Projet Machine Learning pour la Prévision: boosting pour la régression

Yannig Goude

Introduction

Introduit par R. E. Schapire (1990) puis Freund, Schapire, and others (1996) pour la mise en pratique avec adaboost. Voir aussi Freund, Schapire, and Abe (1999).

repose sur 2 éléments de base:

- les base learners ou weak learners
- descente de gradient

idée: si l'on dispose d'une méthode de base (arbre, régresseur...) légèrement meilleure que le hasard, il est possible de **booster** cette méthode pour en faire un prédicteur honorable. Booster consiste à agréger séquentiellement des prédicteurs de base apris sur des sous-ensembles bien choisis des données. C'est un ensemble puissant de procedures d'apprentissage machine pour la régression, la classification.

par rapport au bagging, le boosting gagne à la fois sur la variance et sur le biais

Breiman and others (1998) ont interprété AdaBoost comme une descente de gradient. Cette interprétation a permis de connecter le boosting à des procédures d'optimisation usuelles en inférence statistique.

Cela a permis la généralisation de l'algorithme à :

- classification avec plus de deux classes
- diversité de fonctions de perte
- boosting pour la régression
- ranking

Principes

Imaginons le jeu suivant:

on dispose d'un échantillon $(x_i, y_i)_{i=1,\dots,n}$ et on souhaite ajuster un modèle $F(x)$ minimisant l'erreur quadratique. Supposons que l'on dispose d'un modèle $F(x)$ fourni par un ami. En essayant ce modèle, il semble correct mais pas parfait. Comment faire pour l'améliorer sachant que l'on s'autorise les règles suivantes:

- il est interdit de modifier les paramètres de F.
- on peut ajouter à F un modèle additionnel h "simple" et formé une nouvelle prédiction $F(x) + h(x)$

Idéalement on aimerait choisir h tel que:

$$\begin{aligned} F(x_1) + h(x_1) &= y_1 \\ F(x_2) + h(x_2) &= y_2 \\ &\dots \\ F(x_n) + h(x_n) &= y_n \end{aligned}$$

ie

$$h(x_1) = y_1 - F(x_1)$$

Peut on effectuer cela avec un arbre de régression?

On peut le faire de manière approchée, ajuster un arbre de régression aux “nouvelles données”:

$$(x_1, y_1 - F(x_1)), ..., (x_n, y_n - F(x_n))$$

ou les $y_i - F(x_i)$ sont appelés les résidus.

Si $F + h$ n'est toujours pas satisfaisant, on peut lui ajouter un nouvel arbre et ainsi de suite. Le modèle ainsi obtenu aura bien sur un meilleur ajustement aux données, mais améliore-t-on l'erreur de généralisation? (erreur sur un échantillon test indépendant).

La méthode du gradient

V signifie un espace vectoriel normé de dimension finie dans la suite.

Définition

Soit $J : V \to \mathbb{R}$ une fonctionnelle continue et $v \in V$, on dit que $w \in V - \{0\}$ est une direction de descente en v s'il existe $\rho_0 > 0$ tel que:

$$J(v + \rho w) \leq J(v), \forall \rho \in [0, \rho_0]$$

Il s'agit d'une direction de descente stricte si

$$J(v + \rho w) < J(v), \forall \rho \in [0, \rho_0]$$

Algorithme (descente de gradient)

- Fixer $u_0 \in V$
- $n \leftarrow 0$

répéter jusqu'à *critère d'arrêt*:

- trouver une direction de descente stricte $w_n \in V - \{0\}$
- choisir $\rho_n > 0$

$$u_{n+1} \leftarrow u_n + \rho_n w_n$$

- $n \leftarrow n + 1$

Exemple de critères d'arrêt:

Lorsque J est "aplatie" un exemple de critère d'arrêt est $||u_n - u_{n+1}|| \leq \varepsilon$
Lorsque J est "raide" un exemple de critère d'arrêt est $|J(u_n) - J(u_{n+1})| \leq \varepsilon$

Algorithme (méthode du gradient à pas fixe)

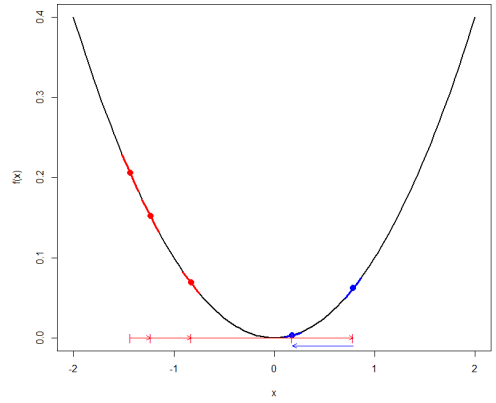
- Fixer $u_0 \in V$, $\rho > 0$
- $n \leftarrow 0$

répéter jusqu'à *critère d'arrêt*:

- $w_n \leftarrow -\Delta J(u_n)$
- si $w_n = 0$ alors retourner u_n - sinon:

$$u_{n+1} \leftarrow u_n + \rho w_n$$

- $n \leftarrow n + 1$



Boosting par descente de gradient

Revenons à notre jeu. Prenons comme fonction de perte $L(y, F(x)) = (y - F(x))^2/2$. On cherche à minimiser, en ajustant F:

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

Prenons $V = \mathbb{R}$ et posons $v_i = F(x_i)$, on a:

$$\frac{\partial J}{\partial v_i} = \frac{\partial \sum_{i=1}^{n} L(y_i, v_i)}{\partial v_i} = F(x_i) - y_i$$

Les résidus peuvent donc être vus comme des gradients négatifs.

$$r_i = y_i - F(x_i) = -\frac{\partial J}{\partial v_i}$$

Pour résumé:

$$
\begin{aligned}
F(x_i) &:= F(x_i) + h(x_i) \\
F(x_i) &:= F(x_i) + y_i - F(x_i) \\
F(x_i) &:= F(x_i) - \frac{\partial J}{\partial v_i} \\
v_i^{k+1} &:= v_i^k - \rho \frac{\partial J}{\partial v_i}(v_i^k)
\end{aligned}
$$

Dans le cas de la régression par MCO, les résidus correspondent au gradient négatif et l'ajustement h revient à mettre à jour le modèle par une méthode de descente du gradient (reste la question du choix du pas de gradient, voir plus loin).

L2 Boosting

On observe un échantillon $(x_i, y_i)_{i=1,\ldots,n}$ de réalisations de (X_i, Y_i), $X_i \in \mathbb{R}^p$ et $Y_i \in \mathbb{R}$. On suppose que les (X_i, Y_i) sont soit iid soit issus d'un processus stationnaire.

Dans ce cadre la minimisation de $E(Y - F(X))^2$ est réalisée par la régresssion $F(x) = E(Y/X = x)$. Le boosting a pour objectif d'obtenir une estimation de F à partir des observations.

On se donne pour cela une estimation de base "faible" (weak learner) appliquée de manière itérative sur les données modifiées (pseudo-réponses). L'estimation finale est une combinaison linéaire des estimations de base.

$$L_n(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2 = \frac{1}{n}\sum_{i=1}^{n} 2L(y_i, f(x_i))$$

On a:

$$\frac{\partial L(v, y)}{\partial v}(f(x)) = y - f(x) = r$$

une étape du L_2 boosting consiste à appliquer une méthode de régression pour estimer itérativement r. Procéder à 2 itérations est connu sous le nom de “twicing” (Tukey).

Notons h_n la procédure de régression de base utilisée comme apprenant faible.

plus précisément c'est une procédure qui, appliquée aux données (X_i, U_i) ou U_i dénotent les pseudos-réponses, retourne:

$$\widehat{g}(.) = \widehat{g}_{X,U}(.) = h_n((X_i, U_i)_{i=1,...,n})$$

U est un vecteur de composantes U_i.

L'apprenant faible peut être représenté dans la suite comme un opérateur:

$$\boldsymbol{H} : U \to (\widehat{g}_{X,U}(X_1), ..., \widehat{g}_{X,U}(X_n))$$

cela inclu par exemple les méthodes de régression linéaire classique, ridge, splines, noyau. . .

Algorithme (boosting L2)

- $m = 0$, $\widehat{F}_0(.) = 0$ - fixer le pas de descente $\nu \in [0, 1]$

Pour $m = 1, ..., M$:

- calcul des résidus $U_i = Y_i - \widehat{F}_{m-1}(X_i)$, $i = 1, ..., n$ - application de l'apprenant faible au vecteur des pseudos réponses de l'étape en cours $U = \widehat{f}_m(.) = \widehat{g}_{(X,U)}(.)$

$$\widehat{F}_m(.) = \widehat{F}_{m-1}(.) + \nu \widehat{f}_m(.)$$

Retourner $\widehat{F}_M$

- remarque: le choix de M est crucial.
- le pas de descente est également important, mais une règle empirique est de le choisir assez petit $\nu = 0.1$ (quitte à augmenter M au prix d'un temps de calcul plus grand).
- M et ν peuvent être estimés par VC ou par erreur OOB.

Proposition Après m itérations de boosting L_2, les valeurs estimés en chaque point d'apprentissage sont:

$$\widehat{F}_m(X) = \sum_{k=0}^{m} (H(I_n - \nu H)^k)(Y) = (I_n - (I_n - \nu H)^m)(Y)$$

L'opérateur du boosting L_2 après m opération $B_m : \mathbb{R}^n \to \mathbb{R}^n$ est donc:

$$B_m = (I_n - (I_n - \nu H)^m)(Y)$$

L'opérateur de la fonction de régression est donc:

$$B_m(Y) = (I_n - (I_n - \nu H)^m)(Y) = Y - (I_n - \nu H)^m(Y)$$

De manière heuristique, si $||I - \nu H|| < 1$ pour une certaine norme d'opérateur, B_m converge vers l'identité quand $m \to \infty$ et donc $B_m(Y)$ converge vers le vecteur des données Y. On doit donc arrêter les itérations à un nombre M pour réaliser un bon compromis biais/variance.

On choisit M soit par VC soit par un critère de pénalisation de type AIC, BIC (pour cela il faut pouvoir estimer les degrés de liberté de B_m, par ex quand c'est possible par $tr(B_m)$).

L2 Boosting linéaire

$$Y_i = \sum_{j=1}^{p} \beta_j X_i^{(j)} + \varepsilon_i$$

ε iid centrés et indépendants des $X_i \in \mathbb{R}^p$. Soit, avec les notations matricielles classiques:

$$Y = X\beta + \varepsilon$$

L'ajustement du modèle par L_2 boosting est basé sur un apprenant linéaire simple coordonnées par coordonnées.

$$\boldsymbol{H} : U \to (\widehat{g}_{X,U}(X_1), ..., \widehat{g}_{X,U}(X_n))$$

En général, la plupart des apprenants faibles emplois une méthode de sélection de variables. Si l'apprenant sélectionne une partie $\widehat{S} \in 1, ..., p$ des variables, on écrira:

$$\boldsymbol{H} : U \rightarrow (\widehat{g}_{X^{\widehat{S}},U}(X_1), ..., \widehat{g}_{X^{\widehat{S}},U}(X_n))$$

ou $\widehat{g}_{X^{\widehat{S}},U}(.)$ ne dépend que des composantes $X^{\widehat{S}}$ de X.

Revenons à un apprenant faible composé d'une régression linéaire simple. On ne sélection qu'une variable à la fois dans $\{1, ..., p\}$ en choisissant la variable explicative qui diminue le plus l'erreur quadratique d'ajustement lors d'une régression linéaire simple.

$$\widehat{g}_{X,U}(x) = \widehat{\alpha}_{\widehat{S}} + \widehat{\beta}_{\widehat{S}} X^{\widehat{S}}$$

Notons U le vecteur pseudo-réponse, supposons le centré: $\bar{U} = 0$.

$$\begin{bmatrix} \widehat{\alpha}_j \\ \widehat{\beta}_j \end{bmatrix} = (X^{(j)^T} X^{(j)})^{-1} X^{(j)^T} U \tag{1}$$

avec $X^{(j)} = [1\ X^{(j)}]$

et $\widehat{S} = j = \text{argmin}_q ||U - \alpha_q - \widehat{\beta}_q X^{(q)}||^2 = \text{argmax}_q \text{corr}(U, X^{(q)})$

- on ajuste donc une régression linéaire simple avec une seule variable explicative sélectionnée, la plus corrélée à U.
- la condition $\bar{U} = 0$ est sans conséquence pour le boosting car on centre toujours la réponse avant.
- l'apprenant peut se simplifier si l'on centre également les covariables, on ajuste alors des régressions simples passant par l'origine

En pratique on utilise la fonction `glmboost` du package `mboost`.

glmboost {mboost} R Documentation

Gradient Boosting with Component-wise Linear Models

Description

Gradient boosting for optimizing arbitrary loss functions where component-wise linear models are utilized as base-learners.

Usage

```
## S3 method for class 'formula'
glmboost(formula, data = list(), weights = NULL,
         na.action = na.pass, contrasts.arg = NULL,
         center = TRUE, control = boost_control(), ...)
## S3 method for class 'matrix'
glmboost(x, y, center = TRUE, weights = NULL,
         na.action = na.pass, control = boost_control(), ...)
## Default S3 method:
glmboost(x, ...)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`weights`	an optional vector of weights to be used in the fitting process.
`na.action`	a function which indicates what should happen when the data contain `NA`s.
`contrasts.arg`	a list, whose entries are contrasts suitable for input to the `contrasts` replacement function and whose names are the names of columns of `data` containing factors. See `model.matrix.default`.
`center`	logical indicating of the predictor variables are centered before fitting.
`control`	a list of parameters controlling the algorithm.
`x`	design matrix. Sparse matrices of class `Matrix` can be used as well.
`y`	vector of responses.
`...`	additional arguments passed to `mboost_fit`, including `weights`, `offset`, `family` and `control`. For default values see `mboost_fit`.

Avec les notations adoptées, B_M l'opérateur de boosting L_2 coordonnée par coordonnée s'écrit donc, à l'itération m:

$$B_m = I - (I - \nu H_{\widehat{S}_m})...((I - \nu H_{\widehat{S}_1}))$$

Lorsque les apprenants faibles sont linéaires, le boosting permet une représentation approximativement linéaire, la seule non linéarité provenant de la sélection $\widehat{S}$. Bühlmann (2006) propose donc d'estimer les degrés de liberté du modèle de manière approchée par:

$$\text{trace}(B_m) \approx \text{trace}(I - (I - \nu H_{\widehat{S}_m})...((I - \nu H_{\widehat{S}_1})))$$

Il existe des variantes ou l'apprenant exploite un bloc de variables (au lieu d'une).

L2 Boosting à base de splines

L'utilisation de fonctions splines cubiques de lissage (noeuds splines coïncide avec les données, pénalité basée sur l'int. de la dérivée seconde au carré) comme apprenant faible est proposée par Bühlmann and Yu (2003). On peut également utilisé des P-splines Eilers and Marx (1996) ou des B-splines comme dans mgcv. Les P-splines ont été étudiées dans le cadre du boosting dans Schmid and Hothorn (2008).

Considérons le problème de la régression unidimensionnelle:

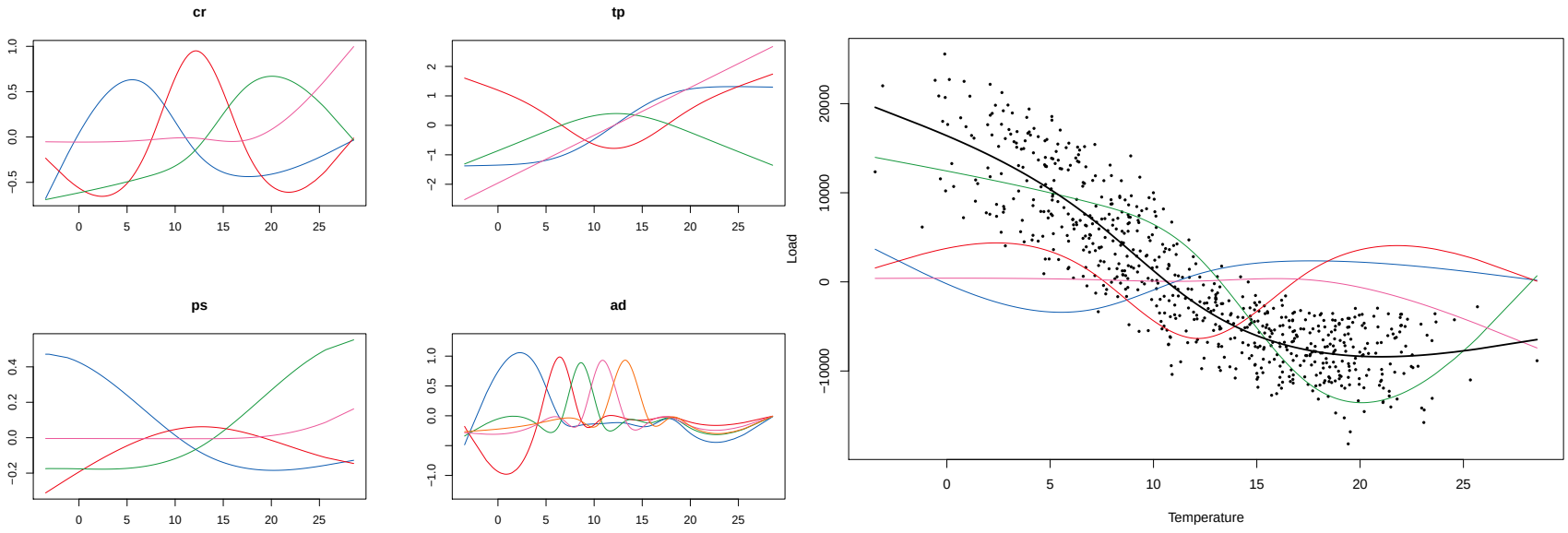
$$Y_i = f(x_i) + \varepsilon_i \; i = 1, ..., n$$

ou f est régulière. Les deux composantes des P-splines sont les B-splines et les pénalisations associées.

On peut approcher f par:

$$f(x) = \sum_{j=1}^{J} \beta_j B_j(x, \delta) = B(x)^T \beta$$

où $B_j(x, \delta)$ est la jème fonction B-spline de degré δ, $B(x) = (B_1(x), ..., B_J(x))^T$. Les fonctions de bases B-splines sont définies par un ensemble de noeud intérieurs et des noeuds additionnels sur les bords.


cr
tp
ps
ad
Load
Temperature

Pour régulariser l'estimation, comme bien souvent on a choisit un nombre important de noeuds, on minimise:

$$Q(\beta) = ||Y - B\beta||^2 + \lambda J(\beta, d)$$

où $J(\beta, d) = \sum_{j=d+1}^{J} (\Delta^d \beta_j)^2$, Δ^d étant l'opérateur de différences finies d'ordre d.

L'apprenant P-spline est alors (à nombre de noeuds fixés, degré choisi, ordre de pénalisation d fixé):

$$H_P(U) = B^T (B^T B + \lambda K)^{-1} B^T U = C_\lambda U$$

où λ est la pénalité, K l'opérateur de pénalisation associé à $J(\beta, d)$.

La fonction `gamboost` du package `mboost` implémente cet ajustement par boosting avec apprenant faible des P-splines.

Gradient Boosting with Smooth Components

Description

Gradient boosting for optimizing arbitrary loss functions, where component-wise smoothing procedures are utilized as base-learners.

Usage

```
gamboost(formula, data = list(),
         baselearner = c("bbs", "bols", "btree", "bss", "bns"),
         dfbase = 4, ...)
```

Arguments

`formula`	a symbolic description of the model to be fit.
`data`	a data frame containing the variables in the model.
`baselearner`	a character specifying the component-wise base learner to be used: `bbs` means P-splines with a B-spline basis (see Schmid and Hothorn 2008), `bols` linear models and `btree` boosts stumps. `bss` and `bns` are deprecated. Component-wise smoothing splines have been considered in Buehlmann and Yu (2003) and Schmid and Hothorn (2008) investigate P-splines with a B-spline basis. Kneib, Hothorn and Tutz (2009) also utilize P-splines with a B-spline basis, supplement them with their bivariate tensor product version to estimate interaction surfaces and spatial effects and also consider random effects base learners.
`dfbase`	an integer vector giving the degrees of freedom for the smoothing spline, either globally for all variables (when its length is one) or separately for each single covariate.
`...`	additional arguments passed to `mboost_fit`, including `weights`, `offset`, `family` and `control`. For default values see `mboost_fit`.

Details

A (generalized) additive model is fitted using a boosting algorithm based on component-wise univariate base-learners. The base-learners can either be specified via the `formula` object or via the `baselearner` argument (see `bbs` for an example). If the base-learners specified in `formula` differ from `baselearner`, the latter argument will be ignored. Furthermore, two additional base-learners can be specified in `formula`: `bspatial` for bivariate tensor product penalized splines and `brandom` for random effects.

Adaboost

On dispose d'un échantillon de données $(x_i, y_i)_{1 \leq i \leq n}$, $X \in \mathbb{R}^p$, $Y \in \{-1, 1\}$

et d'un algorithme de classification $h : \mathbb{R}^p \to \{-1, 1\}$

Le but est de trouver un classifieur convergent vers h^* le classifieur qui minimise le risque associé à la perte exponentielle:

$$h^* = \underset{h}{\operatorname{argmin}} E[\exp(-Yh(X)]$$

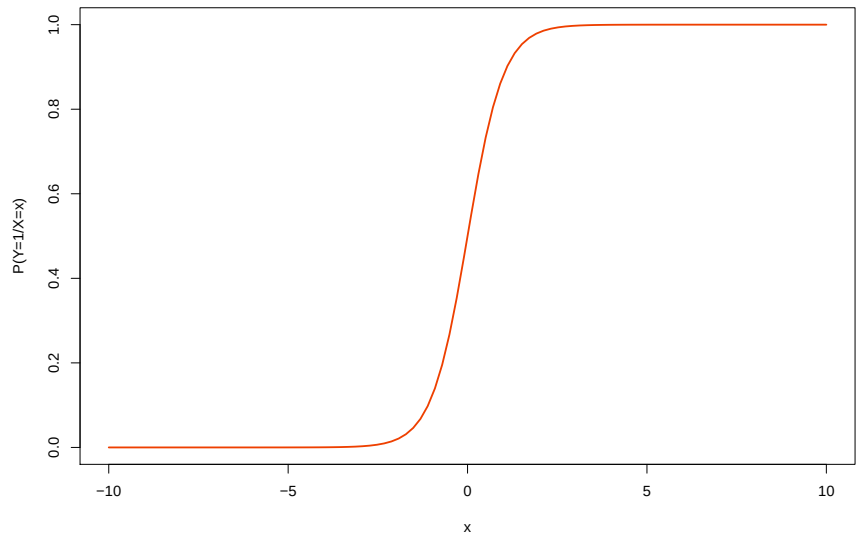
qui a pour optimum (Friedman et al. 2000):

$$f(x) = \frac{1}{2} \log \frac{P(Y = 1/X = x)}{P(Y = -1/X = x)}$$

et donc le classifieur associé $h^*(x) = \operatorname{sign}(f(x))$ (revient à choisir 1 si $P(Y = 1/X = x) > 1/2$)

ce qui équivaut à modéliser:

$$P(Y = 1/X = x) = 1/(1 + \exp(-2h^*(x)))$$



On dispose d'un algorithme de classification $h : \mathbb{R}^p \to \{-1, 1\}$. Le but est de trouver une suite de classifieurs h_t et des poids α_t telle que le prédicteur:

$$f_t(x) = f_{t-1}(x) + \alpha_t h_t(x)$$

réalise une faible erreur (ici erreur de classification).

On prend comme fonction de perte $\phi(y, \widehat{y}) = \exp(-y\widehat{y})$ et on cherche à minimiser:

$$E_n = \sum_{i=1}^{N} \exp(-y_i f_t(x_i))$$

$$E_n = \sum \exp(-y_i f_{t-1}(x_i)) \exp(-\alpha_t y_i h_t(x_i))$$

en notant $\omega_i^t = \exp(-y_i f_{t-1}(x_i))$:

$$E_n = \sum_{y_i = h_t(x_i)} \omega_i^t \exp(-\alpha_t) + \sum_{y_i \neq h_t(x_i)} \omega_i^t \exp(\alpha_t)$$

$$E_n = \sum_{i=1}^{n} \omega_i^t \exp(-\alpha_t) + \sum_{y_i \neq h_t(x_i)} \omega_i^t [\exp(\alpha_t) - \exp(-\alpha_t)]$$

minimiser E_n revient à minimiser le 2e terme. En supposant α_t indé. de h_t, celà revient pour déterminer h_t à minimiser:

$\sum_{y_i \neq h_t(x_i)} \omega_i^t$, erreur de classification pondérée (poids aux données pour lesquelles f_{t-1} s'est trompé).

le prédicteur boosting est donc obtenu séquentiellement, en zoomant à chaque étape sur les données mal prévues à l'étape précédente

le α_t optimal est ensuite obtenu en minimisant E_n:

$$\partial E_n/\partial \alpha_t = \sum_{y_i \neq h_t(x_i)} \omega_i^t \exp(\alpha_t) - \sum_{y_i = h_t(x_i)} \omega_i^t \exp(-\alpha_t)$$

s'annule en:

$$\alpha_t = \frac{1}{2} \log \frac{\sum_{y_i = h_t(x_i)} \omega_i^t}{\sum_{y_i \neq h_t(x_i)} \omega_i^t}$$

Soit, en notant $\varepsilon = P_t(Y \neq h_t(X))$ (proba. empirique sous la loi w_t)

$$\alpha_t = \frac{1}{2} \log \frac{1-\varepsilon}{\varepsilon}$$

Algorithme (Adaboost)

- initialiser les poids des observations à $\omega_i = 1/N$
- pour t de 1 à T:
  - estimer un classifieur h_t sur les données pondérés par ω_i
  - calculer:

$$\varepsilon_t = \sum_{i=1}^{n} \omega_i 1(y_i \neq h_t(x_i)) / \sum_{i=1}^{n} \omega_i$$

  - puis $\alpha_t = \log(\frac{1-\varepsilon_t}{\varepsilon_t})$
  - mettre à jour les poids:

$$\omega_i \leftarrow \omega_i \exp(\alpha_t 1(y_i \neq h_t(x_i))$$

- calculer le classifieur final:

$$f_T(x) = \text{sign}[\sum_{t=1}^{T} \alpha_t h_t(x)]$$

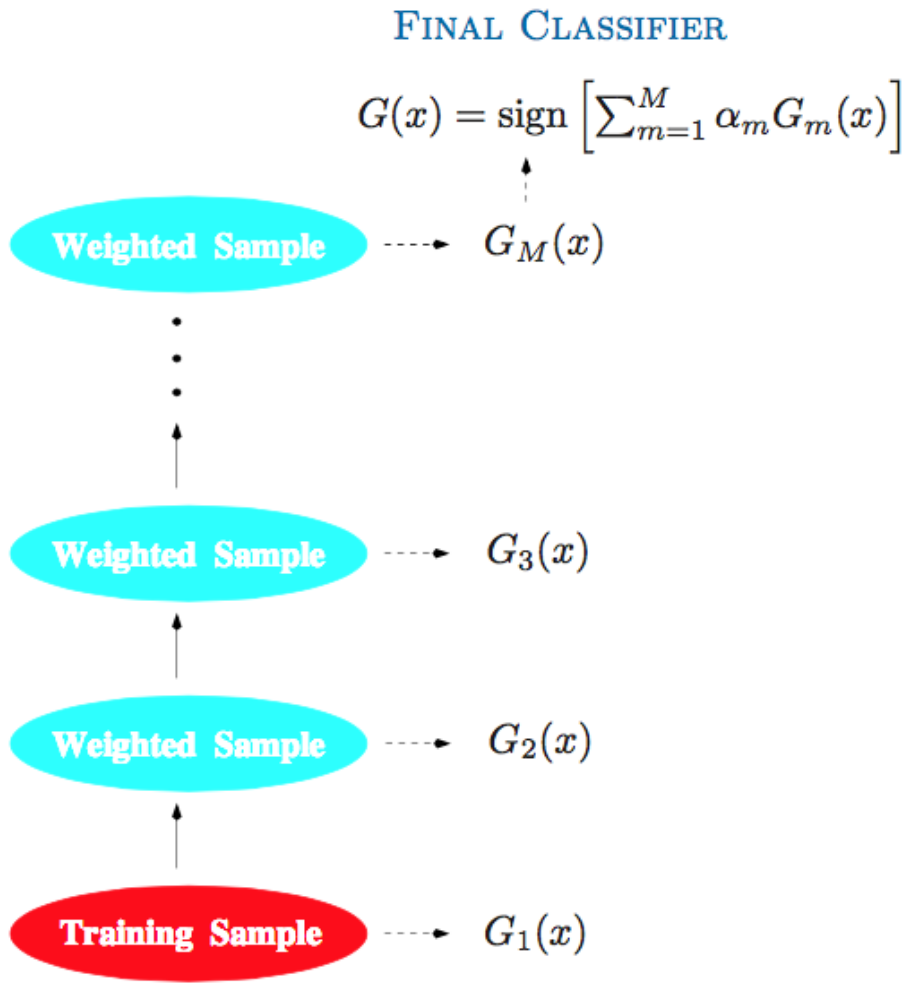


FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Retour sur le gradient boosting: choix du pas de gradient

Dans l'algorithme de GB, zoomons sur les étapes:

- estimer un modèle (weak) sur -gradient
- estimer le nombre d'itérations

une idée intéressante pour réaliser ces 2 étapes conjointement est de partitionner les données en 2 échantillons indépendants:

- tirer au hasard un échantillon S_p de données parmis n estimer un modèle (weak) sur le gradient
- estimer l'erreur oob à l'étape m de l'algorithme

ainsi, l'on réalise une pierre 2 coups: randomisation (effet bagging) & on dispode d'une erreur oob pour éviter l'overfitting.

Implémentations

- package `gbm`: gradient boosting avec arbre CART comme week-learners
- package `mboost`: gradient boosting avec penalized regression splines d'une seule variable comme weak learners, B-splines, modèles linéaires, stumps
- package `xgboost`: gradient boosting avec arbre CART ou modèles linéaires comme week-learners, package de référence pour beaucoup de compétitions de ML, implémentation très efficace
- package `catboost`: https://github.com/catboost gradient boosting avec arbre CART, passage à l'échelle sur GPU possible
- package `lightgbm`: gradient boosting avec arbre CART , GOSS (Gradient Based One Side Sampling) (inclusion des obs avec fort gradient et tirage aléatoire des obs avec faible gradient pour accélerer le calcul des arbres)

`catboost` et `lightgbm` effectuent des transformations des variables continues en variables discrètes pour gagner en temps de calcul/compression des données. Ils sont reconnus pour leur implémentation efficace.

Calibration

Il s'agit ici d'une démarche purement empirique. Dans le cas ou l'apprenant faible est un arbre, voilà comment il est conseillé de procéder:

- choisir un pas de gradient faible, entre 0.05 et 0.2
- calculer le nombre d'arbre optimal (nombre d'itérations)
- à pas de gradient et nombre d'arbre fixé, calibrer les paramètres des arbres (cf CART)
- si possible baisser ensuite le pas de gradient et augmenter le nombre d'arbre

References

Breiman, Leo, and others. 1998. "Arcing Classifier (with Discussion and a Rejoinder by the Author)." *The Annals of Statistics* 26 (3). Institute of Mathematical Statistics: 801–49.

Bühlmann, Peter, and Bin Yu. 2003. "Boosting with the L 2 Loss: Regression and Classification." *Journal of the American Statistical Association* 98 (462). Taylor & Francis: 324–39.

Eilers, Paul HC, and Brian D Marx. 1996. "Flexible Smoothing with B-Splines and Penalties." *Statistical Science*. JSTOR, 89–102.

Freund, Yoav, Robert E Schapire, and others. 1996. "Experiments with a New Boosting Algorithm." In *Icml*, 96:148–56. Citeseer.

Freund, Yoav, Robert Schapire, and Naoki Abe. 1999. "A Short Introduction to Boosting." *Journal-Japanese Society for Artificial Intelligence* 14 (771-780). JAPANESE SOC ARTIFICIAL INTELL: 1612.

Schapire, Robert E. 1990. "The Strength of Weak Learnability." *Machine Learning* 5 (2). Springer: 197–227.

Schmid, Matthias, and Torsten Hothorn. 2008. "Boosting Additive Models Using Component-Wise P-Splines." *Computational Statistics & Data Analysis* 53 (2). Elsevier: 298–311.