Apprentissage statistique : théorie et méthodes
M1MINT

Agathe Guilloux & Geneviève Robin

Le problème de classification binaire

On a des données d'apprentissage (learning data) pour des individus $i = 1, \ldots, n$. Pour chaque individu i :

- on a un vecteur de covariables (features) $x_i \in \mathcal{X} \subset \mathbb{R}^d$
- la valeur de son label $y_i \in \{-1, 1\}$.
- on suppose que les couples (X_i, Y_i) sont des copies i.i.d. de (X, Y) de loi inconnue et que l'on observe leurs réalisations (x_i, y_i) $(i = 1, \ldots, n)$.

But

- On veut, pour un nouveau vecteur X_+ de features, prédire la valeur du label Y_+ par $\hat{Y}_+ \in \{-1, 1\}$
- Pour cela, on utilise les données d'apprentissage $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ pour construire un **classifieur** $\hat{c}$ de telle sorte que

$$\hat{Y}_+ = \hat{c}(X_+).$$

et $\hat{Y}$ est proche de Y_+ (dans un sens à préciser).

Plan

Classifieurs constants sur une partition

On va considérer

- une partition $\mathcal{A} = \{A_1, \ldots, A_M\}$ de $\mathcal{X}$ (qui peut dépendre des données)
- l'ensemble $\mathcal{F}_{\mathcal{A}}$ des fonctions constantes sur $\mathcal{A}$
- la perte 0/1 $\ell(y, y') = \mathbb{1}_{yy' \leq 0}$

on cherche alors un classifieur $\widehat{c}$ qui vérifie

$$\widehat{c}_{\mathcal{A}} = \operatorname*{argmin}_{c \in \mathcal{F}_{\mathcal{A}}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, c(x_i)) = \operatorname*{argmin}_{c \in \mathcal{F}_{\mathcal{A}}} \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}_{y_i c(x_i) \leq 0}.$$

Vote majoritaire

En classification binaire, on sait alors que $\widehat{c}_{\mathcal{A}}$ vérifie pour $x \in A_m$

$$\widehat{c}_{\mathcal{A}}(x) = \begin{cases} 1 & \text{si } \#\{i : x_i \in A_m, y_i = 1\} > \#\{i : x_i \in A_m, y_i = -1\} \\ -1 & \text{sinon} \end{cases}$$

$$= \begin{cases} 1 & \text{si } \bar{y}_{A_m} > 0 \\ -1 & \text{sinon} \end{cases}$$

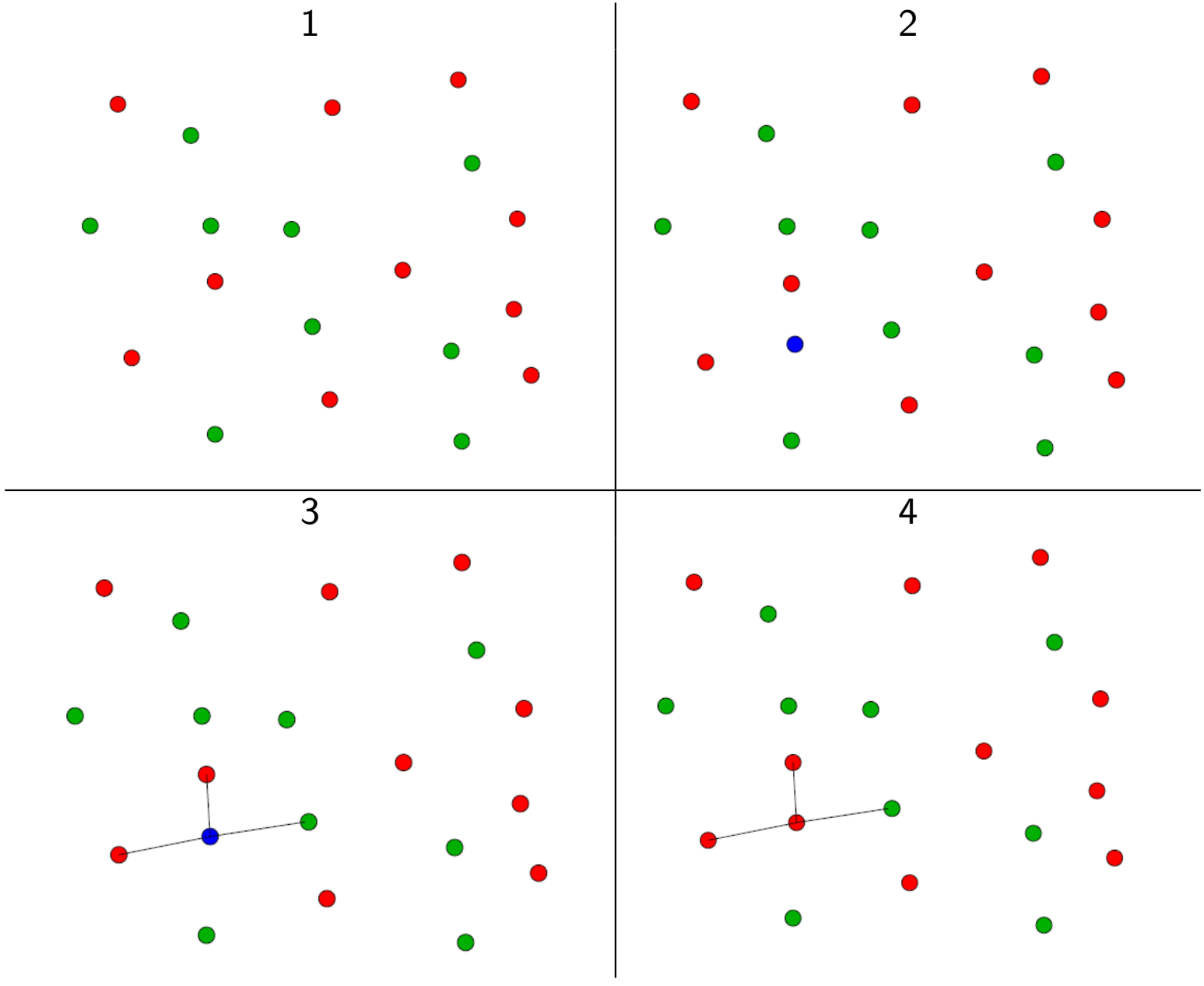
En classification multi-classes, on posera pour $x \in A_m$

$$\widehat{c}_{\mathcal{A}}(x) = \underset{k \in \{1,\dots,K\}}{\arg\max} \ \#\{i : x_i \in A_m, y_i = k\}$$

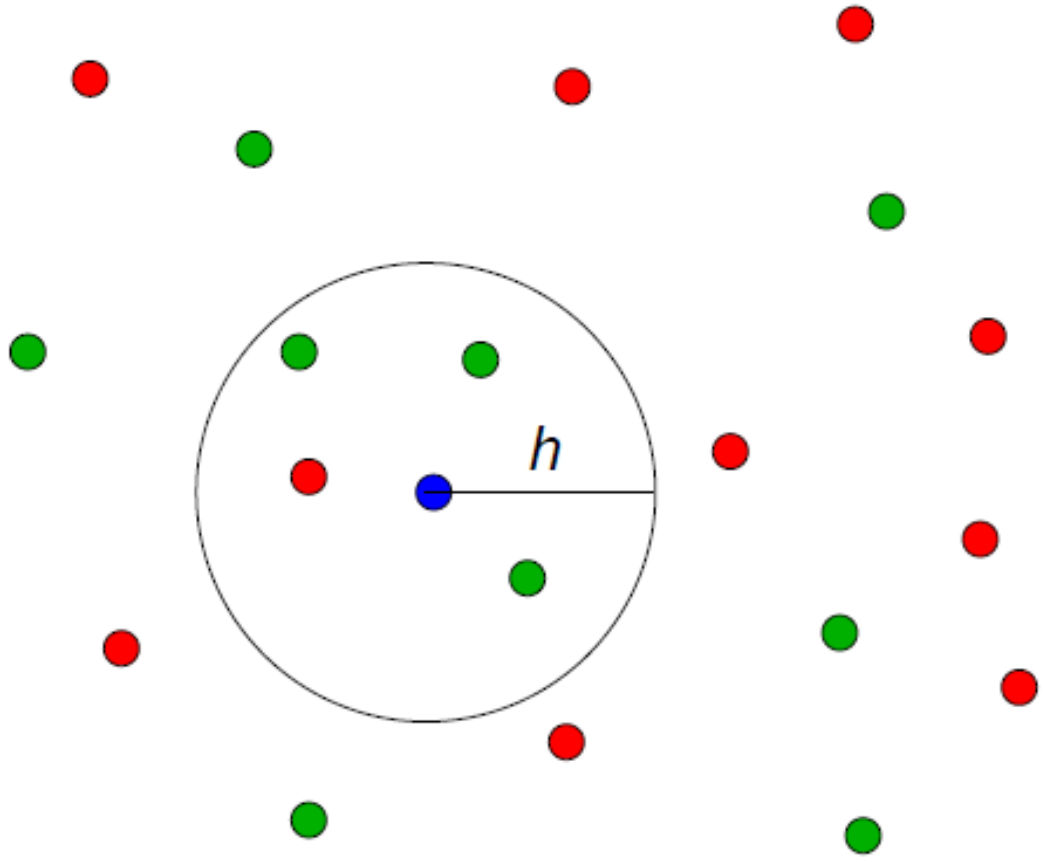
Il reste à choisir la partition $\mathcal{A} = \{A_1, \dots, A_M\}$ de $\mathcal{X}$!

Plan

Exemple: k plus proches voisins (avec $k = 3$)



Exemple: k plus proches voisins (avec $k = 4$)



k plus proches voisins

On considère l'ensemble $\mathcal{I}_x$ composé des k indices de $\{1, \ldots, n\}$ pour lesquels les distances $\|x - x_i\|$ sont minimales.

On pose

$$\widehat{c}(x) = \underset{l \in \{-1,1\}}{\arg\max} \#\{i \in \mathcal{I}_x, y_i = l\}.$$

- En pratique, il faut choisir la distance
- et k !!

Partition

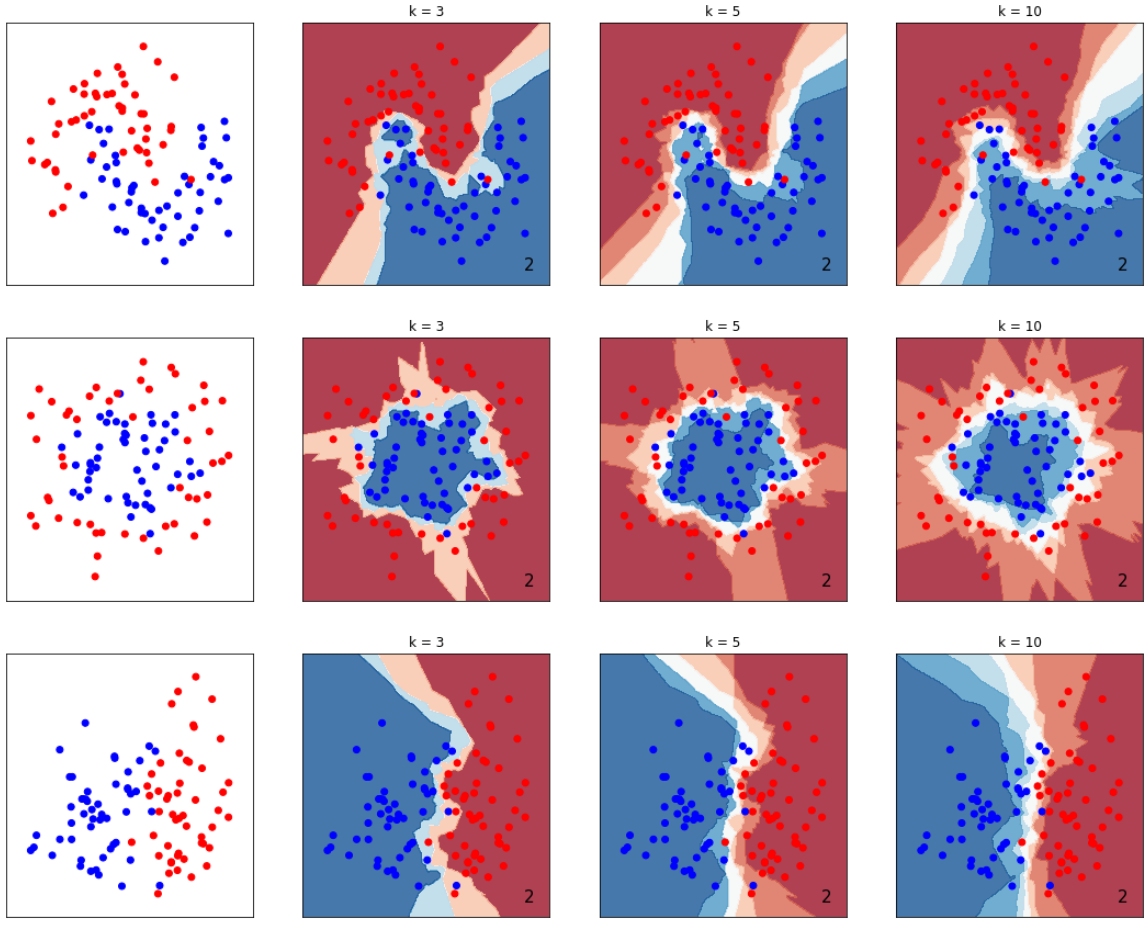
On remarque que $\mathcal{I}_x$ appartient à l'ensemble $\{\phi^1, \ldots, \phi^M\}$ des combinaisons de k éléments parmi n avec

$$M = \binom{n}{k}.$$

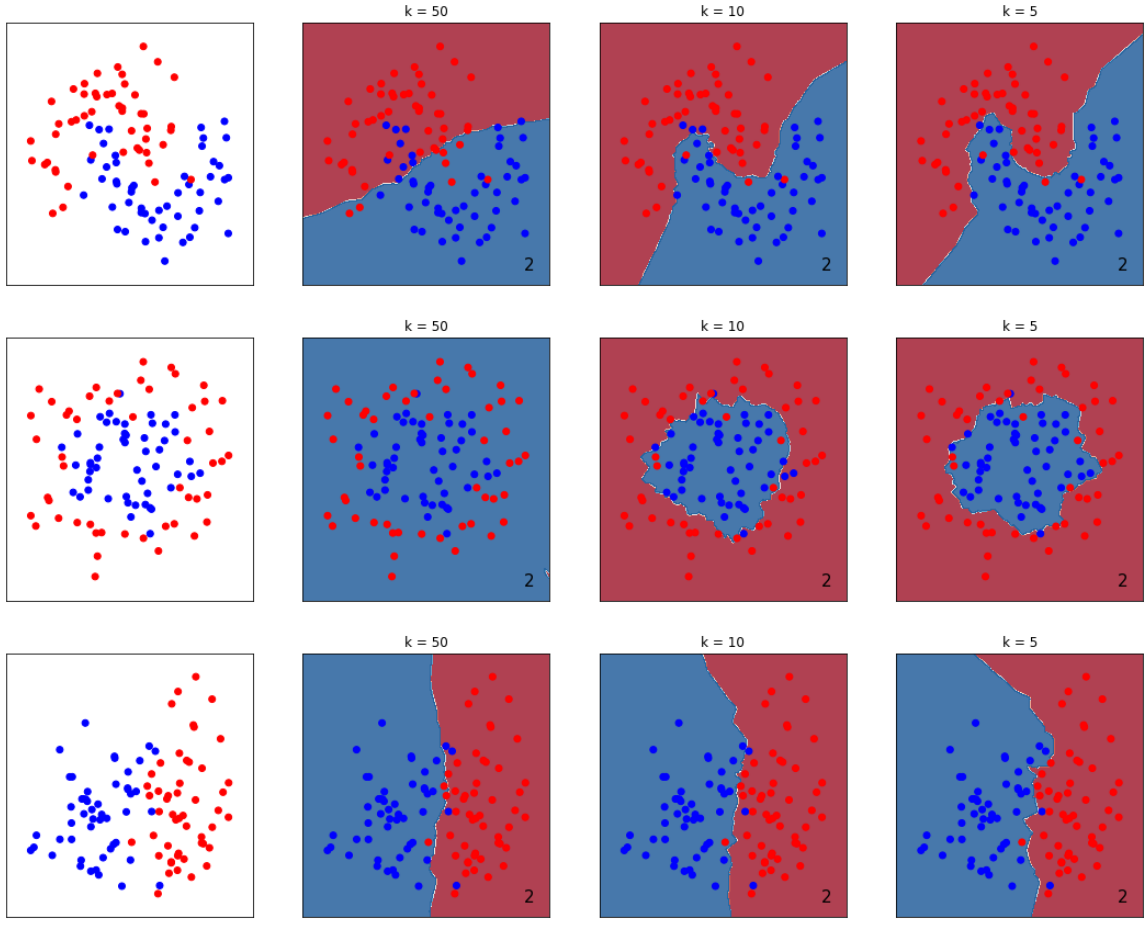
On peut donc poser

$$A_m = \{x \in \mathcal{X}, \mathcal{I}_x = \phi^m\}.$$

k-NN



k-NN

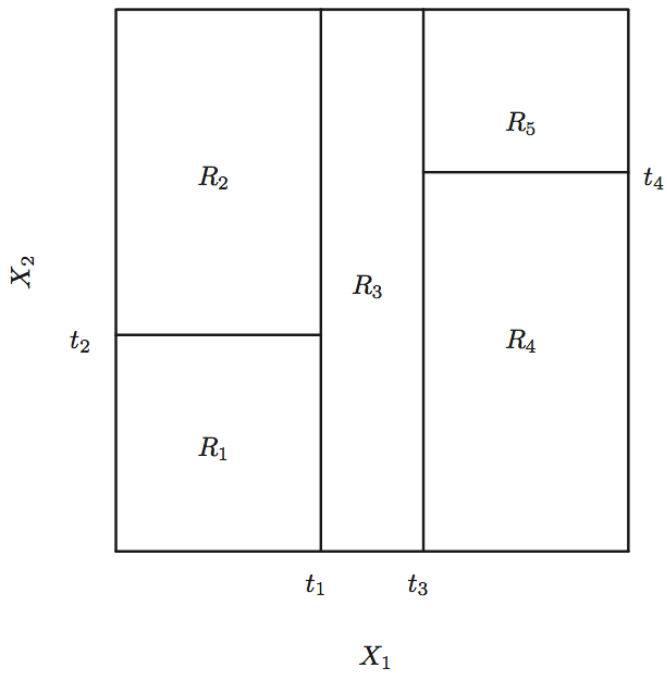


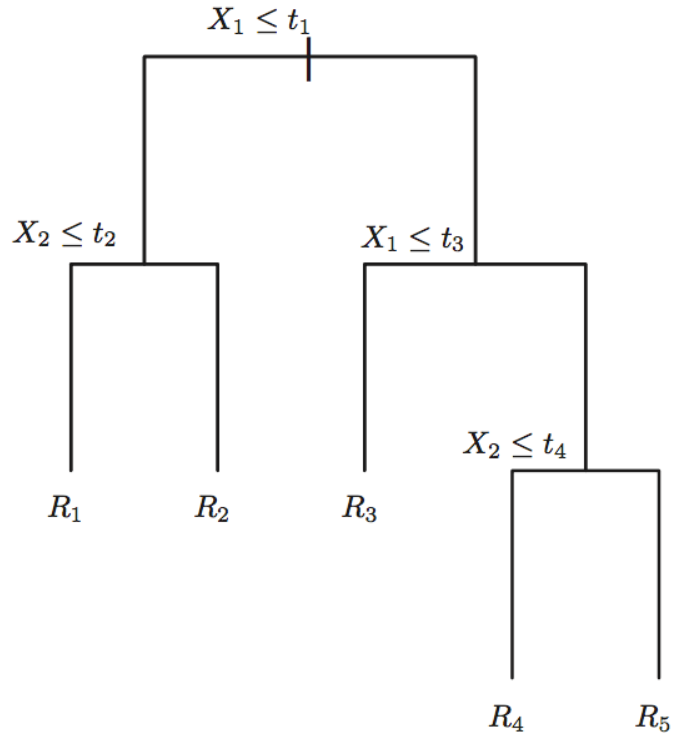
Plan

Construction de l'arbre

Approche "top-bottom"

- On commence par une région qui contient toutes les données
- On coupe récursivement les régions par rapport à une variable et une valeur de cette variable





Heuristique:

On veut choisir la valeur du “split” de telle sorte que les deux nouvelles régions sont les plus **homogènes** possible...

L'**homogénéité** peut se définir par différents critères

- la variance empirique
- l'indice de Gini
- l'entropie.

Arbre de classification à partir de l'indice de Gini

On coupe une région R en deux parties R_- and R_+. Pour chaque variable $j = 1, \ldots, p$ et chaque valeur de "split" t, on définit

$$R_-(j,t) = \{x \in R : x^j < t\} \quad \text{et} \quad R_+(j,t) = \{x \in R : x^j \geq t\}.$$

on cherche j et t qui minimisent

$$\text{Gini}(R_-) + \text{Gini}(R_+)$$

where

$$\text{Gini}(R) = \frac{1}{|\{i, x_i \in R\}|} \sum_{k \in \mathcal{C}} \hat{p}_{R,k}(1 - \hat{p}_{R,k})$$

où $\hat{p}_{R,k}$ est la proportion d'observations avec le label k dans l'ensemble des $\{i, x_i \in R\}$.

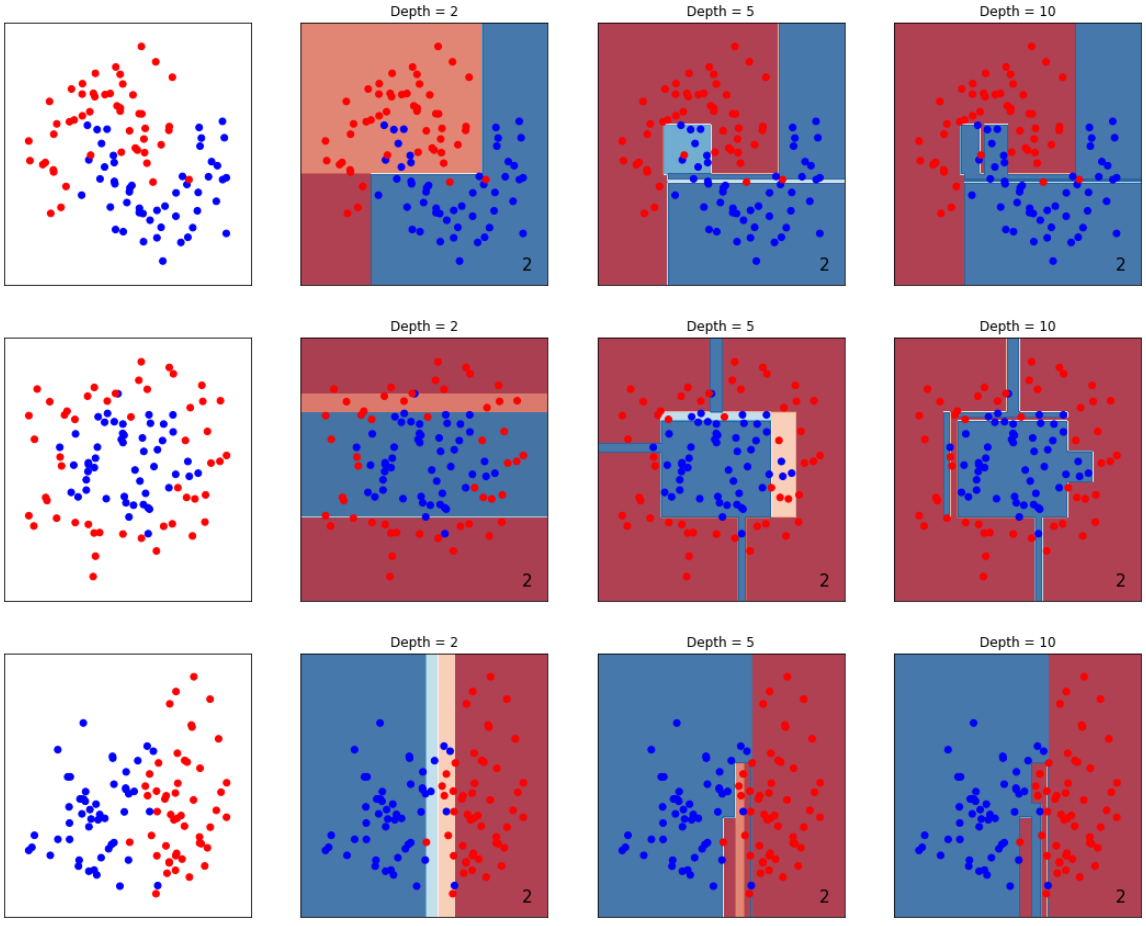
Algorithmes CART, C.4.5

- l'algorithme CART utilise l'indice de Gini
- l'algorithme C.4.5 (pas implementé dans `sklearn`) utilise l'entropie

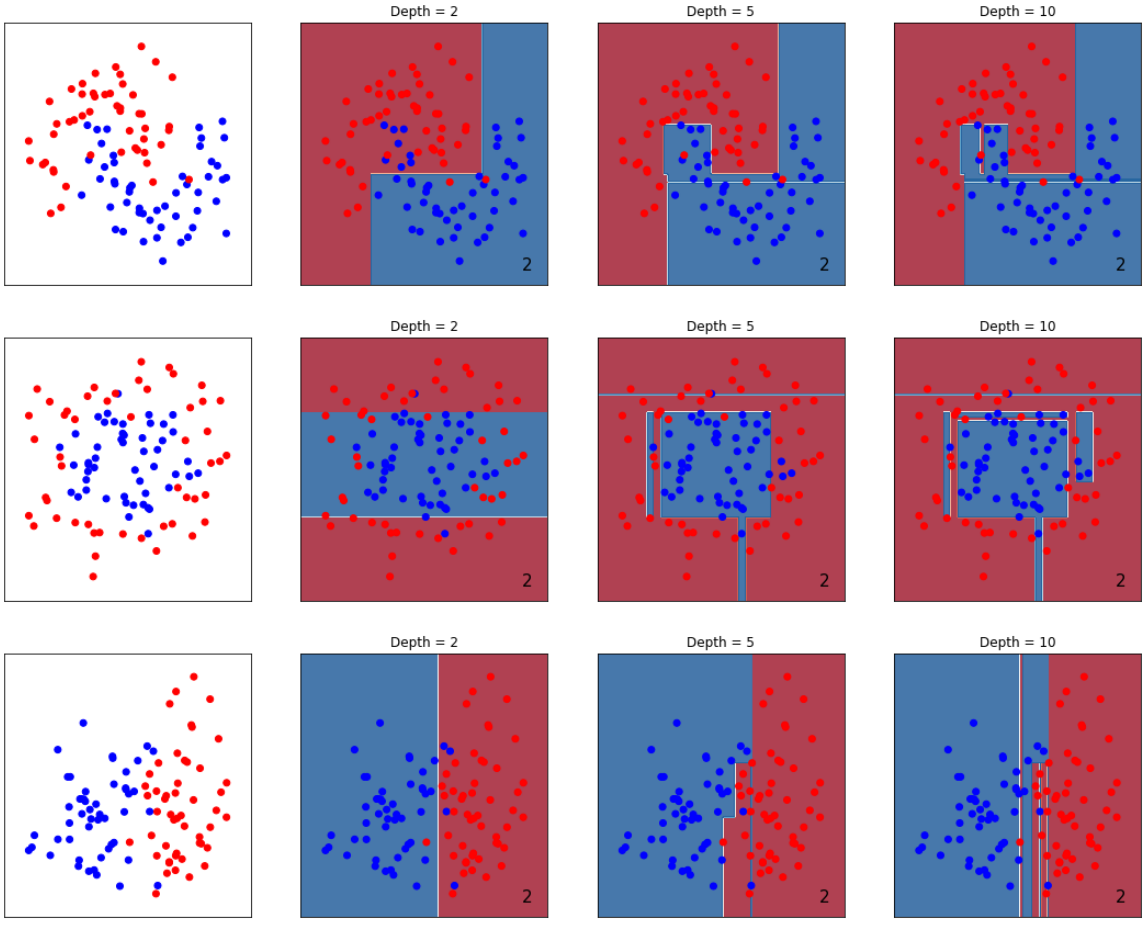
$$E(R) = -\sum_{k \in \mathcal{C}} \hat{p}_{R,k} \log(\hat{p}_{R,k})$$

- il y a d'autres critères possibles (χ^2, etc)

CART


Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10

CART


Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10
Depth = 2
Depth = 5
Depth = 10

Arbre de régression avec moindres carrés

On coupe une région R en deux parties R_- and R_+. Pour chaque variable $j = 1, \ldots, p$ et chaque valeur de "split" t, on définit

$$S(j,t) = \{x \in R : x_j < t\} \quad \text{and} \quad \bar{S}(j,t) = \{x \in R : x_j \geq t\}.$$

on cherche j et t qui minimisent

$$\sum_{i:x_i \in R_-(j,t)} (y_i - \bar{Y}_{R_-(j,t)})^2 + \sum_{i:x_i \in \bar{R}_+(j,t)} (y_i - \bar{Y}_{R_+(j,t)})^2$$

où

$$\bar{Y}_R = \frac{1}{|\{i : x_i \in R\}|} \sum_{i:x_i \in R} y_i.$$

Règles d'arrêt et algorithmes dérivés

Règles d'arrêt

On arrête l'algorithme quand

- l'arbre a atteint une taille maximale (fixée à l'avance)
- le nombre de feuilles atteint une valeur maximale (fixée à l'avance)
- quand les effectifs des noeuds terminaux atteignent une valeur minimale (fixée à l'avance)

En pratique

En pratique, ce sont des algorithmes instables et qui sur-apprennent, on les utilisent dans des algorithmes plus complexes qui "mélangent" des arbres

- les forêts alétoires (random forests)
- le boosting.

Plan

Classifieurs faibles / weak learners

Weak learners

- On considère un ensemble de classifieurs faibles (weak learners) $\mathcal{H}$ tels que
- chaque classifieur $c : \mathbb{R}^d \to \{-1, 1\}$ est très simple (par exemple un petit arbre)
- Bagging = Bootstrap Aggregating : on combine des classifieurs calculés à partir d'échantillons bootstrapés.
- Le bagging et les forêts aléatoires font partie des **méthodes d'ensemble/ensemble methods** puisqu'ils combinent des learners faibles pour en fabriquer un meilleur.

Bootstrap d'Efron (1)

A partir des données $\mathcal{D}_n = \Big\{(x_1, y_1), \ldots, (x_n, y_n)\Big\}$, on construit des

$$\mathcal{D}_1^\star = \Big((x_{1,1}^\star, y_{1,1}^\star), \ldots, (x_{1,n}^\star, y_{1,n}^\star)\Big)$$

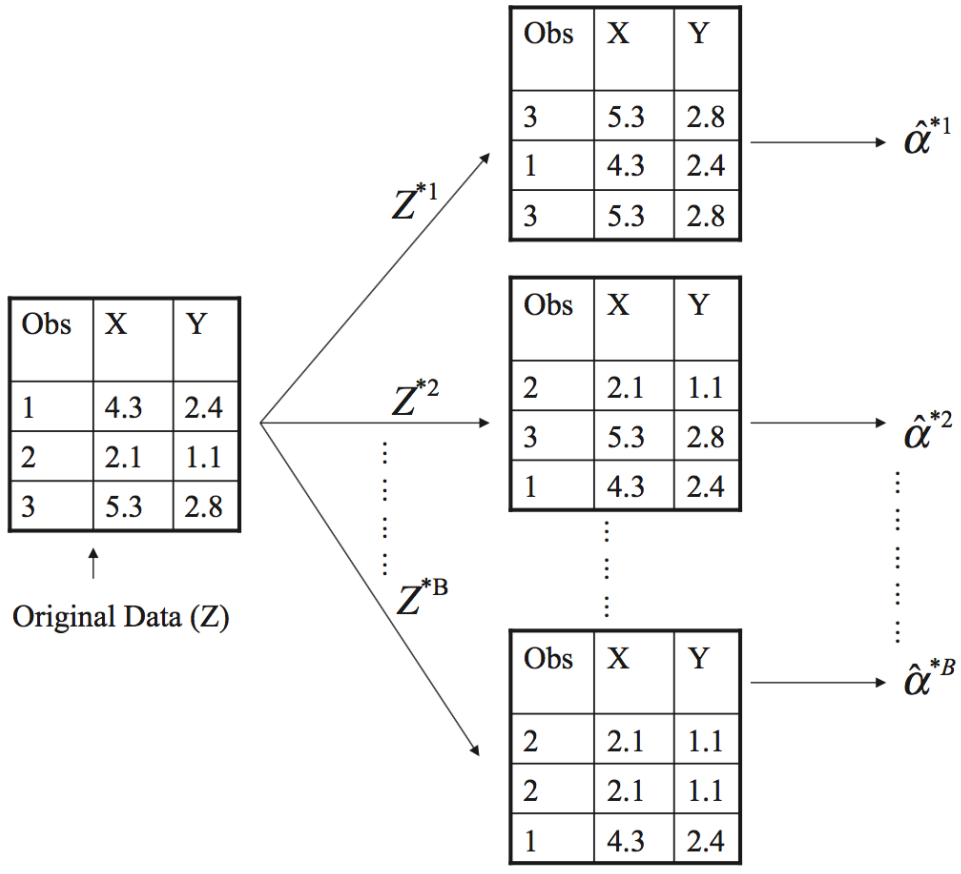
$$\ldots$$

$$\mathcal{D}_b^\star = \Big((x_{b,1}^\star, y_{b,1}^\star), \ldots, (x_{b,n}^\star, y_{b,n}^\star)\Big)$$

$$\ldots$$

en tirant les $(x_{b,i}^\star, y_{b,i}^\star)$ aléatoirement et avec remise dans $\mathcal{D}_n$, voir Efron 1982.

Bootstrap d'Efron (2)



Bootstrap d'Efron (3)

A partir de chaque échantillon bootstrapé $\mathcal{D}_b^\star$, on construit un classifieur faible $\hat{c}_b^\star$:

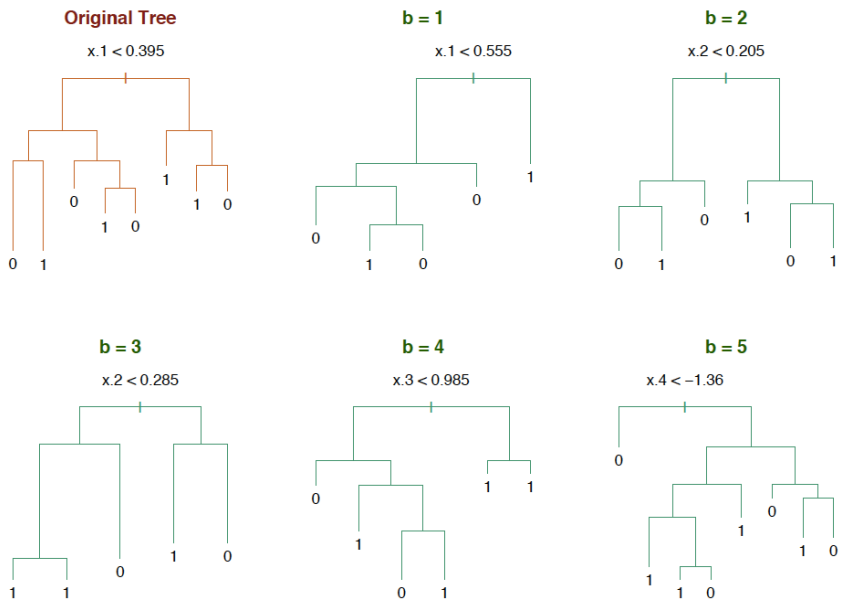


Figure 1: From J. Friedman, Hastie, and Tibshirani 2001

Bagging

On forme alors l'agrégation des classifieurs faibles calculés sur les échantillons bootstrapés (see Breiman 1996)

$$, \frac{1}{B} \sum_{b=1}^{B} \hat{c}_b^{\star}$$

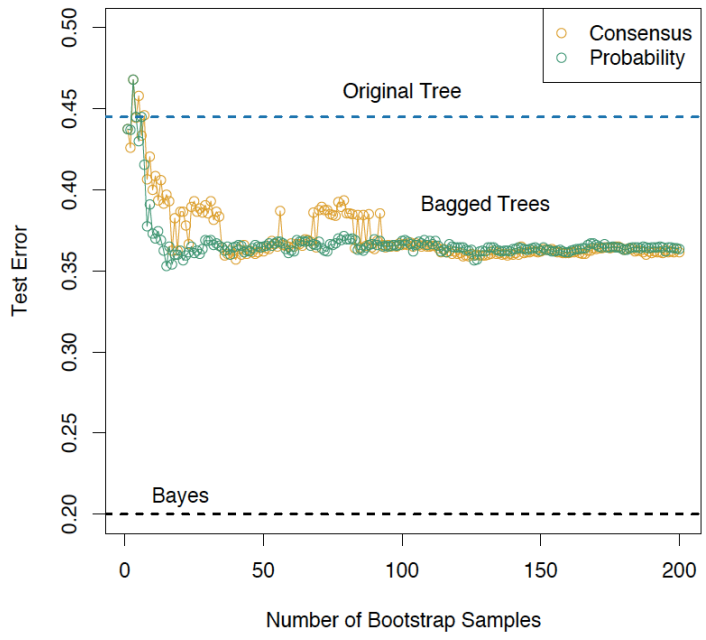


Figure 2: From J. Friedman, Hastie, and Tibshirani 2001

Plan

Algorithme des forêts aléatoires

Il est clair que les $\hat{c}_b^\star$ sont dépendants, on risque donc d'augmenter la variance. En effet, si $Z_1, \ldots, Z_B$ sont i.d. avec une corrélation de ρ (que l'on va supposer positive) alors la variance de $1/B \sum_1^B Z_b$ est donnée par

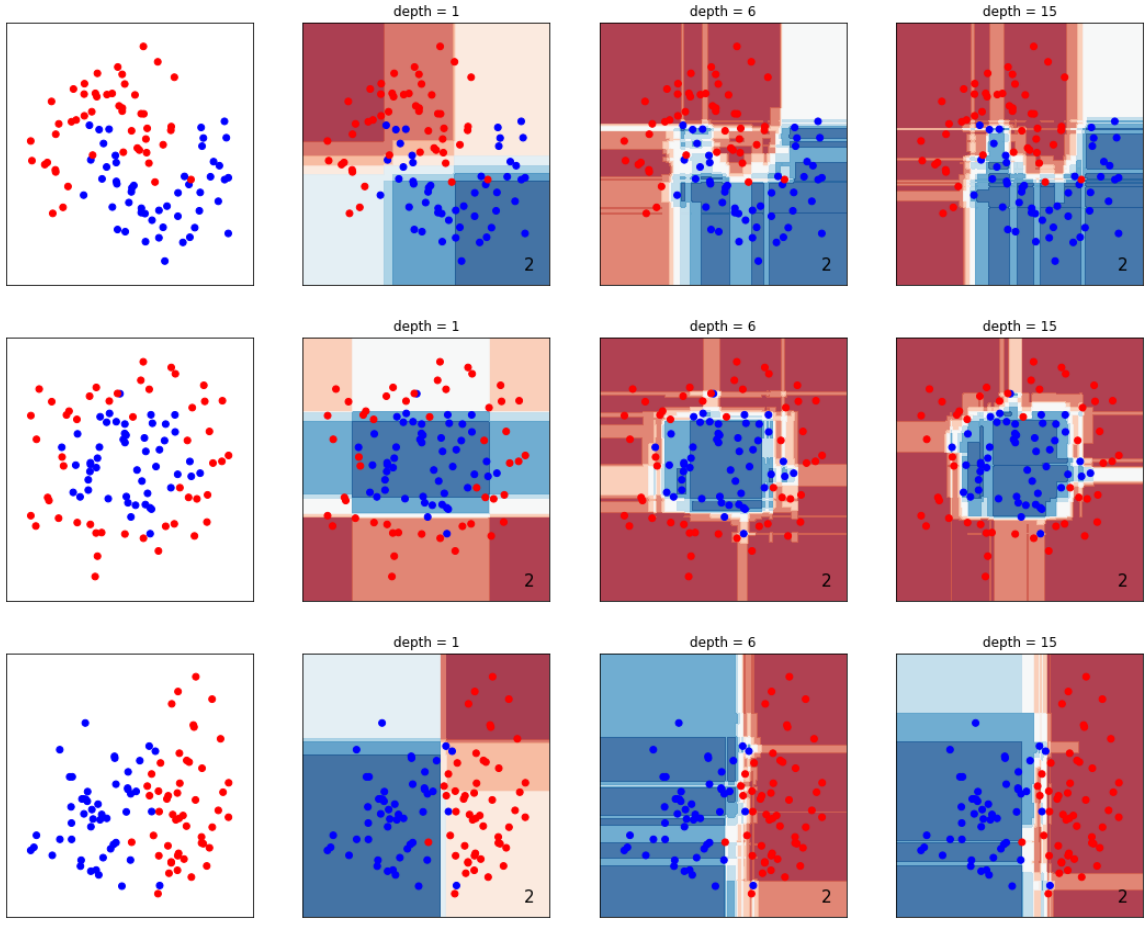
$$\rho \mathbb{V}(Z_b) + \frac{1-\rho}{B} \mathbb{V}(Z_b).$$

Il faut donc faire quelque chose pour "décorréler" les arbres : l'idée est de ne considérer qu'un sous-ensemble aléatoire des features à chaque slipt.

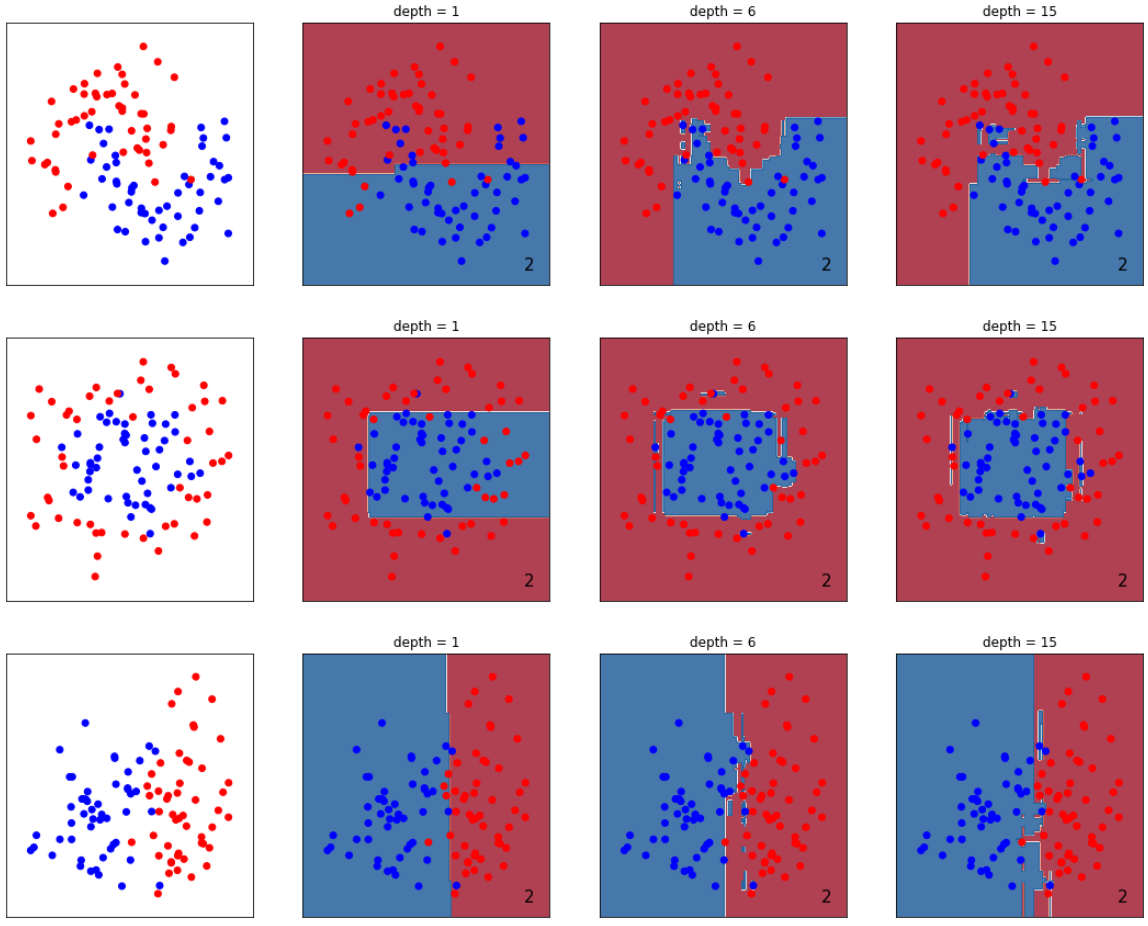
for $b = 1, \ldots, B$ **do**
  Tirer un échantillon bootstrapé $\mathcal{D}_b^\star$ à partir de $\mathcal{D}_n$.;
  Construire sur $\mathcal{D}_b^\star$ un arbre $\widehat{c}^b$ en tirant aléatoirement p variables parmi les d à chaque split.;
end
Result: Combiner les prédictions des B arbres par un vote à la majorité (ou une moyenne)

Algorithm 1: Algorithme des forêts aléatoires

Random forests



Random forests



Recommandations :

- Classification : la valeur par défaut pour p est $\lfloor \sqrt{d} \rfloor$ et la taille minimale d'une feuille est 1.
- Régression : la valeur par défaut pour p est $\lfloor d/3 \rfloor$ et la taille minimale d'une feuille est 5.

En pratique : on fait une cross-validation pour trouver de bons paramètres. Les hyper-paramètres d'une forêt sont donc

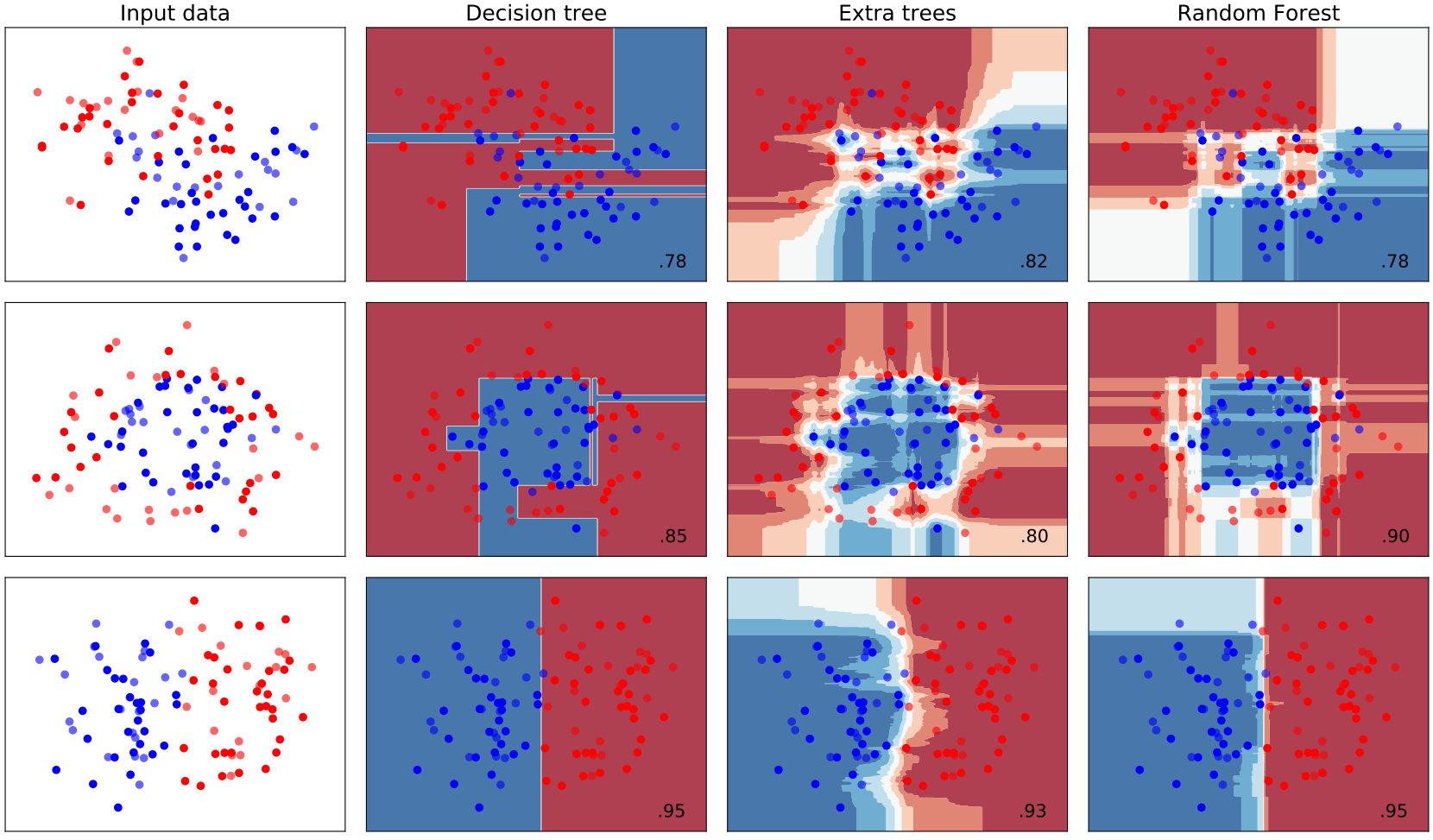
- B le nombre d'arbres dans la forêt
- la taille de chaque arbre ou la taille minimale d'une feuille
- p le nombre de variables à considérer

Une variante des forêts aléatoires : ExtraTrees

Cet algorithme s'appelle ExtraTrees pour Extremely randomized trees. On ne considère plus forcément le bootstrap. Pour chaque arbre et chaque split :

- On choisit un sous-ensemble aléatoire de features
- On sélectionne uniformément un petit nombre de splits possibles pour chaque variables (uniformément sur l'intervalle de valeurs observées)
- On choisit le meilleur split parmi ceux possibles.

Cela accélère beaucoup l'algorithme.


Input data
Decision tree
Extra trees
Random Forest
.78
.82
.78
.85
.80
.90
.95
.93
.95

Plan

Le boosting

Weak learners

- Considérons un ensemble de "weak learners" ou dictionnaire $\mathcal{H}$
- Chaque learner $h : \mathbb{R}^d \to \mathbb{R}$ ou $\mathbb{R}^d \to \{-1, 1\}$ est un learner très simple
- Chaque learner simple est à peine meilleur que celui appris avec les y_i (moyenne)

Exemples de weak learners

- Pour la régression : arbres de régression simple de faible profondeur, glm à quelques variables
- Pour la classification : arbres de décision simple de faible profondeur, modèles logistiques à quelques variables.

Principe du boosting

Un "strong learner"

On combine additivement des weak learners

$$g^{(B)}(x) = \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)$$

avec $\eta^{(b)} \geq 0$ pour espérer en obtenir un meilleur.

- Chaque $b = 1, \ldots, B$ est un pas/itération de boosting
- Le boosting fait partie des **méthodes d'ensemble/ensemble methods** puisqu'il combine des learners faibles pour en fabriquer un meilleur.

Pour aller plus loin : voir Schapire 1999 et J. H. Friedman 2001

Principe du "gradient boosting"

On cherche donc des fonctions $h^{(b)}$ du dictionnaire $\mathcal{H}$ et des réels $\eta^{(b)}$ tels que

$$g^{(B)}(x) = \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x)$$

minimise le risque empirique

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

où ℓ est la fonction de coût (de perte) que l'on se fixe suivant le problème

- en régression linéaire $\ell(y, u) = (1/2)(y - u)^2$
- plus généralement, on peut prendre ℓ comme moins la log-densité dans le modèle considéré.

L'algorithme "greedy"

Si c'était possible, on pourrait définir

$$\hat{g}^{(B)} = \sum_{b=1}^{B} \hat{\eta}^{(b)} \hat{h}^{(b)}(x)$$

avec

$$(\hat{\eta}^{(1)}, \ldots, \hat{\eta}^{(B)}, \hat{h}^{(1)}, \ldots, \hat{h}^{(B)}) = \underset{\eta^{(1)}, \ldots, \eta^{(B)} \in \mathbb{R}, h^{(1)}, \ldots, h^{(B)} \in \mathcal{H}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \sum_{b=1}^{B} \eta^{(b)} h^{(b)}(x))$$

mais même à $\eta^{(1)}, \ldots, \eta^{(B)}$ fixés il faut chercher parmi $\#(\mathcal{H})^B$ solutions possibles.

Plan

Algorithme "greedy-stagewise"

On procède en fait pas-à-pas en définissant une suite $\hat{g}^{(b)}$ avec pour tout $b = 1, \ldots, B$ avec

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \hat{\eta}^{(b+1)} \hat{h}^{(b+1)} \text{ où}$$

$$(\hat{\eta}^{(b+1)}, \hat{h}^{(b+1)}) = \underset{\eta \in \mathbb{R}, h \in \mathcal{H}}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{g}^{(b)}(x_i) + \eta h(x_i))$$

Dans nos problèmes, cette minimisation est également un problème difficile, on va alors procéder

- par descente de gradient
- avec la contrainte que la direction du pas de descente soit une fonction de $\mathcal{H}$.

Rappel

Descente de gradient

Considérons le problème de la minimisation de la fonction convexe et différentiable $J : \mathbb{R}^p \to \mathbb{R}$, la suite d'itérés $\theta^{(b)}$ définis par

$$\theta^{(b+1)} = \theta^{(b)} - \gamma \nabla J(\theta^{(b)})$$

alors

$$J(\theta^{(b+1)}) \leq J(\theta^{(b)})$$

(sous des conditions sur J et γ - cf cours d'optimisation).

Descente non-contrainte

A l'itération b,

- nous avons le learner $\hat{g}^{(b)} = \hat{g}^{(b)} + \eta\, \vec{0}$,
- on cherche à faire un pas de descente de gradient (pour l'instant quelconque) pour la fonction à optimiser $u \mapsto \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, \hat{g}^{(b)}(x_i) + \eta u(x_i))$.

Attention : u est une fonction de $\mathbb{R}^p \to \mathbb{R}$. Mais on ne s'intéresse qu'à ses valeurs aux points $x_1, \ldots, x_n$, on l'identifie donc à vecteur de $\mathbb{R}^n$

$$\left(\frac{1}{n} \nabla_{u_i} \sum_{i=1}^{n} \ell(y_i, \hat{g}^{(b)}(x_i) + \eta u_i) \right) = \eta \left(\frac{1}{n} \nabla_{y'} \ell(y_i, \hat{g}^{(b)}(x_i) + \eta u_i) \right)$$

en considérant $(y, y') \mapsto \ell(y, y')$.

Descente non-contrainte à rester dans $\mathcal{H}$

Le pas de gradient à considérer est donc dans la direction

$$\delta^{(b+1)} = \frac{\eta}{n} \begin{pmatrix} \left(\nabla_{y'} \ell(y_1, \hat{g}^{(b)}(x_1) + \eta u_1) \right)_0 \\ \cdots \\ \left(\nabla_{y'} \ell(y_n, \hat{g}^{(b)}(x_n) + \eta u_n) \right)_0 \end{pmatrix} = \frac{\eta}{n} \begin{pmatrix} \nabla_{y'} \ell(y_1, \hat{g}^{(b)}(x_1)) \\ \cdots \\ \nabla_{y'} \ell(y_n, \hat{g}^{(b)}(x_n)) \end{pmatrix}$$

Exemple dans le modèle linéaire

Dans le modèle linéaire, on prend

- $\ell(y, y') = \frac{1}{2}(y - y')^2$ avec
- $\nabla_{y'} \ell(y, y') = -(y - y')$.

Le pas de gradient est donc dans la direction

$$\delta_{lm}^{(b+1)} = \frac{\eta}{n} \begin{pmatrix} \nabla_{y'} \ell(y_1, \hat{g}^{(b)}(x_1)) \\ \dots \\ \nabla_{y'} \ell(y_n, \hat{g}^{(b)}(x_n)) \end{pmatrix} = \frac{\eta}{n} \begin{pmatrix} -(y_1 - \hat{g}^{(b)}(x_1)) \\ \dots \\ -(y_n - \hat{g}^{(b)}(x_n)) \end{pmatrix}.$$

Retour au problème contraint

Le problème d'optimisation au départ est

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \hat{\eta}^{(b+1)}\hat{h}^{(b+1)} \text{ où}$$

$$(\hat{\eta}^{(b+1)}, \hat{h}^{(b+1)}) = \underset{\eta\in\mathbb{R}, h\in\mathcal{H}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \hat{g}^{(b)}(x_i) + \eta h(x_i))$$

il faudrait donc que $\delta^{(b+1)}$ soit dans $\mathcal{H}$, ce qui ne peut pas être assuré à cette étape.

Pour s'assurer de rester dans le dictionnaire $\mathcal{H}$, on va prendre la fonction de $\mathcal{H}$ la plus proche de $\delta^{(b+1)}$ au sens des moindres carrés (de la norme ℓ_2 aux points d'observation):

$$\hat{h}^{(b+1)} = \hat{h} \text{ avec } (\hat{h}, \hat{\nu}) = \underset{h\in\mathcal{H}, \nu\in\mathbb{R}}{\operatorname{argmin}} \frac{1}{n}\sum_{i=1}^{n} \left(\delta^{(b+1)}(i) - \nu h(x_i)\right)^2.$$

La contrainte dans le modèle linéaire

Dans le modèle linéaire, cela s'écrit

$$\hat{h}^{(b+1)} = \hat{h} \text{ avec } (\hat{h}, \hat{\nu}) = \underset{h \in \mathcal{H}, \nu \in \mathbb{R}}{\text{argmin}} \frac{2}{n} \sum_{i=1}^{n} \left(\delta^{(b+1)}(i) - \nu \{ -(y_i - \hat{g}^{(b)}(x_i)) \} \right)^2.$$

On essaie donc d'apprendre un modèle (faible) sur les résidus $(-(y_i - \hat{g}^{(b)}(x_i)))$ du modèle précédent : aux points où $g^{(b)}$ n'est pas très performant (grand résidus).

Algorithme du gradient boosting

On optimise enfin en η pour obtenir l'algorithme suivant.

Data: Posons $\hat{g}^{(0)} = \hat{a}$ avec $\hat{a} = \text{argmin}_{a\in\mathbb{R}} \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, a)$

for $b = 1, \ldots, B$ **do**

$$\delta^{(b+1)} \leftarrow -\Big(\nabla_u \ell(y_i, \hat{g}^{(b)}(x_i) + \eta u)\Big)_0 \quad i = 1, \ldots, n;$$

$$\hat{h}^{(b+1)} \leftarrow \hat{h} \text{ avec } (\hat{h}, \hat{\nu}) = \text{argmin}_{h\in\mathcal{H},\nu\in\mathbb{R}} \sum_{i=1}^{n} \Big(\nu h(x_i) - \delta^{(b+1)}(i)\Big)^2;$$

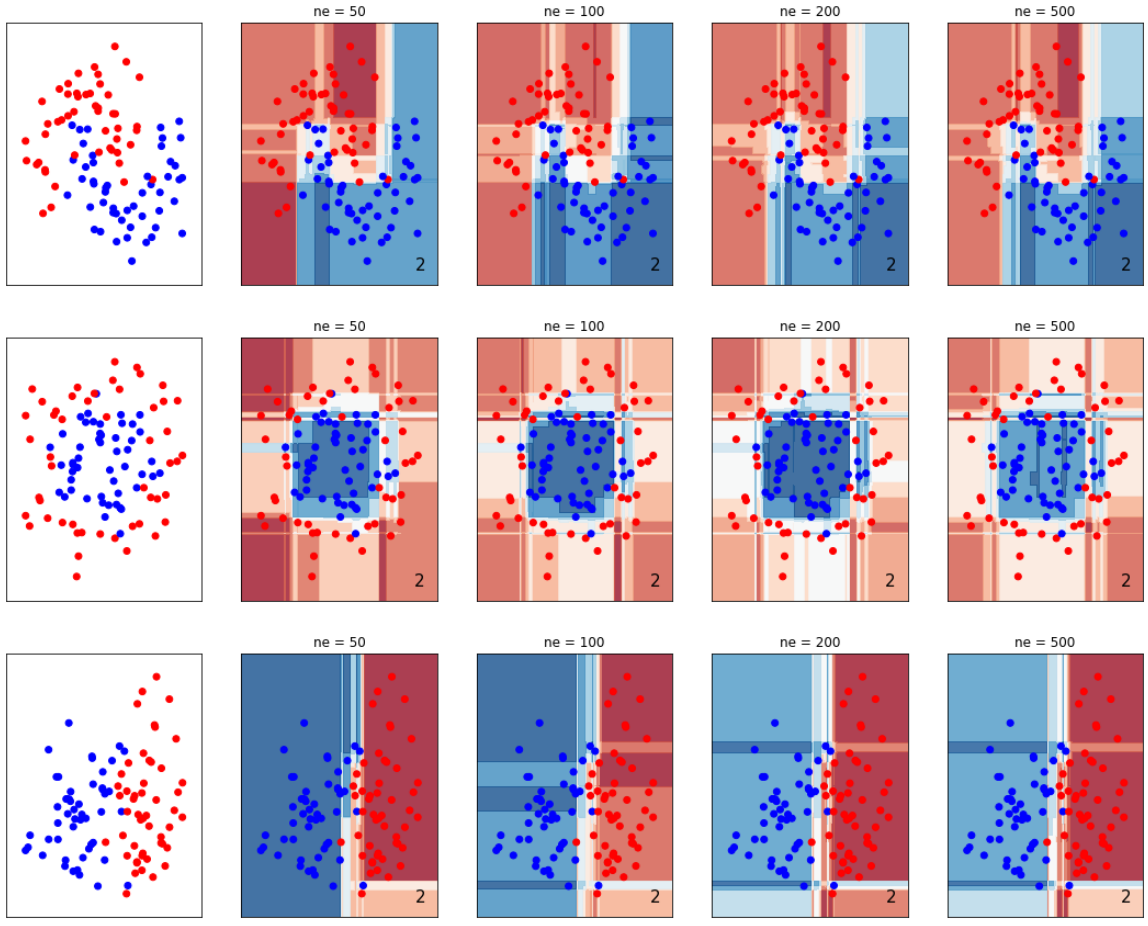
$$\hat{\eta}^{(b+1)} \leftarrow \text{argmin}_{\eta\in\mathbb{R}} \sum_{i=1}^{n} \ell(y_i, \hat{g}^{(b)}(x_i) + \eta h^{(b+1)}(x_i));$$

end

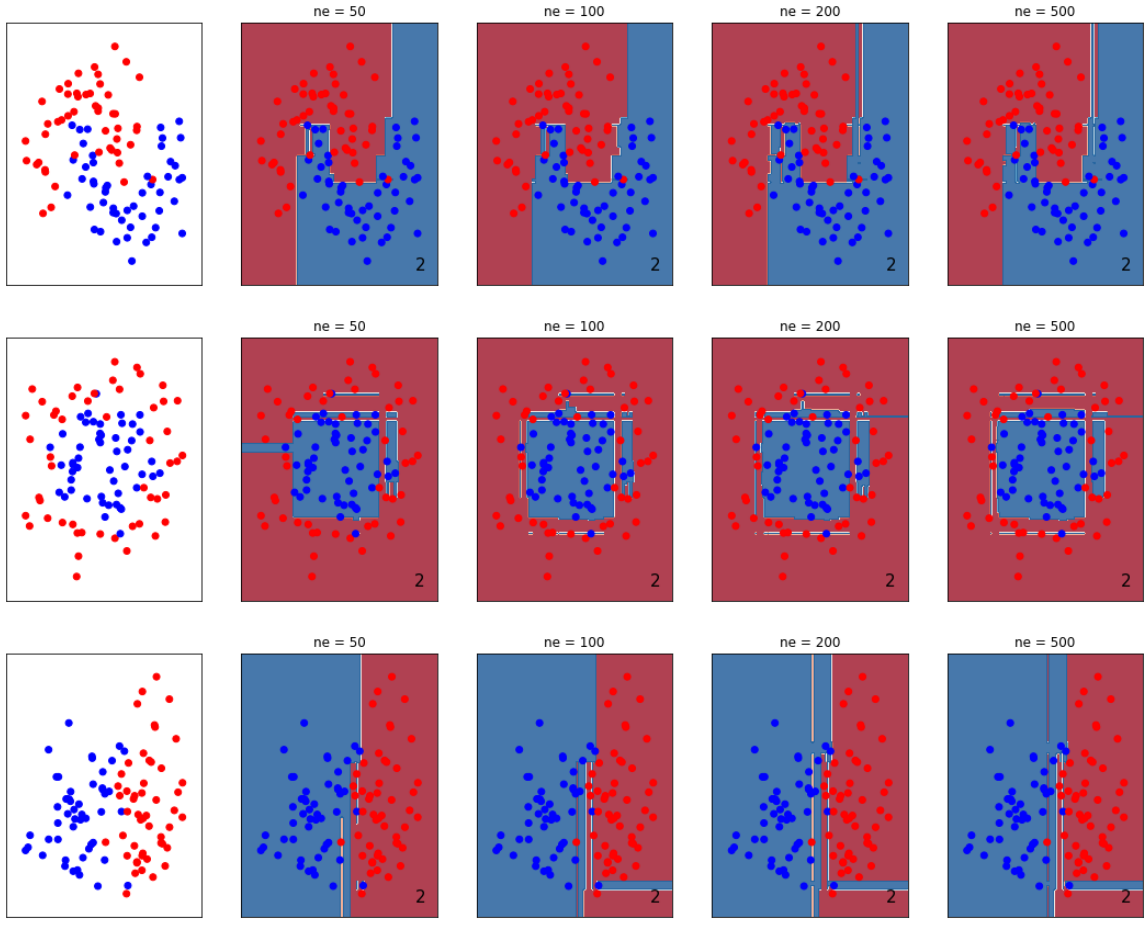
return *Boosting learner* $\widehat{g}^{(B)}(x) = \sum_{b=1}^{B} \widehat{\eta}^{(b)}\widehat{h}^{(b)}(x)$

Algorithm 2: Gradient boosting

Boosting



Boosting



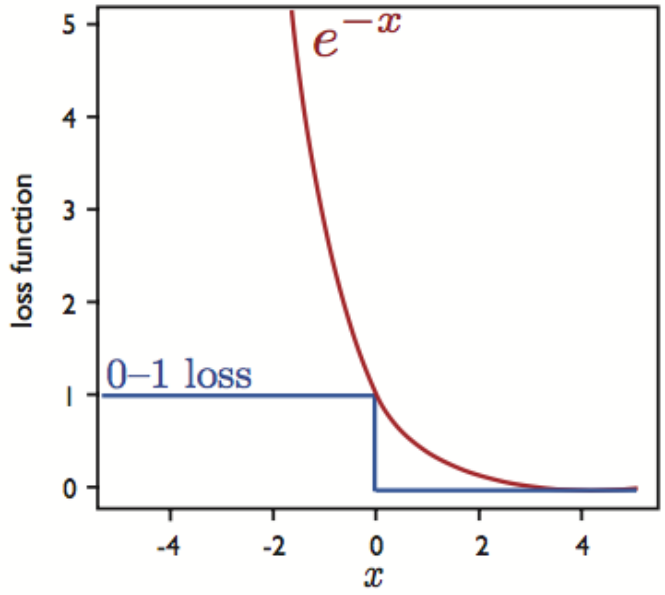
Plan

Pour la classification

L'algorithme de boosting le plus connu en classification est AdaBoost (ADAptive BOOSTing). Il optimise la perte exponentielle

$$\ell(y, u) = \exp(-yu)$$

qui est une approximation de la perte 0/1 $\mathbb{1}_{y \neq u}$.



On va montrer qu'on peut aussi le voir comme un algorithme qui pondère séquentiellement les observations mal-classées.

A l'itération b (1)

On définit donc

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \hat{\eta}^{(b+1)}\hat{h}^{(b+1)} \text{ où}$$

$$(\hat{\eta}^{(b+1)}, \hat{h}^{(b+1)}) = \underset{\eta\in\mathbb{R}, h\in\mathcal{H}}{\text{argmin}}\ R_n(h,\eta) = \underset{\eta\in\mathbb{R}, h\in\mathcal{H}}{\text{argmin}}\ \frac{1}{n}\sum_{i=1}^{n}\ell(y_i, \hat{g}^{(b)}(x_i) + \eta h(x_i))$$

mais

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\ell(y_i, \hat{g}^{(b)}(x_i) + \eta h(x_i)) &= \frac{1}{n}\sum_{i=1}^{n}\exp(-y_i\hat{g}^{(b)}(x_i) + \eta h(x_i)) \\ &= \frac{1}{n}\sum_{i=1}^{n}\exp(-y_i\hat{g}^{(b)})(x_i)\exp(-y_i\eta h(x_i)) \\ &= \frac{1}{n}\sum_{i=1}^{n}w^{(b)}(i)\exp(-y_i\eta h(x_i)).\end{aligned}$$

avec $w^{(b)}(i) \propto exp(-y_i\hat{g}^{(b)}(x_i))$.

A l'itération b (2)

$$\frac{1}{n}\sum_{i=1}^{n} w^{(b)}(i)\exp(-y_i\eta h(x_i)) = \sum_{i:y_i h(x_i)=1} w^{(b)}(i)e^{-\eta} + \sum_{i:y_i h(x_i)=-1} w^{(b)}(i)e^{\eta}$$
$$= e^{-\eta} + (e^{\eta} - e^{-\eta}) \sum_{i:y_i h(x_i)=-1} w^{(b)}(i)$$

L'optimisation en h donne donc

$$\widehat{h}^{(b+1)} = \operatorname*{argmin}_{h\in\mathcal{H}} \sum_{i:y_i h(x_i)=-1} w^{(b)}(i) = \operatorname*{argmin}_{h\in\mathcal{H}} \sum_{i=1}^{n} w^{(b)}(i)\mathbb{1}_{y_i h(x_i)<0}$$

c'est le classifieur qui minimise l'erreur de prédiction repondérée.

A l'itération b (3)

L'optimisation en η donne donc

$$\widehat{\eta}^{(b+1)} = \frac{1}{2} \log \left(\frac{1 - \varepsilon(b+1)}{\varepsilon(b+1)} \right)$$

où

$$\varepsilon(b+1) = \sum_{i=1}^{n} w^{(b)}(i) \mathbb{1}_{y_i \hat{h}^{(b+1)}(x_i) < 0}$$

.

On peut donc écrire l'update des poids

$$\begin{aligned} w^{(b+1)}(i) &\propto \exp(-y_i \hat{g}^{(b+1)}(x_i)) = \exp(-y_i \hat{g}^{(b)}(x_i)) \exp(-y_i \hat{\eta}^{(b+1)} \hat{h}^{(b+1)}(x_i)) \\ &= \exp w^{(b+1)}(i) \exp(-y_i \hat{\eta}^{(b+1)} \hat{h}^{(b+1)}(x_i)). \end{aligned}$$

.

Data: Posons $w^{(0)}(i) = 1/n$ pour $i = 1, \ldots, n$

for $b = 1, \ldots B$ **do**

$h^{(b)} \in \text{argmin}_{h \in H} \sum_{i=1}^{n} w^{(b-1)}(i) \mathbb{1}_{y_i h(x_i) < 0}$;

$\varepsilon^{(b)} \leftarrow \sum_{i=1}^{n} w^{(b-1)}(i) \mathbb{1}_{y_i h^{(b)}(x_i) < 0}$;

$\eta^{(b)} \leftarrow \frac{1}{2} \log \left(\frac{1-\varepsilon^{(b)}}{\varepsilon^{(b)}} \right)$;

$w^{(b)}(i) \leftarrow w^{(b-1)}(i) e^{-\eta^{(b)} y_i h^{(b)}(x_i)}$; $w^{(b)}(i) \leftarrow w^{(b)}(i) / \sum_{i=1}^{n} w^{(b)}(i)$;

end

return *boosting classifieur* $\widehat{g}^{(B)}(x) = \sum_{b=1}^{B} \widehat{\eta}^{(b)} \widehat{h}^{(b)}(x)$

Algorithm 3: Adaboost

Plan

En pratique

Il reste à fixer les hyper-paramètres

- pour le dictionnaire $\mathcal{H}$:
  - si on choisit des arbres, il reste à fixer leurs profondeurs
  - si on choisir des glm, il reste à fixer le nombre de variables qu'ils contiennent
- et B le nombre d'itérations

Régularisation

Pour rendre la performance de l'algorithme moins dépendant du choix de B, on régularise en ajoutant le paramètre

$$\hat{g}^{(b+1)} = \hat{g}^{(b)} + \lambda \hat{\eta}^{(b+1)} \hat{h}^{(b+1)}$$

que l'on choisit à la fin par cross-validation.