Apprentissage, réseaux de neurones et modèles graphiques (RCP209)

Méthodes d'agrégation

Marin FERECATU

(prenom.nom@cnam.fr)

http://cedric.cnam.fr/vertigo/Cours/ml2/

Département Informatique

Conservatoire National des Arts & Métiers, Paris, France

Plan du cours

Objectif

"La raison d'être des statistiques, c'est de vous donner raison." — Abe Burrows

Méthodes d'agrégations :

- Bagging
- Forets Aléatoires
- Boosting

Plan du cours

Avantages et défauts des arbres de décision

Avantages :

- Modèle "white box" : le résultat est facile à conceptualiser et à visualiser
- Ils nécessitent peu de préparation de données (e.g. normalisation, etc.)
- Le cout d'utilisation des arbres est logarithmique
- Capables d'utiliser des données catégorielles et continues
- Capables de gérer des problèmes multi-classe
- Bon comportement par rapport aux outliers
- Gèrent bien les données manquantes

Avantages et défauts des arbres de décision

Problèmes :

- Parfois les arbres générés ne sont pas équilibrés (ce qui implique que le temps de parcours n'est plus logarithmique). Il est donc recommandé d'équilibrer la base de donnée avant la construction, pour éviter qu'il y a une classe dominante (en terme de nombre d'exemples d'apprentissage)
- Sur-apprentissage : parfois les arbres générés sont trop complexes et généralisent mal (solution : élagage, le contrôle de la profondeur de l'arbre et de la taille des feuilles)
- Ils sont **instables** : des changements légères dans les données produisent des arbres très différents. Changements des nœuds proches de la racine affectent beaucoup l'arbre résultant. Ce sont des **estimateurs de variance élevée**.

Estimateurs de variance élevée

Estimateurs de variance élevée :

- Réduction de variance
- Moyenne des estimateurs, calculés sur des données légèrement différentes

Bagging et **Random Forests** : utiliser le hasard pour améliorer les performances des algorithmes de base (arbres de décision CART).

Algorithmes proposés par Breiman, et beaucoup étudiés récemment :

- L. Breiman. *Bagging predictors*, Machine Learning, 24(2), 1996.
- L. Breiman. *Random forests*, Machine Learning, 45, 2001.

Plan du cours

Bagging

Base d'apprentissage :

- Attributs : $A_1, ..., A_p$, classe : C
- Données d'apprentissage : $(x_i, y_i), x_i \in R^p, y_i \in R, i = 1, ..., N$
- y_i peuvent être des valeurs continues ou discrètes (étiquettes des classes)
- $x_i = (a_1^{(i)}, ..., a_p^{(i)})$

On considère $G(x)$ un modèle de prédiction appris sur un échantillon de données $z = \{(x_i, y_i)\}_{i=1}^n$ (e.g. arbre de décision CART)

Bagging

Bagging (Breiman, 1996) :

- On tire au hasard dans la base d'apprentissage B échantillons avec remise $z_i, i = 1, \ldots, B$ (chaque échantillon ayant n points) — appelés échantillons "bootstrap"
- Pour chaque échantillon i on calcule le modèle $G_i(x)$
- Régression : agrégation par la moyenne $G(x) = \frac{1}{B} \sum_{i=1}^{B} G_i(x)$
- Classification : agrégation par vote $G(x) = \text{Vote majoritaire}\,(G_1(x), \ldots, G_B(x))$

Bagging

C'est l'estimateur moyenne qui aide a réduire la variance :

- $X_1, X_2, \ldots, X_n$ variables aléatoires i.i.d. de moyenne μ et variance σ^2
- $\frac{1}{n}(X_1 + X_2 + \cdots + X_n)$ est de variance σ^2/n

Critère performance et calcul de B : l'erreur OOB (Out Of Bag).

- Pour chaque échantillon x_k on construit un prédicteur (de type forêt aléatoire) en utilisant seulement les arbres G_i tel que $x_i \notin z_i$ (donc x_i ne fait pas partie de l'échantillon de bootstrap de G_i).
- Agrégation des erreurs OOB obtenues pour chaque échantillon (par la moyenne par exemple).
- On choisi B ou l'erreur se stabilise et ne descend plus.
- Si B est grand, l'erreur OOB est presque identique à celle obtenue par validation croisée de type un contre tous (N-Fold)

Bagging

Défaut du bagging :

- Les estimateurs G_i ne sont pas en réalité indépendants.
- G_i sont calculés sur des échantillons qui se recouvrent fortement (tirage avec remise), et donc ils sont corrélés.

$X_1, X_2, \ldots, X_B$ variables aléatoires i.d. (mais pas indépendantes) de moyenne μ, variance σ^2 et corrélation $\rho = Corr(X_i, X_j), \forall i \neq j$.
Alors $Y = \frac{1}{B}(X_1 + X_2 + \cdots + X_B)$ est de variance :

$$Var(Y) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Quand B est grand le 2eme terme est négligeable mais le 1er non.
L'idée des forets aléatoires est de baisser la corrélation entre les G_i a l'aide d'une étape supplémentaire de randomisation.

Plan du cours

Forêts aléatoires

Forêts aléatoires :

- Amélioration du bagging pour les arbres de décision CART
- Objectif : rendre les arbres utilisés plus indépendants (moins corrélés)
- Bons résultats surtout en grande dimension
- Très simple à mettre en œuvre
- Peu de paramètres

Forêts aléatoires

Forêts aléatoires (Breiman, 2001) :

- On tire au hasard dans la base d'apprentissage B échantillons avec remise $z_i, i = 1, \ldots, B$ (chaque échantillon ayant n points)
- **Pour chaque échantillon i on construit un arbre CART $G_i(x)$ selon un algorithme légèrement modifié : à chaque fois qu'un noeud doit être coupé (étape "split") on tire au hasard une partie des attributs (q parmi les p attributs) et on choisit le meilleur découpage dans ce sous-ensemble.**.
- Régression : agrégation par la moyenne $G(x) = \frac{1}{B} \sum_{i=1}^{B} G_i(x)$
- Classification : agrégation par vote $G(x) = \text{Vote majoritaire}\,(G_1(x), \ldots, G_B(x))$

Les arbres sont moins corrélés car ils sont appris sur des ensembles d'attributs qui se recouvre pas beaucoup.

En général : $q = \sqrt{p}$

Forêts aléatoires

Forêts aléatoires (Breiman, 2001) :

- On se limite en général à des arbres pas très profonds (pour le Bagging il faut des arbres profonds pour limiter leur corrélation : mais les arbres très profonds souffrent de sur-apprentissage)
- Chaque arbre est petit donc moins performant, mais l'agrégation compense pour ce manquement (chaque attribut se retrouve typiquement dans plusieurs arbres)
- Comme pour le Bagging on utilise l'erreur OOB pour prévenir le sur-apprentissage

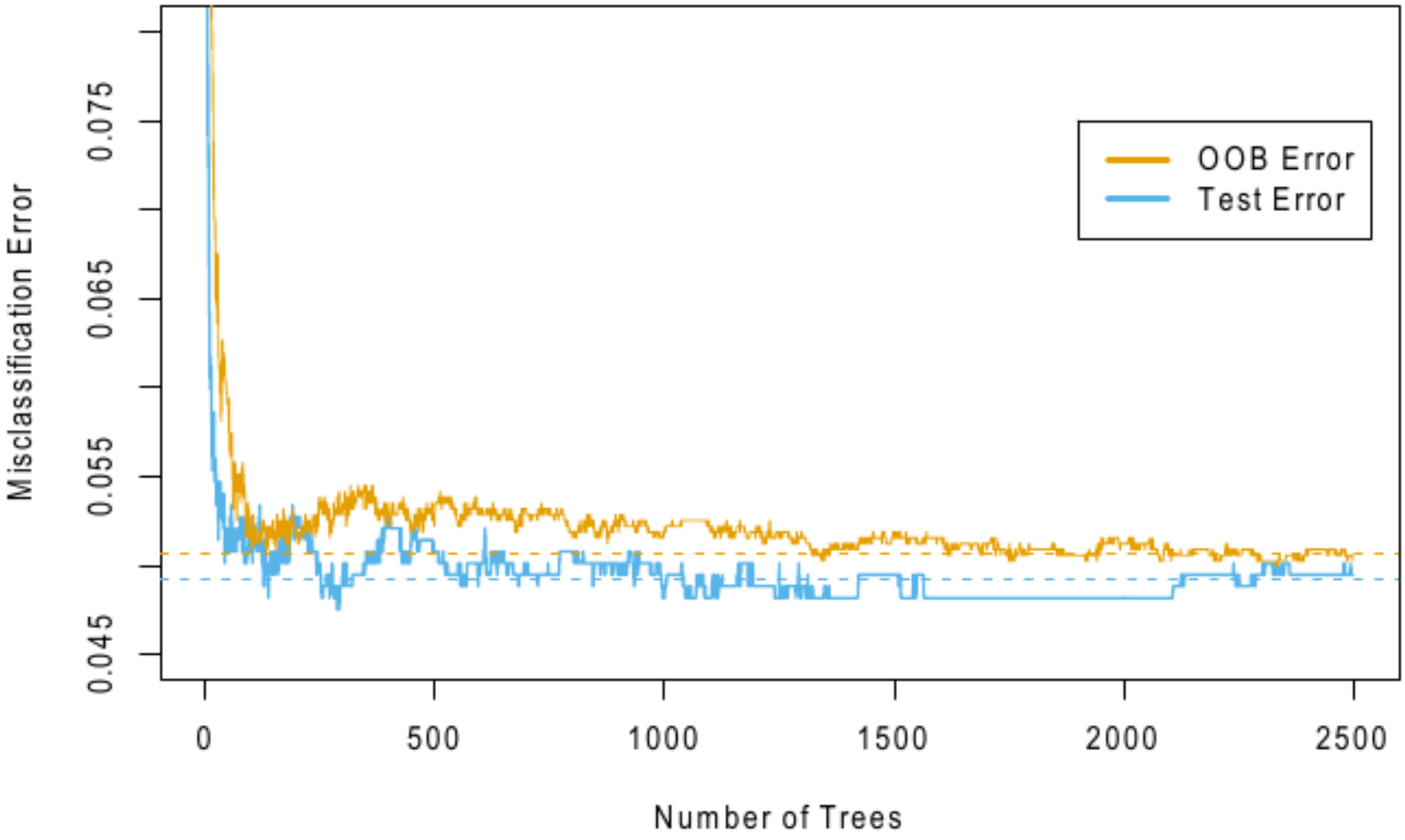
Forêts aléatoires

Paramètres (valeurs par défaut) :

- Classification : $q = \sqrt{p}$, taille nœud minimale 1 ;
- Régression : $q = p/3$, taille nœud minimale 5.

En pratique les valeurs "idéales" dépendent beaucoup de la base (et il faut les trouver par cross-validation.)

Forêts aléatoires



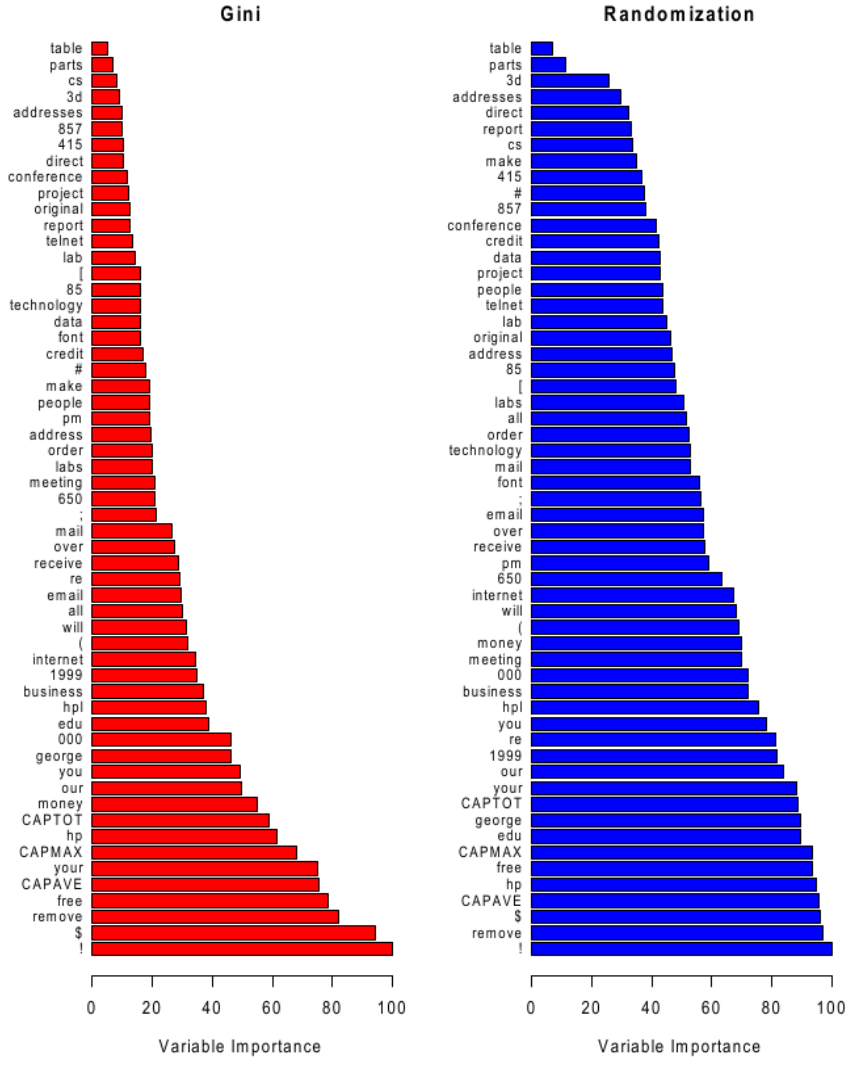
OOB vs erreur de test sur la base "Spambase".

Forêts aléatoires

L'importance des attributs :

- Gini : Le changement dans l'impureté (ou gain d'information) dans chaque noeud où l'attribut est testé, cumulé sur tous les arbres de la foret.
- Erreur OOB : après avoir construit l'arbre no b, tous les échantillons OOB sont évalués et l'erreur mesurée. Ensuite on permute aléatoirement les valeurs sur chaque attribut j dans les échantillons OOB et on mesure le taux d'erreur à nouveau. La valeur finale est la dégradation moyenne (changement du taux d'erreurs) sur tous les arbres ($b = 1..B$).

Forêts aléatoires



L'importance des attributs : Gini (gauche) vs OOB error (droite).

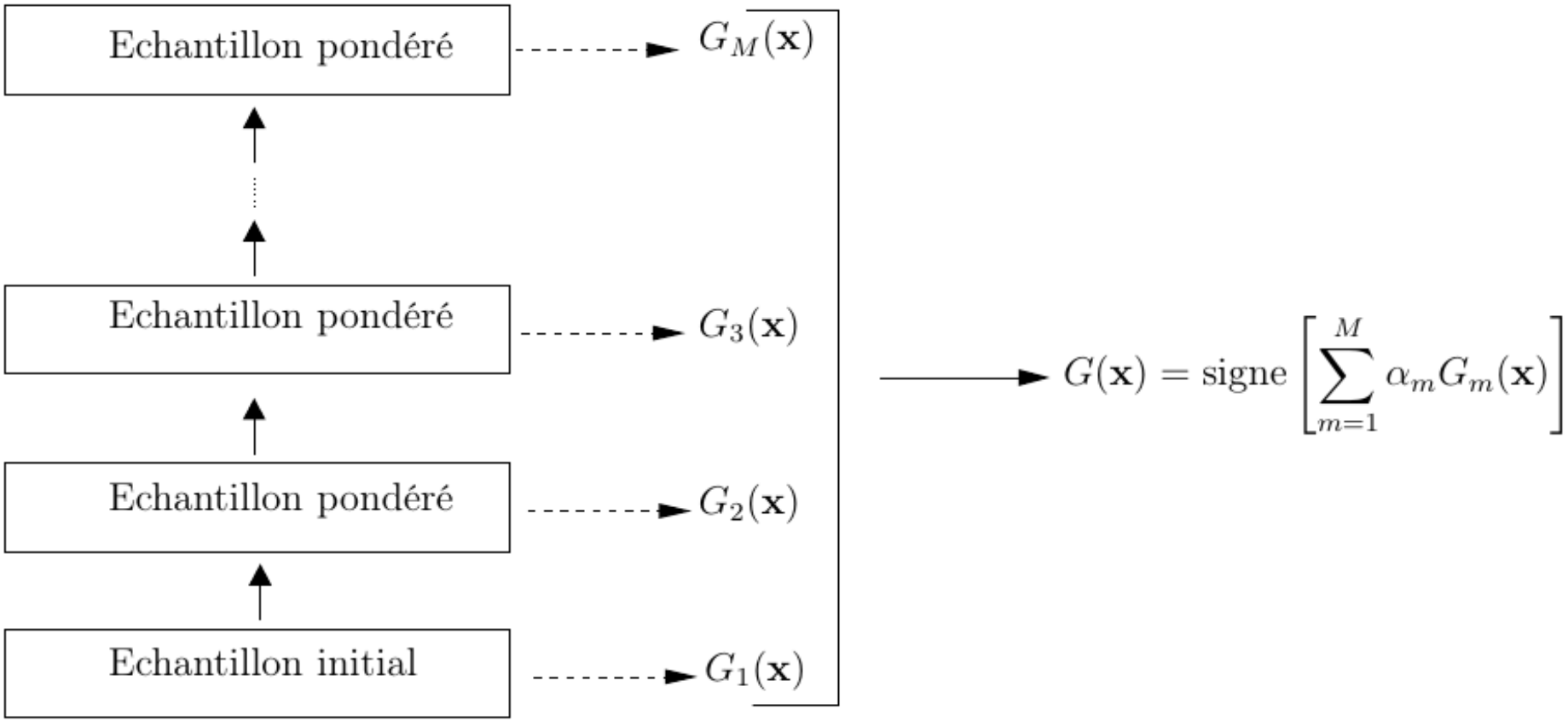
Plan du cours

Boosting

Boosting :

- Combine les sorties de plusieurs classifieurs faibles (weak learners) pour obtenir un résultat plus fort.
- Classifieur faible : un comportement de base meilleur que l'aléatoire (taux d'erreurs sous 0.5 pour une classification binaire)
- Données d'apprentissage : $(x_1, y_1), \ldots, (x_n, y_n)$
- Une famille $\mathcal{G}$ de classifieurs faibles

Boosting


Echantillon pondéré
$G_M(\mathbf{x})$
Echantillon pondéré
$G_3(\mathbf{x})$
Echantillon pondéré
$G_2(\mathbf{x})$
Echantillon initial
$G_1(\mathbf{x})$
$G(\mathbf{x}) = \text{signe}\left[\sum_{m=1}^{M} \alpha_m G_m(\mathbf{x})\right]$

Algorithme AdaBoost

1. Initialiser les poids $w_i = 1/n, i = 1..n$

2. Pour $m = 1$ **à** M **:**

A : Choisir le classifieur $G_m(\cdot) \in \mathcal{G}$ qui minimise l'erreur pondérée par les poids $w_1, \ldots, w_n$ sur la base d'apprentissage :

$$G_m(\cdot) = \arg \min_{G_i \in G} \sum_{i=1}^{n} w_i I(y_i \neq G_i(x_i))$$

La fonction unité $I(\cdots)$ aide à compter le nombre d'erreurs, c.t.d. $I(y_i \neq G_i(x_i))$ retourne 1 si $y_i \neq G_i(x_i)$ et 0 le cas contraire.

B : Calculer le taux d'erreurs :

$$e_m = \frac{\sum_{i=1}^{n} w_i I(y_i \neq G_i(x_i))}{\sum_{i=1}^{n} w_i}$$

Algorithme AdaBoost

C : Calculer le poids α_m du classifieur $G_m(\cdot)$:

$$\alpha_m = \log\left(\frac{1 - e_m}{e_m}\right)$$

Si le taux d'erreurs e_m est petit on est devant un bon classifieur, son poids α_m dans le mélange final sera donc grand. Au contraire, un classifieur qui fait beaucoup d'erreurs, aura moins d'impact (α_m petit).

D : Réajuster les poids :

$$w_i = w_i \exp(\alpha_m I(y_i \neq G_i(x_i))), i = 1 \ldots, n$$

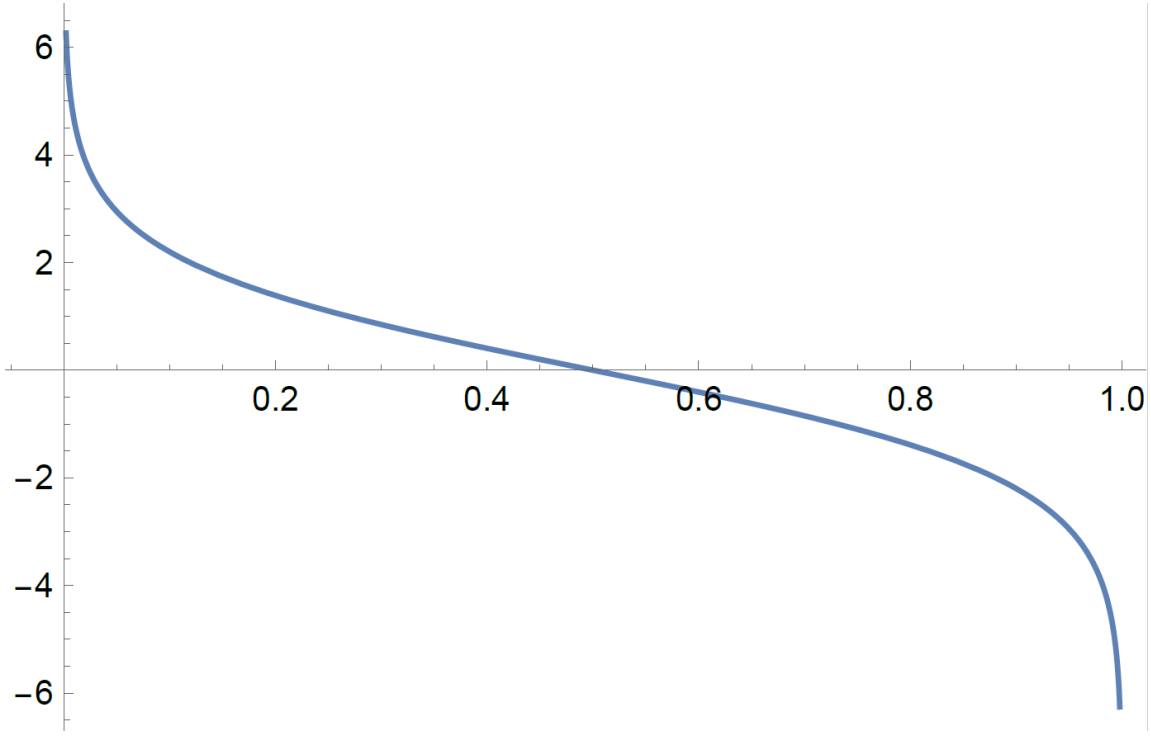
Si x_i est bien classé par G_i (donc $G_i(x_i) = y_i$) alors w_i reste inchangé (car $\exp(0) = 1$). Dans le cas contraire, x_i est un cas difficile et son poids est augmenté, d'autant plus si α_m est grand (ce qui signifie qu'un bon classifieurs n'arrive pas a bien classé cet échantillon).

3. Le classifieur final est :

$$G(x) = \text{signe}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

Algorithme AdaBoost

Courbe α_m vs. e_m



On voit que l'erreur e_m doit être inférieure à 0.5, sinon α_m devient négatif, d'où la nécessité que les classifieurs faibles soit un peu meilleurs qu'un choix aléatoire.

Boosting comme modèle additif

Modèle additif : somme de fonctions simples (weak learners) de paramètres γ_m :

$$f(x) = \sum_{i=1}^{M} \beta_m b(x; \gamma_m)$$

Ces modèles sont appris en minimisant une fonction de perte $L(\cdot)$ moyennée sur les exemples d'apprentissage :

$$\min_{\beta_1, \gamma_1, \ldots, \beta_M, \gamma_M} \sum_{i=1}^{n} L(y_i, \sum_{i=1}^{m} \beta_m b(x_i; \gamma_m))$$

Optimisation sur $2M$ variables : problème complexe (M de l'ordre des centaines).
Solution : minimiser séquentiellement par rapport a une seule paire de variables β, γ
Stratégie appelée Forward Stagewise Additive Modeling (FSAM).

Boosting comme modèle additif

Forward Stagewise Additive Modeling :

1. Initialiser $f_0(x) = 0$
2. Pour $m = 1$ à M :

– $(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + \beta b(x, \gamma))$

– Mettre $f_m(x) = f_{m-1}(x_i) + \beta_m b(x, \gamma_m)$

A chaque itération m, les paramètres β_m et γ_m sont choisis tel que le modèle courant $f_m(\cdot)$ atteint une perte minimale sur l'ensemble d'apprentissage.

On peut montrer que AdaBoost est équivalent avec FSAM avec une fonction de perte exponentielle : $L(y, f(x)) = \exp(-yf(x))$.

Voir les notes de cours.

Gradient Boosting

– FSAM optimise itérativement la perte totale (Loss) – souvent un processus lent

– Gradient boosting : ajouter une nouvelle composante qui diminue étape par étape la fonction de perte.

– Similaire à une descente de gradient $F(\rho)$: $\rho = \rho - \eta \nabla F$ A l'itération m : cherche $h_m(x)$ tel que $f_{m-1}(x_i) + h_m(x_i) = y_i (i = 1, \ldots, n)$.

On veut que $h_m(x_i)$ approxime bien les résidus $y_i - f_{m-1}(x_i)$: le principe est de combler les différences entre les prédictions courantes et les valeurs y_i.

Ensuite on pose $f_m(x) = f_{m-1}(x) + h_m(x)$ et on passe à l'itération suivante.

Supposons qu'on utilise une perte quadratique :

$$L(y, f_{m-1}(x)) = \frac{1}{2} \sum_{i=1}^{n} (y_i - f_{m-1}(x_i))^2$$

Calculons les dérivées par rapport à f_{m-1} :

$$\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)} = -(y_i - f_{m-1}(x_i))$$

Gradient Boosting

Perte quadratique : les résidus sont donnés par l'inverse du gradient de la fonction perte :

$$r_{mi} = y_i - f_{m-1}(x_i) = -\frac{\partial L(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)}, i = 1, \ldots, n$$

Apprendre un classifieur faible $h_m(x)$ sur les résidus $\{x_i, r_{mi}\}_{i=1}^n$ et mettre à jour le modèle de mélange :

$$f_m(x) = f_{m-1}(x) + \gamma_m h_m(x)$$

Le paramètre γ_m est donné par :

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + \gamma h_m(x_i))$$

Le procédé cherche donc itérativement le modèle $f(x)$ par une descente de gradient qui minimise la fonction de perte $L(\cdot)$.

Gradient Boosting

L'algorithme commence avec une fonction constante :

$$f_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

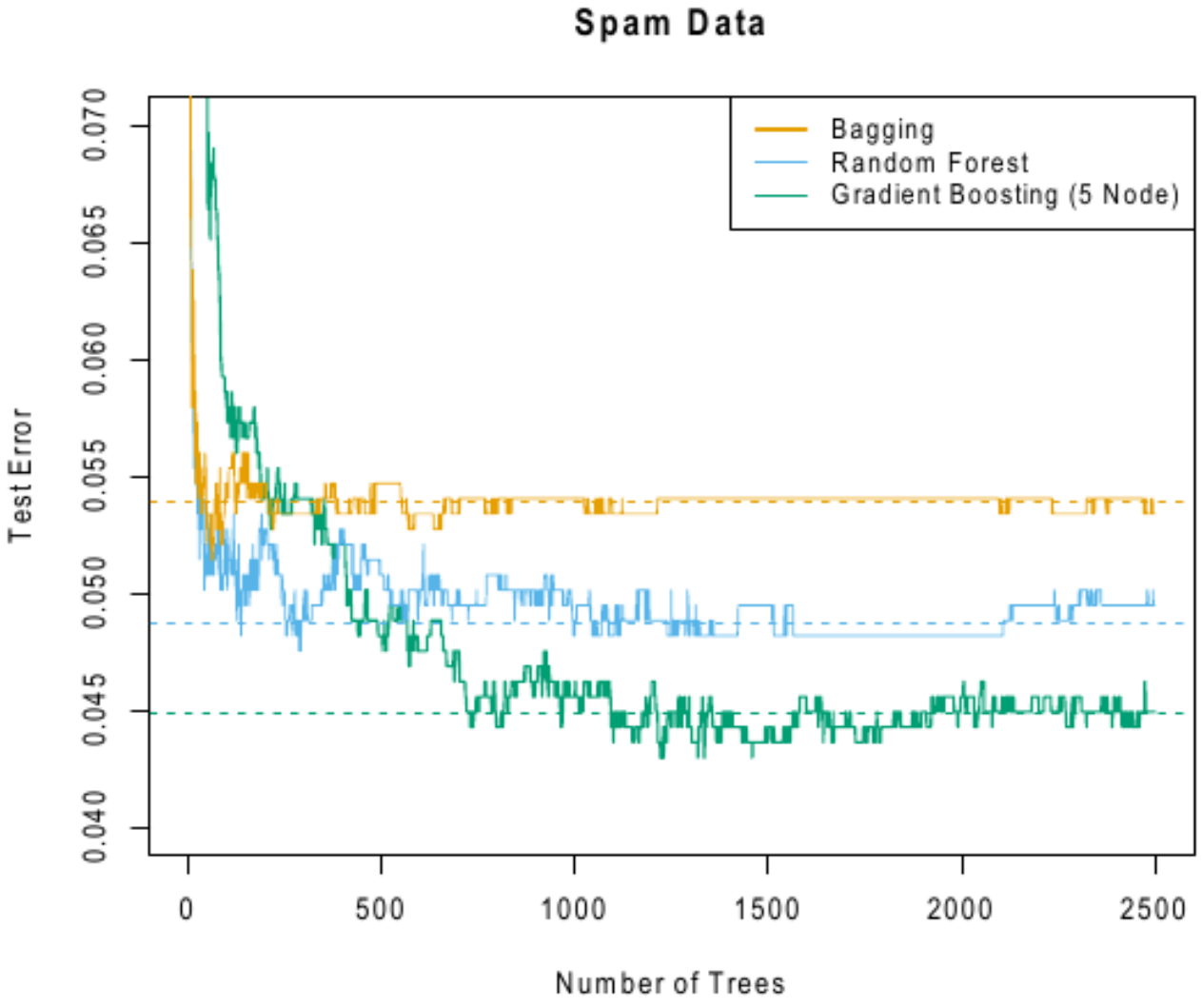
et continue pendant M itérations.

Fonction de perte pas quadratique : les valeurs r_{mi} ne sont plus les résidus par rapport à la cible.

Mais le principe reste le même : l'apprenant faible h_m "tire" le modèle $f_{m-1}(\cdot)$ dans une direction qui minimise la fonction objectif $L(\cdots)$ (puisque la mise a jour se déroule dans le sens inverse du gradient)

Dans ce cas les valeurs r_{mi} s'appellent **pseudo-résidus**.

Boosting


Spam Data
Bagging
Random Forest
Gradient Boosting (5 Node)
Test Error
0.040
0.045
0.050
0.055
0.060
0.065
0.070
0
500
1000
1500
2000
2500
Number of Trees

Références

Livres et articles :

- L. Breiman. *Bagging predictors*, Machine Learning, 24(2), 1996.
- L. Breiman. *Random forests*, Machine Learning, 45, 2001.
- Hastie, Tibshirani, Friedman, *The elements of statistical learning : Data mining, inference, and prediction*, New York, Springer Verlag, 2006
- Laurent Rouvière, *Introduction aux méthodes d'agrégation : boosting, bagging et forêts aléatoires*, polycopié cours, (https ://perso.univ-rennes2.fr/laurent.rouviere)