha_{(t)} = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$

$w_i^{(t)} = w_i^{(t-1)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$

Poser $H_T = \sum_{t=0}^T \alpha_t h_t$

return H_T

Boosting IX

Proposition 2.1: Borne Erreur sur Adaboost

L'erreur empirique du classifieur obtenue à l'aide de Adaboost est bornée par

$$\mathcal{R}_S(H_T) \leq \exp \left[-2 \sum_{t=1}^T \left(\frac{1}{2} - \varepsilon_t \right)^2 \right].$$

De plus, si pour tout $t \in \llbracket 1, T \rrbracket$, $\gamma \leq \left(\frac{1}{2} - \varepsilon_t \right)$, alors :

$$\mathcal{R}_S(H_T) \leq \exp(-2\gamma^2 T).$$

Boosting X

Démonstration

En utilisant le fait que la loss exponentielle est une borne supérieure de la fonction indicatrice, *i.e.*

$$\forall(\mathbf{x}, y) \mathbb{1}_{\{yh(\mathbf{x}) < 0\}} \leq \exp(-yh(\mathbf{x})).$$

On peut majorer le risque empirique $\mathcal{R}_S(H_T)$:

$$\begin{aligned} \mathcal{R}_S(H_T) &= \frac{1}{m} \sum_{i=1}^m \mathbb{1}_{\{y_i h(\mathbf{x}_i) < 0\}}, \\ &\leq \frac{1}{m} \sum_{i=1}^m \exp(-y_i H_T(\mathbf{x}_i)). \end{aligned}$$

Boosting XI

On peut exprimer cette dernière somme à l'aide de notre facteur de normalisation Z_t et de la définition des poids $w_i^{(t+1)}$. En effet, nous avons :

$$w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t},$$

↓ by recursion

$$= \frac{1}{m} \times \frac{\exp(-\sum_{s=1}^t \alpha_s h_s(x_i))}{\prod_{s=1}^t Z_s}$$

Ainsi notre risque empirique peut être borné par :

$$\mathcal{R}_S(H_T) \leq \frac{1}{m} \sum_{i=1}^m \exp(-y_i H_T(\mathbf{x}_i)),$$

Boosting XII

↓ définition de H_T et la récurrence précédente

$$\begin{aligned} &\leq \frac{1}{m} \sum_{i=1}^m \left(m \prod_{t=1}^T Z_t \right) w_i^{(T+1)}, \\ &\leq \prod_{t=1}^T Z_t. \end{aligned}$$

On se concentre maintenant sur le facteur de normalisation et on va regarder comment on peut l'écrire comme une fonction de l'erreur en classification ε_t pour tout $t \in \llbracket 1, T \rrbracket$.

$$Z_t = \sum_{i=1}^m w_i^{(t)} \exp(-\alpha_t y_i h_t(\mathbf{x}_i)),$$

Boosting XIII

$$\begin{aligned}
&= \sum_{i:y_i h_t(\mathbf{x}_i)=1} w_i^{(t)} \exp(-\alpha_t) + \sum_{i:y_i h_t(\mathbf{x}_i)=-1} w_i^{(t)} \exp(\alpha_t), \\
&= (1 - \varepsilon_t) \exp(-\alpha_t) + \varepsilon_t \exp(\alpha_t), \\
&= (1 - \varepsilon_t) \sqrt{\frac{\varepsilon_t}{1 - \varepsilon_t}} + \varepsilon_t \sqrt{\frac{1 - \varepsilon_t}{\varepsilon_t}}, \\
&= 2\sqrt{\varepsilon_t(1 - \varepsilon_t)}.
\end{aligned}$$

Comme le risque empirique $\mathcal{R}_S(H_T)$ est borné par le produit des facteurs de normalisation, nous avons :

$$\mathcal{R}_S(H_T) \leq \prod_{t=1}^T Z_t,$$

Boosting XIV

$$\leq \prod_{t=1}^T 2\sqrt{\varepsilon_t(1-\varepsilon_t)},$$

↓ utilisant $4\varepsilon_t(1-\varepsilon_t) = 1 - 4\left(\frac{1}{2} - \varepsilon_t\right)^2$

$$\leq \prod_{t=1}^T \sqrt{1 - 4\left(\frac{1}{2} - \varepsilon_t\right)^2},$$

↓ pour tout $x \in \mathbb{R}$ $1 - x \leq \exp(-x)$

$$\leq \prod_{t=1}^T \exp\left[-2\left(\frac{1}{2} - \varepsilon_t\right)^2\right],$$

↓ propriété de l'exponentielle

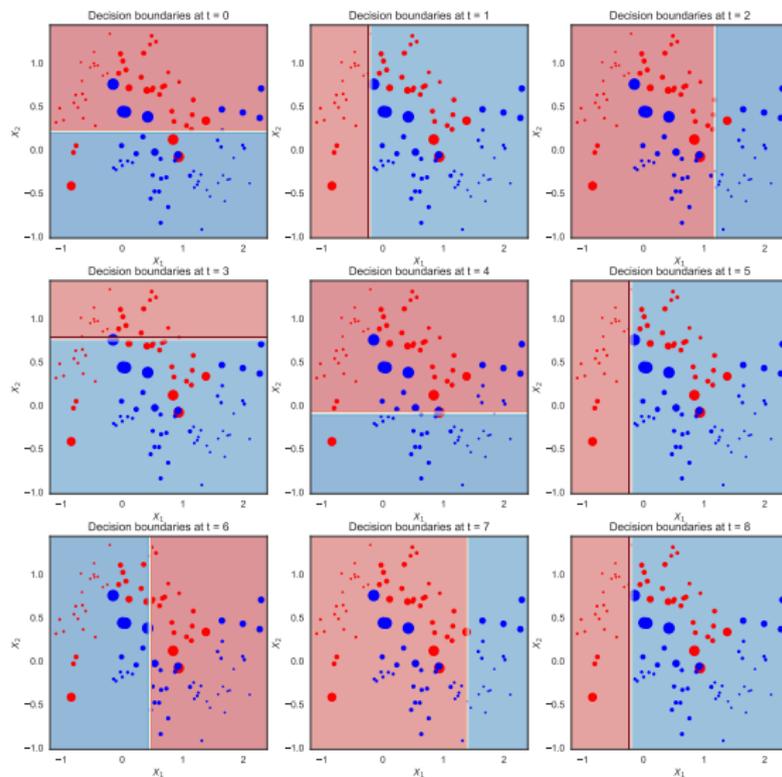
$$\leq \exp\left[-2\sum_{t=1}^T \left(\frac{1}{2} - \varepsilon_t\right)^2\right].$$

Boosting XV

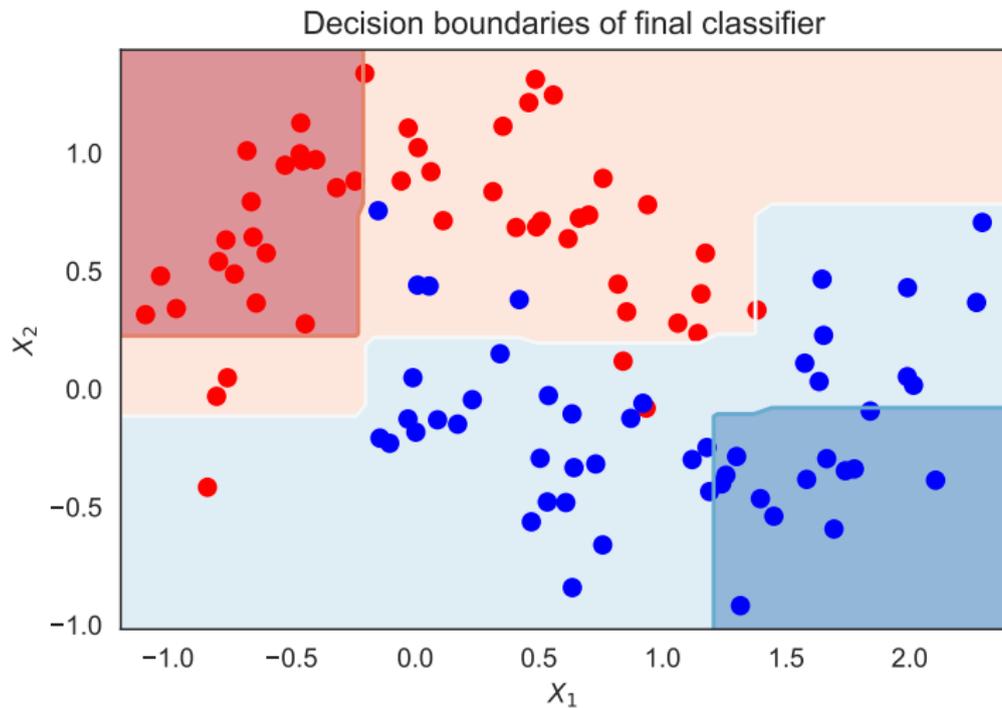
Ce qui achève la première partie de la preuve. De plus, si pour tout t nous avons $\gamma \leq \left(\frac{1}{2} - \varepsilon_t\right)$:

$$\begin{aligned}\mathcal{R}_S(H_T) &\leq \exp \left[-2 \sum_{t=1}^T \left(\frac{1}{2} - \varepsilon_t \right)^2 \right], \\ &\quad \downarrow \text{ using the assumption} \\ &\leq \exp \left[-2 \sum_{t=1}^T \gamma^2 \right], \\ &\leq \exp \left[-2T\gamma^2 \right].\end{aligned}$$

Boosting XVI



Boosting XVII



Boosting XVIII

Pour finir

Le boosting permet donc de combiner des classifieurs faibles pour en créer un plus puissant en travaillant sur la pondération des exemples.

L'algorithme Adaboost ainsi présenté a la structure que la descente de gradient par coordonnées ! Où l'idée ne serait plus d'apprendre les classifieurs, mais partant d'un classifieur, on souhaiterait trouver par un algorithme de descente de gradient, la bonne pondération à effectuer. On pourrait parler de descente de gradient dans un espace de fonctions et non plus dans l'espace des variables.

Ce lien avec une descente de gradient se retrouve dans une approche de de boosting nommé **Gradient Boosting** [Friedman, 2000].

Gradient Boosting I

Motivations

L'algorithme Adaboost présenté précédemment repose sur la loss exponentielle.

Cependant, cette loss n'est pas nécessairement la plus appropriée pour tous les contextes que l'on est amené à rencontrer mais aussi selon le contexte applicatif.

L'algorithme de **Gradient boosting** va se présenter comme un algorithme de descente de gradient dans un espace fonctionnelle, *i.e.* à chaque itération, nous allons apprendre un classifieur h_t en se basant sur les erreurs (que l'on appellera pseudo-résidus) effectuées par les précédents modèles.

$$H_t = H_{t-1} + \alpha_t h_t \quad (1)$$

où H_{t-1} est combinaison linéaire des $t - 1$ premiers modèles avec leurs poids respectifs.

Gradient Boosting II

Les apprenants faibles (ou weak learner) sont entraînés sur les pseudo-résidus r_i du modèle courant. Ces pseudo-résidus sont définis comme le gradient négatif, $-g_t$, de la fonction de loss ℓ par rapport à la prédiction courante $H_{t-1}(\mathbf{x}_i)$:

$$r_i = g_t(\mathbf{x}_i) = - \left[\frac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)} \right].$$

Une fois les résidus r_i calculés, on résout le problème d'optimisation suivant :

$$(h_t, \alpha_t) = \arg \min_{\alpha, h} \sum_{i=1}^m (r_i - \alpha h(\mathbf{x}_i))^2.$$

Gradient Boosting III

L'idée est d'apprendre un modèle, ainsi que le poids correspondant, de façon à minimiser, "combler" les erreurs effectuées par les précédentes itérations. Une fois que le modèle est appris, la mise à jour (1) est appliquée.

Gradient Boosting IV

Input: Ensemble $S = \{\mathbf{x}_i, y_i\}_{i=1}^m$, une loss ℓ , nombre d'itérations T

Output: Un modèle sign $\left(H_0(\mathbf{x}) + \sum_{t=1}^T \alpha^t h_{a^t}(\mathbf{x})\right)$

begin

Hypothèse initiale $H_0(\mathbf{x}_i) = \arg \min_{\rho \in \mathbb{R}} \sum_{i=1}^m \ell(y_i, \rho) \quad \forall i = 1, \dots, m$

for $t = 1, \dots, T$ **do**

Calculer les pseudo-résidus :

$$\tilde{y}_i = -\frac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)}, \quad \forall i = 1, \dots, m$$

Apprendre un modèle pour les fitter :

$$a^t = \arg \min_{a \in \mathbb{R}^d} \sum_{i=1}^m (\tilde{y}_i - h_a(\mathbf{x}_i))^2$$

Apprendre le poids du classifieur :

$$\alpha_t = \arg \min_{\alpha \in \mathbb{R}^+} \sum_{i=1}^m \ell(y_i, H_{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$$

Mise à jour $H_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i) + \alpha_t h_{a^t}(\mathbf{x}_i)$

return H^t

XGBoost I

On va finir ces présentations sur le boosting en retournant sur nos arbres de décisions et présentant **XGBoost** [Chen and Guestrin, 2016] qui est un algorithme de *Gradient Tree Boosting*, *i.e.* qui faire du boosting sur des arbres.

- Utilise des arbres de faible profondeur
- Permet l'optimisation de n'importe quelle loss
- Se base sur une approximation d'ordre 2
- Apprentissage rapide, même sur des données volumineuses

Regardons son fonctionnement.

XGBoost II

On considère une loss ℓ (on ne fait d'hypothèse particulière dessus) et le problème d'optimisation suivant :

$$\min \sum_{i=1}^m \ell(y_i, \hat{y}_i) + \beta \mathcal{L} + \frac{\lambda}{2} \sum_{j=1}^{\mathcal{L}} (f_j^{(t)})^2. \quad (2)$$

où $\beta \mathcal{L}$ et $\frac{\lambda}{2} \sum_{j=1}^{\mathcal{L}} (h_t^{(j)})^2$ sont deux termes de régularisations qui contrôlent le nombre mais aussi le poids des feuilles $f_t^{(j)}$ pour l'arbre appris à l'itération t .

Les modèles sont appris de façon additives, notons $\hat{y}^{(t-1)}$, la valeur prédite par les $t - 1$ premières fonctions h_k , i.e. $\hat{y}_i^{(t-1)} = \sum_{k=1}^{t-1} h_k(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i)$.

XGBoost III

Regardons maintenant comment est appris le modèle suivant.

Pour cela, commençons par réécrire la quantité (2) à minimiser comme suit :

$$\sum_{i=1}^m \ell(y_i, \hat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)) + \beta \mathcal{L} + \frac{\lambda}{2} \sum_{j=1}^{\mathcal{L}} (h_t^{(j)})^2. \quad (3)$$

En pratique, [Chen and Guestrin, 2016] considère simplement une approximation d'ordre de deux la fonction qu'ils souhaitent optimiser. Cette approximation d'ordre 2 est déterminée en fonction de la valeur prédite à l'itération précédente, *i.e.* $\hat{y}_i^{(t-1)}$. On notera respectivement g and f les dérivées première et seconde de notre fonction de loss ℓ par rapport à $\hat{y}^{(t-1)}$.

XGBoost IV

On peut réécrire l'équation (3) comme suit :

$$\sum_{i=1}^m \left[\ell(y_i, \hat{y}_i^{(t-1)}) + h_t(\mathbf{x}_i)g(\mathbf{x}_i) + \frac{1}{2}h_t^2(\mathbf{x}_i)f(\mathbf{x}_i) \right] + \beta\mathcal{L} + \frac{\lambda}{2} \sum_{j=1}^{\mathcal{L}} (f_t^{(j)})^2. \quad (4)$$

Rappelons que l'on souhaite apprendre la fonction $\mathbf{f}_t = (h_t^{(j)})_{j=1, \dots, \mathcal{L}}$.
 Considérons alors une feuille j et notons I_j l'ensemble des indices i tels que \mathbf{x}_i appartienne à la feuille l_j . Ainsi, en utilisant (4), la fonction $h_t^{(j)}$ doit minimiser la quantité V_j pour un certain indice j :

$$V_j = \sum_{i \in I_j} \left[g(\mathbf{x}_i)h_t^{(j)}(\mathbf{x}_i) + \frac{1}{2} (\lambda + f(\mathbf{x}_i)) (h_t^{(j)}(\mathbf{x}_i))^2 \right]. \quad (5)$$

XGBoost V

Cette fonction est convexe et admet donc un minimum, donné en résolvant l'*Equation d'Euler*, i.e. la fonction $f^{(t)}$ pour laquelle le gradient s'annule. Cette solution est donnée par :

$$h_t^{(j)} = -\frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}. \quad (6)$$

Focalisons nous maintenant sur la façon dont on va effectuer un split, une séparation à un noeud donné. Maintenant que l'on a trouvé le poids optimal à appliquer à chaque feuille (6), on peut calculer la valeur optimal V_j^* de la loss en utilisant (5). Ce qui nous donne

XGBoost VI

$$\begin{aligned}
 V_j^* &= \sum_{i \in I_j} \left[\underbrace{-g(\mathbf{x}_i) \frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}}_{\text{blue}} \right. \\
 &\quad \left. + \underbrace{\frac{1}{2} (\lambda + f(\mathbf{x}_i)) \left(-\frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} \right)^2}_{\text{red}} \right], \\
 &= -\frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} + \frac{1}{2} \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}, \\
 V_j^* &= -\frac{1}{2} \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}.
 \end{aligned}$$

XGBoost VII

Cette formule est utilisée pour mesurer la pureté d'une feuille. Elle peut se voir comme une généralisation de l'impureté de Gini mais pour n'importe quelle loss ℓ . En utilisant cette nouvelle mesure, ils définissent ensuite leur critère de séparation, *i.e.* le gain associé à chaque séparation :

$$\frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g(\mathbf{x}_i)\right)^2}{\sum_{i \in I_L} f(\mathbf{x}_i) + \lambda} + \frac{\left(\sum_{i \in I_R} g(\mathbf{x}_i)\right)^2}{\sum_{i \in I_R} f(\mathbf{x}_i) + \lambda} - \frac{\left(\sum_{i \in I} g(\mathbf{x}_i)\right)^2}{\sum_{i \in I} f(\mathbf{x}_i) + \lambda} \right] - \beta,$$

où $I = I_L \cup I_R$ pour un arbre binaire and le paramètre β est utilisé pour contrôler le nombre de feuilles

XGBoost VIII

Exercices

Considérons notre algorithme de Gradient Tree Boosting que nous venons de présenter.

Déterminer le poids des feuilles de l'algorithme pour les loss suivantes :

1. la loss quadratique :

$$\ell(y, \hat{y}^{(t-1)}) = \frac{1}{2}(y - \hat{y}^{(t-1)})^2$$

2. la loss logistique :

$$\ell(y, \hat{y}^{(t-1)}) = -y \ln\left(p(\hat{y}^{(t-1)})\right) - (1 - y) \ln\left(1 - p(\hat{y}^{(t-1)})\right),$$

où p désigne la fonction logistique.

Références I

-  Breiman, L. (2001).
Random forests.
Machine Learning, 45(1) :5–32.
-  Chen, T. and Guestrin, C. (2016).
Xgboost : A scalable tree boosting system.
In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794.
ACM.
-  Freund, Y. and Schapire, R. E. (1999).
A short introduction to boosting.
In *In Proceedings of the Sixteenth IJCAI*, pages 1401–1406. Morgan Kaufmann.

Références II



Friedman, J. H. (2000).

Greedy function approximation : A gradient boosting machine.
Annals of Statistics, 29 :1189–1232.



Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012).

Foundations of Machine Learning.
The MIT Press.