Arbres et Forêts Aléatoires

Master parcours SSD - UE Apprentissage Statistique II

Pierre Mahé - bioMérieux & Université de Grenoble-Alpes

Ce cours

Arbres de classification et de régression

- méthode ancienne, classique et populaire

A la base de la méthode des forêts aléatoires

- basée sur le bagging : aggrégation par bootstrap
- très performante sur de nombreux problèmes
- facile à paramétrer

Ce cours : des arbres aux forêts aléatoires

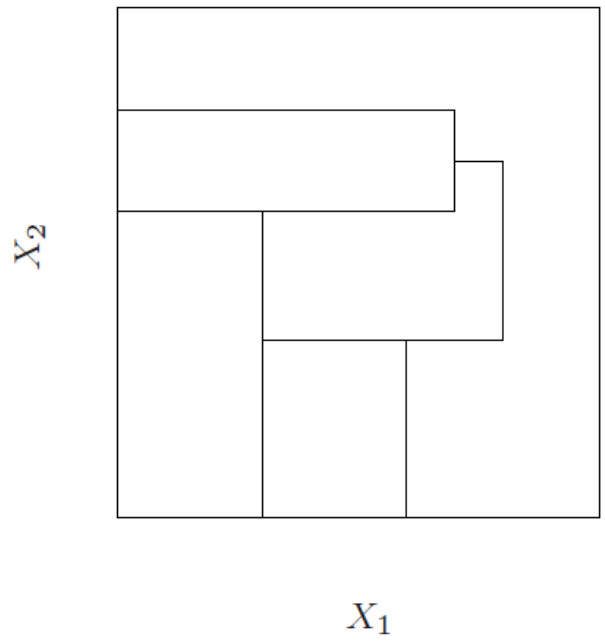
- arbres : motivation, formalisation, algorithmes et limites
- aggrégation d'arbres et bagging
- du bagging aux forêts aléatoires

Introduction

Principe

Principe [1] :

1. découper l'espace d'entrée en régions
2. estimer & prédire une valeur par région



⇒ Régression : valeur moyenne dans la région

⇒ Classification : classe majoritaire dans la région

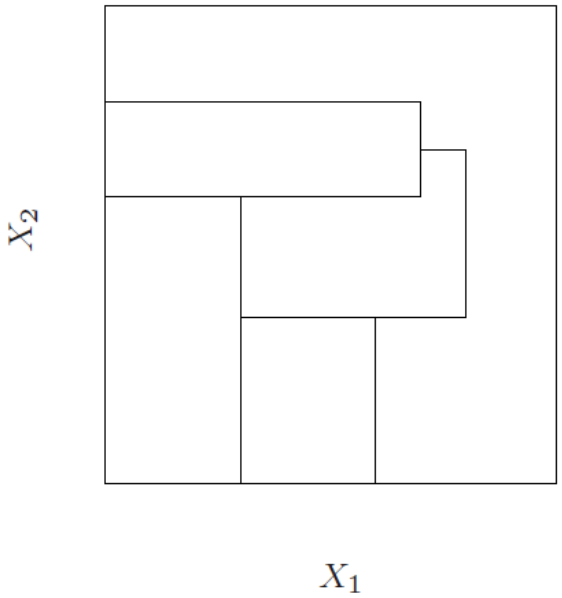
1. Illustrations tirées de Hastie et al. (2001) (chapitre 9)

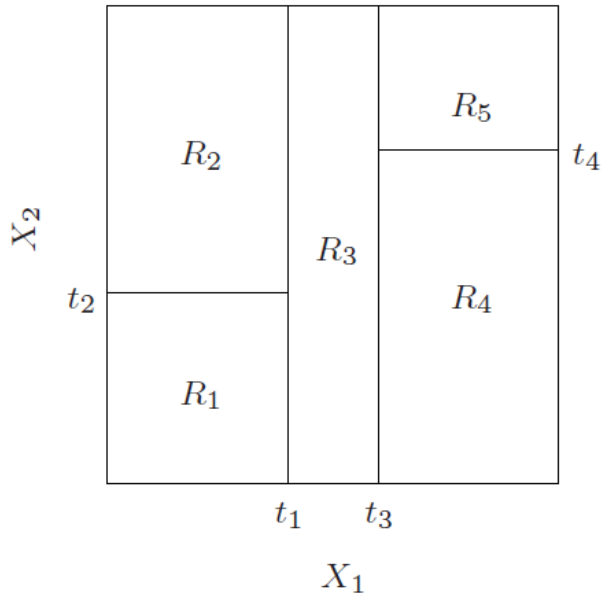
Principe

1ère question : comment construire ces régions ?

Approche par partitionnement binaire récursif

- binary recursive splitting



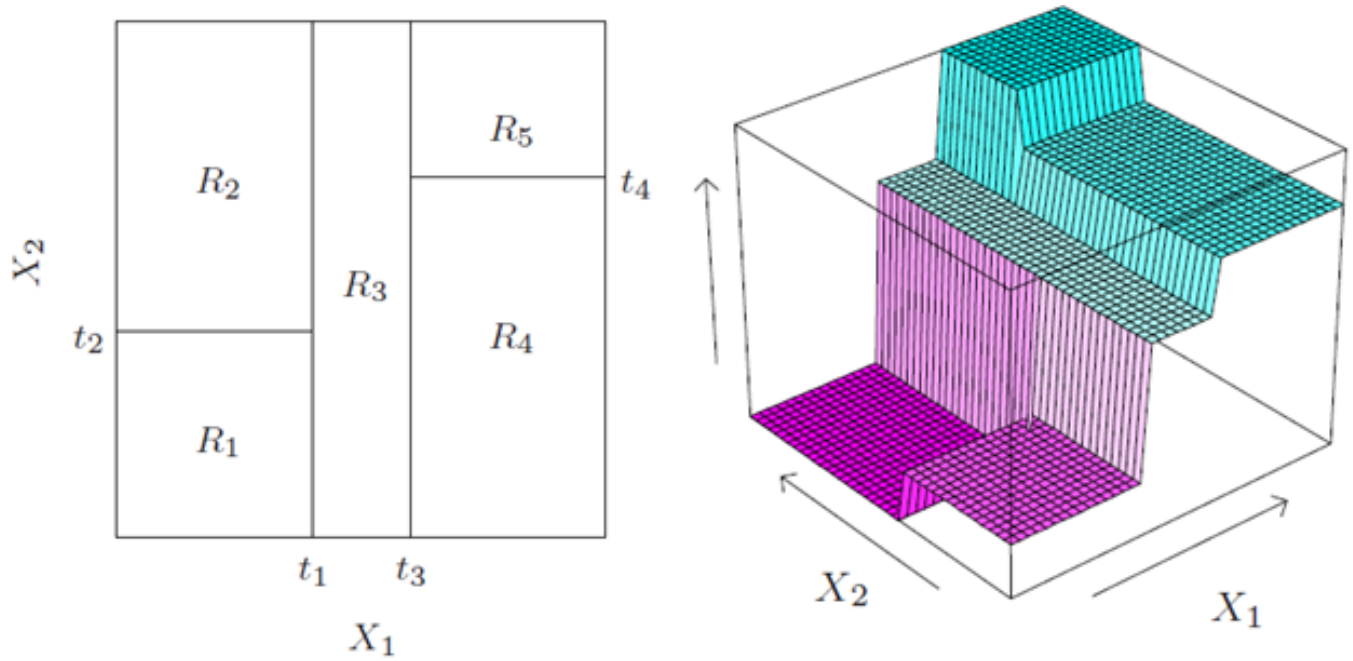


⇒ Chaque étape : 1 région divisée selon 1 seuil sur 1 variable

- moins général mais plus facile à construire

Principe

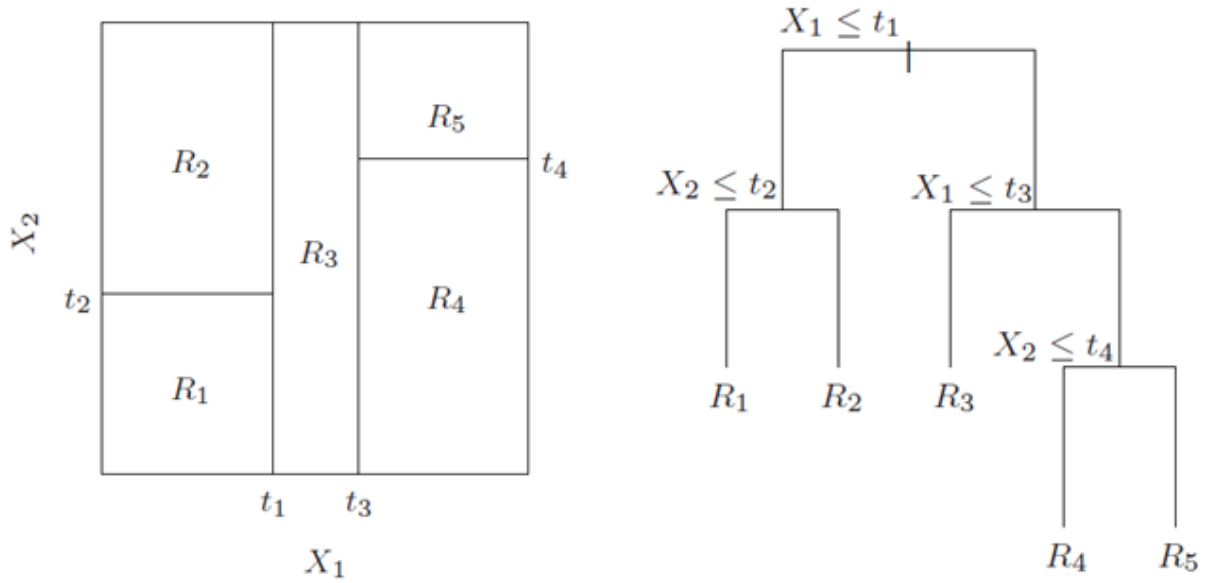
Illustration : régression



$\Rightarrow$ fonction non linéaire et non continue

Des arbres ?

Le processus récursif peut se traduire comme un arbre :



- racine de l'arbre : tout l'espace
- feuille de l'arbre : 1 région
  - régions imbriquées les unes dans les autres
- noeud (interne) : 1 seuil qui coupe une région en deux
  - test logique basé sur la valeur d'**une variable**

Arbres de décision

Avantages :

- interprétabilité du modèle
- mécanisme de prédiction proche du processus humain
  - e.g., domaine médical
- interprétation en tant qu'arbre valables pour $d >> 2$
  - fonction de prédiction très difficile à se représenter

Arbres de décision

Avantages :

- interprétabilité du modèle
- mécanisme de prédiction proche du processus humain
  - e.g., domaine médical
- interprétation en tant qu'arbre valables pour $d >> 2$
  - fonction de prédiction très difficile à se représenter

Questions :

- quel(s) critère(s) pour couper une région en deux ?
- quand arrêter le découpage ?

$\Rightarrow$ algorithme CART

CART : Classification And Regression Trees

Construction de l'arbre

Algorithme récursif

Initialisation : noeud racine

- contient l'ensemble du jeu de données / tout l'espace

On introduit un noeud :

- en séparant en deux groupes les observations affectées à un noeud de l'arbre
- selon un critère objectif
- tant qu'un critère d'arrêt n'est pas atteint

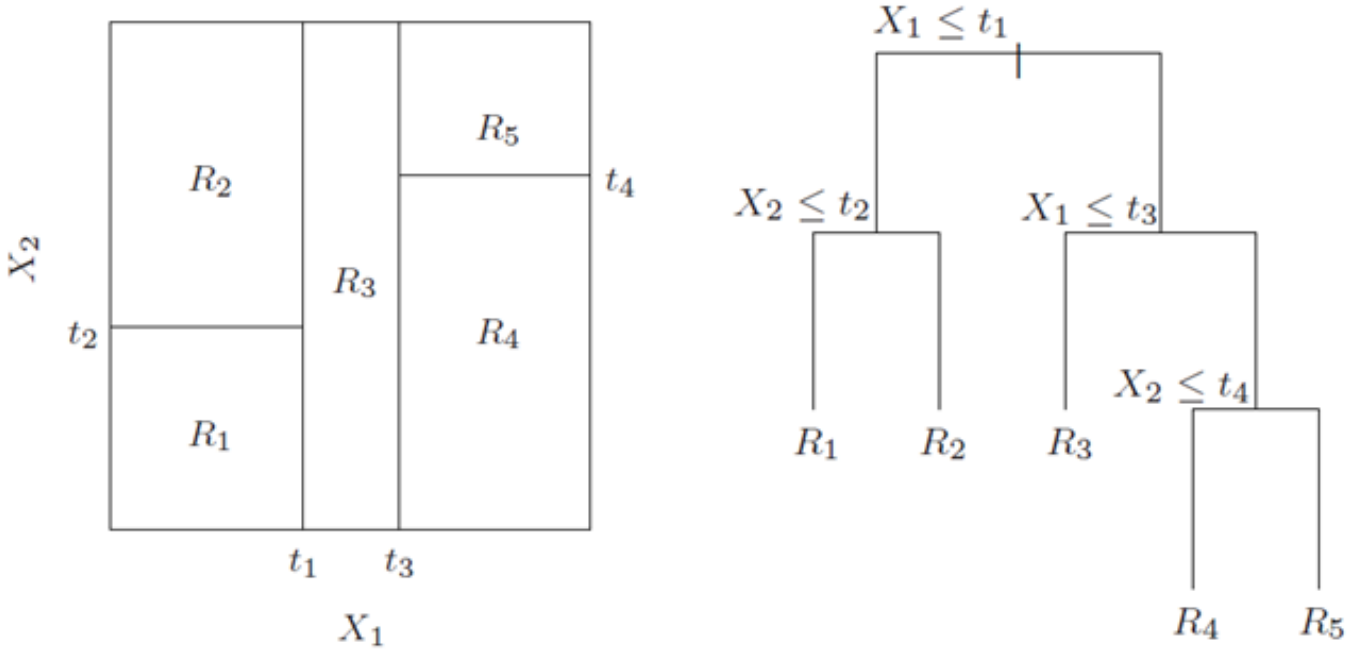
Construction de l'arbre

Comment séparer en 2 les observations affectées à un noeud ?

Construction de l'arbre

Comment séparer en 2 les observations affectées à un noeud ?

$\Rightarrow$ en considérant 1 seuil défini sur 1 variable



$\Rightarrow$ principe décliné pour la classification et la régression

Arbres de classification - formalisation

Comment séparer au mieux les instances affectées au noeud ?

Arbres de classification - formalisation

Comment séparer au mieux les instances affectées au noeud ?

$\Rightarrow$ en cherchant deux ensembles les plus purs possibles.

Arbres de classification - formalisation

Comment séparer au mieux les instances affectées au noeud ?

$\Rightarrow$ en cherchant deux ensembles les plus purs possibles.

Formalisation (pour la classification) :

- problème de classification à K catégories
- le noeud i contient n_i instances
- on définit le vecteur $P_i \in \mathbb{R}^K$:

$$\boxed{P_i[k] = \frac{1}{n_i} \sum_{j=1}^{n_i} \mathbf{1}(y_{C_i(j)} = k)}$$

où C_i est l'ensemble des instances du noeud i

$\Rightarrow$ les proportions des différentes classes au sein du noeud

Arbres de classification - critères d'impureté

Critères d'impureté du noeud i :

1. indice de Gini :

$$G(i) = 1 - \sum_{k=1}^{K} Pi[k]^2$$

2. entropie :

$$H(i) = -\sum_{k=1}^{K} Pi[k] \log_2(Pi[k])$$

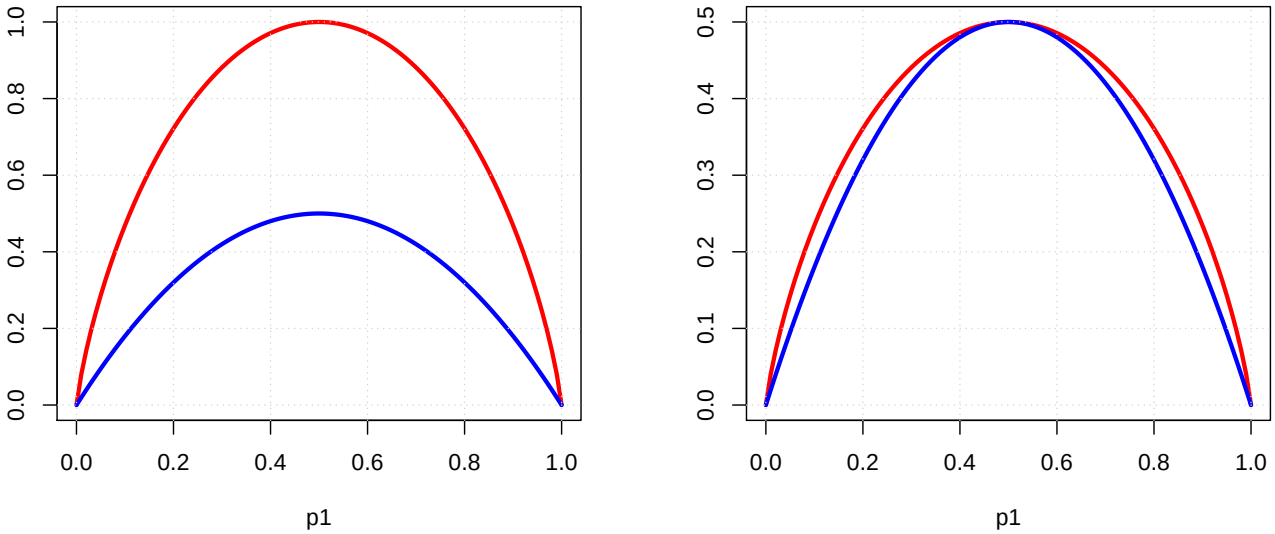
(avec par convention $0 \log(0) = 0$)

$\Rightarrow$ distribution uniforme : valeurs élevées
$\Rightarrow$ classe(s) majoritaire(s) : valeurs faibles

Arbres de classification - critères d'impureté

Illustration : classification binaire

- $P_i = \begin{bmatrix} P_i[1] & P_i[2] \end{bmatrix} = \begin{bmatrix} P_i[1] & 1 - P_i[1] \end{bmatrix} = \begin{bmatrix} p_1 & 1 - p_1 \end{bmatrix}$



- **à gauche** : entropie et Gini en fonction de p_1
- **à droite** : idem avec entropie divisée par 2

$\Rightarrow$ critères très similaires

- entropie = mesure du désordre, théorie de l'information

Arbres de classification - critère objectif

Comment séparer au mieux les instances affectées au noeud ?

$\Rightarrow$ en cherchant deux ensembles les plus purs possibles.

Arbres de classification - critère objectif

Comment séparer au mieux les instances affectées au noeud ?

$\Rightarrow$ en cherchant deux ensembles les plus purs possibles.

Critère objectif : moyenne pondérée des impuretés

$$\boxed{\frac{n_G}{n_i} I_G + \frac{n_D}{n_i} I_D}$$

- $n_i = \#$ d'instances affectées au noeud i (à couper)
- $n_G, n_D = \#$ d'instances affectées aux noeuds fils
  - Gauche et Droit
- $I_G, I_D =$ mesure d'impureté des noeud fils
  - i.e., entropie ou Gini

Arbres de classification - critère objectif

Comment séparer au mieux les instances affectées au noeud ?

$\Rightarrow$ en cherchant deux ensembles les plus purs possibles.

Critère objectif : moyenne pondérée des impuretés

$$\boxed{\frac{n_G}{n_i} I_G + \frac{n_D}{n_i} I_D}$$

- n_i = # d'instances affectées au noeud i (à couper)
- n_G, n_D = # d'instances affectées aux noeuds fils
  - Gauche et Droit
- I_G, I_D = mesure d'impureté des noeud fils
  - i.e., entropie ou Gini

$\Rightarrow$ problème : minimiser ce critère

Arbres de classification - critère objectif

Comment séparer au mieux les instances affectées au noeud ?

$\Rightarrow$ en cherchant deux ensembles les plus purs possibles.

Critère objectif : moyenne pondérée des impuretés

$$\boxed{\frac{n_G}{n_i} I_G + \frac{n_D}{n_i} I_D}$$

$\Rightarrow$ minimiser ce critère = **algorithme glouton** (greedy) :

- on considère tous les couples (variable,seuil)
- on évalue le critère
- on sélectionne le meilleur couple (variable,seuil)

Rappel : (1 variable / 1 seuil) = principe de l'algorithme

Arbres de classification - critères d'arrêt

Quand arrêter le découpage des noeuds ?

Arbres de classification - critères d'arrêt

Quand arrêter le découpage des noeuds ?

Critère d'arrêt automatique : plus de gain en pureté

- le noeud est déjà parfaitement pur
- on ne peut pas le purifier davantage

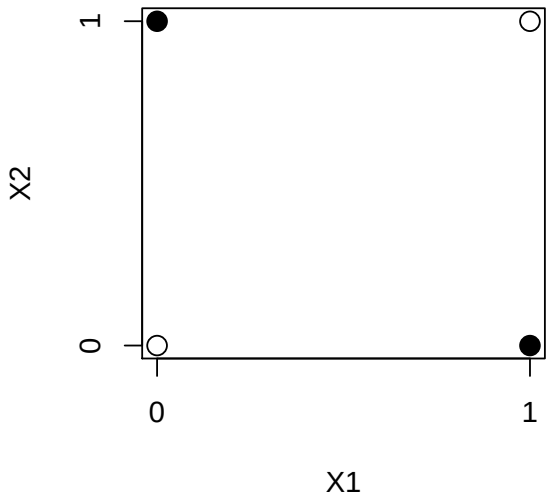
Arbres de classification - critères d'arrêt

Quand arrêter le découpage des noeuds ?

Critère d'arrêt automatique : plus de gain en pureté

- le noeud est déjà parfaitement pur
- on ne peut pas le purifier davantage

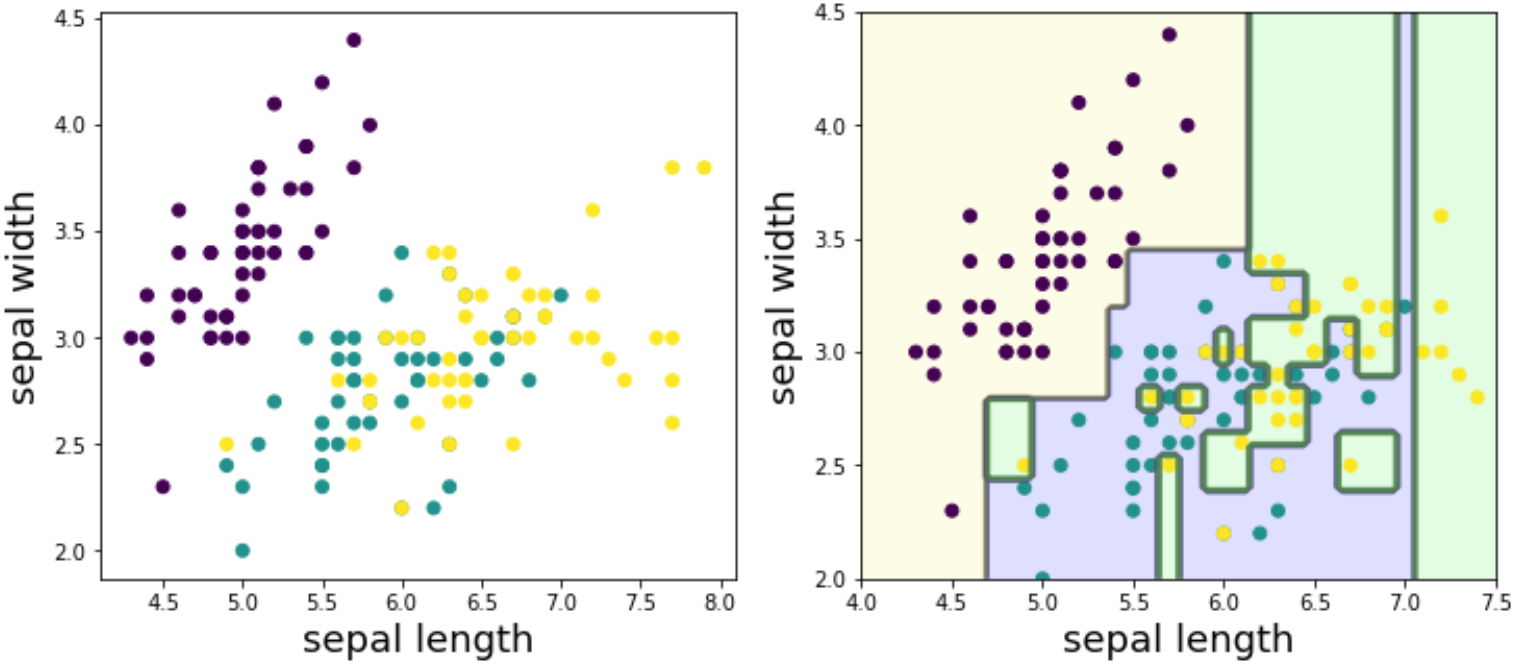
$\Rightarrow$ Exemple : XOR



- impureté maximum
  - $p_1 = p_2 = 0.5$
- constante si on découpe

Arbres de classification - critères d'arrêt

Illustration : `Iris` dataset



$\Rightarrow$ risque de sur-apprentissage si on laisse trop croître l'arbre

Arbres de classification - critères d'arrêt

Eviter le sur-apprentissage = limiter la complexité de l'arbre
$\Rightarrow$ compromis biais/variance

- simple : biais élevé ; complexe : variance forte

Arbres de classification - critères d'arrêt

Eviter le sur-apprentissage = limiter la complexité de l'arbre
$\Rightarrow$ compromis biais/variance

- simple : biais élevé ; complexe : variance forte

Critères d'arrêt classiques pour l'empêcher de trop croître :

1. s'arrêter quand le gain en pureté est marginal
2. critères sur la structure de l'arbre
   - profondeur maximale
   - nombre de feuilles maximal
   - nombre minimum d'instances par feuille
   - nombre minimum d'instances pour couper un noeud
   - ...

Arbres de classification - critères d'arrêt

Eviter le sur-apprentissage = limiter la complexité de l'arbre
⇒ compromis biais/variance

- simple : biais élevé ; complexe : variance forte

Critères d'arrêt classiques pour l'empêcher de trop croître :

1. s'arrêter quand le gain en pureté est marginal
2. critères sur la structure de l'arbre
   - profondeur maximale
   - nombre de feuilles maximal
   - nombre minimum d'instances par feuille
   - nombre minimum d'instances pour couper un noeud
   - ...

⇒ des hyperparamètres à définir et à optimiser

- e.g., par validation croisée

Arbres de régression

Pour la régression : même méchanisme

Critères d'impureté du noeud i :

1. erreur quadratique moyenne (MSE) :

$$\text{MSE}(i) = \frac{1}{n_i} \sum_{j=1}^{n_i} \left(y_{C_i(j)} - \bar{y}_i \right)^2$$

où $n_i = \#$ instances affectées au noeud i, $\bar{y}_i$ la moyenne de leurs réponses et $\{C_i(j)\}_{j=1:n_i}$ leurs indices.

2. erreur absolue médiane (MAE) :

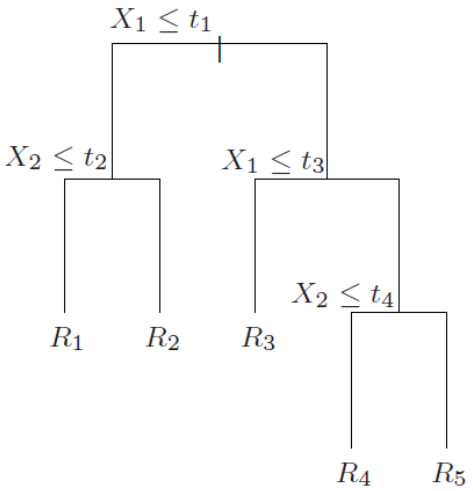
$$\text{MAE}(i) = \text{median}\left\{ \left| y_{C_i(j)} - \tilde{y}_i \right| \right\}$$

où $\tilde{y}_i$ = la médiane des réponses des instances du noeud i.

Règles de prédiction

Mécanisme "top down" pour la prédiction :

1. on part de la racine
2. on progresse dans l'arbre en évaluant les tests définis aux noeuds internes
3. jusqu'à atteindre une feuille



$\Rightarrow$ règles de prédiction :

- classification : classe majoritaire dans la feuille/région
  - **prédiction probabiliste** grâce à sa proportion
- régression : valeur moyenne dans la feuille/région

Arbres de décision - remarques

Avantages :

- interprétabilité
- simple : pas de préparation des données ; multiclasse

Limites :

- instabilité
- fonction non-continue pour la régression

Attention aux prédicteurs catégoriels

- relation d'ordre implicite si encodés comme $1, ..., K$

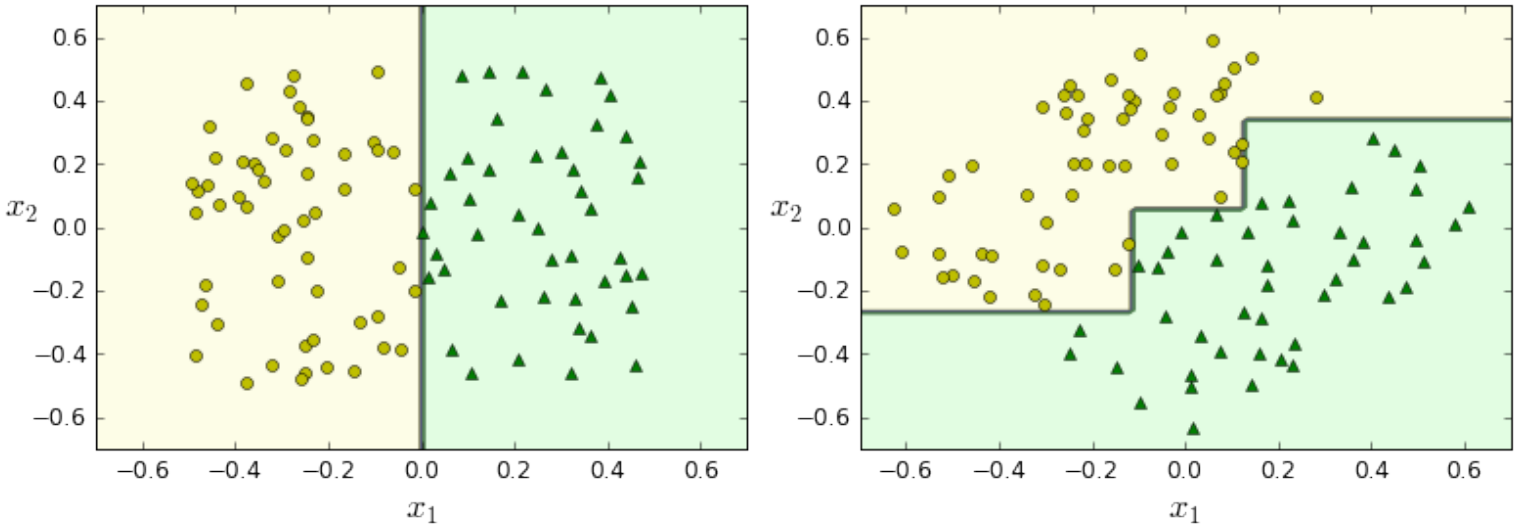
Approche "non paramétrique"

- le # de paramètres augmente avec le jeu de données

Algorithme CART le plus répandu (alternatives = C4.5, ID3)

Arbres de décision - instabilité

Illustration : instabilité à l'"orientation" des données [2] :



Plus généralement : procédure sensible à de petites variations des données d'apprentissage

⇒ le bagging permet de limiter cette instabilité

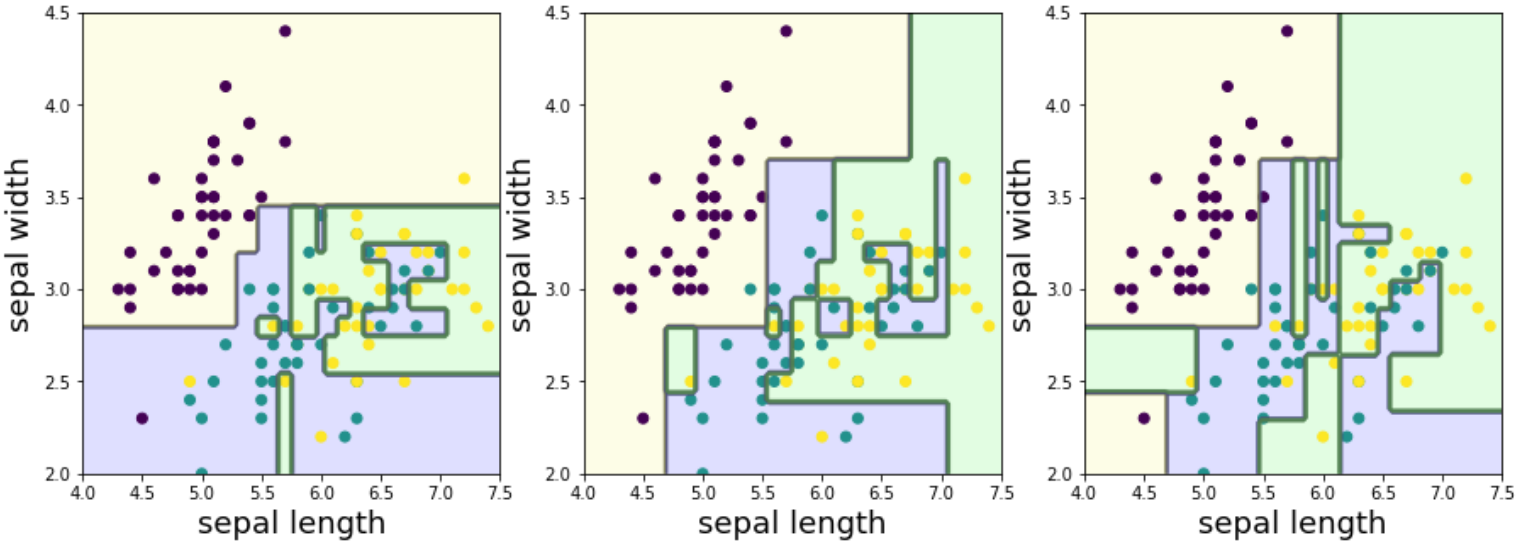
2. Example tiré de Géron (2017).

Bagging

Instabilité des arbres de décisions

Instabilité des arbres de décision : illustration (1/2)

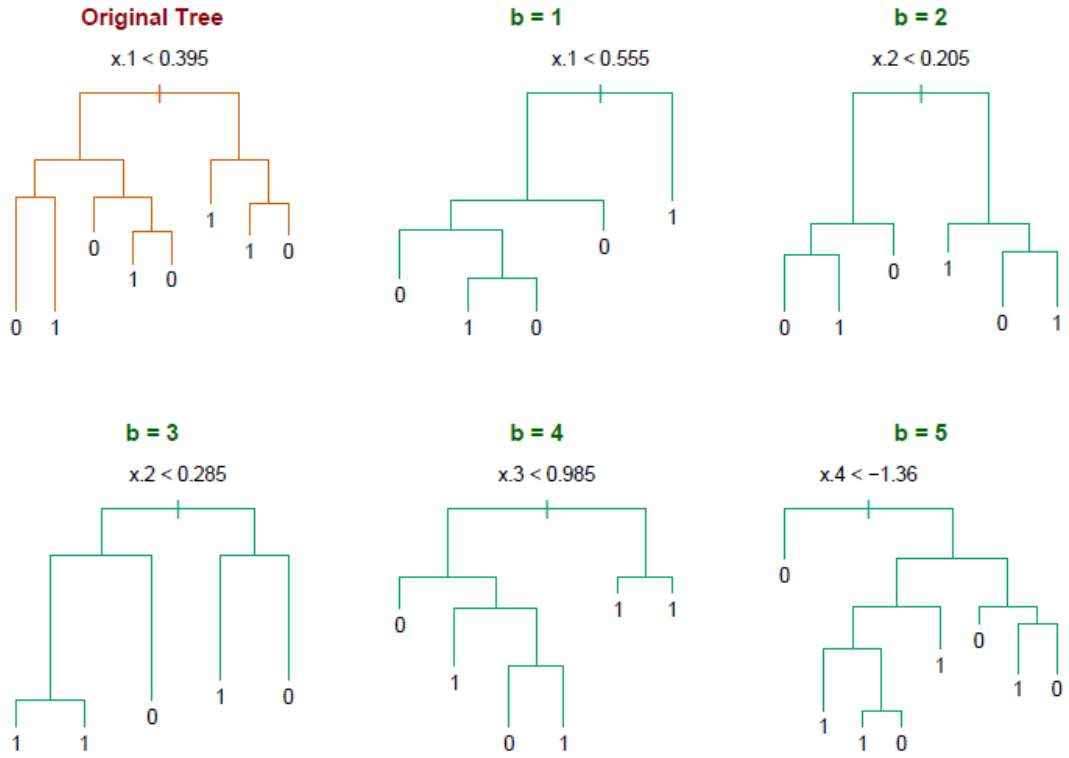
- jeu de données `Iris`, 3 échantillons bootstrap



Instabilité des arbres de décisions

Instabilité des arbres de décision : illustration (2/2)

- données simulées, 5 descripteurs (Hastie et al., 2001)
- 5 échantillons bootstrap



Instabilité des arbres de décisions

Instabilité = limitation sévère des arbres de décision

- compromet la généralisation
- remet en partie en cause leur interprétabilité

Instabilité des arbres de décisions

Instabilité = limitation sévère des arbres de décision

- compromet la généralisation
- remet en partie en cause leur interprétabilité

Stratégie de bagging :

1. générer plusieurs échantillons bootstrap du jeu d'apprentissage
2. construire les arbres correspondants
3. aggréger les prédictions de tous les arbres

⇒ bagging = bootstrap aggregating

Bagging

Bagging : principe générique

- pas spécifique aux arbres...
- ...mais souvent utilisé avec les arbres

Règles de prédiction (typiques) :

- régression : moyenne des prédictions
- classification : classe majoritaire

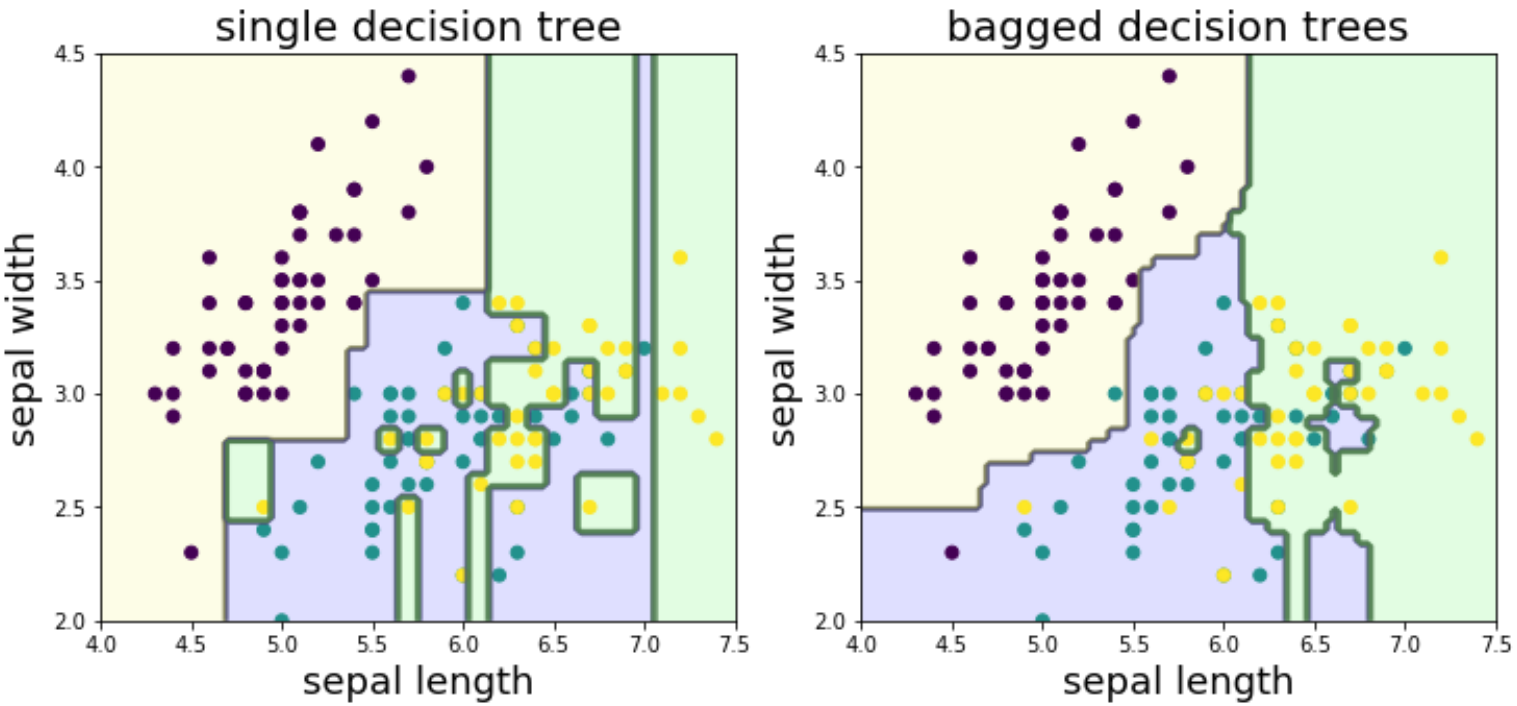
Prédiction probabiliste pour la classification ?

- la proportion de votes peut être optimiste
  - e.g., si ils prédisent tous la bonne classe avec $p = 0.6$
- considérer la moyenne des probabilités de chaque arbre

Bagging

Illustration : `Iris` dataset

- **gauche** : 1 arbre ; **droite** : bagging de 2000 arbres



$\Rightarrow$ capture des caractéristiques plus locales

Bagging et erreur "Out Of Bag"

Echantillon bootstrap : ne contient pas tout le jeu de donnée

Instances hors de l'échantillon = échantillon indépendant

- on peut les classifier par l'arbre correspondant

$\Rightarrow$ on peut estimer la généralisation lors de l'apprentissage

Bagging et erreur "Out Of Bag"

Echantillon bootstrap : ne contient pas tout le jeu de donnée

Instances hors de l'échantillon = échantillon indépendant

- on peut les classifier par l'arbre correspondant

$\Rightarrow$ on peut estimer la généralisation lors de l'apprentissage

Erreur Out Of Bag (OOB) - pour chaque instance :

1. extraire les arbres qui ne l'ont pas utilisée
2. obtenir les prédictions correspondantes
3. les aggréger et comparer à la vraie réponse

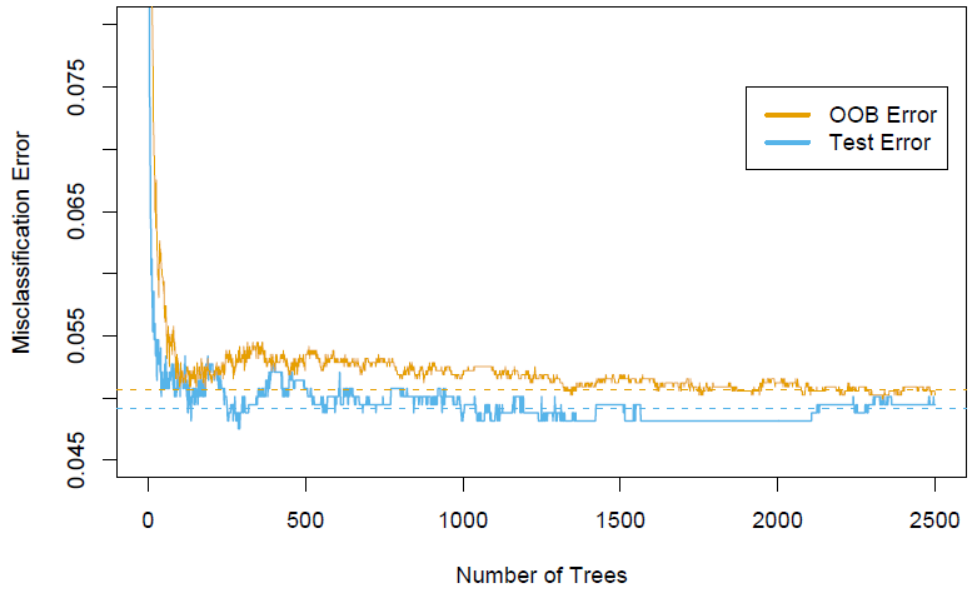
$\Rightarrow$ une caractéristique unique du bagging

Bagging et erreur "Out Of Bag"

Erreur Out Of Bag (OOB) :

- permet d'estimer la généralisation lors de l'apprentissage
- en général proche de l'estimation par validation croisée
- peut aider à savoir quand arrêter la procédure

Illustration (tirée de Hastie et al. (2001)) :



Bagging - remarques

Bagging = bootstrap aggregating

Avantages :

- stabilité $\rightarrow$ meilleure performance de prédiction
- simple à mettre en oeuvre
- accès à l'erreur Out Of Bag

Inconvénients :

- coût calculatoire
- avec les arbres : perte de l'interpretabilité

Random Forests / Forêts Aléatoires

Random Forests

Random Forest = algorithme de bagging basé sur les abres

- forêt aléatoire

Random Forests

Random Forest = algorithme de bagging basé sur les abres

- forêt aléatoire

Différence par rapport au bagging "classique" :

quand on ajoute un noeud à un arbre on considère
$m \leq p$ variables candidates choisies aléatoirement

Random Forests

Random Forest = algorithme de bagging basé sur les abres

- forêt aléatoire

Différence par rapport au bagging "classique" :

quand on ajoute un noeud à un arbre on considère
$m \leq p$ variables candidates choisies aléatoirement

$\Rightarrow$ chaque arbre a individuellement un pouvoir prédictif limité

- i.e., encore plus qu'un arbre classique

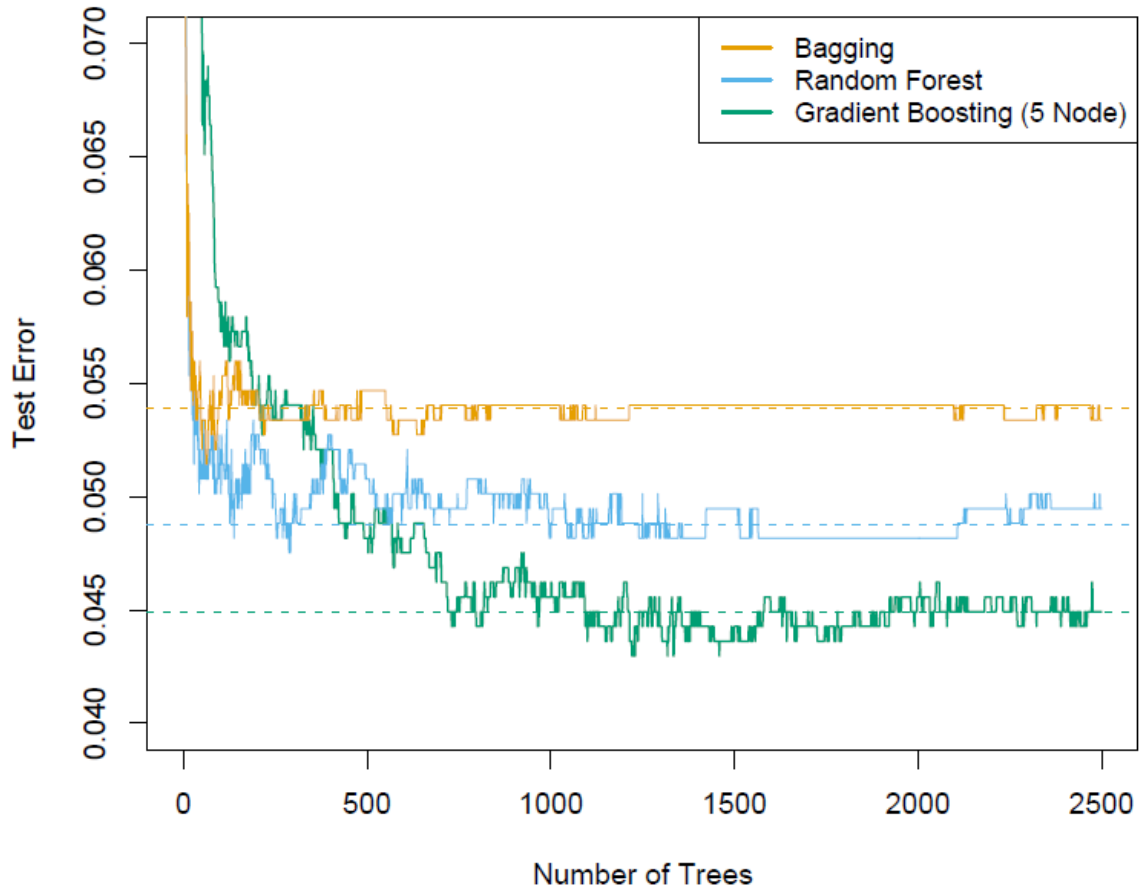
$\Rightarrow$ l'étape aléatoire a pour effet de les dé-corréler davantage

- i.e., encore plus qu'avec uniquement le bootstrap

$\Rightarrow$ les combiner par bagging = un classifieur très performant

Random Forests

Illustration : SPAMS dataset (Hastie et al., 2001) :



(Gradient boosting : autre algorithme non couvert ici)

Random Forests - mise en oeuvre

Même procédure pour classification et régression

- prédiction par bagging - moyenne ou vote majoritaire

Hyper-paramètres :

1. contrôle de la complexité des arbres
   - idem CART
2. nombre d'arbres à combiner
   - idem bagging
3. $m \leq p$: nombre de variables candidates à considérer

Random Forests - mise en oeuvre

Hyper-paramètres : à optimiser

- par validation croisée...ou grâce à l'erreur OOB

Mais :

1. Heuristiques pour le choix de m :
   - classification : $m = \sqrt{p}$
   - régression : $m = p/3$
2. nombre d'arbres : en ajouter ne peut pas faire de mal
   - vrai pour le bagging en général
3. pas nécessaire de contrôler la complexité des arbres (?)
   - e.g., noeuds de taille min. 1 / 5 pour classif / régression

$\Rightarrow$ simple à paramétrer

$\Rightarrow$ point de départ pour une optimisation plus fine

Critère d'importance des variables

Gain en pureté obtenu au noeud i d'un arbre :

$$\begin{aligned}\Delta(i) &= n_i I_i - \left(n_G I_G + n_D I_D\right) \\ &= n_i \left[I_i - \left(\frac{n_G}{n_i} I_G + \frac{n_D}{n_i} I_D \right) \right]\end{aligned}$$

où :

- le noeud i est coupé en (G, D) (Gauche, Droit)
- n_i = nombre d'instances affectées au noeud i
- I_i = critère d'impureté (e.g., Gini, MSE)

$\Rightarrow \sim$ (impureté initiale - moyenne pondérée des impuretés)
$\Rightarrow$ le facteur n_i reflète l'importance du noeud

- n_i elevé $\rightarrow$ i haut dans l'arbre $\rightarrow$ variable importante

Critère d'importance des variables

Critère d'importance d'une variable : somme du gain en pureté qu'elle apporte dans l'ensemble des arbres

Implémentation :

1. On le calcule pour chaque variable :
   - on parcourt tous les abres de la forêt
   - on parcourt tous les noeuds de l'arbre
   - si le noeud i est basé sur cette variable on ajoute $\Delta(i)$
2. On le normalise par la somme des valeurs obtenues

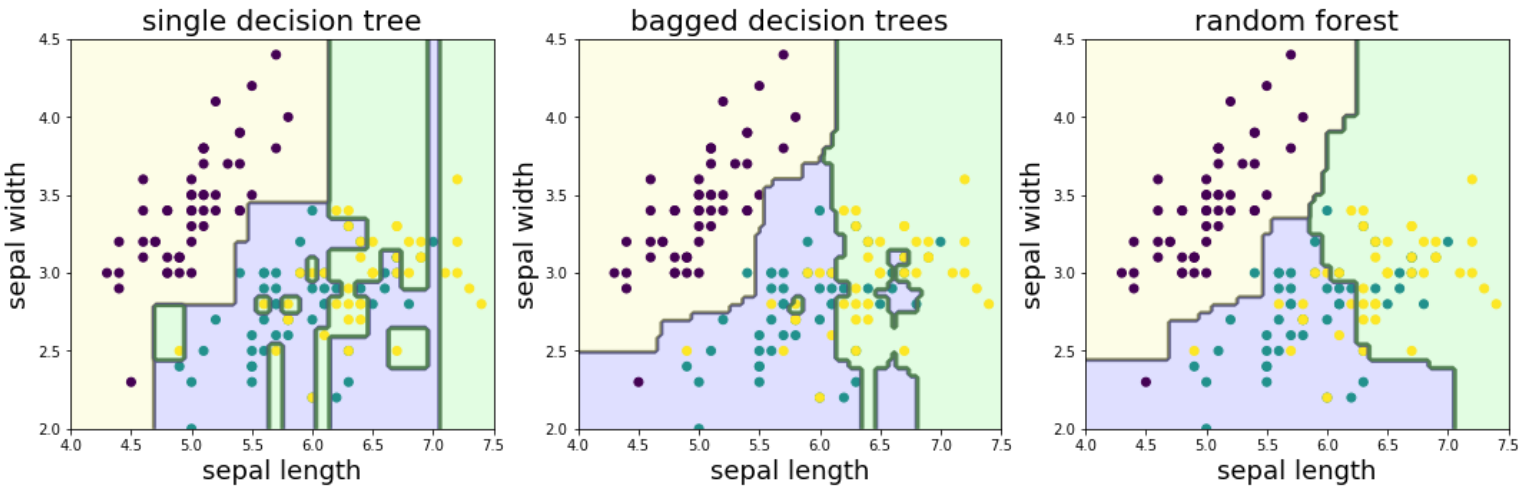
$\Rightarrow$ valeur élevée du critère = variable importante
$\Rightarrow$ somme à 1 sur l'ensemble des variables

(d'autres critères existent, celui-ci est implémenté dans `Sk-Learn`)

Illustration

Illustration : `Iris` dataset

- **gauche** : 1 arbre ; **milieu** : bagging ; **droite** : random forest



$\Rightarrow$ Out Of Bag accuracy :

- bagging $= 74.7\%$
- random forest $= 76\%$

Random Forests - remarques

Random Forests : bagging basé sur les arbres de décision

- sélection aléatoire des variables pour décorréler les arbres

Avantages :

- performant sur de nombreux problèmes
- simple à paramétrer (heuristiques)
- accès à l'erreur Out Of Bag (approche bagging)
- critère d'importance des variables

Inconvénients

- coût calculatoire
- modèle "boîte noire"
  - +/- car critère d'importance des variables...

Conclusion

Conclusion / bilan (1/2)

Ce cours : des arbres aux forêts aléatoires

Arbres de régression et classification

- algorithme CART
- critères d'impureté (Gini, entropie ; MSE, MAE)
- interprétabilité du modèle
- sur-apprentissage / instabilité
- critères de contrôle de complexité

Conclusion / bilan (2/2)

Bagging

- bootstrap aggregating
- principe général, décliné ici aux arbres
- stratégie ensembliste : vote majoritaire / moyenne
- erreur "Out Of Bag" accessible lors de l'apprentissage

Random Forests

- bagging d'arbres
- dimension stochastique pour décorréler les arbres
  - arguments théoriques
- critère d'importance des variables
- performant et simple à paramétrer (heuristiques)

Mise en oeuvre dans Scikit-Learn

Arbres de classification et régression (1/3)

Deux classes du module `tree` :

- `DecisionTreeClassifier` → classification
- `DecisionTreeRegressor` → régression

Paramètres principaux (constructeur) :

- `criterion` : critère d'impureté
  - `gini` (défaut) ou `entropy`
  - `mse` (défaut) ou `mae`
- critères de contrôles de complexité :
  - `max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes, ...`
  - `min_impurity_decrease`
- (+ `max_features` : pour réduire le nombre de variables candidates - pour les random forests)

Arbres de classification et régression (2/3)

Méthodes principales :

- `fit` et `predict`
- `DecisionTreeClassifier` : `predict_proba` et `predict_log_proba`
  - proba = proportion de la classe majoritaire
- `decision_path` : chemin suivi par une/des instance(s)

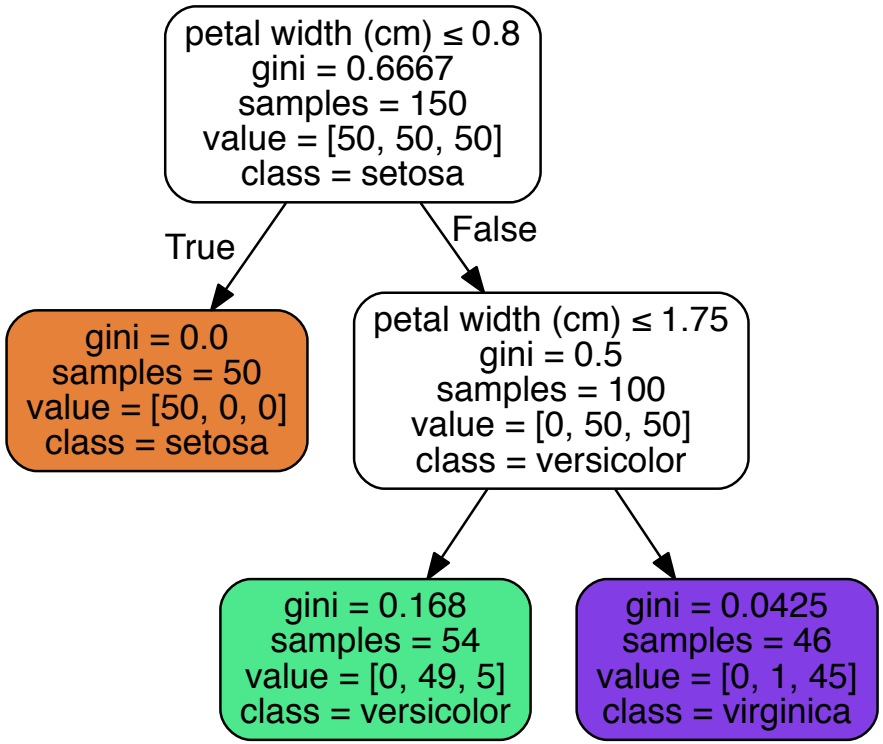
Exemple :

```
# import model
from sklearn.tree import DecisionTreeClassifier
# instantiace model
tree = DecisionTreeClassifier(max_depth=3)
# fit model
tree.fit(X_train,y_train)
# make predictions
preds = tree.predict(X_test)
probs = tree.predict_proba(X_test)
```

Arbres de classification et régression (3/3)

Outils de visualisation :

```
from sklearn.tree import export_graphviz
import graphviz
dot_data = tree.export_graphviz(tree, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("figure.pdf")
```



(NB : à customiser avec le nom des classes et des variables)

Bagging (1/2)

Deux classes du module `ensemble` :

- `BaggingClassifier` → classification
- `BaggingRegressor` → régression

Paramètres principaux (constructeur) :

- `base_estimator` : prédicteur de base
  - par défaut, un arbre de décision
- `n_estimators` : nombre de prédicteurs à aggréger
- `bootstrap` : booléen (vrai par défaut)
  - si faux, on tire sans remise : on parle de pasting
- `oob_score` : booléen indiquant si on veut le calculer
  - on le trouve alors dans le champ **oob_score_**

Bagging (2/2)

Méthodes principales :

- `fit` et `predict`
- `BaggingClassifier` : `predict_proba` et `predict_log_proba`

Exemple :

```
from sklearn.ensemble import BaggingClassifier
# define model
bag = BaggingClassifier(
    DecisionTreeClassifier(), n_estimators = 200,
    oob_score = True, random_state = 22)
# fit
bag.fit(X, y)
```

Random Forest (1/2)

Deux classes du module `ensemble` :

- `RandomForestClassifier` → classification
- `RandomForestRegressor` → régression

Paramètres principaux (constructeur) :

- `n_estimators` : nombre d'arbres dans la forêt
- `criterion` : critère d'impureté
- `max_features` : paramètre m
  - défaut : $\sqrt{p}$ pour classification, p pour régression
- critères de contrôle de la complexité des arbres
  - `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_leaf_nodes`, ...
  - `min_impurity_decrease`
- `oob_score` : booléen indiquant si on veut le calculer

Random Forest (2/2)

Méthodes principales :

- `fit` et `predict`
- `RandomForestClassifier` : `predict_proba` et `predict_log_proba`

Attribut particulier : **feature_importances_**

- le critère d'importance des variables (après appel à `fit`)

Exemple :

```
# fit random model
from sklearn.ensemble import  RandomForestClassifier
# instantiate model
rf = RandomForestClassifier(n_estimators = 500, oob_score = True, random_state= 20)
# learn model
rf.fit(X, y)
```

Références

A. Géron. *Hands-On Machine Learning with Scikit-Learn & TensorFlow*. O'Reilly, 2017.

T. Hastie, R. Tibshirani, and J.. Friedman. *The Elements of Statistical Learning*. Springer, 2001.