TP: sélection de variables et sélection de modèle en régression

Résumé

Comparaison sur le même jeu de données des qualités de prévision de plusieurs modèles obtenus par :
- *sélection, de variables*
- *régression sur composantes (PLS, ACP)*
- *pénalisation (ridge, Lasso)*

1 Les données

Les données (Jobson, 1991) décrivent les résultats comptables de 80 entreprises du Royaume Uni. `RETCAP` est la variable à prédire. Les entreprises sont réparties aléatoirement en deux groupes de 40 entreprises.

1.1 Lecture des données

```
ukcomp.app=read.table("ukcomp1_r.dat",header=T)
ukcomp.test=read.table("ukcomp2_r.dat",header=T)
summary(ukcomp.test)
summary(ukcomp.app)
# Attention, à l'ordre des variables
ukcomp.test=data.frame(ukcomp.test
   [,names(ukcomp.app)])
```

1.2 Description des données

```
cor(ukcomp.app[,-1]) # noter les corrélations
library(FactoMineR)
PCA(ukcomp.app)
```

TABLE 1 – Liste des 13 variables, la première est à expliquer

RETCAP	Return on capital employed
WCFTDT	Ratio of working capital flow to total debt
LOGSALE	Log to base 10 of total sales
LOGASST	Log to base 10 of total assets
CURRAT	Current ratio
QUIKRAT	Quick ratio
NFATAST	Ratio of net fixed assets to total assets
FATTOT	Gross sixed assets to total assets
PAYOUT	Payout ratio
WCFTCL	Ratio of working capital flow to total current liabilities
GEARRAT	Gearing ratio (debt-equity ratio)
CAPINT	Capital intensity (ratio of total sales to total assets)
INVTAST	Ratio of total inventories to total assets

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")
}
```

Estimation du modèle et graphes des résidus.

```
fit.lm=lm(RETCAP~.,ukcomp.app)
summary(fit.lm)
#Regroupement des graphiques sur la meme page
par(mfrow=c(2,2))
#Residus et points influents
plot(fit.lm,las=1)
summary(fit.lm) # noter les p-valeurs
par(mfrow=c(1,1)) # retour au graphique standard
```

2 Sélection de modèle par sélection de variables

Sélection par AIC et backward

```
uk.lmback=step(fit.lm) # noter q
# des paramètres restent non significatifs
anova(uk.lmback)
```

Sélection par BIC et backward

```
#  k=log(n) pour BIC au lieu de AIC.
uk.lmback=step(fit.lm,k=log(40))
anova(uk.lmback) # noter q et les variables
```

Sélection par AIC et forkward

```
fit.lm=lm(RETCAP  ~ 1 , data = ukcomp.app)
uk.lmfor=step(fit.lm,scope=list(lower=~1,
  upper=~WCFTCL+WCFTDT+GEARRAT+LOGSALE+
  LOGASST+NFATAST+CAPINT+FATTOT+INVTAST+
  PAYOUT+QUIKRAT+CURRAT),direction="forward")
anova(uk.lmfor) # noter q
```

Sélection par BIC et forkward

```
fit.lm=lm(RETCAP  ~ 1 , data = ukcomp.app)
uk.lmfor=step(fit.lm,scope=list(lower=~1,
  upper=~WCFTCL+WCFTDT+GEARRAT+LOGSALE+
  LOGASST+NFATAST+CAPINT+FATTOT+INVTAST+
  PAYOUT+QUIKRAT+CURRAT),direction="forward",
  k=log(40))
anova(uk.lmfor) # noter q et les variables
```

Sélection par AIC et stepwise

```
fit.lm=lm(RETCAP  ~ 1 , data = ukcomp.app)
uk.lmboth=step(fit.lm,scope=list(lower=~1,
  upper=~WCFTCL+WCFTDT+GEARRAT+LOGSALE+
  LOGASST+NFATAST+CAPINT+FATTOT+INVTAST+
  PAYOUT+QUIKRAT+CURRAT),direction="both")
anova(uk.lmboth) # noter q, les variables
```

Sélection par BIC et stepwise

```
fit.lm=lm(RETCAP  ~ 1 , data = ukcomp.app)
uk.lmboth=step(fit.lm,scope=list(lower=~1,
  upper=~WCFTCL+WCFTDT+GEARRAT+LOGSALE+
  LOGASST+NFATAST+CAPINT+FATTOT+INVTAST+
  PAYOUT+QUIKRAT+CURRAT),direction="both",
  k=log(40))
anova(uk.lmboth)  # noter q, les variables
```

Commentaires sur le mode de sélection des algorithmes et l'importance de la pénalisation appliquée par chaque critère. Tentative de les départager par l'algorithme de Furnival et Wilson qui explore potentiellement toutes les possibilités. L'algorithme est associé au C_p de Mallows.

Recherche exhaustive et Cp

```
library(leaps)
par(mfrow=c(1,1))
#Extraction des variables explicatives
ukcomp=ukcomp.app[,2:13]
#Recherche du meilleur modèle pour chaque q
uk.choix=leaps(ukcomp,ukcomp.app[,"RETCAP"],
    method="Cp",nbest=1)
uk.choix$Cp #valeurs des Cp du meilleur modèle
plot(uk.choix$size-1,uk.choix$Cp)
# Fixer la dimension  / complexité optimale
t=(uk.choix$Cp==min(uk.choix$Cp))
# Liste des variables explicatives
colnames(ukcomp)[uk.choix$whi[t]]
```

D'autres stratégies (R^2 ajusté) conduiraient encore à d'autres modèles. Retenir celui ci-dessous minimisant le Cp.

Recherche exhaustive et Cp

```
lm.uk0=lm(RETCAP ~ WCFTDT+LOGSALE+NFATAST+CURRAT,
   data=ukcomp.app)
mean((predict(lm.uk0,newdata=ukcomp.test)-ukcomp.test
```

```
   [,"RETCAP"])**2)
```

3 Sélection de modèle projection sur composantes orthogonales

3.1 Régression PLS

```
library(pls)
# nombre optimal de composantes par
# validation croisée
uk.simpls= mvr(RETCAP~.,ukcomp.app, ncomp=12,
   validation="CV", method="simpls")
summary(uk.simpls)
#graphique
plot(uk.simpls)
#noter le nombre optimal de composantes
#Calcul des prévisions
pred.uk=predict(uk.simpls,as.matrix
(ukcomp.test[,2:13]),4)
mean((pred.uk-ukcomp.test[,"RETCAP"])**2)
```

3.2 Régression sur composantes principales

```
uk.pcr = pcr(RETCAP~.,ukcomp.app, ncomp=12,
   validation="CV")
summary(uk.pcr) # noter le nombre optimal
#Calcul des prévisions
pred.uk=predict(uk.pcr,as.matrix
   (ukcomp.test[,2:13]),8)
mean((pred.uk-ukcomp.test[,"RETCAP"])**2)
```

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.

4 Sélection de modèle par pénalisation

4.1 Pénalisation ridge

Comportement des coefficients

Calcul des coefficients pour différentes valeurs du paramètre lambda.

```
library(MASS)
ridge.uk=lm.ridge(RETCAP ~  .,data=ukcomp.app,
    lambda=seq(0,0.4,0.001))
par(mfrow=c(1,1))
plot(ridge.uk)
```

Pénalisation optimale par GCV

```
select(ridge.uk) # noter la valeur puis estimer
ridgeopt.uk=lm.ridge(RETCAP ~  .,data=ukcomp.app,
  lambda=0.033)
```

On peut aussi utiliser une fonction explicite de validation croisée pour tracer l'erreur en fonction de lambda.

Prévision et erreur

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

```
coeff=coef(ridgeopt.uk)
fit.rid=rep(coeff[1],nrow(ukcomp.app))+
    as.vector(coeff[-1]%*%
    t(data.matrix(ukcomp.app[,-1])))
plot(fit.rid,ukcomp.app[,"RETCAP"])
res.rid=fit.rid-ukcomp.app[,"RETCAP"]
plot.res(fit.rid,res.rid,titre="")
```

Prévision de l'échantillon test

```
prediction=rep(coeff[1],nrow(ukcomp.test))+
   as.vector(coeff[-1]%*%t(data.matrix
   (ukcomp.test[,-1])))
mean((ukcomp.test[,1]-prediction)^2)
```

4.2 Pénalisation Lasso

Les résultats sont obtenus par la librairie `lasso2`.

Construction du modèle

```
lasso.uk=l1ce(RETCAP~.,ukcomp.app,bound=(1:30)/30,
trace=TRUE,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.

Visualisation des coefficients

```
plot(lasso.uk)
```

Sélection de la pénalité par validation croisée

```
gg.uk=gcv(lasso.uk)
gcv.uk=gg.uk[,4]
min(gcv.uk)
lasso.uk.select=l1ce(RETCAP ~ .,ukcomp.app,bound=27/30,
   absolute.t=FALSE)
coef=coef(lasso.uk.select)
```

Prévision et erreur

```
fit.lasso= coef[1]+   as.vector(coef[-1]%*%
    t(data.matrix(ukcomp.app[,-1])))

plot(fit.lasso,ukcomp.app[,"RETCAP"])
abline(0,1)

res.lasso=fit.lasso-ukcomp.app[,"RETCAP"]
plot.res(fit.lasso,res.lasso,titre="Residus Lasso")
```

Prévision de l'échantillon test

```
pred.lasso= coef[1]+   as.vector(coef[-1]%*%
    t(data.matrix(ukcomp.test[,-1])))

mean((pred.lasso-ukcomp.test[,"RETCAP"])^2)
```