Agrégation séquentielle : boosting

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

De quoi s'agît-il ?

Gradient Boosting = Gradient Descent + Boosting

De quoi s'agît-il ?

- Premier algorithme de "boosting" [Freund and Schapire, 1997].
- Contruire une famille de règles qui sont ensuite agrégées.
- Processus récursif : la règle construite à l'étape k dépend de celle construite à l'étape $k-1$

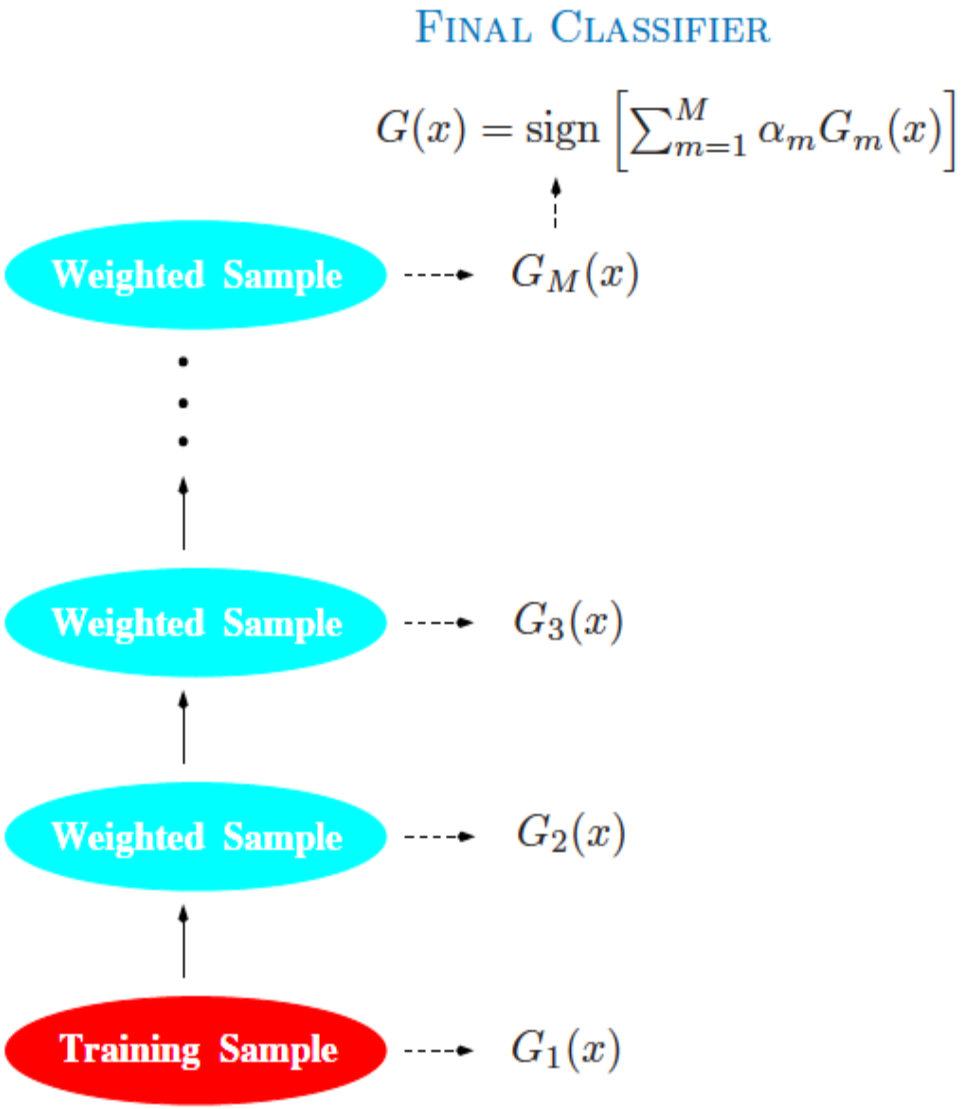
Un peu d'histoire

- 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 (Hastie et al. 2009)



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:*

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

- **Exemple :** arbre à 2 feuilles.

Schéma ou idée

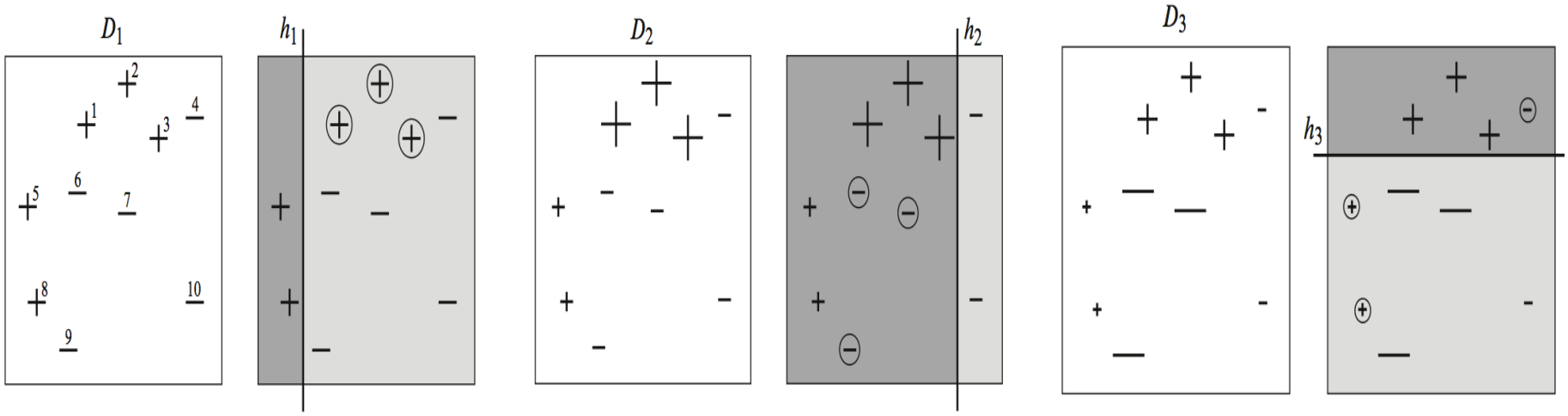


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

Algorithme dit Adaboost.M1

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

Algorithm of [Freund and Schapire 1997]:

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

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

   c. Calcul de : $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
   d. Réajuster les poids + normalisation

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

Output:

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

Schéma ou idée

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

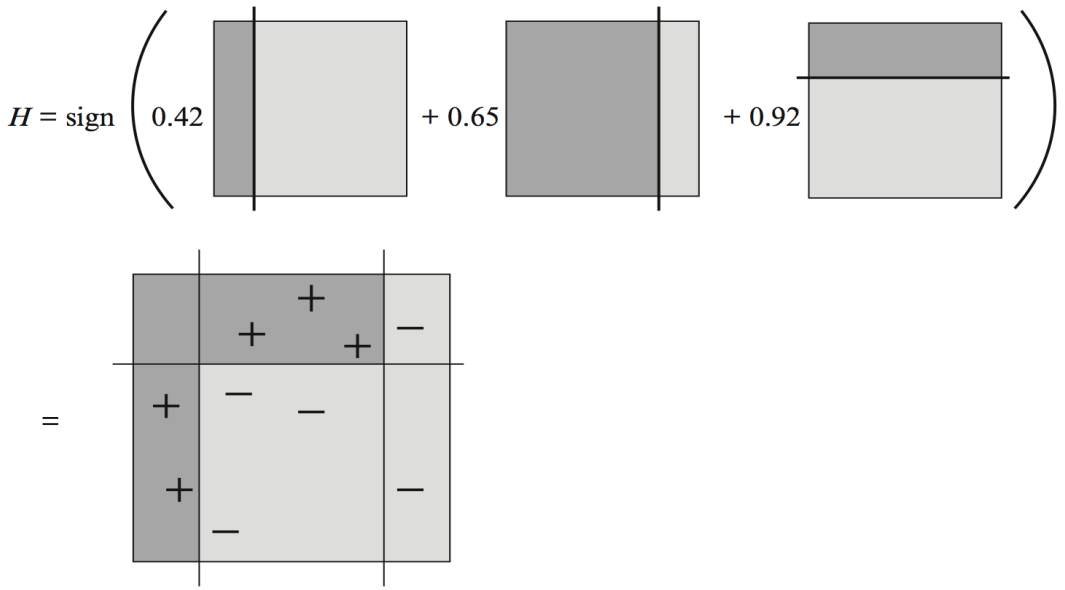


Figure: AdaBoost. Source: Figure 1.2 of [Schapire and Freund, 2012]

Cet algorithme a été introduit en 1996 par Yoav Freund and Rob Shapire (prix Gödel 2003)

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 $w_1, \ldots, w_n$.
- Les poids $w_1, \ldots, w_n$ sont mis à jour à chaque itération : si le $i^{\text{ième}}$ individu est bien classé son poids est inchangé, sinon il est augmenté.
- Le poids α_m de la règle g_m augmente avec la performance de g_m mesurée sur d_n : α_m augmente lorsque e_m diminue (il faut néanmoins que g_m ne soit pas trop faible : si $e_m > 0.5$ alors $\alpha_m < 0$!!!).

Erreur empirique d'apprentissage

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

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

- γ_m désigne le gain de la règle g_m par rapport à une règle **pûrement aléatoire**

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

Propriété: [Freund and Schapire, 1999]

$$L_n\left(\widehat{g}_M\right) \leq \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right).$$

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

Erreur empirique d'apprentissage (suite)

Ils ont montré que

$$L_n\left(\widehat{g}_M\right) = \frac{1}{n}\sum_{i=1}^{n} 1_{y_i \neq \widehat{g}_M(x_i)} \leq \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right) \leq \exp\left(-2M\gamma^2\right)$$

Erreur de généralisation

Définition : C'est l'erreur moyenne attendue sur un échantillon test

$$L\left(\widehat{g}_M\right) = \mathbb{P}\left[Y \neq \widehat{g}_M(X)\right]$$

Borne obtenue par Freund & Schapire

$$L\left(\widehat{g}_M\right) \leq L_n\left(\widehat{g}_M\right) + \mathcal{O}\left(\sqrt{\frac{MV}{n}}\right)$$

où V est la dimension de Vapnik-Chervonenkis de la famille de règles de classification faibles (3 dans l'exemple simple).

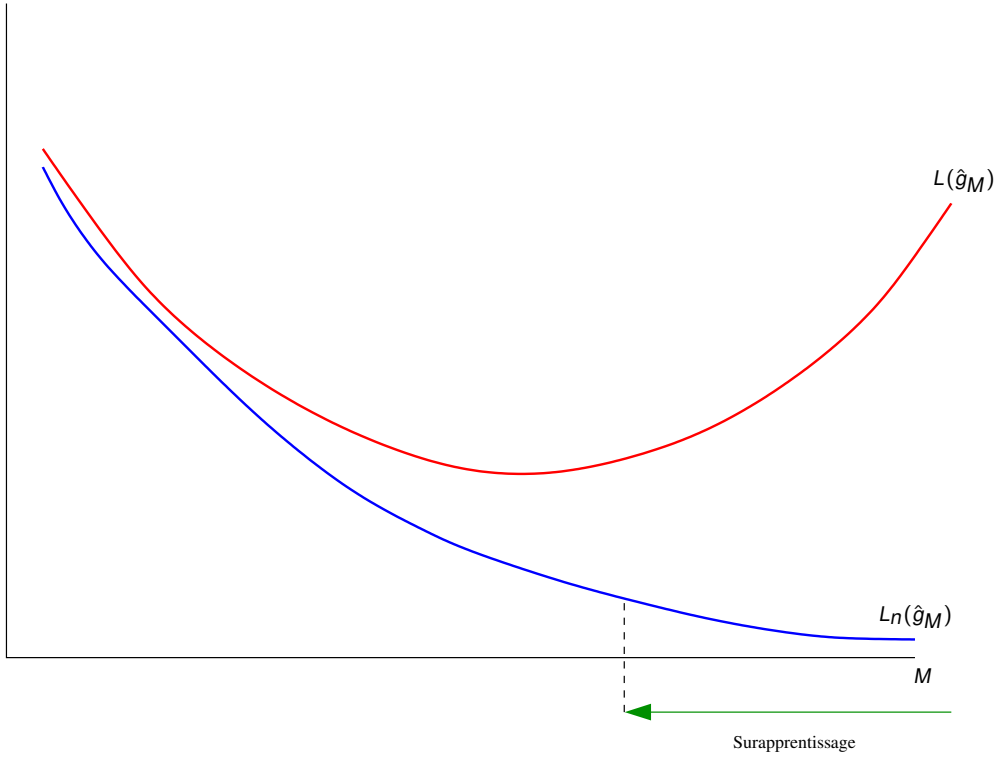
Erreur de généralisation (suite)

Interprétation Il peut y avoir du sur-ajustement

- Le compromis biais/variance ou erreur approximation/estimation est régulé par le nombre d'itérations M :
  1. M petit $\rightarrow$ premier terme (approximation) domine
  2. 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).

Sur-apprentissage: Qu'est-ce que c'est ?

C'est ce qui se passe quand en complexifiant le modèle l'erreur d'apprentissage baisse, alors que l'erreur de généralisation se remet à augmenter.

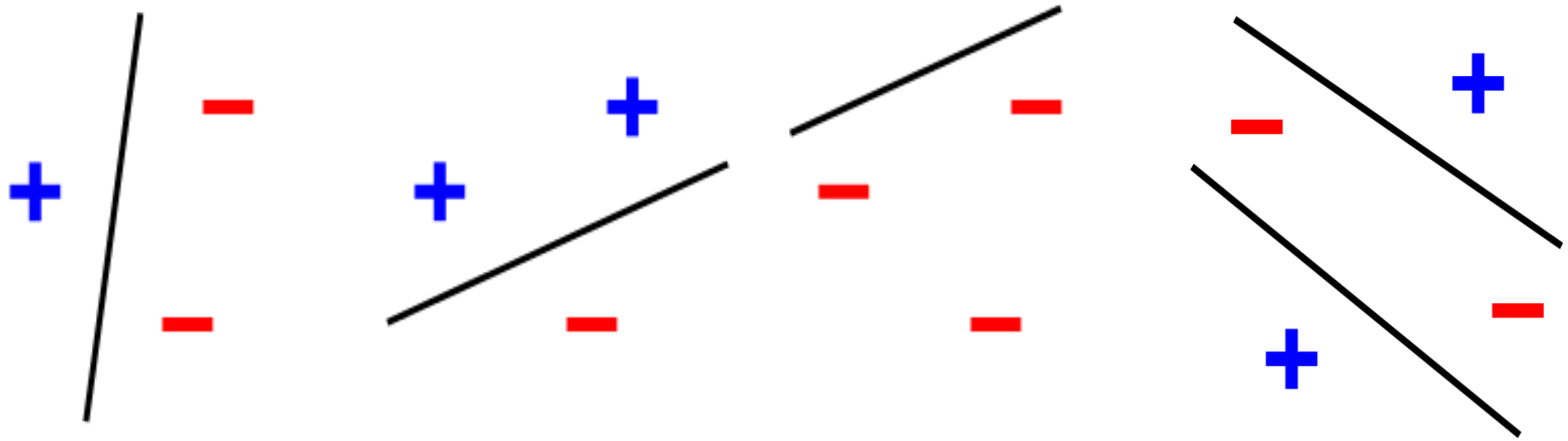


Dimension de Vapnik-Chervonenkis : Qu'est-ce que c'est ?

C'est une mesure de la capacité d'un algorithme de classification statistique

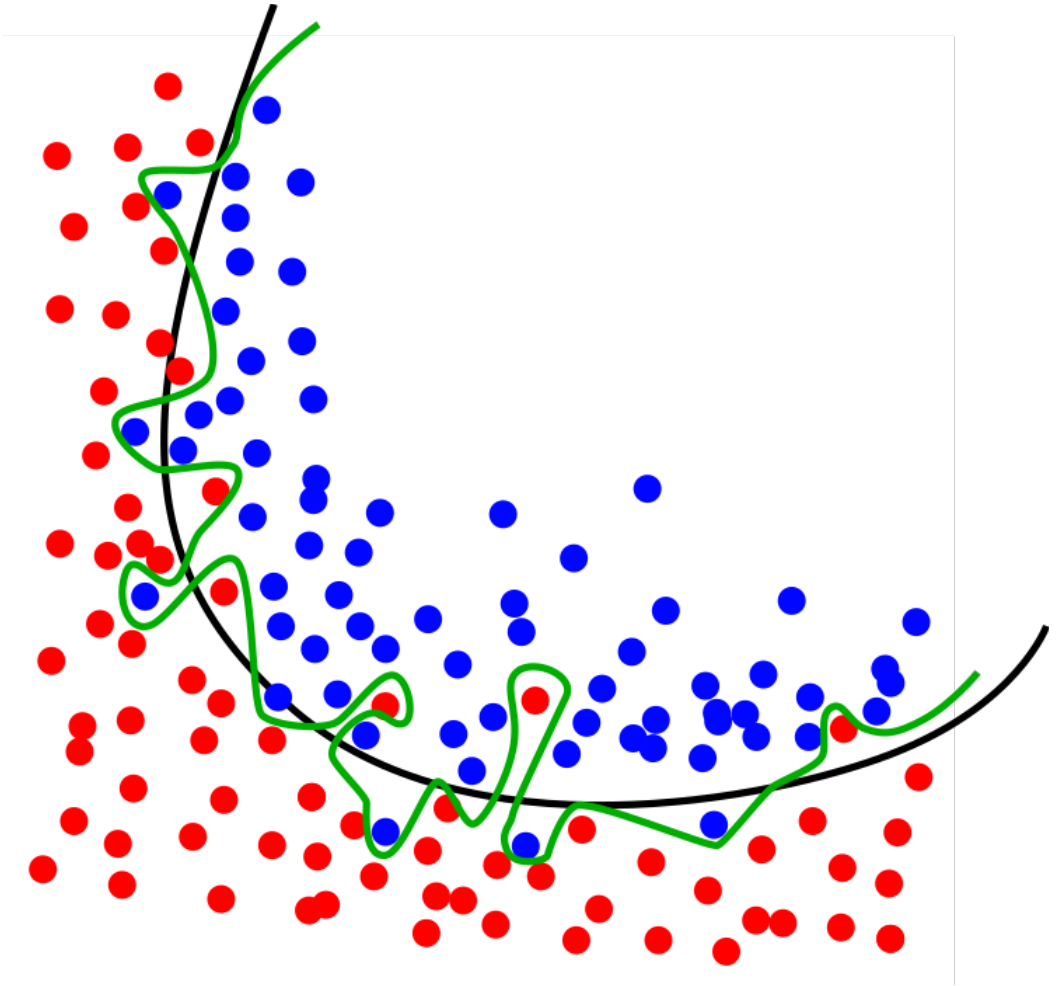
- cardinal du plus grand ensemble de points que l'algorithme peut **pulvériser**
- **Pulvériser:** un modèle de classification g_θ pulvérise un ensemble de données $E = (x_1, x_2, \ldots, x_n)$ si, pour tout étiquetage E, il existe θ tel que g_θ ne fasse aucune erreur dans l'évaluation de cet ensemble de données.
- Une droite en dimension 2 : on peut pulvériser 3 points mais pas 4 points!

Dimension de Vapnik-Chervonenkis



Dimension de Vapnik-Chervonenkis (suite)

Un modèle de dimension VC trop haute risque le sur-apprentissage par un modèle complexe trop adapté aux données d'apprentissage



Gradient boosting pour la régression (intuitif)

- Nous disposons de $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, et une fonction $\hat{f}$ qui minimise l'erreur quadratique moyenne.
- On fait une petit vérification et constate quelques écarts à la vérité : $\hat{f}(x_1) = 0.8$ alors que $y_1 = 0.9$, et $\hat{f}(x_2) = 1.4$ et $y_2 = 1.3, \ldots$. Comment améliorer $\hat{f}$?
- On ne peut pas modifier $\hat{f}$.
- On peut ajouter un modèle (arbre de régression) h à $\hat{f}$ et la prédiction sera donnée par $\hat{f}(x) + h(x)$.

Gradient boosting pour la régression

Solution simple

$$\hat{f}(x_1) + h(x_1) = y_1$$
$$\hat{f}(x_2) + h(x_2) = y_2$$
$$\hat{f}(x_3) + h(x_3) = y_3$$
$$\dots$$
$$\hat{f}(x_n) + h(x_n) = y_n$$

Gradient boosting pour la régression

Peut-on obtenir un arbre h tel que

$$
\begin{aligned}
h(x_1) &= y_1 - \hat{f}(x_1) \\
h(x_2) &= y_2 - \hat{f}(x_2) \\
h(x_3) &= y_3 - \hat{f}(x_3) \\
&\dots \\
h(x_n) &= y_n - \hat{f}(x_n)
\end{aligned}
$$

Oui mais une approximation !

Gradient boosting pour la régression

Peut-on obtenir un arbre h tel que

$$
\begin{aligned}
h(x_1) &= y_1 - \hat{f}(x_1) \\
h(x_2) &= y_2 - \hat{f}(x_2) \\
h(x_3) &= y_3 - \hat{f}(x_3) \\
&\dots \\
h(x_n) &= y_n - \hat{f}(x_n)
\end{aligned}
$$

Oui mais une approximation !

$$
\big(x_1, y_1 - \hat{f}(x_1)\big), \big(x_2, y_2 - \hat{f}(x_2)\big), \dots, \big(x_n, y_n - \hat{f}(x_n)\big)
$$

Gradient boosting pour la régression

Une solution simple

- $y_i - \hat{f}(x_i)$ sont les résidus. La partie qui échappe à $\hat{f}$.
- Le rôle de h est de compenser les lacunes de $\hat{f}$.
- Si la nouvelle fonction de régression estimée $\hat{f} + h$ demeure insatisfaisante, on peut ajouter d'autres arbres de régression.

Modélisation additive linéaire

Modélisation additive linéaire

Contexte: Classification ou régression (presque le même que pour AdaBoost)

- On a toujours une variable $y \in \{-1, 1\}$ ou $y \in \mathbb{R}$ à inférer à partir de règles faibles.
- Cette fois-ci, on se donne un fonction de coût (ou déviance) $L(y, g)$ que l'on cherche à minimiser.

Approche: On modélise à chaque fois le résidu produit par la solution précédente, on a donc

$$\widehat{g}_M(x) = \sum_{m=1}^{M} \beta_m g_m(x) = \widehat{g}_{M-1}(x) + \beta_M g_M(x)$$

Algorithme Forward staging additive modeling

Entrée: Les éléments nécessaires sont

- un jeu de données $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- un ensemble de règles faibles
- le nombre M d'itérations

Initialisation: $\widehat{g}_0(x)$

Itération: pour $m = 1$ à M

1. ajuster la règle faible g_m et calculer un coefficient β_m qui minimise
$$\sum_{i=1}^{n} L\big(y_i, \widehat{g}_{m-1}(x_i) + \beta_m g_m(x_i)\big)$$
2. $$\widehat{g}_m(x) = \widehat{g}_{m-1}(x) + \beta_m g_m(x)\big)$$

Sortie: La prédiction est $\operatorname{sign}\widehat{g}_M(x)$ (en classification)

Justification du boosting : minimisation de risque empirique

Pertes théorique et empirique

- (X, Y) couple aléatoire à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$. Étant donnée $\mathcal{G}$ une famille de règles, on se pose la question de trouver la meilleure règle dans $\mathcal{G}$.
- Choisir la règle qui minimise une fonction de perte, par exemple

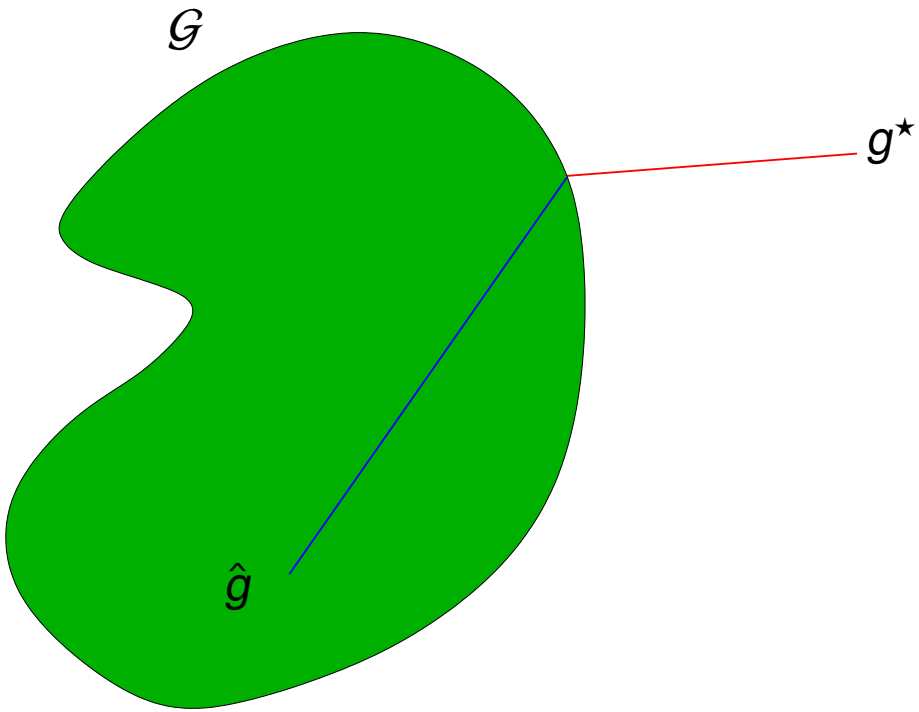
$$L(g) = \mathbb{P}\big(Y \neq g(X)\big).$$

Problème : la fonction de perte n'est pas calculable

- Idée : choisir la règle qui minimise la version empirique de la fonction de perte :

$$L_n(g) = \frac{1}{n} \sum_{i=1}^{n} 1_{g(X_i) \neq Y_i}.$$

Erreurs d'estimation et d'approximation



$$L(\hat{g}) - L^* = L(\hat{g}) - \inf_{g \in \mathcal{G}} L(g) + \inf_{g \in \mathcal{G}} L(g) - L^*.$$

Risque convexifié

Problème : la fonction

$$\begin{aligned} \mathcal{G} &\to \mathbb{R} \\ g &\mapsto \frac{1}{n}\sum_{i=1}^{n} 1_{g(X_i)\neq Y_i} \end{aligned}$$

est généralement difficile à minimiser.
Idée : trouver une autre fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ telle que

$$\begin{aligned} \mathcal{G} &\to \mathbb{R} \\ g &\mapsto \frac{1}{n}\sum_{i=1}^{n} \ell\big(Y_i, g(X_i)\big) \end{aligned}$$

soit "facile" à minimiser (si la fonction fonction $v \mapsto \ell(u, v)$ est convexe par exemple).

Fonction de perte

- La fonction de perte $\ell(y, g(x))$ mesure l'écart entre la quantité à prévoir $y \in \{-1, 1\}$ et $g(x)$.

Fonction de perte

- La fonction de perte $\ell(y, g(x))$ mesure l'écart entre la quantité à prévoir $y \in \{-1, 1\}$ et $g(x)$.
- Elle doit donc prendre des valeurs
  - élevées lorsque $yg(x) < 0$
  - faibles lorsque $yg(x) > 0$

Fonction de perte

- La fonction de perte $\ell(y, g(x))$ mesure l'écart entre la quantité à prévoir $y \in \{-1, 1\}$ et $g(x)$.
- Elle doit donc prendre des valeurs
  - élevées lorsque $yg(x) < 0$
  - faibles lorsque $yg(x) > 0$
- Exemple:
  1. $\ell(y, g(x)) = 1_{yg(x)<0}$
  2. $\ell(y, g(x)) = \exp\big(-yg(x)\big)$ (présente l'avantage d'être convexe en le second argument).

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \operatorname{argmin} \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \operatorname{argmin} \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

- Stratégie : étant donnée un n échantillon i.i.d $(X_1, Y_1), \ldots, (X_n, Y_n)$ de même loi que (X, Y), on cherche à minimiser la version empirique de $\mathbb{E}\left[\ell\big(Y, g(X)\big)\right]$:

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

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \text{argmin}\, \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

- Stratégie : étant donnée un n échantillon i.i.d $\big(X_1, Y_1\big), \ldots, \big(X_n, Y_n\big)$ de même loi que $\big(X, Y\big)$, on cherche à minimiser la version empirique de $\mathbb{E}\left[\ell\big(Y, g(X)\big)\right]$:

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

- Approche récursive : approcher g^* par $\widehat{g}_M(x) = \sum_{m=1}^{M} \beta_m g_m(x)$ où g_m et β_m sont construits de façon récursive.

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \text{argmin}\, \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

- Stratégie : étant donnée un n échantillon i.i.d $(X_1, Y_1), \ldots, (X_n, Y_n)$ de même loi que (X, Y), on cherche à minimiser la version empirique de $\mathbb{E}\left[\ell\big(Y, g(X)\big)\right]$:

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

- Approche récursive : approcher g^* par $\widehat{g}_M(x) = \sum_{m=1}^{M} \beta_m g_m(x)$ où g_m et β_m sont construits de façon récursive.
- Méthode : utiliser une approche numérique (descente de gradients, Newton-Raphson).

Fonctions de coût pour la classification

Exponentielle

$$L(y, g) = \exp(-yg)$$

- On peut prouver qu'on retrouve AdaBoost !!
- Pourtant l'idée est très différente

Logistique $L(y, g) = \log\left(1 + \exp(-2yg)\right)$

- Similaire à AdaBoost a priori
- Moins sensible aux observations mal classées

Fonctions de coût pour la régression

Quadratique

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

- sensible aux valeurs aberrantes ou extrêmes

Absolue

$$\ell(y, g) = |y - g|$$

- Plus robuste, mais moins précis pour les petites erreurs

Huber

$$\ell(y, g) = (y - g)^2 1_{|y-g| \leq \delta} + \left(2\delta|y - g| - \delta^2\right) 1_{|y-g| > \delta}$$

- combine les bonnes propriétés des deux fonctions précédentes

Un petit rappel

Nous faisons ici un bref rappel sur la méthode de Newton-raphson dans le cas simple de la minimisation d'une fonction strictement convexe $J : \mathbb{R} \to \mathbb{R}$. Si on désigne par $\tilde{x}$ la solution du problème de minimisation, la méthode consiste à construire une suite (x_k) qui converge vers $\tilde{x}$. La suite est tout d'abord initialisée en choisissant une valeur x_0. On cherche alors $x_1 = x_0 + h$ tel que $J'(x_1) \approx 0$. Par un développement limité, on obtient l'approximation

$$J'(x_0 + h) \approx J'(x_0) + hJ''(x_0).$$

Comme $J'(x_0 + h) \approx 0$, il vient $h = -\big(J''(x_0)\big)^{-1} J'(x_0)$. Si on pose $\lambda = \big(J''(x_0)\big)^{-1}$, alors $x_1 = x_0 - \lambda J'(x_0)$ et on déduit la formule de récurrence

$$x_k = x_{k-1} - \lambda J'(x_{k-1}).$$

Newton Raphson

- On note $\boldsymbol{g_m} = \big(g_m(x_1), \ldots, g_m(x_n)\big)$, et

$$J\big(\boldsymbol{g_m}\big) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, g_m(x_i)\big).$$

Newton Raphson

- On note $\boldsymbol{g_m} = \big(g_m(x_1), \ldots, g_m(x_n)\big)$, et

$$J\big(\boldsymbol{g_m}\big) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(y_i, g_m(x_i)\big).$$

- La formule de récurrence de l'algorithme de Newton-Raphson est donnée par

$$\boldsymbol{g_m} = \boldsymbol{g_{m-1}} - \lambda \nabla J\big(\boldsymbol{g_{m-1}}\big),$$

où $\lambda > 0$ désigne le pas de descente de gradient.

Newton Raphson

- On note $\boldsymbol{g_m} = \big(g_m(x_1), \ldots, g_m(x_n)\big)$, et

$$J\big(\boldsymbol{g_m}\big) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(y_i, g_m(x_i)\big).$$

- La formule de récurrence de l'algorithme de Newton-Raphson est donnée par

$$\boldsymbol{g_m} = \boldsymbol{g_{m-1}} - \lambda \nabla J\big(\boldsymbol{g_{m-1}}\big),$$

où $\lambda > 0$ désigne le pas de descente de gradient.

- **Inconvénients**
  a. Cet algorithme permet de calculer l'estimateur uniquement en les points du design $x_1, \ldots, x_n$.
  b. Ne prend pas en compte une éventuelle régularité de la fonction à estimer (si x_i est proche de x_j alors $g^*(x_i)$ est proche de $g^*(x_j)$).

Boosting par descente du gradient

Entrées :

- $(x_1, y_1), \ldots, (x_n, y_n)$ l'échantillon, λ un paramètre de régularisation tel que $\lambda > 0$ et M le nombre d'itérations.

a. Initialisation : $\widehat{g}_0(\cdot) = \text{argmin}_c \sum_{i=1}^{n} \ell(y_i, c)$

b. **Pour** $m = 1$ à M :

1..1 Calculer l'opposé du gradient et l'évaluer aux points d'observation

$$r_{im} = -\frac{\partial}{\partial g(x_i)} \ell\big(y_i, g_m(x_i)\big)\Big|_{y=y_i, g(x_i)=\widehat{g}_{m-1}(x_i)}, \quad i = 1, \ldots, n.$$

2..2 ajuster une règle faible g_m sur l'échantillon $\big(x_1, r_{1m}\big), \ldots, \big(x_n, r_{nm}\big)$

3..3 Mise à jour : $\widehat{g}_m(x) = \widehat{g}_{m-1}(x) + \lambda g_m(x)$.

c. **Sortie** : La règle $\widehat{g}_M(x)$ pour la régression et $\text{sign}\widehat{g}_M(x)$ pour la classification.

Boosting par descente du gradient avec des arbres

- Notation formelle d'un arbre

$$T(x, \Theta) = \sum_{j=1}^{J} \gamma_j 1(x \in R_j)$$

où $\Theta = \{R_j, \gamma_j\}_1^J$

- Un arbre boosté donnera

$$\widehat{f}_M(x) = \sum_{m=1}^{M} T(x, \Theta_m)$$

Boosting par descente du gradient avec des arbres

Entrées :

- $(x_1, y_1), \ldots, (x_n, y_n)$ l'échantillon, λ un paramètre de régularisation tel que $\lambda > 0$ et M le nombre d'itérations.

a. Initialisation : $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} \ell(y_i, \gamma)$

b. **Pour** $m = 1$ à M :

1..1 Calculer l'opposé du gradient et l'évaluer aux points d'observation

$$r_{im} = -\frac{\partial}{\partial g(x_i)} \ell\big(y_i, f_m(x_i)\big)\Big|_{y=y_i, f(x_i)=f_{m-1}(x_i)}, \quad i = 1, \ldots, n.$$

2..2 ajuster un arbre sur l'échantillon $\big(x_1, r_{1m}\big), \ldots, \big(x_n, r_{nm}\big)$ qui donne les feuilles $R_{jm}, j = 1, \ldots J_m$.

3..3 Pour $j = 1, \ldots, J_m$ calculer

$$\gamma_{jm} = \text{argmin}_\gamma \sum_{x_i \in R_{jm}} \ell\big(y_i, f_{m-1}(x_i) + \gamma\big)$$

4..4 Mise à jour : $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$.

c. $\widehat{f}(x) = f_M(x)$