Apprentissage Machine / Statistique

Agrégation de modèles

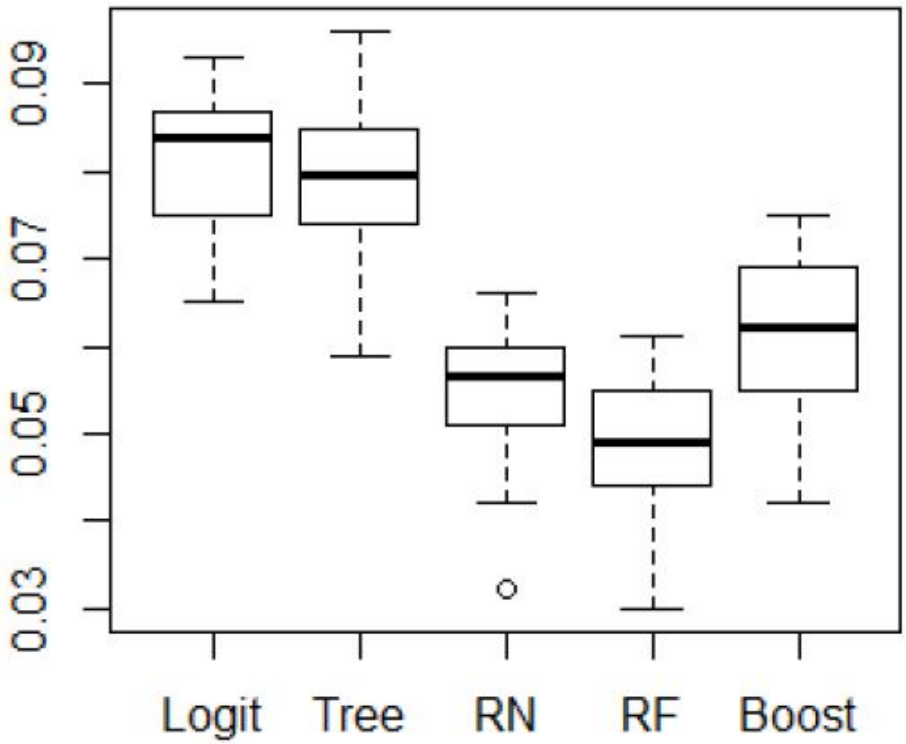
PHILIPPE BESSE

INSA de Toulouse
Institut de Mathématiques

Introduction

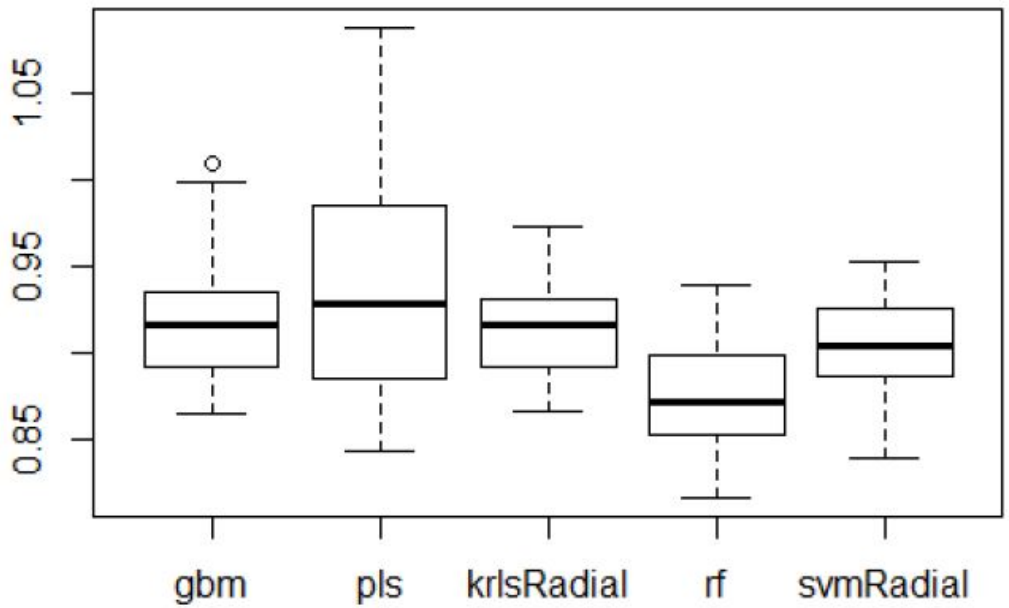
- Stratégies adaptatives (boosting) ou aléatoires (bagging)
- Combinaison ou agrégation de modèles (presque) sans sur-ajustement
- Apprentissage machine (machine learning) et Statistique
- Comparatifs heuristiques et propriétés théoriques
- Bagging pour bootstrap aggregating (Breiman, 1996)
- Forêts aléatoires (random forests) (Breiman, 2001)
- Du Boosting (Freund et Shapiro,1996) déterministe et adaptatif à l'*extrem gradient boosting*
- Toute méthode de modélisation non linéaire
- Méthodes efficaces : Fernandez-Delgado et al. (2014), *Kaggle*

	Logit	Tree	RN	RF	Boost
Moyenne	0.0820	0.078	0.055	0.049	0.061



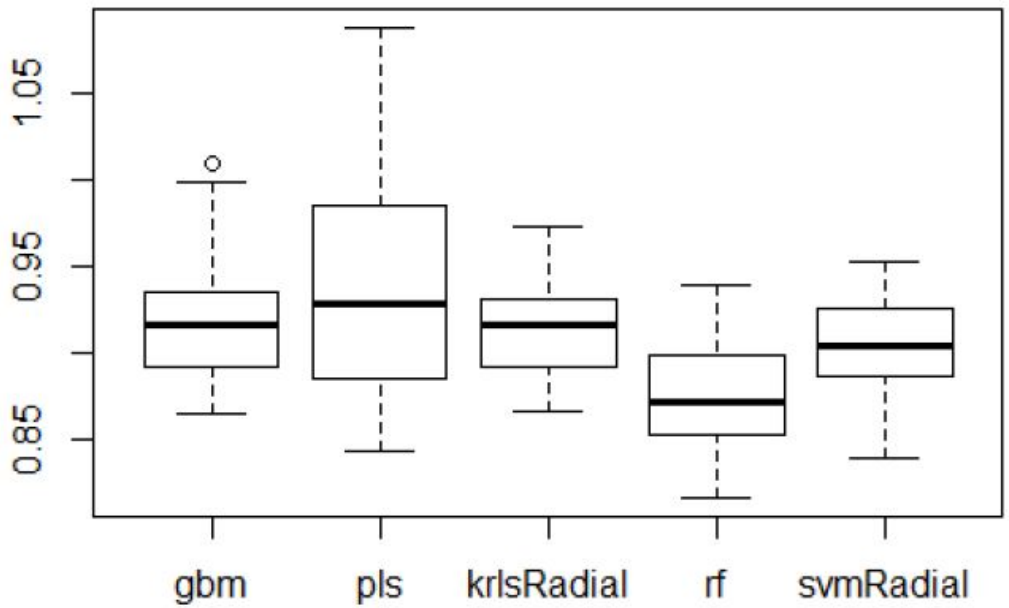
Spams : Comparaison par validation croisée Monte Carlo des erreurs de prévision de détections de pourriels

	gbm	pls	krlsRadial	rf	svmradial
Moyenne	0.92	0.94	0.92	0.87	0.90



Criblage virtuel de molécule : prévision de la capacité d'une molécule à traverser la barrière du cerveau

	gbm	pls	krlsRadial	rf	svmradial
Moyenne	0.92	0.94	0.92	0.87	0.90



Marketing bancaire : Score d'appétance de la carte Visa Premier

Bootstrap aggregating : principe

- Soit Y une variable à expliquer quantitative ou qualitative
- $X^1, \ldots, X^p$ les variables explicatives
- $f(\mathbf{x})$ un modèle fonction de $\mathbf{x} = \{x^1, \ldots, x^p\} \in \mathbb{R}^p$
- $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ échantillon de loi F et de taille n
  - $f(.) = E_F(\widehat{f}_{\mathbf{z}})$ estimateur sans biais de variance nulle
  - B échantillons indépendants $\{\mathbf{z}_b\}_{b=1,B}$
  - Y quantitative : $\widehat{f}_B(.) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(.)$ (moyenne)
  - Y qualitative : $\widehat{f}_B(.) = \arg\max_j \text{card}\left\{b \mid \widehat{f}_{\mathbf{z}_b}(.) = j\right\}$ (vote)
- Principe : Moyenner des prévisions indépendantes pour réduire la variance
- B échantillons indépendants remplacés par B réplications bootstrap

Bagging : algorithme

- Soit $\mathbf{x}_0$ à prévoir et
- $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ un échantillon
- **pour** $b = 1$ **à** B
  - Tirer un échantillon bootstrap $\mathbf{z}_b^*$
  - Estimer $\widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ sur l'échantillon bootstrap
- Calculer l'estimation moyenne $\widehat{f}_B(\mathbf{x}_0) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ ou le résultat du vote

Bagging : utilisation

- Estimation *bootstrap out-of-bag* de l'erreur de prévision : contrôle de la qualité et du sur-ajustement
- CART pour construire une famille d'arbres binaires
- Trois stratégies d'élagage sont alors possibles :
  1. garder un arbre complet pour chacun des échantillons
  2. arbre d'au plus q feuilles
  3. arbre complet élagué par validation croisée
- Première stratégie compromis entre calculs et qualité de prévision : faible biais de chaque arbre et variance réduite par agrégation

Bagging : problèmes

- Temps de calcul et contrôle de l'erreur
- Stockage de tous les modèles de la combinaison
- Modèle boîte noire

Forêts aléatoires : principe

- Amélioration du bagging d'arbres binaires
- Variance de B variables corrélées : $\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$
- Ajout d'une randomisation pour rendre les arbres plus indépendants
- Choix aléatoire des variables
- Intérêt : grande dimension

Forêts aléatoires : algorithme

- Soit $\mathbf{x}_0$ à prévoir et $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ un échantillon **pour** $b = 1$ **à** B
  - Tirer un échantillon bootstrap $\mathbf{z}_b^*$
  - Estimer un arbre avec randomisation des variables :
  - Pour chaque nœud, tirage aléatoire de m prédicteurs
- Calculer l'estimation moyenne $\widehat{f}_B(\mathbf{x}_0) = \frac{1}{B}\sum_{b=1}^{B} \widehat{f}_{\mathbf{z}_b}(\mathbf{x}_0)$ ou le vote

Forêts aléatoires : utilisation

- Élagage : Arbres de taille q, ou complet.
- La sélection aléatoire des m prédicteurs ($m = \sqrt{p}$ en classification, $\frac{p}{3}$ en régression) accroît la variabilité
- Chaque modèle de base est moins performant mais l'agrégation est performante
- Évaluation itérative de l'erreur out-of-bag

Aide à l'interprétation : indices d'importances

- Mean Decrease Accuracy
- Mean Decrease Gini

Forêts aléatoires : implémentations

R

- `randomForest` : interface du programme Fortran77
- `ranger`

Weka version en java

Scikit-learn analogue de la version originale

Spark/MLlib

- arbre "scalable" du projet PLANET
- Deux paramètres supplémentaires

  `subsamplingRate = 1.0`

  `maxBins=32`

Autres utilisations

- Proximités ou similarités des observations
- Atypiques ou anomalies
- Classification non supervisée
- Imputation de données manquantes `missForest`
- Détection d'anomalies (*outlier*)
- Durée de vie : *survival forest*

Boosting : principe

- Améliorer les compétences d'un faible classifieur (Schapire, 1990 ; Freund et Schapire, 1996)
- AdaBoost (Adaptative boosting) prévision d'une variable binaire
- Réduire la variance mais aussi le biais de prévision
- Meilleure méthode off-the-shelf
- Agrégation d'une famille de modèles récurents
  Chaque modèle est une version adaptative du précédent en donnant plus de poids, lors de l'estimation suivante, aux observations mal ajustées
- Variantes : type de la variable à prédire (binaire, k classes, réelles), fonction perte (robustesse)

AdaBoost discret

- Fonction δ de discrimination $\{-1, 1\}$
- Soit $\mathbf{x}_0$ à prévoir et
- $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ un échantillon
- Initialiser les poids $\boldsymbol{w} = \{w_i = 1/n \; ; \; i = 1, \ldots, n\}$
- **pour** $m = 1$ **à** M
  - Estimer δ_m sur l'échantillon pondéré par $\boldsymbol{w}$
  - Calculer le taux d'erreur apparent : $\widehat{\mathcal{E}}_p = \frac{\sum_{i=1}^n w_i \mathbf{1}\{\delta_m(\boldsymbol{x}_i) \neq y_i\}}{\sum_{i=1}^n w_i}$
  - Calculer les logit : $c_m = \log((1 - \widehat{\mathcal{E}}_p)/\widehat{\mathcal{E}}_p)$
  - Nouvelles pondérations (normalisation) :
    $w_i \leftarrow w_i . \exp\left[c_m \mathbf{1}\{\delta_m(\boldsymbol{x}_i) \neq y_i\}\right] ; i = 1, \ldots, n$
- Résultat du vote : $\widehat{f}_M(\boldsymbol{x}_0) = \text{signe}\left[\sum_{m=1}^M c_m \delta_m(\boldsymbol{x}_0)\right]$

Boosting : utilisation

- Arbre comme modèle de base
- Recommandation : q entre 4 et 8
- Version aléatoire : Arcing (Breiman, 1998)
- Empiriquement, l'erreur de prévision peut continuer à décroître après que l'erreur d'ajustement se soit annulée
- Attention aux données bruitées (erreur de label), source de dérive ou sur-apprentissage.
- Boosting réduit la variance comme le bagging mais aussi le biais

Boosting pour la régression : algorithme

- Soit $\mathbf{x}_0$ à prévoir et $\mathbf{z} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ un échantillon
- Initialiser $\mathbf{p} = \{p_i = 1/n \ ; \ i = 1, \ldots, n\}$
- **pour** $m = 1$ à M
  - Tirer avec remise dans $\mathbf{z}$ un échantillon $\mathbf{z}_m^*$ suivant $\mathbf{p}$
  - Estimer $\widehat{f}_m$ sur l'échantillon $\mathbf{z}_m^*$
  - Calculer à partir de l'échantillon initial $\mathbf{z}$:

$$l_m(i) = l\left(y_i, \widehat{f}_m(\mathbf{x}_i)\right) \quad i = 1, \ldots, n; \quad (l : \text{fonction perte})$$

$$\widehat{\mathcal{E}_m} = \sum_{i=1}^{n} p_i l_m(i); \quad w_i = g(l_m(i)) p_i \quad p_i \leftarrow \frac{w_i}{\sum_{i=1}^{n} w_i}$$

- Moyenne ou médiane des $\widehat{f}_m(\mathbf{x}_0)$ pondérées par $\log(\frac{1}{\beta_m})$

Boosting : utilisation

- l peut être exponentielle, quadratique ou la valeur absolue
- $L_m = \sup_{i=1,\dots,n} l_m(i)$ maximum de l'erreur observée par le modèle $\widehat{f}_m$ sur l'échantillon initial

$$g(l_m(i)) = \beta_m^{1-l_m(i)/L_m}$$

$$\text{avec} \quad \beta_m = \frac{\widehat{\mathcal{E}_m}}{L_m - \widehat{\mathcal{E}_m}}$$

- Algorithme arrêté ou réinitiallisé à des poids uniformes si l'erreur se dégrade trop : si $\widehat{\mathcal{E}_m} < 0.5 L_m$

Boosting : interprétation

- Approximation de f par un modèle additif pas à pas (Hastie et col., 2001)

$$\widehat{f}(\boldsymbol{x}) = \sum_{m=1}^{M} c_m \delta(\boldsymbol{x}; \gamma_m)$$

- c_m est un paramètre
- δ le classifieur de base fonction de $\boldsymbol{x}$ et dépendant d'un paramètre γ_m
- l une fonction perte

Modèle additif : optimisation

- $(c_m, \gamma_m) = \arg\min_{(c,\gamma)} \sum_{i=1}^{n} l(y_i, \widehat{f}_{m-1}(\boldsymbol{x}_i) + c\delta(\boldsymbol{x}_i;\gamma))$
- $\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + c_m\delta(\boldsymbol{x};\gamma_m)$ améliore l'ajustement précédent
- f binaire, $l(y, f(\boldsymbol{x})) = \exp[-yf(\boldsymbol{x})]$

$$(c_m, \gamma_m) = \arg\min_{(c,\gamma)} \sum_{i=1}^{n} \exp\left[-y_i(\widehat{f}_{m-1}(\boldsymbol{x}_i) + c\delta(\boldsymbol{x}_i;\gamma))\right]$$

$$= \arg\min_{(c,\gamma)} \sum_{i=1}^{n} w_i^m \exp\left[-cy_i\delta(\boldsymbol{x}_i;\gamma)\right] \text{avec} \quad w_i = \exp[-y_i\widehat{f}_{m-1}(\boldsymbol{x}_i)]$$

- w_i^m : poids fonction de la qualité de l'ajustement précédent

Modèle additif : solution

- Deux étapes : classifieur optimal puis optimisation de c_m

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} \mathbf{1}\{y_i \neq \delta(\boldsymbol{x}_i;\gamma)\} \quad \text{et} \quad c_m = \frac{1}{2}\log\frac{1-\widehat{\mathcal{E}}_p}{\mathcal{E}_p}$$

  avec $\widehat{\mathcal{E}}_p$ erreur apparente de prévision

- les w_i sont mis à jour avec : $w_i^{(m)} = w_i^{(m-1)}\exp[-c_m]$
- Adaboost approche f pas à pas par un modèle additif en utilisant une fonction perte exponentielle
- D'autres fonctions perte (robustesse)
  - LogitBoost : $l(y,f(\boldsymbol{x})) = \log_2(1+\exp[-2yf(\boldsymbol{x})]$
  - L^2Boost : $l(y,f(\boldsymbol{x})) = (y-f(\boldsymbol{x}))^2/2$

GBM : Principe 1

- Gradient Boosting Models (Friedman, 2002-2009)
- dans le cas d'une fonction perte différentiable
- Principe :
  - Construire une séquence de modèles de sorte qu'à chaque étape, chaque modèle ajouté à la combinaison, apparaisse comme un pas vers une meilleure solution
  - Ce pas est franchi dans la direction du gradient de la fonction perte approché par un arbre de régression

GBM : Principe 2

- Modèle adaptatif précédent :

$$\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + c_m \delta(\boldsymbol{x}; \gamma_m)$$

- Transformé en une descente de gradient

$$\widehat{f}_m = \widehat{f}_{m-1} - \gamma_m \sum_{i=1}^{n} \nabla_{f_{m-1}} l(y_i, f_{m-1}(x_i)).$$

- Recherche d'un meilleur pas de descente γ :

$$min_\gamma \sum_{i=1}^{n} \left[l\left(y_i, f_{m-1}(x_i) - \gamma \frac{\partial l(y_i, f_{m-1}(x_i))}{\partial f_{m-1}(x_i)} \right) \right].$$

GBM en régression : algorithme

- Soit $\mathbf{x}_0$ à prévoir
- Initialiser $\widehat{f}_0 = \arg\min_\gamma \sum_{i=1}^n l(y_i, \gamma)$
- **pour** $m = 1$ **à** M
  - Calculer $r_{mi} = -\left[\frac{\delta l(y_i, f(\boldsymbol{x}_i))}{\delta f(\boldsymbol{x}_i)}\right]_{f=f_{m-1}}$; $i = 1, \ldots, m$
  - Ajuster un arbre de régression δ_m aux $(\mathbf{x}_i, r_m i)$
  - Calculer $\gamma_{jm} = \text{arg}min_\gamma \sum_{i=1}^n l(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma \delta_m(\boldsymbol{x}_i))$
  - Mise à jour : $\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \gamma_m \delta_m(\boldsymbol{x})$
- Résultat : $\widehat{f}_M(\boldsymbol{x}_0)$

GBM : utilisation avec R

- Discrimination : autant de probabilités que de classes
- Coefficient de rétrécissement (*schrinkage*)

$$\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}\{\boldsymbol{x} \in R_{jm}\}$$

- Équilibre entre rétrécissement et nombre d'itérations
- Profondeur maximale des arbres

GBM : utilisation avec `Scikit-learn`

- Importance des variables (cf. forêt aléatoire)
- Autres paramètres

  `max_features` nombre de variables pour construire un arbre (cf. *RF*)

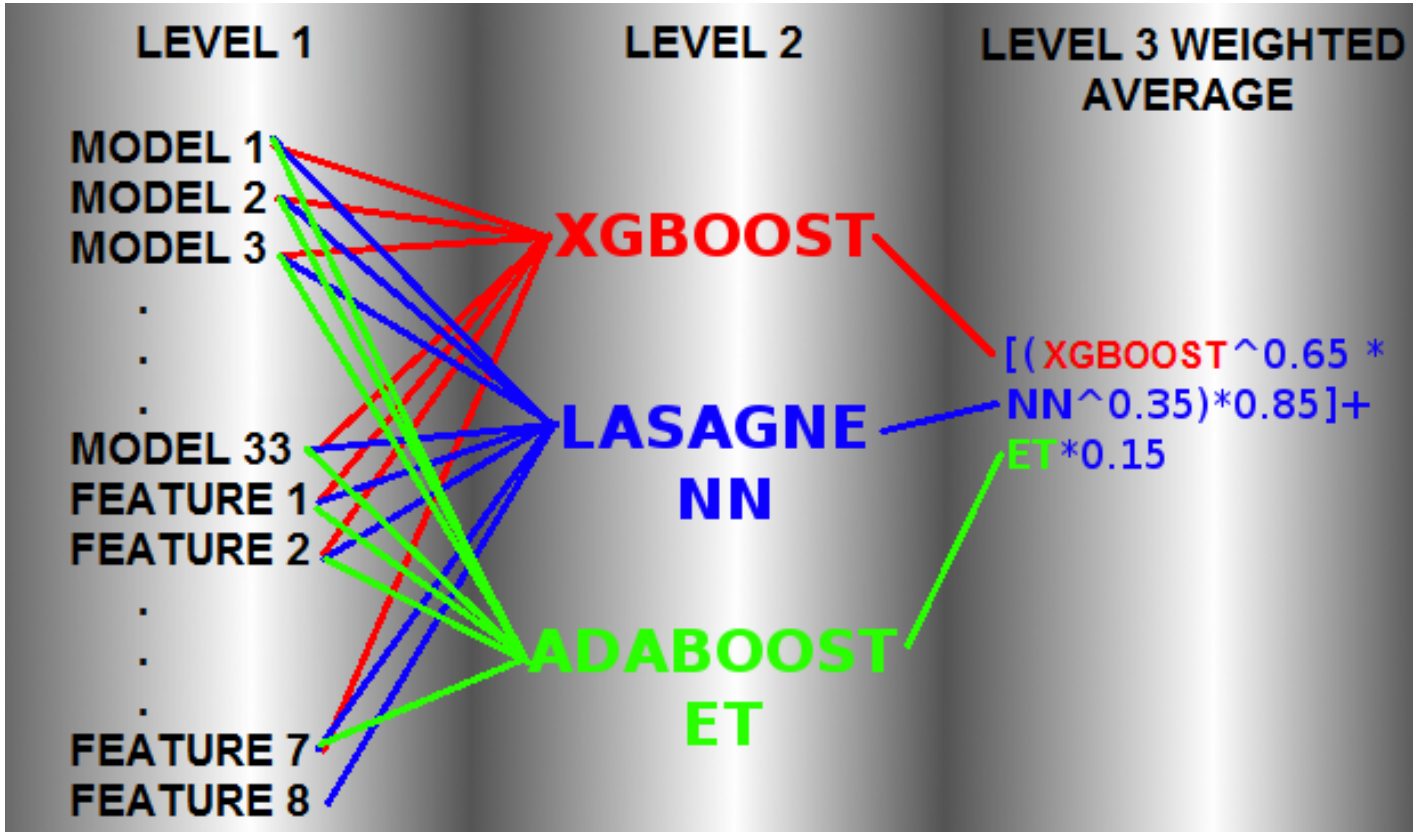
  `subsample` *Stochastic gradient boosting* : sous-échantillonnage à chaque étape

  `min_sampes_leaf, min_weight_fraction_leaf`

  `max_leaf_node`

`XGBoost` : motivation

- Algorithme de Chen et Guestrin (2016)
- Pénalisation supplémentaire pour contrôle du sur-apprentissage
- Problème : nombre de paramètres à optimiser
- Astuces d'implémentation pour parallélisation
- Environnements : R (`caret`), Python, Julia, GPU, *Amazon Web Service*, Spark...
- Solutions gagnantes des concours *Kaggle*



Kaggle : Identify people who have a high degree of Psychopathy based on Twitter usage

`XGBoost` : pénalisation

- Fonction perte L par pénalisation de l

$$\mathcal{L}(f) = \sum_{i=1}^{n} l(\widehat{[y)}_i, y_i) + \sum_{m=1}^{M} \Omega(\delta_m)$$

$$\Omega(\delta) = \alpha|\delta| + \frac{1}{2}\beta||\boldsymbol{w}||^2$$

- $|\delta|$ nombre de feuilles de l'arbre δ
- $\boldsymbol{w}$ vecteur des valeurs attribuées à chaque feuille
- Ω mélange de pénalisation l_1 et l_2

`XGBoost` : astuces

- Approximation du gradient par développement de Taylor : sommations et parallélisation
- Complexité des divisions : quantiles des distributions
- Algorithme tolérant aux Données manquantes : gradient calculé sur les valeurs présentes
- Indicateur d'importance des variables
- Gestion des matrices creuses

`XGBoost` : paramètres supplémentaires

`alpha` pénalisation de type Lasso (l_1) sur la complexité de l'arbre de régression estimant le gradient

`lambda` pénalisation de type *ridge* (l_2)

`gamma` réduction minimale de la perte pour accepter une division

`tree_method` algorithme glouton de recherche des divisions ou simplification (quantiles ou regroupement de classes)

`sketch_eps` contrôle le nombre de classes

`scale_pos_weight` à prendre en compte pour des classes déséquilibrées

Autres paramètres d'optimisation des performances

`XGBoost` : stratégie d'optimisation

- Principe :
- Valeurs par défaut et optimiser les paramètres par ordre de décroissance de l'influence supposée
- Une stratégie parmi d'autres :
  1. Nombre d'arbres
  2. Profondeur maximale *vs.* nombre d'observations minimales par feuille
  3. Réduction minimale de la perte
  4. Taux d'échantillonnage *vs.* nombre de variables utilisées
  5. Nombres d'arbres *vs.* rétrécissement
- Nécessité : puissance de calcul (GPU)

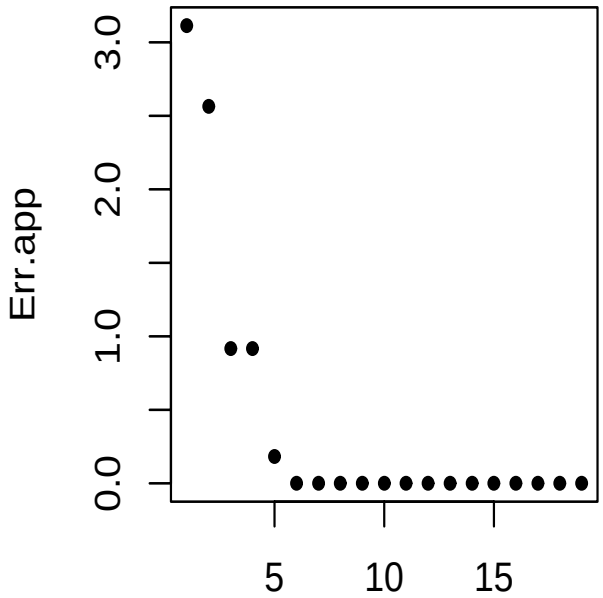
Super apprenti en régression

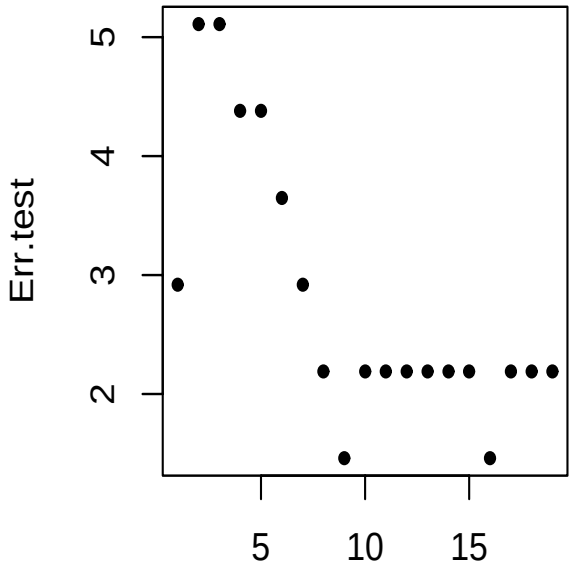
- Super learner (van der Laan et al. 2007)
- Estimer des modèles variés de régression
  modèle linéaire, PLS, arbres, neurones, svm, agrégation...
- Combinaison linéaire convexe des prévisions
- Optimisation par validation croisée

Cancer : prévisions

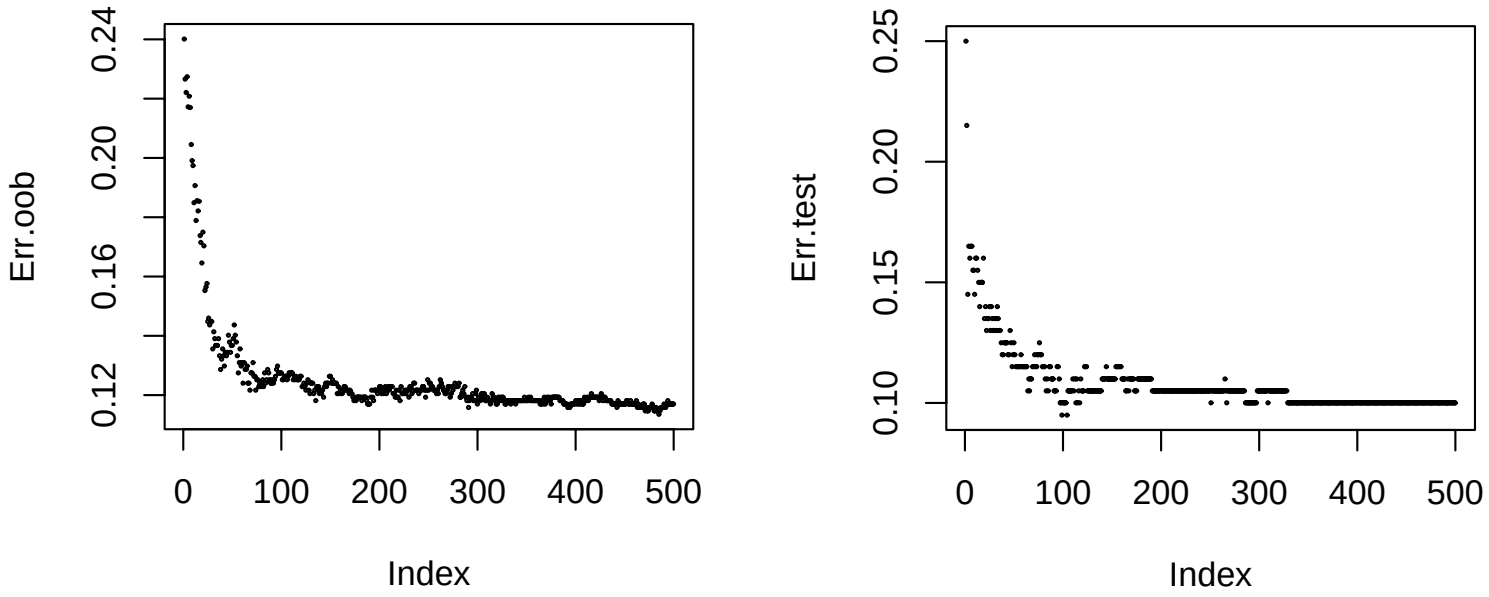
Matrices de confusion :

	bagging(ipred)		adaboost(gbm)		random forest	
	benign	malignant	benign	malignant	benign	malignant
benign	83	3	84	1	83	0
malignant	3	48	2	50	3	51

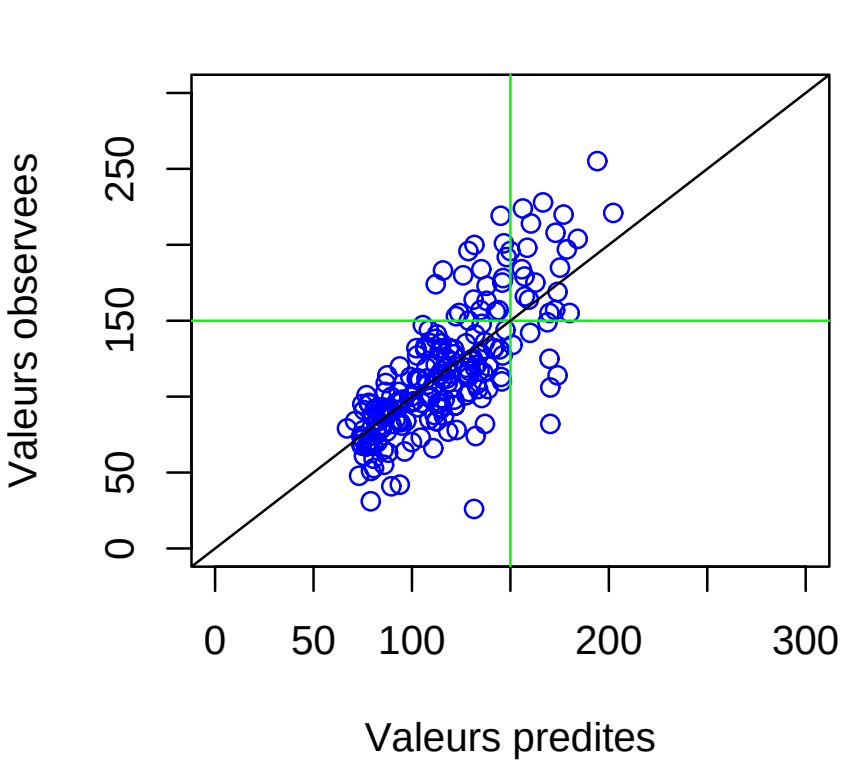


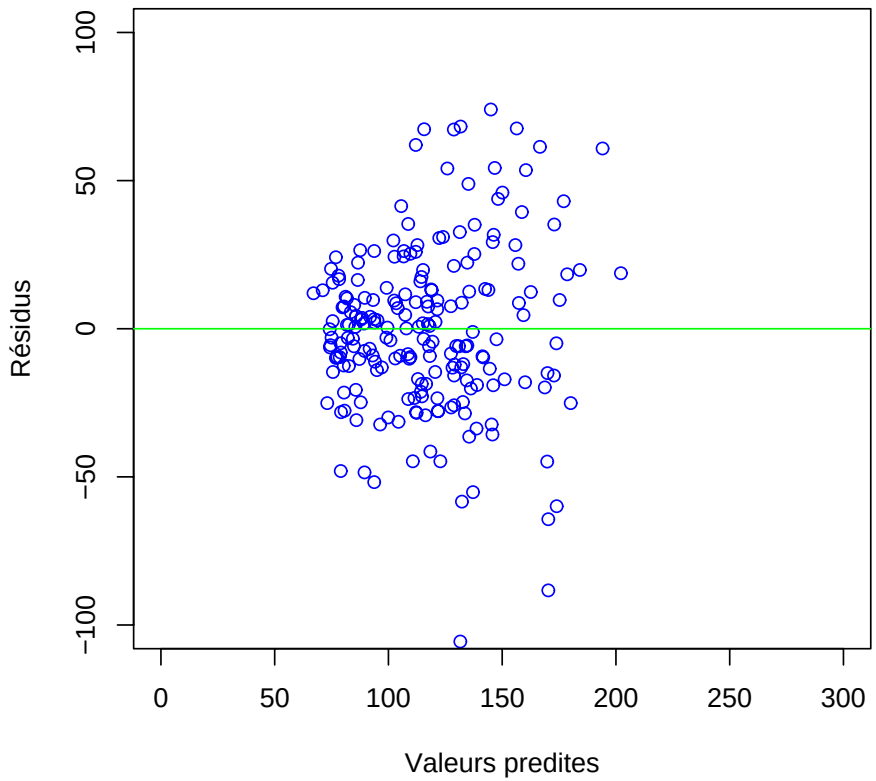


Cancer : *Evolution des taux d'erreur de l'apprentissage et du test en fonction du nombre d'arbres dans AdaBoost*



Banque : Évolution du taux de mal classés "out-of-bag" et sur l'échantillon test en fonction du nombre d'arbres de la forêt





Ozone : Valeurs observées et résidus de l'échantillon test en fonction des valeurs prédites par une forêt aléatoire