TP STATISTIQUE EN GRANDE DIMENSION

AGBLODOE Komi/ M2 SSD

7 janvier 2020

TP 1

Soit X la matrice de données de taille $n.p$. Dans ce TP, il s'agit de montrer empiriquement qu'il y a plusieurs inconvénients à utiliser le maximum de vraissemblance lorsqu'on est en présence de données en grande dimension.

1

Ici on génère une matrice X obtenue à partir de la loi normale standard de taille $n = 20$ pour $p = 19$ co-variables.

```
set.seed(1)
n<-20
p<-19

x<-matrix(rnorm(n*p),n,p)
head(x)
```

2

Ensuite on trace le maximum des variances des estimateurs des coefficients, du modèle linéaire en fonction du nombre de co-variables p.

```
bigvar<-numeric(p)

for (i in 1:p){
 var<-solve(crossprod(x[,1:i]))#var(hat beta)=sigma^2 Var
 bigvar[i]<-max(diag(var))
}

plot(1:p,bigvar)

#ou soit on peut utiliser le log pour mieux voir l'augmentation des variances

plot(1:p,log(20*bigvar,2),ylab="La plus grande variance",xlab = "Nombre de colonnes")
#logAxis(2, Base=2 )

bigvar[15]/bigvar[1]
```

On a utilisé le log pour mieux voir l'augmentation des variances.

3

On remarque que quand p s'approche de n, la variance augmente largement de valeur. On est donc en présence d'une non convergence.

4

$MSE = Variance + (Biais)^2$ Pour un bon modèle, il faut que le MSE tende vers 0. On a vu que lorsque le nombre de co-variables devient grand, la variance augmente. Ce qui fait que le MSE ne tend pas vers 0. Ceci explique les limites de ces modèles en grande dimension.

Reprenons l'exo avec un p plus grand que n

```
set.seed(1)
n<-20
p<-25
x<-matrix(rnorm(n*p),n,p)

bigvar<-numeric(p)

#for (i in 1:p){
  #var<-solve(crossprod(x[,1:i]))
 # bigvar[i]<-max(diag(var))
#}

#plot(1:p,bigvar)
```

Pour p supérieur à n, cela ne marche pas car la matrice n'est plus inversible. Il y a une colinéarité entre certaines colonnes.

1.2 Problème de sélection de variables

1

Ici, on considère une matrice de données X de n = 25 observations sur p = 100 co-variables distribuées aléatoirement et une matrice XX de même dimension mais distribuée uniformément sur (0,1). Puis une variable d'intérêt y constituée de n nombres aléatoires indépendants et issus d'une loi normale centrée-réduite. On selectionne un modéle avec seulement 5 variables par BIC et on répète ceci 100 fois et on observe les estimations $\hat{\beta}$ de β.

```
set.seed(1)
n <- 25
p <- 100
xnam <- paste0("V", 1 :p)
form <- as.formula(paste("y ~ ", paste(xnam, collapse= "+")))

N <- 100
res <- cover <- pred <- NULL
pb <- txtProgressBar(1, N, style=3)
```

```
for (i in 1 :N) {
  X <- (matrix(runif(n*p), n, p))
  Data <- as.data.frame(X)
  y <- rnorm(n)
  j <- which.max(abs(crossprod(X, y - mean(y))))
  XX <- (matrix(runif(n*p), n, p))
  pData <- as.data.frame(XX)
  yy <- rnorm(n)
  fit0 <- lm(y~1, data=Data)
  fit <- step(fit0, scope=form, direction="forward", trace=0, k=log(n), steps=5)
  res <- rbind(res, summary(fit)$coef[-1,])
  pred <- c(pred, yy - predict(fit, pData))
  cover <- c(cover, apply(confint(fit)[-1,,drop=FALSE], 1, prod) <= 0)
  setTxtProgressBar(pb, i)
}

head(res[,1])
```

2 Tracé de l'histogramme du slide 17

```
hist(res[,1],breaks = 50, probability = TRUE, xlab = 'beta hat')
```

Les valeurs estimées sont loin de réfléter la vraie valeur qui est égale à 0. La plupart de ces estimations tournent autour des valeurs +1.5 et -1.5. La distribution d'échantillonnage de $\hat{\beta}$ est grossièrement déformée car nous avons utilisé l'ensemble des données pour la sélection du modèle. On est en présence d'un phénomène de surapprentissage puisque le vrai modèle dans ce cas est le modèle nul (intercept seulement). Donc nous pouvons dire que le critère de BIC ne peut pas être utilisé pour une sélection précise des variables dans les problèmes de grande dimension ou de malédictions de données.

3 Calculer les indicateurs du Slide 19 et d'autres slides

```
summary(res[,1])

mse<-mean(pred^2)

er<-crossprod(pred)/length(pred)#MSPE

nrow(res)/N # Limite supérieure du nombre de variables à sélectionner

median(res[,"Pr(>|t|)"]) # p-valeur médiane

range(res[,"Pr(>|t|)"]) # p-valeur intervalles de confiance à 95%

mean(cover)
```

Le quartile est de -1.51 et le troisème quartile est de 1.337. En moyenne, les modèles sélectionnés ont obtenu une erreur quadratique moyenne (MSE) de 2.15.

1.3 DEVOIR

L'Analyse en composantes principales est une méthode statistique exploratoire permettant une description essentiellement graphique de l'information contenue dans des grands tableaux de données. Dans la plupart des applications, il s'agit d'étudier p variables mesurées sur un ensemble de n individus. Lorsque n et p sont grands on cherche à synthétiser la masse d'informations sous une forme exploitable et compréhensible.

1. Le problème majeur pouvant surgir dans une ACP en grande dimension concerne le choix du nombre d'axes à retenir pour l'interprétation. Les conséquences de ce choix sont importantes car si le nombre d'axes n'est pas correctement choisi, on peut introduire un bruit ou une perte d'informations importante dans l'analyse.
2. Une des étapes les plus importantes en statistique en grande dimension est la vérification des hypothèses et conclusions. Avec l'augmentation du nombre d'hypothèses, dont chacune est testée à un seuil α, le risque d'erreur de rejet de l'hypothèse nulle augmente. Une analyse en grande dimension consiste souvent à considérer des centaines (voir des milliers) d'inférences.
3. Le *familiy-wise error rates* (FWER) est la probabilité de faire une ou plusieurs fausses découvertes, ou des erreurs de type I lors de l'exécution de tests d'hypothèses multiples. Les procédures de contrôle de type FWER ne sont efficaces que pour des petites familles d'inférences. Dans le cas de nombreuses familles d'inférences en grande dimension, il est plus efficace de contrôler par une méthode *false discovery rates* (FDR), c'est-à-dire la valeur attendue du taux de rejets erronés sur l'ensemble des rejets.

TP2

2 - Sélection de modèle

2-1 Les données

Ces données proviennent d'une étude qui a examiné la corrélation entre le niveau d'antigène spécifique de la prostate et un certain nombre de mesures cliniques chez les hommes sur le point de subir une prostatectomie radicale.

Téléchargement des données

```
library(lasso2)

data("Prostate")
```

Nous vérifions que les données correspondent bien au tableau de description et
faisons quelques statistiques descriptives : dim, names, summary, cor, hist

```
summary(Prostate)
Prostate$svi <- as.factor(Prostate$svi)
Prostate$gleason <- as.factor(Prostate$gleason)
```

Les variables svi et gleason sont qualitatives.

```
Prostate$svi=as.factor(Prostate$svi)
Prostate$gleason=as.factor(Prostate$gleason)

dim(Prostate)
```

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

hist(Prostate$lcavol)
hist(Prostate$lweight)
hist(Prostate$age)
hist(Prostate$lbph)
#hist(Prostate$svi)
hist(Prostate$lcp)
#hist(Prostate$gleason)
hist(Prostate$pgg45)
```

Il s'agit d'une matrice de données comportant 97 lignes et 9 colonnes. On remarque aussi que la distribution des variables lpsa, lcavol, weight et âge semblent suivre une loi normale.

Données train et test

Les valeurs de lpsa sont rangées par ordre croissant. Nous divisons ici le jeu de données en un échantillon d'apprentissage pour estimer les modèles et en un échantillon de test pour comparer les erreurs de prédiction. On conserve 1/4 des données pour l'échantillon de test.

```r
ind.test=4*c(1:22)

Prostate.app=Prostate[-ind.test,]

Prostate.test=Prostate[c(ind.test),]
```

On donne quelques statistiques descriptives des données : Prostate.app et Prostate.test

```r
dim(Prostate.test)
dim(Prostate.app)

ntest=length(Prostate.test$lpsa)
napp=length(Prostate.app$lpsa)

summary(Prostate.app)
summary(Prostate.test)
```

On obteint donc un jeu d'apprentissage comportant 75 lignes et 9 colonnes et un jeu de test de 22 lignes et 9 colonnes.

2.2 Modèle linéaire complet

Une fonction utile de graphe des résidus

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

2.2.1 Estimation du modèle et graphes des résidus

```
modlin=lm(lpsa~., data=Prostate.app)
summary(modlin)

#Résidus

res=residuals(modlin)

#Regroupement des graphiques sur la même 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)
```

Les p-valeurs de certaines variables comme lcavol, lweight, age et svi1 sont rès significatives. Selon la répartition des points (uniformément répartis), on peut dire que les résidus sont sans biais. En regardant l'histogramme des résidus, nous pouvons dire que leur distribution suit une loi normale avec des quantiles compris entre -1.86 et +1.5.

2.2.2 Erreur d'apprentissage

Calculons l'erreur d'apprentissage

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

L'erreur d'apprentissage obtenue est de 0.46.

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)
```

On obtient 0.45 comme valeur de l'erreur sur l'échantillon de test.

Les erreurs d'apprentissage et de test sont donc presqu'égales.

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)
```

Nom des variables : gleason1 =6, gleason2 =7, gleason3 =8, gleason4 =9 , la somme des coefficients associés à ces variables est nulle. svi1=0, svi2=1. 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)
```

La sélection par crière AIC et backward nous fournit comme meilleur modèle celui en fonction des variables lcavol, lweight, age, lbph, svi. Ceci avec un AIC = 41.19. On note que certaines variables comme âge et lbph 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)
```

Avec la sélection par AIC et forward, on obtient presque les mêmes résultats qu'avec la sélection par AIC backward.

3.3 Sélection par AIC et stepwise

```
modselect=stepAIC(modlin2,~.,trace=TRUE, direction=c("both"))
summary(modselect)
```

On retrouve ici le meme 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)
```

Avec la méthode BIC et stepwise, on obtient moins de variables dans le meilleur sélectionné comparé aux méthodes précédentes. Le modèle sélectionné est plus parcimonieux dans le cas de la méthode BIC et stepwise.

3.5 Erreur sur l'échantillon d'apprentissage

Modèle stepwise AIC

```
mean((predict(modselect)-Prostate.app[,"lpsa"])**2)
```

Avec la méthode stepwise AIC, on trouve pour valeur 0.49 comme erreur sur l'échantillon d'apprentissage.

Modèle stepwise BIC

```
mean((predict(modselect_BIC)-Prostate.app[,"lpsa"])**2)
```

Avec la méthode stepwise BIC, on trouve pour valeur 0.53 comme erreur sur l'échantillon d'apprentissage.

3.6 Calcul de l'erreur sur l'échantillon test

Modèle stepwise AIC

```
mean((predict(modselect,newdata=Prostate.test)-Prostate.test[,"lpsa"])**2)
```

Avec la méthode stepwise AIC, on trouve pour valeur 0.46 comme erreur sur l'échantillon test.

Modèle stepwise BIC

```
mean((predict(modselect_BIC,newdata=Prostate.test)-Prostate.test[,"lpsa"])**2)
```

Avec la méthode stepwise BIC, on trouve pour valeur 0.40 comme erreur sur l'échantillon test.

Les modèles sélectionnés ont une erreur plus grande que le modèle linéaire comprenant toutes les variables sur l'échantillon d'apprentissage. 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)
library(lasso2)

mod.ridge=lm.ridge(lpsa~.,data=Prostate.app, lambda=seq(0,20,0.1))

par(mfrow=c(1,1))
plot(mod.ridge)
```

Evolution 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

Ici, on calcule 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)
```

-les 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
```

On calcule ici 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
```

Nous traçons ensuite des valeurs prédites en fonction des valeurs observées

```
plot(Prostate.app$lpsa,fit.rid)
abline(0,1)
```

On calcule et on fait le tracé des résidus

```
res.rid=fit.rid-Prostate.app[,"lpsa"]
plot.res(fit.rid,res.rid,titre="Tracé des résidus")
```

Selon la répartition des points (uniformément répartis), on peut dire que les résidus sont sans biais.

Erreurd'apprentissage

```
mean(res.rid**2)
```

On trouve pour valeur 0.478 comme erreur d'apprentissage.

4.4 Prévision sur l'échantillon test

Les 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)
```

Les 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)
```

On trouve pour valeur 0.424 comme erreur sur l'échantillon test.

L'erreur d'apprentissage est légèrement plus élevée que pour le modèle linéaire sans pénalisation. 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

5.1 Librairie Lasso2

```
library(lasso2)
```

5.2 Construction du modèle

```
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 L1 du vecteur des coeficients 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

On visualise ici les coefficients du modèle

```
coefficients=coef(l1c.P)
plot(l1c.P,col=1:11,lty=1:3,type="l",main="Coefficients avec le terme constant")
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, main="Coefficients après suppression du terme constant")
legend("topleft",legend=
         colnames(coefficients[,-1]),col=1:10,lty=1:3)
```

En fonction de différentes valeurs de la pénalité, on obtient différentes valeurs pour les coefficients. Afin de faire un bon choix de la pénalité, nous procédons par la méthode de validation croisée.

5.4 Sélection de la pénalité par validation croisée

On procède à la validation croisée pour sélectionner la pénalité.

```
vc=gcv(l1c.P)
crit.vc=vc[,"gcv"]
bound_opt=vc[which.min(crit.vc),"rel.bound"]

l1c.opt <- l1ce(lpsa ~ ., Prostate.app,
                bound=bound_opt, absolute.t=FALSE)
coef=coef(l1c.opt)
```

Par la méthode de validation croisée, on obtient 0.95 comme valeur de la meilleure pénalité qui nous a servi à optimiser notre modèle précédent en utilisant que cette meilleure pénalité dans le modèle.

5.5 Erreur d'apprentissage

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

On trouve pour valeur 0.46 comme erreur d'apprentissage.

5.6 Erreur sur l'échantillon test

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

On trouve pour valeur 0.44 comme erreur sur l'échantillon test.

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)
```

On construit ici des vecteurs indicatrices pour les variables qualitatives

```
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)
```

On crée une 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 les 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")
```

Ici, on a tracé les coefficients des différentes variables du modèle Lasso.

5.11 Sélection de la pénalité par validation croisée

Nous appliquons la méthode de validation croisée afin de sélectionner la meilleure pénalité.Ensuite, on optimise le modèle avec la valeur de la meilleure pénalité trouvée par la méthode de 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)
```

Nous trouvons la valeur de 0.0444 comme meilleure pénalité.

5.12 Erreur d'apprentissage

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

Avec le modèle optimisé, nous trouvons 0.48 comme valeur de l'erreur d'apprentissage.

5.13 Erreur sur l'échantillon test

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

Avec le modèle optimisé, nous trouvons environ 0.39 comme erreur de prédiction commise sur le jeu de test.

Il est à noter que l'erreur de test est un peu plus petite que l'erreur d'apprentissage.

5.14 Elastic Net

La méthode Elastic Net est une méthode qui permet de résoudre les problèmes liés à la méthode Lasso (comme par exemples le problème de colinéarité entre les variables et le problème du nombre de variables à sélectionner lié au fléau de la dimension).

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)
```

Ici, nous avons crée un modèle avec 0.5 comme paramètre alpha. On a procédé ensuite par la méthode de validation croisée pour choisir le meilleur paramètre lambda. Ceci nous a permis d'optimiser notre modèle. On utilise donc le modèle optmisé pour faire des prédictions et calculer les erreurs commises sur le jeu d'apprentissage et sur le jeu de test.

Erreur d'apprentissage

```
app.elnet=predict(a.elnet,newx=xx.app)
mean((app.elnet-Prostate.app[,9])^2)
```

Nous trouvons la valeur 0.6 comme erreur d'apprentissage.

Erreur de prédiction

```
predi.elnet=predict(a.elnet,newx=xx.test)
mean((predi.elnet-Prostate.test[,9])^2)
```

Nous trouvons la valeur 0.37 comme erreur de test.

L'erreur commise sur le jeu de test est plus petite que celle commise sur le jeu d'apprentissage.

TP3 Chap3 Pratique de la régression Ridge et Elasticnet

Dans ce TP, ous nous situons dans le cadre de la régression logistique avec une variable cible qualitative binaire. Les propriétés de régularisation de ridge et elasticnet devraient se révéler décisives. Encore faut-il savoir/pouvoir déterminer les valeurs adéquates des paramètres de ces algorithmes. Ils pèsent fortement sur la qualité des résultats.

3.1 Données et régression logistique glm()

3.1.1 Base de données "adult"

Nos données sont dérivées de la base "Adult Data Set". Elle a été retravaillée: les variables quantitatives ont été discrétisées, puis transformées en variables indicatrices via un codage disjonctif complet; les variables catégorielles ont également été binarisées via le meme procédé. En définitive, nous disposons de $p = 123$ variables explicatives, toutes binaires. La variable cible est également binaire, définie dans positive, negative.

3.1.2 Importation des bases d'apprentissage et de test - Quelques vérifications

```
#changer le dossier courant
setwd("C:/Users/im2ag/Desktop/M2 SSD/COURS ET TP STAT EN GRDE DIMENSION")

#charger les données d'apprentissage
data <- read.csv("adult.csv")
head(data)
print(dim(data))
any(is.na(data))
```

On sépare les données en un jeu d'apprentissage et de test

```
ind.test=4*c(1:8138)

DTrain=data[-ind.test,]

DTest=data[c(ind.test),]
```

```r
dim(DTrain)
dim(DTest)
nrow(DTest)
```

Nous vérifions la répartition des classes. Le modèle par défaut consisterait a prédire systématiquement la classe majoritaire negative.

Répartition de la variable income

```r
print(prop.table(table(DTrain$income)))
print(prop.table(table(DTest$income)))
```

La répartition des classes de l'échantillon de test est très similaire a celle de l'échantillon d'apprentissage. Avec le modèle par défaut, prédiction systématique de la classe majoritaire négative, le taux d'erreur serait de 23.28%. Il s'agit de faire mieux avec les différentes variantes de la régression logistique que nous mettrons en oeuvre dans ce qui suit.

Nous plaçons les variables explicatives et la variable cible dans deux structures distinctes,une matrice pour les premières, un vecteur pour la seconde.

```r
#XTrain <- as.data.frame(DTrain[,-10])
#yTrain <- as.numeric(DTrain[,10])

XTrain <- as.data.frame(DTrain[,-10])
yTrain <- as.factor(DTrain[,10])

is.matrix(XTrain)
```

Nous placons également la matrice des descripteurs dans une structure dédiée. Matrice en test

```r
XTest <- as.matrix(DTest[,-10])
```

La fonction glm() du package stats est l'outil privilégié pour la régression logistique sous R. Nous l'appliquons à l'échantillon d'apprentissage.

```r
#reg1 <- glm(income~., data = DTrain, family = "binomial")
#summary(reg1)

reg1 <- glm(as.factor(DTrain$income)~., data = DTrain, family = "binomial")
summary(reg1)
```

Le modèle est inutilisable. Certains coeficients n'ont pas été estimés et correspondent a la valeur NA (NotAvailable).Nous ne pouvons ni les interpréter, ni les déployer sur des individus supplémentaires. De fait, nous n'avons meme pas essayé d'appliquer le modèle sur l'échantillon test pour en mesurer les performances.

3.1.3 La librairie glmnet

Elle est efficace (rapidité des calculs, qualité des résultats), et surtout, elle propose toute une panoplie d'outils qui se révèlent précieux dans l'analyse exploratoire, lors de la recherche des combinaisons adéquates des paramètres λ et α.

Régression logistique non-régularisée

```r
library(glmnet)
```

Nous désactivons la standardisation (standardize = FALSE) des variables explicatives parce que nous savons qu'elles sont toutes binaires, définies de facto sur la meme échelle. Sur une base quelconque, en l'absence d'informations précises sur les variables, il est plus prudent d'activer l'option.

Nous lancons la régression logistique sans paramètre de régularisation ($\lambda = 0$)

```r
library(caret)
setwd("C:/Users/im2ag/Desktop/M2 SSD/COURS ET TP STAT EN GRDE DIMENSION")
#Importation
adult <- read.csv("adult.csv", sep=",", header=TRUE)

#Discrétisation
adult <- data.frame(apply(adult, 2, as.factor))

#One hot encoding
dmy <- caret::dummyVars(~.,adult[-which(names(adult)=="income")])
adultOH <- data.frame(cbind(predict(dmy,newdata = adult[-which(names(adult)=="income")]), adult$income)
names(adultOH)[length(names(adultOH))] <- "income" #Remise du bon nom de variable pour la dernière

#Séparation train/test
trainIndex <- createDataPartition(adultOH$income, p=0.01, list=FALSE, times=1)
DTrain <- adultOH[trainIndex,]
DTest <- adultOH[-trainIndex,]

print(prop.table(table(DTrain$income)))
print(prop.table(table(DTest$income)))

XTrain <- as.matrix(DTrain[,-which(names(adult)=="income")])
yTrain <- as.matrix(DTrain[,which(names(adult)=="income")])

XTest <- as.matrix(DTest[,-which(names(adult)=="income")])

reg2 <- glmnet(XTrain,yTrain,family="binomial",standardize=FALSE,lambda=0,nlambda = 1)

print(reg2)
```

Nous appliquons le modele sur léchantillon test avec predict().

```r
yp2 <- predict(reg2,XTest,type="class",s=c(0))

print(table(yp2))
```

Les prédictions sont reparties entre les deux classes, il n'y a pas de prédiction systématique.

Taux d'erreur

```r
print(sum(DTest$income!= yp2)/nrow(DTest))
```

Les difficultés (le ratio p/ntrain élevé et, surtout, les variables constituées de constantes) ont eu raison de l'algorithme d'apprentissage en l'absence de contraintes sur les coefficients. L'inspection des variables préalablement à l'apprentissage est donc très importante. Il aurait fallu exclure demblée ces variables à probleme. Nous continuons en l'état néanmoins en espérant que les mécanismes de régularisation des régressions ridge et elasticnet nous aident à résoudre le problème.

Régression Ridge

```
ridge2<-glmnet(XTrain, yTrain,family="binomial",standardize=FALSE,alpha=0)
plot(ridge2,xvar="lambda")
```

A lissue de l'apprentissage, nous disposons d'un vecteur de coefficients β^i pour chaque λ_i. Tous les coefficients sont nuls lorsque $\lambda = \lambda_{max}$. Nous pouvons afficher une "Ridge path coefficients" permettant de juger de la dispersion des coefficients au regard de λ. Contrairement à la régression Lasso, Ridge ne sait pas fixer sélectivement les coefficients à la valeur 0 et ne permet donc pas de réaliser une sélection de variables.

Afin d'optimiser le paramètre lambda, nous procédons par une validation croisée.

```
set.seed(1)
cv.ridge2 <- cv.glmnet(XTrain,yTrain,family="binomial",type.measure="class", nfolds=10,alpha=0,keep=TRU

plot(cv.ridge2,main="Taux d'erreur en validation croisée vs.log(lambda)")
```

Nous affichons ci-dessous la suite de λ_i; le taux d'erreur associé; le nombre de coefficients non-nuls (qui n'évolue pas puisque nous utilisons une régression ridge). Affichage

```
print(cbind(cv.ridge2$lambda,cv.ridge2$cvm,cv.ridge2$nzero))
```

Identifier visuellement les éléments importants dans cette liste n'est pas aisé. Nous le faisons par le calcul. Tout dabord, nous affichons l'erreur minimale

```
print(min(cv.ridge2$cvm))
```

```
#lambda qui minimise l'erreur
print(cv.ridge2$lambda.min)

#son logarithme
print(log(cv.ridge2$lambda.min))

#lambda le plus élevé en 1-se rule
print(cv.ridge2$lambda.1se)

#son logarithme
print(log(cv.ridge2$lambda.1se))
```

Prédiction sur l'échantillon test

```
yr2 <- predict(cv.ridge2,XTest,s=c(cv.ridge2$lambda.min,cv.ridge2$lambda.1se),
               type="class")

print(head(yr2))
```

Nous avons une matrice avec autant de colonnes que de valeurs de λ essayé. Le nombre de lignes correspond à ntest, l'effectif de léchantillon test.

Nous calculons les taux d'erreur pour les deux prédictions.

```
#erreur : lambda.min
print((sum(DTest$income!= yr2[,1])/nrow(DTest)))

#erreur : lambda.1se
print((sum(DTest$income!= yr2[,2])/nrow(DTest)))
```

Les deux font mieux que la prédiction systématique. Malgré les embuches, la régression ridge a su tirer parti des données.

3.1.4 Régression elasticnet

```
enet3 <- glmnet(XTrain,yTrain,family="binomial",standardize=FALSE,alpha=0.8)
plot(enet3,xvar="lambda")

print(enet3$lambda)#liste des valeurs de lambda testés
print(cbind(enet3$lambda,enet3$df))# nombre de variables selectionnées vs.lambda (alpha = 0.8)
```

On procède par validation croisée pour l'optmisation

```
cv.enet3 <- cv.glmnet(XTrain,yTrain,family="binomial",type.measure="class",
                      nfolds= 10,alpha=0.8,foldid=cv.ridge2$foldid)
plot(cv.enet3,main="Taux d'erreur en validation croisée vs.log(lambda)")

#lambda min
print(cv.enet3$lambda.min)

#prediction
ye3 <- predict(cv.enet3,XTest,s=c(cv.enet3$lambda.min),type="class")
print(table(ye3))

#taux d'erreur
print(sum(DTest$classe!= ye3)/nrow(DTest))
```

Après la validation croisée, on a fait la prédiction en utilisant le jeu de test et en prenant la valeur minimale de λ obtenue par validation croisée. L'erreur de prédiction est ensuite calculée.

TP4 Exemples sur le lien du mail.

Exemple 1

Generate data

```
library(MASS)
library(glmnet)
```

A partir de la loi normale, nous générons une matrice de données de taille $n = 1000$ (individus) et $p = 5000$ (variables). Nous la partageons ensuite en un jeu d'apprentissage (2/3) et un jeu de test(1/3). Nous prenons dans l'exemple 1, 15 comme nombre de vrais prédicteurs.

```r
# Generate data
set.seed(19875)  # Set seed for reproducibility
n <- 1000  # Number of observations
p <- 5000  # Number of predictors included in model
real_p <- 15  # Number of true predictors
x <- matrix(rnorm(n*p), nrow=n, ncol=p)
y <- apply(x[,1:real_p], 1, sum) + rnorm(n)

# Split data into train (2/3) and test (1/3) sets
train_rows <- sample(1:n, .66*n)
x.train <- x[train_rows, ]
x.test <- x[-train_rows, ]

y.train <- y[train_rows]
y.test <- y[-train_rows]
```

Fit models

Ici, dans un premier temps, nous implémentons les modèles Lasso, Ridge et Elastic Net avec notre jeu d'apprentissage. Dans un second temps, nous procédons par la méthode de validation croisée en utilisant 10-folds pour différentes valeurs du paramètre alpha pris dans la séquence de 0 à 1 par pas de 0.1.

```r
# Fit models
# (For plots on left):
fit.lasso <- glmnet(x.train, y.train, family="gaussian", alpha=1)
fit.ridge <- glmnet(x.train, y.train, family="gaussian", alpha=0)
fit.elnet <- glmnet(x.train, y.train, family="gaussian", alpha=.5)


# 10-fold Cross validation for each alpha = 0, 0.1, ... , 0.9, 1.0
# (For plots on Right)
for (i in 0:10) {
  assign(paste("fit", i, sep=""), cv.glmnet(x.train, y.train, type.measure="mse",
                                            alpha=i/10,family="gaussian"))
}
```

Plot solution path and cross-validated MSE as function of λ

On trace ici les erreurs quadratiques moyennes des différents modèles.

```r
# Plot solution paths:
par(mfrow=c(3,2))
# For plotting options, type '?plot.glmnet' in R console
plot(fit.lasso, xvar="lambda")
plot(fit10, main="LASSO")

plot(fit.ridge, xvar="lambda")
plot(fit0, main="Ridge")

plot(fit.elnet, xvar="lambda")
plot(fit5, main="Elastic Net")
```

MSE on test set

Nous faisons des prédictions sur le jeu de test en utilisant les différents modèles et nous calculons par la suite les erreurs quadratiques moyennes commises afin de savoir lequel de ces modèles est le meilleur.

```
yhat0 <- predict(fit0, s=fit0$lambda.1se, newx=x.test)
yhat1 <- predict(fit1, s=fit1$lambda.1se, newx=x.test)
yhat2 <- predict(fit2, s=fit2$lambda.1se, newx=x.test)
yhat3 <- predict(fit3, s=fit3$lambda.1se, newx=x.test)
yhat4 <- predict(fit4, s=fit4$lambda.1se, newx=x.test)
yhat5 <- predict(fit5, s=fit5$lambda.1se, newx=x.test)
yhat6 <- predict(fit6, s=fit6$lambda.1se, newx=x.test)
yhat7 <- predict(fit7, s=fit7$lambda.1se, newx=x.test)
yhat8 <- predict(fit8, s=fit8$lambda.1se, newx=x.test)
yhat9 <- predict(fit9, s=fit9$lambda.1se, newx=x.test)
yhat10 <- predict(fit10, s=fit10$lambda.1se, newx=x.test)

mse0 <- mean((y.test - yhat0)^2)
mse1 <- mean((y.test - yhat1)^2)
mse2 <- mean((y.test - yhat2)^2)
mse3 <- mean((y.test - yhat3)^2)
mse4 <- mean((y.test - yhat4)^2)
mse5 <- mean((y.test - yhat5)^2)
mse6 <- mean((y.test - yhat6)^2)
mse7 <- mean((y.test - yhat7)^2)
mse8 <- mean((y.test - yhat8)^2)
mse9 <- mean((y.test - yhat9)^2)
mse10 <- mean((y.test - yhat10)^2)
```

Parmi les modèles implémentés, compte tenu des erreurs obtenues, le modèle Lasso est le meilleur dans cet exemple. C'est le Lasso qui marche parce que le nombre de vrais prédicteurs choisi est très petit (15 vrais prédicteurs). Lasso est un modèle parcimonieux qui ne choisit qu'une variable dès qu'il y a colinéarité entre certaines variables. De plus il ne sélectionne pas plus que n variables quand p est très supérieur à n. Le lasso est bon pour capter un petit nombre de prédicteurs à travers plusieurs.

Exemple 2

Nous refaisons la même chose qu'à l'exemple 1 en prenant 1500 comme nombre de vrais prédicteurs.

Generate Data

```
library(MASS)
library(glmnet)
```

A partir de la loi normale, nous générons une matrice de données de taille $n = 1000$ (individus) et $p = 5000$ (variables). Nous la partageons ensuite en un jeu d'apprentissage (2/3) et un jeu de test(1/3). Nous prenons dans l'exemple 2, 1500 comme nombre de vrais prédicteurs.

```
# Generate data
set.seed(19874)
```

```
n <- 1000    # Number of observations
p <- 5000     # Number of predictors included in model
real_p <- 1500  # Number of true predictors
x <- matrix(rnorm(n*p), nrow=n, ncol=p)
y <- apply(x[,1:real_p], 1, sum) + rnorm(n)

# Split data into train and test sets
train_rows <- sample(1:n, .66*n)
x.train <- x[train_rows, ]
x.test <- x[-train_rows, ]

y.train <- y[train_rows]
y.test <- y[-train_rows]
```

Fit models

Nous implémentons les modèles Lasso, Ridge et Elastic Net avec notre jeu d'apprentissage. Ensuite, nous procédons par la méthode de validation croisée en utilisant 10-folds pour différentes valeurs du paramètre alpha pris dans la séquence de 0 à 1 par pas de 0.1

```
# Fit models:
fit.lasso <- glmnet(x.train, y.train, family="gaussian", alpha=1)
fit.ridge <- glmnet(x.train, y.train, family="gaussian", alpha=0)
fit.elnet <- glmnet(x.train, y.train, family="gaussian", alpha=.5)


# 10-fold Cross validation for each alpha = 0, 0.1, ... , 0.9, 1.0
fit.lasso.cv <- cv.glmnet(x.train, y.train, type.measure="mse", alpha=1,
                          family="gaussian")
fit.ridge.cv <- cv.glmnet(x.train, y.train, type.measure="mse", alpha=0,
                          family="gaussian")
fit.elnet.cv <- cv.glmnet(x.train, y.train, type.measure="mse", alpha=.5,
                          family="gaussian")

for (i in 0:10) {
  assign(paste("fit", i, sep=""), cv.glmnet(x.train, y.train, type.measure="mse",
                                            alpha=i/10,family="gaussian"))
}
```

Plot solution path and cross-validated MSE as function of λ

On trace ici les erreurs quadratiques moyennes des différents modèles.

```
# Plot solution paths:
par(mfrow=c(3,2))
# For plotting options, type '?plot.glmnet' in R console
plot(fit.lasso, xvar="lambda")
plot(fit10, main="LASSO")

plot(fit.ridge, xvar="lambda")
plot(fit0, main="Ridge")
```

```
plot(fit.elnet, xvar="lambda")
plot(fit5, main="Elastic Net")
```

MSE on test set

Nous faisons des prédictions sur le jeu de test en utilisant les différents modèles et nous calculons par la suite les erreurs quadratiques moyennes commises afin de savoir lequel de ces modèles est le meilleur.

```
yhat0 <- predict(fit0, s=fit0$lambda.1se, newx=x.test)
yhat1 <- predict(fit1, s=fit1$lambda.1se, newx=x.test)
yhat2 <- predict(fit2, s=fit2$lambda.1se, newx=x.test)
yhat3 <- predict(fit3, s=fit3$lambda.1se, newx=x.test)
yhat4 <- predict(fit4, s=fit4$lambda.1se, newx=x.test)
yhat5 <- predict(fit5, s=fit5$lambda.1se, newx=x.test)
yhat6 <- predict(fit6, s=fit6$lambda.1se, newx=x.test)
yhat7 <- predict(fit7, s=fit7$lambda.1se, newx=x.test)
yhat8 <- predict(fit8, s=fit8$lambda.1se, newx=x.test)
yhat9 <- predict(fit9, s=fit9$lambda.1se, newx=x.test)
yhat10 <- predict(fit10, s=fit10$lambda.1se, newx=x.test)

mse0 <- mean((y.test - yhat0)^2)
mse1 <- mean((y.test - yhat1)^2)
mse2 <- mean((y.test - yhat2)^2)
mse3 <- mean((y.test - yhat3)^2)
mse4 <- mean((y.test - yhat4)^2)
mse5 <- mean((y.test - yhat5)^2)
mse6 <- mean((y.test - yhat6)^2)
mse7 <- mean((y.test - yhat7)^2)
mse8 <- mean((y.test - yhat8)^2)
mse9 <- mean((y.test - yhat9)^2)
mse10 <- mean((y.test - yhat10)^2)
```

Le modèle Lasso ne marche pas dans ce cas car le nombre de vrais prédicteurs (1500) est supérieur au nombre d'individus (1000) et Laaso ne peut pas choisir plus de 1000 prédicteurs. Compte-tenu des valeurs des obtenues, dans cet exemple c'est le modèle Ridge qui est le meilleur modèle. Ce modèle est généralement très bon pour la prédiction (même en grande dimension) mais n'est pas très interpétable.

Exemple 3

Dans cet exemple, à partir de la loi normale, nous générons une matrice de données de taille $n = 100$ (individus) et $p = 50$ (variables). Nous la partageons ensuite en un jeu d'apprentissage et un jeu de test.

```
# Generate data

set.seed(19873)
n <- 100    # Number of observations
p <- 50     # Number of predictors included in model
CovMatrix <- outer(1:p, 1:p, function(x,y) {.7^abs(x-y)})
x <- mvrnorm(n, rep(0,p), CovMatrix)
y <- 10 * apply(x[, 1:2], 1, sum) +
  5 * apply(x[, 3:4], 1, sum) +
```

```
  apply(x[, 5:14], 1, sum) +
  rnorm(n)

# Split data into train and test sets
train_rows <- sample(1:n, .66*n)
x.train <- x[train_rows, ]
x.test <- x[-train_rows, ]

y.train <- y[train_rows]
y.test <- y[-train_rows]
```

Fit models

Nous implémentons les modèles Lasso, Ridge et Elastic Net avec notre jeu d'apprentissage. Ensuite, nous procédons par la méthode de validation croisée en utilisant 10-folds pour différentes valeurs du paramètre alpha pris dans la séquence de 0 à 1 par pas de 0.1

```
fit.lasso <- glmnet(x.train, y.train, family="gaussian", alpha=1)
fit.ridge <- glmnet(x.train, y.train, family="gaussian", alpha=0)
fit.elnet <- glmnet(x.train, y.train, family="gaussian", alpha=.5)


# 10-fold Cross validation for each alpha = 0, 0.1, ... , 0.9, 1.0
fit.lasso.cv <- cv.glmnet(x.train, y.train, type.measure="mse", alpha=1,
                          family="gaussian")
fit.ridge.cv <- cv.glmnet(x.train, y.train, type.measure="mse", alpha=0,
                          family="gaussian")
fit.elnet.cv <- cv.glmnet(x.train, y.train, type.measure="mse", alpha=.5,
                          family="gaussian")

for (i in 0:10) {
  assign(paste("fit", i, sep=""), cv.glmnet(x.train, y.train, type.measure="mse",
                                            alpha=i/10,family="gaussian"))
}
```

Plot solution path and cross-validated MSE as function of λ

On trace ici les erreurs quadratiques moyennes des différents modèles.

```
# Plot solution paths:
par(mfrow=c(3,2))
# For plotting options, type '?plot.glmnet' in R console
plot(fit.lasso, xvar="lambda")
plot(fit10, main="LASSO")

plot(fit.ridge, xvar="lambda")
plot(fit0, main="Ridge")

plot(fit.elnet, xvar="lambda")
plot(fit5, main="Elastic Net")
```

MSE on test set

Nous faisons des prédictions sur le jeu de test en utilisant les différents modèles et nous calculons par la suite les erreurs quadratiques moyennes commises afin de savoir lequel de ces modèles est le meilleur.

```
yhat0 <- predict(fit0, s=fit0$lambda.1se, newx=x.test)
yhat1 <- predict(fit1, s=fit1$lambda.1se, newx=x.test)
yhat2 <- predict(fit2, s=fit2$lambda.1se, newx=x.test)
yhat3 <- predict(fit3, s=fit3$lambda.1se, newx=x.test)
yhat4 <- predict(fit4, s=fit4$lambda.1se, newx=x.test)
yhat5 <- predict(fit5, s=fit5$lambda.1se, newx=x.test)
yhat6 <- predict(fit6, s=fit6$lambda.1se, newx=x.test)
yhat7 <- predict(fit7, s=fit7$lambda.1se, newx=x.test)
yhat8 <- predict(fit8, s=fit8$lambda.1se, newx=x.test)
yhat9 <- predict(fit9, s=fit9$lambda.1se, newx=x.test)
yhat10 <- predict(fit10, s=fit10$lambda.1se, newx=x.test)

mse0 <- mean((y.test - yhat0)^2)
mse1 <- mean((y.test - yhat1)^2)
mse2 <- mean((y.test - yhat2)^2)
mse3 <- mean((y.test - yhat3)^2)
mse4 <- mean((y.test - yhat4)^2)
mse5 <- mean((y.test - yhat5)^2)
mse6 <- mean((y.test - yhat6)^2)
mse7 <- mean((y.test - yhat7)^2)
mse8 <- mean((y.test - yhat8)^2)
mse9 <- mean((y.test - yhat9)^2)
mse10 <- mean((y.test - yhat10)^2)
```

Dans cet exemple, compte-tenu des erreurs obtenues, nous pouvons dire que c'est le modèle Elastic Net qui est le meilleur modèle. On peut aussi voir que la meilleure solution est proche de Ridge mais pour $\alpha = 0$ on obtient une erreur quadratique un peu plus grande comparée aux erreurs pour les autres valeurs de α.

On peut conclure, au regard de nos divers résultats, qu'en fonction de la complexité du modèle, chacune des trois méthodes (Lasso, Ridge et Elastic Net) peut marcher.

DEVOIR: Différence entre la descente de gradient et la descente de coordonnées

La Descente de Gradient est un algorithme d'optimisation qui permet de trouver le minimum de n'importe quelle fonction convexe, c'est-à-dire une fonction qui a l'allure d'une belle vallée qui descend progressivement vers un unique minimum. A l'inverse, une fonction non-convexe est une fonction qui présente plusieurs minimums locaux et l'algorithme de Gradient Descent ne doit pas être utilisé sur ces fonctions, au risque de se bloquer au premier minima rencontré. En Machine Learning, on va utiliser l'algorithme de la Descente de Gradient dans les problèmes d'apprentissage supervisé pour minimiser la fonction de coût, qui justement est une fonction convexe (par exemple l'erreur quadratique moyenne). C'est cet algorithme qui fait que la machine apprend, c'est-à-dire trouve le meilleur modèle.L'algorithme est itératif et procède donc par améliorations successives. L'algorithme du gradient est également connu sous le nom d'algorithme de la plus forte pente ou de la plus profonde descente (steepest descent, en anglais) parce que le gradient est la pente de la fonction linéarisée au point courant et est donc, localement, sa plus forte pente (notion qui dépend du produit scalaire).

Alors que l'algorithme de descente de coordonnées typique utilise le gradient le long de chaque coordonnée, ou le long d'un hyperplan obtenu à partir d'un groupe de coordonnées, pour trouver le minimiseur le long de cette direction. La descente de coordonnées est un algorithme d'optimisation qui minimise successivement le long des directions de coordonnées pour trouver le minimum d'une fonction. À chaque itération, l'algorithme détermine une coordonnée ou un bloc de coordonnées via une règle de sélection de coordonnées, puis minimise exactement ou inexactement sur l'hyperplan de coordonnées correspondant tout en fixant toutes les autres coordonnées ou blocs de coordonnées. Une recherche de ligne le long de la direction des coordonnées peut être effectuée à l'itération actuelle pour déterminer la taille de pas appropriée. La descente de coordonnées est applicable à la fois dans des contextes différenciables et sans dérivé. La descente de coordonnées est basée sur l'idée que la minimisation d'une fonction multivariable peut être atteint en le minimisant dans une direction à la fois, c'est-à-dire en résolvant des problèmes d'optimisation univariés (ou du moins beaucoup plus simples) dans une boucle. 1 Dans le cas le plus simple de descente de coordonnées cycliques , on itère cycliquement dans les directions, une à la fois, en minimisant la fonction objectif par rapport à chaque direction de coordonnées à la fois.

Malgré le taux d'apprentissage de la procédure de descente de gradient (qui pourrait en effet accélérer la convergence), la comparaison entre les deux est juste au moins en termes de complexité.

TP5 Analyse de données fonctionnelles

1.2 Partie I

```
library(fda)
```

1.2.1 Examen des données

On charge les données et on retire les individus ayant des données manquantes de la manière suivante

```
dat0 = as.matrix(read.table("C:/Users/im2ag/Desktop/M2 SSD/COURS ET TP STAT EN GRDE DIMENSION/SGD/TP/Si

na_count = rowSums(is.na(dat0))#Vecteur des données manquantes par ligne

dat = dat0[na_count == 0,]#Ici on considère les journées sans les valeurs manquantes
```

La variable heures contient le temps dans la journee en heures : toutes les 6 minutes (240 points).

```
heures = c(0 :239)/10
```

1.

(a) On vérifie la répartition des valeurs pour les débits.

```
colnames(dat)<-heures
hist(dat,breaks = 1000)
```

La distribution présente une queue qui nous fait penser à un processus de Poisson.

(b) Expliquer pourquoi on peut appliquer la méthode de stabilisation de la variance pour un processus de Poisson.

On peut appliquer la méthode de stabilisation de la variance pour un processus de Poisson car ce faisant, on obtient une variance égale à 1 dans le cas du processus de Poisson. Ce qui permet de réduire les grandes valeurs.

(c) Est-ce que cela améliore la queue de distribution?

Pour en savoir plus, traçons l'histogramme de la racine carrée de nos données

```
hist(sqrt(dat), breaks=100, xlab="", main="Histogramme de la racine carrée des données")
```

Cette répartition est moins asymétrique comparée au graphe précédent. On remarque aussi que la stabilisation améliore bien la queue de la distribution.

2. Traçons des courbes pour quelques journées (au moins 5)

Nous traçons ici quelques courbes des données brutes et des données stabilisées.

```
matplot(heures,dat[5,], type="l")

for (i in 1:4 ){
lines(heures,dat[i,], type="l")
}

matplot(heures,sqrt(dat)[5,], type="l")

for (i in 1:4 ){
lines(heures,sqrt(dat)[i,], type="l")
}

dat = sqrt(dat)
plot(dat["19/04/2011",], type = "l", xlab="", ylab="")
lines(dat["03/01/2011",], col = "red")
lines(dat["15/04/2010",], col = "blue")
lines(dat["04/02/2010",], col = "green")
lines(dat["01/09/2010",], col = "yellow")
```

Les courbes des données stabilisées présentent moins de bruit comparées à celles des données brutes (bien sûr sans les données manquantes).

3. Traçons la courbe du débit moyen au cours de la journée.

```
plot(colMeans(dat), type="l")
```

Cette courbe présente une forme bimodale. Les deux pics représentent une augmentation du trafic aux horaires correspondants. Elle est un peu proche de la répartition des débits pour les données stabilisées avec la racine carrée.

1.2.2 Lissage des données

1. Lissage de la moyenne

Ici, on essaie avec plusieurs valeurs de λ pour en trouver une qui lisse correctement la moyenne tout en gardant un SSE correct. Ensuite, nous traçons la moyenne correspondante.

```
require(fda)

datmoy<-colMeans(dat)

nderiv = 2
norder = nderiv + 2
fdnames=list("heure", "jour", "debit")
basisobj = create.bspline.basis(breaks=heures, norder=norder)
lambda = 0.01
fdParobj = fdPar(fdobj=basisobj, Lfdobj=nderiv, lambda=lambda)
dat.fd = smooth.basis(argvals=heures, y=datmoy, fdParobj, fdnames=fdnames)$fd
plot(heures, datmoy, type="l", ylab="")
lines(dat.fd, col = "red")
```

En utilisant le code ci-dessus, pour les valeurs de λ choisies parmi (0.01,0.1,0.5,1,2,), on constate que pour $\lambda = 0.01$, on obtient une courbe plus lisse de la moyenne.

2. Lissage d'une courbe

On choisit de lisser la deuxième courbe du jeu de données. Nous faisons évoluer arbitrairement la valeur de λ jusqu'à obtenir une courbe assez lisse. On s'arrête à $\lambda = 1.5$. On représente en bleu sur le graphe ci-dessous, la courbe lissée.

```
nderiv = 2
norder = nderiv + 2
fdnames=list("heure", "jour", "debit")
basisobj = create.bspline.basis(breaks=heures, norder=norder)
lambda = 1.5
fdParobj = fdPar(fdobj=basisobj, Lfdobj=nderiv, lambda=lambda)
dat.fd = smooth.basis(argvals=heures, y=dat[2,], fdParobj, fdnames=fdnames)$fd
plot(heures, dat[2,], type="l", ylab="")
lines(dat.fd, col = "blue", lwd = 2)
```

3. Lissage de plusieurs courbes

On lisse les mêmes courbes qui on été tracées dans la section précédente et on fait un tracé joint des courbes lissées (pour une valeur raisonnable de λ)

```
nderiv = 2
norder = nderiv + 2
fdnames=list("heure", "jour", "debit")
basisobj = create.bspline.basis(breaks=heures, norder=norder)
lambda = 1.5
fdParobj = fdPar(fdobj=basisobj, Lfdobj=nderiv, lambda=lambda)
dat.fd.1 = smooth.basis(argvals=heures, y=t(dat[1:5,]), fdParobj, fdnames=fdnames)$fd
plot(dat.fd.1)
```

1.3 Partie II

On fait une ACP fonctionnelle sur les données disponibles (avec $\lambda = 0$), et on représente la courbe de variance expliquée par les 7 premières valeurs propres.

```
 dat.fd.2 = smooth.basis(argvals=heures, y=t(dat), fdParobj, fdnames=fdnames)$fd
pca.fdParobj = fdPar(fdobj=basisobj, lambda=0)
dat.pca = pca.fd(dat.fd.2, nharm = 8, harmfdPar=pca.fdParobj)
plot(dat.pca$values[1:7],type='b', ylab= "Variance expliquée")
```

On remarque que que $%lambda = 0$ marche très bien: la courbe de départ est bien lisse. Compte tenu de la décroissance de la courbe, on peut considérer les 3 ou les 4 premières composantes.

Représentons les 3 premiers axes principaux

```
save=par(mfrow=c(4,1), cex=0.5, mar=c(2,2,2,2), oma=c(0,0,2,0))
plot(dat.pca, nx=150, harm=1:3)
par(save)
```

Pour observer l'influence du lissage, on applique le même procédé avec un paramètre de lissage significatif. Prenons $\lambda = 10$.

```
pca.fdParobj = fdPar(fdobj=basisobj, lambda=10)
dat.pca = pca.fd(dat.fd.2, nharm = 8, harmfdPar=pca.fdParobj)
round(cumsum(dat.pca$varprop[1:8])*100)
```

Les 3 premiers axes principaux sont représentés ci-dessous :

```
save=par(mfrow=c(4,1), cex=0.5, mar=c(2,2,2,2), oma=c(0,0,2,0))
plot(dat.pca, nx=150, harm=1:3)
par(save)
```

Dans cette partie, on essaie de regarder si l'ACP fonctionnelle sur la racine carrée des données est plus intéressante. On conserve la valeur $\lambda = 0$.

```
sqdat=sqrt(dat)
lambda=1
fdParobj = fdPar(fdobj=basisobj, Lfdobj=nderiv, lambda=lambda)
sqdat.fd = smooth.basis(argvals=heures, y=t(sqdat), fdParobj, fdnames=fdnames)$fd
pca.fdParobj = fdPar(fdobj=basisobj, lambda=0.0)
sqdat.pca = pca.fd(sqdat.fd, nharm = 8, harmfdPar=pca.fdParobj)
round(cumsum(sqdat.pca$varprop[1:8])*100)
```

On voit que la répartition des valeurs propres est plus régulière, même si l'inertie expliquée par les 3 premiers axes est pesque la même. Les composantes principales ci-dessous sont toujours interprétables de la même manière, mais sont plus marquées. Globalement, cette analyse n'apporte pas grand-chose de neuf.

```
save=par(mfrow=c(4,1), cex=0.5, mar=c(2,2,2,2), oma=c(0,0,2,0))
plot(sqdat.pca, nx=150, harm=1:3)
par(save)
```

Gestion des valeurs manquantes

On voudrait analyser aussi les journées dont le nombre de données manquantes est inférieur à 10 (mais non nul).

Nous faisons une sélection de ces données puis on affiche le nombre d'individus des différents jeux de données.

```
dat.1.9na=dat0[(na_count >= 1) & (na_count <= 9),]
nrow(dat0)
nrow(dat)
nrow(dat.1.9na)
```

Les données sélectionnées représentent à peu près 10% des données d'origine. Il y a un intérêt donc de les conserver.

Le code ci-dessous permet de lisser une journée avec des données manquantes. On choisit arbitrairement la courbe.

```
library(fda)
i=5
fdParobj.i = fdParobj
dat.i=dat.1.9na[i,]
hr=heures[!is.na(dat.i)]
dt=dat.i[!is.na(dat.i)]
fdParobj.i$lambda = fdParobj$lambda * length(hr)/length(heures)
dat.i.fd = smooth.basis(argvals=hr, y=dt, fdParobj.i, fdnames=fdnames)$fd
```

Afin de faire une comparaison, on représente les données avec la moyenne

```
dat.fd.2 = smooth.basis(argvals=heures, y=t(dat^2), fdParobj, fdnames=fdnames)$fd
dummy=plot(dat.i.fd, lwd=3, col='orange',
  main='donnees brutes et version lissee pour une journee reconstituée')
lines(mean(dat.fd.2))
points(hr,dt, pch='.')
```

On fait la même chose dans une boucle. On va créer une matrice de coefficients dont les colonnes seront formées par les vecteurs de coefficients des courbes lisses individuelles. On utilise la fonction cbind() pour placer les coefficients de chaque courbe dans une matrice commune.

```
fdParobj.x = fdParobj
coeffs=NULL
# fonction qui lisse la courbe de donnees x et retourne ses coefficients
unecourbe = function (x) {
 hr=heures[!is.na(x)]
 dt=x[!is.na(x)]
 fdParobj.x$lambda = fdParobj$lambda * length(hr)/length(heures)
 x.fd = smooth.basis(argvals=hr, y=dt, fdParobj.x)
 # le <<- modifie la variable globale 'coeff', pas une variable locale
 coef(x.fd)
 }
# on applique la fonction a toutes les lignes une par une
coeffs=apply(dat.1.9na, 1, unecourbe)
# on obtient un tableau de la bonne dimension
```

```
dim(coeffs)

dat.1.9na.fd = fd(coef=coeffs, basisobj=basisobj, fdnames=fdnames)

dummy=plot(dat.1.9na.fd, lty=1)
lines(mean(dat.1.9na.fd), lwd=3, col='orange')
```

On trace ensuite les courbes des moyennes des différentes données

```
dat.0.9na.fd = fd(coef=cbind(coef(dat.fd.2), coef(dat.1.9na.fd)), basisobj=basisobj, fdnames=fdnames)
dummy = plot(mean(dat.0.9na.fd),ylim=c(0,200))
lines(mean(dat.fd.2), col=2)
lines(mean(dat.1.9na.fd), col=3)
legend("topleft", c("total", "NA=0", "1 <= NA <= 9"), fill=1:6)
```

On constate que les données pour lesquelles des données sont manquantes ont une moyenne de débit plus basse entre 8 et 16h.