UNIVERSITÉ PIERRE ET MARIE CURIE

INSTITUT DE STATISTIQUE DE
L'UNIVERSITÉ PARIS 6

TROISIÈME ANNÉE ISUP -
MASTER 2 DE STATISTIQUES - MAJOR DATA SCIENCE

RÉSEAUX DE NEURONES ARTIFICIELS SOUS LE LOGICIEL R

Professeure :
Annick VALIBOUZE

Année 2017-2018

Le logiciel R est un logiciel de statistiques libre. Il est constitué de packages permettant de réaliser des études statistiques.

Le package nnet est utilisé pour le Percepteur Multi Couches. Nous allons le comparer avec diverses méthodes.

>

S3 method for class 'formula'

nnet(formula, data, weights, ..., subset, na.action, contrasts = NULL)

Default S3 method :

nnet(x, y, weights, size, Wts, mask, linout = FALSE, entropy = FALSE, softmax = FALSE, censored = FALSE, skip = FALSE, rang = 0.7, decay = 0, maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000, abstol = 1.0e-4, reltol = 1.0e-8, ...)

1. Interpolation - Comparaison de nnet avec le modèle polynomial

```
> # PMC avec sin(x) bruit\'ees et comparaison avec le mod\'ele polynomial

> library(nnet)
> library(MASS)
>  # Les donn\'ees
>  x <-sort(10*runif(50))
> y <- sin(x)+0.2*rnorm(x)
> data <- data.frame(x,y)
> # Construction du PMC linout=TRUE pour \^etre entre 0 et 1
> nn <- nnet(x,y,size=4,maxit=100,linout=TRUE)
> # weights :  13
  initial  value 33.754687
  iter  10 value 22.612081
  iter  20 value 15.456868
  iter  30 value 7.875047
  iter  40 value 7.225153
  iter  50 value 6.314550
  iter  60 value 4.749729
  iter  70 value 3.729921
  iter  80 value 2.492440
  iter  90 value 1.725077
  iter 100 value 1.625412
  final  value 1.625412
  stopped after 100 iterations
> plot(x,y,col="blue",pch=16)
> # Le PMC en mode de fonctionnement
> lines(x1,predict(nn,data.frame(x=x1)),col="red")
```

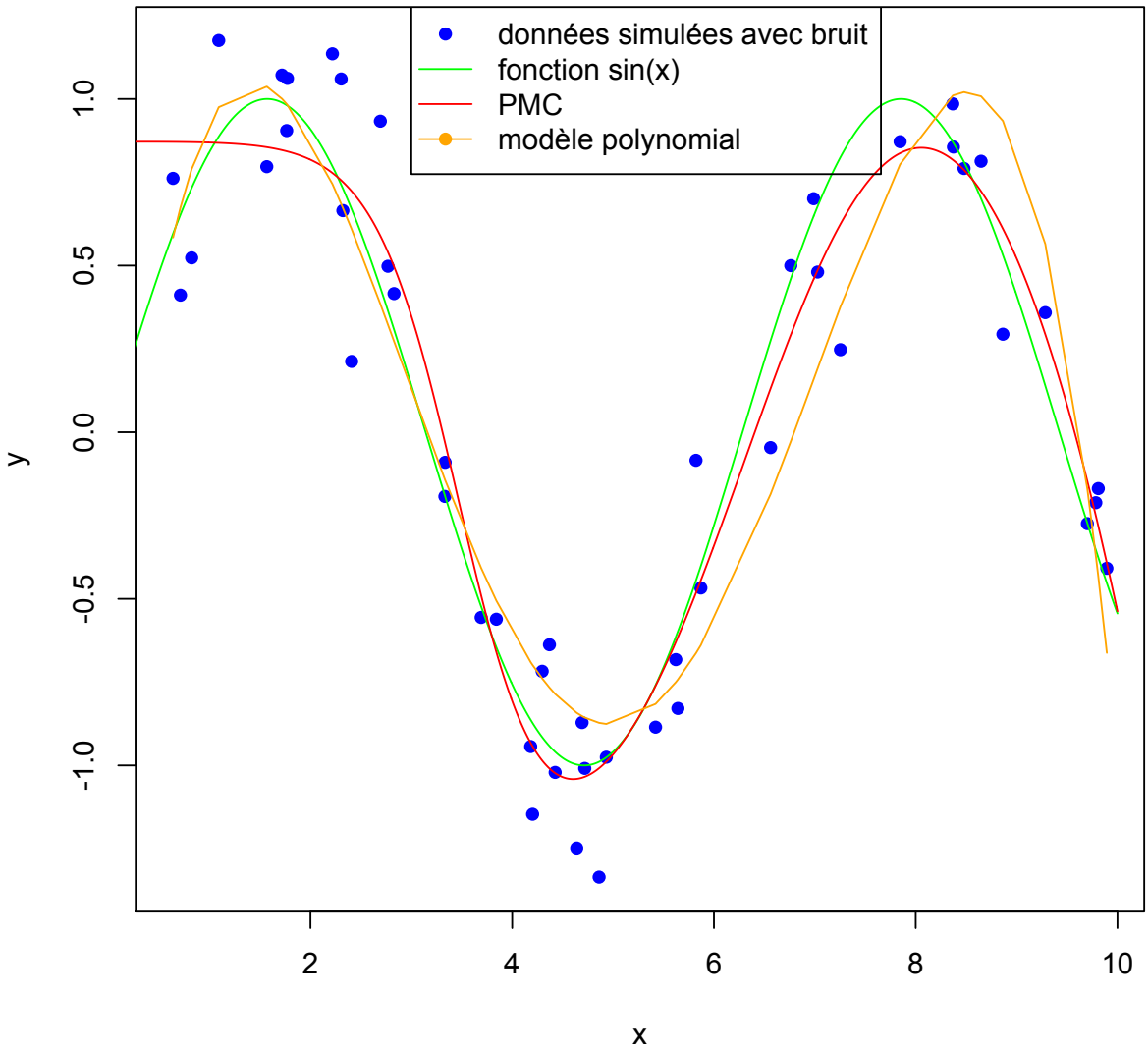
```
> # La courbe  sin(x)
>  x1<-seq(0,10,by=0.01)
> lines(x1,sin(x1),col="green")
> # Comparaison avec le mod\'ele polynomial

> modelPoly <- lm(y ~ x + I(x^2) + I(x^3) + I(x^4))
> lines(x,predict(modelPoly), col="orange")
> # L\'egende
> legend(3,1.3,c("donnees simulees avec bruit","fonction sin(x)","PMC",
> "modele polynomial"),lty=c(0,1,1,1),pch=c(16,-1,-1-1),
> col=c("blue","green","red","orange"))
```

Pour information, les 50 valeurs de x et le réseau nn sont donnés ci-dessous :

```
> x
 [1] 0.2281719 0.3257386 0.7966766 0.8112717 0.8703705 0.9161668 1.0302398
  [8] 1.1647974 1.2524789 1.2924630 1.6025026 1.6492091 1.9138583
1.9759473
 [15] 2.1243115 2.1298380 2.4696194 2.9106624 3.0283761 3.1990436
4.3323697
 [22] 4.3391228 4.3732048 4.4037807 5.0232916 5.3745347 5.3895886
5.5184110
 [29] 5.5541743 5.7767234 5.8857700 6.7648483 6.8127191 6.8559518
7.1900042
 [36] 7.4616387 7.5611733 7.7424133 7.8223810 7.8449987 7.9895721
8.0002264
 [43] 8.2089625 8.3361941 8.6121161 8.8354869 8.9015127 9.4589206
9.6280282
 [50] 9.9027404

> summary(nn)
 a 1-4-1 network with 13 weights
  options were - linear output units
   b->h1 i1->h1   5.19  -6.82
  b->h2 i1->h2
    6.68  -1.05
   b->h3 i1->h3
   -2.69   0.40
   b->h4 i1->h4
    1.77  -2.80
    b->o  h1->o  h2->o  h3->o  h4->o
   15.39   1.72 -11.92 -19.85  -4.74
```



2. Autre exemple : Discrimination avec “Iris”

Pour cet exemple de discrimination, nous allons utiliser le jeu de données Iris, utilisé par Fisher et disponible sur le logiciel R. Ce sont des données sur les mesures en centimètres de la longueur et la largeur sépale ainsi que la largeur et la longueur des pétales de 3 types de fleurs (setosa, versicolor, et virginica). Notre variable à expliquer est le type de fleurs (Species), comme il s’agit d’une variable de nature qualitative, il nous faut utiliser un modèle de discrimination.

Pour une discrimination, la première chose à faire est de découper notre jeu de données en 2 parties de manière aléatoire, la première partie va être utilisée pour l’apprentissage et la seconde pour la validation :

```
> appindex = sample(1 :nrow(iris), round(2*nrow(iris)/3), replace = FALSE)

> app = iris[appindex,]
> val = iris[-appindex,]
> val
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
9	4.4	2.9	1.4	0.2	setosa
12	4.8	3.4	1.6	0.2	setosa
21	5.4	3.4	1.7	0.2	setosa
23	4.6	3.6	1.0	0.2	setosa
25	4.8	3.4	1.9	0.2	setosa
26	5.0	3.0	1.6	0.2	setosa
30	4.7	3.2	1.6	0.2	setosa
32	5.4	3.4	1.5	0.4	setosa
35	4.9	3.1	1.5	0.2	setosa
46	4.8	3.0	1.4	0.3	setosa
47	5.1	3.8	1.6	0.2	setosa
48	4.6	3.2	1.4	0.2	setosa
51	7.0	3.2	4.7	1.4	versicolor
52	6.4	3.2	4.5	1.5	versicolor
57	6.3	3.3	4.7	1.6	versicolor
58	4.9	2.4	3.3	1.0	versicolor
60	5.2	2.7	3.9	1.4	versicolor
64	6.1	2.9	4.7	1.4	versicolor
65	5.6	2.9	3.6	1.3	versicolor
66	6.7	3.1	4.4	1.4	versicolor
70	5.6	2.5	3.9	1.1	versicolor
75	6.4	2.9	4.3	1.3	versicolor
79	6.0	2.9	4.5	1.5	versicolor
80	5.7	2.6	3.5	1.0	versicolor
82	5.5	2.4	3.7	1.0	versicolor
85	5.4	3.0	4.5	1.5	versicolor
88	6.3	2.3	4.4	1.3	versicolor
89	5.6	3.0	4.1	1.3	versicolor
92	6.1	3.0	4.6	1.4	versicolor
95	5.6	2.7	4.2	1.3	versicolor
107	4.9	2.5	4.5	1.7	virginica
108	7.3	2.9	6.3	1.8	virginica
111	6.5	3.2	5.1	2.0	virginica
123	7.7	2.8	6.7	2.0	virginica
124	6.3	2.7	4.9	1.8	virginica

```
   133          6.4         2.8          5.6         2.2  virginica
   135          6.1         2.6          5.6         1.4  virginica
   136          7.7         3.0          6.1         2.3  virginica
   138          6.4         3.1          5.5         1.8  virginica
   139          6.0         3.0          4.8         1.8  virginica
   140          6.9         3.1          5.4         2.1  virginica
   141          6.7         3.1          5.6         2.4  virginica
   142          6.9         3.1          5.1         2.3  virginica
   143          5.8         2.7          5.1         1.9  virginica
   144          6.8         3.2          5.9         2.3  virginica
   148          6.5         3.0          5.2         2.0  virginica
   149          6.2         3.4          5.4         2.3  virginica
>
```

Par la suite, nous ajustons notre modèle de réseau de neurones sur la base d'apprentissage en utilisant le package nnet :

```
> library(nnet)
```

Le pas d'apprentissage λ est initialisé avec le paramètre "decay" et nous utilisons nnet avec la classe "formula" :

```
> model.dis = nnet(Species ~ ., data = app, size = 2, decay = 0.001)
# weights :  19
   initial  value 113.674985
   iter  10 value 46.420430
   iter  20 value 45.840142
   iter  30 value 45.591998
   iter  40 value 11.388218
   iter  50 value 7.137760
   iter  60 value 7.028923
   iter  70 value 6.587153
   iter  80 value 5.747309
   iter  90 value 5.675102
   iter 100 value 5.652180
   final  value 5.652180
   stopped after 100 iterations
> summary(model.dis)
a 4-2-3 network with 19 weights
   options were - softmax modelling  decay=0.001
   b->h1 i1->h1 i2->h1 i3->h1 i4->h1
   -0.27  -0.65  -1.92   3.04   1.44
   b->h2 i1->h2 i2->h2 i3->h2 i4->h2
   10.56   0.26   1.34  -2.07  -3.48
   b->o1 h1->o1 h2->o1
    3.07  -9.86   4.53
   b->o2 h1->o2 h2->o2
   -6.39   5.52   6.63
```

```
b->o3 h1->o3 h2->o3
 3.28   4.33 -11.16
```

Remarquons que notre modèle a été ajusté sur la base d'apprentissage et que pour avoir une idée de sa performance, on peut calculer le taux d'erreur de classement lorsque la prédiction est réalisée sur le corpus de validation :

val[,-5] : toutes les colonnes de val, sauf la 5 ième, celle que l'on désire prédire avec predict et notre réseau model.dis

```
> val[, -5]
```

val[, 5] : la dernière colonne

```
> val[, 5]
 [1] setosa     setosa     setosa     setosa     setosa     setosa

 [7] setosa     setosa     setosa     setosa     setosa     setosa

[13] setosa     setosa     setosa     versicolor versicolor versicolor
[19] versicolor versicolor versicolor versicolor versicolor versicolor
[25] versicolor versicolor versicolor versicolor versicolor versicolor
[31] versicolor versicolor versicolor virginica  virginica  virginica
[37] virginica  virginica  virginica  virginica  virginica  virginica
[43] virginica  virginica  virginica  virginica  virginica  virginica
[49] virginica  virginica
Levels : setosa versicolor virginica
```

```
> pred = predict(model.dis, newdata = val[, -5], type = "class")
```

Voyons notre prédiction et comparons-là ensuite dans la matrice mat avec la réalité

```
> pred
 [1] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"
 [6] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"

[11] "setosa"     "setosa"     "setosa"     "setosa"     "setosa"

[16] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[21] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[26] "versicolor" "versicolor" "versicolor" "versicolor" "versicolor"
[31] "versicolor" "versicolor" "versicolor" "versicolor" "virginica"
[36] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
[41] "virginica"  "virginica"  "versicolor" "virginica"  "virginica"
[46] "virginica"  "virginica"  "virginica"  "virginica"  "virginica"
```

```
> mat = table(pred, val[, 5])
> taux = sum(diag(mat))/sum(mat)
> mat

pred         setosa versicolor virginica
```

```
setosa          15           0          0
versicolor       0          18          2
virginica        0           0         15
```

A “l’oeil nu”, il y a deux erreurs : deux virginica estimées versicolor ; ce que confirme la matrice mat. Notre taux de réussite est :

```
> taux
```

```
[1] 0.96
```

Pour notre première tentative, nous avons pris des paramètres par défauts. A présent, l’objectif va être de trouver le size et le decay optimal pour notre réseau de neurones. En effet, toute méthode d’apprentissage a ses propres paramètres que l’utilisateur doit ajuster pour construire un bon modèle à la fois adapté aux données de la base d’apprentissage et performant du point de vue de la prédiction sur les nouvelles observations. Les 2 principaux paramètres de notre modèle sont :

- size : le nombre de neurones de la couche cachée
- decay : le pas d’apprentissage (paramètre de décomposition)

En faisant varier ces deux paramètres, on s’aperçoit que plus le modèle est complexe (plus size est grand), mieux il apprend sur les données et plus il est performant du point de vue de la prédiction car chaque neurone supplémentaire permet de prendre en compte des profils spécifiques de neurones d’entrée. On peut également remarquer qu’au-delà d’un certain seuil, la performance n’augmente plus, voire diminue la capacité de généralisation du réseau, c’est le sur-apprentissage. Il n’existe pas de règle générale mais des règles empiriques. La taille de la couche cachée doit être :

- soit égale à celle de la couche d’entrée (Wierenga et Kluytmans, 1994)
- soit égale à 75% de celle-ci (Venugopal et Baets, 1994)
- soit égale à la racine carrée du produit du nombre de neurones dans la couche d’entrée et de sortie (Shepard, 1990).

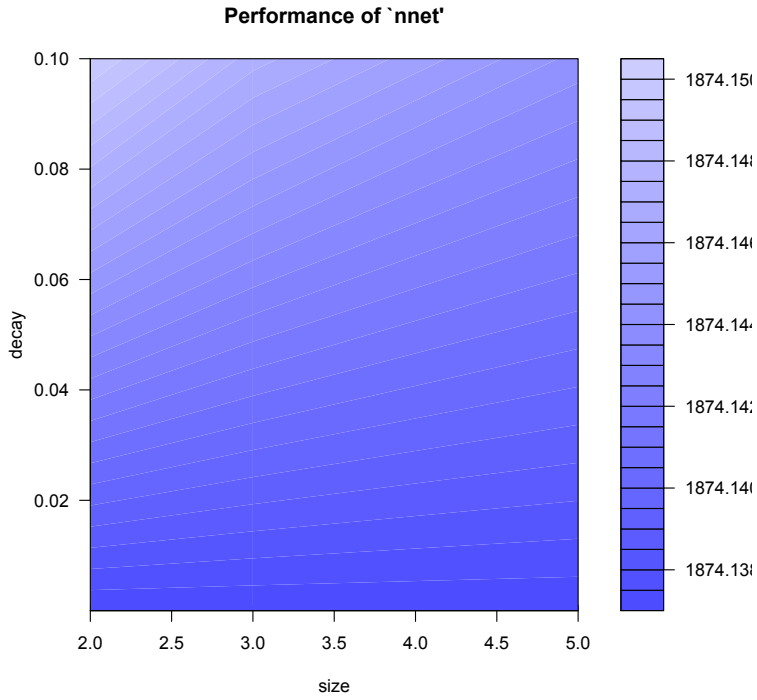
En faisant varier le decay, nous constatons que plus la décomposition de nos neurones est importante, plus l’est la performance et réciproquement, s’il y a trop de décompositions, le modèle redevient moins performant. La problématique consiste à trouver le bon paramétrage du modèle.

A cette fin, nous utilisons la fonction tune du package e1071 pour essayer de répondre à cette question : library(e1071) ; il vous faudra probablement installer ce paquetage (voir onglet “Paquetages & Données))

```
> library(e1071)
```

```
> tune . model = tune . nnet (Species ~ ., data = app, size = c (1, 3, 5),
decay = c (0.1, 0.001, 0.000001))
```

```
> plot(tune.model)
```




```
> tune.model

   Parameter tuning of ’nnet’ :
   - sampling method : 10-fold cross validation
   - best parameters :
   size decay
      3   0.1
   - best performance : 0.03
```

```
> model = nnet(Species ~ ., data = app, size = 3, decay = 0.1, maxit = 100)
# weights :  27
   initial  value 120.055082
   iter  10 value 54.792859
   iter  20 value 42.481984
   iter  30 value 29.301858
   iter  40 value 22.516971
   iter  50 value 21.820011
   iter  60 value 21.425149
   iter  70 value 21.334124
   final  value 21.334122
   converged
```

```
> summary(model)
a 4-3-3 network with 27 weights
   options were - softmax modelling  decay=0.1
```

```
b->h1 i1->h1 i2->h1 i3->h1 i4->h1
 0.18   0.30   1.06  -1.76  -0.74
b->h2 i1->h2 i2->h2 i3->h2 i4->h2
-2.65  -1.68  -1.52   2.73   2.44
b->h3 i1->h3 i2->h3 i3->h3 i4->h3
 0.18   0.30   1.06  -1.76  -0.74
b->o1 h1->o1 h2->o1 h3->o1
-1.50   2.64  -1.32   2.64
b->o2 h1->o2 h2->o2 h3->o2
 2.80  -1.99  -3.38  -1.99
b->o3 h1->o3 h2->o3 h3->o3
-1.30  -0.65   4.70  -0.65
```

```
> pred = predict(model, newdata = val[, -5], type = " class ")

> mat = table(pred, val[, 5])

> taux = sum(diag(mat))/ sum(mat)

> mat
```

```
pred         setosa versicolor virginica
 setosa          15          0         0
 versicolor       0         18         0
 virginica        0          0        17
```

```
> taux

[1] 1

>
```

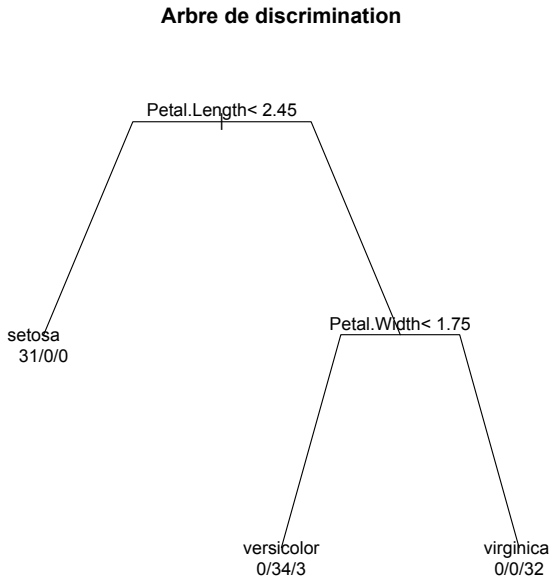
Il n'y a plus d'erreur.

A présent, on peut réaliser une discrimination sur le même jeu de données. Nous allons utiliser le package rpart. On commence par réaliser le même découpage avec une partie pour l'apprentissage et une autre pour la validation. Puis on calcule notre modèle et on trace l'arbre de décision :

```
> library(rpart)

> model = rpart(Species~., data = app, method = " class ")

> plot(model, uniform = TRUE, branch = 0.5, margin = 0.1, main = " Arbre de discrimination ")
```



> text(model, all = FALSE, use .n = TRUE)

Ensuite, comme pour les réseaux de neurones, nous allons rechercher les paramètres optimaux pour la fonction rpart. Ici, nous avons à faire varier les paramètres suivants :

- minsplit : nombre minimal d'observations d'un noeud.
- cp : coefficient de complexité du modèle.
- xval : nombre de validations croisées
- maxdepth : profondeur maximale de l'arbre, sachant qu'à la racine, la profondeur vaut 0.

> tune . model = tune . rpart (Species ~ ., data = app, minsplit = c (15, 20, 25), cp = c (0.00001, 0.000001), maxcompete = 4, maxsurrogate = 5,
usesurrogate = 2, xval = c (10, 15), surrogatestyle = 0, maxdepth = c (25, 30))

> tune.model

```
Parameter tuning of 'rpart.wrapper' :
- sampling method : 10-fold cross validation
- best parameters :
minsplit    cp maxcompete maxsurrogate usesurrogate xval surrogatestyle
      15 1e-05          4            5            2   10              0
maxdepth
      25
- best performance : 0.09
```

On peut déjà remarquer qu'en utilisant le PCM, la performance est meilleure (0.03) qu'à l'utilisation de l'arbre de décision (best performance : 0.09). Ainsi nous pouvons utiliser la fonction avec les paramètres optimaux :

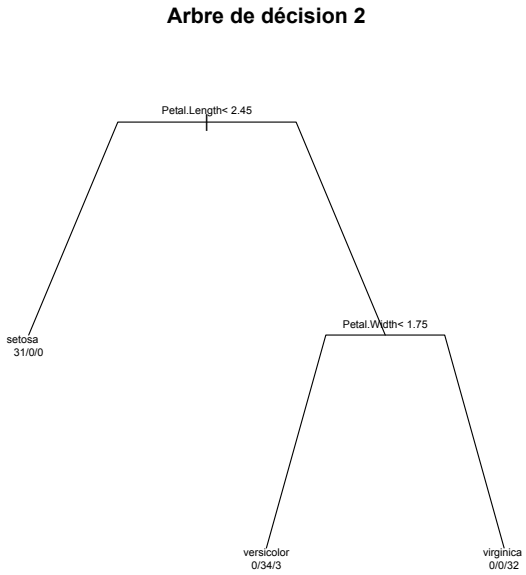
```
> model = rpart (Species
., data = app, method = " class ",
    control = rpart . control (cp = 0.00001, minsplit = 15, maxcompete = 4, maxsurrogate = 5,
    usesurrogate = 2, xval = 10, surrogatestyle = 0, maxdepth = 25))
```

L'arbre qui nous intéresse est celui qui minimise l'erreur standard (xerror + xstd dans notre modèle). Si plusieurs arbres minimisent cette quantité, on privilègie l'arbre le plus petit. Ainsi il faut réaliser un élagage puis on peut afficher l'arbre de décision :

```
> cpTab = as . data . frame(model $ cptable)
> ind = cpTab $ xerror + cpTab $ xstd
> cpTab = cbind . data . frame(cpTab, ind)
> monCp = cpTab[which . min(cpTab $ ind), " CP "]
> elag = prune(model, cp = monCp)
```

Affichons l'arbre :

```
> plot(elag, uniform = TRUE, branch = 0.5, margin = 0.1, main = " Arbre de decision 2")
```




```
> text(elag, all = FALSE, use .n = TRUE, cex = 0.6)
```

Enfin, nous allons utiliser notre arbre élagué pour la prédiction de notre base de validation et construire la matrice de confusion :

```
> prev = predict(elag, newdata = val[, -5], type = " class ")
> mat = table(prev, val[, 5])
> taux = sum(diag(mat))/ sum(mat)
> mat
```

```
  prev          setosa versicolor virginica
   setosa           15          0         0
   versicolor        0         18         2
   virginica         0          0        15
> taux
[1] 0.96
```

Le taux d'erreur de prédiction est moins bon pour la discrimination (4%) que pour le réseau de neurones (0%). Cependant, la différence est minime et dépend de l'aléa du découpage de notre jeu de données. De plus, le critère de qualité de la prédiction (best performance) est meilleur pour le réseau de neurones que pour la discrimination avec un arbre de décision. Ainsi, on peut affirmer que le Perceptron Multi-Couches est de qualité similaire à une technique de discrimination pour faire de la prévision.

```
>
```