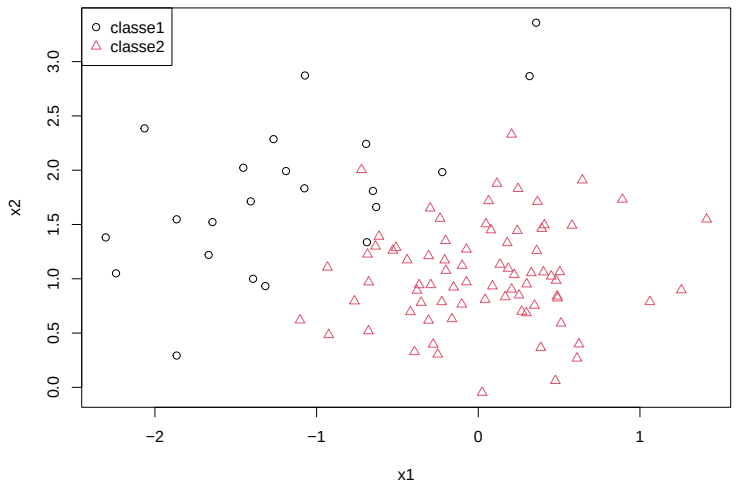
Apprentissage automatique

TP8 : arbres de classification

Exercice 1. On utilise dans cet exercice les données synth_train.txt et synth_test.txt.

```
train <- read.table(file="../data/synth_train.txt", header=TRUE)
Xtrain <- train[,-1]
Ytrain <- train$y
plot(Xtrain, pch=Ytrain, col=Ytrain)
legend("topleft", legend=c("classe1", "classe2"), pch=1:2, col=1:2)
```



1. Lire les aides sur les fonctions `rpart` et `rpart.control` du package `rpart`.

   ```
   library(rpart)
   help(rpart)
   help(rpart.control)
   ```

   Quelle est par défaut la matrice de coûts ? La fonction d'impureté ? Les critères d'arrêt ? Comment sont gérées les données manquantes ? Quelle est la différente entre une question `competitor` et une question `surrogate` ?

2. Charger les données d'apprentissage.

   ```
   train <- read.table(file="../data/synth_train.txt", header=TRUE)
   ```

   Appliquer la fonction `rpart` aux données d'apprentissage.

   ```
   tree <- rpart(y~., data=train, method="class",
                 control=list(maxcompete=0))
   ```

   Expliquer les résultats de la fonction `print`

```
print(tree)
```

Puis visualiser l'arbre de classification.

```
library(rpart.plot)
rpart.plot(tree, extra = 1)
```

Quel est le taux d'erreur d'apprentissage ?

```
#(1+7)/100=8% d'erreur
```

3. Expliquer les résultats de la fonction `summary`. Pourquoi l'algorithme s'arrête ici après une seule division ?

```
summary(tree)
```

```
# Improve = réduction de l'impureté première division (avec Gini) :
# 1-0.22^2-0.78^2-0.16*(1-0.938^2-0.062^2)-0.84*(1-0.083^2-0.917^2)
# arrêt des divisions car  n=16 dans node2 (minsplit=20) et
# cp < 0.01 dans node3 probablement.
```

4. Charger le jeu de données test.

```
test <- read.table(file="../data/synth_test.txt", header=TRUE)
```

Prédire les données test avec cet abre de classification.

```
pred.test <- predict(tree, newdata=test, type="class")
```

Quel est le taux d'erreur test ?

```
sum(pred.test!=test$y)/length(pred.test) # taux erreur test
```

5. Estimer les probabilités à posteriori des données test.

```
prob <- predict(tree, newdata=test, type="prob")
```

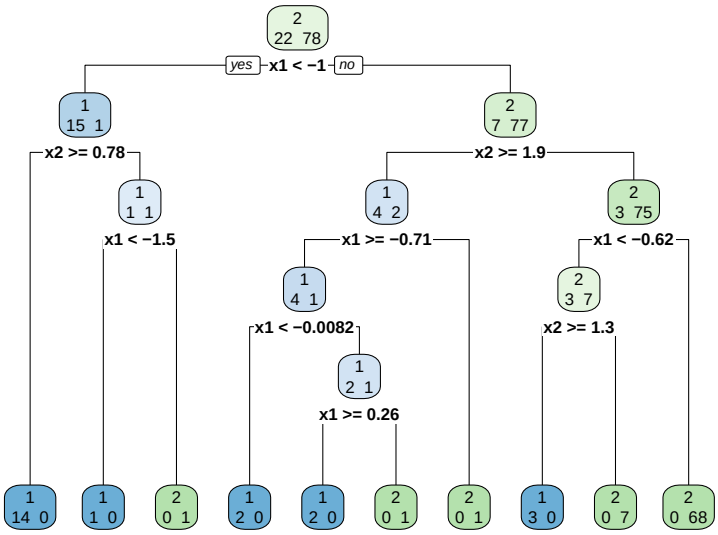
Pourquoi les probabilités à posteriori des premières données test sont toutes identiques ?

```
head(prob)
```

6. Construire maintenant l'arbre de longueur maximale avec comme seul critère d'arrêt `minsplit`=2.

```
tree <- rpart(y~., data=train, minsplit=2, cp=0,
              method="class",
             control=list(maxcompete=0))
```

```
library(rpart.plot)
rpart.plot(tree, extra = 1)
```



Retracer à la main cet arbre en utilisant la numérotation des noeuds de la fonction `summary`, sans les questions binaires mais en indiquant à chaque noeud la valeur du paramètre de complexité, le nombre d'observations et leur répartition dans chaque classe (en souligant le nombre de mals classés).

```
summary(tree)
```

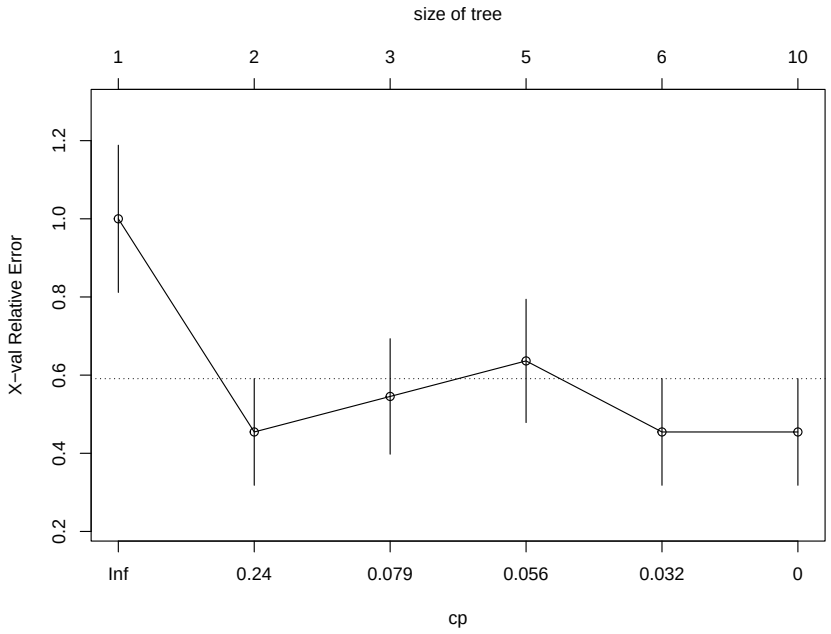
7. Expliquer les résultats de la fonction `printcp`.

```
printcp(tree)
#tree$cptable
```

Quels sont les sous-arbres élagués associés ?

8. Executer le code ci-dessous :

```
plotcp(tree)
```



Retrouver les valeur en abscisse de ce graphique à partir des paramètres de complexités des noeuds de l'arbre.

```
cp <- tree$cptable[,1]
#beta1=inf
#beta2=sqrt(cp2*cp1)
sqrt(cp[2]*cp[1])
#beta3
sqrt(cp[3]*cp[2])
```

Expliquer le lien entre la taille de l'arbre (axe du haut) et la valeur du paramètre de complexité (axe du bas).

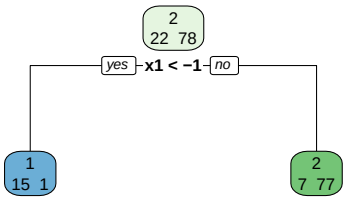
```
# Le sous-arbre élagué à 5 feuilles (4 splits) est optimal
# pour le paramètre de complexité de 0.056 .
```

Expliquer les valeurs représentées en ordonnées. Expliquer la signification de la ligne pointillée.

```
xerror <- tree$cptable[,4]
xstd <- tree$cptable[,5]
min(xerror)
min(xerror)+xstd[which.min(xerror)] #ligne pointillée
```

9. Choisir une valeur du paramètre de complexité et élaguer l'arbre avec la fonction `prune`.

```
tree2 <- prune(tree, cp=0.24)
tree2
rpart.plot(tree2, extra = 1)
```



Exercice 2. On reprend le jeu de données où 1260 exploitations agricoles saines ou défaillantes sont décrites par $p = 22$ critères économiques et financiers.

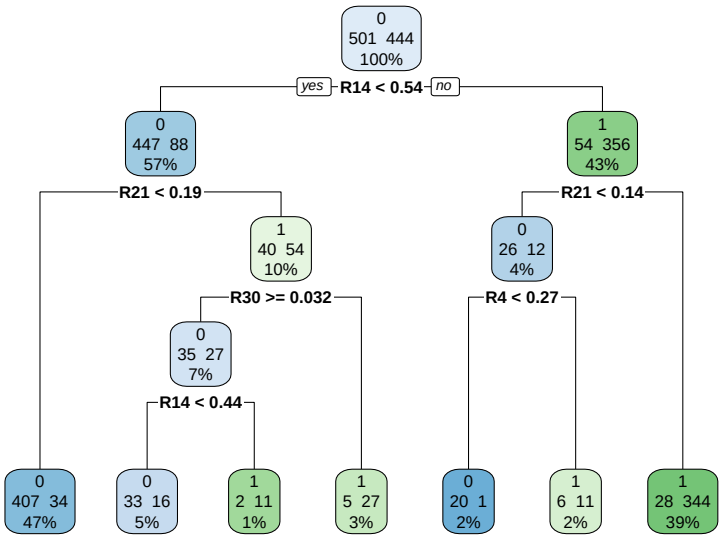
```
load("../data/Desbois_complet.rda")
```

1. Découper aléatoirement les $n = 1260$ exploitations agricoles en 945 exploitations pour les données d'apprentissage et 360 exploitations pour les données test.

```
set.seed(10)
tr <- sample(1:nrow(data), 945)
train <- data[tr,]
test <- data[-tr,]
```

2. Constuire l'arbre de classification en laissant les valeurs par défaut. Visualisez l'arbre avec la fonction `rpart.plot`.

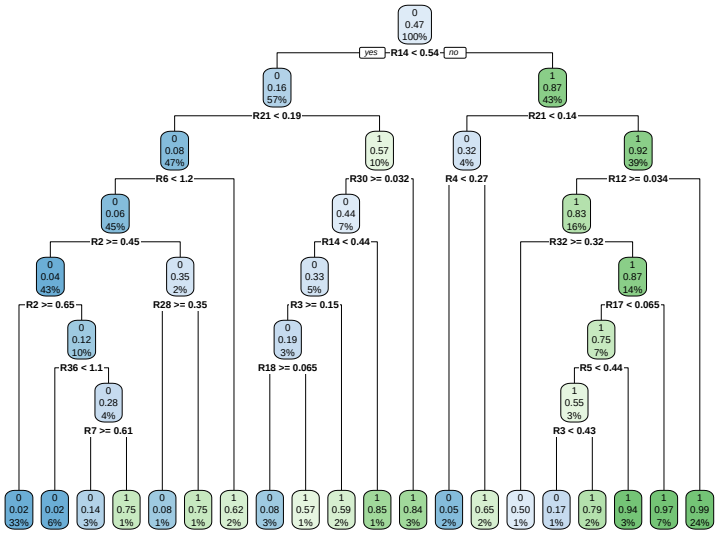
```
tree <- rpart(DIFF~., data=train, method="class")
rpart.plot(tree, extra=101)
```




```
#rpart.plot(tree, extra=106)
```

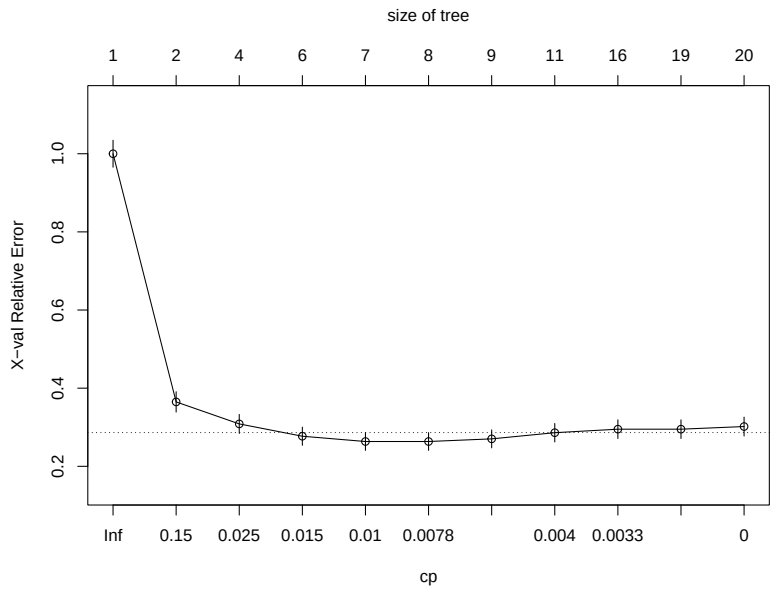
3. Constuire maintenant l'arbre de longeur maximale.

```
tree <- rpart(DIFF~., data=train, cp=0, method="class")
rpart.plot(tree, extra=106)
```



4. Faire le graphique des erreurs de validation de la suite des sous-arbres optimaux. Recommencez plusieurs fois. Que constatez-vous ? Quelle valeur du paramètre de complexité pourriez-vous choisir pour élaguer cet arbre ?

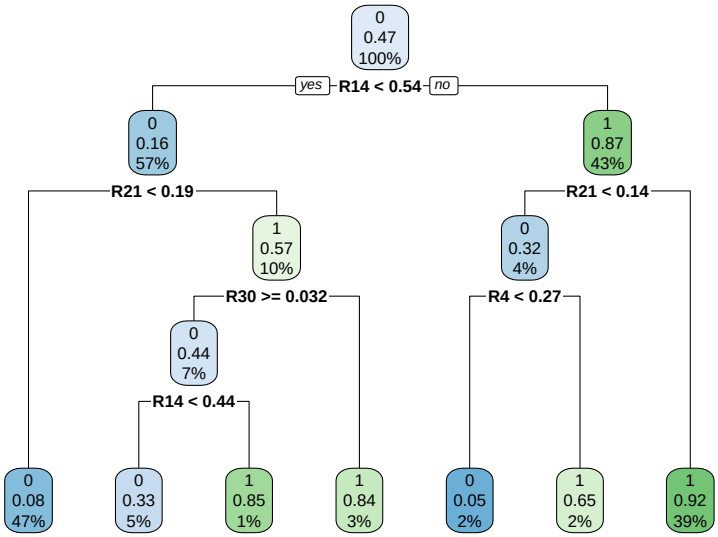
```
tree <- rpart(DIFF~., data=train, cp=0, method="class")
plotcp(tree)
```




```
# On constate que la valeur du cp.min est variable et que
#celle du cp.1se est plus stable
# par exemple cp=0.015 (mais cela dépend du
#découpage 10-folds)
tree2 <- prune(tree, cp=0.015)
rpart.plot(tree2, extra=106)
```

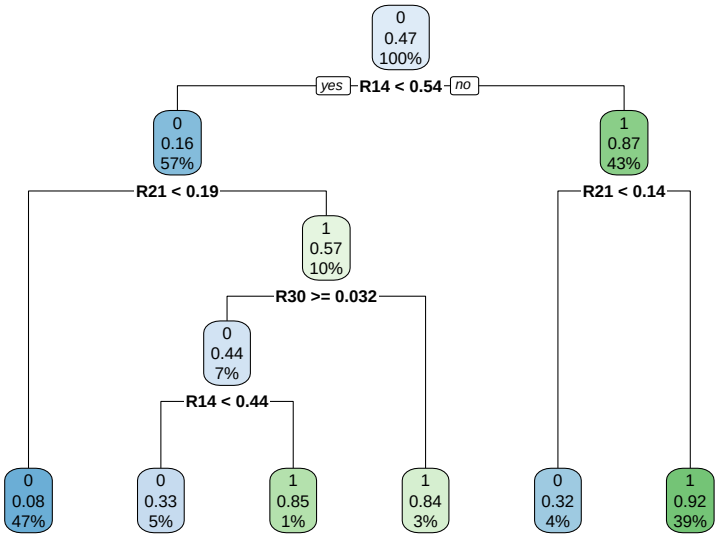
5. Elaguer cet arbre en choisissant **automatiquement** la valeur `cp.min` du paramètre de complexité.

```
cpmin <-  tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
tree2 <- prune(tree, cp=cpmin)
rpart.plot(tree2, extra=106)
```



6. Elaguer cet arbre en choisissant automatiquement la valeur `cp.1se` du paramètre de complexité.

```
ligne <- min(tree$cptable[,"xerror"])+
  tree$cptable[,"xstd"][which.min(tree$cptable[,"xerror"])]
cp1SE <-  tree$cptable[min(which(
  tree$cptable[,"xerror"]<=ligne)),"CP"]
tree3 <- prune(tree, cp=cp1SE)
rpart.plot(tree3, extra=106)
```



7. Prédire les données test. Quel est le taux d'erreur ?

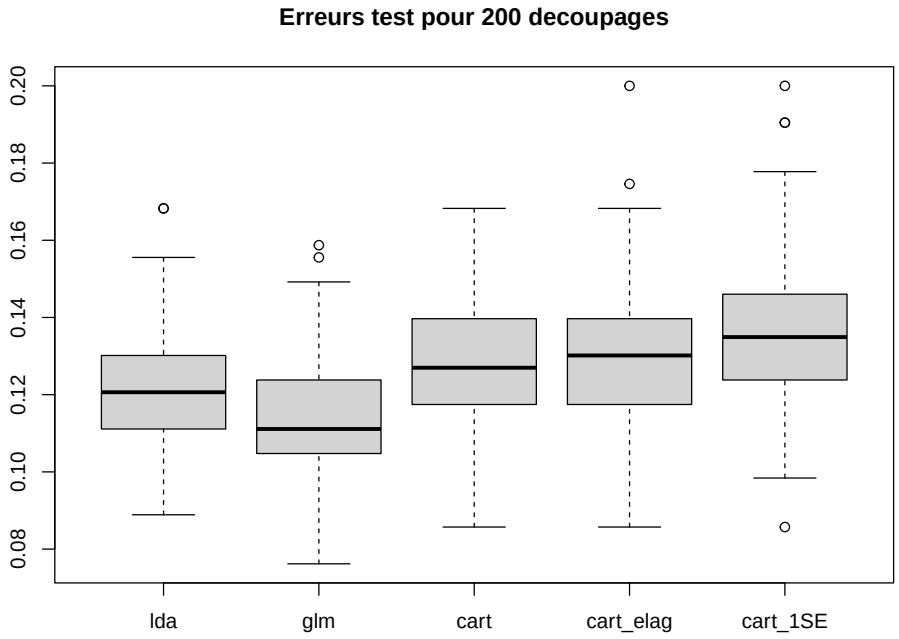
```r
pred.test <- predict(tree3, newdata=test[,-1],type="class")
sum(pred.test!=test$DIFF)/length(pred.test) # taux erreur test
```

8. Tracer la courbe ROC et calculer le AUC.
9. Comparer à partir de $B = 200$ découpages aléatoires des données (945 observations d'apprentissage et 315 observations test à chaque découpage), le taux d'erreur des méthodes `lda`, `glm` (pour la régression logistique, CART (avec le paramétrage par défaut), CART avec élagage (avec `cp.min`) CART avec élagage et `cp.1se`.

```r
B <- 200
err_lda <- rep(NA,B)
err_glm <- rep(NA,B)
err_rpart <- rep(NA,B)
err_elag <- rep(NA,B)
err_elag2 <- rep(NA,B)

for (b in 1:B)
{
  tr <- sample(1:nrow(data),945)
  train <- data[tr,]
  test <- data[-tr,]

  #Estimation des modèles
  library(MASS)
  g1 <- lda(DIFF~.,data=train) #LDA
  g2 <- glm(DIFF~.,data=train,family=binomial) #logistique
  g3 <- rpart(DIFF~., data=train,method="class")
  tree <- rpart(DIFF~., data=train,cp=0,method="class")
  cpopt <-  tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
  g4 <- prune(tree, cp=cpopt)
  ligne <- min(tree$cptable[,"xerror"])+
    tree$cptable[,"xstd"][which.min(tree$cptable[,"xerror"])]
  #max(which(tree$cptable[,"xerror"]<=ligne))
  cp1SE <-  tree$cptable[min(which(tree$cptable[,"xerror"]<=ligne)),"CP"]
  g5 <- prune(tree, cp=cp1SE)

  #Predictions et erreur test
  pred1<- predict(g1,test[,-1])$class
  err_lda[b] <- sum(pred1!=test$DIFF)/length(test$DIFF)
  prob <- predict(g2,test[,-1],type="response")
  pred2 <- 1*I(prob>0.5)+0*I(prob<=0.5)
  err_glm[b] <- sum(pred2!=test$DIFF)/length(test$DIFF)
  pred3 <- predict(g3,test[,-1],type="class")
  err_rpart[b] <- sum(pred3!=test$DIFF)/length(test$DIFF)
  pred4 <- predict(g4,test[,-1],type="class")
  err_elag[b] <- sum(pred4!=test$DIFF)/length(test$DIFF)
  pred5 <- predict(g5,test[,-1],type="class")
  err_elag2[b] <- sum(pred5!=test$DIFF)/length(test$DIFF)
}
```

```
err <- data.frame(lda=err_lda,glm=err_glm,cart=err_rpart,
                  cart_elag=err_elag, cart_1se=err_elag2)
save(err,file="../data/exo2_tp8.rda")
```

```
load("../data/exo2_tp8.rda")
boxplot(err,main="Erreurs test pour 200 decoupages")
```



Conclure.