Arbres Binaires de décision et extensions agrégées : Bagging, Boosting et Random Forests

F. Panloup
LAREMA-Université d'Angers

—

Cours : Apprentissage Statistique en Grande Dimension

—

Arbres binaires de décision

Construction d'un arbre binaire

La segmentation par arbre est une approche non-paramétrique de la classification ou de la régression.

- But : expliquer une variable réponse (qualitative ou quantitative) à l'aide d'autres variables.

- Principe : construire un arbre à l'aide de divisions successives de l'échantillon en deux segments (appelés aussi **noeuds**) homogènes par rapport à la réponse Y (qualitative ou quantitative) en utilisant l'information de p variables $X_1, \ldots, X_p$ (qualitatives ou quantitatives, ordonnables ou non).

Forme générale du résultat

L'arbre ainsi construit est structuré comme suit :

- Racine comportant l'échantillon complet.
- Branches ou segments divisant successivement les sous-échantillons.
- Feuilles ou Noeuds terminaux formant une partition de l'échantillon.

Un premier exemple

- Variables : Age, Revenu et Sexe
- Chaque division/coupe sépare chaque noeud en deux noeuds fils *plus homogènes que le noeud père*
- Homogènes en quel sens ? Définir un critère... (selon le type de réponse)

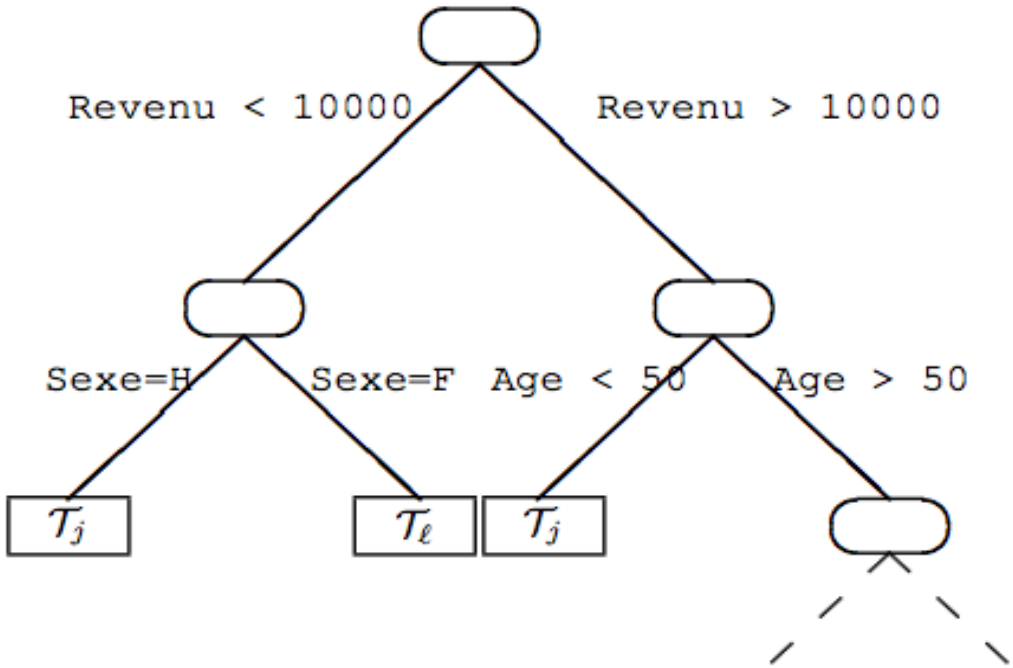


FIGURE 1 – Exemple élémentaire d'arbre de classification.

Partitionnement de l'espace

- Les feuilles de l'arbre (noeuds terminaux) forment une partition de l'espace.
- Cette partition va constituer la règle de classification ou de régression.
- Si X quantitatif, feuilles représentables via un partitionnement dyadique de l'espace (représentant l'échantillon global) :

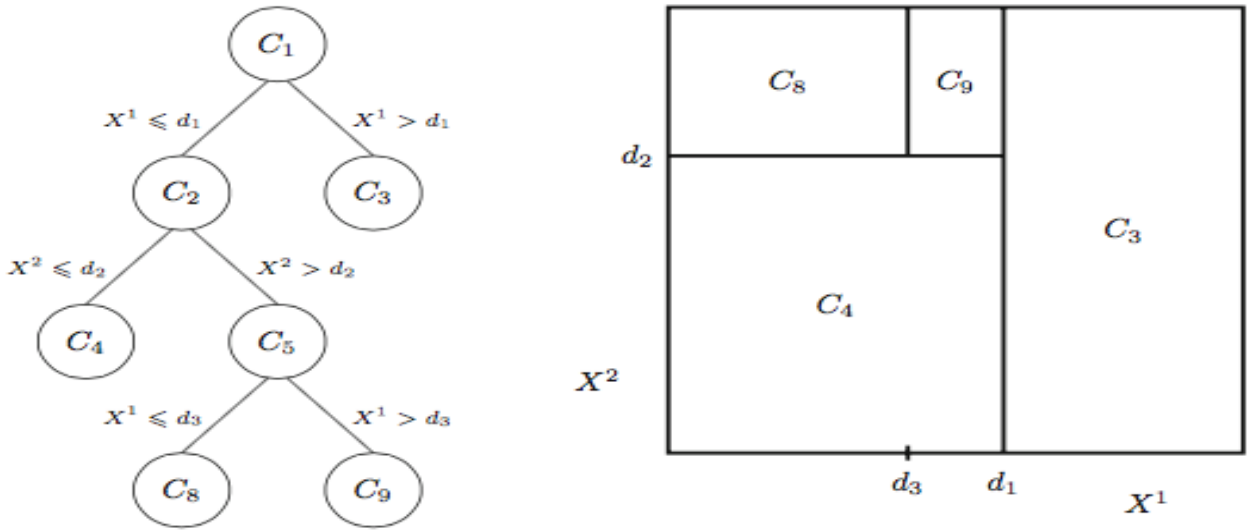
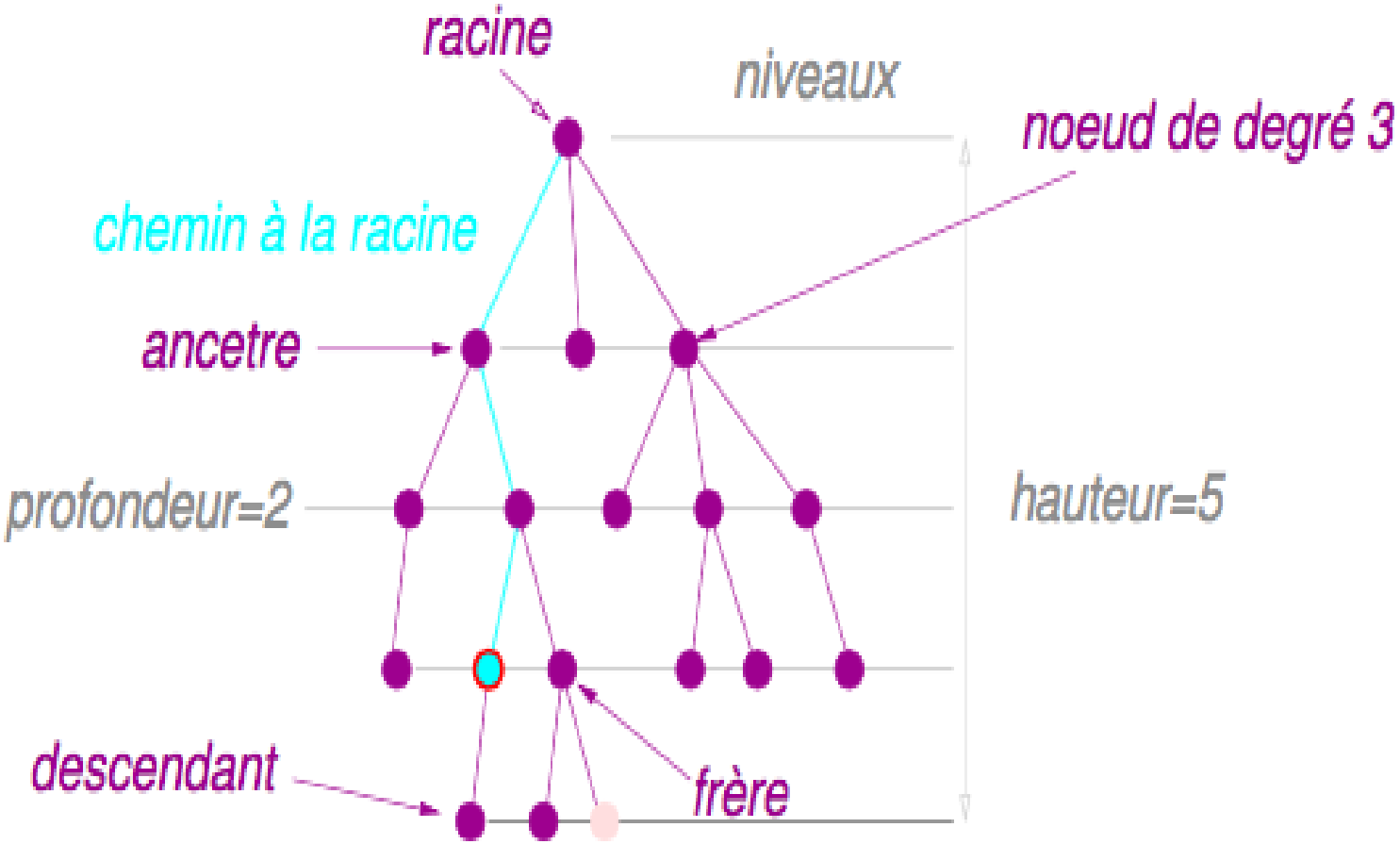


FIGURE 2 – *Construction d'un arbre avec variables explicatives quantitatives et pavage dyadique de l'espace. Chaque nœud père engendre deux fils conséquence d'une division ou coupe définie par le choix d'une variable : X^1 ou X^2 et d'une valeur seuil, successivement d_1, d_2, d_3. À chaque pavé de l'espace ainsi découpé, est finalement associée une valeur ou une classe de Y.*

Un peu de vocabulaire

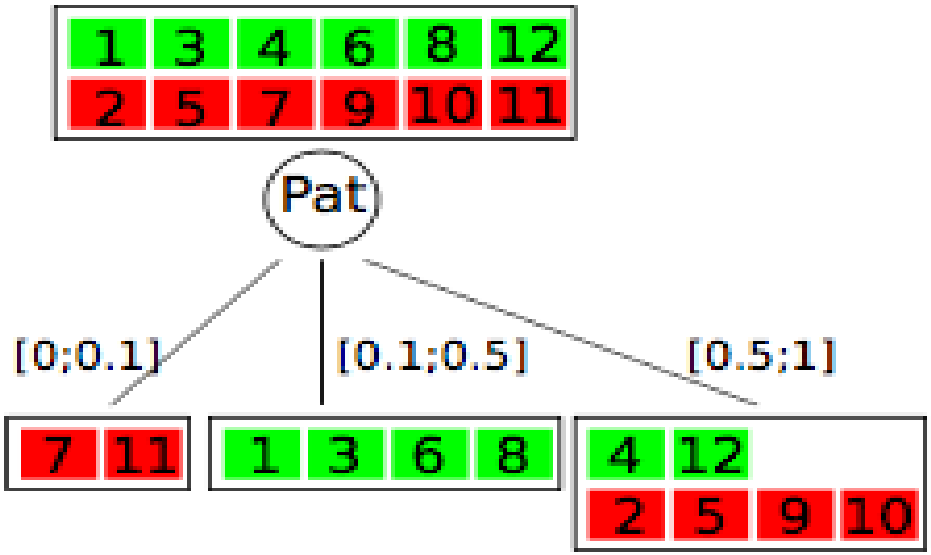


Un exemple

	Alt	Bar	F/S	Hun	Pat	Pri	Rai	Res	Typ	Dur	Wai
x_1	Y	N	N	Y	0.38	$$$	N	Y	French	8	Y
x_2	Y	N	N	Y	0.83	$	N	N	Thai	41	N
x_3	N	Y	N	N	0.12	$	N	N	Burger	4	Y
x_4	Y	N	Y	Y	0.75	$	Y	N	Thai	12	Y
x_5	Y	N	Y	N	0.91	$$$	N	Y	French	75	N
x_6	N	Y	N	Y	0.34	$$	Y	Y	Italian	8	Y
x_7	N	Y	N	N	0.09	$	Y	N	Burger	7	N
x_8	N	N	N	Y	0.15	$$	Y	Y	Thai	10	Y
x_9	N	Y	Y	N	0.84	$	Y	N	Burger	80	N
x_{10}	Y	Y	Y	Y	0.78	$$$	N	Y	Italian	25	N
x_{11}	N	N	N	N	0.05	$	N	N	Thai	3	N
x_{12}	Y	Y	Y	Y	0.89	$	N	N	Burger	38	Y

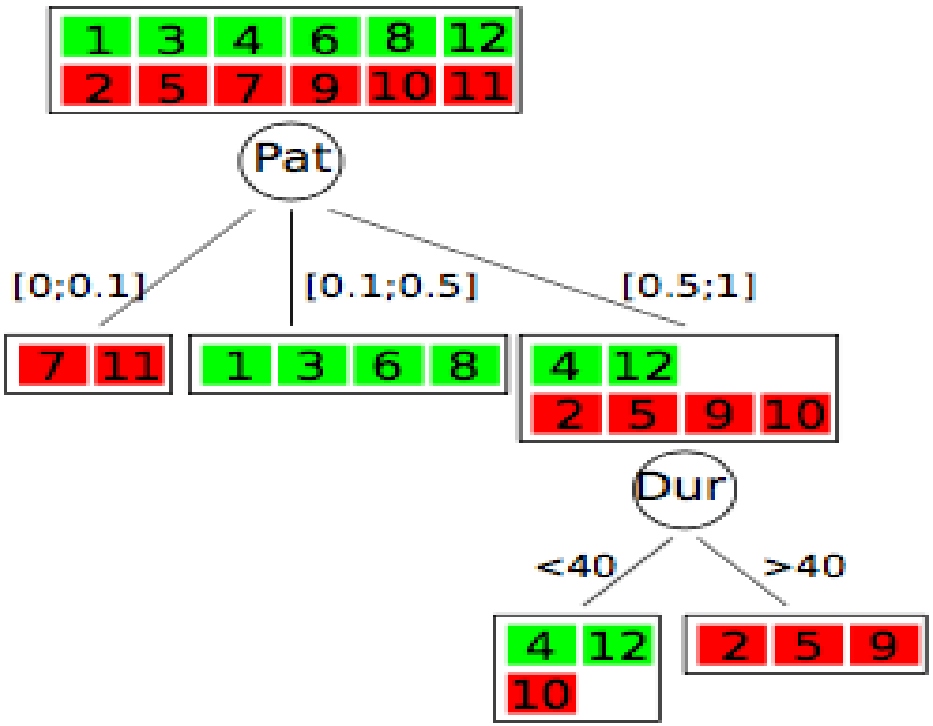
But : expliquer la variable *Wai* (pour “wait”) (ou déterminer un prédicteur de Wai).

Un exemple



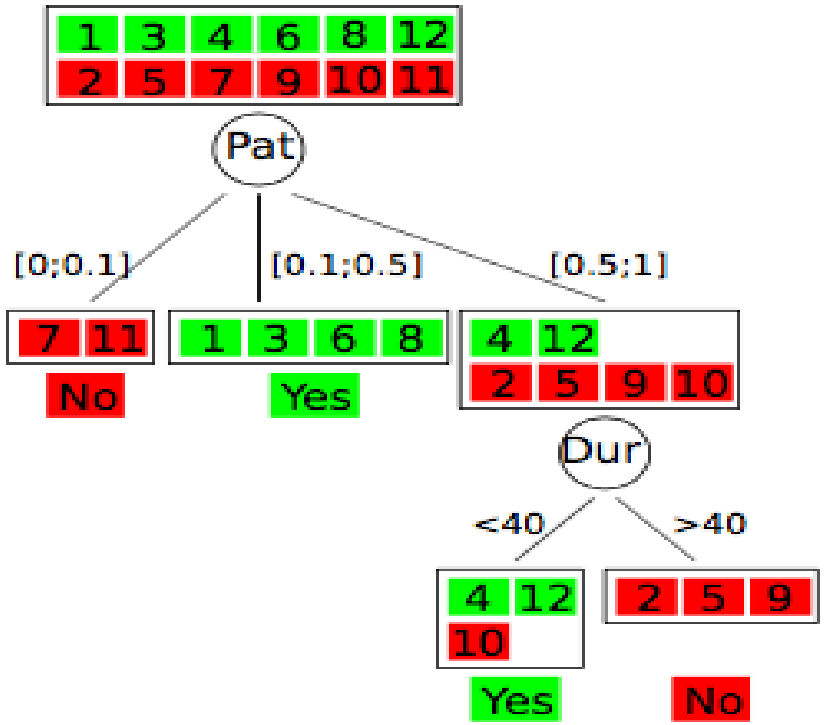
On choisit de séparer les données via la variable Pat en premier lieu (dans cet exemple, on autorise la division en 3, dans la suite, il n'y aura que des arbres binaires).

Un exemple



Les deux noeuds de niveau 1 à gauche ne sont plus divisibles. On continue sur le groupe de droite. On arrête la division à l'étape suivante (même s'il reste un noeud non *homogène*).

Un exemple



Règle de décision : on affecte la réponse “yes” si le noeud terminal (feuille) contient une proportion plus importante de “yes”, “no” sinon. On a fabriqué un prédicteur !

Quelques remarques/questions avant une construction effective

- Une fois l'arbre construit, on affecte à chaque sous-groupe (feuille) une classe si Y qualitatif : la plus représentée dans la feuille si on se base sur l'erreur de classification usuelle, une valeur Y quelconque si Y quantitatif : par exemple, la moyenne des Y_i appartenant à la feuille semble un choix naturel. Formellement, si $(R_t)_t$ désigne les régions de $\mathbb{R}^p$ relatives à la partition finale, la règle de décision s'écrit :

$$\hat{f}(x) = \sum_t \hat{y}_t 1_{x \in R_t}.$$

- Doit-on utiliser toutes les variables explicatives ? Non en général . . . (le surapprentissage guette. . .). N.B. On peut utiliser plusieurs fois la même variable (d'où le faible intérêt des arbres non binaires).
- En pratique, l'ensemble des arbres binaires constructibles à partir de p variables croît (au moins) exponentiellement avec p. Il faut donc trouver une stratégie raisonnable (en coût de calcul) permettant de choisir un "bon" arbre au sens usuel du terme (*i.e.* avec un bon compromis biais/variance).

Méthodes CART/C4.5

- CART : Classification and Regression Tree, introduit par Breiman (1984).
- C4.5 : Quinlan (1993), même principe de construction mais basé sur un critère différent.
- Principe général : $(X_1, \ldots, X_p)$ l'ensemble des variables quantitatives ou qualitatives explicatives, Y réponse quantitative ou qualitative,
- 1 - Construire un arbre maximal noté $A_{\max}$.
- 2 - Ordonner les sous-arbres selon une séquence emboîtée suivant la décroissance d'un critère pénalisé de déviance ou de taux de mal-classés.
- 3 - Sélectionner "le" sous-arbre optimal : c'est la procédure d'*élagage* (Pruning).

Construction d'un arbre binaire maximal

- Critère de division : basé sur la définition d'une fonction d'hétérogénéité ou de désordre.
- L'hétérogénéité d'un noeud t se mesure par une fonction positive notée Q_t qui doit être
  - nulle si le noeud est homogène : tous les individus appartiennent à la même modalité ou prennent la même valeur de Y.
  - "maximale" lorsque les valeurs de Y sont équiprobables ou très dispersées.
- Admissibilité : Une division est dite admissible si aucun des noeuds descendants n'est vide, *i.e.* si la règle de division n'a pas conduit à classer tous les individus dans un même noeud fils.
- La croissance de l'arbre maximal s'arrête à un noeud donné, qui devient donc terminal selon un critère d'homogénéité : soit il est parfaitement homogène, soit on fixe un seuil, sur la non-homogénéité ou sur le nombre d'individus (ex: moins de 2 individus dans le noeud), sous lequel on arrête la division.

Critère d'homogénéité et Règle de division

A chaque noeud, la règle consiste à choisir la variable et le choix du découpage qui fait le plus diminuer l'hétérogénéité. Notons $\mathcal{D}$ l'ensemble des divisions admissibles. Notons le noeud père t,

- t_l et t_r, ses noeuds fils,
- Q_t l'hétérogénéité au noeud t, N_t le nombre d'individus au noeud t
- pour une division admissible $d \in \mathcal{D}$, $Q_{t_l}(d)$ et $Q_{t_r}(d)$ les hétérogénéités associées.

La division se fait alors en déterminant

$$\text{Argmax}_{d\in\mathcal{D}} \left(Q_t - \frac{N_{t_g}(d)}{N_t} Q_{t_l}(d) - \frac{N_{t_r}(d)}{N_t} Q_{t_r}(d) \right)$$

ou de manière équivalente

$$\text{Argmin}_{d\in\mathcal{D}} \left(N_{t_g}(d) Q_{t_l}(d) + N_{t_r}(d) Q_{t_r}(d) \right).$$

A propos des divisions

- dans le cadre quantitatif, *i.e.* ordonné, continu ou discret à valeurs dans $\mathcal{X}$ ($\mathbb{R}$ si continu), l'ensemble des divisions possibles est :

$$\mathcal{D} = \{\{X_j \leq s\}\{X_j > s\}, j \in \{1, \ldots, p\}, s \in \mathcal{X}\}.$$

- dans le cadre qualitatif, on a a priori $\frac{1}{2}(2^{m_j} - 2) = 2^{m_j - 1} - 1$ choix par variable j à valeurs dans un ensemble à m_j éléments.

- Exemple : si X_j est à valeurs dans A, B, C, alors, on peut tester A vs $\{B, C\}$, B vs $\{A, C\}$, et C vs $\{A, B\}$ (soit $2^2 - 1$ choix).

Q_t pour la classification

Il existe plusieurs choix possibles. On suppose que Y est à valeurs dans $\{1, \dots, m\}$ et on note pour un noeud fixé t, $\hat{p}_{t,k}$ la proportion d'individus de type k au noeud t.

1. Erreur de classification : si on affecte une classe au noeud t, alors elle correspondra à celle qui est la plus représentée. Notons-la

$$Q_t = 1 - \hat{p}_{t,k(t)}.$$

   Excepté le cas de classification binaire, ce critère n'est pas compétitif car il ne mesure pas l'hétérogénéité (voir TD). On lui préfère l'indice de Gini (CART) ou l'entropie de Shannon (C4.5) :

2. Indice de Gini :

$$Q_t = \sum_{k \neq k'} \hat{p}_{t,k} \hat{p}_{t,k'} = \sum_{k=1}^{m} \hat{p}_{t,k}(1 - \hat{p}_{t,k}).$$

3. Entropie de Shannon.

$$Q_t = -\sum_{k=1}^{m} \hat{p}_{t,k} \log \hat{p}_{t,k}.$$

   (voir TD pour interprétation de Gini et entropie).

Q_t pour la régression

Supposons Y quantitative. Dans ce cas, le choix usuel consiste à minimiser la variance intra-noeuds : notons $\mathcal{I}_t$ l'ensemble des individus au noeud t.

$$V(t) = \frac{1}{|\mathcal{I}_t|} \sum_{i \in \mathcal{I}_t} (y_i - \bar{y}_t)^2$$

où

$$\bar{y}_t = \frac{1}{|\mathcal{I}_t|} \sum_{i \in \mathcal{I}_t} y_i.$$

Remarque: Dans les différents cas, la décroissance de l'hétérogénéité est une quantité visualisable sur le graphe via la longueur du segment qui relie le noeud père au noeuds fils.

Elagage

Pour un arbre A, on note $\mathcal{F}_A$ l'ensemble des feuilles (noeuds terminaux) et K_A le nombre de feuilles. La qualité d'ajustement de A est définie par :

$$Q(A) = \sum_{t \in \mathcal{F}_A} N_t Q_t(A).$$

A nouveau Q_t désigne une mesure de l'hétérogénéité (erreur de classif., Gini, entropie, variance intra-noeuds,...).

Proposition

Pour tout sous-arbre A de $A_{\max}$ (construit via la règle de division) satisfait $Q(A) \geq Q(A_{max})$. En d'autres termes, la quantité $Q(A)$ décroît avec les divisions.

Afin de pouvoir sélectionner un modèle en évitant le surapprentissage, on pénalise cette quantité par le nombre de feuilles :

$$C_\alpha(A) = Q(A) + \alpha K_A$$

L'élagage est alors possible.

Comment élaguer ?

Etape 1 : Sélection de sous-arbres emboîtés. Il s'agit d'une procédure itérative où à la fois

1. La taille de l'arbre va réduire jusqu'à retour à l'état racine.
2. Le paramètre α va être calibré à chaque itération.

On initialise à $A_0 = A_{\max}$ et $\alpha_0 = 0$ puis à chaque itération :

1. En faisant croître α, on trouve un α_1 minimal et un noeud t_1 pour lequel $C_\alpha(A_0^{t_1}) = C_0(A_0)$ où $A_0^{t_1}$ désigne l'arbre élagué sous t_1.
2. On choisit donc l'arbre élagué pour lequel la division a le moins fait diminuer Q_t. On note A_1 cet arbre et on continue la procédure avec la même règle.

On a ainsi fabriqué une suite de sous-arbres candidats à être optimaux.
Etape 2 : Parmi ces candidats, on choisit le meilleur par estimation de l'erreur de prédiction sur un échantillon test ou par validation croisée.

Avantages/Inconvénients

1. Avantages :
   - Non-paramétrique, capable de gérer tous types de variables
   - D'interprétation simple et visualisable.
2. Inconvénients :
   - Pas de performances garanties théoriquement.
   - Manque de régularité dans les résultats dans le cas de la régression.
   - Instabilité importante lorsque la complexité du problème augmente
3. Solutions/Améliorations pour rendre cette approche plus robuste.
   - Boosting/Bagging
   - Random Forests... (au prochain épisode)

Bagging,Random Forests, Boosting

Agrégation de modèles

Les méthodes décrites ci-après entrent dans la famille des méthodes d'agrégation de modèles.

- Qu'est-ce que l'agrégation de modèles ?
- Comme son nom l'indique, l'idée est d'agréger des modèles/algorithmes afin d'en augmenter les qualités prédictives.

- Dans la suite, nous présentons 3 méthodes d'agrégation dans le cadre des arbres de décision. Les idées relatives au boosting et au bagging (et même celles de la construction des random forests) sont néanmoins transposables à d'autres problèmes.

Bootstrap

Les méthodes ci-après font usage de la méthode dite de *bootstrap*, qui consiste à faire usage de sous-échantillons tirés aléatoirement et avec remise de l'échantillon complet (introduit par Efron au début des années 80) :

- Soit $(Z_1, \ldots, Z_n)_n$ un échantillon.
- $(i_1, \ldots, i_k)$ une suite d'indices tirés aléatoirement avec remise dans $\{1, \ldots, n\}$.
- $(Z_{i_1}, \ldots, Z_{i_k})$ est un échantillon boostrap issu de $(Z_1, \ldots, Z_n)_n$ (on peut choisir $k = n$).
- On le note plus simplement $(Z_1^{*,b}, \ldots, Z_k^{*,b})$, $b \in \{1, \ldots, B\}$ où B le nombre d'échantillons bootstrap.

- Supposons par exemple que l'on cherche à estimer un paramètre θ. Chaque échantillon produit un estimateur $\hat{\theta}^*(b)$.
- La quantité $\hat{\theta^*} = \frac{1}{B} \sum_{b=1}^{B} \hat{\theta}^*(b)$ fournit alors un nouvel estimateur de θ qui apporte de l'information. Par exemple,

Bootstrap (suite)

- La variance empirique bootstrap

$$\hat{\sigma}_B^2 = \frac{1}{B-1} \sum_{b=1}^{B} (\hat{\theta^*}(b) - \hat{\theta^*})^2$$

  fournit un estimateur de la variance.

- Malgré le rééchantillonnage qui implique évidemment l'absence d'indépendance entre les échantillons bootstrap, ce procédé permet de produire une estimation de la variance sans utiliser de nouvelles variables.

- Dans le cadre des arbres, ce procédé va être utilisé pour agréger des arbres et améliorer les performances de classification en réduisant généralement la variance.

Bagging

Il s'agit de la version la plus simple d'agrégation par moyennisation bootstrap.

Principe :

- On tire B échantillons bootstrap issus de l'échantillon (d'entraînement). Chaque échantillon produit un prédicteur $\hat{f}_b(.)$.
- Dans le cadre quantitatif, on note $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x)$.
- Dans le cadre qualitatif usuel, on note $\hat{f}_{bag}(x) = \mathrm{Argmax}_{k=1}^{m} \sum_{b=1}^{B} 1_{\{\hat{f}_b(x)=k\}}$ (Autrement dit, on procède par vote majoritaire).
- $\hat{f}_{bag}$ est l'estimateur obtenu par bagging.
- L'estimation OOB (Out-Of-Bag) de l'erreur de prédiction est obtenue en calculant pour chaque indice i, la prédiction bootstrap $\hat{Y}_i$ issue de l'ensemble des échantillons bootstrap ne contenant pas l'indice i. L'erreur OOB est alors
  - $\frac{1}{n}\sum_{k=1}^{n}(Y_i - \hat{Y}_i)^2$ dans le cas quantitatif (pour la fonction de perte usuelle)
  - $\frac{1}{n}\sum_{k=1}^{n} 1_{Y_i \neq \hat{Y}_i}$ dans le cadre qualitatif (avec la fonction de perte usuelle).

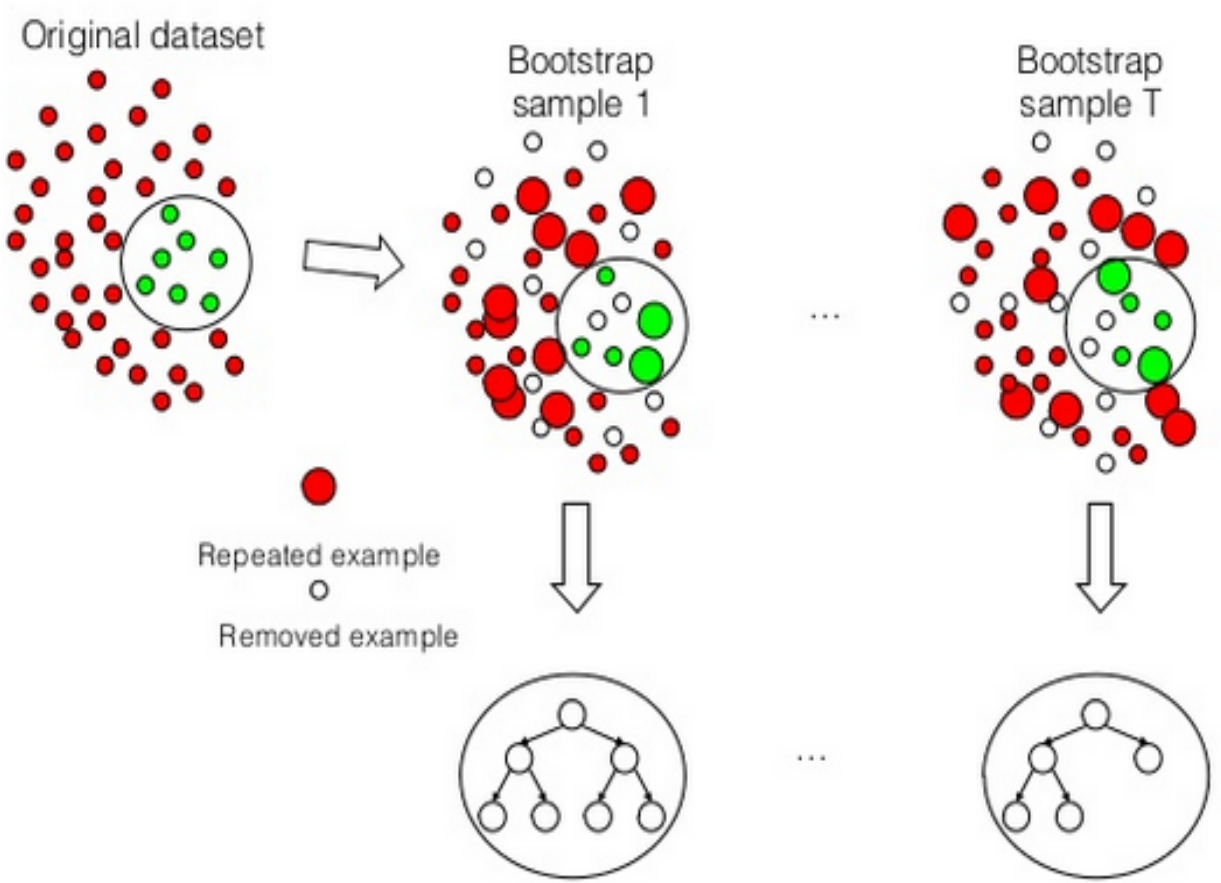
Bagging pour les arbres de décision

Principe/Remarques :

- Construire B arbres de décision issus des B échantillons bootstrap.
- Associer un prédicteur à chaque arbre "bootstrappé".
- Moyenniser comme présenté dans le slide précédent.

- En général, seule la première phase de CART est conservée : il n'y a pas d'élagage. L'idée est que le surapprentissage potentiel est compensé par la moyennisation bootstrap.
- L'erreur out-of-bag peut être calculée au fil de l'ajout d'échantillons bootstrap mais ce procédé est coûteux.
- Ce principe permet en pratique d'améliorer les performances mais réduit l'interprétabilité (ou le caractère explicatif) puisque la structure d'arbre est remplacée par une moyenne (cette remarque vaut aussi pour les algorithmes présentés dans la suite). On peut néanmoins envisager de quantifier l'importance des variables en comptabilisant leurs (et leurs niveaux) d'apparitions dans chacun des arbres mis en jeux.
- Généralement, le bootstrap fonctionne lorsque les arbres en jeu ne sont pas trop (positivement) corrélés (voir TD).

Bagging


Original dataset
Bootstrap sample 1
Bootstrap sample T
...
Repeated example
Removed example
...

Forêts aléatoires

Afin de limiter la corrélation entre les arbres issus du bagging, le principe des forêts aléatoires est de randomiser les variables mises en jeu dans la division et de n'en choisir que m parmi les p disponibles. **Détails** :

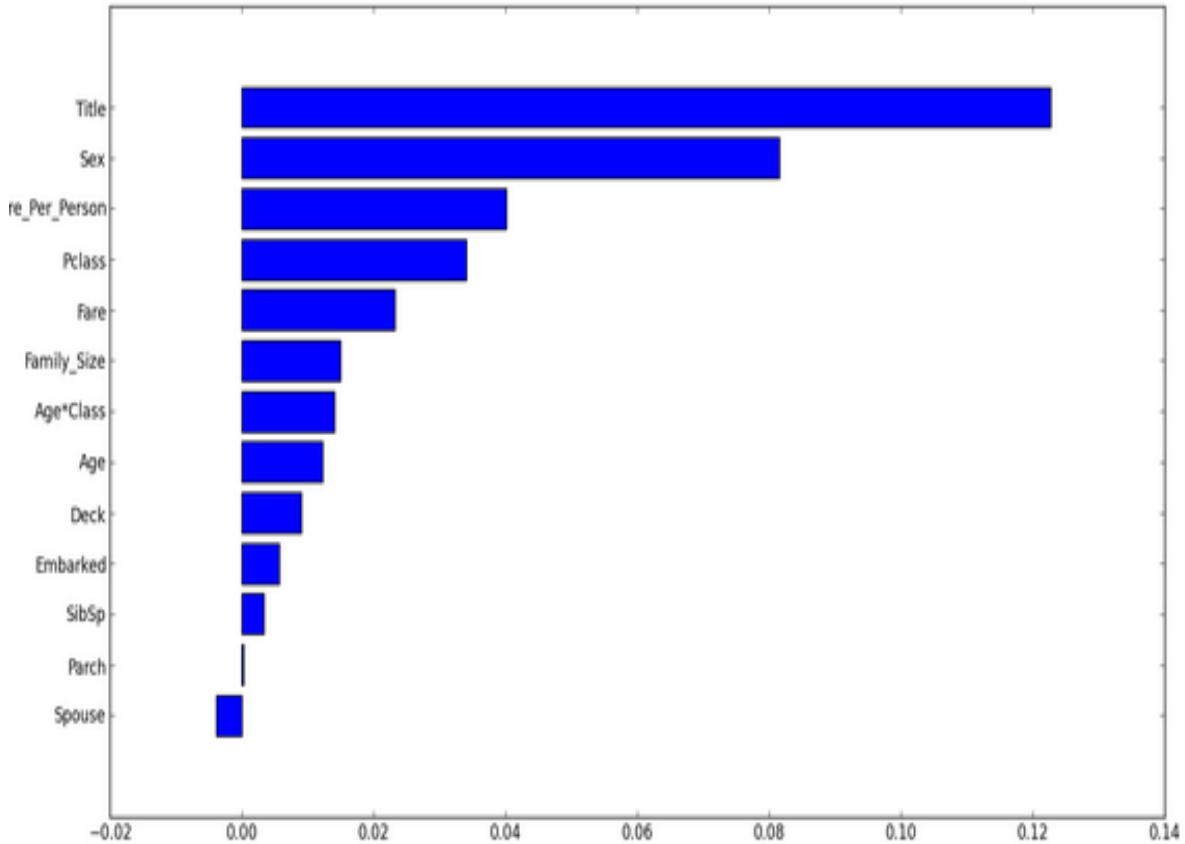
- On tire B échantillons bootstrap issus de l'échantillon.
- Pour chaque échantillon bootstrap, on construit l'arbre de décision maximal avec la règle suivante : à chaque noeud, on choisit la meilleure division possible basée sur m variables tirées aléatoirement parmi les p (variantes possibles).
- On tire un prédicteur $\hat{f}_b$ de chaque échantillon bootstrap.
- On conclut comme pour le bagging.

Remarques

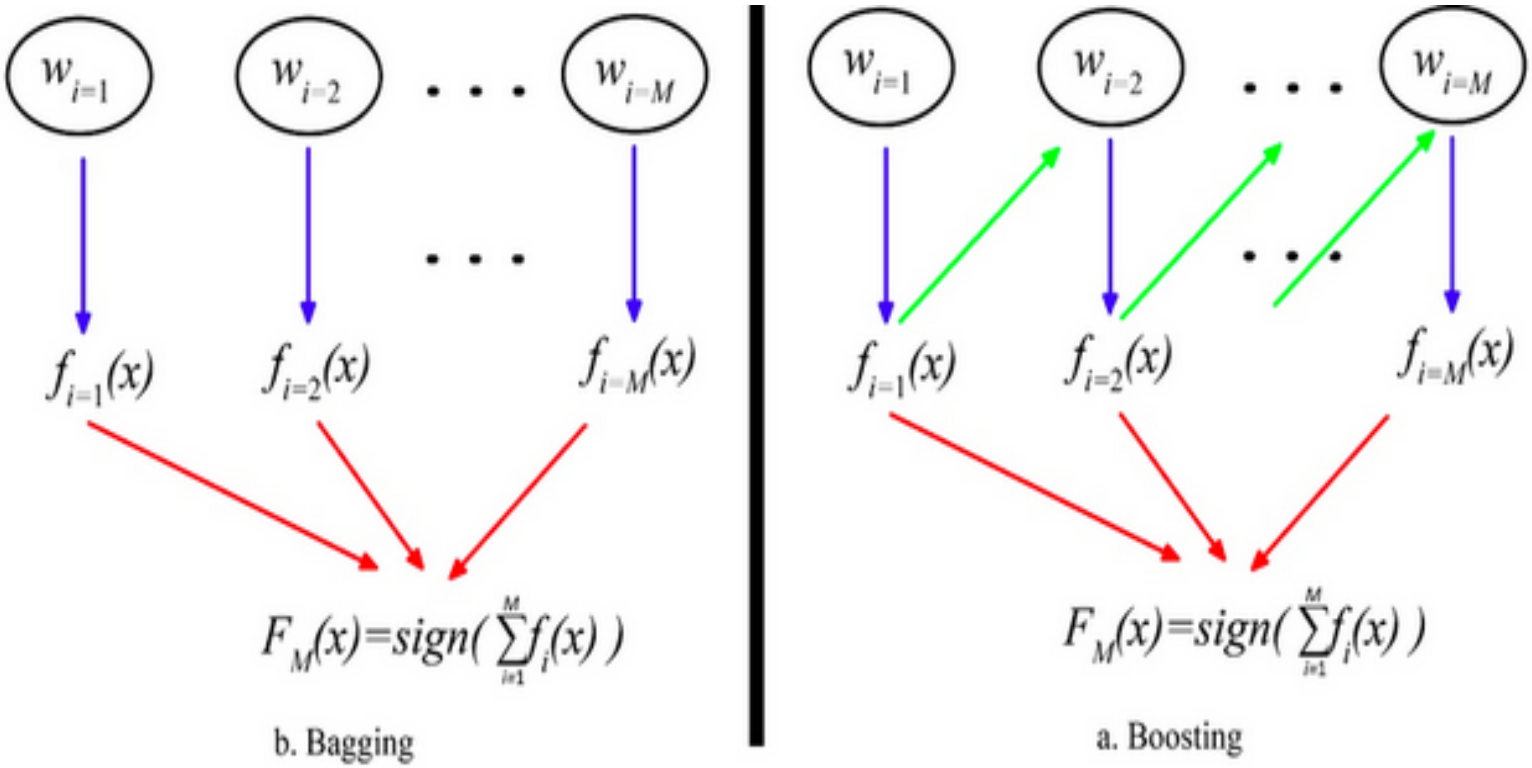
- Intérêt : réduire la corrélation entre les arbres : par exemple, si une variable joue un rôle prédictif trop important, elle apparaîtra dans la plupart des arbres baggés dans le haut de l'arbre.
- En pratique $m \approx \sqrt{p}$.
- Méthode très performante, quelques résultats théoriques récents (*e.g.* Arlot).

Forêts aléatoires

Possibilité de mesure l'importance de chaque variable (fonction `importance` dans `R`, package `randomForest`) afin de "récupérer" un peu d'interprétabilité.



Boosting


$w_{i=1}$
$w_{i=2}$
$w_{i=M}$
$f_{i=1}(x)$
$f_{i=2}(x)$
$f_{i=M}(x)$
$F_M(x)=sign(\sum_{i=1}^{M} f_i(x))$
b. Bagging
$w_{i=1}$
$w_{i=2}$
$w_{i=M}$
$f_{i=1}(x)$
$f_{i=2}(x)$
$f_{i=M}(x)$
$F_M(x)=sign(\sum_{i=1}^{M} f_i(x))$
a. Boosting

Boosting

A l'origine le boosting a été introduit pour améliorer les performances de faibles classifieurs en les agrégeant intelligemment. Par rapport au bagging qui n'est conçu "que" pour réduire la variance, le boosting a pour objectif de réduire également le biais.

Principe

1. Construction itérative d'une agrégation de modèles.
2. A chaque étape, on augmente le poids des observations mal ajustées/mal prédites.
3. On voit donc a priori le double-effet potentiel : réduction de la variance par agrégation et diminution biais via 2.

Pour l'instant, le propos est vague. Les algorithmes ayant ce point de vue sont nombreux. On se concentre ici sur la présentation de l'algorithme historique : *Adaboost*.

Adaboost

- Soit $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ un échantillon d'apprentissage.
- On note $\mathbf{w} = \{w_i, i = 1, \ldots n\}$ les poids relatifs à chaque observation.
- On initialise les poids w_i associés à chaque observation à $1/n$.

Algorithme dans le cas de la classification : A l'étape m,

- On fabrique un prédicteur $\hat{f}_m$ sur le modèle m en minimisant une erreur de prédiction pondérée :

$$\hat{f}_m = \text{Argmin}_{f \in \mathcal{F}} \sum_{i=1}^{n} w_i 1_{y_i \neq f(x_i)}.$$

- On calcule l'erreur pondérée

$$\mathcal{E}_m = \frac{\sum_{i=1}^{n} w_i 1_{\{\hat{f}_m(x_i) \neq y_i\}}}{\sum_{i=1}^{n} w_i}.$$

- SI $\mathcal{E}_m > 1/2$, on arrête. Sinon, on calcule le logit $\alpha_m = \frac{1}{2} \log\left((1 - \mathcal{E}_m)/\mathcal{E}_m\right)$.
- On actualise les poids de la manière suivante :

$$w_i \leftarrow Z_m^{-1} w_i \exp(\alpha_m 1_{\{\hat{f}_m(x_i) \neq y_i\}}), \ldots i = 1, \ldots, n,$$

Adaboost

Le classifieur final est (dans le cas où les classes sont -1 et 1) :

$$\hat{f}(x) = \operatorname{sgn}\left(\sum_{m=1}^{M} \alpha_m \hat{f}_m(x)\right).$$

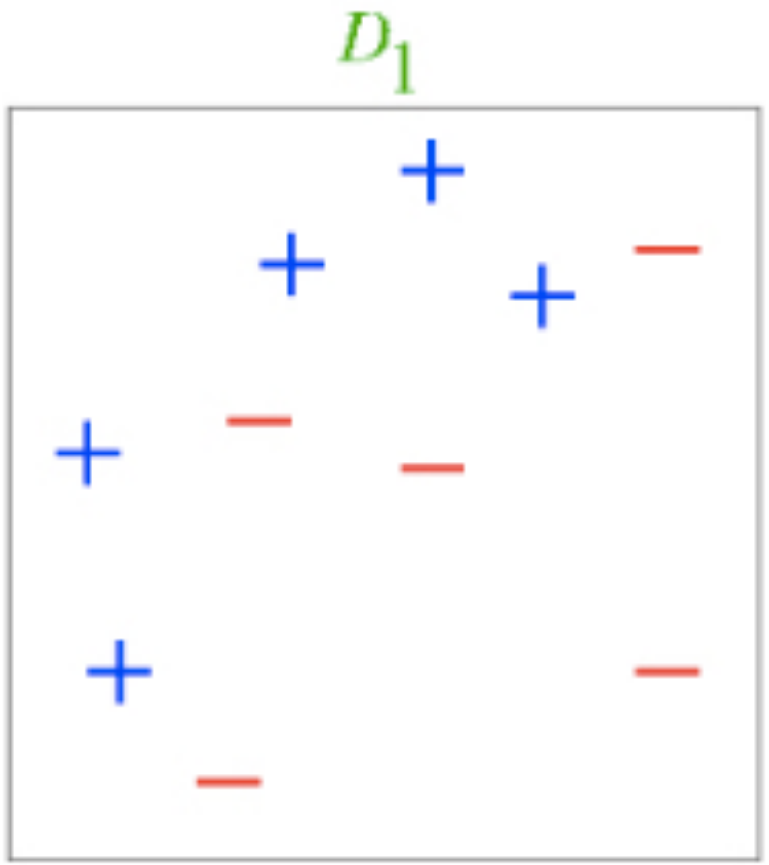
Remarque La règle d'actualisation des poids est fabriquée pour que :

- Le poids soit augmenté/diminué si mal classé/bien classé.
- Après plusieurs itérations, le poids va donc se concentrer sur les variables difficiles à classer.
- Le nouveau modèle joue donc le rôle d'ajusteur de ces "dernières" variables.

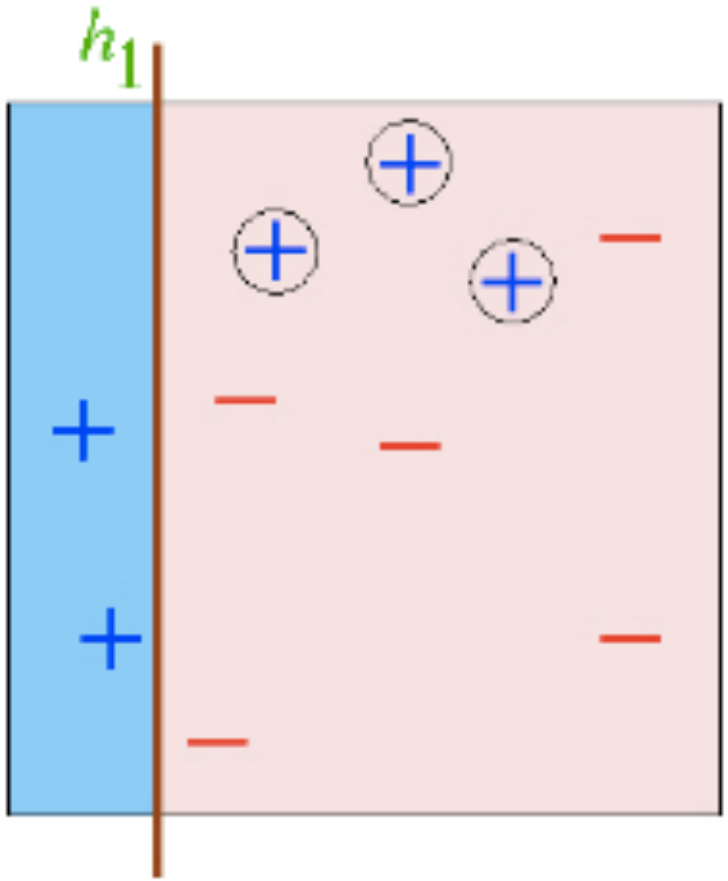
Beaucoup de variantes de Adaboost.

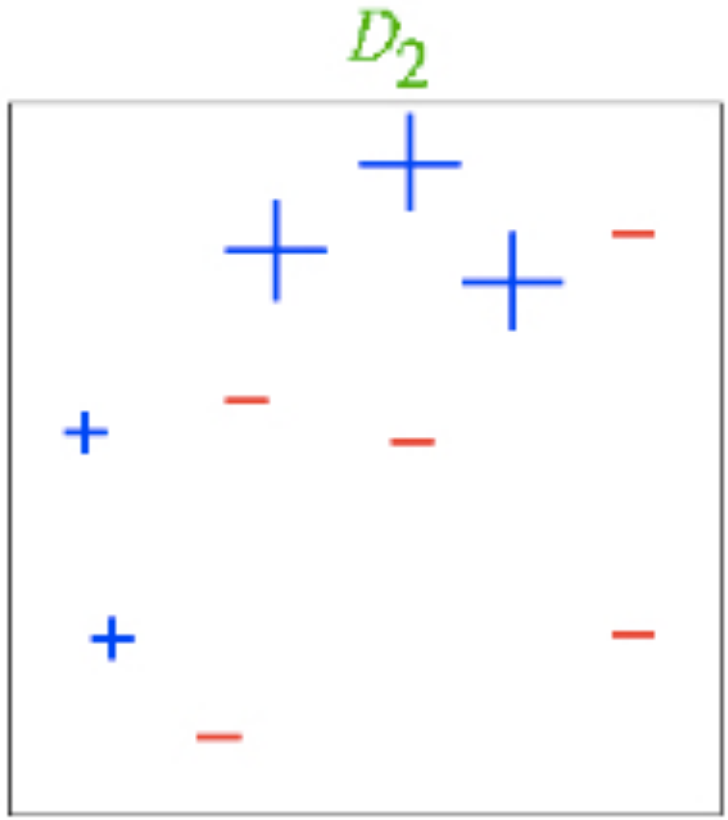
Adaboost sur un exemple jouet

Issu de `https://www.lri.fr/~antoine/Courses/Master-ISI/Tr-boosting-06x4.pdf`.



Adaboost sur un exemple jouet

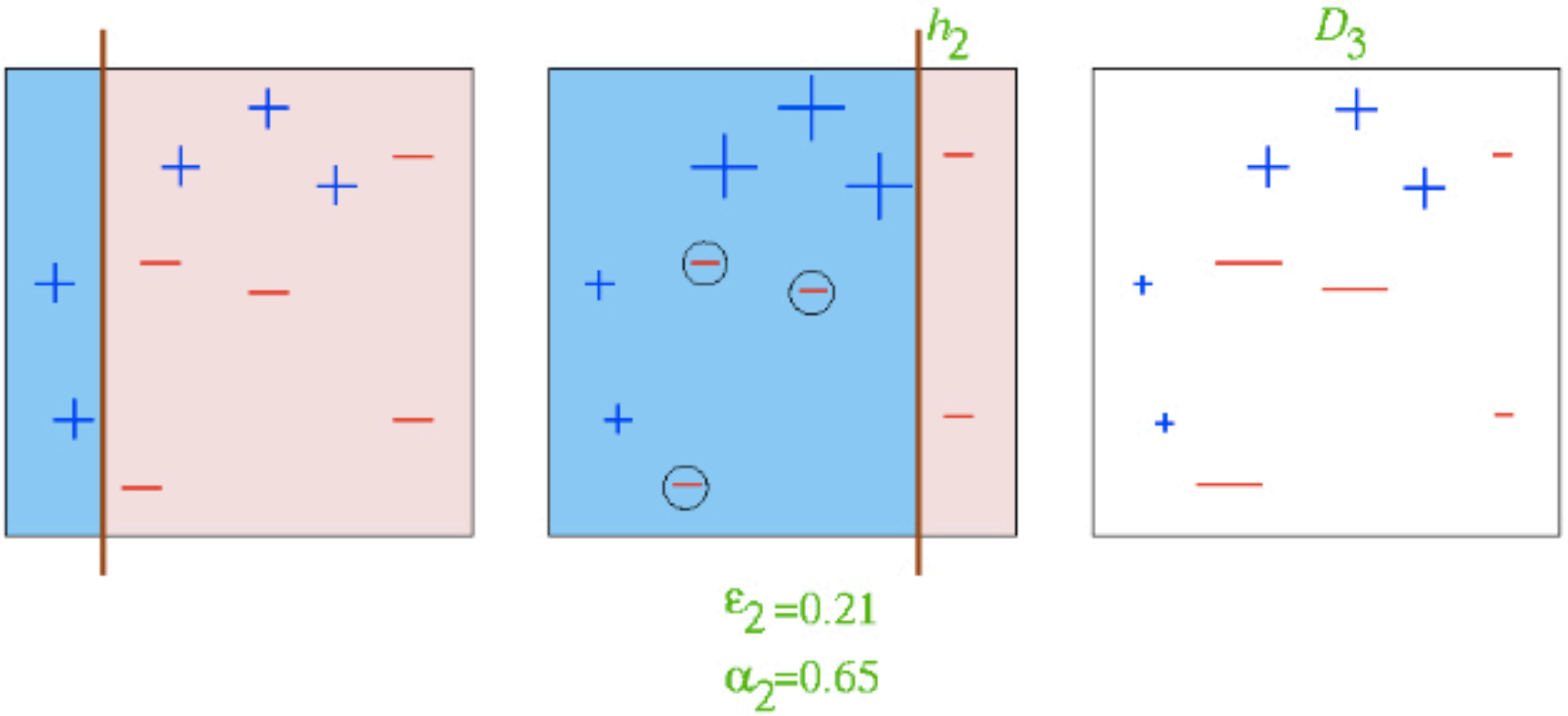




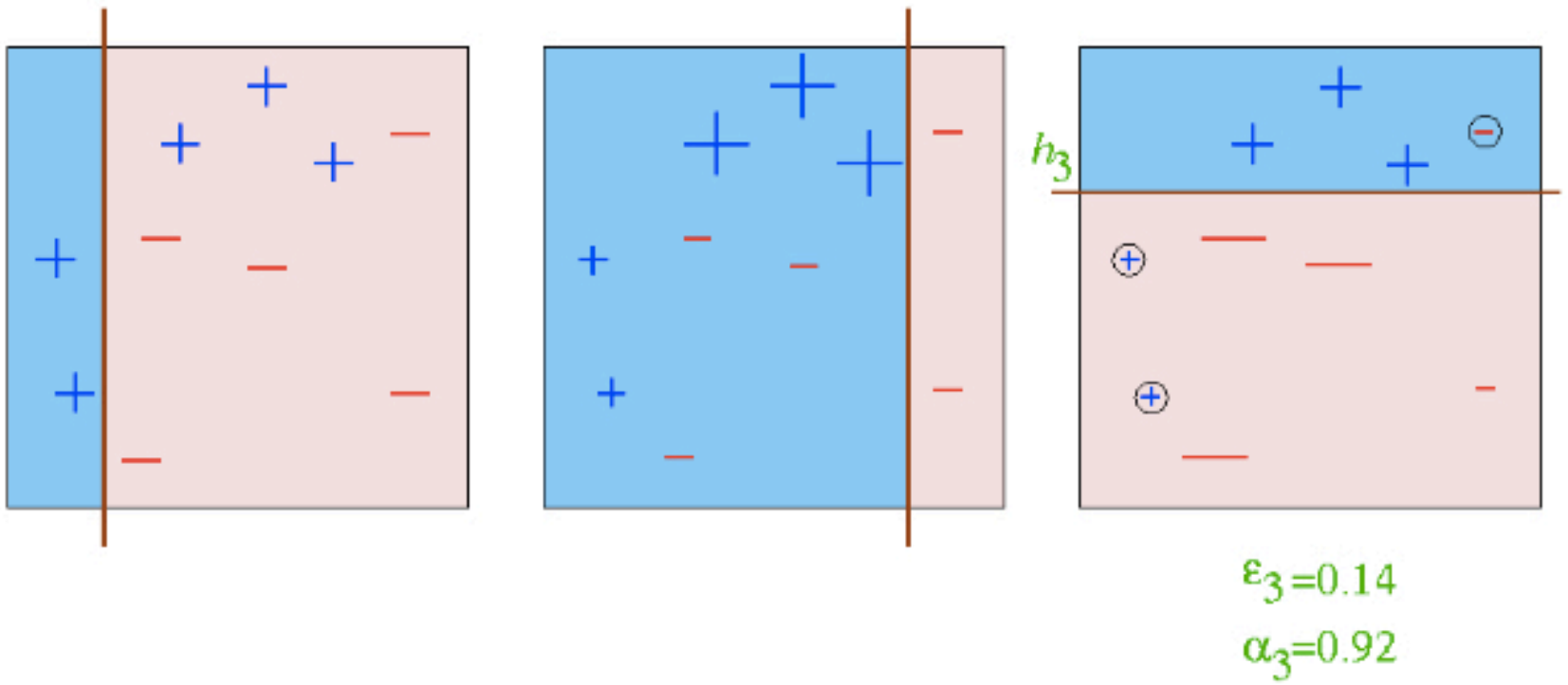
$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

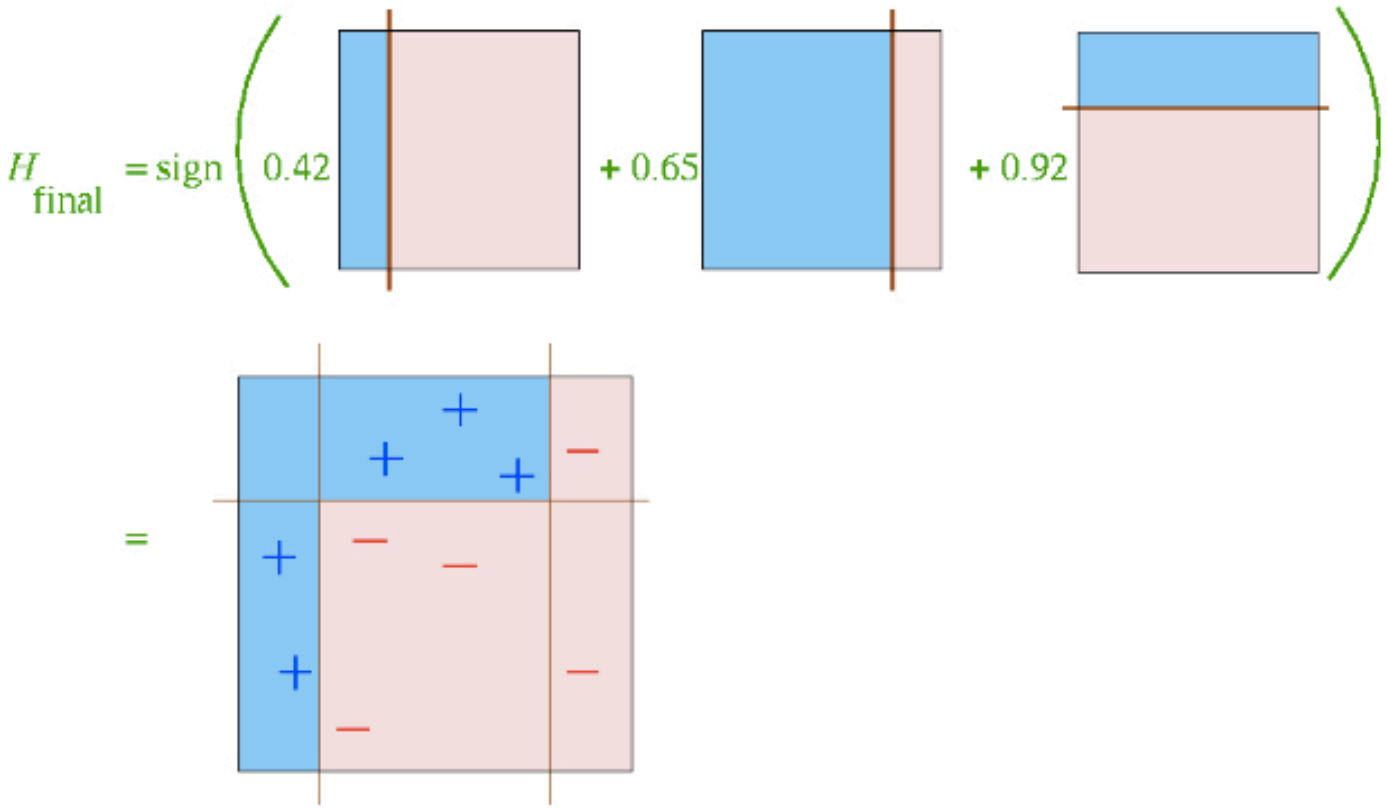
Adaboost sur un exemple jouet



Adaboost sur un exemple jouet



Adaboost sur un exemple jouet



Boosting pour les arbres

Ce principe peut donc s'appliquer à l'agrégation d'arbres

- en fabriquant à chaque étape un nouvel arbre basé sur un échantillon bootstrap tiré selon la loi (pondérée) associée aux ω_i (par exemple par méthode CART).
- Pour plus de détails, voir Hastie et. al., Elements of Stat. Learning.
- Aujourd'hui, les descendants d'Adaboost se nomment "Gradient Boosting" et depuis quelques années "Extrem Gradient Boosting" (**XGBoost**) dont la célébrité vient des performances remarquables (voir prochain TP pour un développement sur ce sujet).

Gradient Boosting : Principe

1. On suppose que l'on a comme objectif "ultime" de fabriquer une combinaison linéaire de classifieurs faibles optimale pour une certaine fonction de perte. On cherche toujours f sous la forme

$$f = \sum_{j=1}^{m} \lambda_j f_j$$

2. Idée du gradient boosting : recherche pas à pas selon la direction du gradient.
3. Plus précisément, on commence par initialiser l'algorithme par un classifieur f_0 simple (en régression, la meilleure constante) puis on va tenter de mimer une descente de gradient en cherchant dans une famille de candidats celui qui ressemble le plus à une vraie descente de gradient.
4. Le point délicat à comprendre est qu'il s'agit d'une descente de gradient fonctionnelle, *i.e.* on cherche idéalement la meilleure direction au sens des fonctions.

Algorithme

Algorithme 2 Boosting par descente de gradient fonctionnelle.

Entrées :

- $\mathbf{x}$ l'observation à prévoir
- h une règle faible
- $d_n = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$ l'échantillon
- λ un paramètre de régularisation tel que $0 < \lambda \leq 1$.
- M le nombre d'itérations.

1. Initialisation :

$$g_0(.) = \underset{c}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, c)$$

2. **Pour** $m = 1$ à M :

   (a) Calculer l'opposé du gradient $-\frac{\partial}{\partial g} L(y, g)$ et l'évaluer aux points $g_{m-1}(\mathbf{x}_i)$:

$$U_i = - \left[\frac{\partial \ell(y_i, g(\mathbf{x}_i))}{\partial g(\mathbf{x}_i)} \right]_{\Big| g(\mathbf{x}_i) = g_{m-1}(\mathbf{x}_i)}, \quad i = 1, \ldots, n.$$

   (b) Ajuster la règle faible sur l'échantillon $(\mathbf{x}_1, U_1), \ldots, (\mathbf{x}_n, U_n)$, on note h_m la règle ainsi définie.

   (c) Mise à jour : $g_m(\mathbf{x}) = g_{m-1}(\mathbf{x}) + \lambda h_m(\mathbf{x})$.

3. **Sortie :** la règle $g_M(\mathbf{x})$.

Choix du λ

Ce choix peut être optimisé

Algorithm 5 *Gradient Tree Boosting* pour la régression

Soit $\mathbf{x}_0$ à prévoir

Initialiser $\widehat{f}_0 = \arg\min_\gamma \sum_{i=1}^n l(y_i, \gamma)$

for $m = 1$ à M **do**

Calculer $r_{mi} = -\left[\frac{\partial l(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f=f_{m-1}}$; $i = 1, \ldots, m$

Ajuster un arbre de régression δ_m aux couples $(\boldsymbol{x}_i, r_{mi})_{i=1,\ldots,n}$

Calculer γ_m en résolvant : $min_\gamma \sum_{i=1}^n l(y_i, f_{m-1}(\boldsymbol{x}_i) + \gamma\delta_m(\boldsymbol{x}_i))$.

Mise à jour : $\widehat{f}_m(\boldsymbol{x}) = \widehat{f}_{m-1}(\boldsymbol{x}) + \gamma_m\delta_m(\boldsymbol{x})$

end for

Résultat : $\widehat{f}_M(\boldsymbol{x}_0)$.

Vers XGBoost

- XGBoost : extension du gradient boosting avec des paramètres supplémentaires dont celui en particulier de la
- régularisation L^1 et L^2 (cf cours LASSO).
- Références : Description rapide : `https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-agreg.pdf`
- Détails de programmation : `https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/`