TP ozone : arbre de régression et de décision

Résumé

Prévision du pic d'ozone par arbre de régression et de décision.

1 Introduction

Deux librairies, `tree` et `rpart`, proposent les techniques CART avec des algorithmes analogues à ceux développés dans Splus mais moins de fonctionnalités ; la librairie `rpart` fournissant des graphes plus explicites, des options plus détaillées et une procédure d'élagage plus performante est préférée. Cette fonction intègre une procédure de validation croisée pour évaluer le paramètre de pénalisation de la complexité.

Différents paramètres contrôlent l'exécution : `cp` désigne la complexité minimale pour la construction de l'arbre maximal, le nombre minimal d'observation par nœud, le nombre de validations croisées (par défaut 10)... (cf. l'aide en ligne `?rpart.control` pour plus de détails). En fait la documentation des fonctions concernées est un peu "laconique" et il est difficile d'en comprendre tous les détails sans suivre leur évolution par les forums d'utilisateurs. C'est souvent un des travers des logiciels en accès libre mais la communauté est très "réactive" pour répondre aux questions à condition qu'elles ne soient pas trop "naïves". Il est en effet important de consulter les archives pour ne pas reposer des problèmes déjà résolus et abondamment commentés.

Enfin, deux types d'arbre peuvent être estimer selon que l'on considère que la variable à modéliser est la concentration d'ozone (arbre de régression) ou directement le dépassement du seuil (arbre de discrimination ou de décision).

2 Estimation et élagage de l'arbre de régression

```
library(rpart)  # Chargement de la librairie
```

La première estimation favorise un arbre très détaillé c'est-à-dire avec un faible coefficient de pénalisation de la complexité de l'arbre et donc du nombre de feuilles important.

```
tree.reg=rpart(O3obs~.,data=datappr,
    control=rpart.control(cp=0.001))
summary(tree.reg)  # description de l'arbre ou encore
print(tree.reg)
plot(tree.reg)  # Tracé de l'arbre
text(tree.reg)  # Ajout des légendes des noeuds
```

Il est probable que l'arbre présente trop de feuilles pour une bonne prévision. Il est donc nécessaire d'en réduire le nombre par élagage. C'est un travail délicat d'autant que la documentation n'est pas très explicite et surtout les arbres des objets très instables.

Une première façon d'estimer l'erreur par validation croisée consiste à utiliser les fonctionnalités de la fonction `rpart` qui intègre la construction et le calcul d'erreurs pour toute une séquence de coefficients de pénalisation. Cela conduit au tracé (`plotcp`) de la décroissance de l'estimation de l'erreur relative (erreur divisée par la variance de la variable à modéliser) en fonction du coefficient de complexité, c'est-à-dire plus ou moins aussi aussi en fonction de la taille de l'arbre ou nombre de feuilles. Attention, cette relation entre complexité et nombre de feuilles n'est pas directe car l'erreur est calculée sur des arbres estimés à partir d'échantillons aléatoires ($k-1$ morceaux) différents et sont donc différents les uns des autres avec pas nécessairement le même nombre de feuilles, ils partagent juste le même paramètre de complexité au sein de la même famille de modèles emboîtés. Le choix optimal suggéré dans l'aide est la valeur du cp la plus à gauche en dessous de la ligne.

```
# tableau des valeurs de cp et erreurs relatives
printcp(tree.reg)
plotcp(tree.reg)
```

Comme la procédure intégrée de validation croisée est relativement "frustre" et surtout mal explicitée dans la documentation, une autre fonction est proposée (`xpred.rpart`) permettant de mieux contrôler la décroissance de la complexité. La commande suivante calcule les prévisions obtenues par 10-fold validation croisée pour chaque arbre élagué suivant les valeurs du coefficient

de complexité. La séquence de ces valeurs est implicitement celle fournit par `rpart`.

```
xmat=xpred.rpart(tree.reg)
xerr=(xmat-datappr[,"O3obs"])^2
apply(xerr,2,sum)
```

La valeur de `cp` optimale (pas nécessairement celle ci-dessous) est alors utilisée avant de tracer le graphe des résidus qui prend une forme particulière.

```
tree.reg=rpart(O3obs~.,data=datappr,
   control=rpart.control(cp=0.005568448))
plot(tree.reg)  # Tracé de l'arbre
text(tree.reg)  # Ajout des légendes des noeuds
# calcul et graphe des résidus
fit.tree=predict(tree.reg)
res.tree=fit.tree-datappr[,"O3obs"]
plot.res(fit.tree,res.tree)
```

Le contenu de l'arbre n'est pas très explicite ou alors il faudrait aller lire dans le `summary` les informations complémentaires. Un arbre plus détaillé est construit par l'intermédiaire d'un fichier postscript. Ce fichier est créé dans le répertoire de lancement de R. Différentes options sont disponibles permettant de gérer le titre et autres aspects du graphique : `?post`. Une autre solution pour obtenir un arbre interprétable consiste à utiliser la librairie `partykit`

```
post(tree.reg)
library(partykit)
plot(as.party(tree.reg))
```

2.1 Estimation et élagage d'un arbre de discrimination

Dans le cas d'une discrimination, le critère par défaut est l'indice de concentration de Gini ; il est possible de préciser le critère d'entropie ainsi que des poids sur les observations, une matrice de coûts ainsi que des probabilités a priori (`?rpart` pour plus de détails).

```
tree.dis=rpart(DepSeuil~.,data=datappq,
   parms=list(split='information'),cp=0.001)
plot(tree.dis)
text(tree.dis)
```

L'élagage fait appel à la même procédure intégrée de validation croisée que pour la régression :

```
printcp(tree.dis)
plotcp(tree.dis)
```

ou par validation croisée explicite :

```
xmat = xpred.rpart(tree.dis)
# Comparaison des valeurs prédite et observée
xerr=datappq$DepSeuil!= (xmat>1.5)
# Calcul et affichage des estimations des taux d'erreur
apply(xerr, 2, sum)/nrow(xerr)
```

Faire plusieurs exécutions en modifiant aussi la séquence de paramètres... Les choix sont-ils confirmés ? Faire un choix de pénalisation avant de réestimer l'arbre.

```
tree.dis=rpart(DepSeuil~.,data=datappq,
   parms=list(split='information'),cp=0.017930478 )
plot(tree.dis)
text(tree.dis)
post(tree.dis)
plot(as.party(tree.dis))
```

Remarquer quelles sont les variables sélectionnées par l'arbre, retrouve-t-on les mêmes que celles sélectionnées par la régression logistique.

Calculer l'erreur apparente ou par re-substitution mesurant la qualité de l'ajustement.

```
#matrice de confusion
table(predict(tree.dis,data=datappq,type='class'),
   datappq$DepSeuil)
```

2.2 Prévision de l'échantillon test

Différentes prévisions sont considérées assorties des erreurs estimées sur l'échantillon test. Prévision quantitative de la concentration, prévision de dé-

passement à partir de la prévision quantitative et directement la prévision de dépassement à partir de l'arbre de décision.

```
# Calcul des prévisions
pred.treer=predict(tree.reg,newdata=datestr)
pred.treeq=predict(tree.dis,newdata=datestq,
   type="class")
# Erreur quadratique moyenne de prévision
sum((pred.treer-datestr[,"O3obs"])^2)/nrow(datestr)
# Matrice de confusion pour la prévision du
# dépassement de seuil (régression)
table(pred.treer>150,datestr[,"O3obs"]>150)
# Même chose pour l'arbre de discrimination
table(pred.treeq,datestq[,"DepSeuil"])
```

Noter les taux d'erreur. Attention, ne plus modifier le modèle pour tenter de diminuer l'erreur sur l'échantillon test, cela conduirait à un biais par sur-ajustement.

3 Comparaison des courbes ROC

```
library(ROCR)
ROCregtree=pred.treer/300
predregtree=prediction(ROCregtree,datestq$DepSeuil)
perfregtree=performance(predregtree,"tpr","fpr")
ROCdistree=predict(tree.dis,
   newdata=datestq,type="prob")[,2]
preddistree=prediction(ROCdistree,datestq$DepSeuil)
perfdistree=performance(preddistree,"tpr","fpr")
# tracer les courbes ROC en les superposant
# pour mieux comparer
plot(perflogit,col=1)
plot(perfregtree,col=2,add=TRUE)
plot(perfdistree,col=3,add=TRUE)
```

Une méthode de prévisoin d'occurence de dépassement du pic de pollution est-elle globalement meilleure ?