Post-élagage des arbres de décision

La méthode CART

Ricco RAKOTOMALALA
Université Lumière Lyon 2

Arbre de décision – Classification Tree

Algorithme d'apprentissage supervisé qui fait référence en machine learning

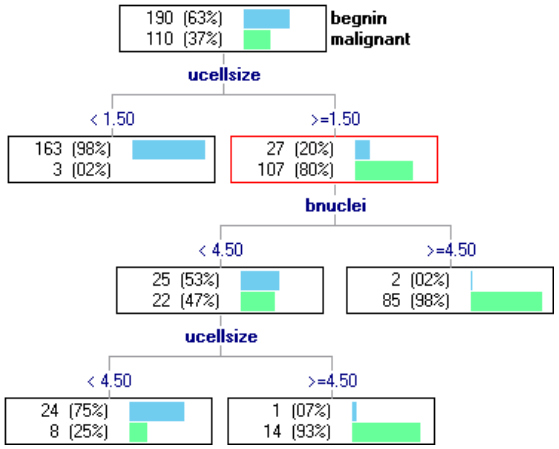
Objectif : Classement, prédire le mieux possible les valeurs d'une variable cible qualitative à partir d'un ensemble de variables explicatives quelconques (quantitatives, qualitatives). Le modèle s'exprime sous la forme d'un arbre de décision que l'on peut convertir sans pertes en un ensemble de règles logiques.

Quelques avantages :

- Modèle d'affectation explicite et très simple à appréhender
- Mécanisme intégré de sélection des variables pertinentes
- S'applique directement aux problèmes multi-classes
- Capacité à traiter des très grandes bases

Quelques inconvénients :

- Instabilité sur les petites bases
- Peut représenter des concepts complexes mais les algorithmes usuels peinent parfois à les identifier (ex. problème XOR)
- Performances en prédiction fortement dépendantes de la taille de l'arbre, paramètres associés difficiles à définir (pre-pruning)



Exemple d'arbre sur la base « Brest Cancer » (cible binaire, diagnostic = {begnin, malignant}). Logiciel SIPINA.

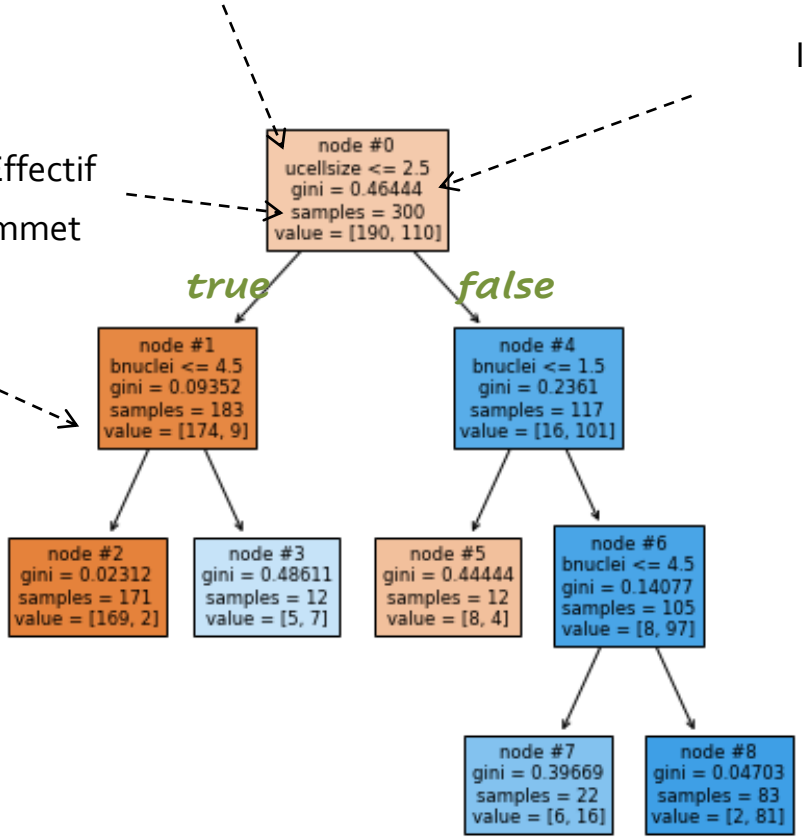
Arbre de décision

Informations associées à l'arbre (ex. scikit-learn 1.2.2 pour Python)

Variable de segmentation et seuil calculé

« samples » :Effectif associé au sommet

« value » : Distribution des classes



Indice de Gini, indice de diversité ou d'impureté

$$r(t) = \sum_{k=1}^{K} \frac{n_k}{n}\left(1 - \frac{n_k}{n}\right)$$

$$0.4644 = \frac{190}{300}\left(1 - \frac{190}{300}\right) + \frac{110}{300}\left(1 - \frac{110}{300}\right)$$

Remarque : l'on peut voir l'indice de Gini comme une variance pour variables qualitatives. L'arbre se présenterait comme une forme de décomposition de la variance (minimisation de l'impureté intra-feuilles).

1 feuille = 1 règle, par ex. Sommet #2

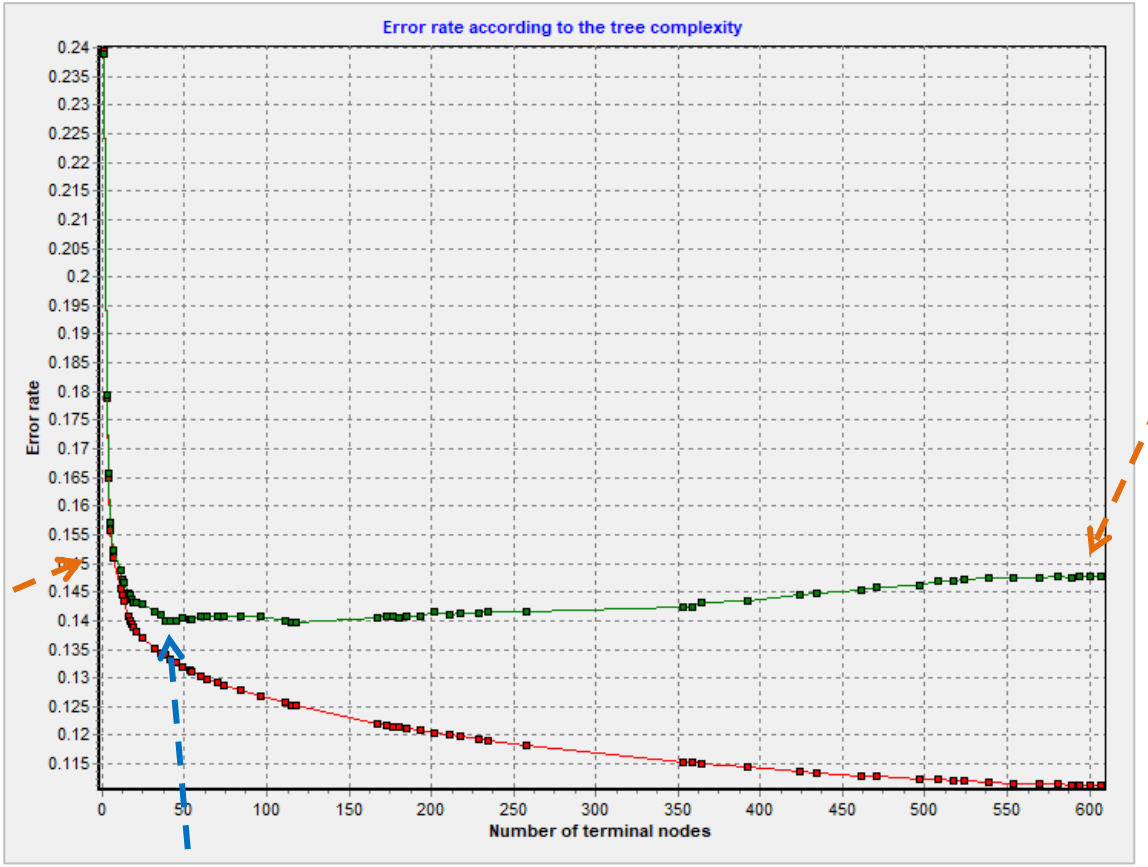
Si ucellsize <= 2.5 et bnuclei <= 4.5 Alors conclusion = 'begnin'

5 feuiles ➔ 5 règles prédictives ici (#2, 3, 5, 7 et 8)

Arbre de décision

Les performances de l'arbre dépendent fortement de sa taille

Erreur en apprentissage (resubstitution) vs. erreur en test.



Sous-apprentissage, l'algorithme n'exploite pas suffisamment les informations fournies par l'échantillon d'apprentissage

Sur-apprentissage, l'algorithme surexploite les spécificités de l'échantillon d'apprentissage, non transposable à la population.

La bonne taille de l'arbre (en nombre de feuilles) est ici. Comment identifier cette zone ? C.-à-d. comment paramétrer l'algorithme pour qu'il produise cette solution ? Sachant que l'échantillon test ne doit pas intervenir dans le processus !

PLAN

1. Position du problème
2. Techniques de pré-élagage (pre-pruning)
3. Technique de post-élagage – L'approche CART
4. Conclusion
5. Bibliographie

Techniques de pré-élagage

Intervenir durant la phase d'expansion de l'arbre, lors des segmentations successives

Pré-élagage

Définir la morphologie de l'arbre

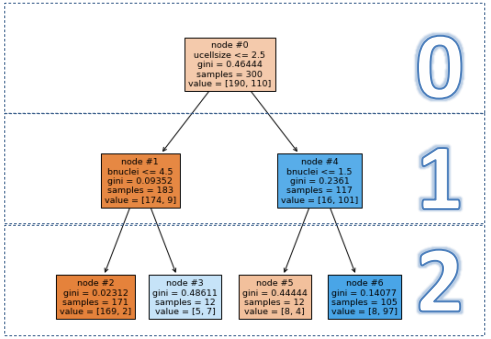
Pas vraiment pré-élagage au sens strict du terme, il s'agit ici de définir a priori la morphologie de l'arbre : profondeur, nombre de feuilles.

Ex. scikit-learn 1.2.2

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

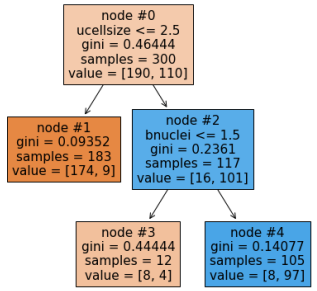
max_depth = 2



max_leaf_nodes : *int, default=None*

Grow a tree with `max_leaf_nodes` in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

max_leaf_nodes = 3



Il y avait une segmentation candidate sur ce sommet mais l'algorithme a préféré celle de droite parce qu'elle induisait une réduction relative de l'impureté plus élevée.

Pré-élagage

Basé sur le support (1)

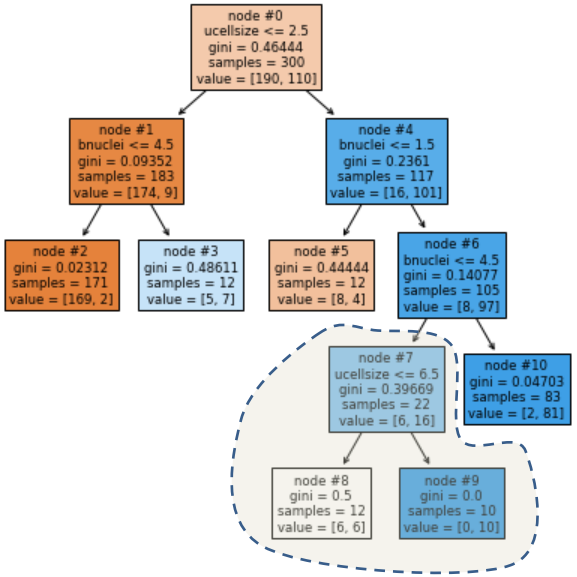
Une règle n'a d'intérêt que si elle concerne une fraction suffisamment représentative des individus ➔ Limiter la taille de l'arbre en fonction des effectifs des sommets.

Taille minimale pour segmenter.

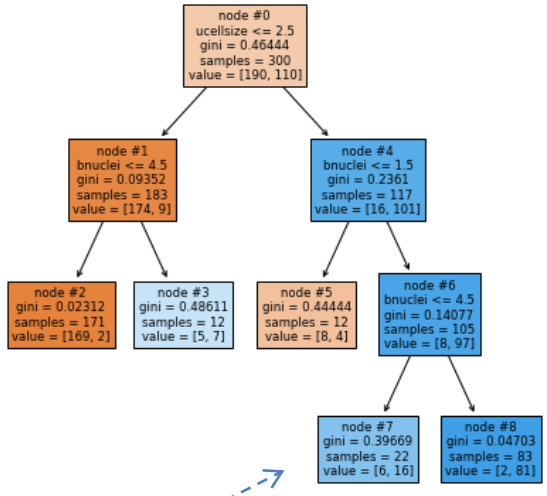
min_samples_split : *int or float, default=2*

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.



min_samples_split = 20



La segmentation n'a pas été initiée

min_samples_split = 25

Pré-élagage

Basé sur le support (2)

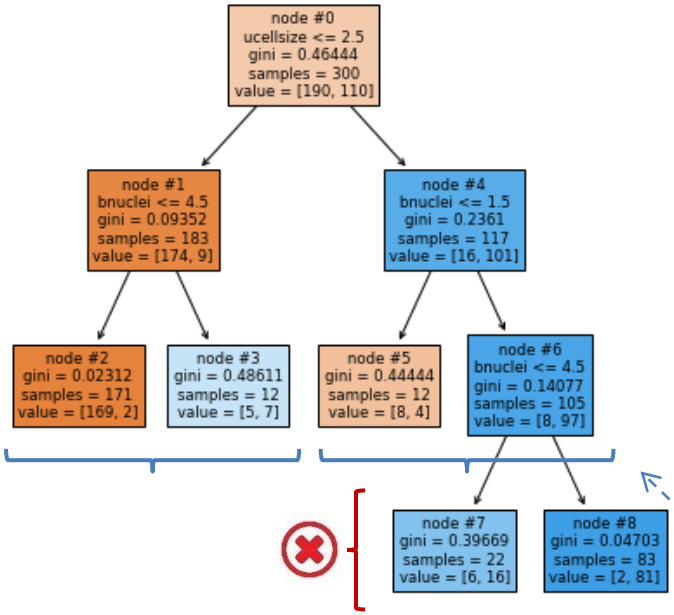
Effectif d'admissibilité

min_samples_leaf : *int or float, default=1*

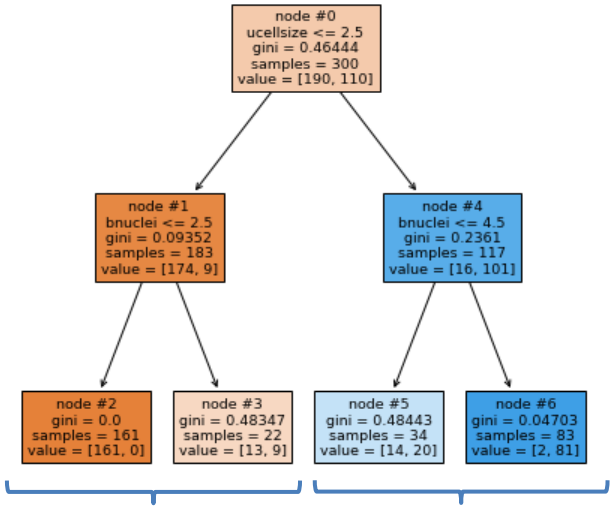
The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least `min_samples_leaf` training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

- If int, then consider `min_samples_leaf` as the minimum number.
- If float, then `min_samples_leaf` is a fraction and `ceil(min_samples_leaf * n_samples)` are the minimum number of samples for each node.

min_samples_leaf = 10



min_samples_leaf = 20



Change les caractéristiques des segmentations, et limite la profondeur de l'arbre.

Pré-élagage

Basé sur la qualité de la segmentation

La réduction de l'impureté est-elle significative lors d'une segmentation ?

min_impurity_decrease : *float, default=0.0*

A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

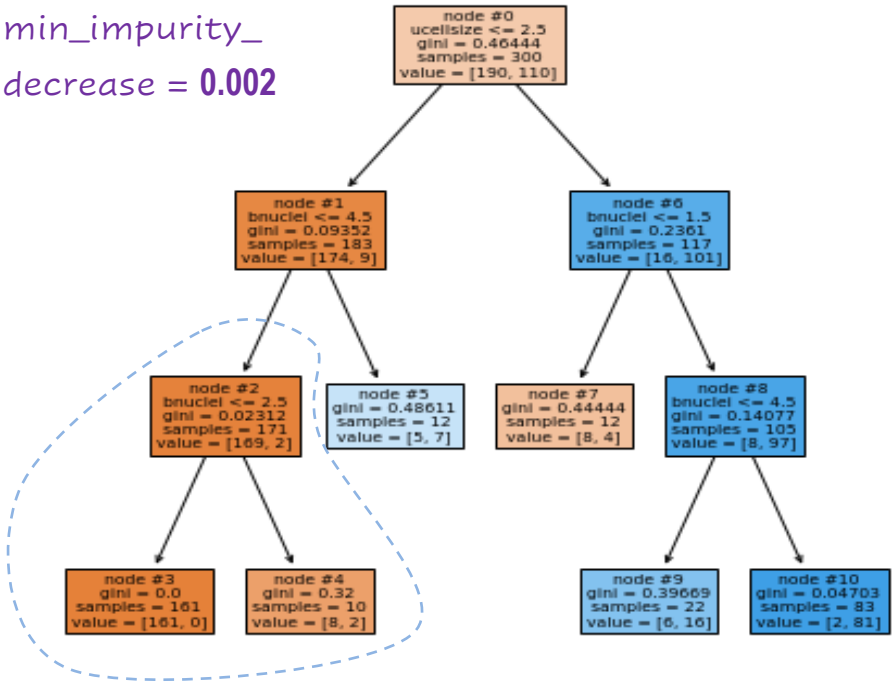
The weighted impurity decrease equation is the following:

```
N_t / N * (impurity - N_t_R / N_t * right_impurity
                    - N_t_L / N_t * left_impurity)
```

where `N` is the total number of samples, `N_t` is the number of samples at the current node, `N_t_L` is the number of samples in the left child, and `N_t_R` is the number of samples in the right child.

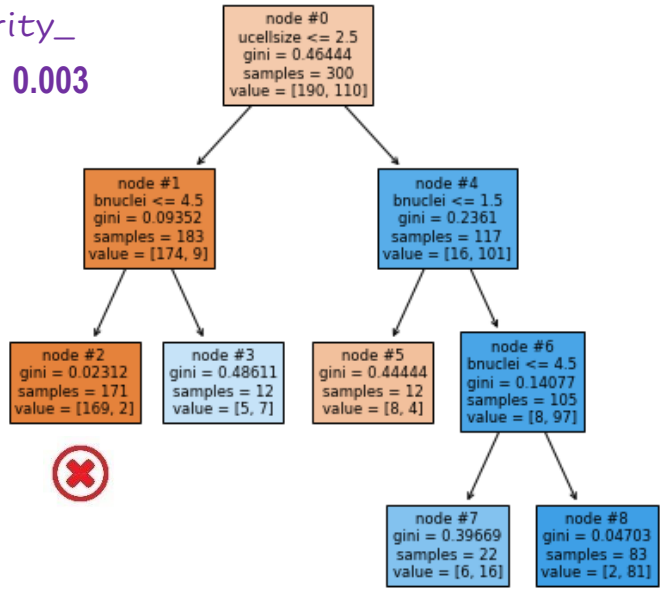
`N`, `N_t`, `N_t_R` and `N_t_L` all refer to the weighted sum, if `sample_weight` is passed.

min_impurity_decrease = **0.002**



$$\frac{171}{300}\left(0.02312 - \frac{161}{171} \times 0.0 - \frac{10}{171} \times 0.32\right) = 0.0025$$

min_impurity_decrease = **0.003**



Mais fixer la bonne valeur du seuil n'est pas des plus intuitif !

Techniques de post-élagage

La méthode CART

Classification And Regression Trees (Breiman et al., 1984)

Implémentation de la librairie « scikit-learn » pour Python

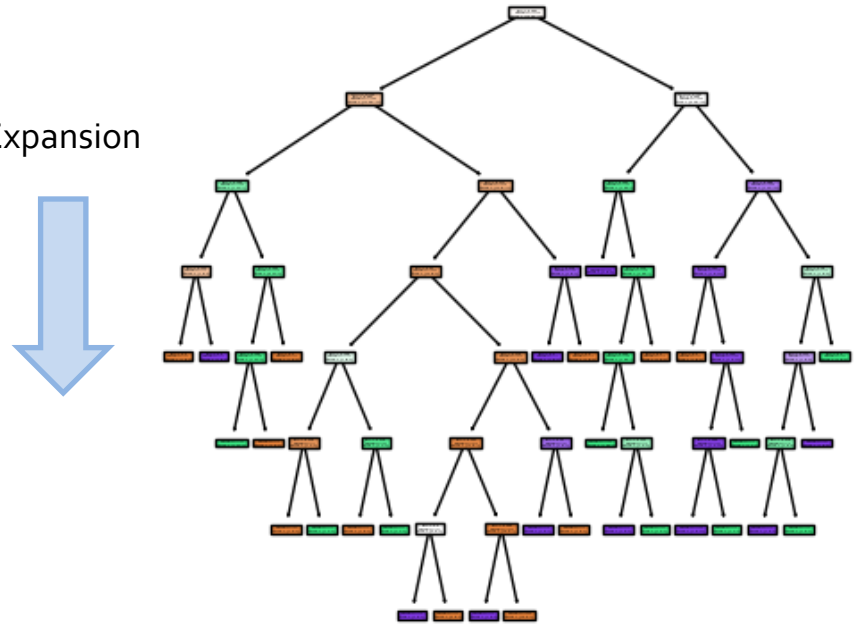
Post-élagage

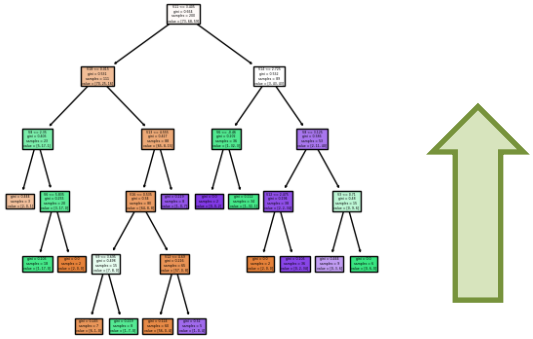
Mécanisme

Construire l'arbre en 2 temps :

1. Construire un arbre « maximal » avec un critère de pureté et sans pré-élagage pour appréhender un maximum d'information disponible (phase « growing »)
2. Le réduire en supprimant des sous-branches dans un deuxième temps en prenant un compte un critère de performances en prédiction (phase « pruning »).

Expansion



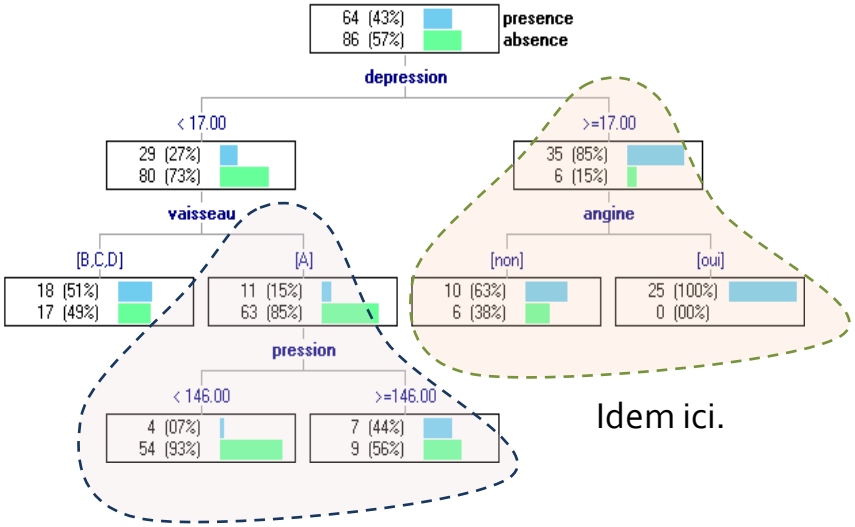


Réduction par post-élagage, suppression itérative des sous-branches de l'arbre « non-pertinentes » en partant du bas.

Post-élagage

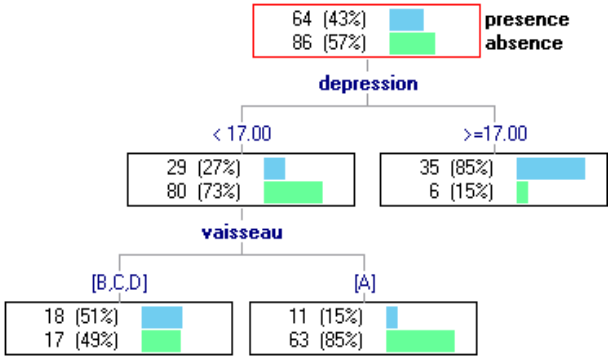
Approche basée sur les cohérences des prédictions

Une première approche évidente : élagage basé sur les inversions des prédictions sur les feuilles sœurs issues du même nœud père.



Idem ici.

Les deux feuilles mènent à la même conclusion, les supprimer (et donc convertir le sommet père en feuille) ne modifiera en rien les prédictions de l'arbre.



Arbre après post-élagage, avec exactement les mêmes propriétés prédictives c.-à-d. chaque individu de la population sera classé exactement de la même manière.

Post-élagage

Approches basées sur des heuristiques (modifient peu ou prou les caractéristiques prédictives du modèle)

Idées : Voir le post-élagage comme une forme de régularisation avec l'espoir que la simplification de l'arbre préserve ou, mieux, améliore les performances en prédiction.

Quelques références : Plusieurs approches existent (ex. C4.5 basée sur une estimation corrigée des proportions d'erreurs ; Decision Tree Learner de Knime, basée sur la théorie de la description minimale des messages [arbitrage entre la vraisemblance et la complexité] ; etc.) ; etc.

Approche CART : L'approche CART (Breiman et al., 1984) est très populaire parce qu'elle cumule les qualités : application des principes ci-dessus ; lissage de l'exploration de l'espace des solutions (ne nécessite pas d'explorer toutes les combinaisons possibles) ; possibilité d'intégrer la préférence à la simplicité avec la règle de l'écart-type ; outil graphique pour aider au paramétrage si on souhaite peser d'avantage sur la solution définitive. On s'intéresse à l'implémentation de la librairie « scikit-Learn » pour Python.

ccp_alpha : *non-negative float, default=0.0*

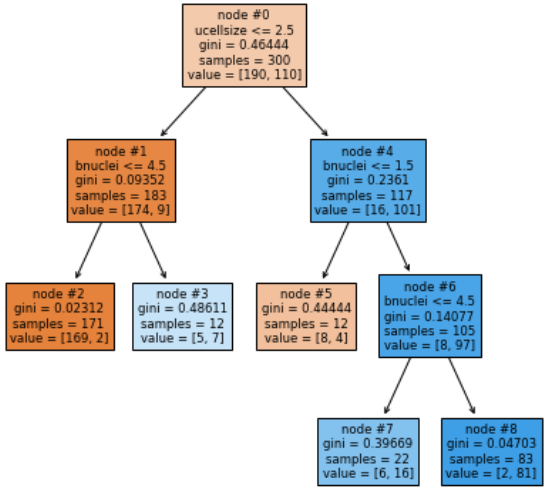
Complexity parameter used for Minimal Cost-Complexity Pruning. The subtree with the largest cost complexity that is smaller than `ccp_alpha` will be chosen. By default, no pruning is performed. See Minimal Cost-Complexity Pruning for details.

Post-élagage

CART - Notion de coût-complexité associé à un arbre

Coût complexité d'un arbre T

$$R_\alpha(T) = R(T) + \alpha\,|T|$$



($\alpha \geq 0$) est un paramètre de régularisation, plus il est grand, plus les arbres complexes (avec beaucoup de feuilles) sont pénalisés

|T| est le nombre de feuilles de l'arbre

R(T) est une mesure de qualité associé à l'arbre, souvent taux d'erreur dans la littérature, l'impureté pour la librairie « scikit-learn » (indice de Gini pour nous dans ce support)

$$R(T) = \frac{171}{300}0.02312 + \frac{12}{300}0.48611 + \frac{12}{300}0.44444$$

$$+\frac{105}{300}\,0.14077 + \frac{22}{300}0.39669 + \frac{83}{300}0.04703 = 0.0925$$

Pour un α fixé, l'algorithme d'élagage cherche à définir le sous-arbre qui minimise le coût-complexité (qui n'est pas sans rappeler les mesures telles que l'AIC ou le BIC que l'on connaît bien par ailleurs).

Mais comment choisir la valeur « optimale » de α ?

Post-élagage

Chemin de coût-complexité (1)

Il n'est pas nécessaire de tester au petit bonheur la chance (!) différentes valeurs de α. Avec le « chemin de coût complexité », nous pouvons calculer celles (α croissant) qui aboutissent à une succession d'arbres emboîtés de taille décroissante jusqu'à l'arbre réduit à la racine.

Pour chaque sommet non-terminal t, nous pouvons calculer son impureté r(t) et l'impureté des feuilles associées R(t). La valeur de α qui permet d'élaguer les branches correspondantes est définie par (|t| est le nombre de feuilles) :

$$r(t) + \alpha = R(t) + \alpha \times |t|$$

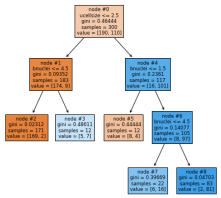
Par conséquent ⇒ $$\alpha(t) = \frac{r(t) - R(t)}{|t| - 1}$$

Remarque : Attention, il peut y avoir des ex-aequo c.-à-d. une même valeur de α peut amener à supprimer plusieurs branches en parallèle.

Post-élagage

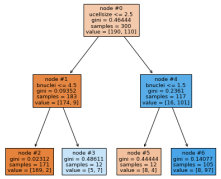
Chemin de coût-complexité (2)

Valeurs de α et scénarios d'arbres correspondants.



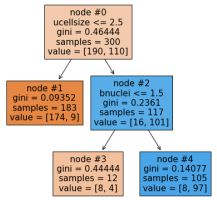
$\alpha = 0$

Arbre maximal, 5 feuilles



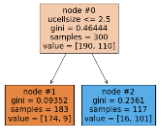
$\alpha = 0.00716$

Arbre à 4 feuilles



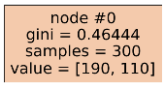
$\alpha = 0.02442$

Arbre à 3 feuilles



$\alpha = 0.025032$

Arbre à 2 feuilles



$\alpha = 0.31531$

Arbre réduit à la racine

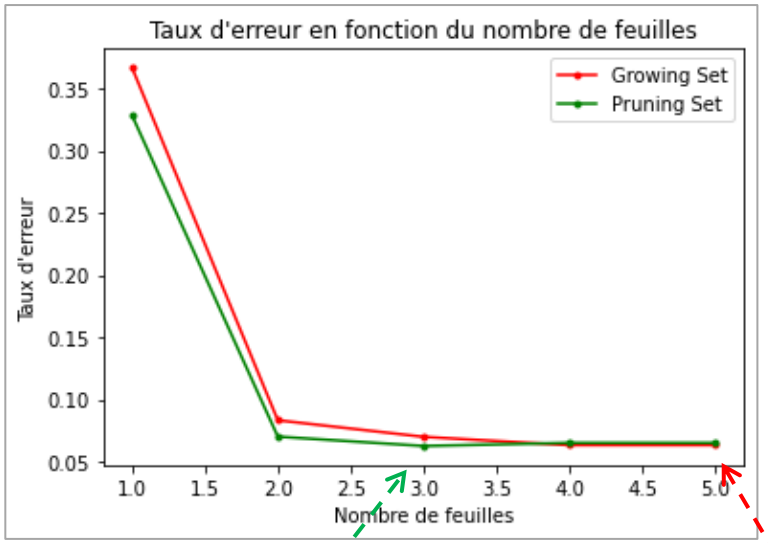
Remarque : scikit-learn 1.2.2 semble renvoyer des valeurs un peu différentes de la formule usuelle, mais l'idée est bien la même.

Lequel parmi ces arbres choisir ? Dit autrement, lequel de ces arbres est le plus performant en prédiction ?

Post-élagage

Mesure des performances prédictives

Le plus simple est d'utiliser un échantillon spécifique pour mesurer les performances prédictives. On parle, selon les logiciels, de « pruning set », ou « validation set », ou « tuning set ».



Remarque : la base « Breast Cancer » est un cas particulier très facile à apprendre. Les deux courbes se confondent presque.

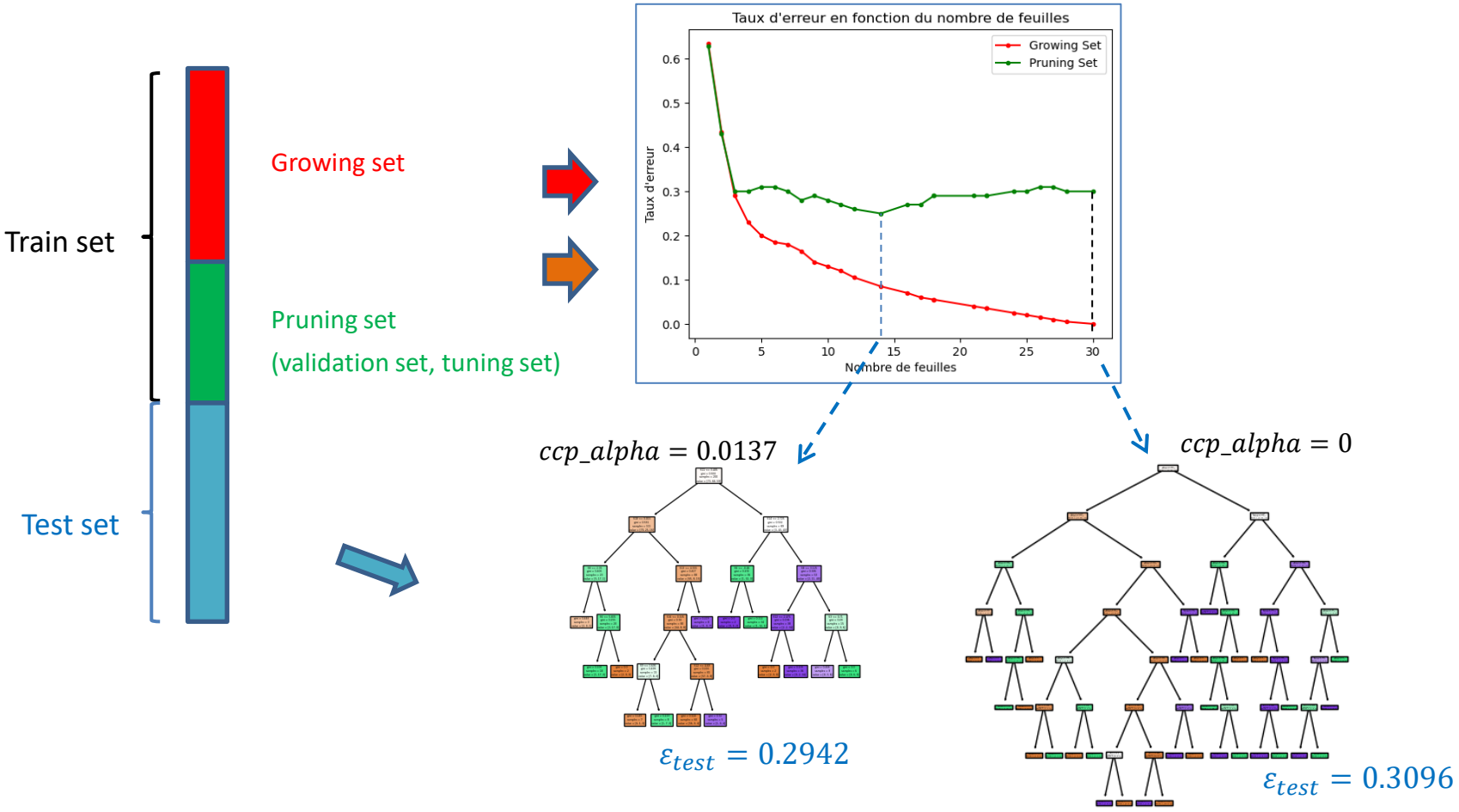
On constate sur le « pruning set » que l'arbre à 3 feuilles s'avère finalement être le plus performant.

Le taux d'erreur sur l'échantillon d'entraînement (« growing set ») baisse constamment à mesure que la taille de l'arbre augmente.

Post-élagage

Un exemple plus réaliste

« Post-élagage des arbres de décision – Python / Scikit-learn »
https://www.youtube.com/watch?v=if6QEtJP77E
Base de données « Waveform » (Breiman et al., 1984)



Remarque : Si la taille de la base est petite, réserver un échantillon à part supplémentaire (« pruning set ») est pénalisant. Nous avons intérêt à utiliser la validation croisée pour obtenir une mesure « honnête » des performances des arbres correspondants aux différentes valeurs de α (cf. « Build Better Decision Trees with Pruning », E. Krueger, juin 2021).

Post-élagage

Règle de l'écart-type (1-SE rule)
One Standard-Error Rule

CART introduit une régularisation supplémentaire (une préférence à la simplicité) avec la « règle de l'écart-type ».

(1) Plutôt que d'opter pour l'arbre « optimal » sur l'ensemble d'élagage (pruning set), et introduire peut-être une sur dépendance à cet échantillon (la mesure est entachée de variabilité liée à l'échantillonnage)

(2) On va choisir le plus petit arbre dont l'erreur ne s'éloigne pas trop de l'erreur de la solution optimale.

Si ε^* est l'erreur de l'arbre optimal (avec α^*), c'est une proportion, son écart-type s'écrit :

$$\sigma^* = \sqrt{\frac{\varepsilon^*(1-\varepsilon^*)}{n}}$$

n est la taille d'échantillon d'élagage

On choisira le plus petit arbre (avec $\alpha' \geq \alpha^*$) tel que l'erreur (ε') est :

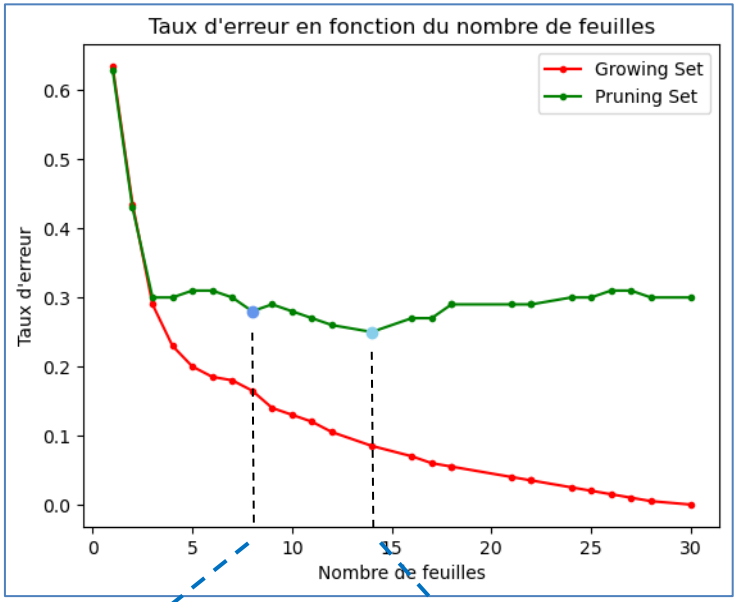
$$\varepsilon' \leq \varepsilon^* + 1 \times \sigma^*$$

Un peu comme une borne haute de l'intervalle de confiance de l'erreur optimale.

➔ L'idée est de simplifier d'avantage l'arbre tout en préservant les performances prédictives.

Post-élagage

Règle de l'écart-type
Exemple Waveform

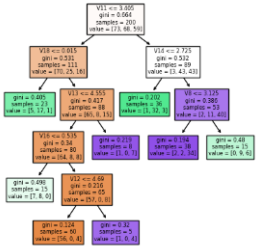


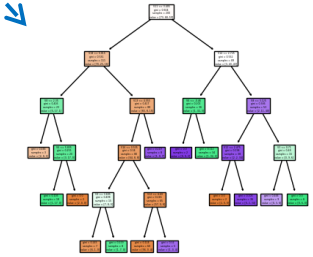
Plus petit arbre dont l'erreur en pruning est inférieure ou égale à la borne (0.2933)

Nombre de feuilles = 8

$\varepsilon' = 0.28$

$\varepsilon'_{test} = 0.2967$





Nombre de feuilles = 14

$\varepsilon^* = 0.25$

$\sigma^* = 0.0433$ $(n_{prune} = 100)$

➡ Borne erreur = 0.2933

$\varepsilon^*_{test} = 0.2942$

Conclusion : Le procédé « 1-SE Rule » permet de réduire encore plus la taille de l'arbre tout en préservant (au mieux) les performances en généralisation (en test). Les situations d'amélioration du taux d'erreur en test sont très rares.

Conclusion

Conclusion

- Les performances des arbres de décision reposent fortement sur la détermination de leur complexité (en nombre de feuilles pour un arbre binaire)

- Les techniques de pre-pruning agissent lors de la phase d'expansion (growing), définir les bonnes valeurs des hyperparamètres associés est très difficile.

- Les techniques de post-pruning agissent dans un deuxième temps. Après la construction d'un arbre maximal, elles le réduisent en supprimant les branches inefficaces.

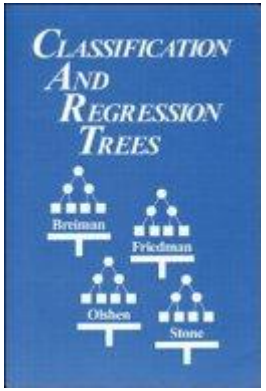
- La méthodologie CART propose une approche lisible pour définir des scénarios d'arbres concurrents et choisir la solution la plus performante. Nous pouvons de surcroit introduire une préférence à la simplicité additionnelle. La réduction de la taille de l'arbre est souvent spectaculaire dans les problèmes réels.

Bibliographie

Références

Breiman L., Friedman J., Olshen R.A., Stone C.J., « Classification and Regression Trees », Chapman & Hall, 1984.

Documentation « scikit-learn » 1.2.2, « Decision Trees ».



Tutoriels

YouTube, « Post-élagage des arbres de décision - Python / scikit-learn », mai 2023.

Tutoriel Tanagra, « La méthode CART dans Tanagra et R (package rpart) », septembre 2008.

Tutoriel Tanagra, « CART – Détermination de la taille de l'arbre », février 2008.