Chapitre 4

Arbres

Les commandes utilisées dans ce chapitre font appel aux packages suivants

```
> library(rpart)
> library(rpart.plot)
> library(vip)
```

Les méthodes par arbres permettent de faire de la prévision dans le cas de la régression (Y continue) et de la classification (Y qualitative). Les différences entre ces deux contextes sont minimes, nous présenterons donc la méthodologie dans un cadre général incluant ces deux situations.

Un arbre va chercher une partition de l'espace des X de manière à ce que les valeurs de Y à l'intérieur d'un même élément de la partition soient les plus proches possibles. Il s'agit donc de trouver des zones homogènes vis-à-vis de Y. Il existe différents procédés pour obtenir ces partitions, et donc plusieurs types d'arbres (CHAID, C4.5...). Nous nous focaliserons dans ce chapitre sur la méthode CART (*Classification And Regression Trees*) présentée dans Breiman *et al.* (1984) qui reste la méthode de référence. Nous illustrerons les différents étapes de la construction d'un arbre sur les données de classification binaire suivantes :

```
> don.2D.arbre <- gen_class_bin2D(n=150,graine=3210,bayes=0.05)$donnees
> ggplot(don.2D.arbre)+aes(x=X1,y=X2,color=Y)+geom_point()
```

On visualise sur la figure 4.1 que le carré $[0,1]^2$ contient des zones dominées par le groupe 0 (en bas à droite par exemple) et d'autres par le groupe 1 (en haut à gauche). Un arbre va tenter d'identifier automatiquement ces zones, il va donc s'agir de chercher une partition qui soit optimale vis-à-vis d'un critère de performance. Il n'est bien entendu pas possible d'envisager toutes les partitions du carré $[0,1]^2$, d'une part la complexité algorithmique serait trop importante et d'autre part on risquerait de sur-ajuster les données. Les arbres CART vont se

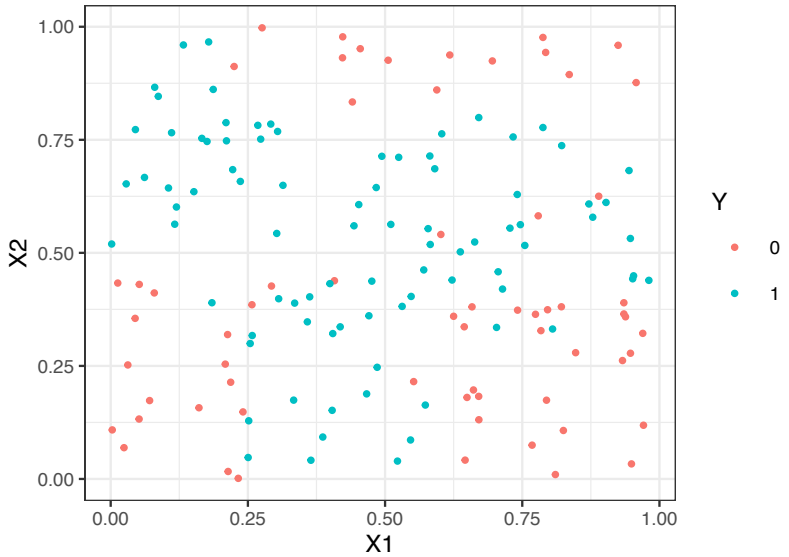


FIGURE 4.1 – Échantillon de taille 150 avec 2 variables explicatives et une variable binaire à expliquer.

restreindre à des partitions en rectangles construits à partir de *divisions binaires successives* de l'espace.

Plus précisément l'algorithme va tout d'abord chercher à séparer les données en 2 parties à partir d'une droite orthogonale aux axes du carré, il va donc falloir trouver une variable de coupure X_j parmi $\{X_1, X_2\}$ ainsi qu'un seuil de coupure s. Le carré en sera ainsi divisé en deux rectangles $\{X_j \geq s\}$ et $\{X_j < s\}$. La figure en haut à gauche de la figure 4.2 propose de couper selon la variable X_2 et le seuil 0.39. Les 2 rectangles obtenus sont ensuite redécoupés (étapes 2 et 3) selon la variable X_1 et le seuil 0.58 (étape 2) et X_2 avec le seuil 0.79 (étape 3). On répète encore ce procédé pour arriver à l'étape 5 à une partition en 6 rectangles. Un algorithme de prévision pourra se déduire de cette dernière partition en considérant le vote majoritaire des observations dans chaque rectangle.

La partition peut se visualiser simplement ici car nous avons 2 variables explicatives. On utilise plus souvent un visuel en forme d'arbre (d'où le nom de la méthode) pour représenter l'algorithme de prévision. La figure 4.3, obtenue avec la fonction `rpart.plot` qui sera présentée plus tard, propose ce visuel pour les 6 rectangles obtenus. Les coupures sont ici présentées de façons successives, on sépare tout d'abord selon la condition $X_2 < 0.39$, si la réponse est oui on descend par la gauche et on considère la nouvelle condition $X_1 \geq 0.58$, sinon on va à droite avec la condition $X_2 \geq 0.79$ et ainsi de suite. On obtient au final 6 ensembles, qui seront appelés *nœuds terminaux* ou *feuilles*, sur la dernière ligne de l'arbre qui correspondent aux 6 rectangles de la dernière étape de la figure 4.2. Le premier ensemble correspond par exemple au rectangle en bas à gauche

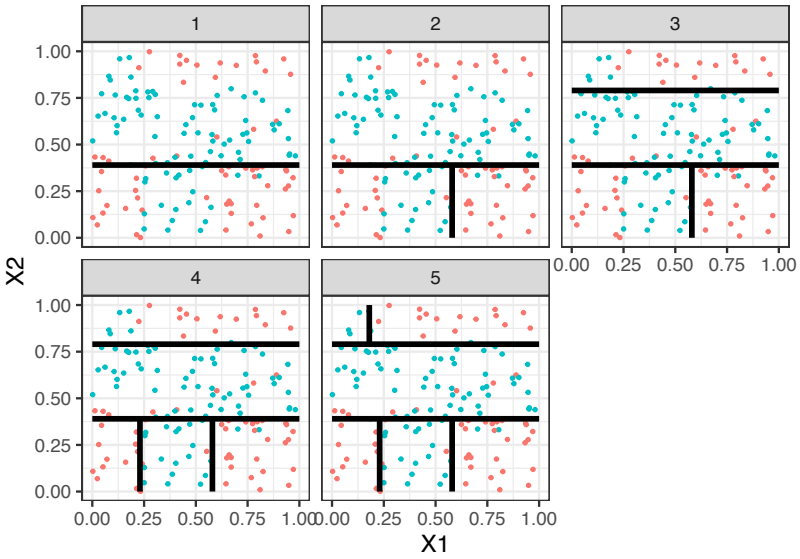


FIGURE 4.2 – Exemple de coupures successives.

qui ne contient que des observations du groupe 0. C'est pourquoi on prédira 0 dans cet ensemble.

Cette partie introductive a juste présenté les grandes idées associées aux méthodes par arbres, de nombreuses questions se posent. La première concerne le choix des coupures, comment les données vont choisir les bonnes variables et les bons seuils ? Se pose ensuite la question du choix du meilleur arbre. Doit-il comporter un grand nombre de coupures ? Ne risque t-il pas de sur-ajuster si c'est le cas ? Nous répondons à ces questions dans les parties suivantes.

4.1 Vocabulaire sur les arbres

Un arbre va se construire en répétant des divisions successives de l'espace des X en deux parties. À chaque étape, ces divisions vont créer des sous-espaces appelés *nœuds*. Le premier nœud, appelé *nœud racine* et noté $\mathcal{N}$, contient toutes les observations $(x_i, y_i), i = 1, \ldots, n$. Ce nœud va être découpé en choisissant une variable de coupure X_j et un seuil $s \in \mathbb{R}$ en deux nœuds fils $\mathcal{N}_1(j, s)$ et $\mathcal{N}_2(j, s)$ comme indiqué sur la figure 4.4.

Les deux nœuds fils $\mathcal{N}_1(j, s)$ et $\mathcal{N}_2(j, s)$ seront ensuite découpés en deux autres nœuds fils et ainsi de suite. Une fois les coupures terminées, on obtient une partition de l'espace des X dont les éléments sont appelés *nœuds terminaux* ou *feuilles*. Ces nœuds correspondent aux nœuds qui ne sont plus découpés, les autres nœuds sont appelés *noeuds internes*. Pour l'arbre de la figure 4.3 on a par exemple 6 nœuds terminaux et 5 nœuds internes. Les prévisions issues de

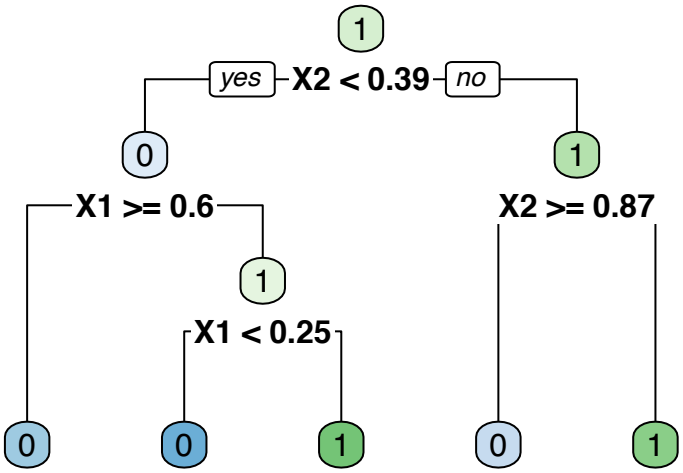


FIGURE 4.3 – Arbre correspondant à la figure 4.2.

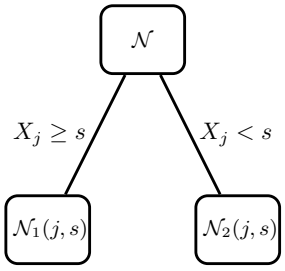


FIGURE 4.4 – $\mathcal{N}$ et ses 2 noeuds fils.

l'arbre se feront à partir des nœuds terminaux : on identifiera d'abord le nœud terminal où un nouvel individu x tombe et on prédira

— la moyenne des y_i tels que les x_i sont dans le même nœud que x :

$$f_n(x) = \frac{1}{|\mathcal{N}(x)|} \sum_{i:x_i \in \mathcal{N}(x)} y_i$$

en régression ($\mathcal{N}(x)$ désigne le noeud terminal de l'arbre qui contient x).
— le groupe majoritaire des y_i tels que les x_i sont dans le même nœud que x :

$$f_n(x) = \underset{j=1,\dots,K}{\operatorname{argmax}} \sum_{i:x_i \in \mathcal{N}(x)} \mathbf{1}_{y_i=j}$$

en classification. On pourra également estimer les probabilités $\mathbf{P}(Y = j|X = x)$, et donc obtenir des fonctions de score en classification binaire, en considérant la proportion d'individus du groupe j dans le nœud de x

$$\frac{1}{|\mathcal{N}(x)|} \sum_{i:x_i \in \mathcal{N}(x)} \mathbf{1}_{y_i=j}.$$

Obtenir des prédicteurs par arbre n'est donc pas difficile. Pour que les prévisions soient de bonne qualité, il va falloir définir des stratégies efficaces pour bien découper les nœuds et choisir le nombres de coupures.

4.2 Notion d'impureté pour découper les nœuds

La première question qui se pose est donc relative à la manière de découper un nœud. Une fois qu'on saura comment procéder, on appliquera la même procédure pour découper chaque nœud. Choisir une coupure revient à sélectionner une variable X_j et un seuil s. Il faut donc trouver un couple $(j, s) \in \{1, \dots, d\} \times \mathbb{R}$. Ce choix va se faire en définissant un critère de performance pour tous les couples (j, s), on choisira ensuite le couple qui optimise ce critère.

Rappelons que l'objectif initial est de trouver une partition de l'espace des X en zones homogènes vis-à-vis de Y. Une bonne coupure va donc devoir séparer les données du nœud $\mathcal{N}$ de manière à ce que les observations dans ses deux nœuds fils soient les plus homogènes possibles, c'est-à-dire que les valeurs de Y doivent se ressembler au maximum dans $\mathcal{N}_1(j, s)$ et $\mathcal{N}_2(j, s)$. Cela nous amène à définir une mesure ou fonction d'*impureté* $\mathcal{I}$ pour un nœud telle que $\mathcal{I}(\mathcal{N})$ prend

— des petites valeurs lorsque $\mathcal{N}$ est homogène (et donc pur), c'est-à-dire lorsque les y_i dans $\mathcal{N}$ sont proches ;
— des valeurs élevées lorsque $\mathcal{N}$ est hétérogène (et donc impur), ce qui revient à dire que les y_i sont fortement dispersés dans $\mathcal{N}$.

La définition de la fonction d'impureté va dépendre de la nature de la variable à expliquer. On donnera des exemples dans les sections suivantes pour la régression (Y continue) et la classification (Y qualitative). Une fois l'impureté définie, on cherchera la coupure (j, s) qui minimise l'impureté des nœuds fils ou encore qui maximise le gain d'impureté entre $\mathcal{N}$ et ses deux nœuds fils. Ce gain d'impureté est défini par

$$\Delta(j,s) = p(\mathcal{N})\mathcal{I}(\mathcal{N}) - (p(\mathcal{N}_1(j,s))\mathcal{I}(\mathcal{N}_1(j,s)) + p(\mathcal{N}_2(j,s))\mathcal{I}(\mathcal{N}_2(j,s))), \quad (4.1)$$

où $p(\mathcal{N})$ désigne la proportion d'observations dans le nœud $\mathcal{N}$. Les impuretés sont ainsi pondérées par les tailles de nœuds, cela permet d'éviter d'obtenir des nœuds fils avec des tailles trop différentes. Quelle que soit la nature de Y, la meilleure coupure sera choisie en maximisant (4.1), la principale distinction entre un *arbre de régression* (Y continue) et un *arbre de classification* (Y qualitative) résidera dans la manière de définir l'impureté.

Remarque 4.1. Nous avons considéré ici uniquement des variables X continues. En présence de variables qualitatives, on ne pourra pas choisir un seuil pour couper la variable en 2. On considèrera toutes les partitions binaires de l'ensemble des modalités, et on choisira celle qui maximise le gain d'impureté (voir exercice 4.1).

4.2.1 Impureté en régression

Lorsque les y_i sont dans $\mathbb{R}$, on mesure l'impureté d'un nœud $\mathcal{N}$ par la variance des y_i

$$\mathcal{I}(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{i:x_i \in \mathcal{N}} (y_i - \bar{y}_{\mathcal{N}})^2,$$

où $\bar{y}_{\mathcal{N}}$ désigne la moyenne des y_i telle que $x_i \in \mathcal{N}$. Cette quantité prend effectivement des valeurs faibles lorsque les y_i sont proches et élevées dans le cas contraire.

La figure 4.5 présente deux exemples de coupure d'un même nœud.

Il est facile de voir que la gain d'impureté (4.1) est de 0.125 pour la coupure de gauche contre 2.25 pour celle de droite qui sera donc privilégiée.

4.2.2 Impureté en classification

Plaçons nous maintenant dans un contexte de classification où les y_i sont à valeurs dans $\{1, \ldots, K\}$. Un nœud est pur lorsqu'un groupe particulier apparaît de façon majoritaire dans le nœud. L'impureté d'un nœud va donc se baser sur les proportions d'individus de chaque groupe, elle est définie par

$$\mathcal{I}(\mathcal{N}) = \sum_{j=1}^{K} f(p_j(\mathcal{N}))$$

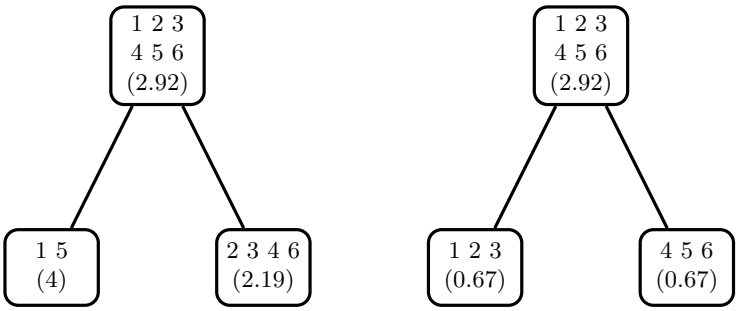


FIGURE 4.5 – 2 exemples de coupure d'un même nœud, l'impureté est précisée entre parenthèses.

où $p_j(\mathcal{N})$ désigne l'ensemble des observations du groupe j dans $\mathcal{N}$ et f est une fonction concave telle que $f(0) = f(1) = 0$. L'impureté se calcule en sommant les impuretés associées à chaque groupe, la contribution du groupe j étant mesurée par $f(p_j(\mathcal{N}))$. Un nœud pur va correspondre au cas où une de ces proportions est proche de 1 et toutes les autres sont proches de 0, c'est pourquoi on demande à f de prendre des valeurs nulles en 0 et 1. La concavité imposera des valeurs plus élevées les proportions s'éloignent de 0 et 1. Deux fonctions f sont généralement utilisées :

— l'*impureté de Gini* définie par $f(p) = p(1-p)$. Elle est utilisée par défaut dans la fonction `rpart`.
— l'*impureté dite d'information* définie par $f(p) = -p\log(p))$. On pourra l'utiliser avec l'option `parms=list(split="information")` dans `rpart`.

Dans la cas binaire, on a $\mathcal{I}(\mathcal{N}) = 2p_1(\mathcal{N})(1-p_1(\mathcal{N}))$ pour Gini et $\mathcal{I}(\mathcal{N}) = -p_1(\mathcal{N})\log(1-p_1(\mathcal{N})) - (1-p_1(\mathcal{N}))\log(p_1(\mathcal{N}))$ pour l'impureté d'information. Ces deux fonctions sont représentées sur la figure 4.6.

Ces deux fonctions ont la même allure, l'utilisateur pourra les comparer mais les résultats diffèrent généralement peu entre ces deux impuretés.

On représente sur la figure 4.7 deux exemples de coupure d'un même nœud pour un problème de classification binaire.

Les gains d'impureté de Gini sont ici de 0.056 pour la coupure de gauche contre 0.5 pour celle de droite qui conduit à deux nœuds purs (et qui sera donc meilleure).

4.3 Choisir un arbre

La section précédente nous a présenté la manière de découper un nœud en deux. On peut donc maintenant construire un grand nombre d'arbres à partir d'un même jeu de données. En effet, le nœud racine $\mathcal{N}_0$ est un premier arbre qui prédira toujours la même valeur. Découper ce nœud va définir un nouvel arbre à

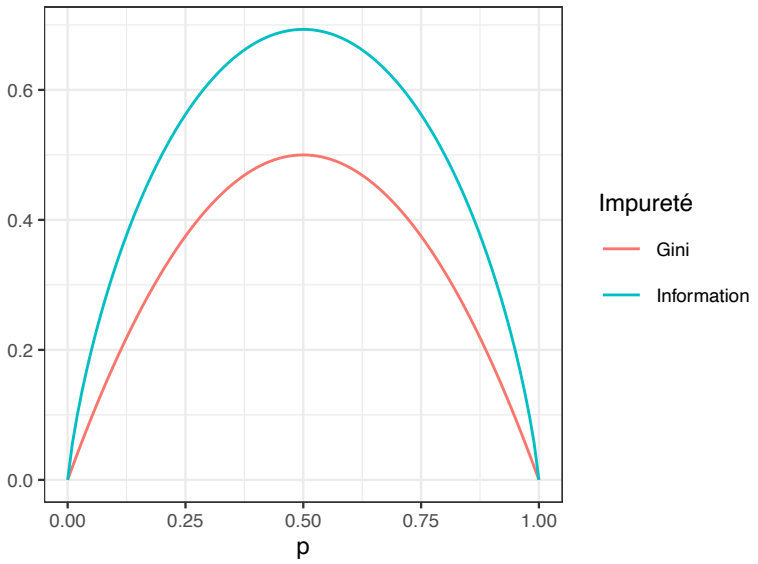


FIGURE 4.6 – Impuretés de Gini et d'information.

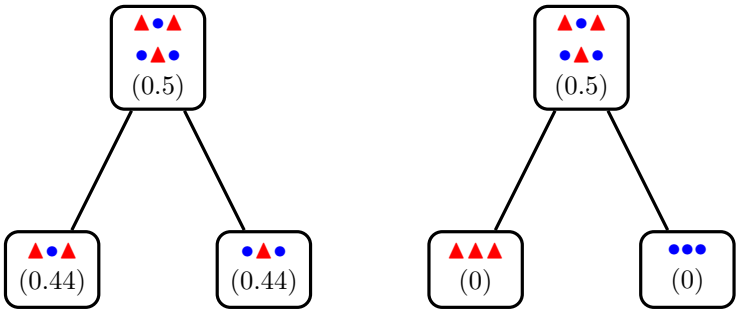


FIGURE 4.7 – 2 exemples de coupure d'un même nœud, l'impureté de Gini est précisée entre parenthèses).

deux nœuds terminaux $\mathcal{N}_1$ et $\mathcal{N}_2$. La découpe de ces deux nœuds peut définir trois nouveaux arbres : un où on découpe juste $\mathcal{N}_1$, un autre ou on découpe juste $\mathcal{N}_2$ et un dernier où on découpe $\mathcal{N}_1$ et $\mathcal{N}_2$, et ainsi de suite...

Se pose alors la question de quand stopper les coupures ? Une première approche peut consister à construire l'*arbre maximal*. Cet arbre est tel que tous ses nœuds terminaux sont purs. Il va donc s'obtenir en multipliant les découpes jusqu'à obtenir des valeurs identiques de Y dans tous ses nœuds terminaux, quitte à avoir une seule observation à l'intérieur. Cet arbre maximal, noté $T_{\max}$ à partir de maintenant, sera parfait en terme d'ajustement mais il éprouvera des difficultés à bien prédire de nouveaux individus, il sur-ajustera. La complexité des arbres est donc mesurée par le nombre de coupures. Il va donc falloir choisir habilement ce nombre de coupures, ou la profondeur de l'arbre, pour bien prédire.

Plusieurs approches peuvent être envisagées. On pourrait d'abord fixer un critère d'arrêt sur les coupures, du genre "si le gain d'impureté ne dépasse pas une valeur seuil, stop". En procédant ainsi on peut se priver de coupures importantes qui si situeraient en bas de l'arbre. Il se peut en effet qu'une coupure apporte peu de gain, mais que des coupures qui se trouvent en dessous de celle-ci soient beaucoup plus pertinentes. C'est pourquoi on n'utilise pas cette technique en général.

Une autre approche, plus pertinente, serait de considérer l'ensemble de tous les arbres binaires possibles. Cet ensemble est constitué par tous les *sous-arbres* de $T_{\max}$. On évaluerait ensuite la performance de chaque sous-arbre, en estimant un critère de prévision par validation croisée par exemple, et on choisirait celui qui prédit le mieux. Bien que cohérente, cette approche est généralement difficile à mettre en œuvre, le nombre de sous-arbres étant souvent trop important. En effet, un sous-arbre de $T_{\max}$ ne s'obtient pas uniquement en enlevant une seule coupure. Un sous-arbre T_1 s'obtiendra en retirant certaines branches à $T_{\max}$, on notera $T_1 \subset T_{\max}$. La figure 4.8 présente deux sous-arbres d'une même arbre.

L'arbre en haut à droite a été obtenu en supprimant la division de $\mathcal{N}_1$ en $\mathcal{N}_3$ et $\mathcal{N}_4$, celui en bas à droite en supprimant cette même coupure ainsi que celle divisant $\mathcal{N}_5$ en $\mathcal{N}_7$ et $\mathcal{N}_8$. Un sous-arbre ne se restreint pas à la suppression de coupures en bas de l'arbre, on peut aussi enlever des branches. Ainsi toujours sur le même exemple, l'arbre à une coupure (avec donc $\mathcal{N}_1$ et $\mathcal{N}_2$ comme nœuds terminaux) est un sous-arbre tout comme l'arbre sans coupure (constitué uniquement du nœud racine). Il est difficile de calculer explicitement le nombre de sous-arbres d'un arbre donné mais on comprend bien que, pour peu que l'arbre maximal ait un nombre de coupures important, il ne sera pas possible d'évaluer la performance de tous les sous-arbres.

La stratégie usuelle pour choisir un sous arbre va fonctionner en deux étapes :

1. Sélectionner une suite de taille raisonnable de sous-arbres de l'arbre maximal ;
2. Choisir un arbre dans cette suite en évaluant un risque de prévision.

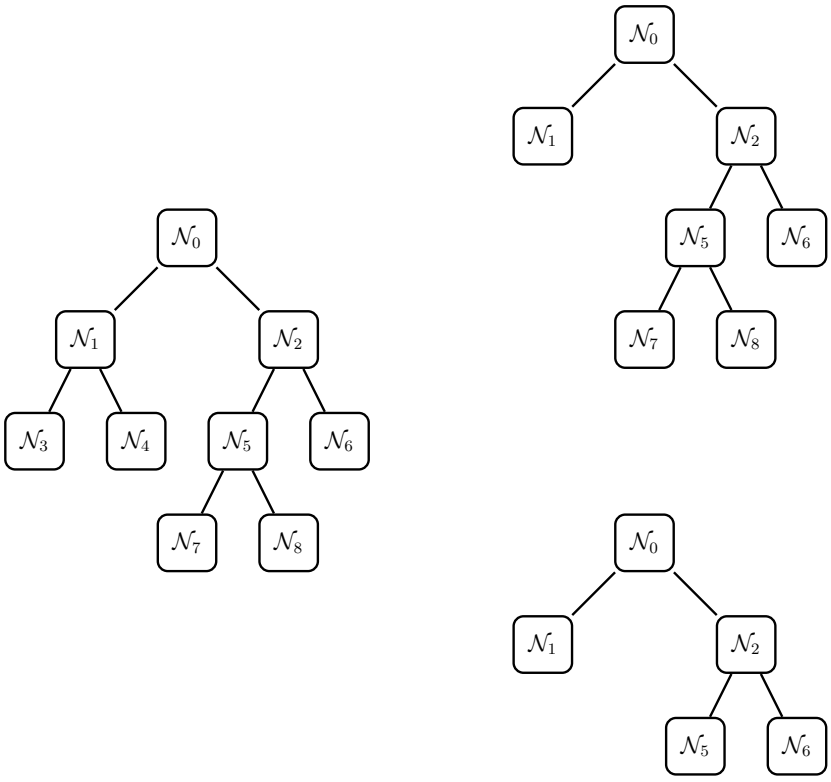


FIGURE 4.8 – 2 sous-arbres (droite) d'un même arbre (gauche).

4.3.1 Élagage

La première étape va s'effectuer en sélectionnant des sous-arbres d'un même arbre (qu'il faut imaginer profond) qui optimisent un critère prenant en compte à la fois l'ajustement et la complexité de l'arbre. On considère T un arbre possédant $|T|$ nœuds terminaux $\mathcal{N}_1, \ldots, \mathcal{N}_{|T|}$ et on désigne par $R_m(T)$ une mesure de l'erreur du nœud terminal $\mathcal{N}_m$, par exemple l'erreur quadratique

$$R_m(T) = \frac{1}{N_m} \sum_{i:x_i \in \mathcal{N}_m} (y_i - \bar{y}_{\mathcal{N}_m})^2$$

en régression ou l'erreur de classification

$$R_m(T) = \frac{1}{N_m} \sum_{i:x_i \in \mathcal{N}_m} \mathbf{1}_{y_i \neq y_{\mathcal{N}_m}}$$

en classification. Les quantités N_m, $\bar{y}_{\mathcal{N}_m}$ et $y_{\mathcal{N}_m}$ représentent respectivement le nombre d'observations, la moyenne des y_i et le groupe majoritaire dans $\mathcal{N}_m$. En sommant toutes ces erreurs

$$\sum_{m=1}^{|T|} N_m R_m(T),$$

on obtient une mesure globale de l'ajustement de T. Minimiser cette dernière quantité reviendra à toujours choisir l'arbre maximal, et donc à sur-ajuster. C'est pourquoi nous allons pénaliser cette erreur par la complexité de l'arbre, c'est-à-dire le nombre de nœuds terminaux. On obtient alors le critère *coût/complexité* défini par

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m R_m(T) + \alpha |T|. \tag{4.2}$$

Ce critère propose un compromis entre qualité d'ajustement et complexité de l'arbre. La balance entre ces deux quantités est contrôlée par le paramètre $\alpha \geq 0$. En effet, lorsque α est proche de 0, l'ajustement va dominer et $C_\alpha(T)$ va favoriser des arbres profonds. À l'inverse des valeurs de α élevées donneront plus de poids à la complexité et privilégieront des arbres avec peu de coupures. Il faudra donc choisir α convenablement mais une question se pose avant, celle de la minimisation de $C_\alpha(T)$ pour une valeur de $\alpha \geq 0$ fixée.

On peut déjà régler les cas extrêmes :

— pour $\alpha = 0$, la solution est l'arbre maximal $T_{\max}$;
— lorsque α tend vers $+\infty$, l'arbre sans coupure (celui constitué uniquement du nœud racine) minimisera $C_\alpha(T)$.

Entre ces deux valeurs, le problème n'a rien de simple à première vue. Il s'agit en effet de minimiser une fonction, dont le paramètre est un arbre, sur l'ensemble de tous les sous arbres de $T_{\max}$. Les deux lemmes suivants, dont on trouvera la preuve dans Breiman *et al.* (1984), vont nous apporter les solutions.

Lemme 4.1. *Si T_1 et T_2 sont deux sous-arbres de T_{max} avec $R_\alpha(T1) = R_\alpha(T2)$. Alors $T_1 \subset T_2$ ou $T_2 \subset T_1$.*

Ce lemme garantit une unique solution de taille minimale. Pour $\alpha \geq 0$ on peut donc noter T_α l'unique sous-arbre de $T_{\max}$ de taille minimale qui minimise $C_\alpha(T)$.

Lemme 4.2. *Si $\alpha > \alpha'$ alors $T_\alpha = T_{\alpha'}$ ou $T_\alpha \subset T_{\alpha'}$.*

Ce lemme garantit une stabilité des solutions lorsque α parcourt $\mathbb{R}^+$. Elles vont être emboîtées les unes dans les autres. On peut résumer ces deux lemmes avec le théorème suivant.

Theorem 4.1. *Il existe une suite finie $\alpha_0 = 0 < \alpha_1 < \cdots < \alpha_M$ avec $M \leq |T_{max}|$ et une suite associée d'arbres emboîtés $(T_{\alpha_m})_m$*

$$T_{max} = T_{\alpha_0} \supset T_{\alpha_1} \supset \cdots \supset T_{\alpha_M} = T_{root}$$

telle que $\forall \alpha \in [\alpha_m, \alpha_{m+1}[$

$$T_m \in \underset{T \subseteq T_{max}}{\operatorname{argmin}} C_\alpha(T).$$

Ce théorème est remarquable. Il assure que le nombre de sous-arbres de $T_{\max}$ qui minimise $C_\alpha(T)$ est majoré par $|T_{\max}|$ et donc par le nombre d'observations n (on ne peut pas découper des nœuds plus de n fois). Dans la plupart des cas, il sera beaucoup plus petit. Il nous apprend de plus que c'est le même sous-arbre qui reste solution entre deux valeurs successives de la suite $(\alpha_m)_m$. On peut le représenter à l'aide du schéma de la figure 4.9.

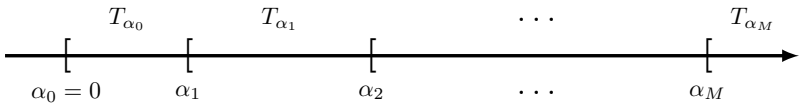


FIGURE 4.9 – Minimisation du critère coût/complexité.

Lorsque $\alpha = 0$ c'est $T_{\max}$ qui va minimiser $C_\alpha(T)$ et ce dernier va rester solution jusqu'à α_1 où un nouvel arbre T_{α_1} devient solution jusqu'à α_2 et ainsi de suite jusqu'à α_M où la solution devient l'arbre racine. Par ailleurs la suite $(T_{\alpha_m})_m$ des solutions est une suite d'arbres emboîtés : T_{α_1} s'obtient en enlevant une branche à $T_{\max}$, T_{α_2} en supprime une à T_{α_1}. Une question importante concerne le calcul des suites $(\alpha_m)_m$ et $(T_{\alpha_m})_m$. Nous ne le présentons pas ici mais il existe un algorithme qui permet d'obtenir ces suites, il est présenté dans l'exercice 4.3. L'idée est de chercher les branches les moins performantes de l'arbre et de les enlever les unes après les autres. Tous les arbres obtenus correspondent donc à l'arbre maximal auquel on a enlevé des branches, c'est pourquoi on parle d'*élagage* (*pruning* en anglais).

Visualisons cette suite d'arbres emboîtés en reprenant les données 2D utilisées en introduction du chapitre. On commence par construire l'arbre de la figure 4.3 avec la fonction `rpart` du package `rpart`

```
> set.seed(321)
> arbre0 <- rpart(Y~.,data=don.2D.arbre)
> rpart.plot(arbre0)
```

Les suites $(\alpha_m)_m$ et $(T_m)_m$ s'obtiennent sur `R` avec la fonction `printcp` :

```
> printcp(arbre0)
##
## Classification tree:
## rpart(formula = Y ~ ., data = don.2D.arbre)
##
## Variables actually used in tree construction:
## [1] X1 X2
##
## Root node error: 64/150 = 0.42667
##
## n= 150
##
##         CP nsplit rel error  xerror     xstd
## 1 0.31250      0   1.00000 1.00000 0.094648
## 2 0.17188      1   0.68750 0.87500 0.092562
## 3 0.13281      2   0.51562 0.65625 0.085923
## 4 0.01000      4   0.25000 0.35938 0.068951
```

On a une suite de 5 arbres emboîtés. La colonne `CP` renvoie les valeurs α_m (également appelée `paramètres de complexité`), `nsplit` donne le nombre de coupures des arbres T_{α_m}, on retrouve bien que la profondeur des arbres augmente lorsque les α_m diminuent. La colonne `rel error` renvoie l'erreur d'ajustement des T_{α_m}, il s'agit ici de l'erreur de classification calculée sur les données d'apprentissage. Sans surprise cette erreur décroît avec la profondeur des arbres : "plus on coupe, mieux on ajuste". On peut remarquer que l'erreur de l'arbre racine vaut exactement 1. Ce sera toujours le cas puisque `rpart` normalise toutes les erreurs par rapport à celle de l'arbre racine. Il faudra donc ici multiplier par 65/150 si on veut les erreurs non normalisées. Les deux dernières colonnes seront expliquées plus tard.

La suite (T_{α_m}) ne descend pas jusqu'à l'arbre maximal. Il est en effet rarement utile d'aller jusqu'à cet arbre pour obtenir le meilleur arbre. On peut néanmoins obtenir une suite plus grande en modifiant les paramètres

- `cp` qui correspond au α_m minimal jusqu'où on veut aller ;
- `minsplit` qui représente la taille minimale d'un nœud pour être découpé (en dessous de `minsplit` on ne découpe plus les nœuds).

de la fonction `rpart`. Par exemple

```
> set.seed(321)
> arbre.max <- rpart(Y~.,data=don.2D.arbre,cp=0,minsplit=2)
> printcp(arbre.max)
##
## Classification tree:
## rpart(formula = Y ~ ., data = don.2D.arbre, cp = 0, minsplit = 2)
##
## Variables actually used in tree construction:
## [1] X1 X2
##
## Root node error: 64/150 = 0.42667
##
## n= 150
##
##          CP nsplit rel error  xerror     xstd
## 1 0.3125000      0   1.00000 1.00000 0.094648
## 2 0.1718750      1   0.68750 0.87500 0.092562
## 3 0.1328125      2   0.51562 0.65625 0.085923
## 4 0.0312500      4   0.25000 0.35938 0.068951
## 5 0.0234375      5   0.21875 0.31250 0.065052
## 6 0.0156250      7   0.17188 0.32812 0.066402
## 7 0.0078125     12   0.09375 0.32812 0.066402
## 8 0.0000000     24   0.00000 0.40625 0.072439
```

On est bien allé jusqu'à l'arbre maximal ici, il contient 14 coupures et tous ses nœuds sont purs puisque son erreur d'ajustement est nulle.

4.3.2 Validation croisée

La phase d'élagage permet de se ramener à une suite d'arbres de cardinal raisonnable. Il reste à sélectionner un arbre dans cette suite, ce qui revient à choisir une valeur dans la suite $(\alpha_m)_m$. Rappelons que, pour une valeur α_m et une nouvelle observation x, un arbre renvoie une prévision $T_{\alpha_m}(x, \mathcal{D}_n)$ qui correspond à la moyenne des y_i (ou le label majoritaire parmi les y_i) qui se trouvent dans le même nœud terminal que x. On pourra ainsi estimer des risques de prévisions pour chaque T_{α_m} à partir des méthodes présentées dans le chapitre 3 et choisir l'arbre qui a le plus petit risque.

Une validation croisée 10 blocs est faite dans `rpart` pour estimer le risque de chaque arbre. Elle est décrite dans l'algorithme 4.1.

Le calcul des β_m à la première étape peut sembler étrange. Il est néanmoins logique dans la mesure où il n'y a aucune raison pour que l'élagage des arbres maximaux construits sur les blocs renvoient les mêmes suites de paramètres de complexité. On propose donc de fixer une valeur β entre les éléments de la suite $(\alpha_m)_m$ comme indiqué sur la figure 4.10. Les valeurs de β choisies correspondent aux moyennes géométriques entre deux valeurs successives des α_m. Ainsi chaque β_m correspond à un unique α_m et donc à un unique sous-arbre T_{α_m}. Le reste

Algorithme 4.1 Validation croisée pour choisir un arbre.

Entrées :

— une suite $(\alpha_m)_m$ telle que $0 < \alpha_1 < \alpha_2 < \cdots < \alpha_M$.

— $\{B_1, \ldots, B_K\}$ une partition de $\{1, \ldots, n\}$ en K blocs.

— Une fonction de perte ℓ.

1. Calculer

$$\beta_0 = 0, \quad \beta_1 = \sqrt{\alpha_1 \alpha_2}, \quad \ldots \quad \beta_{M-1} = \sqrt{\alpha_{M-1}\alpha_M}, \quad \beta_M = +\infty.$$

2. Pour $k = 1, \ldots, K$
   (a) Construire l'arbre maximal sur l'ensemble des données privé du k^{e} bloc, c'est-à-dire $\mathcal{B}^{-k} = \{(x_i, y_i) : i \in \{1, \ldots, n\} \backslash B_k\}$.
   (b) Appliquer l'algorithme d'élagage du théorème 4.1 à cet arbre maximal, puis extraire les arbres qui correspondent aux valeurs $\beta_m, m = 0, \ldots, M \Longrightarrow T_{\beta_m}(., \mathcal{B}^{-k})$.
   (c) Calculer les valeurs prédites par chaque arbre sur le bloc k : $T_{\beta_m}(x_i, \mathcal{B}^{-k}), i \in B_k$.
3. En déduire les erreurs pour chaque β_m :

$$\widehat{\mathcal{R}}(\beta_m) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in B_k} \ell(y_i, T_{\beta_m}(x_i, \mathcal{B}^{-k})).$$

Retourner : une valeur α_m telle que $\widehat{\mathcal{R}}(\beta_m)$ est minimum.

est standard, on entraîne les algorithmes sur les données privées d'un bloc, on prédit ce bloc et on en déduit une valeur du risque pour chaque valeur de β_m.

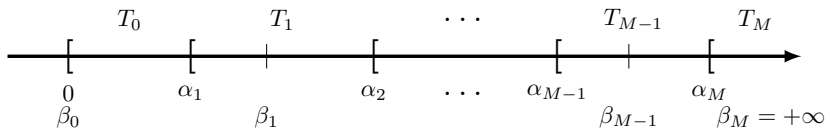


FIGURE 4.10 – Lien entre les α_m et les β_m.

Par défaut `rpart` utilise la perte indicatrice en classification, on a donc

$$\widehat{\mathcal{R}}(\beta_m) = \frac{1}{n} \sum_{k=1}^{K} \sum_{i \in B_k} \mathbf{1}_{T_{\beta_m}(x_i, \mathcal{B}^{-k}) \neq y_i}$$

et la perte quadratique en régression :

$$\widehat{\mathcal{R}}(\beta_m) = \frac{1}{n}\sum_{k=1}^{K}\sum_{i\in B_k}(y_i - T_{\beta_m}(x_i, \mathcal{B}^{-k}))^2.$$

Les résultats de la validation croisée correspondent à la colonne `xerror` de la sortie de `printcp`. Ces erreurs sont toujours normalisées par l'erreur de l'arbre racine. Si on reprend la dernière sortie avec la suite de 7 arbres emboîtés, on lit donc les erreurs de classification par validation croisée dans la colonne `xerror` et on choisira le sous-arbre pour lequel cette erreur est minimale, l'arbre à 5 coupures sur l'exemple. La colonne `xstd` correspond à une estimation de l'écart-type de la colonne `xerror`, nous avons en effet vu dans le chapitre ? ? ? qu'on pouvait déduire des estimateurs de la variance des risques. Connaitre l'écart-type permet de donner une indication sur la précision de l'estimateur de l'erreur de classification. Cela pourra aider l'utilisateur dans son choix d'algorithme. On peut par exemple visualiser les erreurs en fonction de la complexité avec la fonction `plotcp` (figure 4.11).

```
> plotcp(arbre.max)
```

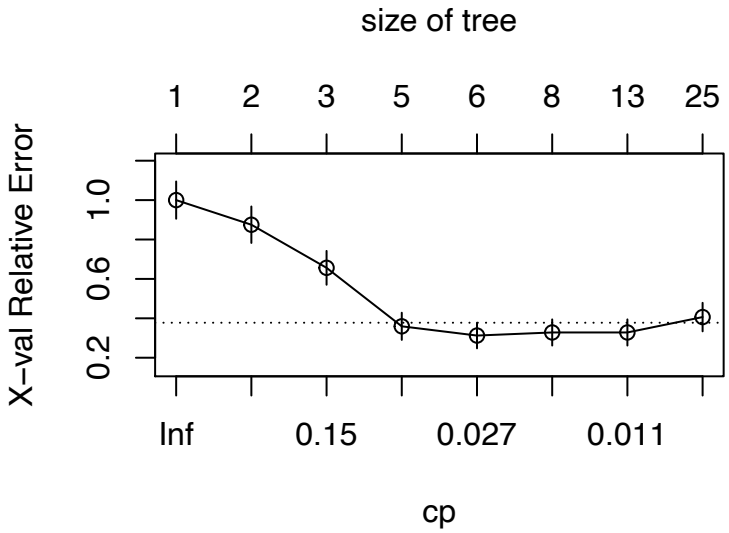


FIGURE 4.11 – Choix de la complexité de l'arbre avec 'plotcp'.

Les erreurs sont représentées en fonction du paramètre de complexité (en bas) ou du nombre de nœuds terminaux (en haut). Des petits traits verticaux, correspondant à plus ou moins un écart-type de l'erreur, sont ajoutés sur chaque point. La ligne en pointillés horizontale correspond à l'erreur minimale plus son écart-type. Elle indique à l'utilisateur que toutes les erreurs en dessous de cette ligne peuvent être considérées comme être du même ordre que l'erreur optimale. Elle n'est pas très utile ici car l'exemple est très simple mais elle peut se révéler

utile dans des cas où l'arbre optimal possède beaucoup de coupures et qu'un arbre, plus parcimonieux, aura une erreur du même ordre. L'utilisateur pourra choisir cet arbre plus simple.

L'extraction de l'arbre optimal se réalise avec la fonction `prune`. On récupère tout d'abord le paramètre de complexité optimal de façon automatique :

```
> cp.opt <- arbre.max$cptable %>% as_tibble() %>%
+   filter(xerror==min(xerror)) %>% dplyr::select(CP) %>%
+   slice(1) %>% as.numeric()
> cp.opt
## [1] 0.0234375
```

On l'extrait et on le visualise (figure 4.12)

```
> arbre.opt <- prune(arbre.max,cp=cp.opt)
> rpart.plot(arbre.opt)
```

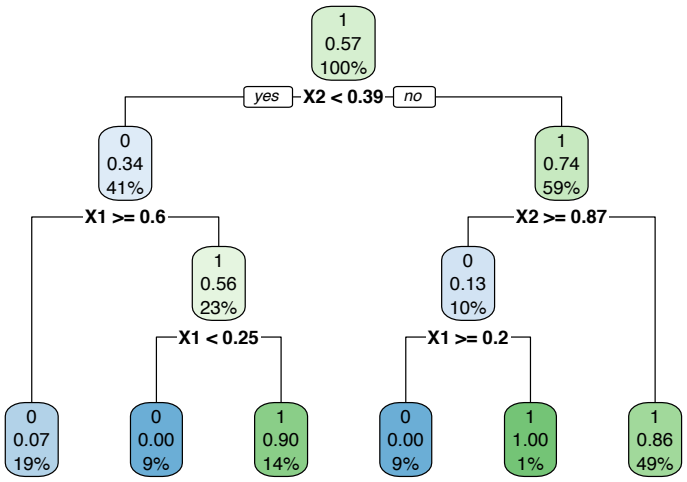


FIGURE 4.12 – Représentation de l'arbre optimal

Une fois l'arbre construit, on pourra l'utiliser pour prédire de nouveaux individus. Considérons par exemple les nouvelles données

```
> new.X
## # A tibble: 4 x 2
##      X1    X2
##   <dbl> <dbl>
## 1 0.473 0.561
## 2 0.770 0.933
## 3 0.216 0.515
## 4 0.414 0.408
```

La fonction `predict.rpart`

```
> predict(arbre.opt,newdata=new.X)
##           0         1
## 1 0.1369863 0.8630137
## 2 1.0000000 0.0000000
## 3 0.1369863 0.8630137
## 4 0.1369863 0.8630137
```

renvoie les estimations des probabilités $\mathbf{P}(Y = j|X = x), j = 0, 1$. On obtiendra les groupes prédits en maximisant ces probabilités

```
> predict(arbre.opt,newdata=new.X,type="class")
## 1 2 3 4
## 1 0 1 1
## Levels: 0 1
```

Les X évoluant dans le carré $[0, 1]^2$ on propose de visualiser l'algorithme de prévision. Il suffit de construire une grille sur le carré

```
> px1 <- seq(0,1,by=0.01)
> px2 <- seq(0,1,by=0.01)
> px <- expand.grid(X1=px1,X2=px2)
```

Puis de calculer les prévisions de chaque point de la grille

```
> prev <- predict(arbre.opt,newdata=px,type="class")
```

Et enfin de colorier les points en fonction de la valeur prédite (figure 4.13)

```
> tbl <- px %>% as_tibble() %>% mutate(Y=as.factor(prev),
+                                      Y1=as.numeric(prev)-1)
> ggplot(tbl)+aes(x=X1,y=X2,fill=Y)+geom_tile(alpha=0.7)
```

L'algorithme obtenu est proche de la règle de Bayes représentée sur la figure 1.2.

4.4 Compléments

4.4.1 Mesure d'importance des variables

La visualisation de l'arbre est clairement importante pour interpréter l'algorithme. Elle permet d'identifier les variables qui interviennent pour découper chaque nœud et aide à interpréter l'algorithme. Elle ne permet néanmoins pas de hiérarchiser les variables en fonction de l'importance qu'elles ont. En effet les variables en haut de l'arbre ne sont pas forcément plus importantes que les autres. Une coupure particulière peut avoir plus d'importance en terme de séparation des Y. Il existe ainsi une mesure, appelée *score d'importance*, qui va noter chaque variable en fonction de son utilité dans la construction de l'arbre.

Rappelons que chaque coupure est sélectionnée en maximisant le gain d'impureté entre un nœud et ses deux nœuds fils. Plus le gain d'impureté est important, meilleure est la coupure. Le score d'importance est défini en sommant les gains

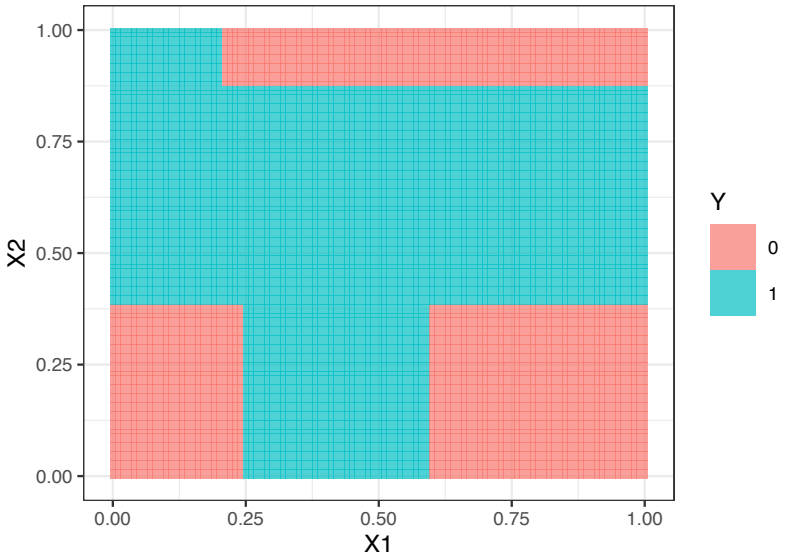


FIGURE 4.13 – Les prévisions de l'arbre optimal.

d'impureté engendrées par chaque variable. Un arbre T possède $|T|$ nœuds terminaux et donc $|T|-1$ nœuds internes (qui sont découpés). On note Δ_t le gain d'impureté (4.1) associé à la coupure du t^e nœud interne et $X_{c(t)}$ la variable utilisée pour découper ce nœud. L'importance de la variable $X_j, j = 1, \ldots, d$ est mesurée par

$$\mathcal{I}_j(T) = \sum_{t=1}^{|T|-1} \Delta_t \mathbf{1}_{c(t)=j}. \tag{4.3}$$

Ces scores sont automatiquement calculés dans la fonction `rpart`, on peut récupérer les importances des variables de l'arbre optimal représenté sur la figure 4.12 avec

```
> arbre.opt$variable.importance
##       X2       X1
## 28.73174 24.28286
```

On pourra les visualiser avec un digramme en barres en utilisant la fonction `vip` du package `vip` (voir section 4.5). Les importances sont relativement proches sur cet exemple, c'est cohérent avec la manière dont les données ont été générées puisqu'aucune des deux variables n'a été privilégiée.

Remarque 4.2. Les importances calculées par `rpart` font aussi intervenir les variables `surrogate`. Ces variables peuvent être vues comme des variables qui se substituent à la variable de coupure lorsque celle-ci n'est pas observée. Nous les présentons dans la section suivante.

4.4.2 Coupures secondaires et gestion des données manquantes

Dans la phase de construction de l'arbre, chaque nœud interne est divisé en deux en choisissant la meilleure coupure parmi toutes. Toutes les coupures sont donc testées. `rpart` conserve quelques coupures en mémoire. Par exemple

```
> summary(arbre.opt)
```

renvoie des informations sur les coupures de chaque nœud. On peut par exemple lire pour le second nœud

```
Node number 2: 63 observations, ...
    class counts:    43    20
   probabilities: 0.683 0.317
  left son=4 (32 obs) right son=5 (31 obs)
  Primary splits:
      X1 < 0.5839991  to the right, improve=10.654410, (0 missing)
      X2 < 0.04222861 to the right, improve= 3.478946, (0 missing)
  Surrogate splits:
      X2 < 0.2990984  to the left,  agree=0.603, adj=0.194, (0 split)
```

On retrouve bien la coupure selon $X_1 < 0.584$ mais une deuxième coupure est proposée ($X_2 < 0.042$). Cette deuxième coupure est la seconde meilleure coupure, elle n'est pas utilisée pour construire l'arbre mais il peut parfois être intéressant d'aller voir les variables qui auraient pu être utilisées pour découper des nœuds. Une autre coupure ($X_2 < 0.299$) est également proposée dans la partie `Surrogate splits`. Cette coupure peut-être vue comme une coupure de substitution à la coupure principale dans le cas où la variable de coupure X_1 n'est pas observée. Cette coupure supplémentaire est calculée en cherchant la coupure (qui n'utilise pas X_1) la plus semblable à la coupure principale. Cela revient à expliquer la coupure, c'est-à-dire une variable binaire $\tilde{Y} = \mathbf{1}_{X_1<0.584}$, par les variables différentes de la variable de coupure. Un arbre à une coupure expliquant $\tilde{Y}$ par les autres variables que X_1 renverra la coupure souhaitée qui sera appelée *coupure surrogate*. Les `surrogate` peuvent se révéler utiles en présence de données manquantes. Considérons en effet un nouvel individu pour lequel on n'a pas observé X_1 :

```
> newX
## # A tibble: 1 x 2
##      X1    X2
##   <dbl> <dbl>
## 1    NA   0.2
```

On souhaite prédire le groupe de ce nouvel individu avec `arbre.opt`. On procède de façon classique en faisant descendre `newX` dans l'arbre de la figure 4.12. La coupure du nœud racine ($X_2 < 0.39$) va envoyer `newX` à gauche dans le nœud 2 qui est découpé selon la règle $X_1 \geq 0.58$. On ne peut pas utiliser cette coupure puisque X_1 n'est pas observé. On va donc utiliser la coupure `surrogate` $X_2 < 0.299$ qui envoie "à gauche" si la condition est vérifiée. C'est le cas ici donc

on fait descendre `newX` dans le nœud en bas à gauche de l'arbre et on s'arrête là car c'est un nœud terminal. On prédira donc pour `newX` le groupe 0 avec une probabilité $\widehat{\mathbf{P}}(Y = 1|X = x) = 0.03$. On peut retrouver cette probabilité avec `predict` :

```
> predict(arbre.opt,newdata=newX)
##           0         1
## 1 0.0952381 0.9047619
```

Les coupures `surrogate` peuvent être également utilisées en présence de données manquantes dans l'échantillon d'apprentissage. L'approche est identique. Si une variable de coupure n'est pas observée pour un individu, on lui applique la coupure `surrogate` pour faire descendre l'individu dans l'arbre. Ce procédé évite de perdre des individus en route !

4.5 Un exemple sur données réelles

On considère les données `SAheart` présentées dans la section 1.2.3. Il s'agit d'expliquer la présence de maladies cardiaques (variable binaire `chd`) par 9 variables. On commence par construire un arbre avec les paramètres par défaut de `rpart`

```
> set.seed(1234)
> T0 <- rpart(chd~.,data=SAheart)
```

et on étudie la suite d'arbres emboîtés

```
> printcp(T0)
##
## Classification tree:
## rpart(formula = chd ~ ., data = SAheart)
##
## Variables actually used in tree construction:
## [1] adiposity age       famhist   ldl       tobacco
## [6] typea
##
## Root node error: 160/462 = 0.34632
##
## n= 462
##
##         CP nsplit rel error  xerror     xstd
## 1 0.12500      0    1.0000 1.00000 0.063918
## 2 0.10000      1    0.8750 0.95625 0.063224
## 3 0.06250      2    0.7750 0.85625 0.061357
## 4 0.02500      3    0.7125 0.83125 0.060825
## 5 0.01875      5    0.6625 0.81250 0.060409
## 6 0.01250      7    0.6250 0.95000 0.063119
## 7 0.01000      8    0.6125 1.00625 0.064011
```

L'erreur d'ajustement (`rel.error`) décroît (ce qui est normal) tandis que l'erreur

de prévision (`xerror`) diminue jusqu'au 5^e arbre pour augmenter par la suite. Cela laisse penser que `T0` est suffisamment profond pour être élaguer. On peut, par sécurité, considérer un arbre plus profond avec

```
> set.seed(432)
> T0 <- rpart(chd~.,data=SAheart,minsplit=10,cp=1e-5)
> plotcp(T0)
```

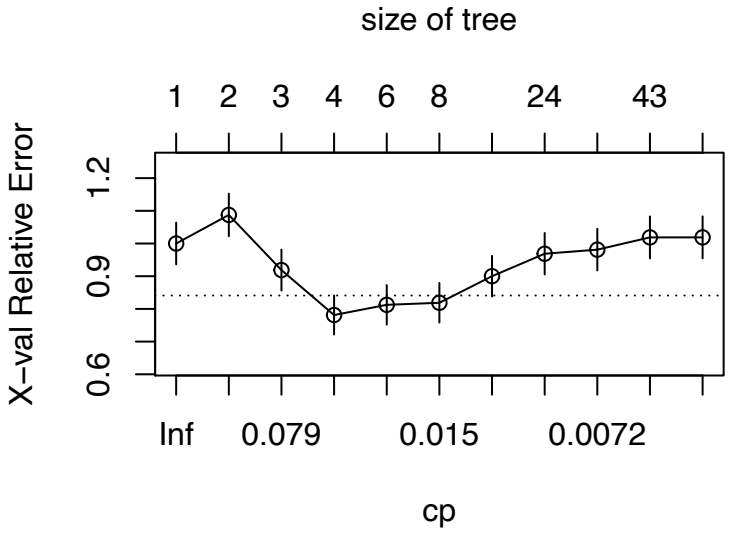


FIGURE 4.14 – Erreur de prévision en fonction de la complexité.

La figure 4.14 montre que l'arbre optimal possède 6 nœuds terminaux. On l'extrait de la suite et le visualise avec (voir figure 4.15)

```
> cp.opt <- T0$cptable %>% as_tibble() %>%
+   filter(xerror==min(xerror)) %>% select(CP) %>%
+   slice(1) %>% as.numeric()
> T_opt <- prune(T0,cp=cp.opt)
> rpart.plot(T_opt)
```

Le nœud racine est découpé par l'âge. On peut interpréter que les personnes âgées ont tendance à être plus touchées par la pathologie puisque 2 nœuds terminaux à droite, regroupant 24% des patients contre 13% pour l'autre nœud terminal, prédisent `chd=1`. Les antécédents familiaux `famhist` semblent aussi avoir une influence. En effet, pour les personnes de plus de 51 ans ayant des antécédents familiaux, la probabilité d'être atteint est de 0.70. On peut accompagner cette interprétation de l'arbre de la visualisation de l'importance des variables (figure 4.16) :

```
> vip(T_opt)
```

C'est encore la variable `age` qui ressort en premier. La deuxième variable à

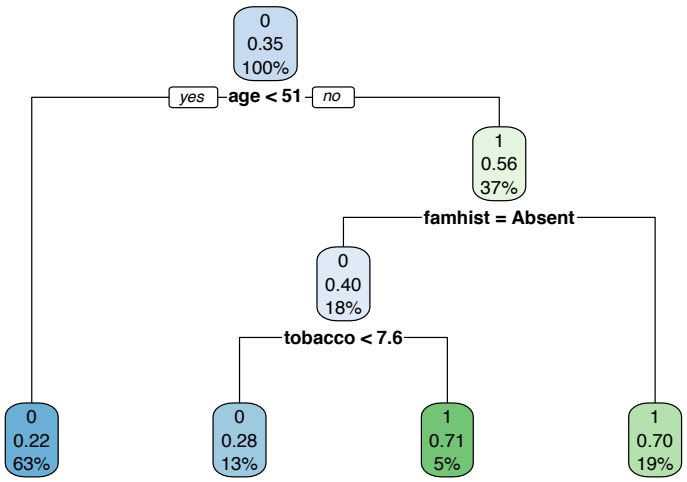


FIGURE 4.15 – L'arbre optimal.

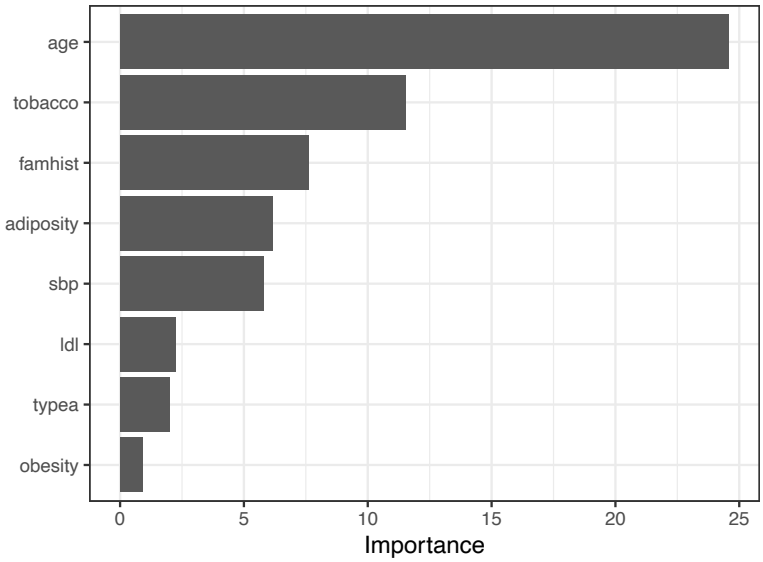


FIGURE 4.16 – Importance des variables.

se distinguer est `tobacco`. Cette variable apparaissait pourtant plus bas dans l'arbre que `famhist`. On peut l'expliquer en regardant de plus près la coupure `tobacco<7.6`. Elle apporte une information claire puisqu'elle séparee un nœud possédant 40% de malades en deux nœuds qui en possèdent 28% et 71%. Le gain d'impureté associée à cette coupure est donc conséquent, ce qui explique sa mesure d'importance. Remarquons enfin que des variables non utilisées pour découper ont une importance non nulles. Cela s'explique par le fait que l'importance est également calculée en utilisant les coupures `surrogate`.

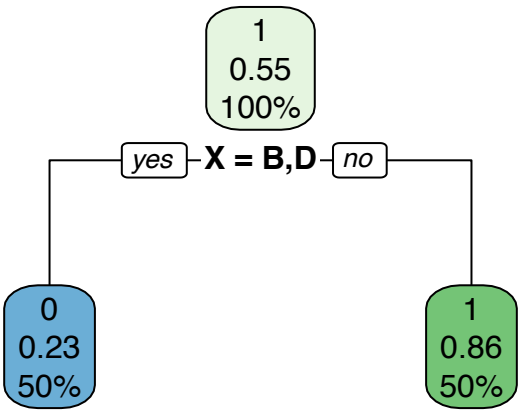
4.6 Exercices

Exercice 4.1 (Couper une variable qualitative).
On considère les données

```
> n <- 100
> X <- factor(rep(c("A","B","C","D"),n))
> Y <- rep(0,n)
> set.seed(1234)
> Y[X=="A"] <- rbinom(sum(X=="A"),1,0.9)
> Y[X=="B"] <- rbinom(sum(X=="B"),1,0.25)
> Y[X=="C"] <- rbinom(sum(X=="C"),1,0.8)
> Y[X=="D"] <- rbinom(sum(X=="D"),1,0.2)
> Y <- as.factor(Y)
> tbl <- tibble(X,Y)
```

1. Construire un arbre permettant d'expliquer Y par X.

```
> arbre <- rpart(Y~.,data=tbl)
> rpart.plot(arbre)
```



2. Expliquer la manière dont la coupure a été sélectionnée.
   La variable étant qualitative, on ne cherche pas un seuil de coupure pour diviser un nœud en 2. On va ici considérer toutes les partitions binaires de l'ensemble $\{A, B, C, D\}$. La meilleure partition est $\{\{A, C\}, \{B, D\}\}$.

Exercice 4.2 (Arbres de régression).
On considère un échantillon de taille 250 simulé selon le modèle `sinus` présenté dans la section 1.2.1 :

```
> tbl.sinus <- gen_sinus(n=250,sig2=0.04,graine=1234)
```

On rappelle qu'on cherche à estimer la fonction `sinus` entre -2π et 2π à partir de l'échantillon `tbl.sinus`.

1. Construire un arbre T_0 très profond et vérifier qu'il sur-ajuste.

   On prend des valeurs de `minsplit` et cp très petites afin d'obtenir un arbre très profond.

   ```
   > set.seed(123)
   > T0.sinus <- rpart(Y~.,data=tbl.sinus,minsplit=2,cp=1e-5)
   ```

   On regarde les dernières lignes de la table

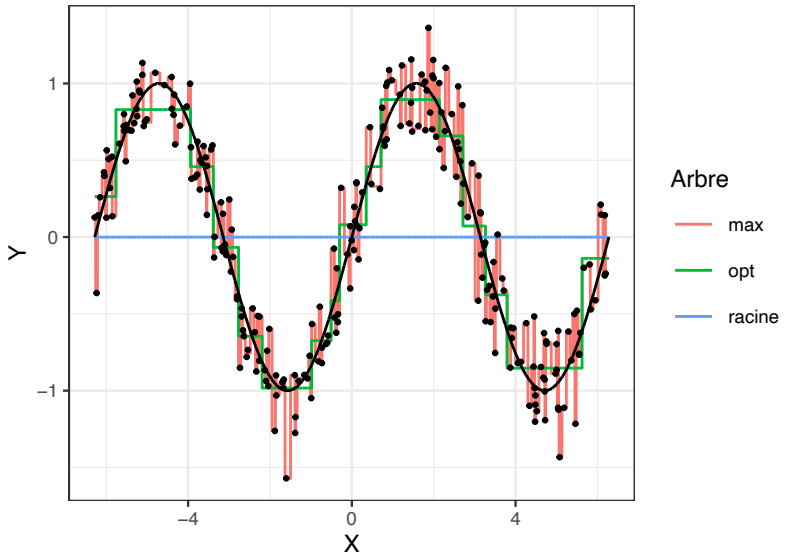
   ```
   > T0.sinus$cptable %>% as_tibble() %>% arrange(CP) %>% head()
   ## # A tibble: 6 x 5
   ##          CP nsplit `rel error` xerror   xstd
   ##       <dbl>  <dbl>       <dbl>  <dbl>  <dbl>
   ## 1 0.00001      215   0.0000927  0.167 0.0143
   ## 2 0.0000102    214   0.000103   0.167 0.0143
   ## 3 0.0000109    213   0.000114   0.167 0.0143
   ## 4 0.0000113    212   0.000125   0.167 0.0143
   ## 5 0.0000117    210   0.000148   0.167 0.0143
   ## 6 0.0000123    209   0.000161   0.167 0.0143
   ```

   L'erreur d'ajustement est proche de 0, on est bien dans la zone de sur-ajustement.

2. Visualiser sur un même graphe l'arbre racine, l'arbre maximal et l'arbre optimal (celui qui minimise l'erreur de prévision).

   On extrait ces 3 arbres de la suite et on calcule les prévisions sur une grille entre -2π et 2π.

   ```
   > cp.opt <- T0.sinus$cptable %>% as_tibble() %>%
   +   filter(xerror==min(xerror)) %>% select(CP) %>%
   +   slice(1) %>% as.numeric()
   > T.opt <- prune(T0.sinus,cp=cp.opt)
   > T.root <- prune(T0.sinus,cp=1)
   > newX <- tibble(X=seq(-2*pi,2*pi,by=0.001),y=sin(X))
   > prev <- tibble(X=newX$X,
   +            opt=predict(T.opt,newdata=newX),
   +            max=predict(T0.sinus,newdata=newX),
   +            racine=predict(T.root,newdata=newX)) %>%
   +   pivot_longer(-X,names_to="Arbre",values_to="Y")
   > ggplot(prev)+geom_line(aes(x=X,y=Y,color=Arbre))+
   +   geom_point(data=tbl.sinus,aes(x=X,y=Y),size=0.5)+
   +   geom_line(data=newX,aes(x=X,y=y),size=0.5)
   ```



Exercice 4.3 (Minimisation du critère coût/complexité).
On considère l'algorithme qui permet de calculer les suites $(\alpha_m)_m$ et $(T_{\alpha_m})_m$ du théorème 4.1. Pour simplifier on se place en classification binaire et on considère les notations suivantes (en plus de celles présentées dans le chapitre) :

- $R(t)$: erreur de classification dans le nœud t pondérée par la proportion d'individus dans le nœud (nombre d'individus dans t sue le nombre total d'individus).
- T^t : la branche de l'arbre T issue du nœud interne t.
- $R(T^t)$: l'erreur de la branche T^t pondérée par la proportion d'individus dans le nœud.

L'algorithme 4.2 présente le calcul explicite des suites $(\alpha_m)_m$ et $(T_{\alpha_m})_m$.

On propose d'utiliser cet algorithme sur l'arbre construit suivant

```
> don.2D.arbre <- gen_class_bin2D(n=150,graine=3210,bayes=0.05)$donnees
> set.seed(123)
> T0 <- rpart(Y~.,data=don.2D.arbre)
> rpart.plot(T0,extra = 1)
```

(voir Figure 4.17). Cet arbre n'est pas l'arbre maximal mais la manière d'élaguer est identique.

1. Calculer pour les 5 nœuds internes de T_0 la fonction $g(t)$.
   On numérote les nœuds internes de haut en bas et de gauche à droite. On commence par le nœud t_5, celui qui correspond à la coupure `X1>=0.31`. On a

$$R(t_5) = \frac{4}{23}\,\frac{23}{150} = \frac{4}{150} \quad \text{et} \quad R(T_0^{t_5}) = \frac{3}{23}\,\frac{23}{150} = \frac{3}{150}.$$

Algorithme 4.2 Minimisation du critère coût/complexité.

Initialisation : on pose $\alpha_0 = 0$ et on calcule l'arbre maximale T_0 qui minimise $C_0(T)$. On fixe $m = 0$. Répéter jusqu'à obtenir l'arbre racine

1. Calculer pour tous les nœuds t internes de T_{α_m}
$$g(t) = \frac{R(t) - R(T^t_{\alpha_m})}{|T^t_{\alpha_m}| - 1}$$
2. Choisir le nœud interne t_m qui minimise $g(t)$.
3. On pose
$$\alpha_{m+1} = g(t_m) \quad \text{et} \quad T_{\alpha_{m+1}} = T_{\alpha_m} - T^{t_m}_{\alpha_m}.$$
4. Mise à jour : $m < -m + 1$.

Retourner : les suites finies $(\alpha_m)_m$ et $(T_{\alpha_m})_m$.

On déduit
$$g(t_5) = \frac{4/150 - 3/150}{2 - 1} = \frac{1}{150}.$$
On fait de même pour les 4 autres nœuds internes et on obtient les résultats suivants :

t	t_1	t_2	t_3	t_4	t_5
$R(t)$	65/150	20/150	22/150	12/150	4/150
$R(T^t_0)$	8/150	2/150	6/150	1/150	3/150
$\|T^t_0\|$	6	3	3	2	2
$g(t)$	11.4/150	9/150	8/150	11/150	1/150

2. En déduire la valeur de α_1 ainsi que l'arbre T_{α_1}.
$g(t)$ est minimum en t_5 On a donc $\alpha_1 = g(t_5)$ et $T_{\alpha_1} = T_0 - T_0^{t_5}$, c'est-à-dire T_0 auquel on enlève la coupure `X1>=0.31`.
3. Retrouver cette valeur en utilisant la fonction `printcp` et représenter l'arbre T_1 en utilisant `prune`.
On se rappelle que `printcp` normalise toutes les erreurs par rapport à celle de l'arbre racine, par conséquent la valeur de α_1 affichée par `printcp` sera
$$\frac{1}{150}\frac{150}{65} = \frac{1}{65} \approx 0.01538.$$
En effet :

```
> printcp(T0)
##
## Classification tree:
## rpart(formula = Y ~ ., data = don.2D.arbre)
##
## Variables actually used in tree construction:
## [1] X1 X2
##
## Root node error: 64/150 = 0.42667
##
## n= 150
##
##         CP nsplit rel error  xerror     xstd
## 1 0.31250      0   1.00000 1.00000 0.094648
## 2 0.17188      1   0.68750 0.89062 0.092887
## 3 0.13281      2   0.51562 0.65625 0.085923
## 4 0.01000      4   0.25000 0.35938 0.068951
```

Et on peut visualiser l'arbre avec (figure 4.18)

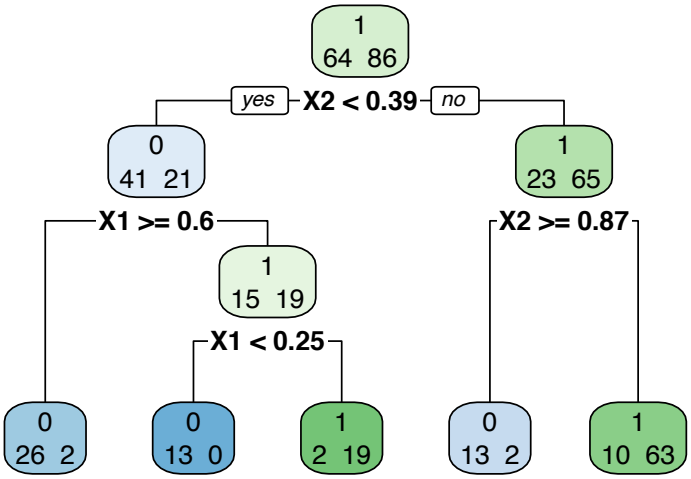


FIGURE 4.17 – L'arbre T_0.

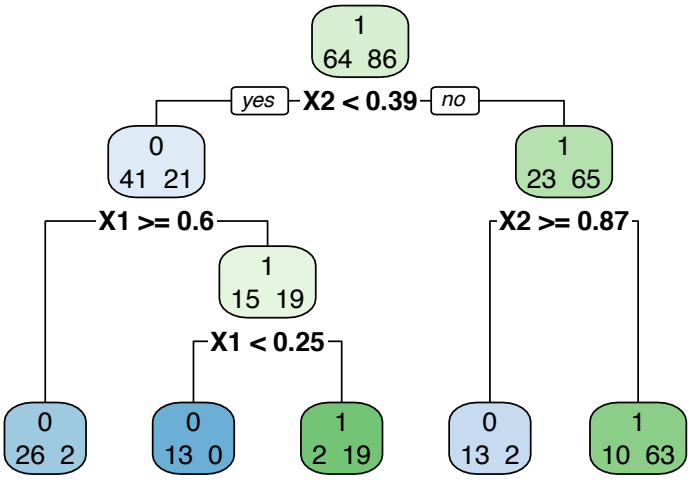


FIGURE 4.18 – L'arbre T_{α_1}.

t	t_1	t_2	t_3	t_4
$R(t)$	65/150	20/150	22/150	12/150
$R(T^t_{\alpha_1})$	9/150	18/150	7/150	1/150
$\lvert T^t_{\alpha_1}\rvert$	5	3	2	2
$g(t)$	14/150	9/150	15/150	11/150

On supprimera ici t_2 avec on posera $\alpha_2 = 9/65 \approx 0.13846$ (on normalise). On peut tracer T_{α_2} (figure 4.19).

```
> T2 <- prune(T0,cp=0.138462)
> rpart.plot(T2,extra = 1)
```

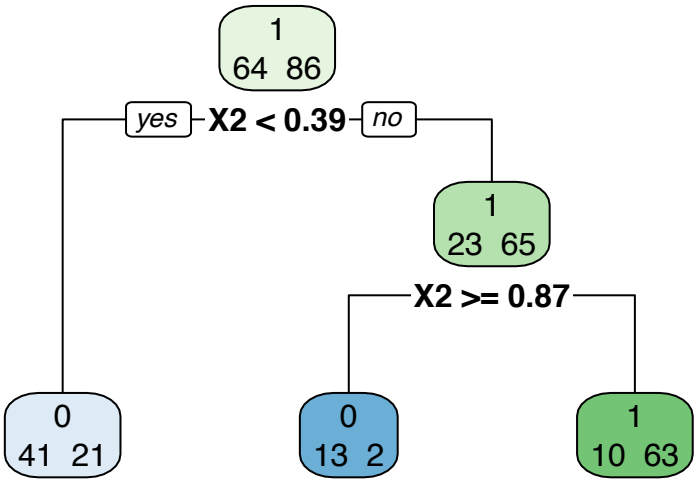


FIGURE 4.19 – L'arbre T_{α_2}.

Exercice 4.4 (Courbe ROC pour un arbre).
On reprend les données `SAheart` utilisées dans 4.5. On souhaite ici évaluer la performance de la procédure qui permet de sélectionner l'arbre à l'aide de la courbe ROC. Pour ce faire, on note $S_n(x, \mathcal{D}_n, \theta)$ l'estimateur de $\mathbf{P}(Y = 1|X = x)$ obtenu à partir de l'arbre sélectionné. Le paramètre θ contient toutes les étapes de construction de l'arbre, notamment l'élagage et la sélection par validation croisée.

1. Tracer la courbe ROC de l'algorithme de prévision $S_n(x, \mathcal{D}_n, \theta)$ en effectuant une validation hold out. On pourra ajouter, à titre de comparaison, les courbes ROC des arbres racine et maximaux.
   On commence par séparer les données en apprentissage/test.

```
> set.seed(432)
> ind.app <- sample(nrow(SAheart),round(2/3*nrow(SAheart)))
> dapp <- SAheart[ind.app,]
> dtest <- SAheart[-ind.app,]
```

On calcule les scores prédits par chaque arbre sur les données test.

```
> T0 <- rpart(chd~.,data=dapp,minsplit=5,cp=1e-4)
> cp.opt <- T0$cptable %>% as_tibble() %>%
+   filter(xerror==min(xerror)) %>% select(CP) %>%
+   slice(1) %>% as.numeric()
> T.opt <- prune(T0,cp.opt)
> T.root <- prune(T0,cp=1)
```

On peut maintenant représenter les courbes ROC

```
> score <- tibble(opt=predict(T.opt,newdata=dtest,type="prob")[,2],
+             racine=predict(T.root,newdata=dtest,type="prob")[,2],
+             max=predict(T0,newdata=dtest,type="prob")[,2],
+             obs=as.numeric(dtest$chd)-1)
> score %>%  pivot_longer(-obs,names_to="Arbre",
+                     values_to="score") %>%
+   ggplot()+aes(d=obs,m=score,color=Arbre)+plotROC::geom_roc()
```

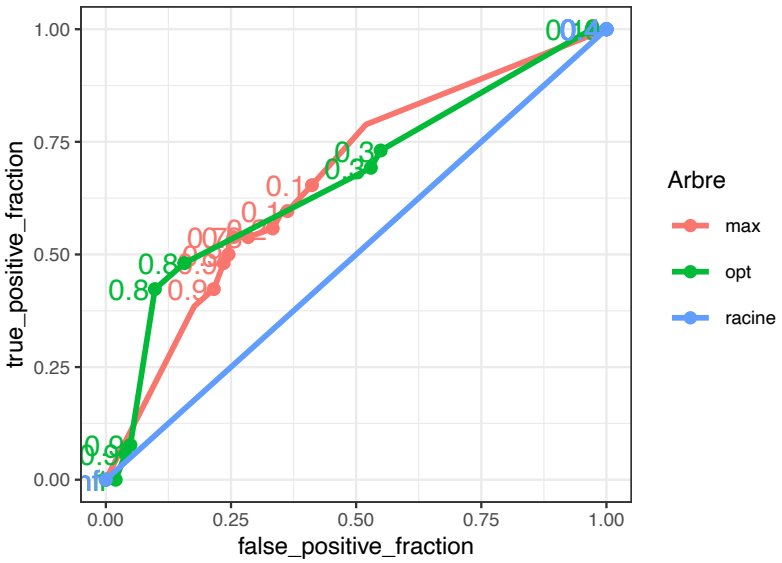


FIGURE 4.20 – Comparaison des courbes ROC par hold out.

et déduire les AUC

```
> score %>% summarize_at(1:3,~pROC::auc(obs,.)) %>%  round(3)
## # A tibble: 1 x 3
##     opt racine   max
##   <dbl>  <dbl> <dbl>
## 1 0.667    0.5 0.667
```

2. Faire le même travail avec une validation croisée 10 blocs.
   On procède de façon similaire mais en faisant une boucle sur chaque bloc.

```
> set.seed(1234)
> blocs <- caret::createFolds(1:nrow(SAheart),10,returnTrain = TRUE)
> score.cv <- matrix(0,ncol=3,nrow=nrow(SAheart))
> for (j in 1:length(blocs)){
+   dapp <- SAheart[blocs[[j]],]
+   dtest <- SAheart[-blocs[[j]],]
+   T0 <- T0 <- rpart(chd~.,data=dapp,minsplit=5,cp=1e-4)
+   cp.opt <- T0$cptable %>% as_tibble() %>%
+ filter(xerror==min(xerror)) %>% select(CP) %>%
+ slice(1) %>% as.numeric()
+   T.opt <- prune(T0,cp.opt)
+   T.root <- prune(T0,cp=1)
+
+   score.cv[-blocs[[j]],1] <- predict(T.opt,newdata=dtest,
+                                      type="prob")[,2]
+   score.cv[-blocs[[j]],2] <- predict(T.root,newdata=dtest,
+                                      type="prob")[,2]
+   score.cv[-blocs[[j]],3] <- predict(T0,newdata=dtest,
+                                      type="prob")[,2]
+ }
> score.cv <- as_tibble(score.cv)
> score.cv$obs <- as.numeric(SAheart$chd)-1
> names(score.cv)[1:3] <- c("opt","racine","max")
>
> score.cv %>%  pivot_longer(-obs,names_to="Arbre",
+                            values_to="score") %>%
+   ggplot()+aes(d=obs,m=score,color=Arbre)+plotROC::geom_roc()
```

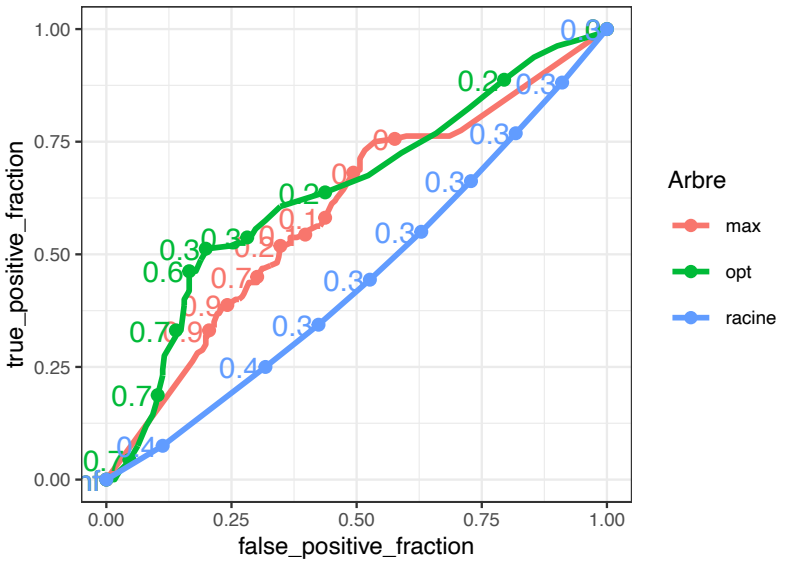


FIGURE 4.21 – Comparaison des courbes ROC par validation croisée.