T. D. n° 2
Arbres de décision

Résumé

Ce document est le TD n° 2 du module Modèle pour la datascience. Il reprend rapidement des éléments du cours et propose une mise en pratique interactive des arbres de décision. La technique de l'arbre de décision est employée en classement pour détecter des critères permettant de répartir les individus d'une population en n classes. On commence par choisir la variable qui sépare le mieux les individus en plusieurs classes (généralement 2). Nous avons alors des sous populations distinctes, à partir d'un ensemble initial, que l'on appellera nœud. On réitère la même opération sur chaque nouveau nœud obtenu jusqu'à ce que la séparation des individus ne soit plus pertinente (nous définirons cette notion de pertinence plus tard). Les nœuds finaux sont appelés *feuilles*. Après la construction de l'arbre nous obtenons des feuilles caractérisant une seule classe. Un individu est affecté à une feuille (sous entendu : une classe), avec une certaine probabilité, lorsqu'il satisfait l'ensemble des règles lui permettant d'aboutir à la feuille. L'ensemble des règles de toutes les feuilles constitue le modèle de classement.

1 Les arbres de régression

Les arbres de décision conçus pour le classement permettent généralement d'effectuer une prédiction, en faisant varier le critère de scission pour chaque nœuds (à l'exception de quelques techniques particulières, comme QUEST, qui combine toutes les variables pour choisir le critère de scission). Dans un arbre de régression, on travaille exclusivement sur des variables quantitatives continues. On cherche à prédire la valeur d'une variable pré-définie en entrée de la fonction, puis on construit un arbre à partir des autres variables explicatives. Le principe de l'arbre est le suivant :

- La variable à expliquer doit avoir une variance plus faible dans les nœds-fils que le nœud-père [1].
- La variable à expliquer doit avoir une moyenne la plus distincte possible d'un nœud-fils à un autre.

Il faut donc choisir des nœuds-fils qui minimisent la variance intraclasse et maximise la variance interclasse. On peut appliquer un test de Fisher-Snedecor au rapport *Variance intraclasse / Variance interclasse*.

On considère qu'il faut séparer un nœud lorsque le rapport des variances a au plus 20% de chance d'être aussi élevé si la variable à expliquer est indépendante de la variables choisie pour scinder le nœud-père. Intuitivement on sépare un nœud lorsqu'on a au moins 20% de chance d'observer une différence sur la variance (entre le nœud initial et le nœud séparé).

1. Au niveau le plus élevé il y a un nœud racine, dit : nœud père. Au niveau directement inférieur, il y a au moins deux nœuds fils

1.1 Chargement des données

On décide d'observer les différences de salaire pour une population de chercheurs dans un service de recherche. Nous avons comme information :

- X : l'ID de la personne évaluée
- *rank* : la fonction du chercheur
- *discipline* : la discipline du chercheur (A ou B)
- *yrs.since.phd* : le nombre d'années depuis le doctorat (dans le monde impitoyable de la recherche, sans doctorat, point de salut)
- *yrs.service* : le nombre d'années dans le service
- *sex* : si la personne est homme ou femme
- *salary* : salaire du chercheur brut annuel (en euros)

Vous pouvez télécharger les données sur `https://vincentarelbundock.github.io/Rdatasets/datasets.html`.

FIGURE 1 – Une prédiction avec un large intervalde confiance

source :`http://www.phdDelirium.com/`

Commencez par charger les données dans **R**. Supprimez la variable ID qui est inintéressante pour notre évaluation.

```
> data <- read.csv2('C:/Users/claey/Documents/cour/My TD/TD5/
   Salaries.csv', sep =',', header = TRUE)
> data$X <- NULL
> salaire <- as.data.frame(data)
> summary(data)
```

```
          rank      discipline  yrs.since.phd    yrs.service
                         sex
  AssocProf   : 64     A   :181   Min.    : 1.00   Min.    : 0.00
     Female   : 39
  AsstProf    : 67     B   :216   1st Qu.:12.00   1st Qu.: 7.00
     Male     :358
  Prof        :266                Median :21.00   Median :16.00
                                  Mean   :22.31   Mean   :17.61
                                  3rd Qu.:32.00   3rd Qu.:27.00
                                  Max.   :56.00   Max.   :60.00

    salary
 Min.   : 57800
 1st Qu.: 91000
 Median :107300
 Mean   :113706
 3rd Qu.:134185
 Max.   :231545
```

À vous !

a) Commentez les valeurs médianes pour les variables quantitatives et les modalités les plus représentées pour les variables qualitatives.

b) Pourquoi la sur-représentation de certaines modalités peut biaiser une prédiction basée uniquement sur une régression linéaire simple ?

1.2 Package *Tree*

Vous allez utiliser la fonction *tree()* du package *Tree*. Cette fonction génère un arbre par partitionnement récursif binaire en utilisant la variable qu'on souhaite prédire (mise dans une formule spécifiée en entrée) et en fractionnement les variables explicatives. La variable à prédire doit être continue. La variable à prédire est divisée en $X < a$ et $X > a$, de façon itérative et en faisant varier la valeur de a. La scission a est la valeure qui maximise la réduction de l' "impureté" . Le fractionnement itératif est poursuivi jusqu'à ce que les nœuds terminaux soient trop petits ou trop peu différentiables pour être divisés. Appliquez la fonction *tree()* sur votre dataframe pour prédire la variable *salary* à partir de toutes les variables (avec le symbole $\sim$). Vous remarquerez dans l'arbre de la figure 2 et 3 que vous avez exclusivement des variables quantitative. *tree()* séléctionne automatiquement les variables quantitatives parmi l'ensemble des variables.

```
> library(tree)
> tree.Lin <- tree(salary ~ yrs.service + yrs.since.phd, data=
    salaire)
> plot(tree.Lin)
> text(tree.Lin, cex=.75)
```

FIGURE 2 – Arbre de régression linéaire sur la variable *salary*

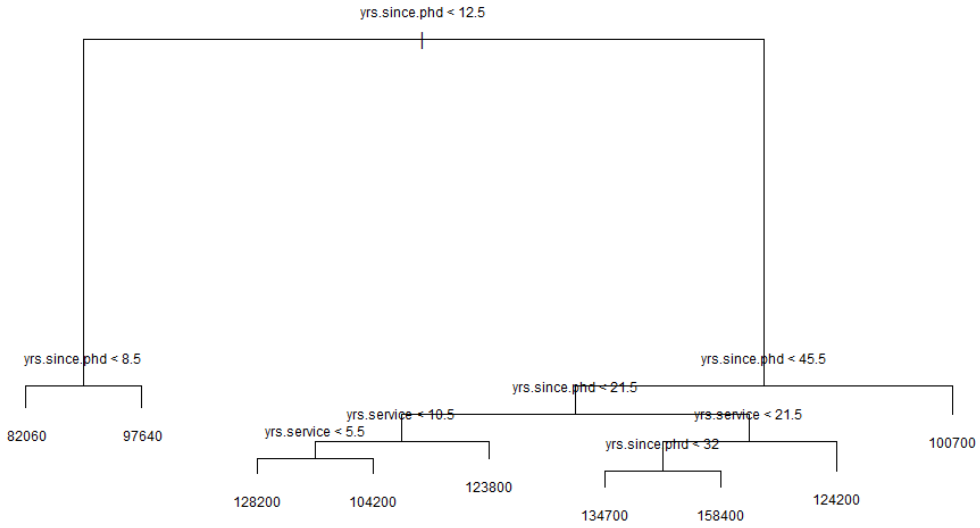
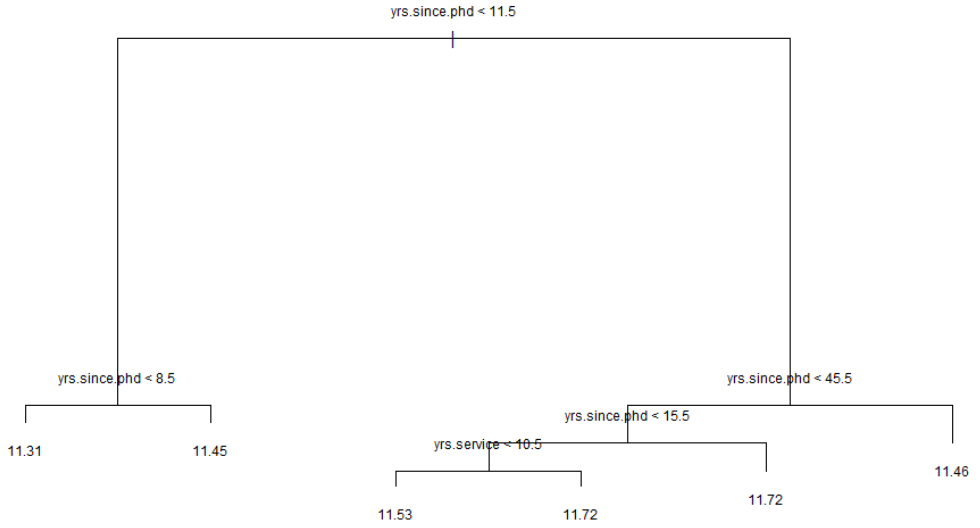


FIGURE 3 – Arbre de régression linéaire sur le logarithme décimale de la variable *salary*



À vous !

a) Quel est le profil du chercheur au meilleur salaire ?

b) Combien de classes ont-été générées ?

c) Créez l'arbre *tree.model* avec la fonction *tree*, en ciblant cette fois le logarithme décimale du salaire (utilisez la fonction *log()* sur la variable *salary*, directement dans la fonction *tree()*).

d) Affichez cet arbre. Vous devez obtenir la figure 3 en sortie.

e) Donnez trois raison d'utiliser le logarithme décimale du salaire plutôt que la valeur du salaire.

1.3 Observation des classes

Pour la suite de ce TD vous allez utiliser l'arbre *tree.model*. Vous allez donc réaliser un certain nombre de classes et vous utiliserez les résultats de votre arbre pour interpréter vos données. La fonction *quantile()* produit des quartiles associés aux probabilités données. La plus petite observation correspond à une probabilité de 0 et la plus grande à une probabilité de 1. En paramètre vous allez rentrer 10 quantils. Vous allez ensuite associer les salaires des chercheurs à un quantile. Grâce à ces deux opérations, vous allez faire représenter les salaires à travers ~~50 nuances de grey~~ 10 nuances de gris. Plus le salaire est foncé, plus il probable de le voir apparaître. Affichez les années de service (*yrs.service*) en fonction des années depuit le sacro saint doctorat (*yrs.since.phd*). Vous ajoutez également la couleur représentative de la probabilité d'avoir ce salaire à vos point. Vous avez donc trois informations. Vous allez également partitionner vos résultats.

```
> salar.deciles <- quantile(salaire$salary, 0:10/10)
> cut.prices    <- cut(salaire$salary, salar.deciles, include.
    lowest=TRUE)
> plot(salaire$yrs.service, salaire$yrs.since.phd, col=grey
    (10:2/11)[cut.prices], pch=20, xlab="yrs.service",ylab="yrs.
    since.phd")

####Partition
> partition.tree(tree.model, ordvars=c("yrs.service","yrs.since.phd
    "), add=TRUE)

# Figure 5
plot(salaire$yrs.since.phd, salaire$salary, pch=19, col=as.numeric(
    salaire$rank))
partition.tree(tree.model, label="Species", add=TRUE)
legend("topright",legend=unique(salaire$rank), col=unique(as.
    numeric(salaire$rank)), pch=19)
```

FIGURE 4 – *yrs.service* en fonction de *yrs.since.phd* (avec la probabilité associée au *salary*) et partitionnement

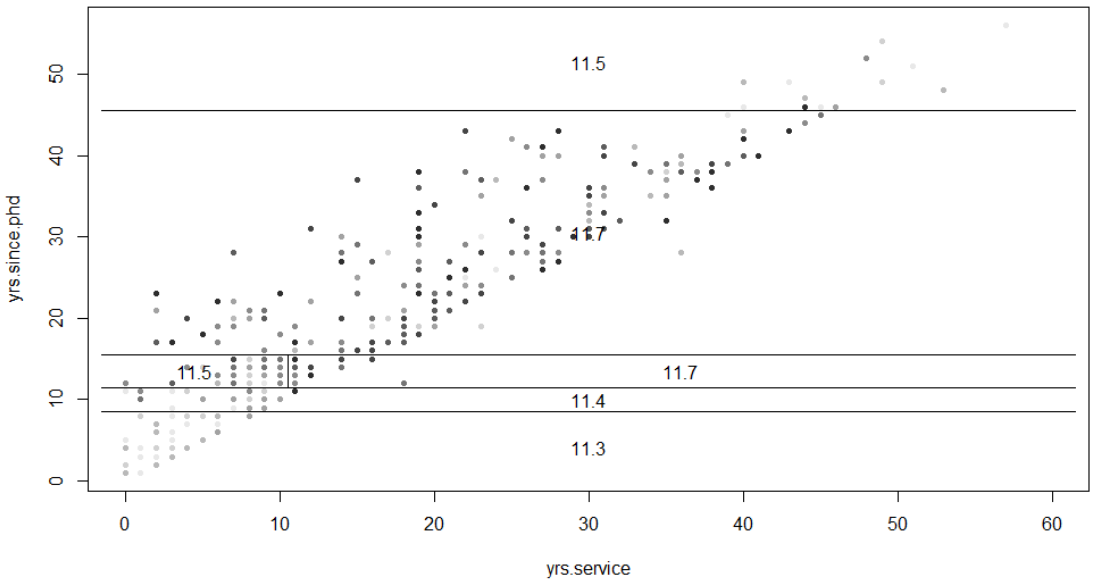
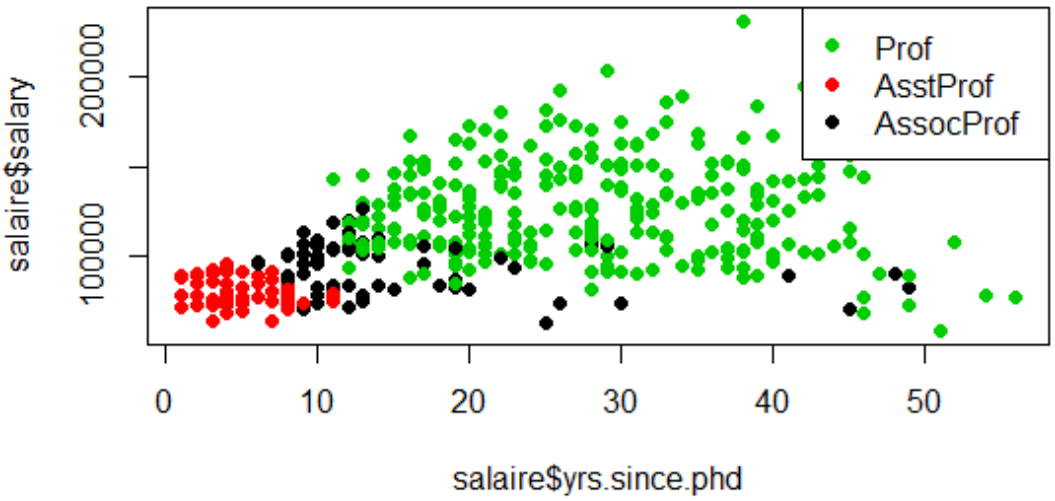


FIGURE 5 –



À vous !

a) Expliquez comment a été réalisé la partition à l'aide de la fonction *partition.tree()*.

b) Commentez les classes de votre arbre par rapport à votre graphique de la figure 4.

c) Interprétez la figure 5.

1.4 Évaluation et amélioration de l'arbre

On décide d'appliquer la fonction *summary()* sur notre arbre *tree.model*. La déviance signifie ici l'erreur quadratique moyenne. La flexibilité d'un arbre est essentiellement contrôlé par le nombre de feuilles qu'ils possède. La fonction d'ajustement de l'arbre dépend de paramètres qui limitent sa croissance. Chaque nœud doit, par exemple, contenir un certain nombre de points, ou encore l'ajout d'un nœud doit réduire la déviance d'au moins une certaine valeur fixée.
Une autre technique d'amélioration est l'utilisation de la fonction *prune.tree()*. Dans des situations complexes, nous pouvons avoir des arbres extrêmement raffinés et donc associés à des modèles de prévisions très instables car fortement dépendants des échantillons qui ont permis leur estimation. On se trouve donc dans une situation de sur-ajustement à éviter au profit de modèles plus parcimonieux donc plus robuste au moment de la prévision. C'est l'objectif de la fonction *prune.tree()*. Le principe de la démarche, introduite consiste à construire une suite emboîtée de sous-arbres à partir de l'arbre maximum par élagage successif puis à choisir, parmi cette suite, l'arbre optimal au sens d'un critère (ici la déviance). La solution ainsi obtenue par un algorithme pas à pas n'est pas nécessairement globalement optimale mais l'efficacité et la fiabilité sont préférées à l'optimalité.

```
> summary(tree.model)

Regression tree:
tree(formula = log(salary) ~ yrs.service + yrs.since.phd, data =
    salaire)
Number of terminal nodes:  6
Residual mean deviance:  0.04121 = 16.11 / 391
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.673700 -0.144300  0.009754  0.000000  0.127400  0.710200

###Nouveau model
> tree.model2 <- tree(log(salary) ~ yrs.service + yrs.since.phd,
    data=salaire, mindev=0.001)
> plot(tree.model2)
> text(tree.model2, cex=.75)


> summary(tree.model2)

Regression tree:
```

```
tree(formula = log(salary) ~ yrs.service + yrs.since.phd, data =
    salaire,
     mindev = 0.001)
Number of terminal nodes:  44
Residual mean deviance:  0.0334 = 11.79 / 353
Distribution of residuals:
     Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
-0.54710 -0.11090 -0.00208  0.00000  0.10890  0.65470

#Elagation
> pruned.tree <- prune.tree(tree.model, best=4)
> plot(pruned.tree)
> text(pruned.tree)
```

FIGURE 6 – *tree.model2*

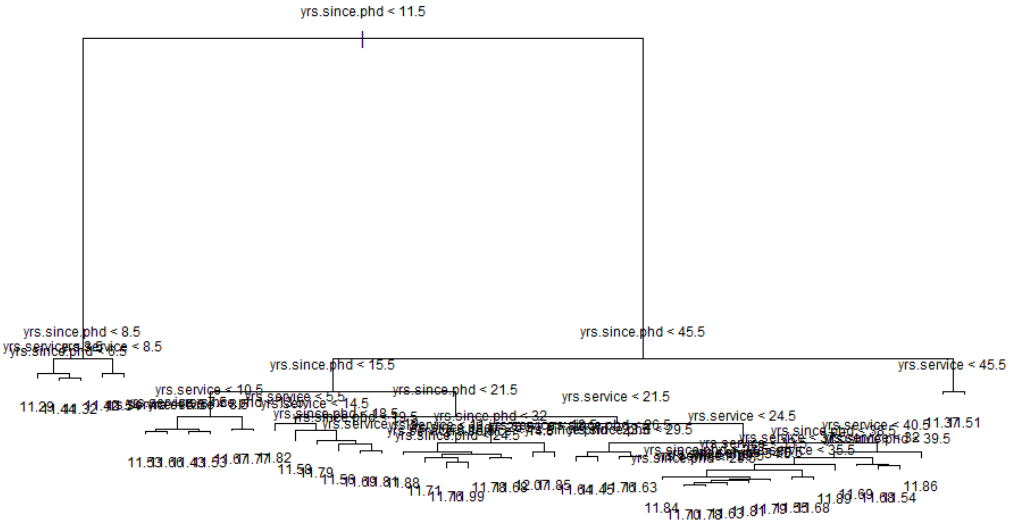
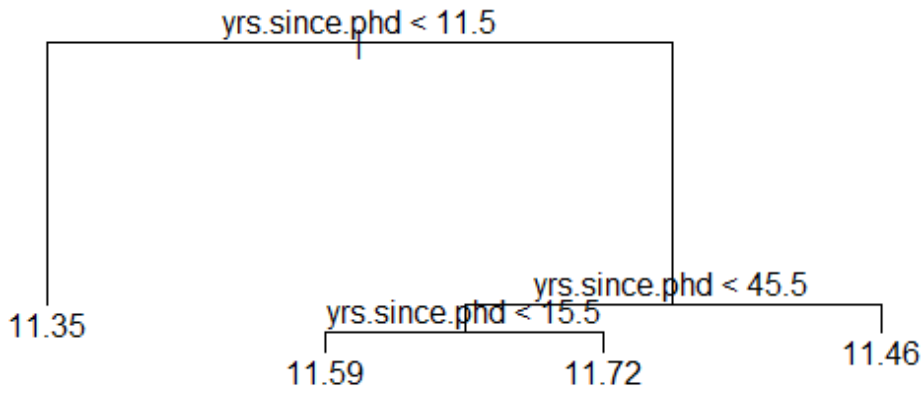


FIGURE 7 – *tree.model* après élagation



À vous !

a) Qu'avons-nous changé dans notre nouvel arbre *tree.model2* ?
b) Définissez le paramètre modifié.
c) Comparez les deux arbres.
d) Quel est l'arbre le plus performant ?
e) Commentez l'élagation de *tree.model*.
f) Élaguez *tree.model2*.
g) Quel est le meilleur arbre ?
h) Comment seront évaluées les prédictions de notre arbre ?

2 Les arbres de classification

Lorsque les variables explicatives sont de qualitatives, vous utiliserez un arbre de classification. Avec des variables qualitatives, le nœud séparera les modalités de cette variables en deux ensembles disjoints. Si nous avons n modalités possible, pour une variable qualitative X, il y aura alors $2^{n-1} - 1$ conditions de séparations possible sur cette variable. L'algorithme utilisé ici ($CART$) va tester toutes les possibilités et chercher à maximiser l'inertie intra-classe et minimiser l'inertie interclasse. Si notre variable cible est-elle aussi qualitative, les classes afficheront probabilité d'avoir une (pour une variable binaire) ou plusieurs modalités. Comme pour les arbres de régression, l'algorithme sélectionne les variables qui ont le meilleur critère de séparation, et s'arrête lorsque la séparation n'est plus utile.

2.1 Chargement des données

Dans cet exemple nous allons observer les passagers du Titanic et identifier les variables qui joué un rôle dans leurs probabilités de survie. Vous pouvez télécharger les données sur `https://vincentarelbundock.github.io/Rdatasets/datasets.html`.
Nous avons 1316 passagers avec les informations suivantes :

- X qui est l'ID du passager
- class : la classe du passager (1ier,2ieme,3ieme)
- age : deux catégories d'age : adulte ou enfant
- sex : si le passager est un homme ou une femme
- survived : si le passager a survécu (yes) ou pas (no)

FIGURE 8 – Un modèle prédictif ?

source :`http://kingsunraj.blogspot.fr`

```
#charger les donnees - attention aux options
> data <- read.csv2('C:/Users/claey/Documents/cour/My TD/TD5/
    titanic.csv', sep =',', header = TRUE)
> titanic <- as.data.frame(data)

###Summary
> summary(titanic)
       X                class         age          sex
          survived
 Min.   :   1.0   1st class:325   adults:1207   man   :869   no :817
 1st Qu.: 329.8   2nd class:285   child : 109   women:447   yes:499
```

```
 Median : 658.5   3rd class:706
 Mean   : 658.5
 3rd Qu.: 987.2
 Max.   :1316.0
> View(data)
```

À vous !

a) Commentez les données.

b) Supprimez la variable X qui n'est pas une donnée exploitable.

c) Quelles sont les variables binaires et quelles sont les variables à plus de deux modalités ?

2.2 Création de l'arbre

Des arbres de classification et de régression peuvent être générés à travers le package *rpart*. La fonction *rpart()* utilise le critère de Gini pour choisir la meilleur séparation de ces nœud (ici via l'algoritme CART). Le coefficient de Gini est équivalent à l'écart moyen relatif (l'écart moyen divisé par la moyenne pour le mettre à l'échelle) : il s'agit donc bien d'une mesure de dispersion de valeurs numériques [2]. Pour conserver la distribution des classes, on utilise l'indice de Gini. Plus les classes sont uniformément distribuées dans un nœud, plus l'indice de Gini est élevé ; plus un nœud est pur, plus son indice de Gini est bas. Dans le cas de deux classes, l'indice de Gini va de 0 (nœud pur) à 0,5 (mélange maximal). Intuitivement, l'indice de Gini mesure la probabilité que deux individus choisis aléatoirement (avec remise) dans un nœud, appartiennent à deux classes différentes.
Á chaque séparation, l'arbre va se construire en cherchant à donner la plus grande pureté à ses nœuds fils, et donc baisser l'indice de Gini.
Le package *party* fournit des arbres de régression non paramétriques pour variables nominales, ordinales, numériques, et des réponses multivariées. La fonction **ctree** s'appuie sur l'inférence conditionnelle (ici via l'algoritme CTREE), et explore pour chaque sous-ensembles, la variance entre les deux inférences conditionnelles. Pour comparer les deux inférences, la fonction utilise un la correction de Bonferroni pour réaliser des comparaisons multiples (on prendra $\alpha = 0,5$). Les coefficients de corrélation permettent de donner une mesure synthétique de l'intensité de la relation entre deux ensemble et de son sens lorsque cette relation est monotone. Le coefficient de corrélation de Pearson permet d'analyser les relations linéaires et le coefficient de corrélation de Spearman les relations non-linéaires monotones (Il existe d'autres coefficients pour les relations non-linéaires et non-monotones). On sélectionnera la variable (et ces deux sous-ensemble de modalité) qui a aboutie à la p-value la plus faible suite au test.

```
> ###Installer et charger
> library(rpart)
```

2. 0 signifie l'égalité parfaite et 1 signifie une inégalité parfaite

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

> str(titanic)
'data.frame':   1316 obs. of  5 variables:
 $ X       : int  1 2 3 4 5 6 7 8 9 10 ...
 $ class   : Factor w/ 3 levels "1st class","2nd class",..: 1 1 1 1
     1 1 1 1 1 1 ...
 $ age     : Factor w/ 2 levels "adults","child": 1 1 1 1 1 1 1 1 1
     1 ...
 $ sex     : Factor w/ 2 levels "man","women": 1 1 1 1 1 1 1 1 1 1
    ...
 $ survived: Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...

> myFormula <- survived ~  class  + age  #

> ##CART##

> fit <- rpart(myFormula, method="class", data=titanic)

> printcp(fit) # display the results

Classification tree:
rpart(formula = myFormula, data = titanic, method = "class")

Variables actually used in tree construction:
[1] age   class

Root node error: 499/1316 = 0.37918

n= 1316

        CP nsplit rel error  xerror     xstd
1 0.162325      0   1.00000 1.00000 0.035272
2 0.024048      1   0.83768 0.87776 0.034258
3 0.010000      3   0.78958 0.78958 0.033295


> plotcp(fit) # visualize cross-validation result (Figure 8)
> summary(fit) # detailed summary of splits


Call:
rpart(formula = myFormula, data = titanic, method = "class")
  n= 1316

         CP nsplit rel error    xerror       xstd
1 0.1623246      0 1.0000000 1.0000000 0.03527222
2 0.0240481      1 0.8376754 0.8777555 0.03425751
3 0.0100000      3 0.7895792 0.7895792 0.03329546

Variable importance
class   age
   78    22
```

```
Node number 1: 1316 observations,    complexity param=0.1623246
  predicted class=no   expected loss=0.3791793  P(node) =1
    class counts:   817   499
   probabilities: 0.621 0.379
  left son=2 (991 obs) right son=3 (325 obs)
  Primary splits:
      class splits as  RLL, improve=51.996280, (0 missing)
      age   splits as  LR,  improve= 4.912016, (0 missing)

Node number 2: 991 observations,    complexity param=0.0240481
  predicted class=no   expected loss=0.2986882  P(node) =0.7530395
    class counts:   695   296
   probabilities: 0.701 0.299
  left son=4 (706 obs) right son=5 (285 obs)
  Primary splits:
      class splits as  -RL, improve=10.645240, (0 missing)
      age   splits as  LR,  improve= 8.872885, (0 missing)

Node number 3: 325 observations
  predicted class=yes  expected loss=0.3753846  P(node) =0.2469605
    class counts:   122   203
   probabilities: 0.375 0.625

Node number 4: 706 observations
  predicted class=no   expected loss=0.2521246  P(node) =0.5364742
    class counts:   528   178
   probabilities: 0.748 0.252

Node number 5: 285 observations,    complexity param=0.0240481
  predicted class=no   expected loss=0.4140351  P(node) =0.2165653
    class counts:   167   118
   probabilities: 0.586 0.414
  left son=10 (261 obs) right son=11 (24 obs)
  Primary splits:
      age splits as  LR, improve=17.99653, (0 missing)

Node number 10: 261 observations
  predicted class=no   expected loss=0.3601533  P(node) =0.1983283
    class counts:   167    94
   probabilities: 0.640 0.360

Node number 11: 24 observations
  predicted class=yes  expected loss=0  P(node) =0.01823708
    class counts:     0    24
   probabilities: 0.000 1.000


> # plot tree Figure  9
> plot(fit, uniform=TRUE)
> text(fit,use.n=TRUE, all=TRUE, cex=.7)
>
> library(rattle)
> library(rpart.plot)
```

```
> library(RColorBrewer)
>
> #On ameliore le graphique qui n'est pas tres lisible : Figure 10
> fancyRpartPlot(fit)

> ##CTREE##
> titanic_tree <- ctree(myFormula, data=titanic)
> print(titanic_tree)

         Conditional inference tree with 4 terminal nodes

Response:  survived
Inputs:  class, age
Number of observations:  1316

1) class == {1st class}; criterion = 1, statistic = 132.951
  2)*  weights = 325
1) class == {2nd class, 3rd class}
  3) class == {2nd class}; criterion = 1, statistic = 25.384
    4) age == {adults}; criterion = 1, statistic = 36.959
      5)*  weights = 261
    4) age == {child}
      6)*  weights = 24
  3) class == {3rd class}
    7)*  weights = 706

> ## Figure 11
> plot(titanic_tree)
```

FIGURE 9 – Résultat de la cross-vall sur *fit* (Méthode CART)

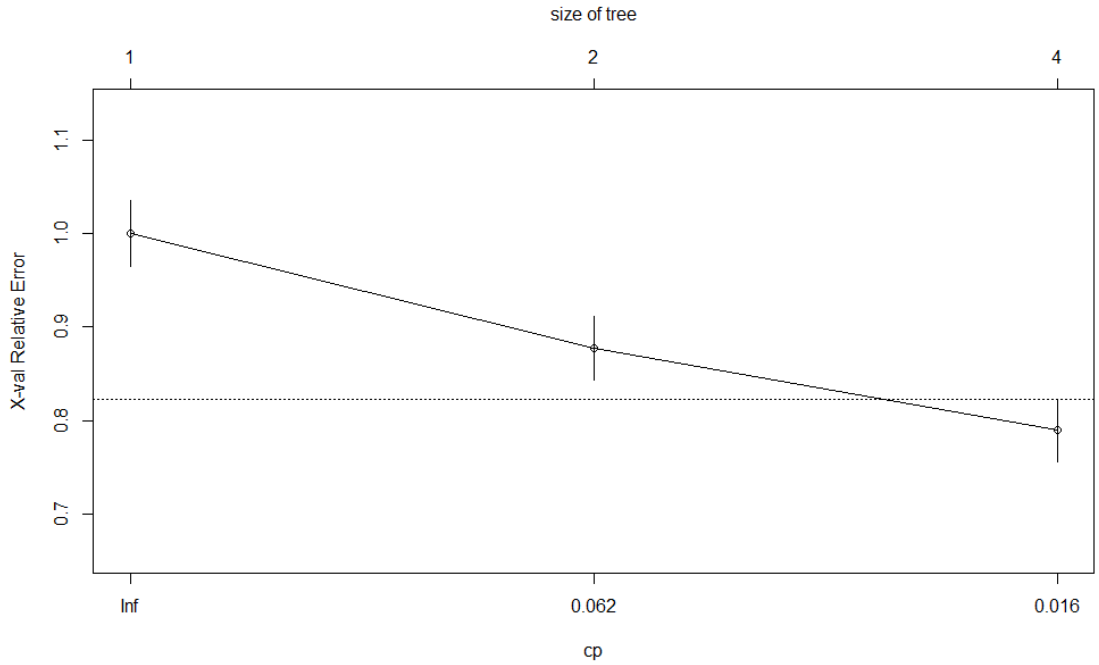


FIGURE 10 – *fit* (Méthode CART)

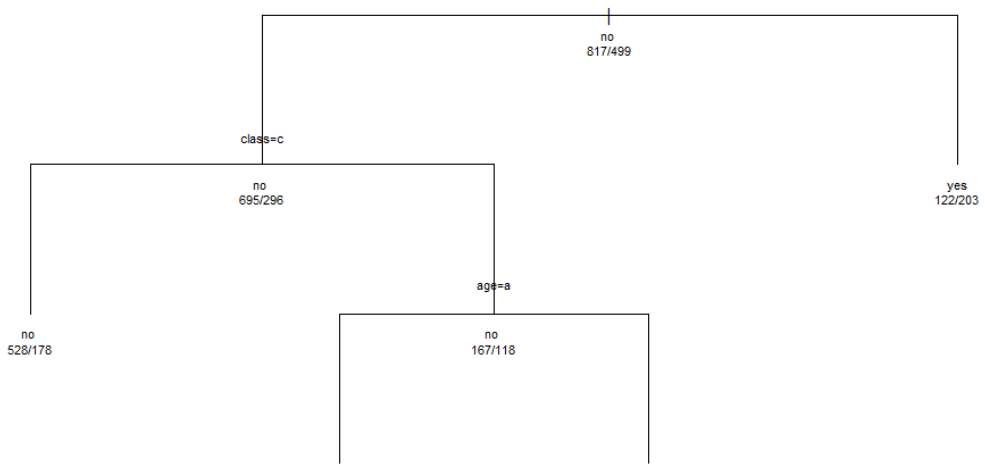


FIGURE 11 – *fit* avec *fancyRpartPlot()* (Méthode CART)

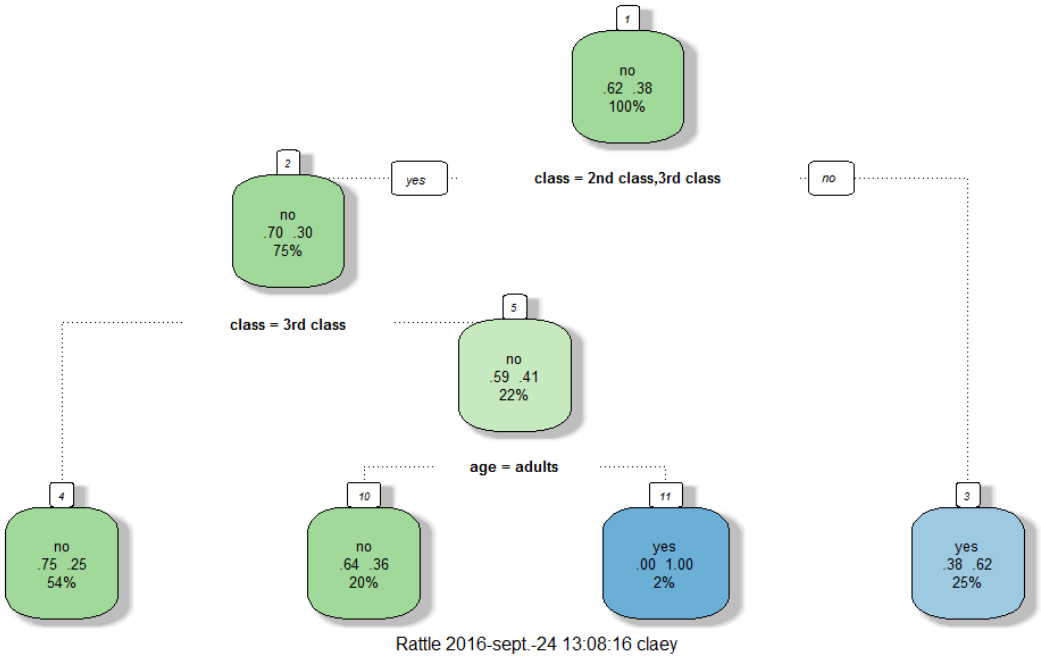
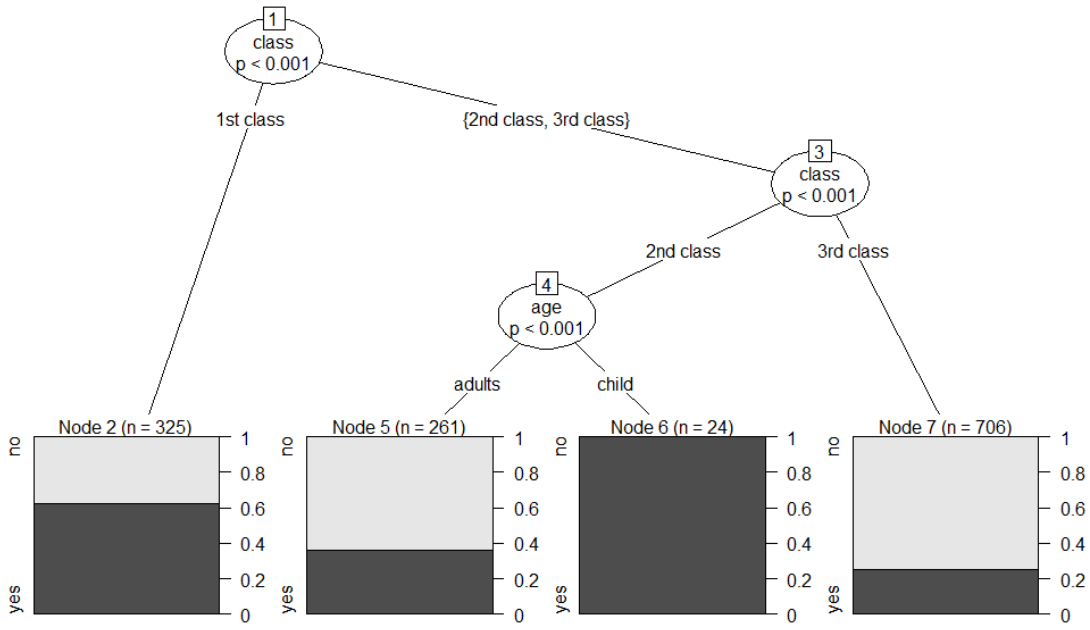


FIGURE 12 – titanic$_t ree(MthodeCTREE)$



À vous !

a) Notez une propriété de l'arbre CARD sur la soustraction *Gini(nœud père)* - *Gini(nœud fils)*.

b) Comparez les deux arbres (*fit* et *titanic_ tree*.

c) Est-il nécessaire d'élaguer l'arbre *ftitanic_ tree* (regardez la doc de la fonction *ctree()*) ? Pourquoi ?

d) Quelle est la différence entre CTREE et CART

e) Réalisez deux arbres : *fit2* et *titanic_ tree2*, *fit2* avec la méthode CART et *titanic_ tree2* avec la méthode CTREE, mais en intégrant la variable sexe dans les variables explicatives.

2.3 Amélioration de l'arbre

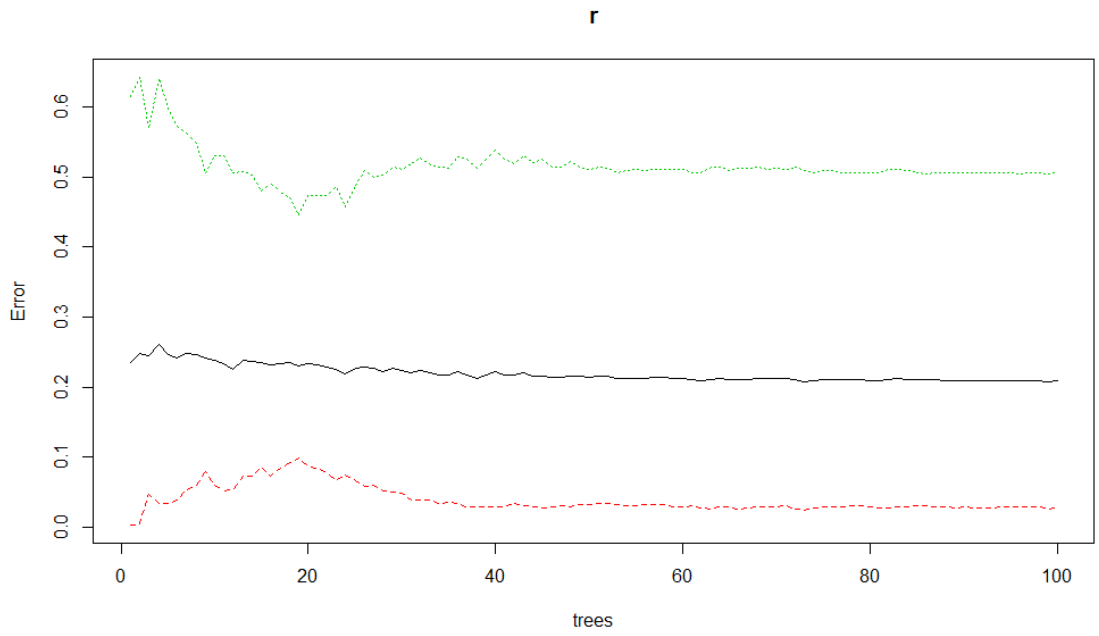
Vous allez utiliser la fonction *predict()* sur votre arbre *titanic_ tree2* et observer ses résultats.
Jusqu'à présent a été abordé la méthode de partitions récursives. Il y a des avantages et des inconvénients à cette méthode. Un inconvénient notable est la sensibilité de ces arbres uniques à l'ordre des prédicteurs, en calculant à chaque itération, un sous ensemble d'arbres partiellement indépendants. Intuitivement, une fois que l'arbre à *spliter*, on ne peut plus utiliser la variables à l'origine du *split*. Une des solution pour remédier à ce problème est l'application de l'algorithme *random forest*.
On tente de comparer notre modèle en appliquant l'algorithme *random forest*. Les forêts d'arbres décisionnels (ou forêts aléatoires de l'anglais random forest classifier)

est un algorithme qui combine les concepts de sous-espaces aléatoires et de bagging [3]. L'algorithme des forêts d'arbres décisionnels effectue un apprentissage sur de multiples arbres de décision entraînés sur des sous-ensembles de données légèrement différents.

```
> predictions <- predict(titanic_tree, titanic)
> table(titanic$survived, predictions)
     predictions
       no yes
  no  695 122
  yes 272 227


> predictions <- predict(titanic_tree2, titanic)
> table(titanic$survived, predictions)
     predictions
       no yes
  no  800  17
  yes 249 250

> library(randomForest)
> r <- randomForest(survived ~., data=titanic, importance=TRUE, do.
   trace=100, ntree=100)
ntree      OOB      1      2
  100:  20.90%  2.69% 50.70%
> print(r)

Call:
 randomForest(formula = survived ~ ., data = titanic, importance =
    TRUE,      do.trace = 100, ntree = 100)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 1

        OOB estimate of  error rate: 20.9%
Confusion matrix:
     no yes class.error
no  795  22  0.02692778
yes 253 246  0.50701403
> plot(r)
> predictions <- predict(r, titanic)
> table(titanic$survived, predictions)
     predictions
       no yes
  no  800  17
  yes 249 250
```

3. Le bootstrap (ou bagging) est une technique évaluant plusieurs échantillons, mais ne crée de « nouveaux échantillons » que par tirage dans l'ancien, avec remise à partir de l'échantillon initial (on parle de ré-échantillonnage).

FIGURE 13 – Résultat d'erreurs du *random forest*



À vous !

a) Comparez les résultats de la fonction *predict()* sur *titanic_tree* et *titanic_tree2*.

b) Comparez les résultats de la fonction *predct()* sur *titanic_tree2* et *r*.

c) Interprétez la figure 12.

d) La fonction *rpart()* et *ctree()* intègrent-elles la fonction *randomForest()* ?