Agrégation de modèles

modèles ensemblistes

masedki.github.io

Université Paris-Saclay & CESP Inserm-1018

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 $Z_1, \ldots, Z_n$, chacune de variance σ^2, la variance de la moyenne $\bar{Z}$ est σ^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 $\hat{f}^{*b}(x)$, $b = 1, \ldots, B$
- 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 $\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e}$.
- Le nombre d'observations qui ne font pas partie d'un tirage bootstrap est $n\left(1 - \frac{1}{n}\right)^n \approx \frac{n}{e}$. 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

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

- 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 $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ et on rappelle la fonction de régression

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

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

$$\mathbb{E}\Big[\big(\hat{m}(x) - m^*(x)\big)^2\Big] = \Big(\mathbb{E}\big(\hat{m}(x)\big) - m^*(x)\Big)^2 + \mathrm{Var}\big(\hat{m}(x)\big).$$

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

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

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 $\theta_b = \theta_b\Big(\mathcal{D}_n\Big)$ 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 $\hat{m}_B(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{m}(x, \theta_b)$.
- $\sigma^2(x) = \mathrm{Var}\Big(\hat{m}(x, \theta_b)\Big)$
- $\rho(x) = \mathrm{corr}\Big(\hat{m}(x, \theta_1), \hat{m}(x, \theta_2)\Big)$, 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 θ. On suppose que les estimateurs $\hat{m}(x, \theta_1), \ldots, \hat{m}(x, \theta_B)$ sont identiquement distribués.
- ***Proposition*** *On a :*

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

Par conséquent

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

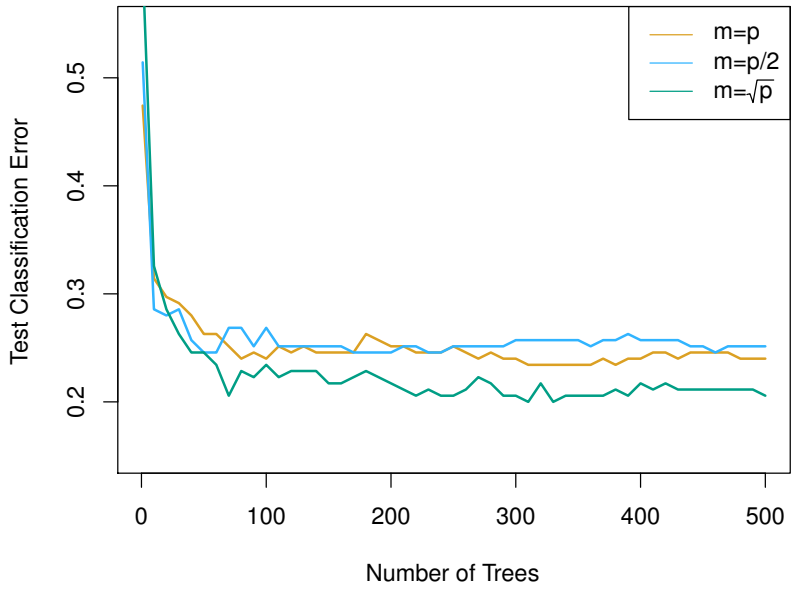
Forêts aléatoires très proche du bagging

- 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 gênes



- 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'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 $k-1$

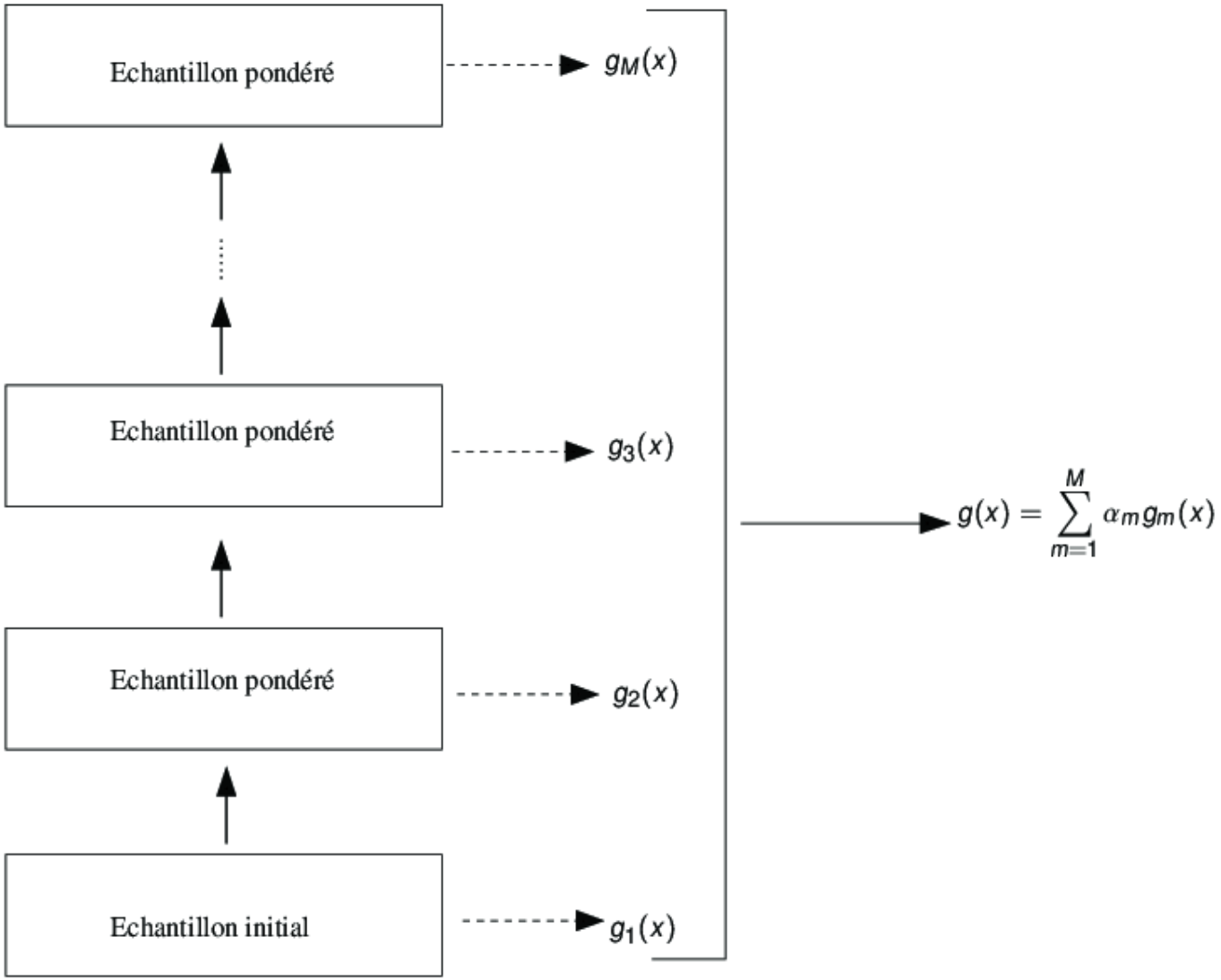
Un peu d'histoire

- 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


Echantillon pondéré
$g_M(x)$
Echantillon pondéré
$g_3(x)$
Echantillon pondéré
$g_2(x)$
Echantillon initial
$g_1(x)$
$g(x) = \sum_{m=1}^{M} \alpha_m g_m(x)$

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

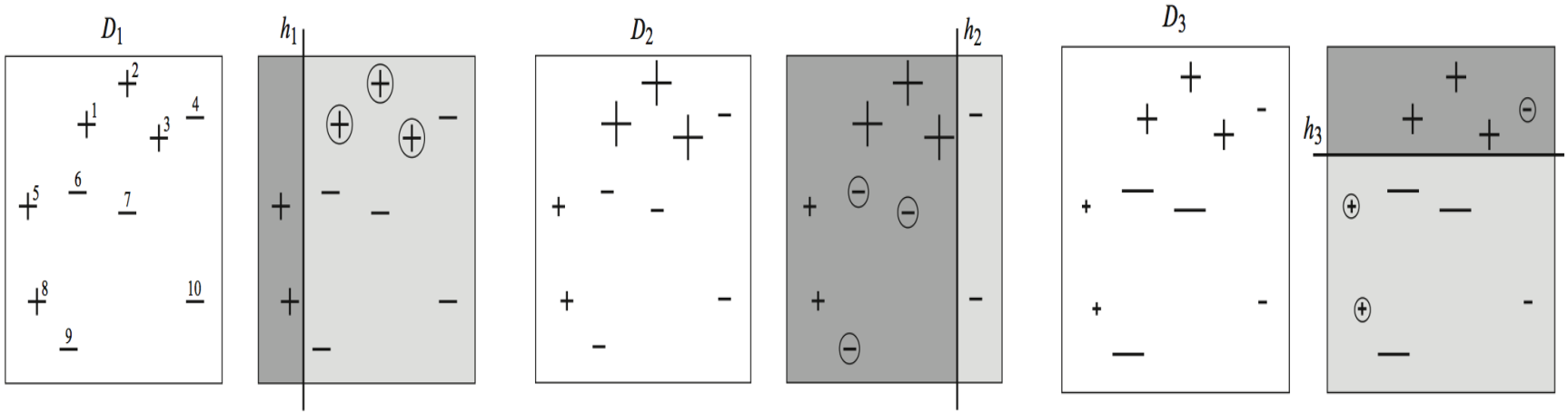


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

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]:

- Initialisser les poids $w_i = \frac{1}{n}, i = 1, \ldots, n$
- **Pour** $m = 1$ à M:
  - 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
  - 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 :

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

Output:

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

Schéma ou idée

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

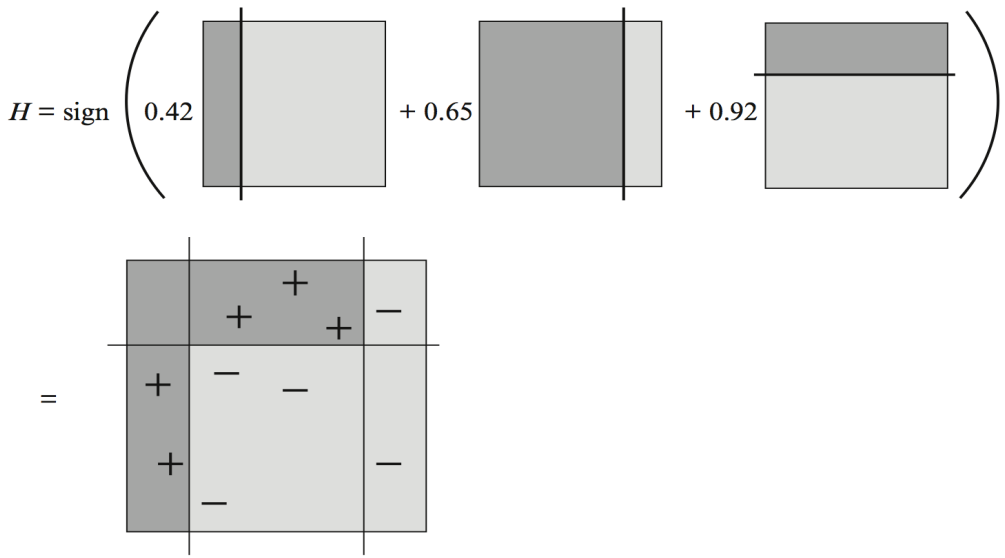


Figure: AdaBoost. Source: Figure 1.2 of [Schapire and Freund, 2012]

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 $\alpha_m < 0$!!!).

Quelques garanties théoriques : contrôle de l'erreur empirique

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

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

- γ_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]

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

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

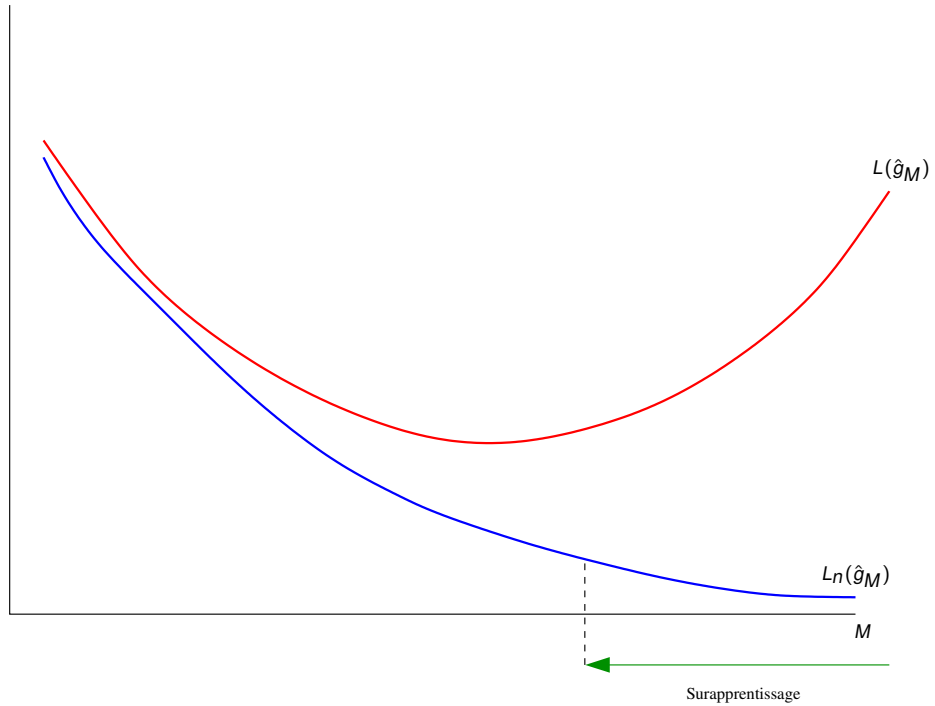
Contrôle de l'erreur de généralisation

Propriétés [Freund and Schapire, 1999]

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

- 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.

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 on 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 $\hat{f}$?
- Nous avons une contrainte : 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 !

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

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.

Risques 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

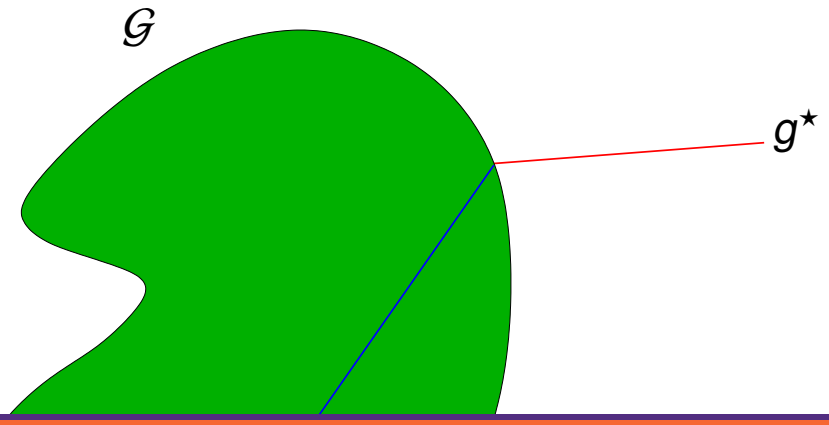
$$\mathcal{R}(g) = \mathbb{P}\big(Y \neq g(X)\big).$$

Problème : la fonction de perte n'est pas calculable

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

$$\mathcal{R}_n(g) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{g(X_i) \neq Y_i}.$$

Erreurs d'estimation et d'approximation



Risque convexifié

Problème : la fonction

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

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 $g(x)$.
- Elle doit donc prendre des valeurs élevées lorsque $yg(x) < 0$ et faibles lorsque $yg(x) > 0$
- Un exemple $\ell(y, g(x)) = \mathbf{1}_{yg(x)<0}$
- Un autre exemple $\ell(y, g(x)) = \exp\big(-yg(x)\big)$ (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

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

- 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 $\hat{g}(x) = \sum_{m=1}^{M} g_m(x)$ où g_m sont construits de façon récursive.
- Méthode : utiliser une approche numérique (descente de gradients, Newton-Raphson).

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

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

- 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

- Cet algorithme permet de calculer l'estimateur uniquement en les points du design $x_1, \ldots, x_n$.
- 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 :

- $d_n = (x_1, y_1), \ldots, (x_n, y_n)$ l'échantillon, λ un paramètre de régularisation tel que $0 < \lambda \leq 1$
- M le nombre d'itérations.

1. Initialisation : $g_0(\cdot) = \text{argmin}_c \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, c\big)$
2. **Pour** $m = 1$ à M :
   - Calculer l'opposé du gradient $-\frac{\partial}{\partial g(x_i)} \ell\big(y_i, g_m(x_i)\big)$ et l'évaluer aux points $g_{m-1}(x_i)$:

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

   - Ajuster la règle faible sur l'échantillon $\big(x_1, U_1\big), \ldots, \big(x_n, U_n\big)$, on note h_m la règle ainsi définie.
   - Mise à jour : $g_m(x) = g_{m-1}(x) + \lambda h_m(x)$.
3. **Sortie** : La règle $\hat{g}_M(x) = g_m(x)$.

Gradient Boosting Algorithm avec arbres

- Initialisation : $g_0(\cdot) = \text{argmin}_c \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, c\big)$
- **Pour** $m = 1$ à M :
  - Calculer l'opposé du gradient $-\frac{\partial}{\partial g(x_i)} \ell\big(y_i, g_m(x_i)\big)$ et l'évaluer aux points $g_{m-1}(x_i)$:

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

  - Ajuster un arbre à K nœuds terminaux sur l'échantillon $\big(x_1, U_1\big), \ldots, \big(x_n, U_n\big)$.
  - Calculer la prévision optimale pour chaque nœuds $\rho_1, \ldots, \rho_K$: Mise à jour :

$$\rho_k = \underset{\rho}{\text{argmin}} \sum_{x_i \in R_k} \ell\big(y_i, g_{m-1}(x_i) + \rho\big),$$

    où R_k contient l'ensemble des x_i qui appartiennent au $k^{\text{ième}}$ nœud de l'arbre.
  - Mise à jour : $g_m(x) = g_{m-1}(x) + \lambda \rho_{k(x)}$, $k(x)$ désignant le numéro du nœud qui contient x.
- **Sortie :** la règle $\hat{g}_M(x)$.

Boosting par descente du gradient pour la régression

- Fixer $\hat{f}(x) = 0$ et $r_i = y_i$ pour tout i de l'ensemble d'entraînement.
- Pour $m = 1, \ldots, M$ faire
  - Ajuster un arbre $\hat{f}_m$ à d nœuds internes ($d + 1$ feuilles) pour prédire les r_i avec x_i
  - Mettre à jour $\hat{f}$ en ajoutant ce nouvel arbre (à un coefficient λ de réduction près)
$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_m(x)$$
  - Mettre à jour les résidus
$$r_i \leftarrow r_i - \lambda \hat{f}_m(x_i)$$
- Retourner le modèle
$$\hat{f}(x) = \sum_{m=1}^{M} \lambda \hat{f}_m(x).$$

Commentaires

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

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

$\Longrightarrow$ lorsque $\lambda \nearrow$ $M \searrow$ 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 λ 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 $p(x)=\mathbb{P}\Big(Y=1|X=x\Big)$. La vraisemblance d'une observation (x,y) s'écrit

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

Souvent on estime β par maximum de vraisemblance.
Le modèle logitboost repose une approche similaire $g:\mathbb{R}^p\to\mathbb{R}$, on pose

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

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

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

où $\tilde{y}=2y-1\in\{-1,1\}$.

Logitboost

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

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

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

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

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

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

- On obtient la règle de classification

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

L_2 boosting : gradient boosting pour la régression

- On s'intéresse à la régression. On désigne par $f : \mathbb{R}^p \to \mathbb{R}$ une régresseur *faible*. Plus précisémment un régesseur (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 applique 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'agorithme fournit un estimateur $\hat{f}_M$ de
$$f^* = \underset{f}{\operatorname{argmin}}\, \mathbb{E}\left[\frac{1}{2}\Big(Y - f(X)\Big)^2\right],$$
c'est-à-dire la fonction de régression $f^*(x) = \mathbb{E}\Big[Y|X = x\Big]$.
- 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 donce 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 ℓ.
- Récapitulatif

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

Table of Contents