TP3 : Apprentissage supervisé

1 Mise en place

Dans le cadre du cours, nous avons introduit deux méthodes relatives au problème d'apprentissage supervisé. La première, l'analyse factorielle discriminante (AFD), était basée sur une ACP et l'autre, dite "classification and regression tree" (CART), était une méthode prédictive construite à partir d'un arbre binaire de décision.

Afin de faciliter la récupération des données et de disposer de certaines fonctions graphiques dans la suite de cette séance, nous commençons par charger le script *tp3.R*,

```
source("http://www.math.univ-toulouse.fr/~xgendre/ens/m2se/tp3.R")
```

Nous utiliserons également le paquet `rpart`. Il est normalement déjà installé pour les terminaux UNIX. Pour ceux qui utilisent leur ordinateur personnel, il est conseillé de lancer l'installation en début de séance car celle-ci peut prendre un certain temps. Pour cela, il faut entrer la commande `install.packages("rpart", dependencies=TRUE)`.

2 Insectes de Lubischew (AFD)

L'exemple que nous avons utilisé en cours pour illustrer l'AFD concerne le jeu de données des insectes de Lubischew. Il s'agit de $p = 6$ mesures morphologiques effectuées sur $n = 74$ insectes issus de trois espèces différentes, numérotées de 1 à 3 dans la suite. Pour récupérer ces données,

```
X <- DataLubischew()
```

Comme d'habitude, chaque ligne de la matrice des données `X` contient les mesures relatives à un individu. De plus, vous noterez que nous disposons ici d'une colonne supplémentaire contenant l'étiquette de chaque individu (*i.e.* le numéro de l'espèce).

Afin de visualiser la différence entre l'ACP de ces données et le résultat obtenu par une AFD, nous commençons par représenter les données dans le plan principal.

```
Xbar <- scale(X[, 1:6], scale = F)
Sigma <- t(Xbar) %*% Xbar/nrow(Xbar)
ACP <- eigen(Sigma)
C <- Xbar %*% ACP$vectors
cos2 <- rowSums(C[, 1:2]^2)/rowSums(C^2)
plot(C[, 1:2], cex = cos2)
```

Commenter brièvement cette représentation. En particulier, vous semble-t-il simple de distinguer les trois groupes d'insectes ? Comparer avec le graphe obtenu par la commande suivante,

```
plot(C[, 1:2], cex = cos2, col = X[, 7])
```

Pour mettre en place la procédure d'AFD, nous avons besoin de la matrice `T` qui affecte à chaque individu un groupe en accord avec son étiquette.

```
T <- matrix(0, nrow = nrow(X), ncol = 3)
for (i in 1:nrow(X)) T[i, X[i, 7]] <- 1
```

Examiner et expliquer le contenu de cette matrice. Chaque étiquette permet de définir un groupe d'individu. En utilisant des poids uniformes, chacun de ses groupes peut être pondéré proportionnellement à son nombre d'individus.

```
W <- diag(rep(1/nrow(X), nrow(X)))
Wbar <- t(T) %*% W %*% T
```

Afin de séparer la partie *inter* et la partie *intra* de la variance, nous considérons la matrice `G` contenant les centres de gravité de chaque groupe et sa version centrée `Gbar`.

```
G <- diag(1/diag(Wbar)) %*% t(T) %*% W %*% X[, 1:6]
g <- colMeans(X[, 1:6])
Gbar <- G - c(1, 1, 1) %*% t(g)
```

Cette dernière matrice nous permet enfin de calculer la matrice de covariance *inter*,

```
SigmaB <- t(Gbar) %*% Wbar %*% Gbar
```

Prenez le temps de comprendre les lignes précédentes. En particulier, vous pouvez vérifier que, par définition de `W`, la matrice `Wbar` est bien diagonale. De plus, ses éléments diagonaux sont bien les poids de chaque groupe. En développant formellement `G[i,j]`, vous pouvez aussi vérifier que les lignes de `G` contiennent les centres de gravité de chaque groupe.

Nous avons vu que l'AFD revient à faire l'ACP de `Gbar` avec la matrice des poids `Wbar` et la distance de Mahalanobis donnée par l'inverse de `Sigma`. Nous procédons donc comme dans les séances précédentes,

```
M <- solve(Sigma)
EigenM <- eigen(M)
P <- EigenM$vectors
Mhalf <- P %*% diag(sqrt(EigenM$values)) %*% t(P)

Gprime <- Gbar %*% Mhalf
AFD <- eigen(t(Gprime) %*% Wbar %*% Gprime)

C <- Xbar %*% Mhalf %*% AFD$vectors
Cbar <- Gprime %*% AFD$vectors
```

Expliquer le calcul de `Mhalf` et pourquoi il est plus simple de travailler avec `Gprime` plutôt qu'avec `Gbar`. En déduire la façon dont nous avons obtenu la matrice `C`.

Dans une AFD avec m groupes, nous savons qu'il n'y a que $\kappa = \min(m-1, p)$ valeurs propres non triviales. Que vaut κ pour nos données ? Commenter les résultats de la commande suivante,

```
cumsum(AFD$values)/sum(AFD$values)
```

Que vaut `AFD$values[1]` et comment s'interprète cette valeur ? Nous terminons par représenter le résultat de notre AFD dans le plan avec les centres de gravité de chaque groupe,

```
plot(C[, 1:2], col = X[, 7])
points(Cbar[, 1:2], pch = 15)
```

Comparer ce graphique avec celui que nous avions obtenu grâce à l'ACP.

3 Des mails et des spams (CART)

Nous nous intéressons maintenant à la procédure CART que nous illustrons sur les données de Hewlett-Packard contenant des messages électroniques et des spams. Nous avons à notre disposition $n = 4601$ messages dont 1813 spams et, pour chaque message, $p = 57$ variables relatives à la fréquence de certains mots, à celle de certains caractères et aux chaînes de lettres majuscules. Ces données peuvent être récupérées par la commande,

```
X <- DataSpam()
```

Notre objectif est encore de mettre en place une procédure d'apprentissage supervisé. Cette fois, il y a donc deux étiquettes "Mail" et "Spam". Expliquer pourquoi l'AFD ne peut nous fournir qu'une réponse limitée dans cette situtation. Afin de savoir si l'étiquette d'un message est "Mail" ou "Spam", nous utiliserons la dernière colonne de `X`, appelée `Status`,

```
X$Status
```

Pour mettre en place la procédure de décision par arbre, nous utiliserons le paquet `rpart`. Il faut commencer par le charger et par regarder l'aide relative à sa fonction principale,

```
library(rpart)
help(rpart)
```

Nous lisons dans l'aide que trois paramètres doivent au moins être fournis à la fonction `rpart` :

- `formula` ce paramètre nous permet d'indiquer quelle est la variable à expliquer et quelles sont les variables explicatives. Sa syntaxe est la même que pour la fonction `lm`. Nous n'entrons pas ici dans les détails des objets de type *formula* de R et nous nous contenterons d'utiliser la variable `spam_formula` définie comme suit,
  ```
  spam_form <- paste("Status ~ ", paste(colnames(X)[1:57], collapse = " + "))
  ```
- `method` comme nous l'avons mentionné durant le cours, la méthode CART peut être utilisée à d'autres fins que l'apprentissage supervisé. Comme nous ne souhaitons ici faire que de la classification, nous passerons la valeur `"class"`.
- `data` il s'agit du jeu de données contenant les variables annoncées dans la formule. Pour nous, il s'agit simplement de `X`.

Nous obtenons donc notre premier arbre par la commande

```
tree <- rpart(formula = spam_form, method = "class", data = X)
```

La commande `print(tree)` affiche les variables ainsi que les seuils choisis pour chaque test binaire et `summary(tree)` donne plus de précisions. Pour afficher l'arbre, il faut exécuter les commandes suivantes,

```
plot(tree, uniform = TRUE)
text(tree, all = TRUE)
```

Prendre le temps de lire l'aide de `text.rpart` afin de voir toutes les informations que vous pouvez afficher sur l'arbre. Commenter les résultats obtenus.

Dans le cadre du cours, nous avons discuté du problème de sur-apprentissage lié à une trop grande adéquation aux données. En particulier, nous avons présenté une procédure pénalisée pour élaguer un arbre qui serait trop "proche" des données. La commande `rpart` procède de cette façon et les paramètres de contrôle sont passés via l'argument `control`. Lisez l'aide sur l'objet `rpart.control` pour plus de détails et considérez l'arbre obtenu par les commandes suivantes,

```
spam_ctrl <- rpart.control(cp = 1e-04)
tree <- rpart(formula = spam_form, method = "class", data = X, control = spam_ctrl)
```

Quel est le rôle du paramètre `cp` ? Faites varier ce paramètre, qu'observez-vous ? Quel lien pouvez-vous faire avec la procédure d'élagage décrite en cours ?

Nous allons finir cette séance en élaguant nous-même l'arbre CART obtenu avec le paramètre `cp=0.0001`. Une méthode courante en pratique pour cela consiste à estimer par validation croisée l'erreur commise par notre procédure (voir le paramètre `xval` de `spam_ctrl`). Cette erreur est calculée par `rpart` et vous pouvez la lire avec la commande `printcp(tree)` dans la colonne `xerror`. Il est aussi possible de visualiser cette erreur en fonction de la taille de l'arbre grâce à la commande

```
plotcp(tree)
```

Quelle valeur de `cp` ce graphe vous suggère-t-il ? Élaguer l'arbre avec cette valeur grâce à la fonction `prune`. Est-ce que le résultat semble satisfaisant ? Quel est le problème ? A l'aide des informations dans `tree$cptable`, proposer une méthode similaire à celle vue en cours pour faire un choix "graphique" du `cp`.