Scénario : sélection de modèles avec R (prostate)

Résumé

Comparaison sur le même jeu de données (prostate) des qualités de prévision de plusieurs modèles obtenus par :

- *Modèle linéaire*
- *Algorithmes de sélection de modèles (critères AIC, BIC)*
- *Régularisation (ridge, Lasso, Elastic Net)*
- *Régression sur composantes (PCR, PLS)*

1 Les données

Nous allons utiliser les données Prostate du package **lasso 2** de R.

" These data [1] come from a study that examined the correlation between the level of prostate specific antigen and a number of clinical measures in men who were about to receive a radical prostatectomy. It is data frame with 97 rows and 9 columns.

Prise en charge des données :

```
library(lasso2)
data(Prostate)
summary(Prostate)
#Les variables {\bf svi} et {\bf gleason}
# sont qualitatives
Prostate$svi=as.factor(Prostate$svi)
Prostate$gleason=as.factor(Prostate$gleason)
```

Vérification des données et description élémentaire :

1. Stamey, T.A., Kabalin, J.N., McNeal, J.E., Johnstone, I.M., Freiha, F., Redwine, E.A. and Yang, N. (1989). Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate : II. radical prostatectomy treated patients, *Journal of Urology* 141(5), 1076–1083.

TABLE 1 – Liste des 9 variables, la dernière est à expliquer.

lcavol	log(cancer volume)
lweight	log(prostate weight)
age	age
lbph	log(benign prostatic hyperplasia amount)
svi	seminal vesicle invasion
lcp	log(capsular penetration)
gleason	Gleason score
pgg45	percentage Gleason scores 4 or 5
lpsa	log(prostate specific antigen)

```
dim(Prostate)
names(Prostate)
summary(Prostate)
cor(Prostate[,-c(5,7)])
hist(Prostate$lpsa)
```

La suite du travail n'écessite un échantillon d'apprentissage pour estimer les modèles et un échantillon test pour comparer les erreurs de prévision.

```
#Les valeurs de lpsa sont rangées par ordre croissant
#On conserve 1/4 des données pour l'échantillon test
ind.test=4*c(1:22)
Prostate.app=Prostate[-ind.test,]
Prostate.test=Prostate[c(ind.test),]
dim(Prostate.test)
dim(Prostate.app)
ntest=length(Prostate.test$lpsa)
napp=length(Prostate.app$lpsa)
summary(Prostate.app)
summary(Prostate.test)
```

2 Modèle linéaire complet

Une fonction utile de graphe des résidus.

```
plot.res=function(x,y,titre="")
{
plot(x,y,col="blue",ylab="Résidus",
   xlab="Valeurs predites",main=titre)
abline(h=0,col="green")
}
```

2.1 Estimation du modèle et graphes des résidus

```
modlin=lm(lpsa~., data=Prostate.app)
summary(modlin) # noter les p-valeurs
#Residus
res=residuals(modlin)
#Regroupement des graphiques sur la meme page
par(mfrow=c(1,2))
hist(residuals(modlin))
qqnorm(res)
qqline(res, col = 2)
# retour au graphique standard
par(mfrow=c(1,1))
plot.res(predict(modlin),res)
```

2.2 Erreur d'apprentissage

```
mean(res**2)
```

2.3 Erreur sur l'échantillon test

```
pred.test=predict(modlin, newdata=Prostate.test)
res.test=pred.test-Prostate.test$lpsa
mean(res.test**2)
```

2.4 Nouvelle paramétrisation

Afin de faciliter l'interprétation des résultats concernant les variables qualitatives, on introduit une nouvelle paramétrisation à l'aide de contrastes. Par défaut, la référence est prise pour la valeur 0 de `svi` et 6 de `gleason`, qui sont les plus petites valeurs. Les paramètres indiqués pour les variables `svi1`, `gleason` 7, 8 et 9 indiquent l'écart estimé par rapport à cette référence. Il est plus intéressant en pratique de se référer à la moyenne des observations sur toutes les modalités des variables qualitatives, et d'interpréter les coefficients comme des écarts à cette moyenne.

```
contrasts(Prostate.app$svi)=
   contr.sum(levels(Prostate.app$svi))
contrasts(Prostate.app$gleason)=
   contr.sum(levels(Prostate.app$gleason))
modlin2=lm(lpsa~., Prostate.app)
summary(modlin2)
# Attention au nom des variables : gleason1 =6,
#gleason2 =7, gleason3 =8,  gleason4 =9 (pas affiché),
#la somme des coefficients associés à ces variables
#est nulle
#svi1=0, svi2=1 (pas affiché)
#la somme des deux coefficients est nulle
```

3 Sélection de modèle par sélection de variables

3.1 Sélection par AIC et backward

```
library(MASS)
modselect_b=stepAIC(modlin2,~.,trace=TRUE,
   direction=c("backward"))
summary(modselect_b)
# noter que des paramètres restent non significatifs
```

3.2 Sélection par AIC et forward

```
mod0=lm(lpsa~1,data=Prostate.app)
modselect_f=stepAIC(mod0,lpsa~lcavol+lweight
   +age+lbph+svi+lcp+gleason+pgg45,data=
   Prostate.app,trace=TRUE,direction=c("forward"))
summary(modselect_f)
```

3.3 Sélection par AIC et stepwise

```
modselect=stepAIC(modlin2,~.,trace=TRUE,
direction=c("both"))
#both est l'option par défaut
summary(modselect)
#On retrouve ici le même modèle qu'avec
#l'agorithme backward
```

3.4 Sélection par BIC et stepwise

```
#  k=log(napp) pour BIC au lieu de AIC.
modselect_BIC=stepAIC(modlin2,~.,trace=TRUE,
   direction=c("both"),k=log(napp))
summary(modselect_BIC)
#Le modèle sélectionné est plus parcimonieux
```

3.5 Erreur sur l'échantillon d'apprentissage

```
#Modèle stepwise AIC
mean((predict(modselect)-Prostate.app[,"lpsa"])**2)
#valeur trouvée 0.49
#Modèle stepwise BIC
mean((predict(modselect_BIC)-
   Prostate.app[,"lpsa"])**2)
# valeur trouvée 0.53
```

3.6 Calcul de l'erreur sur l'échantillon test

```
#Modèle stepwise AIC
mean((predict(modselect,newdata=Prostate.test)
   -Prostate.test[,"lpsa"])**2)
#valeur trouvée 0.46
#Modèle stepwise BIC
mean((predict(modselect_BIC,newdata=Prostate.test)
   -Prostate.test[,"lpsa"])**2)
# valeur trouvée 0.40
```

Les modèles sélectionnés ont une erreur plus grande que le modèle linéaire comprenant toutes les variables sur l'échantillon d'apprentissage (c'est normal !). Sur l'échantillon test, le modèle qui minimise le critère BIC a de meilleures performances que le modèle initial. Les deux modèles sélectionnés sont beaucoup plus parcimonieux que le modèle initial.

4 Sélection de modèle par pénalisation Ridge

4.1 Comportement des coefficients

Calcul des coefficients pour différentes valeurs du paramètre lambda.

```
library(MASS)
mod.ridge=lm.ridge(lpsa~.,data=Prostate.app,
   lambda=seq(0,20,0.1))
par(mfrow=c(1,1))
plot(mod.ridge)
# évolution des coefficients
matplot(t(mod.ridge$coef),lty=1:3,type="l",col=1:10)
legend("top",legend=rownames(mod.ridge$coef),
   col=1:10,lty=1:3)
```

4.2 Pénalisation optimale par validation croisée

```
select(mod.ridge) # noter la valeur puis estimer
mod.ridgeopt=lm.ridge(lpsa ~  .,data=Prostate.app,
   lambda=10.4)
```

4.3 Prévision et erreur d'apprentissage

Pour des raisons obscures, la fonction `predict.ridgelm` n'existe pas, il faut calculer les valeurs prédites à partir des coefficients.

```
#Coefficients du modèle sélectionné :
coeff=coef(mod.ridgeopt)
#On crée des vecteurs pour les variables
#qualitatives
```

```
svi0.app=1*c(Prostate.app$svi==0)
svi1.app=1-svi0.app
gl6.app=1*c(Prostate.app$gleason==6)
gl7.app=1*c(Prostate.app$gleason==7)
gl8.app=1*c(Prostate.app$gleason==8)
gl9.app=1*c(Prostate.app$gleason==9)
#variables quantitatives
lcavol.app=Prostate.app$lcavol
lweight.app=Prostate.app$lweight
age.app=Prostate.app$age
lbph.app=Prostate.app$lbph
lcp.app=Prostate.app$lcp
pgg45.app=Prostate.app$pgg45

#Calcul des valeurs prédites
fit.rid=rep(coeff[1],napp)+coeff[2]*lcavol.app+
coeff[3]*lweight.app+coeff[4]*age.app+
coeff[5]*lbph.app+coeff[6]*svi0.app-
coeff[6]*svi1.app+coeff[7]*lcp.app+
coeff[8]*gl6.app+coeff[9]*gl7.app+
coeff[10]*gl8.app-(coeff[8]+coeff[9]+
coeff[10])*gl9.app+coeff[11]*pgg45.app

#Tracé des valeurs prédites en fonctions
#des valeurs observées
plot(Prostate.app$lpsa,fit.rid)
abline(0,1)
#Calcul et tracé des résidus
res.rid=fit.rid-Prostate.app[,"lpsa"]
plot.res(fit.rid,res.rid,titre="")
#Erreur d'apprentissage
mean(res.rid**2)
#0.478
```

4.4 Prévision sur l'échantillon test

```
#Variables qualitatives
svi0.t=1*c(Prostate.test$svi==0)
svi1.t=1-svi0.t
gl6.t=1*c(Prostate.test$gleason==6)
gl7.t=1*c(Prostate.test$gleason==7)
gl8.t=1*c(Prostate.test$gleason==8)
gl9.t=1*c(Prostate.test$gleason==9)
#Variables quantitatives
lcavol.t=Prostate.test$lcavol
lweight.t=Prostate.test$lweight
age.t=Prostate.test$age
lbph.t=Prostate.test$lbph
lcp.t=Prostate.test$lcp
pgg45.t=Prostate.test$pgg45

prediction=rep(coeff[1],ntest)+coeff[2]*
   lcavol.t+coeff[3]*lweight.t+coeff[4]*age.t+
   coeff[5]*lbph.t+coeff[6]*svi0.t-coeff[6]*svi1.t+
   coeff[7]*lcp.t+coeff[8]*gl6.t+coeff[9]*gl7.t+
   coeff[10]*gl8.t-(coeff[8]+coeff[9]+coeff[10])*
   gl9.t+coeff[11]*pgg45.t
#Erreur sur l'échantillon test
mean((Prostate.test[,"lpsa"]-prediction)^2)
# 0.424
```

L'erreur d'apprentissage est légèrement plus élévée que pour le modèle linéaire sans pénalisation (c'est normal !) l'erreur de test est plus faible. Les performances sont comparables sur l'échantillon test au modèle sélectionné par le critère BIC. En terme d'interprétation, les modèles sélectionnés par AIC et BIC sont préférables.

5 Sélection de modèle par pénalisation Lasso

Les résultats sont obtenus avec la librairie `lasso2` ou avec la librairie `glmnet`

5.1 Librairie Lasso2

5.2 Construction du modèle

```
library(lasso2)
l1c.P <- l1ce(lpsa ~ ., Prostate.app,
   bound=(1:100)/100,absolute.t=FALSE)
```

La borne est ici relative, elle correspond à une certaine proportion de la norme $\mathbb{L}_1$ du vecteur des coefficients des moindres carrés. Une borne égale à 1 correspond donc à l'absence de pénalité, on retrouve l'estimateur des moindres carrés.

5.3 Visualisation des coefficients

```
coefficients=coef(l1c.P)
plot(l1c.P,col=1:11,lty=1:3,type="l")
legend("topleft",legend=colnames(coefficients),
   col=1:11,lty=1:3)

#On supprime le terme constant
penalite_relative=c(1:100)/100
matplot(penalite_relative,coefficients[,-1],
   lty=1:3,type="l",col=1:10)
legend("topleft",legend=
   colnames(coefficients[,-1]),col=1:10,lty=1:3)
```

5.4 Sélection de la pénalité par validation croisée

```
vc=gcv(l1c.P)
crit.vc=vc[,"gcv"]
bound_opt=vc[which.min(crit.vc),"rel.bound"]
#0.95
l1c.opt <- l1ce(lpsa ~ ., Prostate.app,
bound=bound_opt, absolute.t=FALSE)
coef=coef(l1c.opt)
```

5.5 Erreur d'apprentissage

```
#erreur apprentissage
fit=fitted(l1c.opt)
mean((fit-Prostate.app[,"lpsa"])^2)
#0.46
```

5.6 Erreur sur l'échantillon test

```
prediction=predict(l1c.opt,newdata=Prostate.test)
mean((prediction-Prostate.test[,"lpsa"])^2)
#0.44
```

5.7 Librairie glmnet

L'utilisation de la librairie `glmnet` fournit des résultats plus rapides, ce qui peut s'avérer important pour des données de grande dimension. Par contre, on ne peut pas traiter à priori des variables qualitatives. Nous allons donc devoir créer des vecteurs avec des variables indicatrices des diverses modalités pour les variables qualitatives. Nous ne prendrons pas en compte les contrastes.

5.8 Mise en forme des variables

```
#on construit une matrice xx.app d'apprentissage et
#xx.test de test
data(Prostate)
Prostate.app=Prostate[-ind.test,]
Prostate.test=Prostate[c(ind.test),]

x.app=Prostate.app[,-9]
y.app=Prostate.app[,9]
x.app=as.matrix(x.app)
# construction des indicatrices
svi1.app=1*c(Prostate.app$svi==1)
gl7.app=1*c(Prostate.app$gleason==7)
gl8.app=1*c(Prostate.app$gleason==8)
gl9.app=1*c(Prostate.app$gleason==9)

#matrice avec les vecteurs indicatrices
xx.app=matrix(0,ncol=10,nrow=nrow(x.app))
xx.app[,1:6]=x.app[,1:6]
```

```
xx.app[,7:9]=cbind(gl7.app,gl8.app,gl9.app)
xx.app[,10]=x.app[,8]
#on nomme les colonnes avec le noms des variables
colnames(xx.app)=c("lcavol","lweight", "age",
   "lbph","svi1","lcp","gl7","gl8","gl9","pgg45")

#on fait de même pour l'echantillon test
x.test=Prostate.test[,-9]
y.test=Prostate.test[,9]
x.test=as.matrix(x.test)
svi1.test=1*c(Prostate.test$svi==1)
gl7.test=1*c(Prostate.test$gleason==7)
gl8.test=1*c(Prostate.test$gleason==8)
gl9.test=1*c(Prostate.test$gleason==9)

#on construit une matrice avec les vecteurs
#indicatrices
xx.test=matrix(0,ncol=10,nrow=nrow(x.test))
xx.test[,1:6]=x.test[,1:6]
xx.test[,7:9]=cbind(gl7.test,gl8.test,gl9.test)
xx.test[,10]=x.test[,8]
colnames(xx.test)=colnames(xx.app)
```

5.9 Construction du modèle

```
library(glmnet)
out.lasso = glmnet(xx.app,y.app)
l=length(out.lasso$lambda)
b=coef(out.lasso)[-1,1:l]
```

5.10 Visualisation des coefficients

```
# chemin de régularisation du lasso
matplot(t(as.matrix(out.lasso$beta)),type="l",
   col=1:10,lty=1:3)
legend("topleft",legend=colnames(xx.app),
   col=1:10,lty=1:3)
title("Lasso")
```

5.11 Sélection de la pénalité par validation croisée

```
y.app=as.matrix(y.app)
a=cv.glmnet(xx.app,y.app)
#le resultat est dans
lambda.opt=a$lambda.min
app=glmnet(xx.app,y.app,lambda=lambda.opt)
```

5.12 Erreur d'apprentissage

```
appr=predict(app,newx=xx.app)
mean((appr-Prostate.app[,9])^2)
#0.48
```

5.13 Erreur sur l'échantillon test

```
pred=predict(app,newx=xx.test)
mean((pred-Prostate.test[,9])^2)
#0.39
```

On n'obtient pas exactement le même modèle qu'avec Lasso2. La procédure de validation croisée conduit ici à un modèle plus parcimonieux.

5.14 Elastic Net

```
#on peut jouer avec le paramètre alpha, de glmnet
out.elnet <- glmnet(xx.app,y.app,alpha=0.5)
a.elnet=cv.glmnet(xx.app,y.app,alpha=0.5)
lambda.opt=a.elnet$lambda.min
app=glmnet(xx.app,y.app,lambda=lambda.opt)
#erreur apprentissage
app.elnet=predict(a.elnet,newx=xx.app)
mean((app.elnet-Prostate.app[,9])^2)
#0.60
#erreur de prédiction
predi.elnet=predict(a.elnet,newx=xx.test)
```

```
mean((predi.elnet-Prostate.test[,9])^2)
#0.37
```

6 Sélection de modèle projection sur composantes orthogonales

6.1 Régression PLS

```
data(Prostate)
Prostate$svi=as.factor(Prostate$svi)
Prostate$gleason=as.factor(Prostate$gleason)
Prostate.app=Prostate[-ind.test,]
Prostate.test=Prostate[c(ind.test),]
library(pls)
#nombre optimal de composantes par validation
#croisée
simpls= mvr(lpsa~.,data=Prostate.app, ncomp=10,
   validation="CV", method="simpls")
summary(simpls)
#graphique
plot(simpls)
abline(0,1)
#noter le nombre optimal de composantes
plot(simpls,"val")
#Avec leave-one-out
simplsloo= mvr(lpsa~.,data=Prostate.app, ncomp=10,
   validation="LOO", method="simpls")
summary(simplsloo)
#On sélectionne 6 composantes
#Calcul des prévisions
predapp.pcr=predict(simpls,ncomp=6)
resapp.pcr=predapp.pcr-Prostate.app$lpsa
#Erreur d'apprentissage
mean(resapp.pcr**2)
#0.47
#Valeurs prédites sur l'échantillon test
pred.pls=predict(simpls,newdata=Prostate.test,
   ncomp=6)
#Erreur de test
mean((pred.pls-Prostate.test[,"lpsa"])**2)
#0.46
```

6.2 Régression sur composantes principales

```
mod.pcr = pcr(lpsa~.,data=Prostate.app, ncomp=9,
validation="CV")
summary(mod.pcr)
# noter le nombre optimal
plot(mod.pcr,"val")
#On sélectionne 8 composantes
#Calcul des prévisions
predapp.pls=predict(mod.pcr,ncomp=8)
resapp.pls=predapp.pls-Prostate.app$lpsa
#Erreur d'apprentissage
mean(resapp.pls**2)
#0.47
#Valeurs prédites sur l'échantillon test
pred.pcr=predict(mod.pcr,newdata=Prostate.test,
   ncomp=8)
#Erreur de test
mean((pred.pcr-Prostate.test[,"lpsa"])**2)
#0.44
```

Il peut arriver que la régression sur composantes principales ne soit pas adaptée, si les premières composantes principales trouvées n'ont que peu de rapport avec la variable Y. Le nombre de composantes PCR sélectionnées est ici égal à 8 (on ne réduit pas beaucoup la complexité par rapport au modèle initial). Avec la régression PLS, on sélectionne 6 composantes. Dans les 2 cas, on n'a pas beaucoup réduit la complexité des modèles par rapport au modèle linéaire.