# Arbres de décision

Frédéric Santos
CNRS, UMR 5199 PACEA
Courriel : frederic.santos@u-bordeaux.fr

27 mars 2015

## Contenu du document



## 1. Principe général

Les arbres de décision constituent une méthode récente et efficace d'exploration de données, en vue de la prédiction d'une variable qualitative à l'aide de variables de tout type (qualitatives et/ou quantitatives). Cette flexibilité constitue un avantage par rapport à certains outils de classification, prévus pour des prédicteurs d'un seul et même type.

Il s'agit d'une méthode itérative, dite de *partitionnement récursif* des données. En effet, la méthode construit des classes d'individus, les plus homogènes possible, en posant une succession de questions binaires (de type oui/non) sur les attributs de chaque individu.

Contrairement à beaucoup d'outils de classification (régression logistique, SVM, etc.), les arbres de décision sont extrêmement intuitifs et fournissent une représentation graphique, parlante et facile à lire, d'un protocole de classification des individus. Cette représentation graphique est sous forme d'un arbre constitué de *feuilles terminales* (les classes d'individus) obtenues en suivant un chemin le long des *nœuds*, chaque nœud correspondant à une question binaire utilisant une variable du jeu de données.

Les arbres de décision permettent donc, dualement, d'identifier très rapidement les variables les plus discriminantes d'un jeu de données, en fonction de leur présence parfois répétée le long des nœuds.

Un raffinement de la méthode consiste à construire plusieurs arbres afin d'obtenir une *forêt aléatoire* disposant d'un plus fort pouvoir discriminant [Bre01]. Cette méthode ne sera pas traitée dans ce document, mais le package R `randomForest` [LW02], bien documenté, permet de réaliser de telles analyses. Les arbres de décision et les forêts aléatoires, fréquemment employés en médecine [KTK08, MSR+11], sont des outils très efficaces pour de nombreuses problématiques en anthropologie biologique [CHSD13, Lac13].

## 2. Exemple introductif

Un arbre de décision, cherchant la relation entre une variable (qualitative) d'intérêt $Y$ et divers prédicteurs $X_1, X_2, \cdots, X_p$, peut répondre à deux objectifs :

— *Exploration* : comprendre la structure d'un jeu de données, comprendre les relations entre différentes variables, voir les variables les plus discriminantes d'un jeu de données.

— *Prédiction* : à l'aide des règles de décision créées par l'arbre, pouvoir prédire la valeur d'un nouvel individu (i.e., l'affecter à la classe la plus probable de $Y$ en fonction des scores qu'il obtient sur les variables prédictives $X_1, X_2, \cdots, X_p$).

Un jeu de données (fictif) recense le résultat (Admis / Ajourné) de 39 étudiants à l'issue d'une L2 de mathématiques dans une petite université. Les variables étudiées sont :

— `Sexe` : le sexe de la personne (F : femme, M : homme) ;

— `Vit_Parents` : variable binaire indiquant si l'étudiant vit chez ses parents ;

— `Taux_Presence` : pourcentage de présence de l'étudiant en CM et TD durant l'année ;

— `Mention_Bac` : variable ordinale, échelonnée de 0 (sans mention) à 3 (Très Bien), donnant la mention obtenue par l'étudiant à son baccalauréat ;

— `Boursier` : variable binaire indiquant si l'étudiant bénéficie d'une bourse sur critères sociaux ;

— `Salarié` : variable binaire indiquant si l'étudiant dispose d'un emploi à temps partiel, sur ses journées de cours ou ses week-ends.

Un traitement basique du jeu de données par arbre de décision avec R produit la figure suivante :

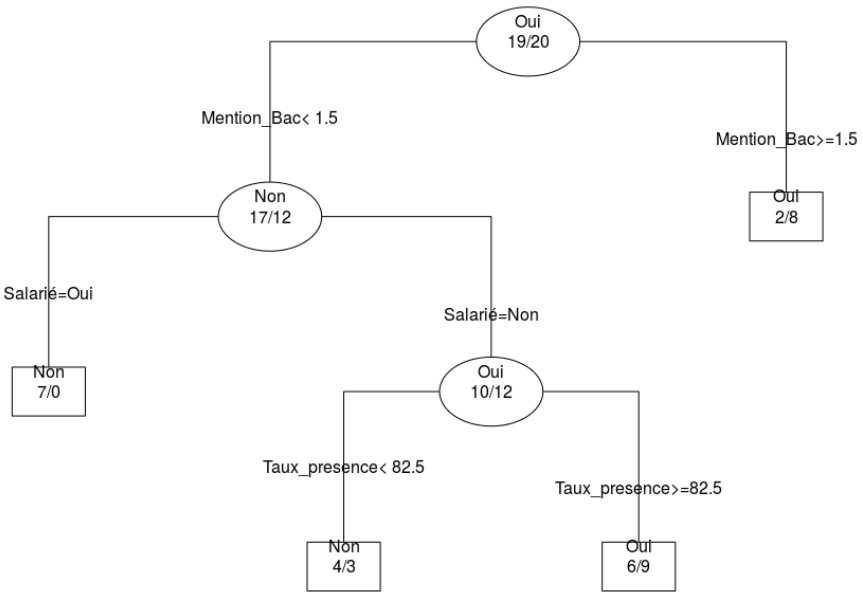


**Figure 1.** — Arbre de décision pour l'étude sur la réussite des étudiants

La lecture en est intuitive, en opérant une série de dichotomies partant de la racine de l'arbre (située en haut du graphique). L'arbre donne une série de règles de classification *conditionnelles* :

— le critère le plus discriminant parmi les variables étudiées est la mention au baccalauréat : les étudiants auteurs d'un bon parcours au lycée (mention Bien ou Très Bien) réussissent généralement leur L2 de mathématiques ;

— pour les étudiants n'ayant pas obtenu une telle mention par le passé (et seulement pour eux), le fait d'être salarié conduit à échouer en L2 ;

— et enfin, pour les étudiants sans bonne mention au bac et non salariés, le critère discriminant est le taux de présence en cours.

Sur la figure 1, les feuilles terminales (donnant la classe d'affectation d'un individu) sont indiquées par des rectangles, et la racine et les nœuds intermédiaires sont encerclés. La règle de décision associée à une feuille est le chemin parcouru pour arriver à cette feuille.

## 3. Données

Un arbre de décision construit avec l'algorithme CART peut fonctionner avec tous types de variables : qualitatives, ordinales et quantitatives continues. Il s'agit d'une grande force de cette méthode, qui permet de créer des règles de décision mixant tous types d'information, ce qui est particulièrement utile en anthropologie biologique. On pourra ainsi prédire le sexe d'un individu en tenant compte de relevés métriques, de l'observation de caractères biologiques discrets, ainsi que de relevés archéologiques indiquant la présence de mobilier.

## 4. Construction de l'arbre

**§4.1. Mesure de la pureté des feuilles.** — Le choix des « questions les plus discriminantes » afin de construire les nœuds de l'arbre peut se faire selon plusieurs critères : l'algorithme CART utilise l'*indice de Gini*, tandis que l'algorithme C4.5 utilise l'*entropie probabiliste*. Ces deux outils mathématiques visent à évaluer la « pureté » de chaque feuille : lorsque l'on se situe à un nœud donné de l'arbre, le but est de créer deux feuilles qui soient plus homogènes que le nœud qui les précède. Il faut donc disposer d'un moyen de mesurer cette homogénéité, ou « pureté ». Grâce à cela, à chaque nœud, le *split* est construit de manière à maximiser le gain d'information apporté par une question donnée sur la connaissance de la variable réponse.

*4.1.1. Entropie probabiliste.* — L'entropie probabiliste est une fonction mathématique créée par Claude Shannon en 1948, pour des questions initialement liées à la théorie du signal. La présentation ci-dessous est fortement inspirée de [PB07].

Définition 1 (Entropie d'une variable aléatoire discrète). — Soit $X$ une variable aléatoire discrète, prenant $n$ valeurs $x_1, \cdots, x_n$ de probabilités d'obtention respectives $p_1, \cdots, p_n$. On appelle entropie de $X$, généralement notée $H(X)$, la quantité

$$H_b(X) = -\sum_{i=1}^{n} p_i \log_b(p_i)$$

où $b$ désigne la base du logarithme (très souvent, $b = 2$).

L'entropie est souvent décrite comme une *mesure du désordre* : elle est nulle lorsque nous connaissons tout des valeurs produites produites par $X$ — typiquement, si $X$ est déterministe et ne renvoie qu'une seule valeur $x_0$ quoiqu'il arrive — et elle est maximale (positive) lorsque $X$ est uniformément répartie — c.-à-d., $p_i = 1/n$ pour tout $i$.

Le concept d'entropie peut assez bien se traduire intuitivement — au moins dans le cas discret. Étant donné que l'entropie mesure le désordre, considérons justement un cas concret de désordre : vos collègues vous font une bonne blague et cachent vos clés dans un des $n$ tiroirs — que l'on numérotera de 1 à $n$ — de l'armoire de la cafétéria. Afin de savoir où se trouvent vos clés, une stratégie basique peut être de poser $n$ fois à vos collègues la question : « sont-elles dans ce tiroir-là ? ». L'inconvénient est bien sûr que si $n$ est grand, vous risquez probablement de vous lasser — et eux aussi.

Une stratégie plus efficace peut être de procéder par dichotomie. Si $n$ est une puissance de 2, c'est-à-dire s'il existe $p \in \mathbb{N}$ tel que $n = 2^p$, il vous suffira tout d'abord de demander si vos clés se trouvent dans un tiroir compris entre 1 et $n/2$. Si la réponse est oui, on repose la question avec 1 et $n/4$, et ainsi de suite. Au total, seules $p = \log_2(n)$ questions auront été posées, ce qui est nettement plus rapide que dans le cas précédent [1]. On retiendra donc que le nombre de questions nécessaires pour localiser les clés parmi les $n$ tiroirs est $\log_2(n)$.

1. Si $n$ n'est pas une puissance de 2, un simple arrondi à la plus proche puissance de 2 (supérieure) réglera le problème et la stratégie de dichotomie fonctionnera tout autant

Imaginons désormais que vos collègues sont encore plus tordus et ont caché vos clés dans un des tiroirs de l'une des $K$ armoires d'un bâtiment. Chacune des $K$ armoires dispose de $n_i$ tiroirs (pour $i \in [\![1, K]\!]$), et on notera $n = n_1 + \cdots + n_K$ le nombre total de tiroirs, toutes armoires confondues, dans lesquels peuvent se trouver vos clés. Pour les cacher, vos collègues ont tout d'abord choisi aléatoirement une armoire en fonction du nombre de tiroirs qu'elle contenait : comme le jeu les amusera bien plus si l'armoire comporte beaucoup de tiroirs, ils se sont fixé pour règle de choisir l'armoire $i$ (pour $i \in [\![1, K]\!]$) avec probabilité $p_i = n_i/n$.

Si vos collègues vous révèlent, à un moment donné, que vos clés se trouvent dans l'armoire $i_0$, alors il suffira comme précédemment de poser $\log_2(n_{i_0})$ questions pour les situer. En revanche, si vous ne savez pas dans quelle armoire chercher, la meilleure stratégie est alors de visiter aléatoirement les armoires en leur donnant les mêmes probabilités $p_i$ que celles choisies par les farceurs ; on retrouvera alors les clés après avoir posé en moyenne un nombre de questions égal à

$$\sum_{i=1}^{K} p_i \log_2(n_i)$$

c'est-à-dire la moyenne sur les $K$ armoires du nombre de questions à poser par armoire, pondérée par les probabilités associées à chacune d'entre elles.

Se pose maintenant une question : laquelle des deux situations suivantes est la plus avantageuse, en termes de nombre de questions à poser ?

(i) Clés cachées dans un des $n$ tiroirs d'une armoire de façon équiprobable

(ii) Clés cachées dans un des $n_i$ tiroirs d'une des $K$ armoires, sachant que $n_1 + \cdots + n_K = n$ et que l'armoire a été choisie aléatoirement avec probabilité $p_i = n_i/n$

Autrement dit, est-il avantageux que les $n$ endroits où chercher soient regroupés en $K$ sous-ensembles auxquels sont rattachés des poids $(p_i)_{1 \leqslant i \leqslant K}$ ? Pour le savoir, il suffit de comparer le nombre de questions à poser dans les cas (i) et (ii), en calculant leur différence que l'on notera $\Delta$ :

$$\begin{aligned}
\Delta &= \log_2(n) - \sum_{i=1}^{K} p_i \log_2(n_i) = \log_2(n) - \sum_{i=1}^{K} p_i \log_2(n \times p_i) \\
&= \log_2(n) - \sum_{i=1}^{K} p_i \left(\log_2(n) + \log_2(p_i)\right) \\
&= -\left(\sum_{i=1}^{K} p_i \log_2(p_i)\right) \geqslant 0
\end{aligned}$$

Ainsi, $\Delta \geqslant 0$ et la situation (ii) est plus avantageuse : $\Delta$ représente l'économie de questions réalisée en sachant (en probabilité) dans quelle armoire se situe la clé. L'entropie est précisément $\Delta$, et atteint son maximum lorsque $p_1 = \cdots = p_K = 1/K$, c'est-à-dire dans le cas d'une distribution uniforme. Elle représente un indice de désordre associé à la distribution de probabilité $(p_i)_{1 \leqslant i \leqslant K}$.

*4.1.2. Indice de Gini.* — L'indice (ou coefficient) de Gini est une mesure, comprise entre 0 et 1, de la dispersion d'une distribution. Il est très souvent utilisé en économie ou en sociologie afin de mesurer les inégalités sociales au sein d'un pays. Dans ce contexte, plus le coefficient est proche de 1 et plus la société est inégalitaire.

On ne s'attardera pas ici sur les détails techniques concernant le coefficient de Gini, que l'on peut trouver par exemple dans [Bre01]. Il est néanmoins important de comprendre l'analogie avec l'entropie probabiliste : comme cette dernière, l'indice de Gini a pour but d'évaluer la pureté de chaque feuille, et de permettre la création de feuilles aussi homogènes (donc « égalitaires ») que possible.

**§4.2. Algorithme de construction.** — L'arbre de décision présenté à titre d'exemple en figure 1 illustre le principe de construction d'un arbre : celui du *partitionnement récursif*. Les individus sont tout d'abord séparés en deux sous-ensembles, puis chacun de ces sous-ensembles est à son tour séparé, etc. Progressivement, les individus de l'ensemble d'apprentissage sont divisés en utilisant des « questions » posées à l'aide des variables du jeu de données. Comme énoncé précédemment, ces questions sont choisies de manière à maximiser l'homogénéité des feuilles, c'est-à-dire à maximiser l'information apportée par les réponses.

On peut virtuellement continuer à construire l'arbre jusqu'à obtenir autant de feuilles que d'individus dans le jeu de données : chacune d'entre elles exprimerait alors la particularité d'un des individus. Bien entendu, cela n'aurait aucun intérêt scientifique, et il s'agit donc de trouver des critères d'arrêt. Il y a plusieurs façons d'arrêter le partitionnement récursif à un stade donné :

— obliger les feuilles terminales à contenir un nombre minimal d'individus (dès que R atteint une étape où une division supplémentaire entraînerait l'obtention de feuilles plus petites que le seuil désiré, le processus s'arrête) ;
— obliger l'arbre à ne pas contenir plus d'un certain nombre de feuilles ;
— interrompre le processus lorsqu'une division supplémentaire n'améliorerait pas la qualité discriminante de l'arbre.

L'algorithme de construction d'un arbre est alors le suivant :

(i) La situation actuelle vérifie-t-elle un critère d'arrêt ? Si oui, on s'arrête ici.
(ii) Si aucun critère d'arrêt n'est vérifié :
  (a) Parmi toutes les variables du jeu de données, trouver celle permettant de poser la question conduisant à un gain d'information maximal.
  (b) Partitionner les individus en fonction de la question trouvée, créer le nœud correspondant sur l'arbre.
  (c) Recommencer l'algorithme en partant du nœud nouvellement créé.

## 5. Élagage de l'arbre

Un objectif important pour toutes les méthodes de classification est d'éviter le sur-ajustement (*overfitting*), c'est-à-dire une capacité à décrire « exagérément » bien le jeu de données d'apprentissage tout en ayant de faibles capacités de généralisation ou de prédiction pour d'autres jeux de données. Dans un tel cas, le modèle construit sur-apprend les différences entre les classes d'individus propres à ce jeu de données : ceci se produit généralement à cause d'un problème de dimensionnement, se traduisant ici par la présence d'un trop grand nombre de feuilles terminales dans l'arbre de décision. Les premiers *splits* sont généralement les plus importants et les moins dépendants de l'échantillon, tandis que les suivants décrivent des particularités plus subtiles, pouvant être propres à l'échantillon dont on dispose. Il est donc souhaitable, afin de garder un niveau correct de généralité, d'*élaguer* l'arbre construit.

On procède classiquement par validation croisée : le jeu de données est divisé en sous-ensembles, jouant tour à tour le rôle d'échantillon d'apprentissage puis de validation. Par exemple, si le jeu de données est divisé en 3 sous-ensemble, les deux premiers peuvent dans un premier temps servir à construire l'arbre, dont on calculera un taux d'erreur de prédiction en utilisant le troisième jeu de données. Dans un second temps, les deux derniers joueront le rôle d'apprentissage tandis que le premier servira de jeu de validation, etc. Un taux d'erreur de prédiction global peut alors être calculé en réunissant les scores obtenus pour chaque sous-ensemble [2].

Un taux d'erreur de prédiction par validation croisée est calculé par R pour différentes tailles de l'arbre (i.e., différents nombres de feuilles terminales) : l'arbre est alors à élaguer au niveau offrant l'erreur minimale.

2. Une stratégie judicieuse est celle du *leave-one-out cross-validation* (LOOCV) : sur les $n$ individus du jeu de données, chaque individu est tour à tour pris seul pour la validation alors que l'arbre a été appris sur les $n - 1$ autres individus.

## 6. Gestion des données manquantes

Les arbres de décision sont très utiles dans le cas de données fortement lacunaires — cas convenant mal aux méthodes discriminantes classiques telles que la régression logistique ou l'analyse discriminante linéaire.

Les données manquantes posent plusieurs problèmes pour construire un arbre de décision.
- Imaginons que l'on dispose d'une variable possédant quelques données manquantes, mais permettant de poser une question extrêmement discriminante. Si l'on choisit cette variable pour procéder à un split, de quel côté du split doit-on envoyer les individus pour lesquels cette variable n'est pas renseignée ? Ou alors, est-il raisonnable d'ôter ces individus du jeu de données ?
- Imaginons qu'un arbre de décision est construit grâce à un jeu de données ne possédant aucune valeur manquante. En revanche, on souhaite utiliser cet arbre pour classer de nouveaux individus inconnus qui, eux, possèdent des valeurs non observées. Comment faire pour leur appliquer la règle de décision ?

**§6.1. Solutions élémentaires.** — Bien entendu, les méthodes « rustiques » communes à toutes les méthodes multivariées peuvent s'appliquer. La plus simple est de supprimer les individus ayant trop de valeurs manquantes. Si les valeurs manquantes sont rares et le nombre d'individu important, il suffit de supprimer les individus ayant au moins une valeur manquante : on se retrouve alors avec un jeu de données complet, et de taille suffisante. Cette solution est évidemment à écarter si le nombre d'individus est faible et le nombre de valeurs manquantes est important. Dualement, supprimer les variables ayant « trop » de valeurs manquantes peut être une approche nécessaire pour les méthodes multivariées classiques, mais n'est pas pertinent pour CART.

De manière générale, les solutions visant à agir directement sur le jeu de données *en amont* de l'analyse — ce qui est la démarche classique pour procéder à une régression logistique par exemple — n'ont ici que peu d'intérêt, puisque CART possède sa propre solution au problème des données manquantes.

**§6.2. Solution proposée par CART.** — CART utilise une procédure de *surrogate splits* (c'est, comme on le verra, la terminologie utilisée dans les résultats renvoyés par R), ou *variables-substituts*, pour s'affranchir du problème des valeurs manquantes. Cette procédure permet notamment de classer de nouveaux individus inconnus, même lorsqu'ils possèdent des valeurs manquantes pour les variables apparaissant sur l'arbre de décision.

L'idée est plutôt simple : lorsqu'on choisit un split basé sur une certaine variable $Z$ du jeu de données, on opère une partition des individus en deux groupes en fonction des valeurs qu'ils prennent pour cette variable $Z$. Or, il se peut qu'une autre variable $Z'$ du jeu de données permette de définir un split donnant sensiblement la même partition, i.e. les deux mêmes groupes. Il se peut même qu'une troisième variable $Z''$ permette de définir une autre partition restant encore relativement proche de celle obtenue grâce à $Z$. On dit alors que les variables $Z'$ et $Z''$ sont des *variables-substituts* à $Z$.

Ainsi, si un individu [3] arrive à un nœud où intervient une variable $Z$ qui correspond chez lui à une donnée manquante, on l'envoie alors à droite ou à gauche en fonction des valeurs qu'il obtient sur la variable-substitut $Z'$ — ou à défaut la variable $Z''$ si $Z'$ est également une donnée manquante chez lui, et ainsi de suite.

**§6.3. D'autres approches probabilistes.** — Il existe encore beaucoup d'autres approches pour le traitement des données manquantes dans un arbre de décision, et ce sujet constitue un sujet de recherche très actif en statistique théorique. Pour un aperçu de quelques autres méthodes modernes et efficaces, on pourra consulter [Haw08].

3. Cet individu peut aussi bien appartenir au jeu de données d'apprentissage qu'à un ensemble d'individus inconnus que l'on souhaite déterminer !

## 7. Mise en œuvre avec R

Pour l'heure, il n'existe pas de fonction ou de package offrant une interface graphique pour réaliser des arbres de décision avec R. Heureusement, les commandes sont relativement simples, et il suffit de trois à cinq lignes de code pour construire et afficher un arbre de décision, avec tous les diagnostics nécessaires à son élagage. La présente section part du principe que le lecteur est plus ou moins familier avec R — même si quelques principes de base seront rappelés. Si ce n'est pas le cas, on consultera avec profit [CGH+12].

Le package `rpart` [TAR13] est dédié aux arbres de décision. Sur un ordinateur connecté à Internet, installer ce package en tapant dans une console R la commande suivante :

```
install.packages("rpart", dep=TRUE)
```

Ensuite, à chaque ouverture d'une session R, ce package devra comme d'habitude être chargé grâce à la commande `library(rpart)`.

Pour servir de support à tout ce qui suit, on utilisera un jeu de données de démonstration inclus dans le package `rpart` : `kyphosis`. Pour le charger, exécuter la commande `data(kyphosis)`ou passer par le menu Données > Données dans les paquets > Lire des données depuis un paquet attaché de l'interface R Commander. Visualiser et obtenir un résumé de ce jeu de données :

- la variable Kyphosis indique si l'enfant a subi une déformation de la colonne vertébrale après une opération chirurgicale ;
- la variable Age donne l'âge de l'enfant (en mois) ;
- la variable Number indique le nombre de vertèbres concernées par l'opération ;
- la variable Start indique le numéro de la première vertèbre opérée.

On peut par exemple se demander si l'âge de l'enfant, le nombre de vertèbres opérées et la position des vertèbres opérées permettent d'évaluer le risque de survenue d'une cyphose à l'issue de l'opération. Pour cela, on va construire un modèle d'arbre de décision grâce à la commande suivante :

```
> arbre = rpart(Kyphosis ~ ., method="class", minsplit=20, xval=81, data=kyphosis)
```

Quelques explications :

- `arbre` est le nom choisi pour l'objet R qui contiendra les résultats ;
- `rpart` est le nom de la commande permettant de créer un arbre de décision ;
- la définition du modèle passe par « Kyphosis ~ . », qui est une *formule* R. La tilde signifie « est à expliquer en fonction de » : on met donc à gauche de la tilde la variable que l'on souhaite expliquer (ici, Kyphosis) et à droite les variables explicatives séparées par des `+`. Ici, on a mis un point à droite : cela signifie « tout le reste ». On aurait donc tout aussi bien pu écrire : `Kyphosis ~ Age + Number + Start`, le modèle aurait été équivalent... mais beaucoup plus long à écrire ;
- `method = "class"` est ici un argument totalement optionnel, mais il permet de rappeler à R que la variable à expliquer est qualitative (c'est une sécurité supplémentaire) ;
- `minsplit=20` signifie qu'il faut au moins 20 individus dans un nœud pour tenter un *split* ;
- l'argument `xval` permet de régler la stratégie de validation croisée : on y indique le nombre de « portions » du jeu de données à créer pour effectuer la validation croisée. Par exemple, `xval = 2` signifiera qu'on divisera le jeu de données en deux parties égales : l'une correspondant à un jeu d'apprentissage, l'autre à un jeu de validation. Ici, l'argument `xval = 81` correspond à la stratégie classique de leave-one-out (LOOCV), puisqu'il y a 81 individus dans le jeu de données ;
- enfin, l'argument `data=kyphosis` permet d'indiquer le nom du jeu de données à utiliser.

On peut obtenir un résumé textuel des *splits* construits en tapant `arbre`, mais les résultats affichés sont peu lisibles, en particulier si l'arbre construit est de grande taille. En revanche, la commande `summary(arbre)` possède un intérêt si l'on désire connaître les variables-substituts intervenant à chaque nœud : elles sont indiquées sont l'appellation `surrogate`. Le nombre de variables-substituts à rechercher pour chaque nœud peut être défini grâce à l'argument `maxsurrogate` de la commande `rpart` (consulter l'aide de la commande pour plus d'informations).

Les commandes suivantes permettent d'afficher l'arbre de décision sous forme graphique :

```
> plot(arbre, uniform=TRUE, margin=0.1, main="Arbre de décision")
> text(arbre, fancy=TRUE, use.n=TRUE, pretty=0, all=TRUE)
```

La commande `plot` ne trace que l'armature de l'arbre, et la commande `text` y ajoute les étiquettes textuelles (dans les nœuds intermédiaires et les feuilles terminales). Ces deux commandes possèdent de très nombreuses options graphiques pour personnaliser l'apparence esthétique de l'arbre : consulter l'aide de ces fonctions [4] pour trouver les réglages correspondant le mieux à votre goût. Ici, les paramètres pris par ces deux commandes ne sont donnés qu'à titre d'exemple et n'ont qu'une importance cosmétique. L'exécution des commandes précédentes permet l'obtention de la figure 2.

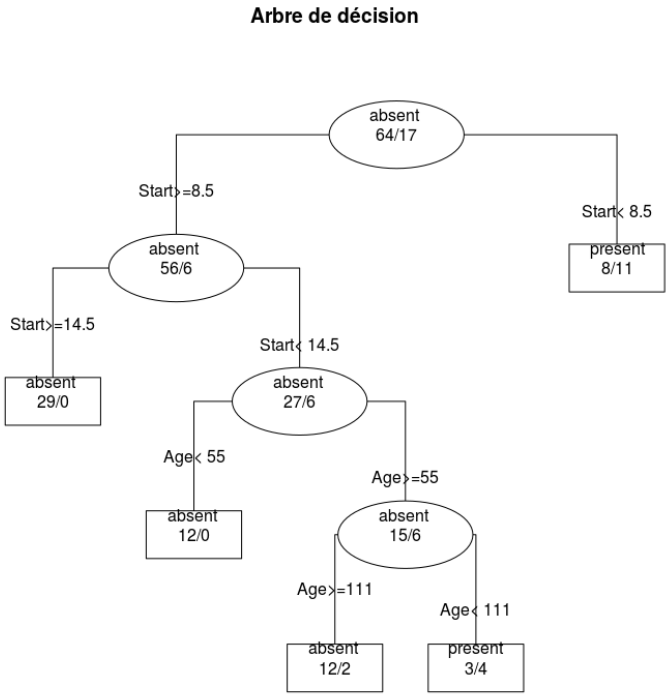


**Figure 2.** — Arbre de décision pour la prédiction de la variable Kyphosis

À ce stade, on peut souhaiter élaguer l'arbre de décision. Pour cela, il est conseillé d'exécuter la commande `plotcp(arbre)` pour déterminer la taille optimale (ou `printcp(arbre)` pour obtenir le même résultat mais sous la forme d'un tableau et non d'un graphique) : cf. figure 3.

On y constate que l'erreur de prédiction commise en validation croisée est minimale si `size of tree = 2`, c'est-à-dire si l'arbre ne comporte que deux feuilles terminales. L'arbre correspondant a une complexité `cp = 0.059` : c'est de cette façon que l'on va indiquer à R la taille d'arbre souhaitée, grâce à la commande suivante.

```
> arbre2 = rpart(Kyphosis ~ ., cp=0.059, data=kyphosis)
```

On demande alors l'affichage de cet arbre élagué, représenté en figure 4. Son taux final de mauvais classement (en apprentissage) est égal à $\frac{6+8}{56+6+8+11} \approx 0.172$, soit 17.2% environ. Cette erreur est celle obtenue sur les données ayant servi à la construction de l'arbre : c'est donc probablement une vision optimiste de la performance réelle de cette arbre — qui fonctionnera sans doute moins bien sur des données inconnues. Quelques lignes de code permettent d'estimer rapidement le taux d'erreur de prédiction en utilisant encore une fois une procédure de validation croisée : consulter [CGH+12] pour plus d'information.

4. Par exemple en exécutant les commandes `help(plot.rpart)` et `help(text.rpart)`

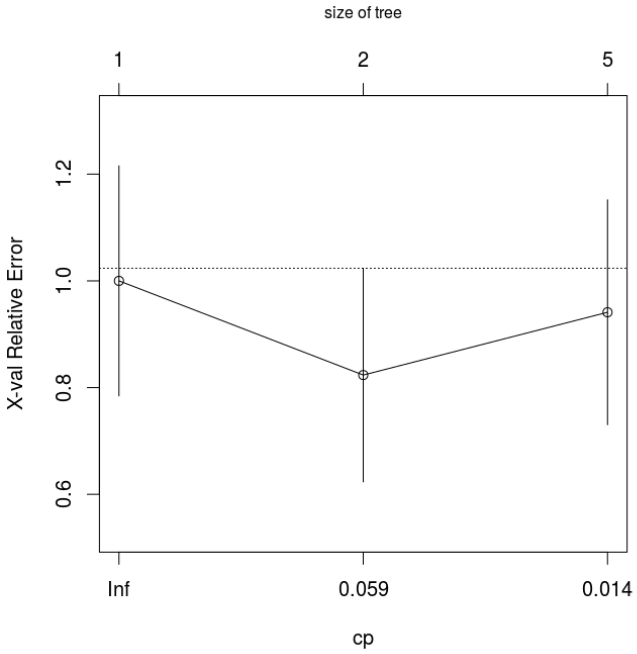


**Figure 3.** — Erreur commise en validation croisée en fonction de la taille de l'arbre

Désormais, l'arbre de décision élagué fournit la règle de décision optimale (au vu des variables prédictives dont on dispose) pour prédire la variable Kyphosis. On peut être conduit à appliquer ultérieurement cette règle sur de nouveaux enfants qui vont être opérés, de manière à estimer avant l'opération leur risque de développer une cyphose. On construit arbitrairement un tableau de données correspondant à trois enfants sur le point d'être opérés :

```
> inco = data.frame(c(60,30,90), c(1,2,3), c(5,12,16))
> colnames(inco) = c("Age", "Number", "Start")
```

L'objet R `inco` est alors le data frame suivant [5] :

```
  Age Number Start
1  60      1     5
2  30      2    12
3  90      3    16
```

La prédiction de la variable Kyphosis pour ces trois individus s'opère alors par la commande `predict` (dont le premier argument est le nom du modèle utilisé, et le second, `newdata`, est le nom du data frame contenant les nouveaux individus) :

```
 > predict(arbre2, newdata=inco, type="class")

     1       2       3
present  absent  absent
```

Selon le modèle, seul le premier enfant semble devoir développer une cyphose à l'issue de son intervention chirurgicale. Pour obtenir les probabilités associées aux modalités `absent` et `present` pour chaque individu, il suffit de retirer l'argument `type="class"`.

5. Il est capital que le data frame contenant les individus à prédire soit calqué sur celui ayant servi à la construction de l'arbre : mêmes noms de colonnes (casse incluse), mêmes types de variables, etc. En particulier, si l'une des variables prédictives est un facteur, on ne doit pas introduire chez les nouveaux individus un niveau de facteur qui était inconnu ou absent lors de l'apprentissage de la règle de décision.

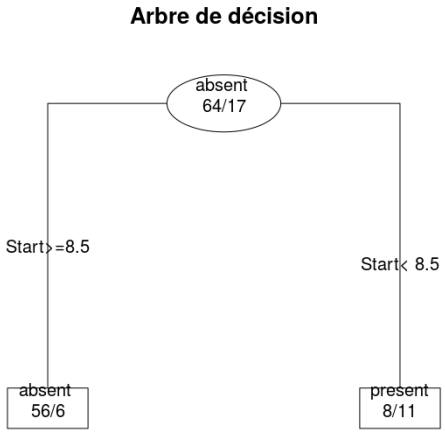


**Figure 4.** — Arbre de décision élagué pour la prédiction de la variable Kyphosis

## Références

[Bre01] Leo Breiman : Random forests. *Machine Learning*, 45(1):5–32, 2001.

[CGH+12] Pierre-André Cornillon, Arnaud Guyader, François Husson *et al.* : *Statistiques avec R*. Presses Universitaires de Rennes, 3e édition, 2012.

[CHSD13] Louise Corron, Jean-Bernard Huchet, Frédéric Santos et Olivier Dutour : Paléopathologie, pseudo-pathologie et taphonomie : Etude pluridisciplinaire et classification de modifications ostéo-archéologiques de restes humains, 2013. Poster présenté au Groupe des Paléopathologistes de Langue Française (GPLF), 12–13 avril, Toulon, France.

[Haw08] Lamis Hawarah : *Une approche probabiliste pour le classement d'objets incomplètement connus dans un arbre de décision*. Thèse de doctorat, Université de Grenoble I, 2008.

[KTK08] Imran Kurt, Mevlut Ture et A. Turhan Kurum : Comparing performances of logistic regression, classification and regression tree, and neural networks for predicting coronary artery disease. *Expert Systems with Applications*, 34(1):366–374, 2008.

[Lac13] Alizé Lacoste Jeanson : Minéralisation des dents permanentes comme indicateur de la maturité sexuelle chez les filles : étude préliminaire, 2013. Mémoire de Master 2, Université de Bordeaux.

[LW02] Andy Liaw et Matthew Wiener : Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[MSR+11] Joao Maroco, Dina Silva, Ana Rodrigues *et al.* : Data mining methods in the prediction of dementia : A real-data comparison of the accuracy, sensitivity and specificity of linear discriminant analysis, logistic regression, neural networks, support vector machines, classification trees and random forests. *BMC Research Notes*, 4(1):299, 2011.

[PB07] Éric Parent et Jacques Bernier : *Le raisonnement bayésien*. Springer-Verlag France, 2007.

[TAR13] Terry Therneau, Beth Atkinson et Brian Ripley : *rpart : Recursive Partitioning*, 2013. R package version 4.1-3.