

## 6 Agrégation de modèles par boosting

## LES GRADIENT BOOSTING MACHINE (GBM) [SF12]

Nous avons vu un exemple de combinaison de modèles basé sur une stratégie aléatoire (bagging : par ex. avec forêts aléatoires).

→ L'enjeu de l'agrégation par boosting est tout à fait  $\neq$  : il s'agit d'une **stratégie adaptative** (boosting).

⇒ Améliore l'ajustement par 1 construction adaptative séquentielle d'estimateurs, puis une combinaison de ces estimateurs pour éviter le surapprentissage.

**Rq** : les principes de bagging / boosting concernent tte modélisation...mais ont principalement un intérêt dans le cas de modèles instables (ex : CART...)

## EVOLUTION PAR RAPPORT AU BAGGING

On traite le problème du biais de l'estimateur en plus de traiter la réduction de variance.

En effet, l'agrégation par bagging ne corrige pas le biais... puisque l'espérance est un opérateur linéaire, et le biais est défini par une espérance.

Or, dans le cas d'arbres individuels simples ("weak learner"), le biais peut être **important**.

⇒ Le boosting construit une famille de modèles récurrente : chaque modèle est une version adaptative du précédent.

## PROCEDURE DU BOOSTING

On peut écrire le boosting comme suit.

A la 1<sup>ère</sup> étape, on estime le modèle  $m_1$  pour  $\mathbf{y}$ , à partir de  $\mathbf{x}$ .

⇒ On en déduit le vecteur d'erreurs  $\epsilon_1$ .

A la 2<sup>ème</sup> étape, on estime le modèle  $m_2$  pour  $\epsilon_1$ , à partir de  $\mathbf{x}$ .

⇒ On en déduit le vecteur d'erreurs  $\epsilon_2$ .

On réitère ce procédé...et on obtient à l'étape  $k$  :

$$m^{(k)}(\mathbf{x}) = \underbrace{m_1(\mathbf{x})}_{\sim y} + \underbrace{m_2(\mathbf{x})}_{\epsilon_1} + \dots + \underbrace{m_k(\mathbf{x})}_{\epsilon_{k-1}} = m^{(k-1)}(\mathbf{x}) + m_k(\mathbf{x}).$$

## STRATÉGIE ADAPTATIVE

Pour s'adapter de proche en proche, on donne **+ de poids dans l'estimation suivante aux observ. mal prédites** précédemment.

Intuitivement, l'algorithme concentre ses efforts sur les observ. les + difficiles à ajuster, tout en limitant l'overfitting par l'agrégation...

Les  $\neq$  algo. de boosting diffèrent par leurs caractéristiques :

- la façon de **pondérer l'importance des indiv. mal estimés** ;
- la façon de **pondérer les modèles** lors de l'agrégation ;
- leur **objectif** (prédire  $Y$  réelle, binaire, ...)) ;
- la **fonction de perte** qui mesure l'erreur d'ajustement (+ ou - sensible aux valeurs atypiques par ex.)

## ALGORITHME D'ORIGINE : ADABOOST

Au départ, cet algorithme est proposé pour un problème de discrimination à 2 classes.

Notons  $\delta$  la fonction de discrimination, à valeurs dans  $\{-1, 1\}$ .

**Algorithme :**

- 1 Soit  $y_0$  à prévoir (connaissant  $x_0$ ), et  $z = \{(x_1, y_1), \dots, (x_n, y_n)\}$  un échantillon.
- 2 On initialise les poids (équipondération au départ) :

$$\omega = \{\omega_i = \frac{1}{n}; i = 1, \dots, n\}$$

③ De  $m = 1$  à  $M$  ( $m$  est le  $m^{\text{ième}}$  modèle) :

- ① on estime  $\delta_m$  sur l'échantillon pondéré par  $\omega$ .
- ② on calcule le taux d'erreur apparent :  $\hat{\epsilon}_p = \sum_{i=1}^n \omega_i \mathbb{1}_{\delta_m(x_i) \neq y_i}$ .
- ③ on calcule les logit relatifs au modèle  $m$  :  $c_m = \ln\left(\frac{1-\hat{\epsilon}_p}{\hat{\epsilon}_p}\right)$ .  
Ainsi,  $\hat{\epsilon}_p \nearrow \Rightarrow c_m \searrow \Rightarrow$  on pondérera + les bons modèles.
- ④ on met à jour les pondérations :

$$\omega_i = \omega_i \exp(c_m \mathbb{1}_{\delta_m(x_i) \neq y_i})$$

Ainsi, on pondère + les observations mal classées...

- ⑤  $m \leftarrow m + 1$ , retour à l'étape 1 de la boucle.

④ Résultat du vote :

$$\hat{\Phi}_M(x_0) = \text{signe} \left[ \sum_{m=1}^M c_m \delta_m(x_0) \right].$$

**Remarque** : il faut vérifier à chaque étape que le modèle courant fait mieux qu'une prévision aléatoire, i.e.

$$\hat{\epsilon}_p < 0,5.$$

Effectivement, le poids  $c_m$  du modèle correspondant devient négatif sinon !

De nombreuses adaptations de cet algo. ont été proposés, avec des fonctions de perte adaptées aux cas où :

- Y quantitative,
- Y qualitative à plusieurs modalités,
- ...

## AUTRES PONDÉRATIONS

Parfois, on utilise des classifieurs pour lesquels il est difficile (voire impossible) d'intégrer une pondération des observations...

La stratégie revient à créer aléatoirement des échantillons (un peu comme en bootstrap), en procédant comme suit :

- chaque modèle sera construit sur un nouvel échantillon ;
- la proba. de tirer (avec remise) chaque observ. est inversement proportionnelle à sa qualité d'ajustement dans l'itération précédente.

C'est ce qu'on appelle des **arcing classifiers** (adaptively resample and combine) (voir les travaux de Breiman).

## ADABOOST AVEC Y CONTINUE

On est donc dans un cadre de régression, où  $Y$  est quantitative.

### Algorithme :

- ① Soit  $y_0$  à prévoir (connaissant  $x_0$ ), et  $z = \{(x_1, y_1), \dots, (x_n, y_n)\}$  un échantillon.
- ② On initialise un vecteur de proba.  $p$  par une loi uniforme (équipondération) :  $p = \{p_i = 1/n\}$
- ③ Pour  $m = 1$  à  $M$  ( $m$  est le  $m^{\text{ième}}$  modèle) :
  - ① on tire avec remise dans  $z$  un échantillon  $z_m^*$  suivant  $p$ .
  - ② on estime  $\hat{\Phi}_m$  sur  $z_m^*$ .
  - ③ on calcule sur l'échantillon initial  $z$  les quantités :

- $l_m(i) = Q(y_i, \hat{\Phi}_m(x_i))$ , pour  $i = 1, \dots, n$  et  $Q$  la perte ;
- $\hat{\epsilon}_m = \sum_i p_i l_m(i)$
- $\omega_i = g(l_m(i)) p_i$ , avec  $g$  continue décroissante ;
- on met à jour les proba. de tirage :  $p_i = \frac{\omega_i}{\sum_i \omega_i}$ .

- ④ **Prévision du modèle agrégé** :  $\hat{\Phi}_M(x_0)$  est la moyenne (ou médiane) des prévisions  $\hat{\Phi}_m(x_0)$ , pondérée par des poids  $\ln(1/\beta_m)$  (cf ci-dessous pour  $\beta_m$ ).

## Remarques :

- $Q$  : souvent perte quadratique, mais p-ê une autre fonction !
- $\beta_m = \frac{\hat{\epsilon}_m}{L_m - \hat{\epsilon}_m}$ , avec  $L_m = \sup_i l_m(i)$ , et  $g(l_m(i)) = \beta_m^{\frac{1-l_m(i)}{L_m}}$
- Condition supp. ajoutée : arrêt et réinitialisation à des poids uniformes si  $\hat{\epsilon}_m < 0.5L_m$  (erreur trop dégradée) ;
- $\beta_m$  : indicateur de la performance du prédicteur  $m$  sur  $z$ .

## VISION PAS À PAS D'ADABOOST

Comme nous l'avons vu, cet algorithme fonctionne pas-à-pas : c'est la raison pour laquelle on l'appelle le **Gradient Boosting** (déplacement : opposé de la pente de fonction à minimiser, cf expansion de Taylor et algo. de Newton).

Une manière d'écrire l'optimisation avec cette vision à l'étape  $m$  (rappel :  $c$  est lié à la performance du modèle) :

$$(c_m, \gamma_m) = \arg \min_{(c, \gamma)} \sum_{i=1}^n Q(y_i, \hat{\Phi}_{m-1}(x_i) + c\delta(x_i, \gamma)).$$

## GENERALISATION : BOOSTING, GRADIENT ADAPTATIF

En d'autres termes, on construit une suite de modèles

$$m^{(k)}(\mathbf{x}) = m^{(k-1)}(\mathbf{x}) + \alpha f^{\star}(\mathbf{x})$$

où

$$f^{\star}(\mathbf{x}) = \arg \min_{f \in \mathcal{W}} \left( \sum_{i=1}^n l(y_i - m^{(k-1)}(\mathbf{x}_i), f(\mathbf{x}_i)) \right),$$

avec  $l$  fonction de perte, et  $\mathcal{W}$  un ensemble de weak-learners...

Rq : ces weak-learners sont svt des CART ("stumps")...Gradient Tree Boosting !

## PARAMETRES DE TUNING

Dans cet algorithme, voici les paramètres de tuning :

- nombre d'itérations : combien d'étapes  $M$  considérer ?
- profondeur des arbres, ...
- “shrinkage”  $\alpha \in [0, 1]$  : assurer une convergence lente !  
Plutôt que  $\epsilon_1 = y - m_1(\mathbf{x})$ , on considère

$$\epsilon_1 = y - \alpha m_1(\mathbf{x}).$$

Rq :

- Algo. très performant car peut corriger biais et variance.  
Néanmoins, les multiples param. de tuning rendent la gestion du surapprentissage difficile...
- En reg. lin.,  $\epsilon \perp \mathbf{X} \Rightarrow$  impossible d'apprendre de nos erreurs !

# XGBOOST

eXtreme Gradient BOOSTing

En...

En...

## 7 Réseau de neurones et Deep Learning

- Introduction
- Neurone formel et fonctionnement d'un perceptron multicouches
- Estimation des paramètres
- Paramétrage du réseau
- Deep learning et autres types de réseaux

## UN PEU D'HISTOIRE

Réseaux de neurones sont une branche de l'IA (Intellig. Artific.) qui a pour but de simuler le comportement du cerveau humain.

→ Approche connexionniste (connaissance répartie), avec des couches... :

- ① entrée,
- ② coeur,
- ③ sortie.

Ds les années 1970, mise en oeuvre difficile car puissance des ordinateurs limitée ⇒ développement de l'approche séquentielle ou symbolique → systèmes experts à connaissance localisée.

## EXPERTISE HUMAINE

**But** : automatiser le principe de l'expertise humaine via 3 concepts :

- ① une base de connaissances : propositions logiques élémentaires,
- ② une base de faits : données, observations,
- ③ un moteur d'inférence : applique les règles expertes sur la base des faits.

⇒ En déduit de  $n_v$  faits (expérience) jusqu'à réaliser l'objectif !

**Pb** : complexité...(algorithmiquement, et en termes de modélisation)

## PRINCIPAUX RÉSULTATS UTILISÉS

Finalement, les réseaux de neurones se sont développés grâce à l'essor de l'informatique...

Et l'approche connexionniste a été relancée, grâce notamment aux deux résultats théoriques principaux suivants :

- l'estimation du gradient par **rétropropagation de l'erreur** (Hopkins, 1982) ;
- l'**analogie avec les modèles Markoviens** en mécanique statistique (Hopfield, 1982).

**Remarque** : large variété d'applications, technique complémentaire de méthodes stats usuelles (MLE, ...).

# RÉSEAU NEURONAL

**Réseau neuronal** : association de neurones formels  $\Rightarrow$  créé un graphe + ou - complexe d'objets élémentaires.

Les  $\neq$  réseaux se distinguent par 4 composantes :

- 1 organisation du graphe (couches, ...);
- 2 niveau de complexité (nb neurones, ...);
- 3 type des neurones (transition, activation);
- 4 objectif (apprentissage supervisé ou non, ...).

## 7 Réseau de neurones et Deep Learning

- Introduction
- Neurone formel et fonctionnement d'un perceptron multicouches
- Estimation des paramètres
- Paramétrage du réseau
- Deep learning et autres types de réseaux

## LE NEURONE FORMEL

Défini sur la base du fonctionnement d'un **neurone biologique** !

C'est un modèle caractérisé par :

- un état interne, noté  $s \in \mathcal{S}$  ;
- des signaux d'entrée, notés  $x_1, \dots, x_p$  ;
- une fonction d'activation :

$$s = h(x_1, \dots, x_p) = f(\alpha_0 + \sum_{j=1}^p \alpha_j x_j) = f(\alpha_0 + \alpha^T x).$$

**Voc** : on appelle  $\alpha$  le vecteur des **poids**,  $\alpha_0$  le **biais** du neurone.

**Rq** : les poids  $\alpha$  sont estimés durant l'apprentissage : mémoire ou "connaissance répartie" du réseau.

## TYPES DE NEURONE

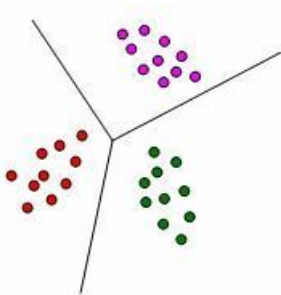
≠ types de neurone se distinguent par leur fonction d'activation  $f$  :

- type linéaire :  $f(x) = x$
- type sigmoïde :  $f(x) = (1 + e^{-x})^{-1}$
- type seuil :  $f(x) = \mathbb{1}_{[0, +\infty[}(x)$
- type radiale :  $f(x) = \sqrt{\frac{1}{2\pi}} \exp\left(-\frac{1}{2}x^2\right)$
- type stochastique :  $f(x) = 1$  avec proba  $(1 + e^{-x/H})^{-1}$ , 0 sinon ; ...

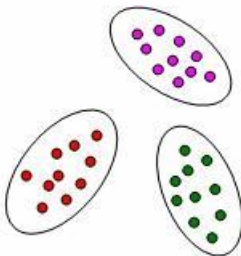
Rq : en data mining, les 2<sup>iers</sup> types de réseaux sont les + utilisés car fonction d'activation est différentiable → adapté à un algo. d'apprentissage impliquant la rétropropagation du gradient.

## DIFFERENCES DE SEPARATION

En fonction de l'activation choisie, les données sont séparées différemment. Exemple ici : fonction linéaire VS fonction radiale...



PMC

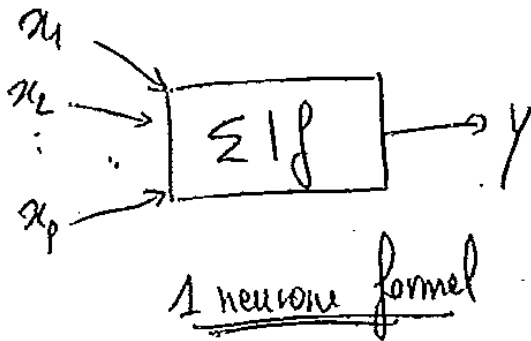


RBF

# AUTRES FONCTIONS D'ACTIVATION

| Fonction              | Définition  | Description  | Intervalle de définition |
|-----------------------|---|--|--------------------------|
| Identité              | $a$   | L'activation du neurone est transmise directement en sortie  | $(-\infty, +\infty)$     |
| Sigmoïdale logistique | $\frac{1}{1 + e^{-a}}$  | Une courbe en "S"  | $(0,1)$                  |
| Tangente hyperbolique | $\frac{e^a - e^{-a}}{e^a + e^{-a}}$   | Une courbe sigmoïdale similaire à la fonction logistique. Produit généralement de meilleurs résultats que la fonction logistique en raison de sa symétrie. Idéale pour les perceptrons multicouches, en particulier, pour les couches cachées  | $(-1, +1)$               |
| Exponentielle         | $e^{-a}$  | La fonction exponentielle négative   | $(0, +\infty)$           |
| Sinus                 | $\sin(a)$   | S'utilise éventuellement si les données sont distribuées radialement. N'est pas utilisé par défaut   | $[0,1]$                  |
| Softmax               | $\frac{\exp(a_i)}{\sum \exp(a_i)}$  | Essentiellement utilisé (mais pas uniquement) pour des tâches de classification. Permet de construire des réseaux de neurones avec plusieurs sorties normalisées ce qui le rend particulièrement adapté à la création de classifications par les réseaux de neurones avec des sorties probabilistes. | $[0,1]$                  |
| Gaussienne            | $\frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x - \mu)^2}{2\sigma^2}\right]$ | Ce type de fonction d'activation isotropique Gaussienne n'est utilisé que par les unités cachées d'un réseau de  |                          |

## SCHÉMA DE FONCTIONNEMENT



## RÉSULTAT FONDAMENTAL

Le résultat suivant est la base de l'approche de modélisation par un réseau de neurones.

Théorème d'**approximation "universelle"** :

Toute fonction régulière peut être approchée uniformément avec une précision arbitraire et dans un domaine fini de l'espace de ses variables par un réseau de neurones comportant une couche de neurones cachés (en nombre fini et possédant tous la même fonction d'activation), et un neurone de sortie de type linéaire.

## LE PERCEPTRON MULTICOUCHES (PMC)

Intéressons nous ici à un réseau “statique” (ou *feedforward*, i.e. pas de boucle rétroactive), dans un but d'apprentissage supervisé.

En voici quelques caractéristiques :

- ➊ **architecture** : PMC composé de couches successives, où 1 couche : ens. de neurones sans connexion entre eux ;
- ➋ **fonction de transfert** : un PMC réalise une transformation des variables d'entrée :

$$Y = \Phi(X_1, X_2, \dots, X_p; \alpha),$$

avec  $\alpha = (\alpha_{jkl})$  pr la  $j^{\text{è}}$  entrée ( $x_j$ ) du  $k^{\text{è}}$  neurone de  $l^{\text{è}}$  couche.

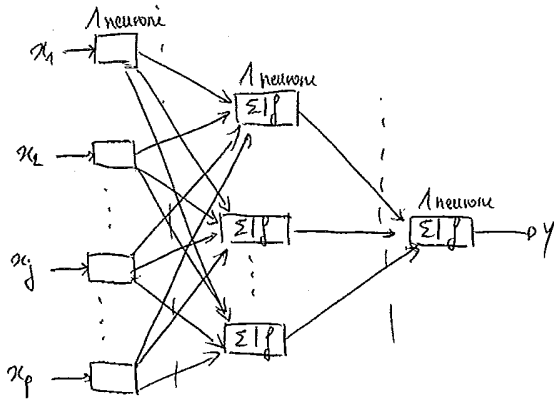
- ③ **généralisation** : cas de la régression, avec un perceptron à une couche cachée de  $q$  neurones, un neurone de sortie.  
⇒ La **fonction de transfert** s'écrit

$$Y = \Phi(x; \alpha, \beta) = \beta_0 + \beta^T z, \quad \text{avec } z_k = f(\alpha_{k0} + \alpha_k^T x),$$

pour  $k = 1, \dots, q$  (identifiant neurone ds couche cachée).

Usuellement, on a

- en rég. : dernière couche avec 1 seul neurone, avec  $f = Id$  ; tandis que neurones couche cachée ont une fonct. sigmoïde ;
- en classif. binaire : neurone de sortie muni de la fonction d'activation sigmoïde ;
- en discrimination à  $m$  classes :  $m$  neurones de sortie, munis de sigmoïde.



non-complétée  
parmi les couches car  
ne modifie pas les  
entrées.

TRANSFERT

le  
e  
d