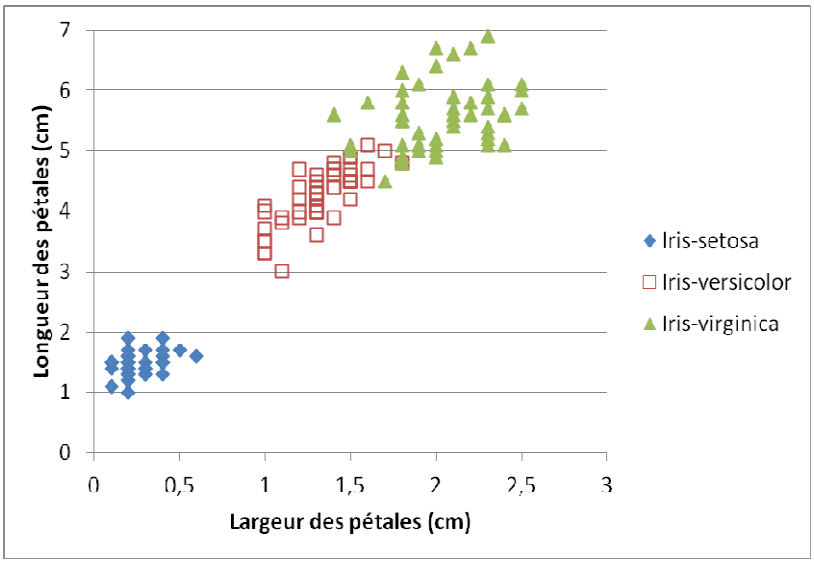
CY TECH	ING2-GI : MACHINE LEARNING ARBRE DE DECISION ET RANDOM FOREST
Rédigé par : Astrid Jourdan	A l'intention de : Elèves d'ING2-GI
Durée : 6h	Dernière modification : 21/09/2022

Exercice 1

Considérons le fameux jeu de données « Iris » (http://archive.ics.uci.edu/ml/datasets/Iris) représenté ci-dessous. Il s'agit de trois types d'Iris caractérisés entre autre par la longueur et la largeur des pétales.



1) Représentez sur le graphique la règle : *Si la longueur des pétales est inférieure à 2 alors l'iris est de type Setosa*.
2) Dans le cas où la longueur des pétales est supérieure à 2, ajoutez sur le graphique la règle : *Si la largeur des pétales est supérieure à 1,7 alors l'iris est de type Virginia, sinon l'iris est de type Versicolor*.
3) Représenter l'arbre correspondant à ces règles.
4) Refaire la même chose en inversant l'ordre de longueur et largeur.

Exercice 2

Considérons l'exemple très simple sur le football.

X_1=Match à domicile ?	X_2=Balance positive ?	X_3=Mauvaises conditions climatiques ?	X_4=Match précédent gagné ?	Y=Match gagné
V	V	F	F	V
F	F	V	V	V
V	V	V	F	V
V	V	F	V	V
F	V	V	V	F
F	F	V	F	F
V	F	F	V	F
V	F	V	F	F

Quelle variable sera choisie à la racine de l'arbre si l'on souhaite maximiser le gain à l'aide de l'entropie ?

Exercice 3

On considère des images en noir et blanc codées sur 4 pixels. Chaque image est donc codée par un élément $(x_1,x_2,x_3,x_4)\in\{0,1\}^4$, où les pixels noirs sont notés 1 et les pixels blancs sont notés 0 et sont numérotés dans l'ordre

x_1	x_2
x_3	x_4

On dispose de la base d'apprentissage ci-dessous pour laquelle les images ont été réparties selon deux classes.

On souhaite construire un arbre de décision sur cet échantillon avec comme variables explicatives (attributs), x_1, x_2, x_3 et x_4.

1) Ecrire la base d'apprentissage sous la forme d'un tableau
2) Quelle variable sera choisie à la racine de l'arbre si l'on souhaite maximiser le gain à l'aide de l'indice de Gini ?
3) Représenter ensuite l'arbre T avec un partage des nœuds suivant les variables dans l'ordre suivant : X_2, X_3, X_1.
4) Proposer un arbre élagué. Calculer l'erreur d'ajustement de chacun des arbres ?
5) On considère l'ensemble de test suivant

Calculer l'erreur de prévision pour les deux arbres.

Exercice 4

L'objectif de ce tutoriel est d'apprendre à construire un arbre de décision avec le package `rpart` du logiciel R. Pour la représentation graphique, nous utiliserons le package `rpart.plot`. Le jeu de données `ptitanic` concerne des passagers du célèbre Titanic. Il s'agit de construire un modèle permettant de prédire si un passager est « survivant » ou « mort » en fonction des autres variables, classe du ticket, âge, sexe,… .

```
library(rpart)
library(rpart.plot)
data(ptitanic)
str(ptitanic)
```

On vérifie la nature des variables notamment que les variables qualitatives ne sont pas considérées comme de variables numériques.

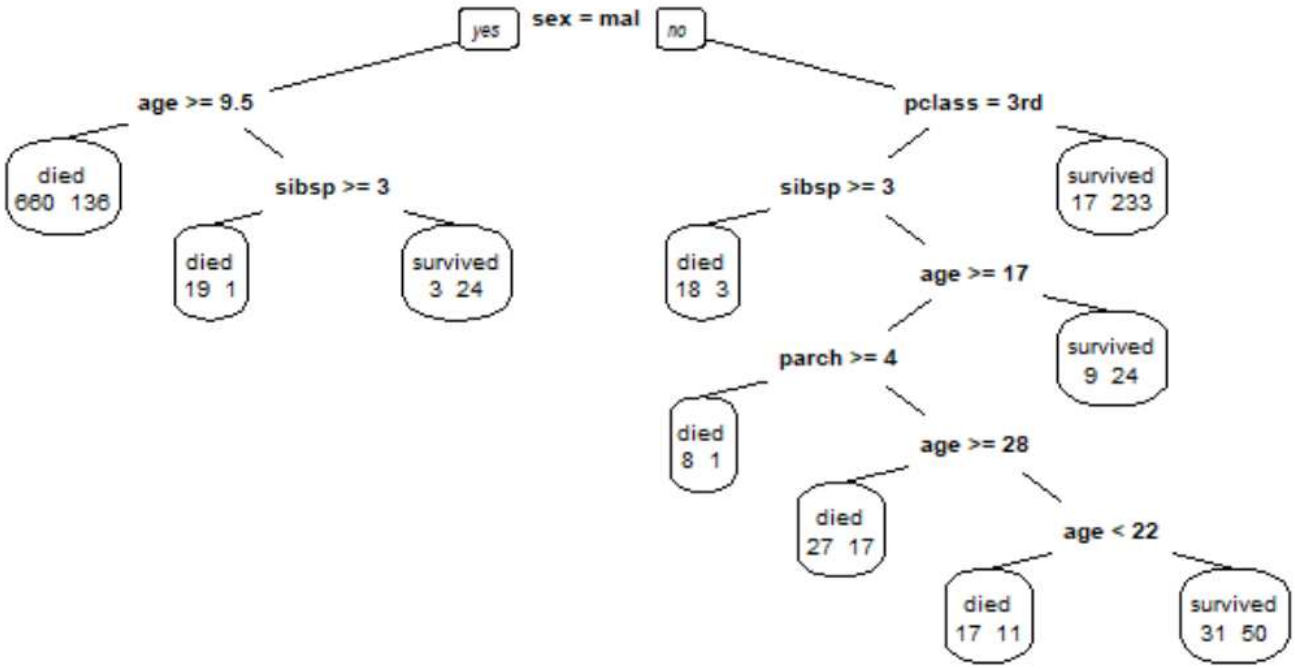
```
> str(ptitanic)
'data.frame':   1309 obs. of  6 variables:
 $ pclass  : Factor w/ 3 levels "1st","2nd","3rd": 1 1 1 1 1 1 1 1 1 1 ...
 $ survived: Factor w/ 2 levels "died","survived": 2 2 1 1 1 2 2 1 2 1 ...
 $ sex     : Factor w/ 2 levels "female","male": 1 2 1 2 1 2 1 2 1 2 ...
 $ age     : 'labelled' num  29 0.917 2 30 25 ...
  ..- attr(*, "units")= chr "Year"
  ..- attr(*, "label")= chr "Age"
 $ sibsp   : 'labelled' int  0 1 1 1 1 0 1 0 2 0 ...
  ..- attr(*, "label")= chr "Number of Siblings/Spouses Aboard"
 $ parch   : 'labelled' int  0 2 2 2 2 0 0 0 0 0 ...
  ..- attr(*, "label")= chr "Number of Parents/Children Aboard"
> 
```

La fonction pour construire un arbre de décision porte le même nom que le package,

`rpart(formula,data)`

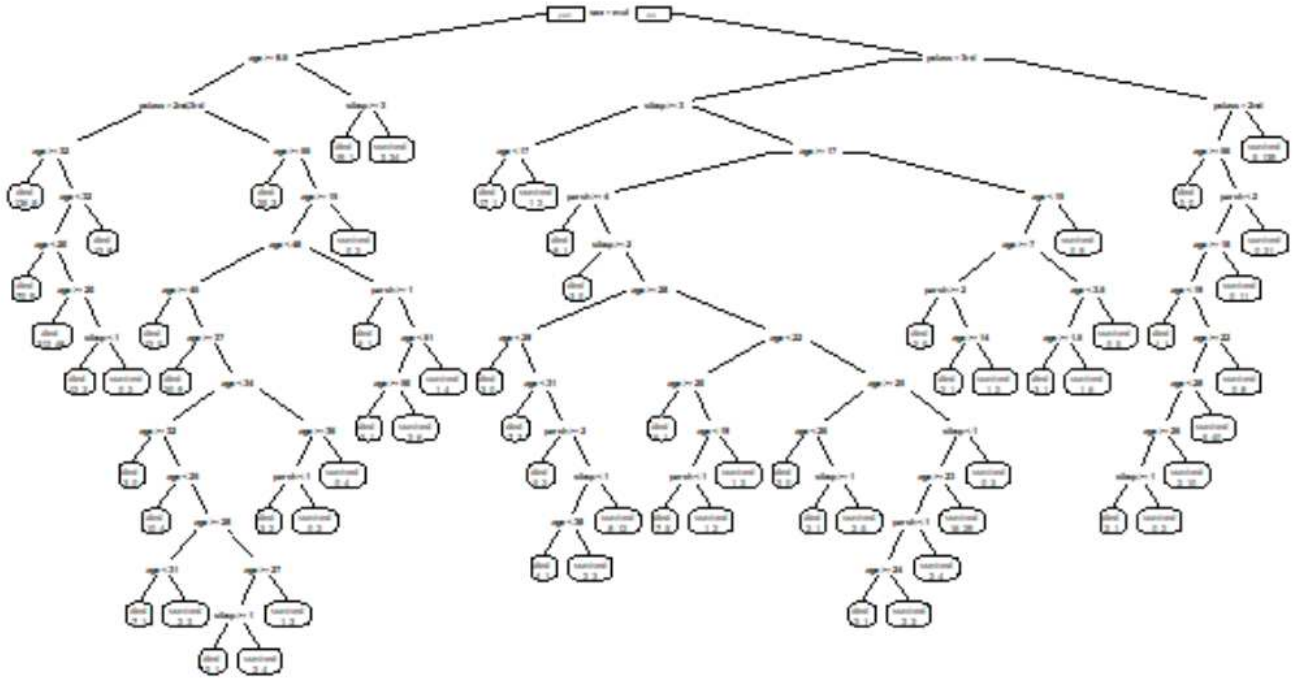
Cette fonction utilise un paramétrage par défaut permettant en autre un élagage automatique de l'arbre. Par défaut elle fonctionne avec l'indice de Gini mais il est possible de calculer le gain avec l'entropie. La fonction `prp` permet de tracer le graphique de l'arbre, l'argument `extra=1` affiche la répartition des classes dans chaque feuille.

```
Arbre=rpart(survived~.,ptitanic)
prp(Arbre,extra=1)
```



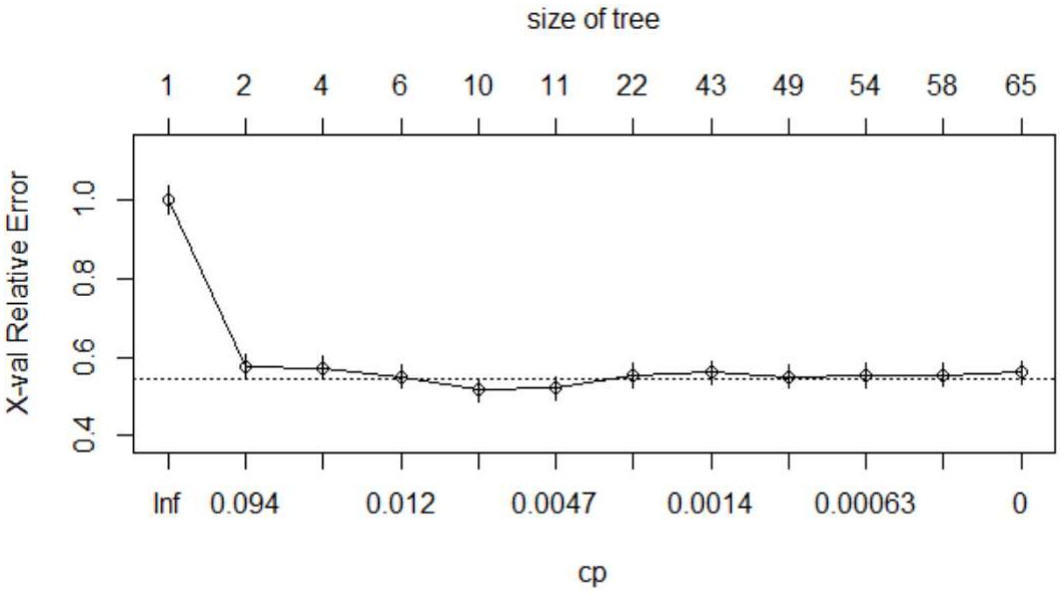
Il est possible de ne pas utiliser le paramétrage par défaut de R et choisir son propre élagage pour l'arbre. Pour cela, on donne une liste de paramètres avec rpart.control. Ci-dessous, on va forcer le paramétrage de façon à obtenir un arbre complexe (`cp=0`) et l'algorithme arrête de faire des split quand le nœud contient moins de 5 observations (`minsplit=5`).

```
Arbre=rpart(survived~.,ptitanic,control=rpart.control(minsplit=5,cp=0))
prp(Arbre,extra=1)
```



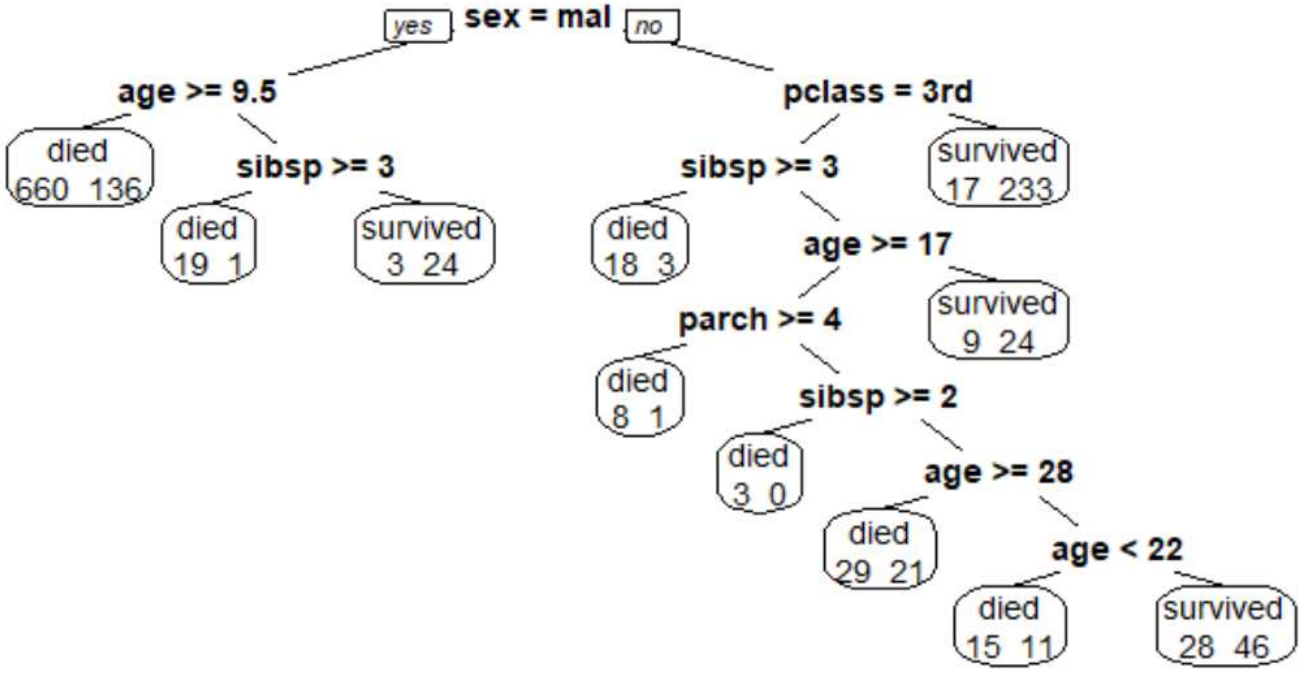
Dans cet arbre, on note qu'il y a des feuilles avec uniquement 2 ou 3 observations. Il y a donc du sur-apprentissage. Afin de savoir à quel niveau élaguer l'arbre, on trace le graphique de l'erreur de prévision en fonction de la complexité du modèle à l'aide de la fonction `plotcp`. A noter que nous n'avons pas construit de base de validation (test) sur notre de jeu de données mais la fonction `plotcp` utilise une méthode de validation croisée.

```
plotcp(Arbre)
```



On remarque que l'erreur décroit jusqu'à environ cp=0.0047 puis remonte. Cela signifie qu'au-delà de cette complexité, l'arbre construit des branches pour des cas particuliers. Il faut donc élaguer l'arbre à ce niveau avec la fonction `prune`.

```
ArbreElague=prune(Arbre,cp=0.0047)
prp(ArbreElague,extra=1)
```



Une fois l'arbre construit, on utilise la fonction `predict.rpart` (ou plus simplement `predict`),

```
predict(model, new data)
```

Cette fonction retourne la probabilité de chaque classe pour toutes les nouvelles observations. Si on souhaite la prévision de la classe, il faut ajouter l'argument `type='classe'`.

```
# Prévision des 5ères lignes du jeu de données en termes de probabilités
predict(ArbreElague,ptitanic[1:5,])
```

```
> predict(ArbreElague,ptitanic[1:5,])
       died  survived
1 0.0680000 0.9320000
2 0.1111111 0.8888889
3 0.0680000 0.9320000
4 0.8291457 0.1708543
5 0.0680000 0.9320000
>
```

```
# Prévision des 5ères lignes du jeu de données en termes de classes
predict(ArbreElague,ptitanic[1:5,] ,type="class")
```

```
> predict(ArbreElague,ptitanic[1:5,],type="class")
       1        2        3        4        5
survived survived survived     died survived
Levels: died survived
>
```

Pour déterminer les performances du modèle, il faut calculer la matrice de confusion entre le vecteur des vraies classes de la variable cible (`ptitanic$survived`) et les classes prédites par le modèle.

```
prevision=predict(ArbreElague,ptitanic,type="class")
MatConf=table(ptitanic$survived,prevision)
print(MatConf)
```

```
          prevision
           died survived
died        752       57
survived    173      327
> 
```

On en déduit le taux de bien classés global (somme de la diagonale de la matrice de confusion/nombre d'observations) et le taux de bien classés par classe (i.e. par ligne de la matrice de confusion).

```
tb=sum(diag(MatConf))/sum(MatConf)
print(paste("Le taux global de bien classés est ",tb))

tb.classes=diag(MatConf)/apply(MatConf,1,sum)
print(paste("Les taux de bien classés par classe sont ",tb.classes))
```

```
> tb=sum(diag(MatConf))/sum(MatConf)
> print(paste("Le taux global de bien classés est ",tb))
[1] "Le taux global de bien classés est  0.824293353705118"
> tb.classes=diag(MatConf)/apply(MatConf,1,sum)
> print(paste("Les taux de bien classés par classe sont ",tb.classes))
[1] "Les taux de bien classés par classe sont  0.929542645241038"
[2] "Les taux de bien classés par classe sont  0.654"
> 
```

On note que la classe « survivant » est moins bien apprise que la classe « mort ». Cela est peut-être dû à une sous-représentation de la classe « survivant » dans le jeu de données qui a servi à construire l'arbre.

```
table(ptitanic$survived)
```

```
> table(ptitanic$survived)

    died survived
     809      500
> 
```

Exercice 5

Soient les deux jeux de données simulées : Test_Classif_dpt.txt et Test_Classif_Correl.txt

- Représenter le nuage de points avec la couleur des classes pour les deux jeux. Un arbre de décision peut-il séparer ces classes ?
- Comparer la complexité des arbres construits dans chacun des cas.

Exercice 6

Pour ajuster une forêt aléatoire, on utilise le package R `randomForest`. Pour s'entraîner, nous allons utiliser le jeu de données « iris ».

```
library("randomForest")
data("iris")
```

La fonction pour construire une forêt aléatoire porte le même nom que le package,

`randomForest(formula,data,ntree,mtry,importance,…)`

On retrouve les arguments classiques pour construire un modèle en R (`formula` et `data`). Les arguments `ntree` et `mtry` sont des hyperparamètres de la forêt,

- `ntree` : nombre d'arbres (défaut est de 500),
- `mtry` : nombre de variables à prendre en compte pour chaque nœud d'arbre (défaut ~√p).

L'argument `importance` doit être fixée `TRUE` si on veut avoir en retour la variable importance indiquant l'importance de chaque variable.

Il existe encore beaucoup d'arguments possibles pour cette fonction, notamment des hyperparamètres de la forêt (cf. help R de la fonction).

```
modele=randomForest(Species~.,iris,importance=T)
show(modele)
```

```
Call:
 randomForest(formula = Species ~ ., data = iris, importance = T)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 4%
Confusion matrix:
           setosa versicolor virginica class.error
setosa         50          0         0        0.00
versicolor      0         47         3        0.06
virginica       0          3        47        0.06
```

Le modèle (type of random forest) est de type classification car la variable cible Species est qualitative (par opposition au type regression). La forêt a été construite avec 500 arbres. Le nombre de variables choisies à chaque split est 2 car le nombre de variables explicatives est 4.

L'erreur oob est estimée à 4%. Pour chaque élément de la base d'apprentissage, il est possible de savoir dans combien de sous-échantillons bootstrap (arbres) il n'apparait pas. Dans l'exemple ci-dessous, l'iris 1 est « out of bag » pour 192 arbres parmi les 500.

```
modele$oob.times
```

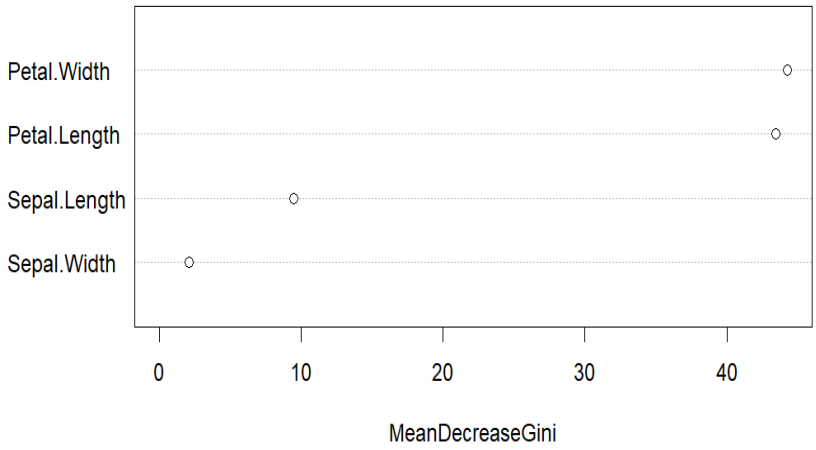
```
  [1] 192 177 175 183 173 186 188 197 195 176 182 179 174 184 177
 [16] 203 184 163 173 194 183 184 187 179 191 173 186 193 185 169
 [31] 180 163 190 184 192 179 178 188 182 186 187 197 172 173 191
 [46] 178 178 174 180 183 168 165 171 188 189 167 201 197 191 179
 [61] 176 172 197 168 199 173 183 180 193 195 192 186 165 180 175
 [76] 178 201 176 190 170 163 190 193 198 190 198 205 185 177 192
 [91] 192 178 175 176 205 168 204 198 201 170 190 166 166 201 163
[106] 178 179 196 188 188 179 189 184 188 199 196 189 184 167 193
[121] 188 161 178 186 196 183 181 160 198 181 190 196 190 183 192
[136] 181 175 182 202 167 197 195 196 177 173 170 194 182 168 182
```

La matrice de confusion sur la base d'apprentissage est automatiquement calculée avec le taux d'erreur pour chaque classe.

L'importance des variables permet de savoir quelles variables ont été discriminantes dans les split et quelles variables n'ont pas eu beaucoup d'impact dans la construction des arbres.

```
modele$importance
varImpPlot(modele)
```

```
             MeanDecreaseGini
Sepal.Length         9.491089
Sepal.Width          2.084927
Petal.Length        43.424933
Petal.Width         44.235533
```



Comme pour tout modèle R, la prévision d'un nouvel élément se fait à l'aide de la fonction predict,

```
predict(modele, newdata, type)
```

L'argument `type` est fixé à `"prob"` si on souhaite avoir la probabilité de chaque classe.

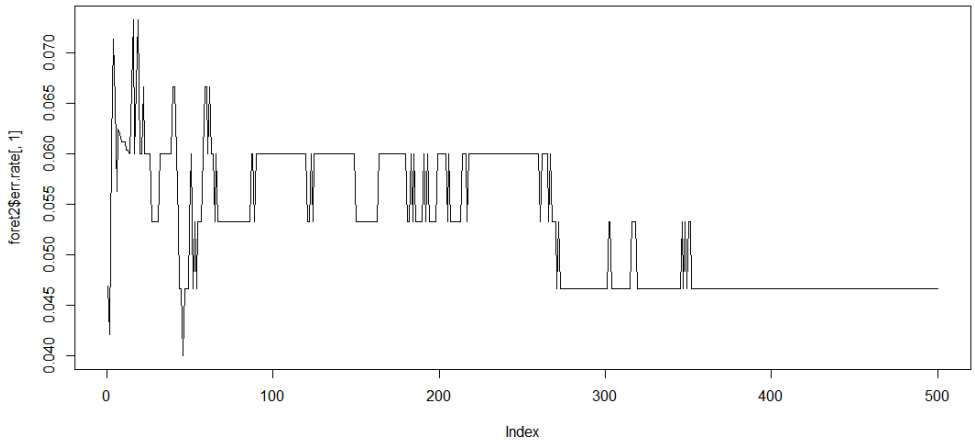
```
Newiris=as.data.frame(t(c(4,3,1,2))) # Il faut un data.frame dans un predict # t=transpose
names(Newiris)=names(iris)[1:4]     # Il faut les mêmes noms que ceux de la base d'apprentissage
predict(modele,Newiris)
predict(modele,Newiris,type="prob")
```

La qualité du modèle va dépendre de son hyperparamétrage. Nous avons vu les hyperparamètres `ntree` et `mtry` mais on peut aussi faire varier `sampsize` qui fixe la taille de l'échantillon "in bag", ou `maxnodes`, qui limite le nombre de nœuds de chaque arbre. Tester plusieurs types de forêts demande donc beaucoup de temps de calcul mais les calculs peuvent se faire en parallèle sur plusieurs nœuds. Ensuite, soit on récupère ce qui nous semble être le meilleur modèle, soit on combine tous les prédicteurs obtenus grâce à la fonction `combine`.

```
foret1=randomForest(Species ~ ., iris, ntree = 100)
foret2=randomForest(Species ~ ., iris, ntree = 500)
modele=combine(foret1, foret2)
```

Une solution pour optimiser le temps de calcul est d'entrainer une forêt de petite taille est afficher l'évolution de son erreur en fonction du nombre d'arbres.

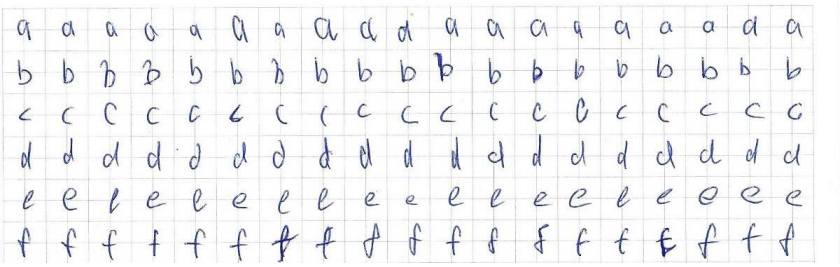
```
plot(foret2$err.rate[,1], type="l")
```



Si cette fonction n'a pas atteint un seuil d'erreur acceptable et si elle continue à décroitre, alors, il faut construire une deuxième forêt et l'agréger à la première avec la fonction `combine`. On procède de façon itérative jusqu'à atteindre la convergence de l'erreur. Sur le jeu de données Iris ci-dessus, on constate que le taux d'erreur est très bas dès le début (en fait il est déjà bas pour un seul arbre) et qu'il converge vers 0.047 aux alentours de 300 arbres. Il était donc inutile de construire une forêt avec 500 arbres, d'où l'intérêt de procéder de façon itérative.

Exercice 7

Utiliser la méthode des forêts aléatoires pour faire de la reconnaissance de caractères avec le jeu de données : http://archive.ics.uci.edu/ml/datasets/Letter+Recognition



(a) Créer une base d'apprentissage (2/3) et une base test (1/3).
(b) Construire le modèle sur la base d'apprentissage. Analyser le modèle obtenu.
(c) Quelles sont les variables discriminantes ou les variables ayant peu d'impact ?
(d) Calculer l'erreur de prévision et la comparer avec l'erreur oob et l'erreur d'ajustement. Y-a-t-il sur-ajustement ?
(e) Faire varier les paramètres de l'arbre (ntree, mtry,…). Comparer les erreurs de prévision et en déduire quels paramètres à une influence sur la qualité du modèle.
(f) Comparer avec un arbre de décision. Faire une analyse quantitative (précision) et qualitative (interprétabilité).