Université de Bordeaux
L3 MIASHS
2022-2023

Balade en forêt : CART et RF

Auteurs
Mathieu Bechade
Ayman Kachmar
Thomas Solana

Tuteur
Vincent Couallier

08/05/2023

Abstract

CART (Classification And Regression Trees) was created to address several needs and challenges in the field of machine learning and data analysis. In this dissertation, we will introduce the CART method by explaining the objectives of this method and how it works. We will examine the nature of binary decision trees by discussing both the construction and pruning of a tree. We will also explore the different types of trees, such as regression and classification trees, highlighting their distinctions. For regression trees, we will make a comparison with classical regression models such as multiple linear regression. For classification trees, we will develop a Python algorithm to build a complete classification tree, and then compare it to the one generated by the Rpart library, in order to evaluate the similarities and the performance of our algorithm. Finally, we will discuss the concept of Random Forest, which is widely used to overcome the limitations of CART methods, notably by improving the accuracy and robustness of predictive trees.

Résumé

Les arbres CART (Classification And Regression Trees) ont été créés pour répondre à plusieurs besoins et défis dans le domaine de l'apprentissage statistique et de l'analyse des données. Dans ce mémoire, nous introduirons la méthode CART en expliquant les objectifs de cette méthode et son fonctionnement. Nous examinerons la nature des arbres binaires de décision en abordant à la fois la construction et l'élagage d'un arbre. Nous explorerons également les différents types d'arbres, tels que les arbres de régression et de classification, en mettant en évidence leurs distinctions. Pour les arbres de régression, nous effectuerons une comparaison avec les modèles de régression classiques tels que la régression linéaire multiple. En ce qui concerne les arbres de classification, nous élaborerons un algorithme Python pour construire un arbre de classification complet, puis le comparerons à celui généré par la bibliothèque Rpart, afin d'évaluer les similitudes et la performance de notre algorithme. Enfin, nous aborderons le concept de Random Forest, largement utilisé pour pallier aux limites des méthodes CART, notamment en améliorant la précision et la robustesse des arbres prédictifs.

TABLE DES MATIÈRES

1. Introduction

Les arbres CART (Classification And Regression Trees) sont un outil d'apprentissage automatique populaire développé par Leo Breiman, Jerome Friedman, Richard Olshen et Charles Stone en 1984. L'idée derrière la création de cette méthode était de fournir un moyen simple et visuel d'aborder des problèmes complexes de classification et de régression. Les arbres CART ont été conçus pour répondre à des questions non linéaires et non paramétriques, ce qui les rend particulièrement adaptés aux problèmes de science des données impliquant de grandes quantités de données et des relations complexes entre les variables.

L'objectif principal des arbres CART est de prédire une variable dite expliquée à l'aide de variables explicatives pour chaque individu. Pour cela, la méthode CART consiste à créer un arbre binaire en divisant les individus en sous-groupes homogènes, selon des règles de décision basées sur les variables explicatives. Ainsi, l'objectif est de sélectionner récursivement la variable qui divise le mieux les données à chaque nœud, jusqu'à ce que certains critères d'arrêt soient atteints. Les arbres CART sont capables de traiter des variables continues et catégorielles, ce qui les rend polyvalents et largement applicables.

Le contexte de la création des arbres CART était marqué par une croissance rapide dans le domaine de l'apprentissage automatique. Les chercheurs cherchaient des méthodes plus efficaces et intuitives pour extraire des connaissances à partir de données de plus en plus volumineuses et complexes. La méthode CART a été conçue comme une alternative aux techniques d'apprentissage automatique traditionnelles, telles que la régression linéaire et l'analyse discriminante, qui présentaient des limites pour traiter des problèmes non linéaires et des interactions entre les variables. Ils sont utilisés dans diverses applications, allant de la finance à la médecine, en passant par la biologie, le marketing et l'industrie et sont appréciés pour leur interprétabilité car ils produisent des règles de décision simples et compréhensibles.

Cependant, il convient de noter que les arbres CART présentent également certaines limites. Ils peuvent être sujets à un surapprentissage, c'est-à-dire qu'ils peuvent créer des modèles trop complexes qui fonctionnent bien sur les données d'apprentissage, mais pas sur les données de test ou de nouvelles données. De plus, les arbres CART sont sensibles aux variations dans les données d'apprentissage, de sorte que de légères modifications peuvent entraîner des arbres très différents. C'est pourquoi Léo Breiman créa en 2001 une extension des arbres CART appelée les Forêts Aléatoires. Cette méthode consiste à combiner plusieurs arbres CART afin de créer un modèle de prédiction plus robuste et précis.

Tout d'abord, nous examinerons les arbres binaires décisionnels dans leur généralité afin de comprendre comment ils sont construits. Ensuite nous irons plus loin en nous penchant sur les différences entre les arbres de régression et de classification. Pour les arbres de régression nous effectuerons une comparaison avec une méthode de régression traditionnelle telle que la régression linéaire multiple. Puis pour la classification, après avoir codé un algorithme Python capable de construire un arbre, nous le comparerons avec la bibliothèque R classique Rpart afin d'évaluer la performance de notre algorithme. Enfin, dans une seconde partie nous nous orienterons vers une solution face aux limites de la méthode CART appelée les Forêts Aléatoires.

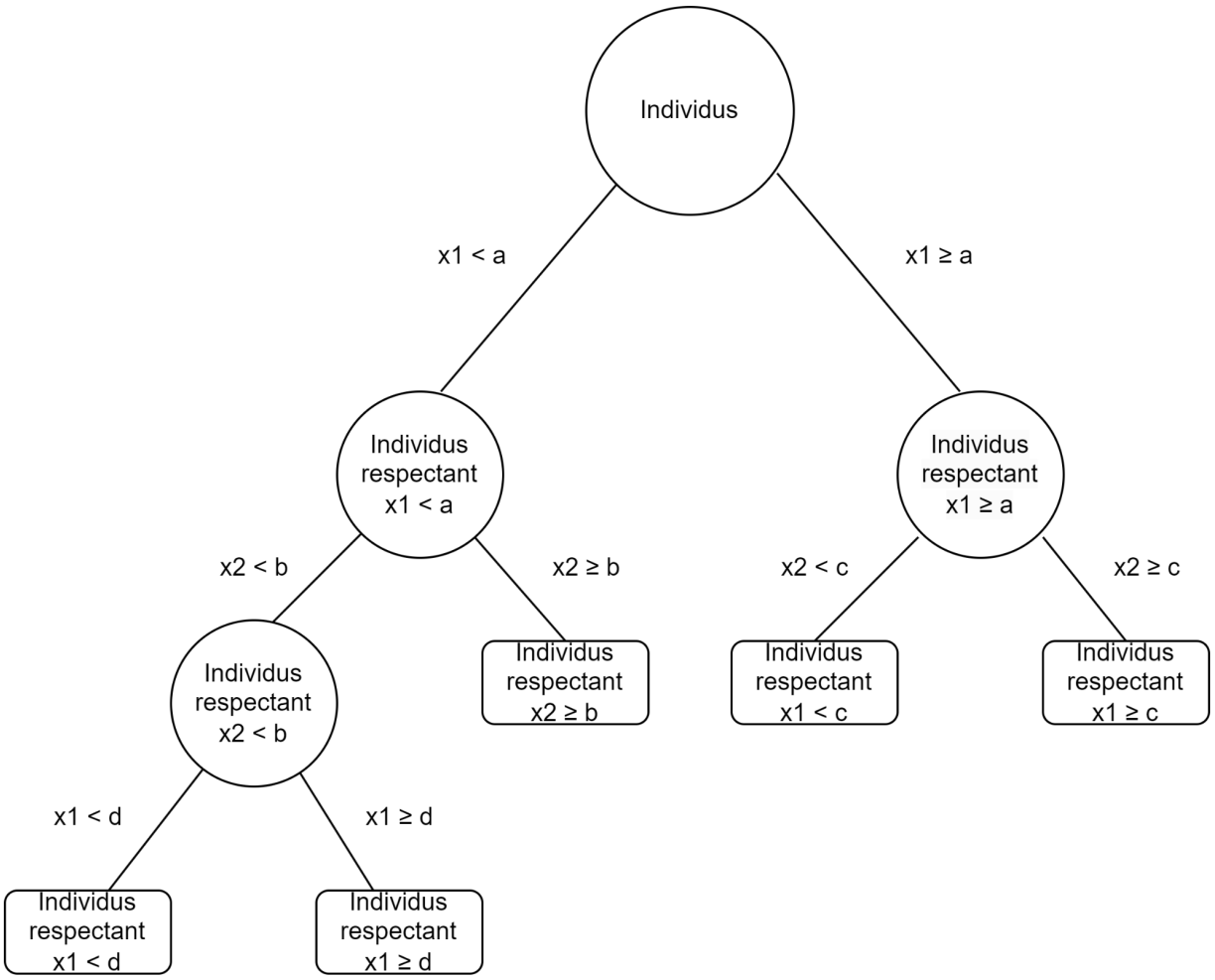
2. Arbres CART

2.1. Les arbres binaires décisionnels

2.1.1. Construction de l'arbre maximal

Un arbre CART (Classification and Regression Tree) est une technique d'apprentissage statistique utilisée pour les tâches de classification et de régression. Il s'agit d'une méthode non paramétrique qui sert à prédire une variable cible (expliquée) d'un individu, en fonction des variables explicatives de celui-ci quelles soient quantitatives ou qualitatives. Lorsque la variable cible sera de nature quantitative, on parlera d'arbre de régression, tandis que lorsqu'elle sera qualitative on parlera d'arbre de classification.

Voici un exemple d'arbre pour que vous puissiez le visualiser.



La méthode CART consiste à construire un arbre binaire en divisant un ensemble d'individus récursivement en deux sous-groupes homogènes en fonction de certaines variables explicatives. Les arbres CART fonctionnent en utilisant une série de tests binaires sur ces variables pour les séparer tels que $x_i < \alpha$ ou $x_i \geq \alpha$ avec $\alpha \in \mathbb{R}$ pour des variables explicatives quantitatives ou $x_i = modalité$ ou $x_i \neq modalité$ pour des variables explicatives qualitatives.

Au début l'ensemble des individus se trouvent dans la racine de l'arbre, deux branches en découlent pour mener à deux nœuds différents. Chaque individu sera attribué au nœud vérifiant la condition binaire de celui-ci. La séparation se répète ainsi jusqu'à qu'une condition d'arrêt (que nous détaillerons après) soit atteinte. A noter, que l'on ne découpe pas un nœud pur, c'est-à-dire un nœud ne contenant que des individus dont les variables cibles sont les mêmes (typiquement en classification). Lorsqu'un nœud n'est plus séparé, on appelle celui-ci une feuille de l'arbre. C'est ici que se trouve la prédiction des prochains individus qui rentreront (à l'aide de leur variables explicatives) dans cette feuille. Il peut s'avérer judicieux de ne pas faire apprendre le modèle sur l'ensemble des données mais d'en garder une petite partie afin de valider le modèle par la suite. C'est-à-dire, tester sur ces nouvelles données l'arbre afin de voir s'il peut s'adapter à de nouvelles données.

L'objectif est d'obtenir une homogénéité maximale intra-groupe et une hétérogénéité entre chaque groupe. C'est-à-dire que la variable cible de chaque individu soit identique dans un même groupe mais différente par rapport à l'autre. Si c'est le cas, on parlera de nœud parfaitement pur, sinon il peut s'avérer impur et on dira donc que la séparation des individus s'est révélée être inneficace.

Pour séparer les individus, il faut donc choisir une variable explicative afin de comparer sa valeur à un seuil (ou une modalité en classification). Le choix de ces deux attributs devra effectuer une séparation en deux groupes homogènes afin de bien représenter l'ensemble des individus dans chaque nœud. En régression l'ensemble des individus est représenté par la moyenne du nœud tandis qu'en classification il est représenté par la modalité majoritaire.

De ce fait, la question essentielle est : Quelle variable explicative va t-on comparer avec quel seuil afin d'effectuer la séparation binaire qui va le mieux représenter l'ensemble des individus (maximiser l'homogénéité intra-groupe) ? On mesure l'homogénéité de chaque groupe d'un arbre de régression différemment de celui d'un arbre de classification.
Nous y reviendrons plus en détail dans les prochaines parties dédiées à chaque type d'arbre.

2.1.2. Condition d'arrêt

La prochaine étape est le choix de la condition d'arrêt. L'utilisation des conditions d'arrêt appropriées dépend des données, du problème à résoudre et des objectifs de l'analyse. Les conditions d'arrêt permettent de contrôler la complexité de l'arbre en évitant qu'il ne devienne trop grand ou trop profond. Un arbre moins complexe est plus facile à interpréter et peut avoir une meilleure capacité de généralisation.

Il existe plusieurs conditions d'arrêts :

- La profondeur : On peut définir une profondeur maximale pour l'arbre (quantifiée par les étages de l'arbre à partir de la racine, initialisée à 0). Ceci permet de contrôler la complexité de l'arbre.

- Nombre minimal d'individus dans un nœud : On peut fixer un seuil pour le nombre minimal d'échantillons requis pour diviser un nœud. En dessous de ce nombre, le nœud devient alors une feuille. Cette condition permet d'éviter de créer des nœuds avec trop peu d'échantillons, ce qui pourrait entraîner du surapprentissage.

- Nombre minimal d'individus dans une feuille : De même, on peut fixer un seuil pour le nombre minimal d'individus requis dans une feuille. Si une division entraîne un nœud fils avec moins d'individus que ce seuil, la division n'aura pas lieu et le nœud parent devient alors une feuille. Cette condition permet également de réduire le risque de surapprentissage.

- Pureté minimale : On peut définir une valeur minimale pour la pureté afin d'autoriser une division. Si une division ne conduit pas à une amélioration suffisante de la pureté, la division n'aura pas lieu et le nœud devient alors une feuille.

Il est important de noter que différentes conditions d'arrêt peuvent être combinées pour obtenir un arbre CART bien équilibré. Dans la pratique, il peut être utile de tester différentes combinaisons de conditions d'arrêt et de paramètres pour trouver le meilleur compromis entre la performance de l'arbre, sa complexité et sa capacité à généraliser à de nouvelles données.

Bien que les conditions d'arrêt puissent être efficaces pour contrôler la croissance de l'arbre, elles ne garantissent pas nécessairement un arbre optimal. L'élagage, en revanche, permet d'optimiser la structure de l'arbre en supprimant les nœuds ou les branches qui n'améliorent pas significativement les performances. Cette combinaison de méthodes permet d'obtenir un arbre avec de meilleures performances sur de nouvelles données.

2.1.3. Élagage

L'élagage (ou "pruning" en anglais) est une technique utilisée pour réduire le surapprentissage des arbres de classification et de régression. En effet, lorsqu'un arbre est construit en entier, s'il n'y a pas de condition d'arrêt, celui-ci ne possède que des feuilles contenant un seul individu (ou plusieurs identiques). On appelle cela du surajustement (overfitting), il se produit lorsque le modèle est trop adapté aux données d'entraînement et ne parvient pas à généraliser correctement pour les nouvelles données inconnues. C'est-à-dire qu'un arbre aura la meilleure prédiction possible sur les données d'apprentissage, mais une prédiction médiocre sur les données dites de validation.

Pour généraliser l'arbre aux nouvelles données il est donc nécessaire d'élaguer l'arbre, lui retirer des branches. Cette phase est identique pour les arbres de régression et pour les arbres de classification et nous étudierons donc ici seulement le cas de la régression. Ainsi, l'objectif est de supprimer des branches ou des nœuds qui ont une faible contribution à la précision du modèle. Cependant, il faut garder en tête que plus on taille un arbre, moins il est performant en termes de prédiction sur les données d'entraînement. Il est donc nécessaire de trouver un équilibre entre un élagage suffisant pour éviter le surajustement et un élagage excessif qui pourrait entraîner un sous-ajustement (underfitting), où le modèle devient trop simple pour prédire avec précision.
On parle ici de trouver un compromis entre la complexité de l'arbre et la précision de la prédiction.

La méthode courante d'élagage développée par Breiman et al. (1984) est l'élagage à coût-complexité (CCP, Cost-Complexity Pruning). Elle évalue le compromis entre la performance de l'arbre et sa complexité. L'élagage à coût-complexité supprime les sous-arbres qui ont une contribution minimale à la réduction de l'erreur de l'arbre, en les remplaçant par des feuilles.

Cette méthode repose sur l'application d'un critère qui permet d'atteindre un équilibre entre l'exactitude des données et la taille de l'arbre. De cette manière, il est possible de déterminer un "meilleur" sous-arbre pour chaque taille.

Le critère en question est : $crit_{\alpha}(T) = R(T) + \alpha|T|$ où $\alpha \geq 0$ et T un arbre.

- $R(T)$ mesure l'erreur d'un arbre et permet donc de traduire la qualité de l'ajustement de l'arbre sur les données d'apprentissage. Cette fonction représente la somme des sommes des résidus pour chaque nœud et plus l'arbre est ajusté aux données, plus la fonction décroît.

  $$R(T) = \sum_{t \in T} R(t) \quad \text{avec} \quad R(t) = \frac{1}{n} \sum_{X_i \in t} (Y_i - \overline{Y}(t))^2 \quad \text{avec } t \text{ un noeud}$$

- $|T|$ représente la complexité du modèle (le nombre de feuilles) et α un poids de cette complexité.

Ainsi $R(T)$ mesure l'ajustement aux données et décroît avec le nombre de feuilles alors que |T| qui quantifie la complexité, croît avec le nombre de feuilles.
Le paramètre α règle la pénalité : plus α est grand, plus les modèles complexes c'est-à-dire comptant beaucoup de feuilles, sont pénalisés. Nous vous redirigeons vers le cours de Christine Malot-Tuleau disponible dans la bibliographie afin d'en savoir plus sur le choix de ce paramètre α.

L'objectif consiste à minimiser la valeur de $crit_{\alpha}(T)$, ce qui permet de trouver un équilibre optimal entre la fidélité aux données (en minimisant $R(T)$) et la simplicité de l'arbre (en minimisant $|T|$). En réduisant $crit_{\alpha}(T)$, on parvient à obtenir un modèle qui s'adapte efficacement aux données sans être excessivement complexe, limitant ainsi le risque de surapprentissage et améliorant la capacité de généralisation du modèle.

En résumé, l'élagage dans les arbres CART est une technique essentielle pour contrôler la complexité du modèle, en supprimant des branches ou des nœuds peu contributifs à la performance globale de

l'arbre. Il aide à éviter le surajustement et améliore la capacité du modèle à généraliser sur de nouvelles données. Ainsi, les praticiens peuvent trouver un équilibre optimal entre la complexité du modèle et sa performance sur des données inconnues, ce qui permet d'obtenir des modèles CART plus robustes et précis.

2.2. Arbres de régression

2.2.1. Définition

Un arbre de régression CART est une méthode d'apprentissage automatique qui prend en entrée un ensemble de variables explicatives, qui peuvent être à la fois quantitatives et qualitatives, et renvoie une prédiction pour une variable cible qui est quantitative. Ils sont particulièrement adaptés aux problèmes de régression où l'objectif est de prédire une valeur continue plutôt qu'une classe. Les arbres de régression sont également utiles pour résoudre des problèmes non linéaires et pour gérer des données présentant des interactions complexes entre les variables.

La prédiction pour une nouvelle observation est obtenue en traversant l'arbre depuis la racine jusqu'à une feuille, en suivant les divisions successives déterminées par les valeurs des variables explicatives de l'observation. La prédiction est alors la moyenne de la variable cible pour toutes les observations de la feuille correspondante.

2.2.2. Séparation des individus : MSE

Dans les arbres de régression CART, la séparation des individus, également appelée conditions de coupure, est une étape cruciale pour construire un modèle précis et interprétable. Les conditions de coupure déterminent comment les individus sont répartis dans les sous-groupes à chaque étape de la construction de l'arbre. Dans cette section, nous expliquerons comment les conditions de coupure sont choisies et quelles sont les considérations importantes pour obtenir un modèle performant. Nous allons étudier la méthode MSE (Mean Squared Error).

L'erreur quadratique moyenne est un indicateur de vérification de la fiabilité d'un modèle et est utilisée pour mesurer l'adéquation du modèle aux données d'apprentissage, ce qui permet de choisir le modèle le plus approprié. Cet outil étudie les écarts entre les valeurs réellement observées et les valeurs prédites par le modèle. L'erreur quadratique est une valeur toujours positive. L'objectif est de minimiser la MSE pondérée des deux sous-groupes résultants de la séparation. Un seuil de coupure optimal est celui qui entraîne la plus petite MSE pondérée. Plus les valeurs obtenues avec le modèle sont proches des valeurs observées, plus les écarts sont faibles et l'erreur quadratique proche de zéro.

La formule est la suivante : $$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$$

Avec $n \in \mathbb{N}$, le nombre de mesure, y_i la valeur observée de la variable dépendante pour la i-ème observation et $\widehat{y_i}$ est la valeur prédite correspondante.

L'erreur quadratique moyenne a différentes caractéristiques :

- Elle se distingue des autres indicateurs par sa facilité de calcul (par exemple, la MAE utilisant la valeur absolue des différences entre les valeurs observées et les valeurs prédites plutôt que leurs carrés est moins sensible aux erreurs importantes que la MSE).

- C'est un indicateur qui est sensible aux valeurs aberrantes. En effet, en élevant au carré la valeur des écarts, chaque anomalie a un poids important dans le calcul de l'erreur quadratique. Pour réduire l'impact des anomalies certains utilisent plutôt la racine carré de l'erreur quadratique (root mean square) comme estimateur de modèle.

- Elle peut être difficile à interpréter. En effet, une erreur quadratique moyenne de 15 peut être gage de fiabilité si la moyenne des valeurs observées est supérieure à 1000. Cependant une erreur quadratique de 15 prouve qu'un modèle est très peu fiable si la moyenne des valeurs observées est inférieure à 100. Ainsi il est toujours important de rapporter l'erreur quadratique moyenne à la moyenne de valeurs observées pour juger de la fiabilité d'un modèle.

2.2.3. Comparaison avec la régression linéaire multiple

La régression linéaire multiple et les arbres de régression CART sont deux méthodes statistiques distinctes utilisées pour modéliser la relation entre une variable dépendante (réponse) et plusieurs variables indépendantes (prédicteurs). Nous allons comparer les deux méthodes en termes de principes, avantages et inconvénients, et voir quand il vaut mieux utiliser une méthode plutôt que l'autre.

Tout d'abord, rappelons le principe de la régression multiple. L'objectif général est d'en savoir plus sur la relation entre plusieurs variables indépendantes ou prédictives et une variable dépendante ou de critère.

Le modèle est défini comme suit : $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \varepsilon$

où Y est la variable dépendante, X_1, X_2, …, X_p sont les variables indépendantes, β_0, β_1, ..., β_p sont les coefficients de régression, et ε est l'erreur résiduelle.

Pour comparer ces deux méthodes statistiques, nous allons les utiliser sur un même dataset, puis comparer leurs prédictions. Le dataset utilisé représente la variation des caractéristiques du sang en fonction du sport pratiqué, de la taille corporelle et du sexe de l'athlète.

Voici un extrait :

	rcc	wcc	hc	hg	ferr	bmi	ssf	pcBfat	lbm	ht	wt	sex	sport
2	3,96	7,5	37,5	12,3	60	20,56	109,1	19,75	63,32	195,9	78,9	f	B_Ball
3	4,41	8,3	38,2	12,7	68	20,67	102,8	21,3	58,55	189,7	74,4	f	B_Ball
4	4,14	5	36,4	11,6	21	21,86	104,6	19,88	55,36	177,8	69,1	f	B_Ball
5	4,11	5,3	37,3	12,6	69	21,88	126,4	23,66	57,18	185	74,9	f	B_Ball
6	4,45	6,8	41,5	14	29	18,96	80,3	17,64	53,2	184,6	64,6	f	B_Ball
7	4,1	4,4	37,4	12,5	42	21,04	75,2	15,58	53,77	174	63,7	f	B_Ball
8	4,31	5,3	39,6	12,8	73	21,69	87,2	19,99	60,17	186,2	75,2	f	B_Ball
9	4,42	5,7	39,9	13,2	44	20,62	97,9	22,43	48,33	173,8	62,3	f	B_Ball
10	4,3	8,9	41,1	13,5	41	22,64	75,1	17,95	54,57	171,4	66,5	f	B_Ball
11	4,51	4,4	41,6	12,7	44	19,44	65,1	15,07	53,42	179,9	62,9	f	B_Ball
12	4,71	5,3	41,4	14	38	25,75	171,1	28,83	68,53	193,4	96,3	f	B_Ball
13	4,62	7,3	43,8	14,7	26	21,2	76,8	18,08	61,85	188,7	75,5	f	B_Ball
14	4,35	7,8	41,4	14,1	30	22,03	117,8	23,3	48,32	169,1	63	f	B_Ball
15	4,26	6,2	41	13,9	48	25,44	90,2	17,71	66,24	177,9	80,5	f	Row
16	4,63	6	43,7	14,7	30	22,63	97,2	18,77	57,92	177,5	71,3	f	Row

Le dataset possède au total 202 observations et 13 variables.
Ici nous avons choisi de prédire la variable numérique “rcc” (qui correspond au nombre de globules rouges dans le sang) en fonction des autres (sans le type de sport).

Les variables explicatives sont : wcc (nombre de globules blancs en 10^{12} par litre), hc (hématocrite en pourcentage), hg (concentration d'hémoglobine en g/dL), ferr (ferritines plasmatiques en ng/dL), bmi (indice de masse corporelle en kg/cm^2), ssf (somme des plis cutanés), pcBfat (pourcentage de masse grasse), lbm (masse maigre en kg), ht (taille en cm), wt (poids en kg) et sex.

Pour ce faire, nous avons procédé en plusieurs étapes. Nous avons commencé par charger les bibliothèques nécessaires et les données, en supprimant la variable "sport" du jeu de données. Ensuite, nous avons effectué une analyse descriptive des variables et avons divisé le jeu de données en ensembles d'entraînement (80%) et de test (20%). Nous avons remarqué 11 variables quantitatives pour une seule variable qualitative.

```
# Diviser le dataset en ensembles d'entraînement et de test
set.seed(123)
trainIndex <- createDataPartition(data$rcc, p = 0.8, list = FALSE)
trainData <- data[trainIndex, ] #données d'entrainement = 80%
testData <- data[-trainIndex, ] #données de test = 20%
```

Ensuite, pour créer le modèle de régression linéaire multiple, nous avons utilisé la fonction stepAIC pour sélectionner les variables les plus significatives.

```
# Créer un modèle de régression linéaire multiple avec stepAIC
linear_model <- lm(rcc ~ ., data = trainData)
step_linear_model <- stepAIC(linear_model, direction = "both")
summary(step_linear_model)
```

On obtient le modèle suivant :

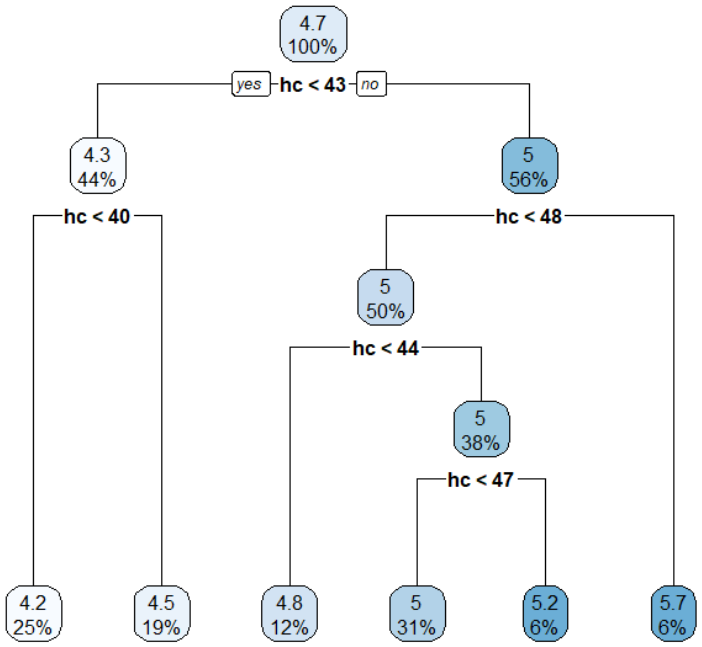
```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.279278   0.156972  -1.779   0.0771 .
hc           0.116031   0.003632  31.949   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Seule la variable “hc”, qui correspond au pourcentage d’hématocrite a été sélectionnée dans ce modèle.

Nous avons ensuite créé l’arbre en mesurant sa complexité avec le paramètre cp :

```
# Créer un modèle d'arbre de régression CART
tree_model <- rpart(rcc ~ ., data = trainData, control = rpart.control(cp = 0.01))
```

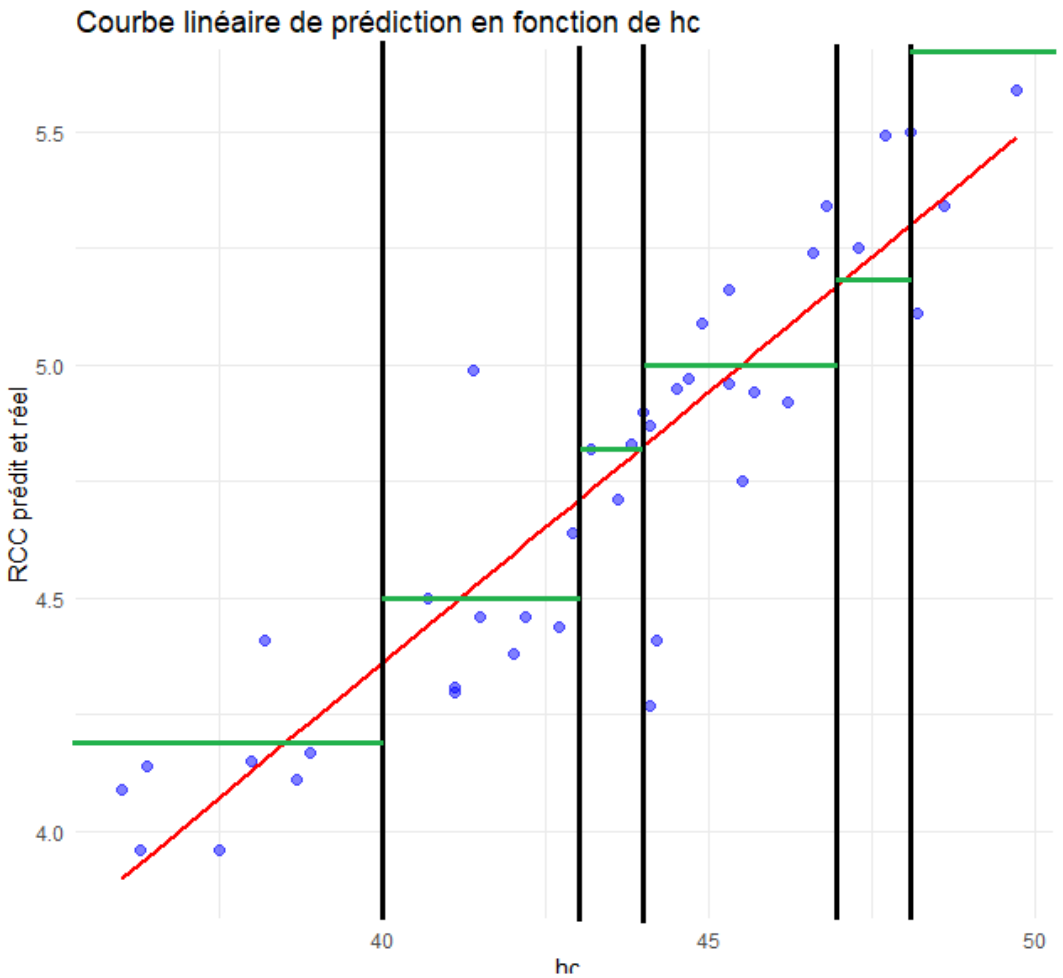
cp = 0.01 signifie que pour qu'une division supplémentaire ait lieu, elle doit améliorer la SSE d'au moins 0,01. Si une division potentielle n'améliore pas suffisamment la SSE, l'élagage se produit pour éviter le surapprentissage. La SSE correspond à la MSE multipliée par le nombre d’observations, c’est la somme totale des erreurs quadratiques. On obtient l'arbre suivant :



On remarque également que seule la variable “hc” est utilisée pour la construction de l’arbre.

Une fois les deux modèles construits, nous avons effectué des prédictions pour ces deux modèles sur les données de test.

Voici un graphique proposant une régression linéaire multiple et une découpe par morceaux pour représenter l'arbre CART :



Graphique représentant les valeurs rcc par rapport à la variable hc avec une régression linéaire multiple et une découpe en morceaux (CART)

—— : Découpe en sous-ensemble conformément à l'arbre de régression

—— : Valeurs prédites pour chaque sous-ensemble

—— : Équation de la droite de régression linéaire multiple

Nous avons également calculé la MSE sur les données de test pour évaluer leur performance. Voici les résultats obtenus :

```
> cat("MSE for linear regression model:", linear_mse, "\n")
MSE for linear regression model: 0.03516774
> cat("MSE for CART tree model:", tree_mse, "\n")
MSE for CART tree model: 0.05857231
```

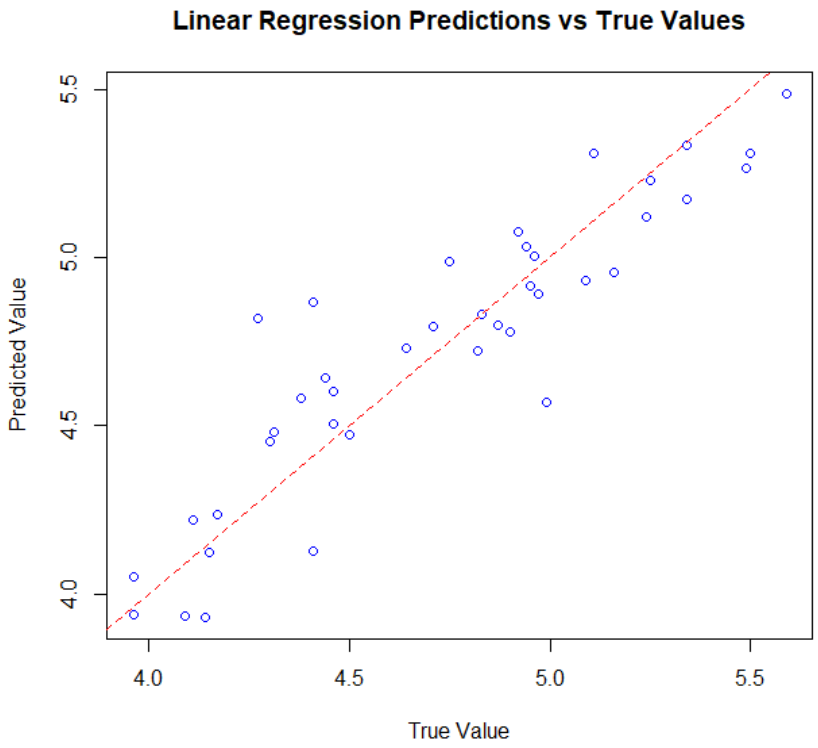
La MSE étant plus faible pour le modèle de régression linéaire multiple, on en déduit qu'il généralise mieux et est plus performant sur des nouvelles données que l'arbre de régression CART.

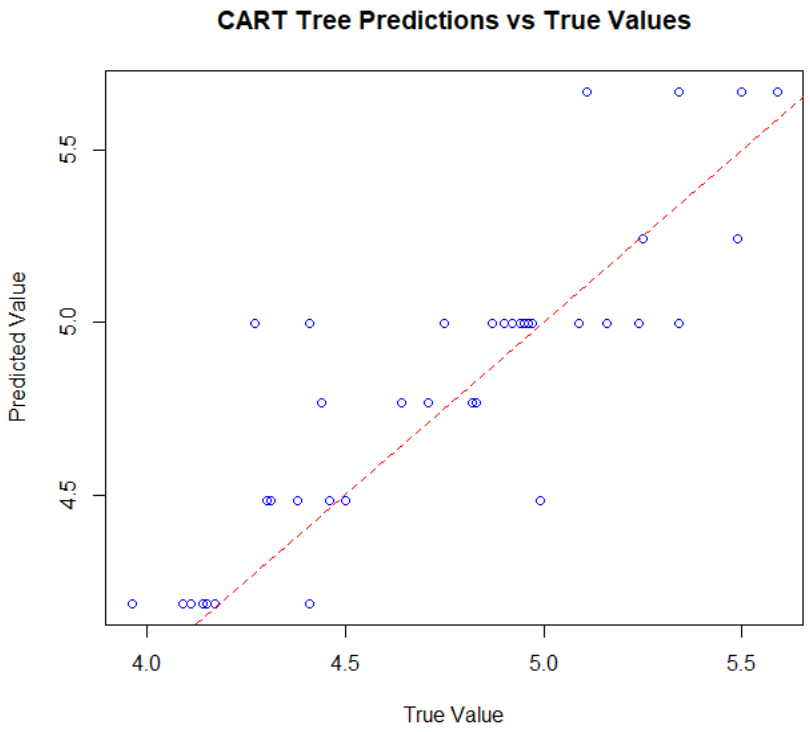
Nous avons aussi calculé la MSE sur les données d'entraînement pour les deux modèles afin de détecter un éventuel surapprentissage. En effet, si la MSE sur les données d'entraînement est beaucoup plus faible que celle sur les données de test, cela peut indiquer que le modèle est sur ajusté aux données. Voici les résultats obtenus :

```
> cat("MSE for linear regression model on training data:", linear_train_mse, "\n")
MSE for linear regression model on training data: 0.02794409
> cat("MSE for CART tree model on training data:", tree_train_mse, "\n")
MSE for CART tree model on training data: 0.03421508
```

Ici on remarque justement que la MSE sur les données d'entraînement est plus faible que celle sur les données de test, cela pourrait indiquer un surapprentissage pour l'arbre de régression CART. Toutefois, la différence n'est pas très grande, il est donc peu probable que cela soit un problème majeur.

Enfin, nous avons créé des graphiques de dispersion pour visualiser les prédictions des deux modèles par rapport aux valeurs réelles sur les données de test :





On voit que les prédictions faites sur les nouvelles données à l'aide de l'arbre CART sont plus dispersées que celles faites grâce au modèle de régression multiple.

Nous pouvons donc dire que dans ce cas là, de part la MSE, et comme le montre ces deux graphiques, la méthode de régression linéaire multiple semble la meilleure pour prédire la variable "rcc" en fonction des autres variables.

La régression linéaire multiple possède :

- des avantages : Facile à interpréter, résultats explicites sous forme d'équation, approche paramétrique, efficace avec des relations linéaires.
- des inconvénients : Hypothèse de linéarité, sensibilité aux valeurs aberrantes, multicolinéarité.

De même, les arbres de régression CART possèdent :

- des avantages : Capable de capturer des relations non linéaires et des interactions, robuste face aux valeurs aberrantes, facile à visualiser et interpréter, adapté aux problèmes de classification et de régression.
- des inconvénients : Risque de surapprentissage, instabilité, moins performant pour les relations linéaires pures.

Il faut donc mieux utiliser la régression linéaire multiple lorsque les relations entre les variables sont linéaires ou que vous souhaitez estimer des coefficients de régression. L'utilisation des arbres de régression CART sont quant à eux utilisés lorsque les relations entre les variables sont non linéaires, complexes ou que vous recherchez un modèle robuste et facile à interpréter.

Dans notre exemple, la régression linéaire multiple (simple au final puisqu'une seule variable est retenue) est meilleure que l'arbre CART car il y a une forte relation linéaire entre la variable expliquée rcc et la variable explicative hc. Cependant, un arbre CART pourrait s'avérer plus efficace lorsque les relations entre les variables ne seraient pas linéaires.

En résumé, le choix entre la régression linéaire multiple et les arbres de régression CART dépend des caractéristiques des données, des hypothèses sous-jacentes et des objectifs de l'analyse.

2.3. Arbres de classification

2.3.1. Définition

Un arbre de classification CART est une méthode prenant en entrée un ensemble de variables explicatives, qui peuvent être à la fois quantitatives et qualitatives, et renvoie une prédiction pour une variable cible qui est qualitative. Ils sont particulièrement adaptés aux problèmes de classification où l'objectif est de prédire une modalité plutôt qu'une valeur continue.

La prédiction pour un nouvel individu est obtenue en traversant l'arbre depuis la racine jusqu'à une feuille, en suivant les divisions successives déterminées par les valeurs des variables explicatives de l'individu. La prédiction est alors la modalité majoritaire de la variable cible pour tous les individus de la feuille correspondante.

2.3.2. Séparation des individus : indice de Gini

Contrairement aux arbres de régression CART, qui prédisent une valeur continue pour une variable cible quantitative, les arbres de classification CART visent à prédire une modalité pour une variable cible qualitative. Pour obtenir une prédiction dans les arbres de classification, la procédure est similaire à celle des arbres de régression : on traverse l'arbre depuis la racine jusqu'à une feuille, en suivant les divisions successives déterminées par les valeurs des variables explicatives. Cependant, au lieu de prédire la moyenne de la variable cible, la prédiction est la modalité majoritaire de la variable cible pour tous les individus de la feuille correspondante.

L'indice de Gini mesure l'impureté d'un nœud, c'est-à-dire à quel point les individus d'un nœud sont mélangés en termes de modalités de la variable cible. Un nœud est dit pur si tous ses individus appartiennent à la même modalité, et impur si les individus sont répartis sur différentes modalités. L'indice de Gini varie entre 0 (pureté maximale) et 1 (impureté maximale).

Soit M l'ensemble des modalités et t un nœud. La formule de l'indice de Gini pour ce nœud est la suivante :

$$\phi(t) = 1 - \sum_{i \in M} p_i(t)^2$$

Lors de la construction d'un arbre de classification, l'objectif est de minimiser l'indice de Gini global de l'arbre. Pour cela, à chaque étape, on choisit la variable explicative et le seuil de coupure qui minimisent la somme pondérée des indices de Gini des deux nœuds fils résultants.

Soit n_g et n_d les nombres d'individus respectifs aux noeuds fils gauche et droit, et n_t le nombre total d'individus dans le noeud parent. t_g et t_d sont les nœuds fils gauche et droit respectifs du noeud t.

La somme pondérée des indices de Gini est calculée comme suit :

$$\phi_p(t) = \phi(t_g) * \frac{n_g}{n_t} + \phi(t_d) * \frac{n_d}{n_t}$$

Ceci revient donc à maximiser à chaque étape de construction de l'arbre : $\phi(t) - \phi_p(t)$

En maximisant cette différence, on vise à obtenir la meilleure séparation possible des individus dans les nœuds fils et à créer des nœuds plus purs, avec des individus appartenant majoritairement à une seule modalité de la variable cible, ce qui réduit l'impureté globale de l'arbre et améliore la précision des prédictions.

2.3.3. Exemple d'application en Python

Dans cette partie, nous montrerons comment la séparation de Gini se fait à chaque étape en calculant l'indice de Gini à chaque fois sur le jeu de données small.csv. Le fichier small.csv contient les données suivantes:

```
Y,X1,X2
A,11,0.7
A,7,0.8
A,14,0.9
A,11,1.1
A,12,1.2
B,13,0.8
B,9,1.0
B,7,1.1
B,10,1.2
```

Dans ce jeu de données, nous avons deux variables explicatives (X1 et X2) et une variable cible qualitative Y avec deux modalités A et B.

Tout d'abord, en se basant sur les formules précédentes, on calcule l'indice de Gini initial pour l'ensemble des données de départ au noeud 0 :

$$\phi(0) = 1 - (\frac{5}{9})^2 + (\frac{4}{9})^2 = 0.4938$$

Ensuite, en parcourant chaque variable et chaque seuil de celle-ci, il faut calculer l'indice de Gini des noeuds fils :

Pour la variable X1, en prenant le premier seuil 11, on remarque que 4 individus respectent la condition X1 < 11, dont 1 ayant pour modalité A et 3 ayant pour modalité B. Ce nœud contient l'ensemble de ces données respectant ces conditions. On en déduit donc pour le noeud fils gauche :

$$\phi(1) = (1 - (\frac{1}{4})^2 + (\frac{3}{4})^2) * (\frac{4}{9}) = 0.1666$$

Pour le nœud fils droit, à l'inverse, 5 individus respectent la condition X1 >= 11 avec 4 ayant pour modalité A et 1 ayant pour modalité B. On en déduit son indice de Gini :

$$\phi(2) = (1 - (\frac{4}{5})^2 + (\frac{1}{5})^2) * (\frac{5}{9}) = 0.1777$$

On en déduit la somme pondérée des indices de Gini :

$$\phi_p(0) = \phi(1) + \phi(2) = 0.3443$$

La différence de Gini donne donc pour le noeud 0, la variable X1 et le seuil 11 :

$$\phi(0) - \phi_p(0) = 0.1494$$

Ces étapes se répètent récursivement de manière à parcourir, dans chaque nœud (parent et leurs fils) qui contient son propre ensemble de données, chaque variable et chaque valeur unique de cette variable.

Le but est de récupérer la différence de Gini maximale, d'identifier la variable et le seuil correspondant pour chacun de ces nœuds, et d'effectuer les séparations optimales (à savoir la maximisation de la différence de Gini) à chaque étape du processus. En combinant ces informations pour tous les nœuds, nous obtenons une série de séparations pour l'ensemble de l'arbre. Une fois ces séparations établies, nous pouvons construire l'arbre de décision qui illustre ces divisions. L'arbre obtenu présente une séparation des données selon les variables X1 et X2, et permet d'identifier les groupes d'individus ayant des caractéristiques similaires en termes de modalités de la variable cible Y.

Pour illustrer cet exemple, nous avons écrit un programme en Python. Nous allons expliquer le but du programme, ce qu'il fait, comment il fonctionne et sa sortie. Le programme utilise un fichier CSV nommé small.csv contenant des données et applique l'algorithme d'arbre de décision pour diviser les données selon les variables X1 et X2.

Le programme (en annexe) est construit autour d'une classe Gini qui contient plusieurs méthodes pour effectuer les différentes étapes de la construction de l'arbre de décision, notamment 3 méthodes importantes :

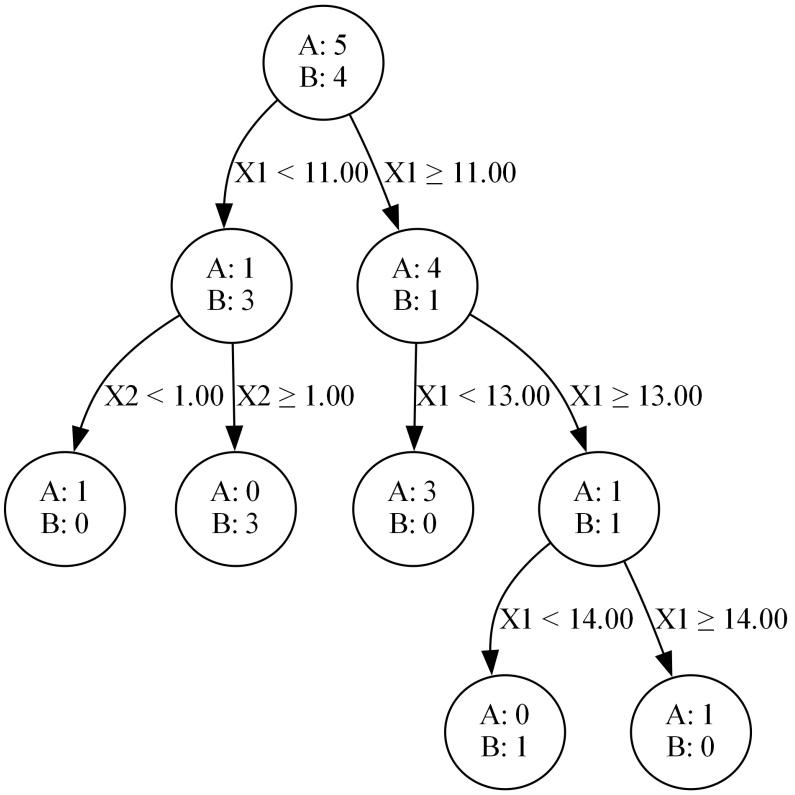
- **“division”** qui est responsable de parcourir toutes les variables et valeurs uniques, de calculer les indices de Gini pour chaque seuil et de retourner la meilleure division possible pour le nœud courant.

- **“recursion”** qui est la clé de la construction de l'arbre. Elle est appelée récursivement pour chaque sous-ensemble de données afin de diviser les données en fonction des seuils optimaux. Elle fait appel à la méthode **“division”** pour obtenir la meilleure division pour le nœud courant, puis effectue des appels récursifs pour les ensembles gauche et droit si les modalités sont différentes.

- **“display”** qui génère une représentation graphique de l'arbre à l'aide de Graphviz. Elle parcourt les nœuds de l'arbre, les affiche en tant que nœuds du graphe et ajoute des liens entre les nœuds parents et enfants. Les étiquettes des arêtes représentent les conditions de séparation et les étiquettes des nœuds indiquent le nombre d'éléments A et B dans chaque nœud.

À la fin de l'exécution du programme, l'arbre est affiché visuellement sous forme d’image avec les nœuds, les conditions de séparation et les nombres d'éléments A et B dans chaque nœud. L'arbre peut ensuite être utilisé pour identifier les groupes d'individus ayant des caractéristiques similaires en termes de modalités de la variable cible Y.

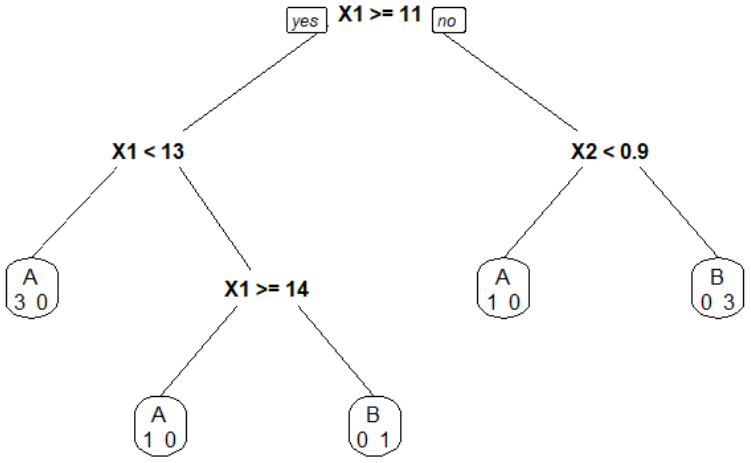
Voilà la sortie du programme à partir du jeu de données de small.csv :



2.3.4. Comparaison avec les bibliothèques R

Afin de valider notre implémentation en Python, nous avons également réalisé la construction de l'arbre de classification en utilisant le langage R et la bibliothèque rpart. Le code se trouve en annexe.

Nous commençons par charger les données du fichier CSV, puis nous décrivons les données et les préparons pour l'analyse. Ensuite, nous utilisons la fonction rpart pour construire l'arbre de décision. Nous utilisons également les fonctions plot, text et prp pour afficher l'arbre graphiquement.



Après avoir exécuté ce code R et comparé les résultats avec notre implémentation en Python, nous avons constaté que les deux approches produisent des arbres de décision identiques, que ce soit dans le choix des variables de séparation ou des seuils. On obtient donc les mêmes prédictions. Cette concordance entre les résultats confirme que notre implémentation en Python est correcte et applique la séparation avec l'indice de Gini conformément à un arbre de classification, pour deux modalités A et B.

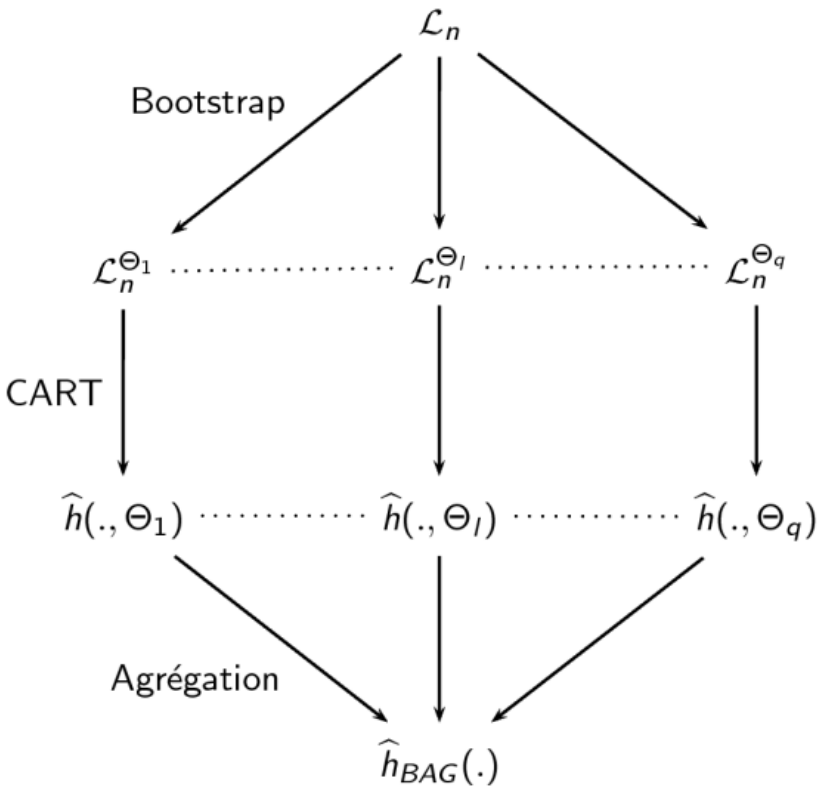
3. Les forêts aléatoires

Les forêts aléatoires, comme leur nom l'indique, sont des ensembles de plusieurs arbres de décision indépendants, dont les prédictions individuelles sont combinées pour aboutir à une prédiction finale plus robuste et précise. Cette approche, basée sur le principe de l'agrégation de modèles, permet de surmonter certains des inconvénients liés aux arbres de décision simples, tels que le surapprentissage et la sensibilité aux variations dans les données d'entraînement. Dans cette partie, nous étudierons les principes fondamentaux des forêts aléatoires, puis nous essaierons de comparer les différents types de prédicteurs pour démontrer l'efficacité des forêts aléatoires.

3.1. Bagging

Cette approche consiste à construire plusieurs prédicteurs individuels en appliquant une méthode de prédiction (ici CART) sur différents échantillons bootstrap. Ceux-ci consistent à tirer aléatoirement un sous-ensemble d'observations avec remise de manière équilibrée. Il reste ensuite à agréger les prédictions de ces prédicteurs pour obtenir un prédicteur final performant.

Voici un schéma qui représente les étapes de cette méthode :



Source : Robin Genuer, Jean-Michel Poggi. Arbres CART et Forêts aléatoires, Importance et sélection de variables. 2017.

Précisément, les étapes consistent à :

1. Générer plusieurs échantillons bootstrap indépendants à partir de l'échantillon d'apprentissage initial L_n. Un échantillon bootstrap $L_n^{\Theta_1}$ est obtenu en tirant aléatoirement n observations avec remise à partir de L_n, chaque observation ayant une probabilité $\frac{1}{n}$ d'être tirée. La variable aléatoire Θ représente ce tirage aléatoire.

2. Appliquer la méthode de prédiction sur chacun des échantillons bootstrap pour obtenir une collection de prédicteurs individuels q notés : $\hat{h}(\Theta_1)$, … , $\hat{h}(\Theta_q)$.

3. Agréger les prédictions des prédicteurs individuels pour obtenir un prédicteur final performant. L'agrégation se fait par la moyenne des prédictions en régression ou par un vote majoritaire en classification.

Agrégation :

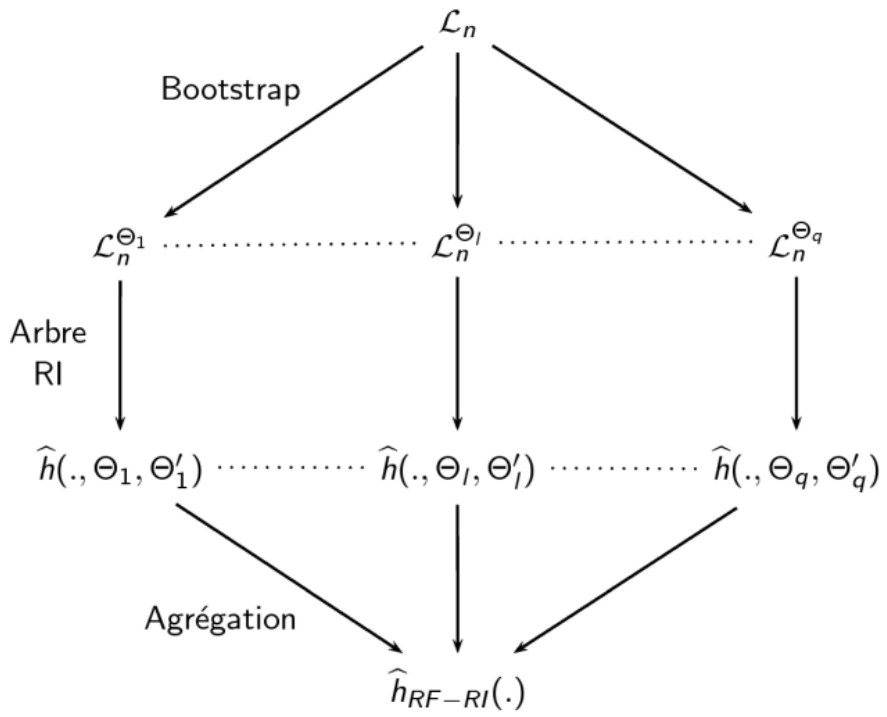
- $\widehat{h}(x) = \frac{1}{q}\sum_{\ell=1}^{q} \widehat{h}(x, \Theta_\ell)$ en régression
- $\widehat{h}(x) = \underset{1\leq c\leq L}{\operatorname{argmax}} \sum_{\ell=1}^{q} \mathbb{1}_{\widehat{h}(x,\Theta_\ell)=c}$ en classification

La performance du Bagging dépend de la méthode de prédiction choisie et du nombre de prédicteurs individuels créés q. En utilisant des échantillons bootstrap pour construire les prédicteurs individuels, on introduit de l'incertitude, ce qui permet de créer des prédicteurs suffisamment diversifiés sans compromettre leurs performances individuelles. Ainsi, le prédicteur agrégé, qui tire parti de la diversité des prédicteurs individuels, contribue à réduire la variance et à améliorer la précision des prédicteurs en augmentant la taille de l'échantillon.

3.2. Random Forest - Random Inputs (RF-RI)

Les Forêts aléatoires à variables d'entrée aléatoires (Random Forests-RI) sont une variante du Bagging qui se distingue par la construction des arbres individuels. Cette méthode, également appelée Random Forests avec Random Inputs, génère plusieurs échantillons bootstrap et applique une version modifiée de l'algorithme CART sur chaque échantillon. Les arbres sont construits en sélectionnant aléatoirement un nombre m de variables pour découper chaque nœud, sans remise, et uniformément parmi p variables. Chaque variable a alors une probabilité 1/p d'être choisie. De plus, l'arbre construit est complètement développé (arbre maximal) et n'est pas élagué. La collection d'arbres obtenus est enfin agrégée (moyenne en régression, vote majoritaire en classification) pour donner le prédicteur Random Forests-RI.

La figure ci-dessous fournit le schéma récapitulatif de l'algorithme RF-RI, où θ désigne le tirage bootstrap et θ' désigne le tirage aléatoire des variables.



Source : Robin Genuer, Jean-Michel Poggi. Arbres CART et Forêts aléatoires, Importance et sélection de variables. 2017.

Le nombre m, qui doit être inférieur ou égal à p, est un paramètre essentiel de la méthode et est fixé au début de la construction de la forêt. Il est identique pour tous les arbres et pour tous les nœuds d'un même arbre. Cependant, les m variables impliquées dans deux nœuds distincts sont généralement différentes. Lorsque m = p, la forêt construite équivaut à un Bagging d'arbres CART non élagués, tandis qu'une forêt avec m = 1 est très différente du Bagging. Dans ce dernier cas, le choix de la variable pour découper un nœud est totalement aléatoire. Afin de choisir la variable m, nous pouvons calculer pour chaque valeur de celle-ci l'erreur OOB (que nous verrons plus tard) et prendre le modèle comprenant la variable m minimisant cette erreur.

Les Random Forests-RI présentent deux sources d'incertitude pour générer la collection de prédicteurs individuels : l'incertitude due au bootstrap et l'incertitude du choix des variables pour découper chaque nœud d'un arbre. Cette méthode perturbe à la fois l'échantillon sur lequel on lance la règle de base et le cœur de la construction de la règle de base. En pratique, les Random Forests-RI, avec un paramètre m bien choisi, améliorent les performances du Bagging.

L'amélioration des performances peut s'expliquer par le fait que l'ajout d'une incertitude supplémentaire pour construire les arbres rend ces derniers encore plus différents les uns des autres, sans pour autant dégrader de manière significative leurs performances individuelles. Le prédicteur agrégé est alors meilleur. La perturbation introduite doit réaliser un compromis entre une trop grande incertitude, qui dégrade les prédicteurs individuels et le prédicteur agrégé, et une trop petite incertitude, qui induit des prédicteurs individuels trop similaires et n'apporte aucune amélioration. Les excellents résultats des Random Forests-RI en pratique suggèrent qu'elles réalisent un bon compromis en injectant la "bonne dose" d'incertitude, avec un paramètre m bien choisi.

3.3. Erreur Out Of Bag (OOB)

L'erreur "Out Of Bag" (OOB) est une méthode d'estimation de la performance d'un modèle de forêt aléatoire.

Dans une forêt aléatoire, chaque arbre est construit à partir d'un échantillon bootstrap, c'est-à-dire un échantillon extrait avec remise de l'ensemble de données d'entraînement. En d'autres termes, lors de la construction de chaque arbre, une partie des données d'entraînement est sélectionnée de manière aléatoire avec remise, et cette partie est utilisée pour entraîner l'arbre. Cependant, étant donné que l'échantillonnage est effectué avec remise, certaines observations ne sont pas utilisées pour entraîner un arbre spécifique. Ces observations non utilisées sont appelées "Out Of Bag" (OOB). En moyenne, environ 37% des observations restent non utilisées pour chaque arbre dans une forêt aléatoire (source : Navnina Bhatia, Data Scientist).

L'erreur OOB est une mesure de la performance du modèle qui est calculée en utilisant ces données OOB. Pour chaque observation OOB, on prédit sa valeur en utilisant uniquement les arbres qui n'ont pas été entraînés sur cette observation. Ensuite, on compare les prédictions $\widehat{Y_i}$ aux véritables valeurs Y_i pour calculer l'erreur OOB. Une erreur OOB faible indique une meilleure performance du modèle.

Après avoir effectué cette opération sur tout l'échantillon, l'erreur OOB du prédicteur est désigné en régression par l'erreur quadratique moyenne : $\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$

et en classification par la proportion d'observations mal classées : $\frac{1}{n}\sum_{i=1}^{n} 1_{\widehat{Y_i} \neq Y_i}$

L'erreur OOB est considérée comme une estimation honnête de la performance du modèle car elle est calculée en utilisant des données qui n'ont pas été utilisées pour entraîner les arbres de décision individuels. Ainsi, elle fournit une évaluation de la performance du modèle sur des données "nouvelles", similaire à l'évaluation sur un ensemble de test ou de validation séparé.

En d'autres termes, l'erreur OOB est une mesure de la capacité de généralisation du modèle. Cela signifie qu'elle mesure comment le modèle se comporte face à des données qu'il n'a jamais vues auparavant. Cette propriété est importante pour évaluer la qualité d'un modèle d'apprentissage automatique, car un bon modèle doit être capable de faire des prédictions précises sur de nouvelles données, et pas seulement sur les données qu'il a déjà vues pendant l'entraînement.

De plus, l'erreur OOB est calculée sans avoir à mettre de côté une partie des données pour la validation ou les tests, ce qui permet d'utiliser l'ensemble des données disponibles pour l'entraînement du modèle. Cela peut être particulièrement avantageux lorsque les données sont limitées, car cela permet d'utiliser plus de données pour entraîner le modèle tout en obtenant une estimation fiable de sa performance.

Nous utiliserons donc par la suite l'erreur OOB pour comparer des prédicteurs entre eux.

3.4. Comparaison des méthodes

Dans cette partie, nous allons comparer les différents prédicteurs que nous avons vu : un arbre CART élagué, un bagging et une forêt aléatoire sur l'échantillon 'datasport' que nous avons utilisé en régression avec les librairies rpart et randomForest. Afin de les comparer, nous utiliserons l'erreur OOB. L'erreur OOB ne pouvant être calculée sur l'arbre de régression, nous appliquerons la même fonction que celle-ci (MSE) mais sur les données de test (non utilisées pour construire l'arbre). Nous verrons aussi l'importance des variables dans chaque modèle, c'est-à-dire à quel point une variable est importante/impacte le modèle.

Tout d'abord nous séparons le dataset : 80% de données d'entraînement et 20% de données de test.

```
# Diviser les données en ensembles d'entraînement et de test
set.seed(123)
trainIndex <- sample(1:nrow(data), 0.8 * nrow(data))
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]
```

Nous créons ensuite l'arbre de régression élagué, le bagging et la forêt aléatoire.

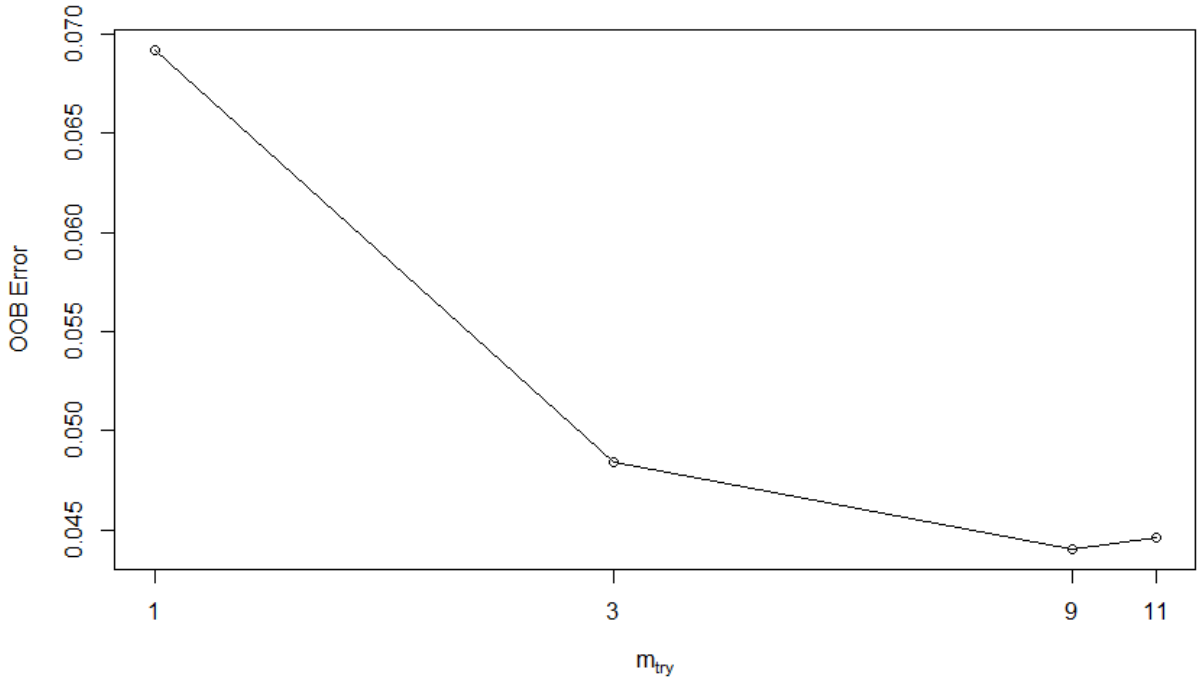
```
# Bagging
set.seed(123)
bagging_model <- randomForest(rcc ~ ., data = trainData, ntree = 500, mtry = ncol(trainData), importance = TRUE)
```

Le Bagging étant la même méthode qu'une forêt aléatoire avec le nombre de variables choisies m = p, on fixe donc l'attribut mtry = ncol(trainData), la variable qu'on souhaite prédire étant automatiquement retirée par la fonction randomForest.

```
# Forêt aléatoire
set.seed(123)
tuned.mtry <- tuneRF(trainData[,-1], trainData$rcc, mtryStart = 1, ntreeTry = 500, stepFactor = 3, improve = 0)
mtry <- tuned.mtry[which.min(tuned.mtry[,"OOBError"]), "mtry"]
set.seed(123)
rf_model <- randomForest(rcc ~ ., data = trainData, ntree = 500, mtry = mtry, importance = TRUE)
```

Quant à la forêt aléatoire, le paramètre qui va faire qu'il va être plus performant que le Bagging est le paramètre mtry. Il est donc primordial de bien définir ce paramètre. Pour cela nous avons calculé l'erreur OOB de la forêt aléatoire pour chaque valeur de mtry avec $mtry \in [1; 11]$.

Sur le graphique ci-dessous, seules quelques valeurs sont affichées.



La valeur du paramètre mtry améliorant le plus la forêt aléatoire est donc $mtry = 9$.

Nous calculons maintenant l'erreur test pour chaque méthode.

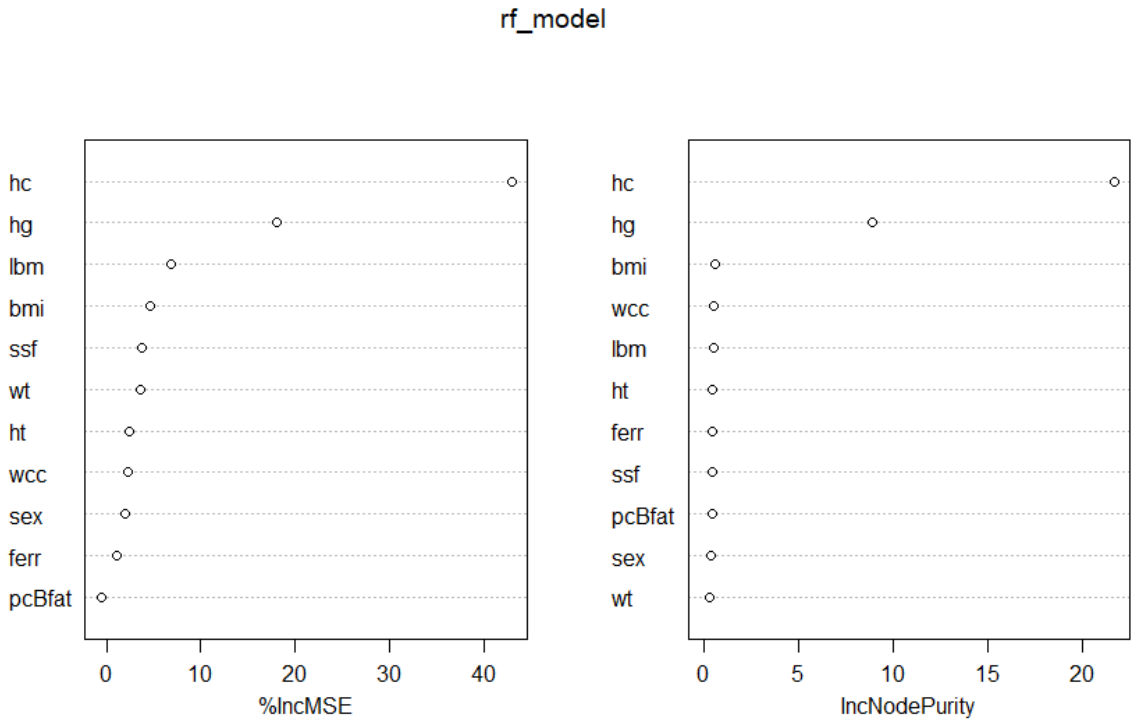
```
# Calculer l'erreur OOB pour chaque méthode
cart_test_error <- sum((predict(cart_optimal, testData) - testData$rcc)^2) / nrow(testData)
bagging_oob <- bagging_model$mse[500]
rf_oob <- rf_model$mse[500]
```

On obtient les résultats ci-dessous :

Prédicteur	Arbre de régression	Bagging	Forêt aléatoire
Erreur test	0.057	0.044	0.043

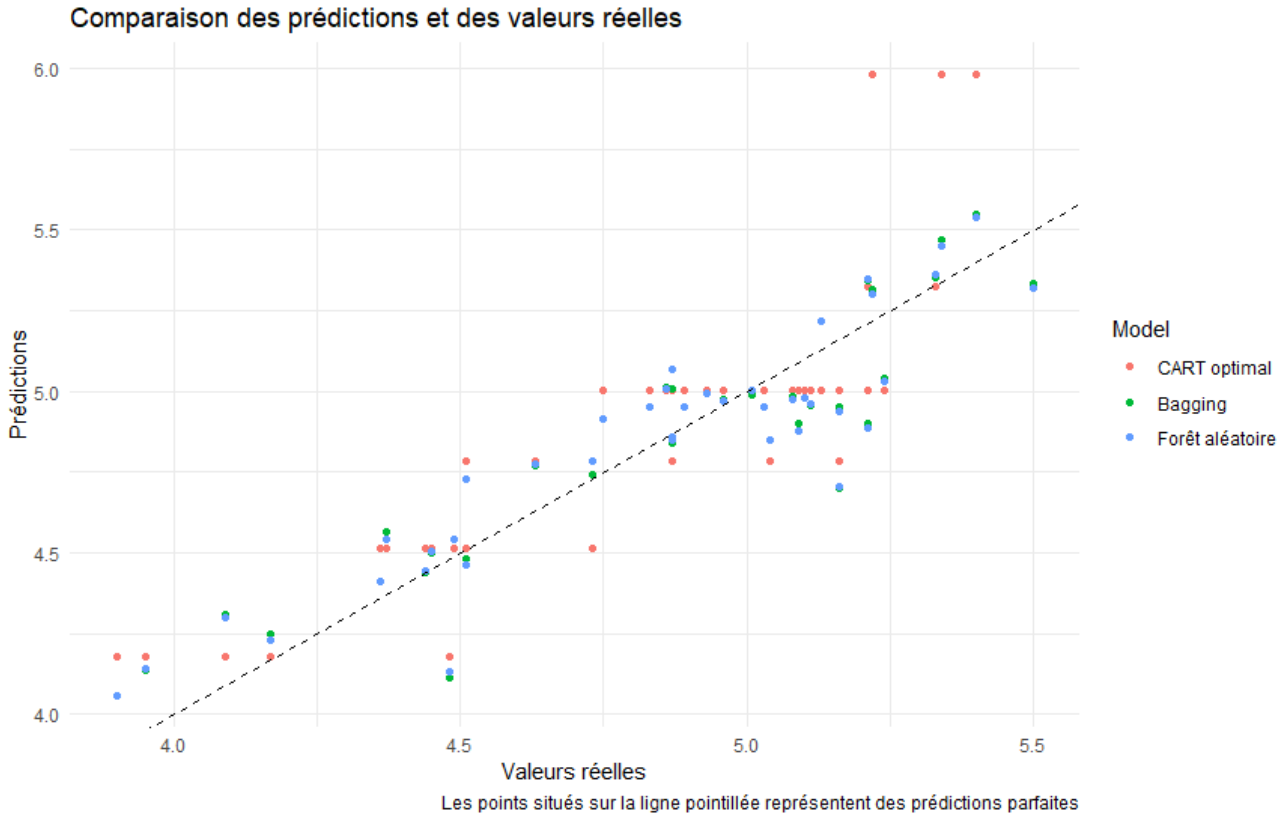
D'après ces résultats, la forêt aléatoire semble offrir les meilleures performances parmi les trois méthodes, avec la plus faible erreur OOB (4,3%). La méthode de bagging a également une erreur OOB relativement faible (4,4%), tandis que l'arbre de régression CART élagué a une erreur légèrement plus élevée sur les données de test (5,7%). Dans certains cas, le bagging peut déjà réussir à réduire suffisamment la corrélation entre les arbres, de sorte que l'ajout du sous-échantillonnage aléatoire des variables dans la forêt aléatoire n'améliore pas significativement les performances.

Nous pouvons désormais s'intéresser à l'importance des variables :



D'après ces graphiques, la variable hc est celle qui contribue le plus au modèle des forêts aléatoires. On obtient le même résultat pour le bagging et comme on l'a vu précédemment, c'était aussi la variable hc qui contribuait majoritairement aux modèles CART et de régression linéaire multiple. On peut noter qu'il serait inutile de refaire un bagging et une forêt aléatoire en ne gardant dans le modèle que les variables hc et hg car cela empêcherait l'aléa générés dans le choix des variables par les forêts aléatoires.

Nous pouvons aussi afficher les prédictions de chaque individu pour chaque modèle :



Finalement, les forêts aléatoires améliorent considérablement la précision des prédictions tout en conservant relativement la même importance à chaque variable.

4. Conclusion

Nous avons exploré au cours de ce mémoire les méthodes d'apprentissage statistiques utilisées dans les problèmes de régression et de classification. Tout d'abord, la méthode CART proposée par Breiman en 1984 a permis d'apporter un moyen simple et visuel de répondre à des enjeux complexes dans le domaine de la science des données. Grâce aux variables explicatives d'une observation, un arbre décisionnel binaire permet de prédire une variable cible à l'aide de critères de séparation en subdivisant récursivement un ensemble de données en sous-ensembles. Cette séparation a pour but de minimiser l'impureté des nœuds en comparant une variable avec un seuil. Le choix de ces critères s'effectue en régression en calculant la moyenne des erreurs au carré et en classification avec l'indice de Gini. Cependant, un arbre complet a tendance à surapprendre les données

d’entraînement et à moins bien généraliser sur des nouvelles données. Si l’arbre est utilisé tel quel, il est donc indispensable d’effectuer un pré-élagage (condition d’arrêt) ainsi qu’un post-élagage (retirer des branches non essentielles à l’amélioration de l’arbre) tout en essayant de trouver un compromis entre la complexité de l’arbre et la précision des prédictions.

Pour les problèmes de régression, nous avons comparé la méthode CART à la régression linéaire multiple sur un jeu de données de taille moyenne. Nous avons pu observer que les variables ont la même importance quelle que soit la méthode utilisée. En ce qui concerne les prédictions sur les données de validation, la méthode de régression linéaire multiple s’avère être la plus juste avec une erreur test (MSE) plus faible que celle de l’arbre de régression. Ainsi, la régression linéaire multiple permet d’offrir une relation explicite sous forme d'équation et se voit être plus performante pour les relations linéaires pures. Toutefois, elle est plus exposée aux valeurs aberrantes et moins interprétable qu’un arbre de régression.

Pour les problèmes de classification, nous avons implémenté un algorithme en Python permettant de simuler la séparation des individus avec l’indice de Gini, seulement pour deux modalités A ou B. En comparant avec les bibliothèques disponibles sur R, nous avons remarqué que les variables et les seuils optimaux pour chaque séparation étaient identiques. Ainsi, les deux arbres de décision obtenus en sortie sont les mêmes, c’est-à-dire que pour chaque individu nous obtiendrons la même prédiction quel que soit l’arbre utilisé.

Cependant, les arbres CART présentent certaines limites, notamment une sensibilité aux variations des données. Pour surmonter ces problèmes, les forêts aléatoires ont été développées. Ce sont des ensembles d'arbres de décision, construits en utilisant des échantillons bootstrap et des sous-ensembles de caractéristiques sélectionnés aléatoirement. En combinant plusieurs arbres CART, les forêts aléatoires abordent les problèmes de surapprentissage et de sensibilité aux variations des données en intégrant la diversité et en réduisant la variance. En effet, comme nous l’avons démontré à l’aide de l’erreur OOB, la forêt aléatoire augmente considérablement la précision des prédictions face à un seul arbre de décision. Aussi, en agrégeant un grand nombre de prédicteurs individuels diversifiés, les forêts aléatoires renforcent considérablement la robustesse du modèle. Ainsi, une modification d'une variable ou d'une observation peut avoir un impact important sur un arbre CART, mais un effet beaucoup moins prononcé sur une forêt aléatoire.

Les forêts aléatoires semblent donc être une méthode plus appropriée dans le domaine de la science des données en offrant plusieurs avantages comparé à une méthode CART, notamment une réduction de la variance, la possibilité d'évaluer l'importance des caractéristiques ainsi qu’une robustesse face aux données bruitées et aux changements de données. Cependant, elles présentent également quelques inconvénients, tels que la complexité accrue notamment avec le nombre de données, le temps de calcul plus long et une interprétabilité réduite par rapport aux arbres de décision CART individuels.

5. Annexe

5.1. Algorithmes

5.1.1. Comparaison d'un arbre de régression et d'une régression linéaire multiple

```
library(rpart)
library(rpart.plot)
library(MASS)
library(caret)
library(ggplot2)

data <- read.csv2("datasport.csv")
data <- subset(data, select = -sport)

# Analyse descriptive des variables numériques
summary(data)
# 1 variables quali, le reste quanti

# Diviser le dataset en ensembles d'entraînement et de test
set.seed(123)
trainIndex <- createDataPartition(data$rcc, p = 0.8, list = FALSE)
trainData <- data[trainIndex, ] #données d'entraînement = 80%
testData <- data[-trainIndex, ] #données de test = 20%

# Créer un modèle de régression linéaire multiple avec stepAIC
linear_model <- lm(rcc ~ ., data = trainData)
step_linear_model <- stepAIC(linear_model, direction = "both")
summary(step_linear_model)
selected_vars <- names(coef(step_linear_model))
cat("Variables sélectionnées dans la régression linéaire multiple:", "\n")
print(selected_vars) #il y 1 variables sélectionnées dans ce modèle

# Créer un modèle d'arbre de régression CART
tree_model <- rpart(rcc ~ ., data = trainData, control = rpart.control(cp = 0.01))

#arbre élagué,
#cp = 0.01 signifie que pour qu'une division supplémentaire ait lieu, elle doit
améliorer la SSE d'au moins 0,01.
#Si une division potentielle n'améliore pas suffisamment la SSE, l'élagage se produit ->
éviter le suraprentissage

rpart.plot(tree_model)
printcp(tree_model)
tree_vars <- unique(tree_model$frame$var)
tree_vars <- tree_vars[tree_vars != "<leaf>"]  # Supprimez les nœuds feuilles
cat("Variables sélectionnées dans l'arbre de régression CART:", "\n")
print(tree_vars) #seule la variable hc est utilisée pour la construction de l'arbe

# Prédire sur l'ensemble de test
linear_preds <- predict(step_linear_model, testData)
```

```r
tree_preds <- predict(tree_model, testData)

# Calculer l'erreur quadratique moyenne (MSE) pour chaque modèle
linear_mse <- mean((testData$rcc - linear_preds)^2)
tree_mse <- mean((testData$rcc - tree_preds)^2)

# Afficher les résultats
cat("MSE for linear regression model:", linear_mse, "\n")
cat("MSE for CART tree model:", tree_mse, "\n")

# Prédire sur l'ensemble d'entraînement
linear_train_preds <- predict(step_linear_model, trainData)
tree_train_preds <- predict(tree_model, trainData)

# Calculer l'erreur quadratique moyenne (MSE) pour chaque modèle sur l'ensemble
d'entraînement
linear_train_mse <- mean((trainData$rcc - linear_train_preds)^2)
tree_train_mse <- mean((trainData$rcc - tree_train_preds)^2)

# Afficher les résultats
cat("MSE for linear regression model on training data:", linear_train_mse, "\n")
cat("MSE for CART tree model on training data:", tree_train_mse, "\n")

# Afficher les résultats
cat("MSE for linear regression model on training data:", linear_train_mse, "\n")
cat("MSE for linear regression model on test data:", linear_mse, "\n")
cat("Difference in MSE for linear regression model:", abs(linear_train_mse -
linear_mse), "\n\n")

cat("MSE for CART tree model on training data:", tree_train_mse, "\n")
cat("MSE for CART tree model on test data:", tree_mse, "\n")
cat("Difference in MSE for CART tree model:", abs(tree_train_mse - tree_mse), "\n")

#différence plus importante entre les MSE d'entraînement et de test pour l'arbre de
régression CART que pour la régression linéaire multiple.
#Cela pourrait indiquer un surapprentissage pour l'arbre de régression CART. Toutefois,
la différence n'est pas très grande,
#il est donc peu probable que cela soit un problème majeur.


# Créer un dataframe pour stocker les prédictions et les valeurs réelles
predictions <- data.frame(
  TrueValue = testData$rcc,
  LinearPrediction = linear_preds,
  TreePrediction = tree_preds
)

# Créer un graphique de dispersion pour la régression linéaire multiple
plot(testData$rcc, linear_preds,
     main = "Linear Regression Predictions vs True Values",
     xlab = "True Value",
     ylab = "Predicted Value",
     col = "blue")
abline(0, 1, col = "red", lty = 2)

# Créer un graphique de dispersion pour les arbres de régression CART
```

```
plot(testData$rcc, tree_preds,
     main = "CART Tree Predictions vs True Values",
     xlab = "True Value",
     ylab = "Predicted Value",
     col = "blue")
abline(0, 1, col = "red", lty = 2)

#la regression linéaire multiple est la meilleure méthode dans ce cas-là (on le remarque
grace au MSE, et graphiquement)

# Préparer les données pour le graphique
plot_data <- data.frame(hc = testData$hc, true_rcc = testData$rcc, linear_preds =
linear_preds)

# Tracer la courbe linéaire de prédiction en fonction de hc
ggplot(plot_data, aes(x = hc, y = true_rcc)) +
  geom_point(color = "blue", alpha = 0.5, size = 2) +
  geom_line(aes(y = linear_preds), color = "red", size = 1) +
  labs(title = "Courbe linéaire de prédiction en fonction de hc",
       x = "hc",
       y = "RCC prédit et réel") +
  theme_minimal()
```

5.1.2. Algorithme Python en classification

```
import pandas as pd
import graphviz
import uuid

class Gini():
    def __init__(self, file) -> None:
        self.df = pd.read_csv(file)
        self.df = pd.DataFrame(self.df)
        self.seuils_optimaux = [] # arbre contenant les informations : les seuils
optimaux, avec leur gini, la variable, les ID de noeuds correspondant, le niveau dans
l'arbre
        self.tree = {} # arbre qui servira pour l'affichage, dictionnaire stockant des
noeuds, leurs noeuds fils et les modalités de ces noeuds fils
        self.main()

    # Méthode pour trouver la meilleure division d'un noeud
    def division(self, df):
        def calcul_gini(noeud, total):
            p1, p2 = noeud.value_counts().get('A', 0), noeud.value_counts().get('B', 0)
            n = p1 + p2
            if n == 0:
                return 0
            return (1 - ((p1/n)**2 + (p2/n)**2)) * (n/total)

        tab_gini = pd.DataFrame(columns=['variable', 'seuil', 'gini'])
        total = len(df)

        i = 0
```

```
        for x in df.columns[1:]:
            for seuil in df[x].unique():
                noeud_gauche = df[df[x] < seuil].Y
                noeud_droite = df[df[x] >= seuil].Y
                gini = calcul_gini(df['Y'], total) - (calcul_gini(noeud_gauche, total) +
calcul_gini(noeud_droite, total))
                tab_gini.loc[i] = [x, seuil, gini]
                i += 1

        tab_gini.sort_values(by="gini", ascending=False, inplace=True)
        tab_gini.reset_index(drop=True, inplace=True)
        return tab_gini.iloc[0]

    # Méthode récursive pour construire l'arbre de décision
    def recursion(self, df, level=0, parent_noeud_id=None):
        result = self.division(df)  # obtenir la meilleure division pour ce noeud

        noeud_id = len(self.seuils_optimaux)  # générer un ID pour ce noeud
        self.seuils_optimaux.append([level, result['variable'], result['seuil'],
result['gini'], noeud_id])

        # récupérer les modalités pour les ensembles gauche et droit
        mg = list(df[df[result['variable']] < result['seuil']].Y)
        md = list(df[df[result['variable']] >= result['seuil']].Y)

        # ajouter le noeud à l'arbre avec ses informations
        self.tree[noeud_id] = {'parent': parent_noeud_id, 'enfant': [], 'mg': mg, 'md':
md}

        # si le noeud a un parent, ajouter ce noeud comme enfant gauche ou droit du
parent
        if parent_noeud_id is not None:
            # ajouté en tant qu'enfant gauche ou droit de son parent
            # en fonction de la majorité des éléments qui sont inférieurs ou égaux au
seuil du parent
            parent_seuil = self.seuils_optimaux[parent_noeud_id][2]
            parent_variable = self.seuils_optimaux[parent_noeud_id][1]
            direction = 'g' if (df[parent_variable] < parent_seuil).sum() >
(df[parent_variable] >= parent_seuil).sum() else 'd'
            self.tree[parent_noeud_id]['enfant'].append((noeud_id, direction))

        # effectuer des appels récursifs pour les ensembles gauche et droit si les
modalités sont différentes
        for c in ['g', 'd']:
            if 'A' in self.tree[noeud_id][f'm{c}'] and 'B' in
self.tree[noeud_id][f'm{c}']:
                new_df = df[df[result['variable']] < result['seuil']] if c == 'g' else
df[df[result['variable']] >= result['seuil']]

                if len(new_df) < len(df):
                    self.recursion(new_df, level + 1, noeud_id)

    # Méthode pour afficher l'arbre de décision
    def display(self):
        dot = graphviz.Digraph(comment='Arbre de décision',
                               graph_attr={'size': '20,20!', 'fontsize': '14'})
```

```
        for noeud in self.seuils_optimaux:
            level, variable, seuil, _, noeud_id = noeud
            mg = self.tree[noeud_id]['mg']
            md = self.tree[noeud_id]['md']

            nb_A_gauche, nb_B_gauche = mg.count('A'), mg.count('B')
            nb_A_droite, nb_B_droite = md.count('A'), md.count('B')

            # afficher les enfants decoulant de ce noeud
            if level > 0:
                enfants = []

                if not ('A' in mg and 'B' in mg):
                    enfants.append((mg, f"{variable} < {seuil:.2f}", nb_A_gauche,
nb_B_gauche))
                if not ('A' in md and 'B' in md):
                    enfants.append((md, f"{variable} ≥ {seuil:.2f}", nb_A_droite,
nb_B_droite))

                for _, label, nb_A, nb_B in enfants:
                    enfant_id = str(uuid.uuid4())
                    dot.node(enfant_id, label=f"A: {nb_A}\nB: {nb_B}", shape="oval")
                    dot.edge(f"{noeud_id}", enfant_id, label=label)

            dot.node(f"{noeud_id}", label=f"A: {nb_A_gauche + nb_A_droite}\nB:
{nb_B_gauche + nb_B_droite}", shape="oval")

            # lier ce noeud avec son parent
            if level > 0:
                parent_id = self.tree[noeud_id]['parent']
                parent_noeud = [n for n in self.seuils_optimaux if n[-1] ==
parent_id][0]
                _, parent_variable, parent_seuil, _, _ = parent_noeud

                enfant_gauche = next((enfant for enfant in
self.tree[parent_id]['enfant'] if enfant[0] == noeud_id and enfant[1] == 'g'), None)
                direction = '<' if enfant_gauche is not None else '≥'
                dot.edge(f"{parent_id}", f"{noeud_id}", label=f"{parent_variable}
{direction} {parent_seuil:.2f}")

        dot.render("arbre_decision", view=True, format="png")

    def main(self):
        self.recursion(self.df)
        self.display()

if __name__ == "__main__":
    Gini('small.csv')
```

5.1.3. Algorithme R en classification

```
library(rpart)
library(rpart.plot)

#Chargement des données
data = read.csv2("data.csv", sep = ",")
data$X2 = as.numeric(data$X2)

#Construction de l'arbre
data.Tree <- rpart(Y~.,data=data,method= "class",
control=rpart.control(minsplit=0,cp=0))
prp(data.Tree,extra=1)
```

5.1.4. Algorithme erreurs OOB

```
# Charger les packages
library(rpart)
library(rpart.plot)
library(randomForest)
library(ggplot2)
library(reshape2)

# Charger le jeu de données
data <- read.csv2('datasport.csv')
data = data[, -13]

# Diviser les données en ensembles d'entraînement et de test
set.seed(123)
trainIndex <- sample(1:nrow(data), 0.8 * nrow(data))
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]

# Arbre de régression CART optimal (élagué)
cart_model <- rpart(rcc ~ ., data = trainData, method = "anova", control =
rpart.control(minsplit=5,cp = 0))
cart_optimal <- prune(cart_model, cp =
cart_model$cptable[which.min(cart_model$cptable[,"xerror"]),"CP"])

# Visualiser l'arbre élagué
rpart.plot(cart_optimal)

# Bagging
set.seed(123)
bagging_model <- randomForest(rcc ~ ., data = trainData, ntree = 500, mtry =
ncol(trainData), importance = TRUE)

# Forêt aléatoire
set.seed(123)
tuned.mtry <- tuneRF(trainData[,-1], trainData$rcc, mtryStart = 1, ntreeTry = 500,
stepFactor = 3, improve = 0)
```

```
mtry <- tuned.mtry[which.min(tuned.mtry[,"OOBError"]), "mtry"]
set.seed(123)
rf_model <- randomForest(rcc ~ ., data = trainData, ntree = 500, mtry = mtry, importance
= TRUE)

# Calculer l'erreur OOB pour chaque méthode
cart_test_error <- sum((predict(cart_optimal, testData) - testData$rcc)^2) /
nrow(testData)
bagging_oob <- bagging_model$mse[500]
rf_oob <- rf_model$mse[500]

# Comparer les erreurs OOB
cat("Erreur sur les données de test pour CART optimal :", cart_test_error,
    "\nErreur OOB Bagging :", bagging_oob,
    "\nErreur OOB Forêt aléatoire :", rf_oob)

# Importance des variables
varImpPlot(bagging_model)
varImpPlot(rf_model)


# Calculer les prédictions pour chaque modèle
cart_pred <- predict(cart_optimal, testData)
bagging_pred <- predict(bagging_model, testData)
rf_pred <- predict(rf_model, testData)

# Créer un dataframe pour les prédictions et les valeurs réelles
predictions <- data.frame(
  True = testData$rcc,
  CART = cart_pred,
  Bagging = bagging_pred,
  RandomForest = rf_pred
)

predictions_long <- reshape2::melt(predictions, id.vars = "True", variable.name =
"Model", value.name = "Prediction")

# Créer un graphique de dispersion pour comparer les prédictions aux valeurs réelles
ggplot(predictions_long, aes(x = True, y = Prediction, color = Model)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed") +
  theme_minimal() +
  labs(title = "Comparaison des prédictions et des valeurs réelles",
       x = "Valeurs réelles",
       y = "Prédictions",
       caption = "Les points situés sur la ligne pointillée représentent des prédictions
parfaites") +
  scale_color_discrete(labels = c("CART optimal", "Bagging", "Forêt aléatoire"))
```

5.2. Bibliographie

Breiman, L. (1984). Classification and regression trees. Routledge.

Robin Genuer, Jean-Michel Poggi. Arbres CART et Forêts aléatoires, Importance et sélection de variables. preprint 2017, HAL.

Lovely Analytics, Un arbre de décision avec R

Apiacoa, Initiation à rpart

Christophe Chesneau, Introduction aux arbres de décision (de type CART)

Christine Malot-Tuleau, Méthodes CART Introduction à la sélection de variables

Yannig Goude, Arbres de régression, CART

Marin Ferecatu, Arbres de décision

Cole Eagland, Tree-Based Methods - Bagging and Random Forests