# Data Mining
# Arbres de décision avec CART / `rpart`

L'objectif de ce TP est de se familiariser avec la pratique de la discrimination fondée sur les arbres de décision (méthode CART – *Classification and Regression Trees*) sous `R`. Un compte-rendu de TP vous est demandé. Ce compte rendu doit contenir le code `R` utilisé et d'abondants commentaires.

1. Commencez par installer un package supplémentaire pour `R`, nommé `rpart.plot`, permettant de représenter un arbre de décision avec de nombreuses informations utiles. Pour cela, utilisez la fonction `install.packages(...)`. L'instruction
   ```
   install.packages("rpart.plot")
   ```
   devrait fonctionner depuis chez vous. Vous pouvez également utiliser l'interface de `RStudio` (il faut chercher un peu). Pour vérifier que la package est bien installé, tapez :
   ```
   library(rpart.plot)
   ```
2. Nous allons travailler avec les données `ptitanic` disponibles dans le package `rpart.plot`. Chargez les données en tapant `data(ptitanic)`, puis lisez l'aide associée afin de comprendre les données.
3. Quel est le type de la variable `survived`. Résumez-la. Les données contiennent-elles des valeurs manquantes ? Expliquez.
4. Afin d'estimer un arbre de décision permettant de prédire la variable `survived` à partir de toutes les autres variables, il suffit de taper :
   ```
   r <- rpart(survived~., data = ptitanic)
   ```
   L'arbre de décision obtenu est *binaire* (chaque noeud interne à deux fils - il s'agit d'une caractéristique de la méthode CART). Il peut être représenté en tapant `rpart.plot(r)`. La classe majoritaire d'un noeud est donnée sur la première ligne de chaque rectangle de couleur. La proportion qui suit correspond à la proportion de survivants. Le pourcentage final correspond au pourcentage des exemples (lignes) satisfaisant tous les tests au-dessus du noeud considéré. Par exemple, pour les exemples d'apprentissage satisfaisant tous les tests au-dessus de la feuille en-bas à droite :
   ```
   ex <- subset(ptitanic, sex != "male" & pclass != "3rd") # exemples
   summary(ex$survived)
   sum(ex$survived == "survived") / nrow(ex) # proportion de survivants
   nrow(ex) / nrow(ptitanic) # pourcentage relat. à tous les exemples
   ```
   Décrivez les passagers correspondants à la feuille considérée.
   Attention, les calculs précédents deviennent plus compliqués lorsque les tests sous-jacents portent sur des variables contenant des valeurs manquantes.
5. Supposons que l'on ait la description suivante d'un nouveau passager :
   ```
     pclass    sex age sibsp parch
        3rd female  19     3     0
   ```

et que l'on souhaite prédire si cette personne a survécu ou pas. La lecture de l'arbre associe cette description à la 4e feuille en partant de la gauche. La prédiction est ainsi la non-survie (avec une probabilité de 86%). Cela peut être automatisé comme suit :

```
np <- data.frame("3rd", "female", 19, 3, 0) # nouveau passager
names(np) <- c("pclass", "sex", "age", "sibsp", "parch")
predict(r, newdata = np) # probabilités
predict(r, newdata = np, type = "class") # classe majoritaire
```

Pour estimer l'*erreur d'apprentissage* de l'arbre, on peut prédire la valeur de `survived` pour tous les exemples utilisés pour l'apprentissage :

```
pred.surv <- predict(r, newdata = ptitanic, type = "class")
```

et comparer le résultat à la réalité :

```
mat.conf <- table(ptitanic$survived, pred.surv)
```

Le tableau de contingence obtenu s'appelle la *matrice de confusion*. Expliquez pourquoi. Vérifiez que l'erreur d'apprentissage de l'arbre vaut 0.175 environ. Que cela signifie-t-il en pratique ?

6. Comme expliqué en cours, l'erreur d'apprentissage est une vision très optimiste de l'erreur réelle car l'arbre est construit pour coller au mieux aux exemples d'apprentissage. Afin d'obtenir une estimation de la *capacité de généralisation* de l'arbre, on utilise souvent un sous-ensemble des données pour l'apprentissage et on estime l'erreur de l'arbre sur les données restantes, appelées *ensemble test*. Par exemple :

```
n <- nrow(ptitanic)
param.app <- 0.7 # proportion des exemples pour l'apprentissage
set.seed(123) # on fixe la graine aléatoire
permut.lignes <- sample(n) # on "mélange" les indices des lignes
sel <- permut.lignes[1:(param.app * n)] # sélection des lignes pour l'apprentissage
app <- ptitanic[sel,] # ensemble d'apprentissage
test <- ptitanic[-sel,] # ensemble de test
nrow(app) + nrow(test) == nrow(ptitanic) # vérification
```

Le modèle est alors estimé sur les données d'apprentissage :

```
r2 <- rpart(survived~., data = app)
```

Puis, l'erreur est calculée sur les données de test par le biais de :

```
pred.surv2 <- predict(r2, newdata = test, type = "class")
```

Donnez la matrice de confusion correspondante et vérifiez que l'erreur de l'arbre sur l'ensemble test est de 0.20 environ (si vous avez une ancienne version de `R`, il se peut que vous ayez un résultat légèrement différent suite à un changement récent dans la fonction `set.seed()`).

7. L'algorithme d'apprentissage est contrôlé par plusieurs paramètres. Parmi les plus importants pour la méthode CART, on trouve `minsplit` et `cp`. Donnez le sens de `minsplit` en consultant `?rpart.control`. Le sens de `cp` est plus complexe à appréhender. On se contentera de retenir que plus `cp` est faible, plus l'arbre est profond. Par exemple :

```
r3 <- rpart(survived~., data = ptitanic, minsplit = 0, cp = 0)
rpart.plot(r3)
```

La pratique de la modélisation par arbre de décision consiste à essayer de choisir les paramètres de l'algorithme d'apprentissage en recherchant un compromis entre une faible erreur sur l'ensemble test et un arbre facilement compréhensible, c'est-à-dire, peu profond. Par exemple, pour trouver une bonne valeur de `cp` à `minsplit` constant, on peut procéder de la façon suivante :

```
minsplit <- 10 # on travaille à minsplit constant
mes.cp <- seq(0, 0.1, by = 0.001) # valeurs de cp à essayer
erreur.test <- numeric(length(mes.cp)) # les erreurs de test correspondantes
for (i in 1:length(mes.cp)) {
   r4 <- rpart(survived~., data = app, cp = mes.cp[i], minsplit = minsplit)
   pred.surv4 <- predict(r4, newdata = test, type = "class")
   mat.conf4 <- table(test$survived, pred.surv4)
   erreur.test[i] <- erreur(mat.conf4) # écrire la fonction erreur(...)
}
plot(mes.cp, erreur.test, type = "l") # erreur de test en fonction de cp
```

Lorsque `minsplit = 10`, `cp = 0.009` semble un choix pertinent. Expliquez pourquoi. Cela suggère de considérer l'arbre suivant comme modèle :

```
r5 <- rpart(survived~., data = ptitanic, cp = 0.009, minsplit = 10)
rpart.plot(r5)
```

8. Que peut-on reprocher à l'approche précédente relativement au choix de l'ensemble d'apprentissage et à l'ensemble test ?
9. Installez le package `mlbench`, chargez le jeu de données `Zoo` et estimez un arbre de décision pour prédire la variable `type`.