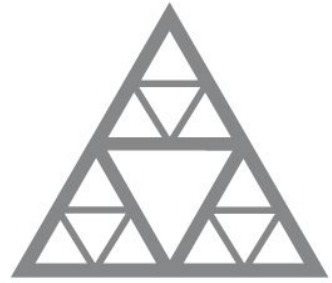
Pratique de l'apprentissage statistique

7. Boosting

V. Lefieux



École des Ponts
ParisTech

Plan

Plan

Principes

- Le boosting est une méthode d'agrégation de prédicteurs récursive : le prédicteur obtenu à l'itération b dépend de celui obtenu à l'itération $b-1$.
- Le boosting consiste à utiliser une règle faible qui apprend tout d'abord sur l'échantillon le plus simple, celui de départ, puis au fur et à mesure sur des échantillons rendus plus complexes par des pondérations adéquates, récursives.
- Contrairement au bagging, le boosting peut conduire à du sur-apprentissage : il faudra donc veiller à limiter le nombre d'itérations.
- C'est une méthode souvent utilisée en pratique au vu des bons résultats obtenus.
- Le boosting permet de traiter des problématiques de régression et de classification supervisée.

Données considérées

- On dispose d'un échantillon de (X, Y) :

$$\mathcal{D}_n = (X_i, Y_i)_{i \in \{1,\dots,n\}} \ .$$

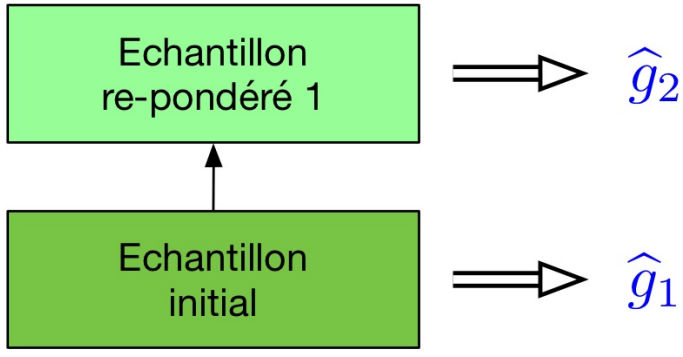
  On note :

$$d_n = (x_i, y_i)_{i \in \{1,\dots,n\}} \ .$$

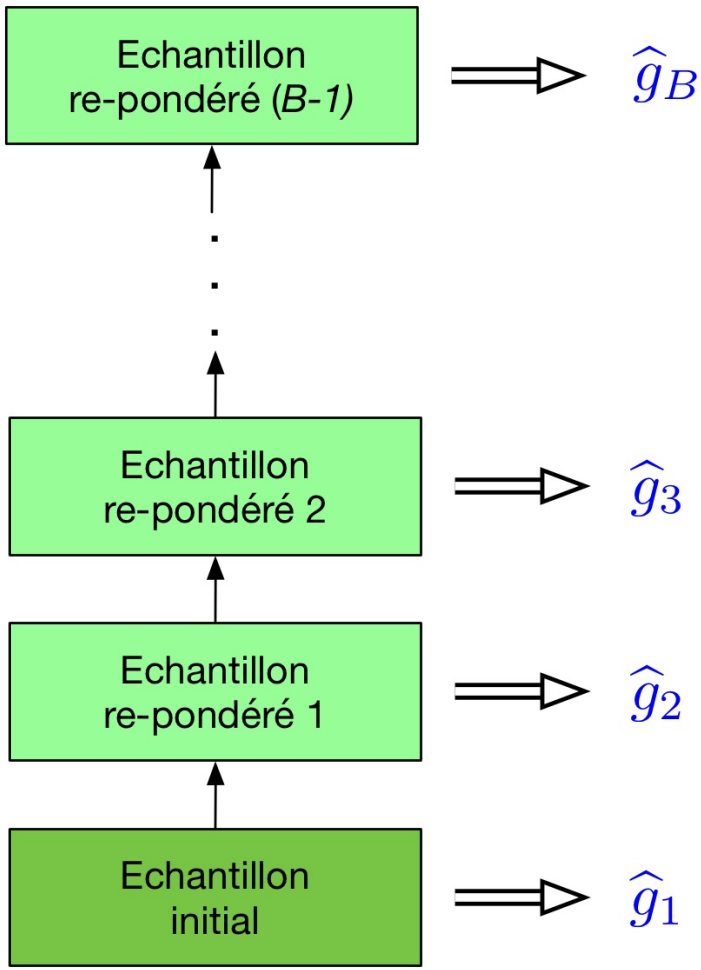
- On considère dans la suite que :
  - $X \in \mathbb{R}^p$:
    Toutes les covariables sont considérés quantitatives.
    Mais il est également possible de considérer des covariables qualitatives.
  - $Y \in \{-1, 1\}$ dans le cas de la classification supervisée :
    On se place dans le cadre d'une classification supervisée binaire
    Mais il est également possible de considérer des classifications supervisées avec K classes.
  - $Y \in \mathbb{R}$ dans le cas de la régression.

Principe I

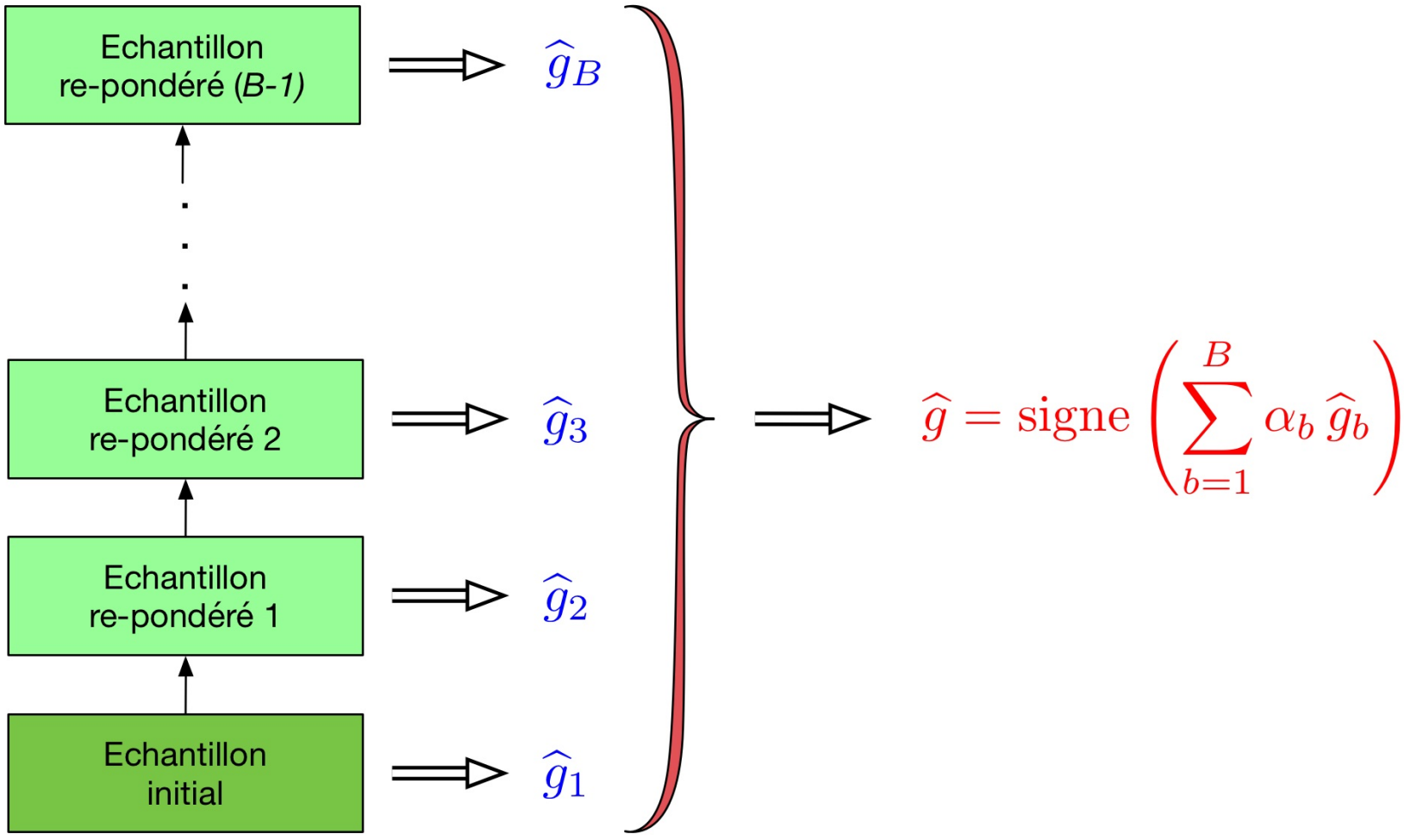
Principe II


Echantillon
re-pondéré 1
$\widehat{g}_2$
Echantillon
initial
$\widehat{g}_1$

Principe III



Principe IV


Echantillon
re-pondéré (B-1)
Echantillon
re-pondéré 2
Echantillon
re-pondéré 1
Echantillon
initial
$\widehat{g}_B$
$\widehat{g}_3$
$\widehat{g}_2$
$\widehat{g}_1$
$\widehat{g} = \text{signe}\left(\sum_{b=1}^{B} \alpha_b \widehat{g}_b\right)$

Enjeux

- Comment repondérer les observations à chaque étape ?
- Quels poids accorder aux estimateurs à chaque étape ?

Plan

Règle faible

- On appelle une règle faible (*weak learner*) un prédicteur légèrement meilleur que le hasard.
- Par exemple, dans le cas de classification :

$$\mathbb{P}\left(g(X) \neq Y\right) = \frac{1}{2} - \gamma \ .$$

où $\gamma > 0$.

- On considère usuellement comme règle faible : 1-plus proche voisin, arbre à 2 feuilles, etc.

Historiquement : AdaBoost (Freund et Schapire, 1997) I

1. Initialisation des poids des individus :

$$\forall i \in \{1, \ldots, n\} : w_i^{(1)} = \frac{1}{n} \ .$$

2. Pour $b \in \{1, \ldots, B\}$:

   2.1 Estimer $\widehat{g}_b$ avec les poids $\left(w_1^{(b)}, \ldots, w_n^{(b)}\right)$ pour l'échantillon.

   2.2 Calcul du taux d'erreur e_b de $\widehat{g}_b$:

$$e_b = \frac{\sum_{i=1}^n w_i^{(b)} \mathbb{1}_{\widehat{g}_b(x_i) \neq y_i}}{\sum_{i=1}^n w_i^{(b)}} \ .$$

   2.3 Calcul de la pénalité :

$$\alpha_b = \ln\left(\frac{1 - e_b}{e_b}\right) \ .$$

   2.4 Calcul des nouveaux poids :

$$\forall i \in \{1, \ldots, n\} : w_i^{(b+1)} = w_i^{(b)} \exp\left(\alpha_b \mathbb{1}_{\widehat{g}_b(x_i) \neq y_i}\right) \ .$$

Historiquement : AdaBoost (Freund et Schapire, 1997) II

3. Le prédicteur obtenu au final est :

$$\widehat{g} = \text{signe}\left(\sum_{b=1}^{B} \alpha_b \, \widehat{g}_b\right) .$$

Remarques

- L'étape 2.1 implique que la méthode retenue soit en mesure prendre en compte des poids. Dans le cas contraire, l'étape 2.1 d'estimation de $\widehat{g}_b$ s'effectue sur un échantillon de dimension n issu du tirage au sort d'observations de d_n avec remise selon les poids $\left(w_1^{(b)}, \ldots, w_n^{(b)}\right)$.
- Les poids sont modifiés de manière à accroitre l'importance des observations mal classées et diminuer celle des observations bien classées.
- Le poids α_b du prédicteur $\widehat{g}_b$ augmente avec sa performance : α_b augmente lorsque e_b diminue.

Quelques résultats théoriques

- On a :

$$R_n(\widehat{g}) \leq \exp\left(-2\sum_{b=1}^{B}\gamma_b^2\right)$$

où $\gamma_b = \frac{1}{2} - e_b$ est le gain de $\widehat{g}_b$ par rapport à une décision basée sur le hasard pur.
Si on est meilleur que le hasard alors le risque empirique tend exponentiellement vers 0 avec B.

- On peut montrer que plus B est grand, plus le biais est faible mais plus la variance est élevée.
Il y a un risque de sur-apprentissage si B est trop important. Il faut donc contrôler B par validation croisée.

- On peut montrer qu'AdaBoost est un algorithme qui permet de minimiser le risque empirique « convexifié », et s'inscrit ainsi dans le cadre des méthodes de gradient boosting.

Plan

Une idée générale : la convexification du risque I

- Si on considère la fonction de perte suivante :

$$\ell\left(y, y'\right) = \mathbb{1}_{y \neq y'} \ ,$$

le risque du prédicteur g vaut :

$$R(g) = \mathbb{E}\left(\mathbb{1}_{g(X) \neq Y}\right) = \mathbb{P}\left(g(X) \neq Y\right) \ .$$

- On aimerait trouver l'estimateur qui minimise ce risque, malheureusement ce dernier n'est pas calculable en pratique.
- On considère alors le risque empirique sur $\mathcal{D}_n$:

$$R_n(g) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{g(X_i) \neq Y} \ .$$

- Malheureusement, ce risque empirique n'est pas convexe et donc difficile à optimiser.

Une idée générale : la convexification du risque II

- On souhaite donc trouver sur une fonction de perte telle que le risque empirique soit convexe, et donc facile à optimiser.
- Il nous suffit pour cela de considérer une fonction $\ell : (u, y) \mapsto \ell(u, y)$ convexe en u.
- Dans le cas de la classification supervisée, on peut par exemple choisir :

$$\begin{array}{rccc} \ell : & \mathbb{R} \times \{-1, 1\} & \to & \mathbb{R}^+ \\ & (u, y) & \mapsto & \exp(-y\, u) \end{array}$$

qui est bien convexe en u.

Optimisation

- On considère donc une fonction de perte telle que le risque empirique soit convexe, et donc facile à optimiser.
- On recherche la solution, de manière récursive, au problème :

$$\min_{g} \frac{1}{n} \sum_{i=1}^{n} \ell \left(g \left(X_i \right), Y_i \right) .$$

- On peut utiliser l'algorithme de Newton-Raphson ou plutôt l'algorithme de descente de gradient fonctionnel.

Algorithme de Newton-Raphson

- Soit :

$$J(g) = \sum_{i=1}^{n} \ell\left(g\left(x_i\right), y_i\right) \ .$$

- Avec la notation :

$$\widehat{g}_b = \left(\widehat{g}_b\left(x_1\right), \ldots, \widehat{g}_b\left(x_n\right)\right)^\top \ ,$$

  la formule de récurrence de l'algorithme de Newton-Raphson est :

$$\widehat{g}_b = \widehat{g}_{b-1} - \lambda \nabla J\left(\widehat{g}_{b-1}\right)$$

  où λ est un paramètre de régularisation.

- La fonction $\widehat{g}_b$ n'est calculées qu'aux points $(x_1, \ldots, x_n)$, c'est notamment pour cela qu'on préfère utiliser l'algorithme de descente de gradient fonctionnel.

Algorithme de descente de gradient fonctionnel : cas de la classification supervisée

1. Initialisation :

$$\widehat{g}_0 = \arg\min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} \ell(c, y_i) \ .$$

2. Pour $b \in \{1, \ldots, B\}$:
   - 2.1 Pour $i \in \{1, \ldots, n\}$: calculer les opposés du gradient :

$$U_i = -\frac{\partial}{\partial g(x_i)} \ell(g(x_i), y_i) \Big|_{g(x_i) = \widehat{g}_{b-\mathbf{1}}(x_i)} \ .$$

   - 2.2 Estimer la règle faible sur l'échantillon $((x_1, U_1), \ldots, (x_n, U_n))$. On obtient ainsi $\widehat{h}_b$.
   - 2.3 Mettre à jour :

$$\widehat{g}_b(x) = \widehat{g}_{b-1}(x) + \lambda \, \widehat{h}_b(x)$$

   où λ est un paramètre de régularisation.

3. Le prédicteur obtenu au final est :

$$\widehat{g} = \text{signe}\left(\widehat{g}_B\right) \ .$$

Algorithme de descente de gradient fonctionnel : cas de la régression

1. Initialisation :

$$\widehat{m}_0 = \arg\min_{c \in \mathbb{R}} \frac{1}{n} \sum_{i=1}^{n} \ell\left(c, y_i\right) .$$

2. Pour $b \in \{1, \ldots, B\}$:
   2.1 Pour $i \in \{1, \ldots, n\}$: calculer les opposés du gradient :

$$U_i = -\frac{\partial}{\partial m\left(x_i\right)} \ell\left(m\left(x_i\right), y_i\right)\Big|_{m(x_i)=\widehat{m}_{b-1}(x_i)} .$$

   2.2 Estimer la règle faible sur l'échantillon $\left(\left(x_1, U_1\right), \ldots, \left(x_n, U_n\right)\right)$. On obtient ainsi $\widehat{h}_b$.

   2.3 Mettre à jour :

$$\widehat{m}_b(x) = \widehat{m}_{b-1}(x) + \lambda\, \widehat{h}_b(x)$$

   où λ est un paramètre de régularisation.

3. Le prédicteur obtenu au final est :

$$\widehat{m} = \widehat{m}_B .$$

Remarques

- On peut mettre en évidence l'agrégation en remarquant que :

$$\widehat{g}_B(x) = \widehat{g}_0(x) + \lambda \sum_{b=1}^{B} \widehat{h}_b(x) \ .$$

- La vitesse de minimisation dépend des deux hyper-paramètres λ et B : si λ diminue, B augmente.

Retour sur AdaBoost

- AdaBoost est équivalent à un problème de gradient boosting avec $\lambda = 1$ et la fonction de perte :

$$\begin{array}{rccc} \ell : & \mathbb{R} \times \{-1, 1\} & \to & \mathbb{R}^+ \\ & (u, y) & \mapsto & \exp(-y\, u) \end{array}$$

- $\widehat{g}(x)$ est un estimateur de :

$$g^\star(x) = \frac{1}{2} \ln \left(\frac{\mathbb{P}(Y = 1 / X = x)}{\mathbb{P}(Y = -1 / X = x)} \right) \ .$$

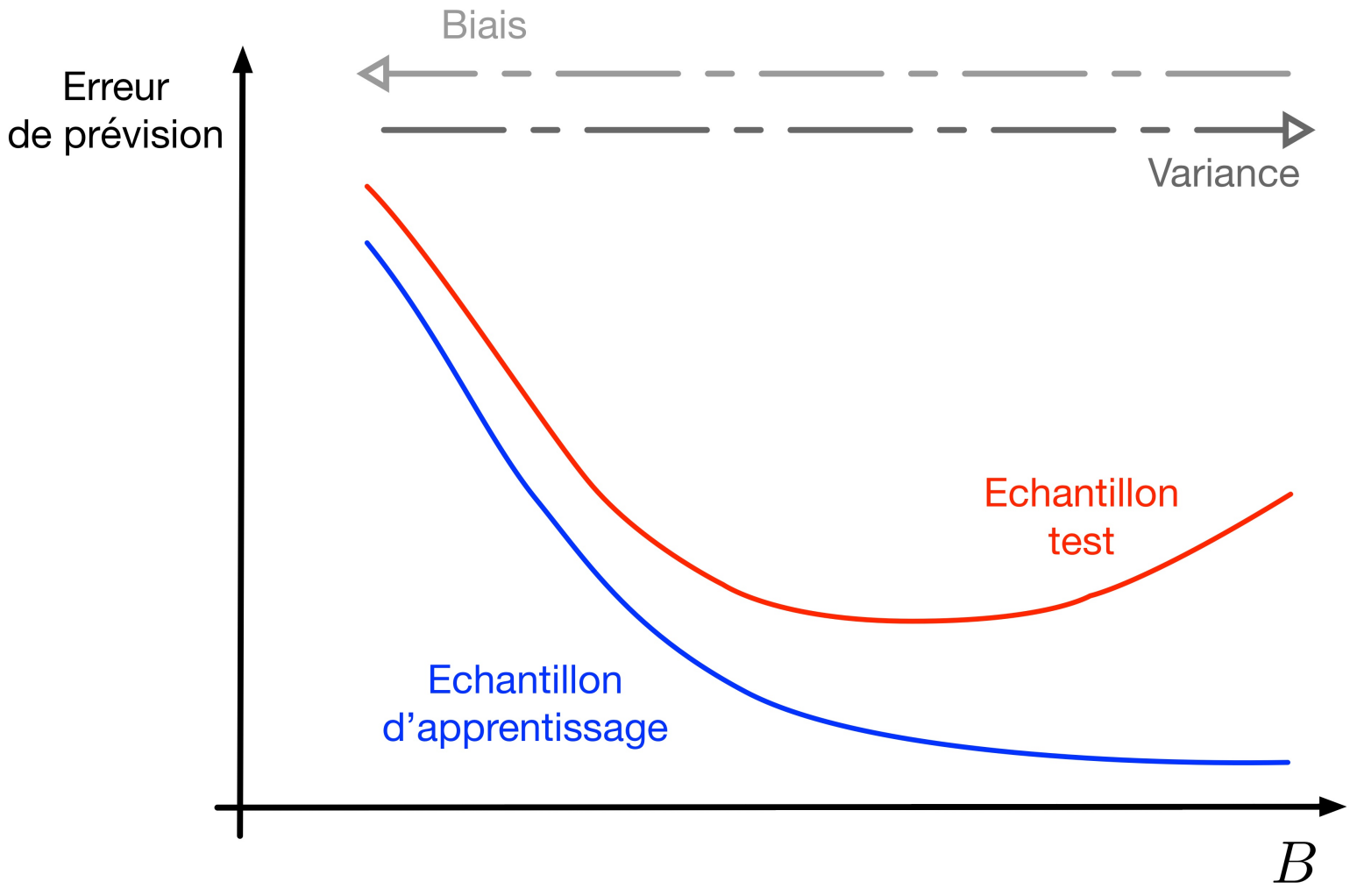
Justification de la règle faible

- Le boosting réduit le biais mais pas la variance.
- La règle faible utilisée doit donc avoir une variance faible (et donc un biais élevé), comme les arbres avec très peu de feuilles.
- Considérer des règles non faibles n'améliore pas les performances de l'algorithme en général.

Choix du nombre d'itérations I

- Le biais diminue avec B.
- Mais lorsque B est « trop » grand, la variance est importante : il existe un risque de sur-apprentissage.
- On choisit B par validation croisée.

Choix du nombre d'itérations II



Plan

LogitBoost I

- Le modèle de régression logistique, entre une variable $Y \in \{0, 1\}$ et des covariables $X = (X_1, \ldots, X_p)^\top \in \mathbb{R}^p$, pose :

$$p(x) := \mathbb{P}(Y = 1 / X = x) = \frac{\exp\left(\beta^\top x\right)}{1 + \exp\left(\beta^\top x\right)} = \frac{1}{1 + \exp\left(-\beta^\top x\right)}$$

où $\beta \in \mathbb{R}^p$ est un paramètre inconnu estimé par maximum de vraisemblance.

- L'idée de LogitBoost est de supprimer l'hypothèse de linéarité de $p(x)$, en posant :

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

où $g : \mathbb{R}^p \to \mathbb{R}$ est une fonction inconnue.

LogitBoost II

- La maximisation de :

$$\ln\left[(p(x))^{y}\,(1-p(x))^{1-y}\right] \quad .$$

est équivalente à la minimisation de :

$$\ln\left[1+\exp\left(-2\tilde{y}g(x)\right)\right]$$

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

- La fonction :

$$\begin{array}{rccc} \ell: & \mathbb{R}\times\{-1,1\} & \to & \mathbb{R}^{+} \\ & (u,y) & \mapsto & \ln\left(1+\exp\left(-2yu\right)\right) \end{array}$$

est bien convexe en u.

LogitBoost III

- LogitBoost désigne donc l'algorithme de gradient boosting avec la fonction de perte précédente.
- Comme pour AdaBoost, la solution $\widehat{g}(x)$ de LogitBoost est un estimateur de :

$$g^\star(x) = \frac{1}{2} \ln \left(\frac{\mathbb{P}(Y = 1 / X = x)}{\mathbb{P}(Y = -1 / X = x)} \right) .$$

- Une fois la fonction $\widehat{g}$ déterminée, on obtient la règle de décision suivante :

$$y = \begin{cases} 1 & \text{si } \widehat{p}(x) \geq \frac{1}{2} \\ -1 & \text{sinon} \end{cases} ,$$

soit

$$y = \begin{cases} 1 & \text{si } \widehat{g}(x) \geq 0 \\ -1 & \text{sinon} \end{cases} .$$

Autres exemples en classification supervisée

On peut considérer différentes fonctions φ telles que :

$$\ell\left(g(x), y\right) = \varphi\left(y\, g(x)\right) ,$$

par exemple :

- Fonction de perte quadratique :

$$\varphi(x) = (1 - x)^2 .$$

- Fonction de perte quadratique tronquée :

$$\varphi(x) = \left[(1 - x)_+\right]^2 .$$

- Fonction de perte Hinge (SVM) :

$$\varphi(x) = (1 - x)_+ .$$

L_2-Boosting I

- Le L_2-Boosting s'applique dans le cadre de la régression.
- L_2-Boosting désigne l'algorithme de gradient boosting avec la fonction de perte :

$$\begin{array}{rccc} \ell: & \mathbb{R}^2 & \to & \mathbb{R}^+ \\ & (u,y) & \mapsto & \frac{1}{2}(u-y)^2 \end{array}$$

- La solution $\widehat{m}(x)$ de L_2-Boosting est un estimateur de :

$$m^\star(x) = \mathbb{E}(Y/X = x) \ .$$

L_2-Boosting II

- Dans l'étape 2.1 du gradient boosting, on obtient :

$$U_i = -\frac{\partial}{\partial m(x_i)} \ell(m(x_i), y_i)\Big|_{m(x_i)=\widehat{m}_{b-1}(x_i)}$$
$$= y_i - \widehat{m}_{b-1}(x_i)$$

- Les U_i sont donc les résidus du prédicteur $\widehat{m}_{b-1}$ à l'étape $(b-1)$.

- Le prédicteur $\widehat{m}_b$ est élaboré via une régression sur les résidus de l'étape $(b-1)$.
  $\widehat{m}_{b-1}$ est amendée en expliquant l'information subsistant dans les résidus de l'étape $(b-1)$.

Le coin R

On peut utiliser plusieurs packages, parmi lesquels :

- Le package `gbm` :
  - La fonction de base est `gbm`.
  - L'option `distribution` permet de spécifier la méthode : `adaboost` pour AdaBoost, `bernoulli` pour LogitBoost, `gaussian` pour L_2-Boosting, etc.
  - L'option `n.trees` permet de choisir le nombre d'arbres.
  - L'option `interaction.depth` permet de choisir la complexité des arbres.
  - L'option `shrinkage` correspond à λ.
  - La fonction `gbm.perf` permet de choisir le nombre d'itérations optimal.

- Le package `caret` :
  - La fonction de base est `train`.
  - On utilise une des options correspondant aux SVM, par exemple :
    - `method="ada"` pour AdaBoost,
    - `method="LogitBoost"` pour LogitBoost.

Références

Boyd, S. et L. Vandenberghe. 2003, *Convex optimization*, Cambridge University Press.

Freund, Y. et R. E. Schapire. 1997, «A decision-theoretic generalization of on-line learning and an application to boosting», *Journal of Computer and System Sciences*, vol. 55, n° 1, p. 119–139.

Schapire, R. E. et Y. Freund. 2012, *Boosting. Foundations and algorithms*, Adaptive Computation and Machine Learning, MIT Press.