

# Agrégation de modèles

modèles ensemblistes

[masedki.github.io](https://masedki.github.io)

Université Paris-Saclay & CESP Inserm-1018



ÉCOLE D'ÉTÉ  
DE SANTÉ PUBLIQUE  
ET D'ÉPIDÉMIOLOGIE  
DE BICÊTRE





# Avantages et inconvénients des arbres

- ▲ Les arbres sont faciles à expliquer à n'importe qui. Ils sont plus faciles à expliquer que les modèles linéaires
- ▲ Les arbres peuvent être représentés graphiquement, et sont interprétables même par des non-experts
- ▲ Ils peuvent gérer des variables explicatives catégorielles sans introduire des variables binaires
- ▼ Malheureusement, ils n'ont pas la même qualité prédictives que les autres approches d'apprentissage.

Cependant, en agrégeant plusieurs arbres de décision, les performances prédictives s'améliorent substantiellement.

# Agrégation par Bagging

- L'agrégation bootstrap ou bagging est méthode de réduction de la variance en apprentissage statistique. Elle est particulièrement utile sur les arbres de décision.
- Rappelons que, sur un ensemble de  $n$  observations indépendantes  $Z_1, \dots, Z_n$ , chacune de variance  $\sigma^2$ , la variance de la moyenne  $\bar{Z}$  est  $\sigma^2/n$ .
- En pratique, il n'est pas possible de moyenner des arbres de décision construits sur de multiples ensembles d'entraînement (pas assez de données observées)

# Bagging pour la régression

- Au lieu de cela, on peut faire du bootstrap en ré-échantillonnant plusieurs fois les données d'apprentissage.
- Alors, à partir de  $B$  échantillons bootstrap, on entraîne une méthode d'apprentissage pour ajuster  $B$  fonctions de régressions, notées  $\hat{f}^{*b}(x)$ ,  $b = 1, \dots, B$
- La fonction de régression *bagguée* est alors

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$$

# Bagging pour la classification

- Sur un problème de classification,  $\hat{f}^{*b}(x)$  renvoie une classe possible pour chaque échantillon bootstrap  $b$ .
- La décision finale  $\hat{f}_{\text{bag}}(x)$  se prend par un vote à la majorité simple parmi les  $B$  prédictions des règles de classification bootstrap.

# Intuitivement

- Cela fonctionne mieux pour les méthodes d'apprentissage à faible biais et à forte variance
- C'est le cas des arbres de décision, en particulier les arbres profonds.
- Sur des gros jeux de données d'entraînement, faire parfois du sous-échantillonnage bootstrap.

## Erreur *Out-Of-Bag* (OOB)

- Il y a une façon simple d'estimer l'erreur de test quand on fait du bagging.
- La clé du bagging est l'entraînement de nombreux  $\hat{f}(x)$  sur des échantillons bootstraps. On peut donc utiliser les observations hors du  $b^{\text{ième}}$  bootstrap pour évaluer chaque  $\hat{f}^{*b}(x)$ .
- Ce qui donne l'algorithme ci-dessous.
  - Pour chaque observation  $(x_i, y_i)$ , calculer  $\hat{y}_i^{\text{OOB}}$  la prédiction en n'utilisant que les estimateurs  $\hat{f}^{*b}(x)$  qui n'ont pas vu cette observation dans leur entraînement
  - Évaluer l'erreur entre  $\hat{y}_i^{\text{OOB}}$  et les  $y_i$  (erreur quadratique moyenne ou taux de mauvaise classification)



## Erreur *Out-Of-Bag* pour l'estimation de l'erreur de test

- La probabilité qu'une observation  $i$  ne fasse pas partie d'un échantillon bootstrap est de  $(1 - \frac{1}{n})^n \approx \frac{1}{e}$ .
- Le nombre d'observations qui ne font pas partie d'un tirage bootstrap est  $n(1 - \frac{1}{n})^n \approx \frac{n}{e}$ . Ces observations sont dites *out-of-bag*.
- Sur  $B$  tirages bootstrap, il y a environ  $\frac{B}{e}$  échantillon qui ne contiennent pas l'observation  $i$ .
- Les arbres de décisions ajustés sur ces échantillons servent à prédire la réponse de l'observation  $i$ . Il y a environ  $\frac{B}{e}$  prédictions.
- On fait la moyenne des ces prédictions pour la régression ou prendre le vote à majorité simple pour la classification pour calculer la prédiction *bagguée* de l'observation  $i$  qu'on notera  $\hat{f}^*(x_i)$ .

# Estimation de l'erreur de test par OOB

- L'erreur quadratique moyenne ( $\propto$  moindres carrés) OOB pour la régression

$$\frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}^*(x_i))^2.$$

- L'erreur de classification OOB

$$\frac{1}{n} \sum_{i=1}^n \mathbf{1}\{y_i \neq \hat{f}^*(x_i)\}.$$

- L'erreur OOB est l'équivalente d'une erreur de test.
- Lorsque  $B$  est grand, on peut montrer que l'erreur OOB est équivalente à l'erreur calculée par validation-croisée one-leave-one-out.

# Mesurer l'importance des variables

- Le bagging améliore la précision d'un modèle au détriment de son interprétation
- On peut obtenir un résumé général de l'importance d'une variable à l'aide des moindres carrés pour le bagging d'arbres de régression et l'indice de Gini pour le bagging d'arbres de classification.
- Pour chaque arbre de régression (ou classification) ajusté sur un échantillon bootstrap, on calcule le nombre de fois où les moindres carrés (ou l'indice de Gini pour la classification) a diminué par une partition d'une variable  $j$ . On fait la moyenne de cet indicateur sur les  $B$  échantillons bootstraps.
- Une grande valeur de cet indicateur indique une importance de la variable  $j$

## Garanties théoriques : un peu de notations

- On note l'échantillon  $\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$  et on rappelle la fonction de régression

$$m^*(x) = \mathbb{E}[Y|X = x].$$

- Pour  $x \in \mathbb{R}^p$ , on considère l'erreur quadratique moyenne d'un estimateur  $\hat{m}$  et sa décomposition biais-variance

$$\mathbb{E}\left[(\hat{m}(x) - m^*(x))^2\right] = \left(\mathbb{E}(\hat{m}(x)) - m^*(x)\right)^2 + \text{Var}(\hat{m}(x)).$$

- Soit l'estimateur  $\hat{m}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^B \hat{m}_b(x)$  obtenue par l'agrégation des fonctions de régression  $\hat{m}_1, \dots, \hat{m}_B$ . Remarquons que si on suppose que les fonctions de régression  $\hat{m}_1, \dots, \hat{m}_B$  i.i.d, on a

$$\mathbb{E}[\hat{m}_{\text{bag}}(x)] = \mathbb{E}[\hat{m}_1(x)] \quad \text{et} \quad \text{Var}[\hat{m}_{\text{bag}}(x)] = \frac{1}{B} \text{Var}[\hat{m}_1(x)].$$

même biais mais la variance diminue

# Garanties théoriques : biais et variance

- Le fait de considérer des échantillons bootstrap introduit un aléa supplémentaire dans l'estimateur. Afin de prendre en compte cette nouvelle source d'aléatoire, on note  $\theta_b = \theta_b(\mathcal{D}_n)$  l'échantillon bootstrap de l'étape  $b$  et  $\hat{m}(\cdot, \theta_b)$  l'estimateur construit à l'étape  $b$ . On écrira l'estimateur final  $\hat{m}_B(x) = \frac{1}{B} \sum_{b=1}^B \hat{m}(x, \theta_b)$ .
- $\sigma^2(x) = \text{Var}(\hat{m}(x, \theta_b))$
- $\rho(x) = \text{corr}(\hat{m}(x, \theta_1), \hat{m}(x, \theta_2))$ , le coefficient de corrélation entre deux estimateurs que l'on agrège (calculés sur deux échantillons bootstrap).
- La variance  $\sigma^2(x)$  et la corrélation  $\rho(x)$  sont calculées par rapport aux lois de  $\mathcal{D}_n$  et de  $\theta$ . On suppose que les estimateurs  $\hat{m}(x, \theta_1), \dots, \hat{m}(x, \theta_B)$  sont identiquement distribués.
- **Proposition On a :**

$$\text{Var}_B(\hat{m}_B(x)) = \rho(x)\sigma^2(x) + \frac{1 - \rho(x)}{B}\sigma^2(x).$$

*Par conséquent*

$$\text{Var}[\hat{m}(x)] = \rho(x)\sigma^2(x).$$

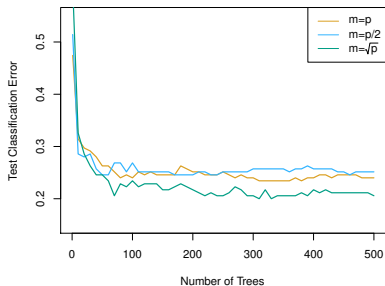
# Forêts aléatoires très proche du bagging

- C'est la même idée que le bagging à l'exception ...
- À chaque partition, on ne considère que  $m$  variables explicatives au hasard parmi les  $p$  variables explicatives du problème.
- Souvent  $m \approx \sqrt{p}$ .

# Forêts aléatoires

- À chaque pas, la partition est contrainte sur un petit nombre  $m$  de variables explicatives choisies au hasard.
- Permet d'avoir des arbres différents.
- Deux arbres similaires sont hautement corrélés, la moyenne d'arbres hautement corrélés ne peut produire une réduction importante de la variance. Penser au cas extrême où tous les arbres sont les mêmes.
- La moyenne d'arbres non-corrélés ou faiblement corrélés permet une réduction importante de la variance.
- Une forêt aléatoire produit des arbres moins corrélés.
- Une forêt aléatoire est équivalente à un bagging si  $m = p$ .

# Illustration : données d'expression de gènes



- Résultats de forêts aléatoires pour prédire les 15 classes à partir du niveau d'expression de 500 gènes
- L'erreur de test (évaluée par OOB) dépend du nombre d'arbres. Les différentes couleurs correspondent à différentes valeurs de  $m$ .
- Les forêts aléatoires améliorent significativement le taux d'erreur de CART (environ 45.7%)



# Plan

- De quoi s'agit-il ?
- Un peu d'histoire
- Gradient boosting pour la régression

De quoi s'agit-il ?

**Gradient Boosting = Gradient Descent + Boosting**

# De quoi s'agit-il ?

- **Premier** algorithme de “boosting” {Freund and Schapire, 1997}.
- Construire une famille de **règles** qui sont ensuite agrégées.
- Processus **récurif** : la règle construite à l'étape  $k$  dépend de celle construite à l'étape  $k - 1$

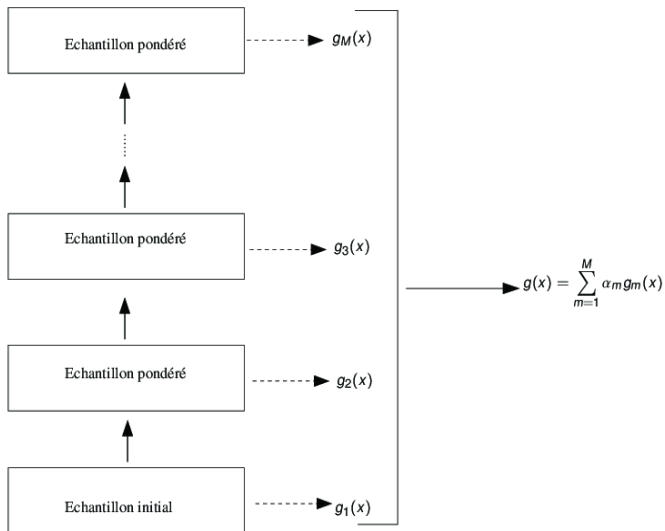
# Un peu d'histoire

- Invention Adaboost, premier algorithme de boosting [Freund et al., 1996, Freund and Schapire, 1997]
- Formulation de l'algorithme Adaboost comme une descente du gradient avec une fonction de perte particulière [Breiman et al., 1998, Breiman, 1999]
- Généralisation de l'algorithme Adaboost au Gradient Boosting pour l'adapter à différentes fonctions de perte [Friedman et al., 2000, Friedman, 2001]

# Principe

- Le **bagging** propose d'agréger des modèles à forte variances.
- Le **boosting** est proposé à l'origine pour des problèmes de classification ensuite adapté à la régression.
- Le **boosting** combine séquentiellement des règles de classification dites **faibles** pour produire une règle de classification précise.
- Nous allons introduire l'algorithme de boosting le plus connu appelé **AdaBoost.M1** introduit par [Freund and Schapire, 1997](#).
- On s'intéresse au problème de classification binaire où  $Y \in \{-1, 1\}$ . Pour un vecteur de variables explicatives,  $g(X)$  est une règle de classification qui prédit une des modalités  $\{-1, 1\}$ .

# Schéma



# Notion de règle faible

- Le terme **boosting** s'applique à des méthodes générales permettant de produire des décisions précises à partir de **règles faibles**.

**Définition :** On appelle *règle de classification faible* une règle légèrement meilleure que le hasard:

$$g \text{ faible si } \exists \gamma > 0 \text{ tel que } \mathbb{P}(g(X) \neq Y) = \frac{1}{2} - \gamma.$$

- Exemple :** arbre à 2 feuilles.

# Schéma ou idée

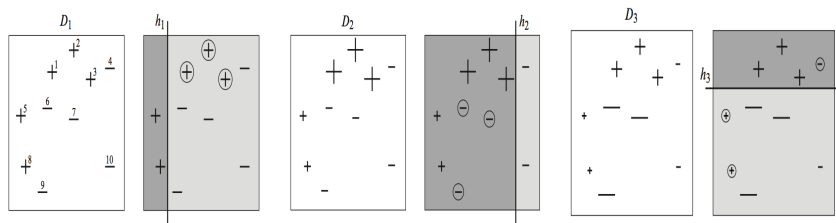


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]



# Algorithme dit Adaboost.M1

**Input :** - Une observation  $x$  à prédire et l'échantillon  $d_n = (x_1, y_1), \dots, (x_n, y_n)$  - Une règle de classification faible et  $M$  le nombre d'itérations

**Algorithm of [Freund and Schapire 1997]:**

- Initialiser les poids  $w_i = \frac{1}{n}, i = 1, \dots, n$
- **Pour**  $m = 1$  à  $M$ :
  - Ajuster la règle faible sur l'échantillon  $d_n$  pondéré par les poids  $w_1, \dots, w_n$ , on note  $g_m(x)$  l'estimateur issu de cet ajustement
  - Calcul du taux d'erreur :

$$e_m = \frac{\sum_{i=1}^n w_i \mathbf{1}_{y_i \neq g_m(x_i)}}{\sum_{i=1}^n w_i}.$$

- Calcul de :  $\alpha_m = \log\left(\frac{1-e_m}{e_m}\right)$
- Réajuster les poids :

$$w_i = w_i \exp\left(\alpha_m \mathbf{1}_{y_i \neq g_m(x_i)}\right), \quad i = 1, \dots, n.$$

**Output:**

$$\widehat{g}_M(x) = \sum_{m=1}^M \alpha_m g_m(x).$$

# Schéma ou idée

$$\hat{H}_3(x) = \sum_{m=1}^3 \alpha_m h_m(x)$$

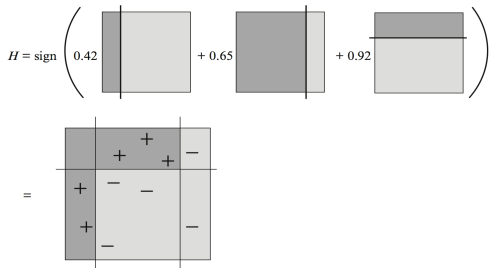


Figure: AdaBoost. Source: Figure 1.2 of [Schapire and Freund, 2012]

# Commentaires

- L'étape 1. nécessite que la règle faible puisse prendre en compte des **poids**. Lorsque ce n'est pas le cas, la règle peut être ajustée sur un **sous-échantillon de  $d_n$**  dans lequel les observations sont tirées avec remise selon les poids  $w_1, \dots, w_n$ .
- Les poids  $w_1, \dots, w_n$  sont mis à jour à chaque itération : si le  $j^{\text{ième}}$  individu est **bien classé** son poids est **inchangé**, sinon il est **augmenté**.
- Le poids  $\alpha_m$  de la règle  $g_m$  **augmente avec la performance de  $g_m$**  mesurée sur  $d_n$  :  $\alpha_m$  augmente lorsque  $e_m$  diminue (il faut néanmoins que  $g_m$  ne soit **pas trop faible** : si  $e_m > 0.5$  alors  $\alpha_m < 0$  !!!).

## Quelques garanties théoriques : contrôle de l'erreur empirique

- $e_m$  désigne le **taux d'erreur calculé sur l'échantillon** de la règle  $g_m$ :

$$e_m = \frac{\sum_{i=1}^n w_i \mathbf{1}_{y_i \neq g_m(x_i)}}{\sum_{i=1}^n w_i}.$$

- $\gamma_m$  désigne le **gain** de la règle  $g_m$  par rapport à une règle **pûrement aléatoire**

$$e_m = \frac{1}{2} - \gamma_m.$$

**Propriété: [Freund and Schapire, 1999]**

$$\mathcal{R}_n(\hat{g}_M) \leq \exp\left(-2 \sum_{m=1}^M \gamma_m^2\right).$$

**Conséquence :**

L'erreur empirique (calculée sur les données) **tend vers 0** lorsque le nombre d'itérations augmente.

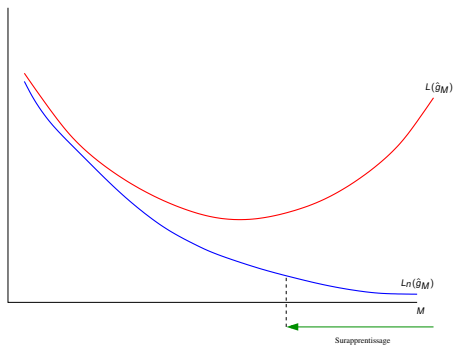
# Contrôle de l'erreur de généralisation

## Propriétés [Freund and Schapire, 1999]

$$\mathcal{R}(\hat{g}_M) \leq \mathcal{R}_n(\hat{g}_M) + \mathcal{O}\left(\sqrt{\frac{MV}{n}}\right)$$

- Le compromis **biais/variance** ou erreur **approximation/estimation** est régulé par le nombre d'itérations  $M$  :
  - $M$  petit  $\rightarrow$  premier terme (approximation) domine
  - $M$  grand  $\rightarrow$  second terme (estimation) domine
- Lorsque  $M$  est (trop) grand, Adaboost aura tendance à **sur-ajuster** l'échantillon d'apprentissage (**sur-ajustement** ou **overfitting**).

# Surapprentissage



**Conséquence :** Il est important de bien choisir  $M$ .

# Généraliser cette idée à la régression

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

$$Y_i = F^*(X_i) + \varepsilon_i, \quad 1 \leq i \leq n,$$

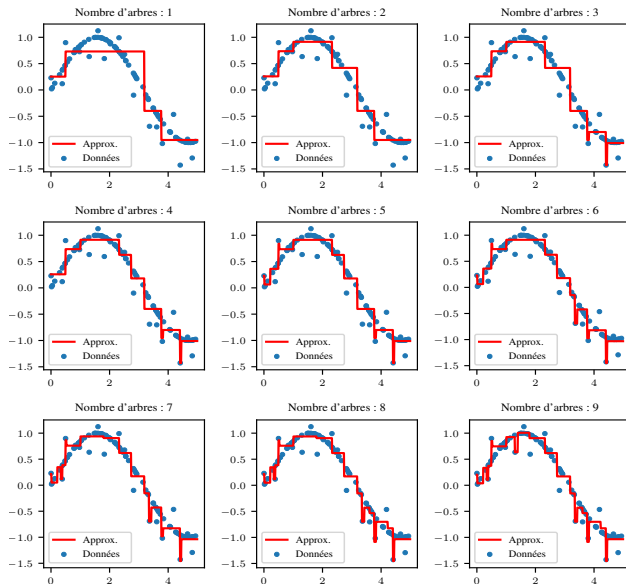
avec  $F^* : \mathbb{R}^d \rightarrow \mathbb{R}$  et les  $\varepsilon_i$  i.i.d. tels que  $\mathbb{E}[\varepsilon_i | X_i] = 0$ .

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

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

où  $h \in \mathcal{H}$  et  $\mathcal{L}(y, h) = \frac{1}{2}(y - h)^2$  est la perte quadratique. Notons  $\tilde{Y}_{0,i} = Y_i - h_0(X_i)$  l'ensemble des résidus. On peut noter  $Y_i = h_0(X_i) + \tilde{Y}_{0,i}$ . Le Boosting améliore les performances de  $h_0$  en lui ajoutant un arbre de décision de même profondeur  $h_1$  qui estime  $\tilde{Y}_{0,i}$ . En pondérant les arbres  $h_0$  et  $h_1$  par des coefficients réels  $\alpha_0$  et  $\alpha_1$  qu'on détaillera. On obtient un nouvel estimateur  $F_1(X_i) = \alpha_0 h_0(X_i) + \alpha_1 h_1(X_i)$ . On peut alors définir les nouveaux résidus  $\tilde{Y}_{1,i} = Y_i - F_1(X_i)$  et répéter le processus.

# Généralisation à la régression





# Est une descente du gradient ?

- À l'itération  $M$ , nous disposons d'un estimateur  $F_M(X) = \sum_{m=0}^M \alpha_m h_m(X)$  où  $F_0 = h_0$ . On peut voir  $F_M(\mathbb{X}) = (F_M(X_1), \dots, F_M(X_n))$  dont la perte peut s'écrire

$$\mathcal{L}(Y_1, \dots, Y_n, F_M(\mathbb{X})) = \sum_{i=1}^n \mathcal{L}(Y_i, F_M(X_i)).$$

- Leo Breiman avait remarqué qu'on pouvait interpréter le boosting comme une méthode de descente. En effet dans cet exemple

$$\tilde{Y}_{m,i} = - \left[ \frac{\partial \mathcal{L}(Y_i, F_m(X_i))}{\partial F_m(X_i)} \right]$$

est la direction de descente maximale en la coordonnée  $i$  du vecteur  $F_m(\mathbb{X})$ .

- $h_m(X_i)$  est une approximation de cette direction de descente maximale et  $\alpha_m$  peut être interprété comme le pas de l'algorithme à l'étape  $m$ .
- $F_m(\mathbb{X}) = F_{m-1}(\mathbb{X}) + \alpha_m h_m(\mathbb{X})$  avec  $h_m(\mathbb{X}) = (h_m(X_1), \dots, h_m(X_n))$ .
- Cette remarque permet de définir un cadre général pour le Boosting et de l'appliquer à des fonctions de pertes diverses.

# Oui, c'est une descente du gradient

- On peut voir une itération comme suit

$$F_{m+1} = F_m - \beta \nabla \mathcal{L}(F) \Big|_{F=F_m}$$

où  $\beta > 0$  est un pas d'apprentissage strictement positif.

- Cela revient à écrire

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

où  $h_m$  est une règle faible (weak learner). Cela revient à trouver une fonction  $h \in \mathcal{H}$  la plus proche possible de l'opposé du gradient  $\nabla \mathcal{L}(F)$  par rapport aux données d'entraînement  $\mathcal{D}_n$ .

- Comme  $\nabla \mathcal{L}(F)$  est un vecteur, on peut chercher  $h$  telle que

$$\sum_{i=1}^n \left( -\frac{\partial \mathcal{L}(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)} - h(x_i) \right)^2$$

soit minimal.

# Boosting par descente du gradient

## Entrées :

- $\mathcal{D}_n = (x_1, y_1), \dots, (x_n, y_n)$  l'échantillon,  $\beta$  un paramètre de régularisation tel que  $\beta > 0$
- $M$  le nombre d'itérations.

1. Initialisation :  $F_0(\cdot) = \operatorname{argmin}_c \frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, c)$

2. **Pour**  $m = 1$  à  $M$  :

- Calculer l'opposé du gradient  $-\frac{\partial}{\partial F(x_i)} \mathcal{L}(y_i, F_m(x_i))$  et l'évaluer aux points  $F_{m-1}(x_i)$  :

$$\tilde{Y}_{m,i} = -\frac{\partial}{\partial F(x_i)} \mathcal{L}(y_i, F(x_i)) \Bigg|_{F(x_i)=F_{m-1}(x_i)}, \quad i = 1, \dots, n.$$

- Ajuster la règle faible sur l'échantillon  $(x_1, \tilde{Y}_{m,1}), \dots, (x_n, \tilde{Y}_{m,n})$ , on note  $h_m$  la règle ainsi définie.
- Mise à jour :  $F_m(x) = F_{m-1}(x) + \beta h_m(x)$ .

3. **Sortie** : La règle  $F_M(x)$ .

# Commentaires

- La sortie  $F_M(x)$  est un **réel**. Si on cherche à prédire le label de  $x$ , on pourra utiliser la règle  $\hat{y} = \text{signe}(F_M(x))$ .
- Pour le choix  $\beta = 1$  et  $\mathcal{L}(y, F(x)) = \exp(-yF(x))$ , cet **algorithme coïncide (quasiment) avec Adaboost**.
- Le **choix de  $\beta$**  est lié au choix du nombre d'itérations  $M$ . Il permet de *contrôler* la vitesse à laquelle on minimise la fonction

$$\frac{1}{n} \sum_{i=1}^n \mathcal{L}(y_i, F(x_i)).$$

$\implies$  lorsque  $\beta \nearrow M \searrow$  et réciproquement.

# Règles faibles

- Comme pour Adaboost, la règle utilisée dans l'algorithme doit être **faible** (légèrement meilleure que le hasard).
- **Booster** une règle non faible se révèle généralement **peu performant**.
- Il est recommandé d'utiliser une règle possédant un **biais élevé** est une **faible variable** (booster permet de réduire le biais, pas la variance).
- On utilise souvent des **arbres** comme règle faible. Pour posséder un biais élevé, on utilisera donc des **arbres avec peu de nœuds terminaux**.

# Paramètres de la fonction `gbm` du package `gbm`

- fonction de perte `distribution`
- nombre d'itérations qu'on a noté  $M$  `n.trees`
- nombre de noeuds terminaux des arbres plus 1 qu'on a noté  $K$  `interaction.depth`
- paramètre de régularisation  $\lambda$  `shrinkage`

# Logitboost : gradient boosting pour la classification

Si  $Y$  est à valeurs dans  $\{0, 1\}$ . La variable  $Y|X = x$  suit une loi de Bernoulli de paramètre  $p(x) = \mathbb{P}(Y = 1|X = x)$ . La vraisemblance d'une observation  $(x, y)$  s'écrit

$$p(x) = \frac{1}{1 + \exp(-x'\beta)} = \frac{\exp(x'\beta)}{1 + \exp(-x'\beta)}.$$

Souvent on estime  $\beta$  par maximum de vraisemblance.

Le modèle logitboost repose une approche similaire  $g : \mathbb{R}^p \rightarrow \mathbb{R}$ , on pose

$$p(x) = \frac{\exp(g(x))}{\exp(g(x)) + \exp(-g(x))} = \frac{1}{1 + \exp(-2g(x))},$$

ce qui donne

$$g(x) = \frac{1}{2} \log \left( \frac{p(x)}{1 - p(x)} \right).$$

Maximiser la log-vraisemblance revient à minimiser son opposé

$$-\left(y \log(p(x)) + (1 - y) \log(1 - p(x))\right) = \log(1 + \exp(-2\tilde{y}g))$$

où  $\tilde{y} = 2y - 1 \in \{-1, 1\}$ .

# Logitboost

- On applique l'algorithme de boosting par descente du gradient à la fonction de perte

$$\ell(y, g) = \log \left( 1 + \exp \left( - 2\tilde{y}g \right) \right).$$

- Après  $M$  itérations, on obtient l'estimateur  $\hat{g}_M$  de

$$g^* = \underset{g}{\operatorname{argmin}} \mathbb{E} \left[ \log \left( 1 + \exp \left( - 2\tilde{Y}g \right) \right) \right].$$

- On peut montrer que  $g^*(x) = \frac{1}{2} \log \left( \frac{p(x)}{1-p(x)} \right)$ , on déduit un estimateur  $\hat{p}_M(x)$  de  $p(x)$  en posant

$$\hat{p}_M(x) = \frac{\exp(\hat{g}_M(x))}{\exp(\hat{g}_M(x)) + \exp(-\hat{g}_M(x))} = \frac{1}{1 + \exp(-2\hat{g}_M(x))}.$$

- On obtient la règle de classification

$$\hat{y} = \begin{cases} 1 & \text{si } \hat{g}_M(x) \geq 0 \iff \hat{p}_M(x) \geq 0.5 \\ 0 & \text{si } \hat{g}_M(x) < 0 \iff \hat{p}_M(x) < 0.5. \end{cases}$$



## $L_2$ boosting : gradient boosting pour la régression

- On s'intéresse à la régression. On désigne par  $f : \mathbb{R}^p \rightarrow \mathbb{R}$  un régresseur *faible*. Plus précisément un régresseur (fortement) biaisé, par exemple :
  - un arbre de décision à deux noeuds terminaux (stumps)
  - un estimateur à noyau avec une grande fenêtre
- Le  $L_2$  boosting consiste à appliquer l'algorithme de boosting par descente du gradient avec la fonction de perte quadratique

$$\ell(y, g) = \frac{1}{2}(y - g)^2.$$

- Après  $M$  itérations, l'algorithme fournit un estimateur  $\hat{f}_M$  de

$$f^* = \operatorname{argmin}_f \mathbb{E} \left[ \frac{1}{2} (Y - f(X))^2 \right],$$

c'est-à-dire la fonction de régression  $f^*(x) = \mathbb{E}[Y|X = x]$ .

- Les variables  $U_i$  de l'étape 2.1 du boosting par descente du gradient s'écrivent  $U_i = y_i - f_{m-1}(x_i)$ . L'étape 2.2 consiste donc simplement à faire une régression sur les résidus du modèle construit à l'étape  $m - 1$ .

# Bilan

- Les algorithmes adaboost, logitboost et  $L_2$  boosting sont donc construits selon le même schéma : ils fournissent un estimateur de  $f^*$  ou  $g^*$  qui minimise la version empirique de l'espérance d'une fonction de perte  $\ell$ .
- Récapitulatif

$\ell(y, f)$ ou $\ell(y, g)$	$f^*$ ou $g^*$	Algorithme
$\exp[-2yg]$	$\frac{1}{2} \log\left(\frac{p(x)}{1-p(x)}\right)$	Adaboost
$\log\left(1 + \exp[-2\tilde{y}g]\right)$	$\frac{1}{2} \log\left(\frac{p(x)}{1-p(x)}\right)$	Logitboost
$\frac{1}{2}(y - f)^2$	$\mathbb{E}[Y X = x]$	$L_2$ boosting

# Table of Contents

1. Agrégation par la moyenne : bagging

2. Agrégation séquentielle : boosting