Scénario: Prévision de dépassement du seuil d'ozone

Résumé

Comparaison sur le même jeu de données des qualités de prévision de plusieurs modèles. Il s'agit de prévoir la concentration du pic d'ozone par analyse de covariance, arbre de régression, réseau de neurones, agrégation de modèles, SVM, puis le dépassement ou directement le dépassement par régression logistique, analyse discriminante, arbre de discrimination, agrégation de modèles, SVM. Les distributions des estimations des erreurs de prévision et les courbes ROC sont comparées à la suite du tirage itératif d'un ensemble d'échantillons tests.

1 Introduction

1.1 Objectif

L'objectif, sur ces données, est d'améliorer la prévision déterministe (`MOCAGE`), calculée par les services de MétéoFrance, de la concentration d'ozone dans certaines stations de prélèvement. Il s'agit d'un problème dit d'*adaptation statistique* d'une prévision locale de modèles à trop grande échelle en s'aidant d'autre variables également prévues par MétéoFrance, mais à plus petite échelle (température, force du vent...). Plus précisément, deux variables peuvent être prévues : soit la concentration quantitative d'ozone, soit le dépassement (qualitatif) d'un certain seuil fixé à $150\mu g$ [1]. Dans chaque cas, deux approches sont considérées : soit prévoir la concentration quantitative puis en déduire l'éventuel dépassement ou bien prévoir directement le dépassement. Dans le premier cas, il s'agit d'abord d'une régression tandis que dans le deuxième il s'agit d'un problème de discrimination ou de régression logistique. Question : quelles sont les meilleures méthodes et stratégies pour prévoir la concentration d'ozone du lendemain d'une part et l'occurence d'un pic de pollution d'autre part.

On se propose de tester différentes méthodes : régression logistique, analyse discriminante, réseau de neurones, arbre de décision, agrégation d'arbres (bagging, boosting, random forest), SVM. L'objectif final, à ne pas perdre de vue, est la comparaison de ces méthodes afin de déterminer la plus efficace pour répondre au problème de prévision. Ceci passe par la mise en place d'un protocole très strict afin de s'assurer d'un minimum d'objectivité pour cette comparaison.

Penser à conserver dans des fichiers les commandes successives utilisées ainsi que les principaux résultats tableaux et graphes.

Toutes les opérations sont réalisées dans R avec l'appui de bibliothèques complémentaires éventuellement à télécharger (`mlbench, MASS, boot, class, e1071, rpart, nnet, ipred, gbm, randomForest`). D'autres logiciels sont bien évidemment utilisables (Splus, SAS, SPSS...). Néanmoins, dans le cas de SAS, cela nécessiterait l'accès au module Enterprise Miner pour certaines méthodes (arbres, neurones) encore peu diffusé car très cher. De plus certaines procédures du module classique SAS/Stat ne comportent pas d'option de sélection automatique de variables quantitatives et qualitatives.

1.2 Rappel : protocole de comparaison

La démarche mise en œuvre enchaîne les étapes suivantes :

i. Après une éventuelle première étape descriptive uni ou multidimensionnelle visant à repérer les incohérences, les variables non significatives ou de distribution exotique, les individus non concernés... et à étudier les structures des données, procéder à un tirage aléatoire d'un échantillon *test* qui ne sera utilisé que lors de la *dernière étape*.

ii. Sur la partie restante qui sera découpée en échantillon d'*apprentissage (des paramètres du modèle) et échantillon de* validation (pour estimation sans biais du taux de mauvais classés), optimiser les choix afférents à chacune des méthodes de régression et/ou discrimination :
   - variables et interactions à prendre en compte dans la régression linéaire ou logistique,
   - variables et méthode pour l'analyse discriminante,

1. Le seuil légal a été porté à $180\mu g$ mais celui-ci est bien trop rarement dépassé par les observations et serait trop difficile à prévoir par estimation d'un modèle.

- nombre de nœuds dans l'arbre de régression ou de classification,
- architecture (nombre de couches, de neurones par couche, fonctions de transferts, nombre de cycles...) du perceptron,
- algorithme d'agrégation,
- noyau et complexité des SVMs.

Remarques :

- En cas d'échantillon petit face au nombre des variables il est recommandé d'itérer la procédure de découpage par validation croisée, c'est le cas de ces données, afin de réduire la variance des estimations des erreurs de classement.

iii. Comparaison finale des qualités de prévision sur la base du taux de mal classés pour le seul échantillon test qui est resté à l'écart de tout effort ou "acharnement" pour l'optimisation des modèles.

- *Attention, ne pas "tricher" en modifiant le modèle obtenu lors de l'étape précédente afin d'améliorer le résultat sur l'échantillon test !*
- Le critère utilisé dépend du problème : erreur quadratique, taux de mauvais classement, AUC (aire sous la courbe ROC)...

1.3 Prise en charge des données

Les données ont été extraites et mises en forme par le service concerné de MétéoFrance. Elles sont disponibles sur la plateforme moodle dans le fichier `ozone.dat` et décrites par les variables suivantes :

JOUR Le type de jour ; férié (1) ou pas (0) ;

O3obs La concentration d'ozone effectivement observée le lendemain à 17h locales correspondant souvent au maximum de pollution observée ;

MOCAGE Prévision de cette pollution obtenue par un modèle déterministe de mécanique des fluides (équation de Navier et Stockes) ;

TEMPE Température prévue par MétéoFrance pour le lendemain 17h ;

RMH2O Rapport d'humidité ;

NO2 Concentration en dioxyde d'azote ;

NO Concentration en monoxyde d'azote ;

STATION Lieu de l'observation : Aix-en-Provence, Rambouillet, Munchhausen, Cadarache et Plan de Cuques ;

VentMOD Force du vent ;

VentANG Orientation du vent.

```
# Lecture des données
ozone=read.table("ozone.dat",header=T)
# Vérification du contenu
summary(ozone)
# Changement du type de la variable jour
ozone[,"JOUR"]=as.factor(ozone[,"JOUR"])
```

Remarquer le type des variables. Il est nécessaire d'en étudier la distribution.

```
hist(ozone[,"O3obs"]);hist(ozone[,"MOCAGE"])
hist(ozone[,"TEMPE"]);hist(ozone[,"RMH2O"])
hist(ozone[,"NO2"]);hist(ozone[,"NO"])
hist(ozone[,"VentMOD"]);hist(ozone[,"VentANG"])
```

Des transformations sont proposées pour rendre certaines distributions plus symétriques et ainsi plus "gaussiennes".

```
ozone[,"SRMH2O"]=sqrt(ozone[,"RMH2O"])
ozone[,"LNO2"]=log(ozone[,"NO2"])
ozone[,"LNO"]=log(ozone[,"NO"])
```

Vérifier de l'opportunité de ces transformations puis retirer les variables initiales et construire la variable "dépassement de seuil".

```
ozone=ozone[,c(1:4,8:13)]
ozone[,"DepSeuil"]=as.factor(ozone[,"O3obs"]>150)
```

pour obtenir le fichier qui sera effectivement utilisé.

1.4 Extraction des échantillons apprentissage et test

Le programme ci-dessous réalise l'extraction du sous-ensemble des données d'apprentissage et de test. Attention, chaque étudiant ou binôme tire un échantillon différent ; il est donc "normal" de na pas obtenir les mêmes modèles.

Utiliser trois chiffres au hasard, en remplacement de "XXX" ci-dessous, comme initialisation du générateur de nombres aléatoires.

```
set.seed(XXX) # initialisation du générateur
# Extraction des échantillons
test.ratio=.2   # part de l'échantillon test
npop=nrow(ozone) # nombre de lignes dans les données
nvar=ncol(ozone) # nombre de colonnes
# taille de l'échantillon test
ntest=ceiling(npop*test.ratio)
# indices de l'échantillon test
testi=sample(1:npop,ntest)
# indices de l'échantillon d'apprentissage
appri=setdiff(1:npop,testi)
```

Construction des échantillons pour la régression (prévision de la concentration).

```
# construction de l'échantillon d'apprentissage
datappr=ozone[appri,-11]
# construction de l'échantillon test
datestr=ozone[testi,-11]
summary(datappr) # vérifications
summary(datestr)
```

Construction des échantillons pour la discrimination (dépassement du seuil).

```
# construction de l'échantillon d'apprentissage
datappq=ozone[appri,-2]
# construction de l'échantillon test
datestq=ozone[testi,-2]
summary(datappq) # vérifications
summary(datestq)
```

2 Prévison par modèle gaussien

Le premier modèle à tester est un simple modèle de régression linéaire mais, comme certaines variables sont qualitatives, il s'agit d'une analyse de covariance. D'autre part, on s'intéresse à savoir si des interactions sont à prendre en compte. Le modèle devient alors polynomiale d'ordre 2 ou quadratique.

2.1 Modèle linéaire

Le modèle de régression linéaire simple intégrant des variables qualitatives correspondant est estimé avec la fonction `aov` mieux adaptée à l'analyse de covariance.

```
# estimation du modèle sans interaction
reg.lm=aov(O3obs~.,data=datappr)
summary(reg.lm)
# Extraction des résidus et des valeurs ajustées
res.lm=reg.lm$residuals
fit.lm=reg.lm$fitted.values
# Graphe des résidus
plot(fit.lm,res.lm)
# Définition d'une fonction pour un graphe coloré et
# des échelles fixes sur les axes
plot.res=function(x,y,titre="titre")
{
plot(x,y,col="blue",xlim=c(0,300),ylim=c(-100,100),
ylab="Résidus",xlab="Valeurs predites",main=titre)
# points(x2,y,col="red")
abline(h=0,col="green")
}
plot.res(fit.lm,res.lm,"")
# Graphe des résidus au modèle MOCAGE
plot.res(datappr[,"MOCAGE"],
   datappr[,"MOCAGE"]-datappr[,"O3obs"],"")
```

Contrôler les graphes des résidus, que conclure de ce modèle ? L'étude suivante met en œuvre toutes les interactions d'ordre 2 pour ajuster le modèle :

2.2 Modèle quadratique

Le modèle de régression quadratique est estimé avec la fonction `glm` qui permet une sélection automatique de modèle. La méthode descendante est utilisée mais celle pas-à-pas pourrait également l'être.

```
# Estimation du modèle de toute interaction d'ordre 2
```

```
reg.glm=glm(O3obs~(.)^2,data=datappr)
# Recherche du meilleur modèle au sens
# du critère d'Akaïke par méthode descendante
reg.glm.step=step(reg.glm,direction="backward")
anova(reg.glm.step,test="F")
# Extraction des valeurs ajustées et des résidus
fit.glm=reg.glm.step$fitted.values
res.glm=reg.glm.step$residuals
# Graphe des résidus
plot.res(fit.glm,res.glm,"")
```

On remarque que la présence de certains interactions ou variables sont pertinentes au sens du critère d'Akaïke mais pas significative au sens du test de Fisher. Cette présence dans le modèle peut être plus finement analysée en considérant une estimation de l'erreur par validation croisée. L'idée est de retirer une à une les variables ou interactions les moins significatives et de voir comment se comporte la validation croisée. D'autre part, si la procédure pas-à-pas conduit à un modèle différent, l'estimation de l'erreur par validation croisée permet également d'optimiser le choix.

L'estimation des erreurs par validation croisée est calculée en utilisant une fonction proposée dans la bibliothèque `boot`.

```
library(boot) # chargement de la bibliothèque
# validation croisée  10-plis
# modèle complet
cv.glm(datappr, reg.glm, K=10)$delta[1]
# modèle "Akaïke"
cv.glm(datappr, reg.glm.step, K=10)$delta[1]
```

La première commande suivante permet de récupérer le modèle optimal avant de redéfinir celui-ci en retirant l'interaction la moins significative avant de réestimer l'erreur par validation croisée.

```
#la liste des effets du modèle optimal
reg.glm.step$formula
cv.glm(datappr, glm(O3obs ~ JOUR + MOCAGE + TEMPE +
STATION + VentMOD + VentANG + LNO2 + LNO + SRMH2O +
JOUR:VentMOD + JOUR:VentANG + MOCAGE:STATION +
MOCAGE:VentMOD + MOCAGE:VentANG + MOCAGE:LNO2 +
MOCAGE:LNO +     MOCAGE:SRMH2O + TEMPE:STATION +
TEMPE:VentMOD + TEMPE:LNO2 +     TEMPE:LNO +
TEMPE:SRMH2O + STATION:VentMOD + STATION:LNO2 +
STATION:LNO + STATION:SRMH2O + VentMOD:LNO2 +
VentMOD:LNO +     VentMOD:SRMH2O + VentANG:LNO2 +
VentANG:LNO+LNO:SRMH2O,data=datappr),K=10)$delta[1]
```

2.3 Prévision de l'échantillon test

Retenir le meilleur modèle obtenu pour prédire l'échantillon test et estimer ainsi sans biais une erreur de prévision. Deux erreurs sont estimées ; la première est celle quadratique pour la régression tandis que la deuxième est issue de la matrice de confusion qui croise les dépassements de seuils prédits avec ceux effectivement observés.

```
# Calcul des prévisions
pred.glm=predict(reg.glm.step,newdata=datestr)
# Erreur quadratique moyenne de prévision
sum((pred.glm-datestr[,"O3obs"])^2)/nrow(datestr)
# Matrice de confusion pour la prévision du
# dépassement de seuil
table(pred.glm>150,datestr[,"O3obs"]>150)
```

Noter ces erreurs pour les comparer avec celles obtenues par les autres méthodes. Par exemple avec la prévision de MOCAGE.

```
# matrice de confusion du modèle MOCAGE
table(datestq[,"MOCAGE"]>150,datestq[,"DepSeuil"])
```

3 Prévision par modèle binomial

Plutôt que de prévoir la concentration puis le dépassement, on peut se poser la question de savoir s'il ne serait pas pertinent de prévoir directement la présence ou l'absence d'un dépassement. La variable à modéliser étant binaire,

c'est la régression logistique qui va être employée. Comme pour la régression, différentes stratégies de choix de modèle peuvent être utilisées et comparées avant d'estimer l'erreur de prévision sur l'échantillon test.

3.1 Régression logistique sans interaction

```
# estimation du modèle complet
log.lm=glm(DepSeuil~.,data=datappq,family=binomial)
# significativité des paramètres
anova(log.lm,test="Chisq")
# Recherche d'un modèle optimal au sens d'Akaïke
log.lm.step=step(log.lm,direction="backward")
anova(log.lm.step,test="Chisq")
# matrice de confusion de l'échantillon
# d'apprentissage et erreur apparente
table(log.lm.step$fitted.values>0.5,
   datappq[,"DepSeuil"])
```

3.2 Régression logistique avec interaction

```
# estimation du modèle complet
log.qm=glm(DepSeuil~(.)^2,data=datappq,
   family=binomial)
```

Avec autant de paramètres, l'algorithme de régression peut rencontrer quelques soucis si la régression ajuste "trop bien" les données. On propose alors une optimisation du modèle par une méthode pas-à-pas à comparer avec le modèle fournit par l'algorithme descendant.

```
# régression avec le modèle minimum
log.qm=glm(DepSeuil~1,data=datappq,family=binomial)
# algorithme stepwise en précisant le plus grand
# modèle possible
log.qm.step1=step(log.qm,direction="both",
  scope=list(lower=~1,upper=~(JOUR + MOCAGE +
  TEMPE + STATION + VentMOD + VentANG + LNO2 +
  LNO + SRMH2O)^2), family=binomial)
# significativité des paramètres
anova(log.qm.step1,test="Chisq")

# algorithme backward
log.qm=glm(DepSeuil~(.)^2,data=datappq,
   family=binomial)
log.qm.step2=step(log.qm,direction="backward",
   family=binomial)
# significativité des paramètres
anova(log.qm.step2,test="Chisq")
```

Deux modèles sont en concurrence, il s'agit de les comparer en minimisant l'erreur calculée par validation croisée. La même procédure est utilisée que pour le modèle linéaire classique mais une autre fonction de coût adaptée à une variable binaire doit être précisée. Elle est définie ci-dessous puis appliquée au calcul de l'erreur apparente de prévision.

```
cout = function(r, pi=0)
   mean(abs(as.integer(r)-pi)>0.5)
# Application de cette fonction pour
# l'erreur apparente ou d'ajustement :
cout(datappq$DepSeuil,log.qm.step1$fitted.values)
cout(datappq$DepSeuil,log.qm.step2$fitted.values)
```

Estimation de l'erreur de prévision par validation croisée.

```
library(boot)
cv.glm(datappq, log.qm.step1, cout,K=10)$delta[1]
cv.glm(datappq, log.qm.step2, cout,K=10)$delta[1]
```

Retenir le "meilleur" modèle.

3.3 Prévision de l'échantillon test

```
pred.log=predict(log.qm.step1,newdata=datestq)
# Matrice de confusion pour la prévision du
# dépassement de seuil
table(pred.log>0.5,datestq[,"DepSeuil"])
```

Mémoriser les résultats obtenus pour comparer avec les autres méthodes.

3.4 Courbe ROC

```
library(ROCR)
roclogit=predict(log.qm.step1,newdata=datestq,
   type="response")
predlogit=prediction(roclogit,datestq[,"DepSeuil"])
perflogit=performance(predlogit, "tpr","fpr")
plot(perflogit,col=1)
```

Il est également possible de construire une courbe ROC en association de la prévision obtenue à partir d'un modèle gaussien. En effet, la variation du seuil théorique de dépassement (150) va faire varier les proportions respectives des taux de vrais et faux positifs. Cela revient encore à faire varier le seuil d'une "proba" pour les valeurs de prévisions divisées par 300.

```
rocglm=pred.glm/300
predglm=prediction(rocglm,datestq[,"DepSeuil"])
perfglm=performance(predglm, "tpr","fpr")
plot(perfglm,col=2,add=TRUE)
```

Les résultats obtenus dépendent évidemment en plus de l'échantillonnage initial entre appentissage et test. Dans le cas où les courbes se croisent, cela signifie qu'il n'y a pas de prévision uniformément meilleure de l'occurence de dépassement. Cela dépend de la sensibilité ou de la spécificité retenue pour le modèle. Ceci souligne l'importance de la bonne définition du critère à utiliser pour le choix d'une "meilleure" méthode. Ce choix dépend directement de celui , "politique" ou "économique" de sensibilité et / ou spécificité du modèle retenu. En d'autres termes, quel taux de fausse alerte, avec des imputations économiques évidentes, est supportable au regard des dépassements non détectés et donc de la dégradation sanitaire de la population à risque ?

C'est une fois ce choix arrêté que le statisticien peut opérer une comparaison des méthodes en présence.

4 Analyse discriminante

4.1 Précautions

L'objectif est de comparer les trois méthodes d'analyses discriminantes disponibles dans R : `lda` paramétrique linéaire (homoscédasticité), `lqa` paramétrique quadratique (hétéroscédasticité) sous hypothèse gaussienne et celle non-paramétrique des k plus proches voisins.

Attention, ces techniques n'acceptent par principe que des variables explicatives ou prédictives quantitatives. Néanmoins, une variable qualitative à deux modalités, par exemple le type de jour, peut être considérée comme quantitative sous la forme d'une fonction indicatrice prenant ses valeurs dans $\{0, 1\}$ et, de façon plus "abusive", une variable ordinale est considérée comme "réelle". Dans ce dernier cas, il ne faut pas tenter d'interpréter les fonctions de discrimination, juste considérer des erreurs de prévision. La variable `Station` n'est pas prise en compte.

La bibliothèque standard de R (MASS) pour l'analyse discriminante ne propose pas de procédure automatique de choix de variable contrairement à la procédure `stepdisc` de SAS mais, dans cet exemple, les variables sont peu nombreuses.

4.2 Estimations des modèles

```
library(MASS) # chargement des librairies
library(class) # pour kNN

# analyse discriminante linéaire
disc.lda=lda(DepSeuil~.,data=datappq[,-4])
# analyse discriminante quadratique
disc.qda=qda(DepSeuil~.,data=datappq[,-4])
# k plus proches voisins
disc.knn=knn(datappq[,c(-4,-10)],datappq[,c(-4,-10)],
   datappq$DepSeuil,k=10)

# erreur apparente de prévision
table(datappq[,"DepSeuil"],
    predict(disc.lda,datappq)$class)
table(datappq[,"DepSeuil"],
    predict(disc.qda,datappq)$class)
```

```
table(datappq[,"DepSeuil"],disc.knn)
```

Noter le manque d'homogénéité des commandes de R issues de librairies différentes. L'indice de colonne négatif (-10) permet de retirer la colonne contenant la variable à prédire de type facteur. Celle-ci est mentionnée en troisième paramètre pour les données d'apprentissage.

4.3 Estimatin de l'erreur par validation croisée

Pour `knn`, le choix du nombre de voisins k peut être optimisé par validation croisée mais la procédure proposée par la bibliothèque `class` est celle *leave-one-out*, donc trop coûteuse en calcul pour des gros fichiers. Il serait simple de la programmer mais une autre bibliothèque (`e1071`) propose déjà une batterie de fonctions de validation croisée pour de nombreuses techniques de discrimination.

```
# erreur par validation croisée
# analyse discriminante linéaire
disc.lda=lda(DepSeuil~.,data=datappq[,-4],CV=T)
# analyse discriminante quadratique
disc.qda=qda(DepSeuil~.,data=datappq[,-4],CV=T)

# estimer le taux d'erreur à partir des
# matrices de confusion
table(datappq[,"DepSeuil"],disc.lda$class)
table(datappq[,"DepSeuil"],disc.qda$class)

# k plus proches voisins
library(e1071)
plot(tune.knn(as.matrix(datappq[,c(-4,-10)]),
   as.factor(datappq[,10]),k=2:20))
```

Remarquer que chaque exécution de la commande précédente donne des résultats différents donc très instables. Faire plusieurs exécutions et déterminer une valeurs "raisonnables" de k.

Comparer le taux d'erreur minima obtenu avec ceux, toujours estimés par validation croisée, des autres versions d'analyse dicriminante.

4.4 Erreur sur l'échantillon test

Les commandes suivantes calculent la matrice de confusion pour la "meilleure" méthode d'analyse discriminante au sens de la validation croisée.

```
# analyse discriminante linéaire
disc.lda=lda(DepSeuil~.,data=datappq[,-4])
table(datestq[,"DepSeuil"],
    predict(disc.lda,datestq[,-4])$class)
```

A titre indicatif, voici l'estimation de l'erreur sur l'échantillon test pour la méthode des k plus proches voisins.

```
disc.knn=knn(as.matrix(datappq[,c(-4,-10)]),
    as.matrix(datestq[,c(-4,-10)]),
    datappq$DepSeuil,k=15)
table(disc.knn,datestq$DepSeuil)
```

Noter le meilleur taux d'erreur en vue d'une comparaison.

4.5 Comparaison des courbes ROC

```
library(ROCR)
ROCdiscrim=predict(disc.lda,
   datestq[,c(-4)])$posterior[,2]
preddiscrim=prediction(ROCdiscrim,datestq$DepSeuil)
perfdiscrim=performance(preddiscrim,"tpr","fpr")
# tracer les ocurbes ROC en les superposant
# pour mieux comparer
plot(predlogit,col=1)
plot(perflogistic,col=1,add=TRUE)
plot(perfdiscrim,col=2,add=TRUE)
```

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

5 Arbres binaires de décision

5.1 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).

5.2 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 surotut mal explicité 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` ou peut être précisée.

```
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`.

```
post(tree.reg)
```

5.3 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,xval=10,
   cp=seq(0.1,0.001,length=30))
xmat = xpred.rpart(tree.dis,xval=10)
# 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)
```

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)
```

5.4 Prévision de l'échantillon test

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

5.5 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 ?

6 Réseaux de neurones

6.1 Introduction

Il s'agit d'estimer un modèle de type "perceptron" avec en entrée les variables qualitatives ou quantitatives et en sortie la variable à prévoir. Des fonctions R pour l'apprentissage d'un perceptron élémentaire ont été réalisées par différents auteurs et sont accessibles sur le réseau. La librairie `nnet` de (Ripley, 1999), est limitée aux perceptrons à une couche. C'est théoriquement suffisant pour approcher d'aussi près que l'on veut toute fonction à condition d'insérer suffisamment de neurones.

Comme pour les arbres, la variable à expliquer est soit quantitative soit qualitative ; la fonction de transfert du neurone de sortie d'un réseau doit être adaptée en conséquence. Elle est choisie linéaire dans le cas quantitatif et sigmoïdale (choix par défaut) comme toutes celles de la couche cachée dans le cas qualitatif. Le paramètre important à déterminer est le nombre de neurones sur la couche cachée parallèlement aux conditions d'apprentissage (temps ou nombre de boucles). Une alternative à la détermination du nombre de neurones est celle du `decay` qui est un paramètre de régularisation analogue à celui utilisé en régression *ridge*. Il pénalise la norme du vecteurs des paramètres et contraint ainsi la flexibilité du modèle. Très approximativement il est d'usage de considérer, qu'en moyenne, il faut une taille d'échantillon d'apprentissage 10 fois supérieure au nombre de poids c'est-à-dire au nombre de paramètres à estimer. On remarque qu'ici la taille de l'échantillon d'apprentissage (832) est modeste pour une application raisonnable du perceptron. Seuls des nombres restreints de neurones peuvent être considérés et sur une seule couche cachée.

6.2 Cas de la régression

```
library(MASS)
library(nnet)
# apprentissage
nnet.reg=nnet(O3obs~.,data=datappr,size=5,decay=1,
   linout=TRUE,maxit=500)
summary(nnet.reg)
```

La commande donne la "trace" de l'exécution avec le comportement de la convergence mais le détail des poids de chaque entrée de chaque neurone ne constituent pas des résultats très explicites ! Contrôler le nombre de poids estimés.

L'optimisation des paramètres nécessite encore le passage par la validation croisée. Il n'y a pas de fonction dans la librairie `nnet` permettant permettant de le faire mais la fonction `tune.nnet` de la librairie `e1071` est adaptée à cette démarche.

```
library(e1071)
plot(tune.nnet(O3obs~.,data=datappr,size=c(2,3,4),
   decay=c(1,2,3),maxit=200,linout=TRUE))
plot(tune.nnet(O3obs~.,data=datappr,size=4:5,decay=1:10))
```

Noter la taille et le "decay" optimaux. Il faudrait aussi faire varier le nombre total d'itérations. Cela risque de prendre un peu de temps ! Noter également que chaque exécution donne des résultats différents... il n'est donc pas très utile d'y passer beaucoup de temps, surtout que les temps d'exécution sont assez long !

Ré-estimer le modèle supposé optimal avant de tracer le graphe des résidus.

```
nnet.reg=nnet(O3obs~.,data=datappr,size=3,decay=2,
   linout=TRUE,maxit=200)
# calcul et graphe des résidus
fit.nnetr=predict(nnet.reg,data=datappr)
res.nnetr=fit.nnetr-datappr[,"O3obs"]
plot.res(fit.nnetr,res.nnetr)
```

6.3 Cas de la discrimination

```
# apprentissage
nnet.dis=nnet(DepSeuil~.,data=datappq,size=5,decay=1)
summary(nnet.reg)
```

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

```
#matrice de confusion
table(nnet.dis$fitted.values>0.5,datappq$DepSeuil)
```

La validation croisée est toujours nécessaire afin de tenter d'optimiser les choix en présence : nombre de neurones, *decay* et éventuellement le nombre max d'itérations. Néanmoins, la fonction `tune.nnet` pose des problèmes dans le cas de la discrimination. Voici donc, à titre pédagogique, le script d'une fonction de k-validation croisée adaptée aux réseaux de neurones.

i. Ouvrir l'éditeur pour la création de la fonction : `fix(CVnn, editor="emacs"))` avec l'éditeur emacs ou un autre sous Linux (kwrite, kate...) ; L'éditeur par défaut étant vi. Utiliser l'éditeur par défaut sous Windows.

ii. Rentrer le texte de la fonction. Les espaces ne sont pas importants, la mise en page est automatique.

```
function(formula, data, size, niter = 1,
   nplis = 10, decay = 0, maxit = 100)
{
   n = nrow(data)
   tmc=0
   un = rep(1, n)
   ri = sample(nplis, n, replace = T)
   cat(" k= ")
   for(i in sort(unique(ri))) {
     cat(" ", i, sep = "")
     for(rep in 1:niter) {
       learn = nnet(formula, data[ri != i,  ],
         size = size,trace = F, decay = decay,
         maxit = maxit)
       tmc = tmc + sum(un[(data$DepSeuil[ri == i]
         == "TRUE") != (predict(learn, data[ri == i,
         ]) > 0.5)])
       }
   }
   cat("\n", "Taux de mal classes")
   tmc/(niter * length(unique(ri)) * n)
}
```

iii. Sauver le texte, quitter l'éditeur

iv. Si des messages d'erreur apparaissent, `CVnn=edit(editor="emacs")` permet de relancer l'éditeur pour les corriger.

Le paramètre `niter` permet de répliquer l'apprentissage du réseau et de moyenner les résultats afin de réduire la variance de l'estimation de l'erreur. Il est par défaut de 1 mais peut-être augmenté à condition de se montrer plus patient. R, langage interprété comme Matlab, n'est pas particulièrement véloce lorsque plusieurs boucles sont imbriquées. Attention, cette fonction dépend des

données utilisées (code TRUE des modalités) mais elle pourrait être facilement généralisée et adaptées à tout type de modélisation.

Tester la fonction ainsi obtenue et l'exécuter pour différentes valeurs de `size` comme (5, 6, 7) et `decay` (0, 1, 2). Il semble plus simple de fixer une valeur un peu grande du nombre de neurones (7) et de ne faire varier que le paramètre de decay entr 0 et 5 avec plusieurs exétcutions pour chaque valeur de ce paramètre. En effet, l'initialisation de l'apprentissage d'un réseau de neurone comme celle de l'estimation de l'erreur par validation croisée sont aléatoires. Chaque exécution donne donc des résultats différents. À ce niveau, il serait intéressant de construire un plan d'expérience à deux facteurs (ici, les paramètres de taille et *decay*) de chacun trois niveaux. Plusieurs réalisations pour chaque combinaison des niveaux suivies d'un test classique d'anova permettraient de se faire une idée plus juste de l'influence de ces facteurs sur l'erreur.

```
CVnn(DepSeuil~.,data=datappq,size=7, decay=0)
...
# exécuter pour différentes valeur du decay
```

Noter la taille et le *decay* optimaux et ré-estimer le modèle pour ces valeurs.

6.4 Prévision de l'échantillon test

Différentes prévisions sont considérées assorties des erreurs estimées sur l'échantillon test. Prévision quantitative de la concentration, prévision de dépassement à partir de la prévision quantitative et directement la prévision de dépassement à partir de l'arbre de décision.

```
# Calcul des prévisions
pred.nnetr=predict(nnet.reg,newdata=datestr)
pred.nnetq=predict(nnet.dis,newdata=datestq)
# Erreur quadratique moyenne de prévision
sum((pred.nnetr-datestr[,"O3obs"])^2)/nrow(datestr)
# Matrice de confusion pour la prévision du
# dépassement de seuil (régression)
table(pred.nnetr>150,datestr[,"O3obs"]>150)
# Même chose pour la discrimination
table(pred.nnetq>0.5,datestq[,"DepSeuil"])
```

Noter les taux d'erreur.

6.5 Comparaison des courbes ROC

```
library(ROCR)
rocnnetr=pred.nnetr/300
prednnetr=prediction(rocnnetr,datestq$DepSeuil)
perfnnetr=performance(prednnetr,"tpr","fpr")
rocnnetq=pred.nnetq
prednnetq=prediction(rocnnetq,datestq$DepSeuil)
perfnnetq=performance(prednnetq,"tpr","fpr")
# tracer les courbes ROC en les superposant
# pour mieux comparer
plot(perflogit,col=1)
plot(perfnnetr,col=2,add=TRUE)
plot(perfnnetq,col=3,add=TRUE)
```

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

7 Agrégation de modèles

7.1 Introduction

Des sections précédentes ont permis d'expérimenter les techniques maintenant classiques de construction d'un modèle de prévision assorties du problème récurrent liés à l'optimisation de la complexité du modèle. Cette section aborde d'autres stratégies dont l'objectif est de s'affranchir de ce problème de choix, par des méthodes se montrant pas ou très peu sensibles au sur-apprentissage ; c'est le cas des algorithmes d'agrégation de modèles.

Sur le plan logiciel, R montre dans cette situation tout son intérêt. La plupart des techniques récentes sont en effet expérimentées avec cet outil et le plus souvent mises à disposition de la communauté scientifique sous la forme d'une librairie afin d'en assurer la "promotion". Pour les techniques d'agrégation de modèles, nous pouvons utiliser les librairies `gbm` et `randomForest` respectivement réalisées par Greg Ridgeway et Leo Breiman. Ce n'est pas systématique, ainsi J. Friedman a retiré l'accès libre à ses fonctions (MART) et

créé son entreprise (Salford).

Les objectifs de cette section sont de

i. tester le *bagging* et le choix des ensembles de variables ainsi que le nombre d'échantillons considérés,
ii. étudier l'influence des paramètres (profondeur d'arbre, nombre d'itérations, *shrinkage*) sur la qualité de la prévision par *boosting* ;
iii. même chose pour les forêts aléatoires (nb de variables tirées (`mtry`à, `nodesize`).
iv. Expérimenter les critères de Breiman qui permettent de mesurer l'influence des variables au sein d'une famille agrégée de modèles. Les références bibliographiques sont accessibles sur le site de l'auteur :
`www.stat.Berkeley.edu/users/breiman`

7.2 Bagging

En utilisant la fonction `sample` de R, il est très facile d'écrire un algorithme de *bagging*. Il existe aussi une librairie qui propose des exécutions plus efficaces. Par défaut, l'algorithme construit une famille d'arbres complets (`cp=0`) et donc de faible biais mais de grande variance. L'erreur *out-of-bag* permet de contrôler le nombre d'arbres ; un nombre raisonnable semble suffire ici.

Régression

L'utilisation est immédiate :

```
library(ipred)
bagging(O3obs~., nbag=50, data=datappr,coob=TRUE)
bagging(O3obs~., nbag=25, data=datappr,coob=TRUE)
```

Discrimination

```
bagging(DepSeuil~., nbag=50, data=datappq,coob=TRUE)
bagging(DepSeuil~., nbag=25, data=datappq,coob=TRUE)
```

Les résultats sont aléatoires et différents pour chaque exécution. Néanmoins, ceux-ci semblent relativement stables et peu sensibles au nombre d'arbres. Enfin, il est possible de modifier certaines paramètres de `rpart` pour juger de leur influence : ci-dessous, l'élagage de l'arbre.

```
bagging(O3obs~., nbag=50,control=rpart.control(cp=0.1),
   data=datappr,coob=TRUE)
```

Cela nécessite une optimisation du choix du paramètre. Remarquer néanmoins que le nombre d'arbres (`nbag`) n'est pas un paramètre "sensible" et qu'il suffit de se contenter d'arbres "entiers".

Enfin, considérer le choix optimal de la régression au sens de l'erreur `oob` comme une estimation par validation croisée puis tracer le graphe des résidus.

```
bag.reg=bagging(O3obs~., nbag=50, data=datappr,
   coob=TRUE)
# calcul et graphe des résidus
fit.bagr=predict(bag.reg)
res.bagr=fit.bagr-datappr[,"O3obs"]
plot.res(fit.bagr,res.bagr)
```

Avec le choix optimal pour la discrimination, calculer l'erreur apparente ou par re-substitution mesurant la qualité de l'ajustement.

```
bag.dis=bagging(DepSeuil~., nbag=50, data=datappq,
   coob=TRUE)
fit.bagq=predict(bag.dis,type="class")
#matrice de confusion
table(fit.bagq,datappq$DepSeuil)
```

Prévision de l'échantillon test

Avec les meilleures combinaisons de paramètres précédentes, estimer les erreurs sur l'échantillon test.

```
bag.reg=bagging(O3obs~., nbag=50, data=datappr)
bag.dis=bagging(DepSeuil~., nbag=50, data=datappq)
pred.bagr=predict(bag.reg,newdata=datestr)
pred.bagq=predict(bag.dis,newdata=datestq,
   type="class")
# Erreur quadratique moyenne de prévision
sum((pred.bagr-datestr[,"O3obs"])^2)/nrow(datestr)
```

```
# Matrice de confusion pour la prévision
#  du dépassement de seuil (régression)
table(pred.bagr>150,datestr[,"O3obs"]>150)
# Même chose pour la discrimination
table(pred.bagq,datestq[,"DepSeuil"])
```

Noter les taux d'erreur.

7.3 Forêt aléatoire

Le programme est disponible dans la librairie `randomForest`. Il est écrit en fortran, donc efficace en terme de rapidité d'exécution, et facile à utiliser grâce à une interface avec R. Les paramètres et sorties sont explicités dans l'aide en ligne.

Régression

L'utilisation est encore immédiate :

```
library(randomForest)
rf.reg=randomForest(O3obs~., data=datappr,
   xtest=datestr[,-2],ytest=datestr[,"O3obs"],
   ntree=500,do.trace=50,importance=TRUE)
```

Discrimination

```
rf.dis=randomForest(DepSeuil~.,
   data=datappq,xtest=datestq[,-10],ytest=datestq[,
   "DepSeuil"],ntree=500,do.trace=50,importance=TRUE)
```

Il peut être tenté une optimisation des paramètres `mtry` (nombre de variables tirés à chaque nœud mais cela n'apparaît pas comme indispensable, la technique est suffisamment robuste.

```
# calcul et graphe des résidus
fit.rfr=rf.reg$predicted
res.rfr=fit.rfr-datappr[,"O3obs"]
plot.res(fit.rfr,res.rfr)
```

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

```
fit.rfq=rf.dis$predicted
table(fit.rfq,datappq$DepSeuil) #matrice de confusion
```

Prévision de l'échantillon test

Estimer les erreurs sur l'échantillon test.

```
pred.rfr=rf.reg$test$predicted
pred.rfq=rf.dis$test$predicted
# Erreur quadratique moyenne de prévision
sum((pred.rfr-datestr[,"O3obs"])^2)/nrow(datestr)
# Matrice de confusion pour la prévision du
# dépassement de seuil (régression)
table(pred.rfr>150,datestr[,"O3obs"]>150)
# Même chose pour la discrimination
table(pred.rfq,datestq[,"DepSeuil"])
```

Comparer les erreurs.

Importances des variables

Le modèle obtenu est ininterprétable mais des coefficients estiment les contributions des variables dans leur participation à la discrimination.

```
sort(round(importance(rf.reg), 2)[,1])
sort(round(importance(rf.dis), 2)[,4])
```

Comparer avec les variables sélectionnées par les autres modèles.

7.4 Boosting

Deux librairies proposent des versions relativement sophistiquées des algorithmes de *boosting* dans R. La librairie `boost` propose 4 approches : `adaboost, bagboost` et deux `logitboost`. Développées pour une problématique particulière : l'analyse des données d'expression génomique, elle n'est peut-être pas complètement adaptée aux données étudiées ; elles se limitent à des prédicteurs quantitatifs et peut fournir des résultats étranges. La

librairie gbm lui est préférée ; elle offre aussi plusieurs versions dépendant de la fonction coût choisie.

La variable à prévoir doit être codée numériquement (0,1) pour cette implémentation. Le nombre d'itérations, ou nombre d'arbres, est paramétré ainsi qu'un coefficient de rétrécissement (*shrinkage*) contrôlant le taux ou pas d'apprentissage. Attention, par défaut, ce paramètre a une valeur très faible (0.001) et il faut un nombre important d'itérations (d'arbres) pour atteindre une estimation raisonnable. La qualité est visualisée par un graphe représentant l'évolution de l'erreur d'apprentissage. D'autre part, une procédure de validation croisée est incorporée ; elle fournit un nombre optimal d'itérations à considérer. Des détails sur cette procédures sont disponibles dans le fichier :

```
library(gbm)
vignette(gbm) # descriptif détaillé de la librairie
```

Régression

```
boost.reg=gbm(O3obs~., data=datappr,
  distribution="gaussian",n.trees=500, cv.folds=10,
  n.minobsinnode = 5,shrinkage=0.03,verbose=TRUE)
# fixer verbose à FALSE pour éviter trop de sorties
plot(boost.reg$cv.error)
# nombre optimal d'itérations
best.iter=gbm.perf(boost.reg,method="cv")
best.iter
# influence des variables
summary(boost.dist,n.trees=best.iter)
```

On peut s'assurer de l'absence d'un phénomène de sur-apprentissage critique en calculant puis traçant l'évolution de l'erreur sur l'échantillon test en fonction du nombre d'arbre dans le modèle :

```
test=numeric()
for (i in 10:500){
pred.test=predict(boost.reg,newdata=datestr,n.trees=i)
err=sum((pred.test-datestr[,"O3obs"])^2)/nrow(datestr)
test=c(test,err)}
plot(10:500,test,type="l")
abline(v=best.iter)
```

La ligne verticale précise le nombre d'itérations sélectionné. Tester ces fonctions en faisant varier le coefficient de rétrécissement.

Discrimination

Attention, la variable à modéliser doit être codée $(0, 1)$ et il fait préciser un autre paramètre de distribution pour considérer le bon terme d'erreur.

```
boost.dis=gbm(as.numeric(DepSeuil)-1~.,data=datappq,
   distribution="adaboost",n.trees=500, cv.folds=10,
   n.minobsinnode = 5,shrinkage=0.03,verbose=FALSE)
plot(boost.dis$cv.error)
# nombre optimal d'itérations
best.iter=gbm.perf(boost.dis,method="cv")
best.iter
# influence des variables
summary(boost.dist,n.trees=best.iter)
```

Comme pour la régression, il est possible de faire varier le coefficient de rétrécissement en l'associant au nombre d'arbres dans le modèle.

```
# calcul et graphe des résidus
fit.boostr=boost.reg$fit
res.boostr=fit.boostr-datappr[,"O3obs"]
plot.res(fit.boostr,res.boostr)
```

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

```
fit.boostq=boost.dis$fit>0.5
#matrice de confusion
table(fit.boostq,datappq$DepSeuil)
```

Echantillon test

La prévision de l'échantillon test et de la matrice de confusion associée sont obtenus par les commandes :

```
boost.reg=gbm(O3obs~., data=datappr,
   distribution="gaussian",n.trees=500, cv.folds=10,
   n.minobsinnode = 5,shrinkage=0.03,verbose=FALSE)
best.iter=gbm.perf(boost.reg,method="cv")
pred.boostr=predict(boost.reg,newdata=datestr,
   n.trees=best.iter)
# Erreur quadratique moyenne de prévision
sum((pred.boostr-datestr[,"O3obs"])^2)/nrow(datestr)
# Matrice de confusion pour la prévision
# du dépassement de seuil (régression)
table(pred.boostr>150,datestr[,"O3obs"]>150)
# Même chose pour la discrimination
pred.boostd=predict(boost.dis,newdata=datestq,
   n.trees=best.ited)
table(as.factor(sign(pred.boostd)),
   datestq[,"DepSeuil"])
```

Quelle stratégie d'agrégation de modèles vous semble fournir le meilleur résultat de prévision ? Est-elle, sur ce jeu de données, plus efficace que les modèles classiques expérimentés auparavant ?

7.5 Comparaison des courbes ROC

Il y a en tout 6 courbes à comparer ; par souci de lisibilité, elles sont séparées en deux groupes et toujours comparées avec le modèle de covariance quadratique initial.

Modèles de régression

```
library(ROCR)
rocbagr=pred.bagr/300
predbagr=prediction(rocbagr,datestq$DepSeuil)
perfbagr=performance(predbagr,"tpr","fpr")

rocrfr=pred.rfr/300
predrfr=prediction(rocrfr,datestq$DepSeuil)
perfrfr=performance(predrfr,"tpr","fpr")
rocbstr=pred.boostr/300
predbstr=prediction(rocbstr,datestq$DepSeuil)
perfbstr=performance(predbstr,"tpr","fpr")
# tracer les courbes ROC en les superposant
# pour mieux comparer
plot(perflogit,col=1)
plot(perfbagr,col=2,add=TRUE)
plot(perfrfr,col=3,add=TRUE)
plot(perfbstr,col=4,add=TRUE)
```

Modèles de discrimination

```
ROCbag=predict(bag.dis,newdata=datestq,type="prob")[,2]
predbag=prediction(ROCbag,datestq$DepSeuil)
perfbag=performance(predbag,"tpr","fpr")

ROCrf=rf.dis$test$vote[,2]
predrf=prediction(ROCrf,datestq$DepSeuil)
perfrf=performance(predrf,"tpr","fpr")

ROCboost=predict(boost.reg,newdata=datestr,
   n.trees=best.ited)
predboost=prediction(ROCboost,datestq$DepSeuil)
perfboost=performance(predboost,"tpr","fpr")

plot(perflogit,col=1)
plot(perfbag,col=2,add=TRUE)
plot(perfrf,col=3,add=TRUE)
plot(perfboost,col=4,add=TRUE)
legend("bottomright",legend=c("acova","bag",
   "randF","boost"),col=c(1:4),pch="_")
```

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

8 Machines à vecteurs supports

8.1 Introduction

Cette section propose d'aborder une nouvelle famille d'algorithmes : les SVM ou (*Support Vector Machines* traduit par Séparateurs à Vaste Marge ou machine à vecteurs support) dont le principe fondateur est d'intégrer l'optimisation de la complexité d'un modèle à son estimation ou plus exactement une partie de cette complexité ; cela concerne le nombre de vecteurs supports. Une bibliothèque de R, réalisée par Chang, Chih-Chung et Lin Chih-Jen, est destinée à cette approche ; elle est intégrée au package `e1071`.

Au delà du principe fondateur de recherche de parcimonie (nombre de supports) incorporé à l'estimation, il n'en reste pas moins que cette approche laisse, en pratique, un certain nombre de choix et réglages à l'utilisateur. Il est donc important d'en tester l'influence sur la qualité des résultats.

i. choix du paramètre de régularisation ou pondération d'ajustement,

ii. choix du noyau,

iii. le cas échéant, choix du paramètre associé au noyau : largeur d'un noyau gaussien, degré d'un noyau polynomial...

Notons la même remarque qu'avec les techniques précédentes sur l'intérêt à mettre en œuvre une approche "plan d'expérience" voire même "surface de réponse" afin d'optimiser le choix des paramètres.

8.2 Régression sur la concentration d'ozone

Malgré les assurances théoriques concernant ce type d'algorithme, les résultats dépendant fortement du choix des paramètres. Nous nous limiterons d'abord au noyau gaussien (choix par défaut) ; la fonction `tune.svm` permet de tester facilement plusieurs situations en estimant la qualité de prévision par validation croisée sur une grille. Le temps d'exécution est un peu long... en effet, contrairement à beaucoup d'algorithmes de modélisation, la complexité de l'algorithme de résolution des SVM croît très sensiblement avec le nombre d'observations mais moins avec le nombre de variables. Cette remarque est importante quant à l'adéquation à trouver entre données et méthode.

Bien qu'initialement développés dans le cas d'une variable binaire, les SVM ont été étendus aux problèmes de régression. L'estimation et l'optimisation du coefficient de pénalisation sont obtenues par les commandes suivantes.

```
svm.reg=svm(O3obs~.,data=datappr)
plot(tune.svm(O3obs~.,data=datappr,
   cost=c(1, 1.5,2,2.5,3,3.5)))
```

Par défaut la pénalisation (cost) vaut 1. Noter la pénalisation optimale pour le noyau considéré (gaussien). Ré-estimer le modèle supposé optimal avant de tracer le graphe des résidus. Comme précédemment, observer que plusieurs exécutions conduisent à des résultats différents et donc que l'optimisaiton de ce critère est pour le moins délicate.

```
svm.reg=svm(O3obs~.,data=datappr,cost=2.5)
# calcul et graphe des résidus
fit.svmr=predict(svm.reg,data=datappr)
res.svmr=fit.svmr-datappr[,"O3obs"]
plot.res(fit.svmr,res.svmr)
```

Observer l'effet "couloir" sur les résidus. Ceci est une conséquence de la fonction d'erreur robuste utilisée pour l'estimation des svm.

8.3 Discrimination

```
#optimisation
plot(tune.svm(DepSeuil~.,data=datappq,
  cost=c(1,1.25,1.5,1.75,2)))
# apprentissage
svm.dis=svm(DepSeuil~.,data=datappq,cost=1.25)
```

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

```
#matrice de confusion
table(svm.dis$fitted,datappq$DepSeuil)
```

Remarque : il faudrait aussi optimiser la valeur du paramètre gamma du noyau gaussien ainsi que le choix du noyau...

8.4 Prévision de l'échantillon test

Avec les "meilleures" combinaisons de paramètres précédentes, estimer les erreurs sur l'échantillon test.

```
svm.reg=svm(O3obs~.,data=datappr,cost=2.5)
svm.dis=svm(DepSeuil~.,data=datappq,cost=1.25)
pred.svmr=predict(svm.reg,newdata=datestr)
pred.svmq=predict(svm.dis,newdata=datestq)
# Erreur quadratique moyenne de prévision
sum((pred.svmr-datestr[,"O3obs"])^2)/nrow(datestr)
# Matrice de confusion pour la prévision
# du dépassement de seuil (régression)
table(pred.svmr>150,datestr[,"O3obs"]>150)
# Même chose pour la discrimination
table(pred.svmq,datestq[,"DepSeuil"])
```

Noter, comparer les taux d'erreur.

8.5 Comparaison des courbes ROC

```
library(ROCR)
rocsvmr=pred.svmr/300
predsvmr=prediction(rocsvmr,datestq$DepSeuil)
perfsvmr=performance(predsvmr,"tpr","fpr")
# re-estimer le modèle pour obtenir des
# probabilités  de classe plutôt que des classes
svm.dis=svm(DepSeuil~.,data=datappq,cost=1.25,
  probability=TRUE)
pred.svmq=predict(svm.dis,newdata=datestq,
  probability=TRUE)
rocsvmq=attributes(pred.svmq)$probabilities[,2]
predsvmq=prediction(rocsvmq,datestq$DepSeuil)
perfsvmq=performance(predsvmq,"tpr","fpr")
# tracer les courbes ROC en les superposant
# pour mieux comparer
plot(perflogit,col=1)
plot(perfsvmr,col=2,add=TRUE)
plot(perfsvmq,col=3,add=TRUE)
```

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

9 Objectif

L'objectif final est d'arriver à une comparaison fine et synthétique des différentes méthodes pour aboutir à des prises de décision :

- Quelle méthode utiliser pour prévoir au mieux la concentration d'ozone ?
- Quelle stratégie et quelle méthode utiliser pour prévoir le dépassement du seuil :
  - Prévision quantitative puis compariaosn au seuil ?
  - Prévision qualitative ?

La taille de l'échantillon test, autour de 200 jours, est relativement modeste pour espérer une bonne précision sur la foi d'un seul échantillon. Pour préciser la comparaison c'est-à-dire pour prendre en compte la variance de l'estimation de l'erreur, le processus est itérée.

10 Itérations de l'estimation des erreurs

Consulter le programme `comparaison_ozone.R`, définir les paramètres : initialisation du générateur (xxx) et nombre d'itérations (xx au moins 30) avant de le faire exécuter en "batch" ou de nuit.

Ce programme lit les données "ozone.dat" et itère N fois le tirage d'un échantillon test pour estimer N erreurs de prévision pour chacune des méthodes de modélisation considérée. Sont calculées sur chaque échantillon test :

- l'erreur quadratique de prévision en régression,
- le taux d'erreur de classification issue de la prévision de dépassement de seuil,
- le taux d'erreur de classification comme prévision d'une variable binaire de dépassement.

Ces résultats sont respectivement stockés dans les matrices `res.reg`, `res.clas.r` et `res.clas.q`.

Pour construire les cournes ROC, le programme stocke également les prévisions pour chacune des méthodes de modélisation considérée et pour chaque

échantillon test dans des variables de type liste et de nom : `list.methode` où méthode est la méthode de modélisation considérée. On se limite aux méthodes apparues les "meilleures" lors de la comparaison des distributions des erreurs. Attention, cette sélection n'est pas forcément "optimale" ; elle pourrait être complétée utilement.

11 Synthèses des résultats

11.1 Prévison de concentration

Calculer, comparer les moyennes et écarts-types des distributions des erreurs quadratiques de prévision. Tracer les diagrammes boîtes parallèles de ces distributions :

```
boxplot(res.reg)
```

Commentaires.

11.2 Prévision de dépassement

Calculer, comparer les moyennes et écarts-types des distributions des taux d'erreur de prévision du dépassement de seuil. Tracer les diagrammes boîtes parallèles de ces distributions :

```
boxplot(data.frame(res.clas.r,res.clas.q))
```

Commentaires.

Tracer les courbes ROC par échantillon test en superposant des diagrammes boîtes visualisant les dispersion des courbes :

```
library(ROCR)
#création des objets ROC
pred <- prediction(list.svmr$predictions,
   list.svmr$labels)
perf.svmr <- performance(pred,"tpr","fpr")
plot(perf.svmr,col="grey82",lty=3)
plot(perf.svmr,lwd=3,avg="vertical",
   spread.estimate="boxplot",add=TRUE)

pred <- prediction(list.svmq$predictions,
   list.svmq$labels)
perf.svmq <- performance(pred,"tpr","fpr")
plot(perf.svmq,col="grey82",lty=3)
plot(perf.svmq,lwd=3,avg="vertical",
   spread.estimate="boxplot",add=TRUE)

pred <- prediction(list.rfr$predictions,
   list.rfr$labels)
perf.rfr <- performance(pred,"tpr","fpr")
plot(perf.rfr,col="grey82",lty=3)
plot(perf.rfr,lwd=3,avg="vertical",
   spread.estimate="boxplot",add=TRUE)

pred <- prediction(list.rfq$predictions,
   list.rfr$labels)
perf.rfq <- performance(pred,"tpr","fpr")
plot(perf.rfq,col="grey82",lty=3)
plot(perf.rfq,lwd=3,avg="vertical",
   spread.estimate="boxplot",add=TRUE)

pred <- prediction(list.log$predictions,
   list.log$labels)
perf.log <- performance(pred,"tpr","fpr")
plot(perf.log,col="grey82",lty=3)
plot(perf.log,lwd=3,avg="vertical",
   spread.estimate="boxplot",add=TRUE)

pred <- prediction(list.mlq$predictions,
   list.log$labels)
perf.mlq <- performance(pred,"tpr","fpr")
plot(perf.mlq,col="grey82",lty=3)
plot(perf.mlq,lwd=3,avg="vertical",
   spread.estimate="boxplot",add=TRUE)
```

Superposer les moyennes par méthode de ces courbes ROC.

```
plot(perf.mlq,col=1,avg="vertical")
plot(perf.log,col=2,avg="vertical",add=TRUE)
plot(perf.rfq,col=3,avg="vertical",add=TRUE)
plot(perf.rfr,col=4,avg="vertical",add=TRUE)
plot(perf.svmr,col=5,avg="vertical",add=TRUE)
plot(perf.svmq,col=6,avg="vertical",add=TRUE)
legend("bottomright",legend=c("acova","logit",
   "rfq","rfr","svmr","svmq"),col=1:6,pch="_")
```

Adapter ces graphiques afin de mettre clairement en évidence la "meilleure" méthode. Préciser ce choix en fonction d'un taux de faux positifs jugé acceptable : 10%, 20%, 40% en comparaison du taux de vrais positifs espéré.