TP: Apprentissage non paramétrique en régression

Résumé

Comparaison sur le même jeu de données des qualités de prévision de plusieurs modèles obtenus par :
- *Splines*
- *Estimateurs à noyaux*
- *Polynomes locaux*
- *Modèles additifs généralisés*

Estimation d'une fonction de régression par projection sur une base d'ondelettes et seuillage ; débruitage d'une image.

1 Les données

Nous utiliserons les données **airquality** du logiciel R. Ces données correspondent à des mesures quotidiennes de la qualité de l'air. La variable à expliquer est le taux d'ozone, les variables explicatives sont
- Le rayonnement solaire
- La température
- La vitesse du vent

```
data(airquality)
summary(airquality)
airquality
```

Suppression des données manquantes et atypiques

```
airquality=airquality[complete.cases(airquality),]
airquality

airquality=airquality[airquality$Ozone<110,]
```

Description des données

TABLE 1 – Liste des 6 variables, la première est à expliquer

Ozone :	Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
Solar.R :	Solar radiation in Langleys in the frequency band 4000 to 7700 Angstroms from 0800 to 1200 hours at Central ParkLog to base 10 of total sales
Wind :	Average wind speed in miles per hour at 0700 and 1000 hours at La Guardia Airport
Temp :	Maximum daily temperature in degrees Fahrenheit at La Guardia Airport
Month :	Numeric Month (1 - 12)
Day :	Numeric Day of month (1 - 31)

```
dim(airquality)
cor(airquality)
```

Création d'un échantillon test

```
ind_app=sample(1:105,80)
airquality_app=airquality[ind_app,]
airquality_test=airquality[-ind_app,]
```

2 Modélisation unidimensionnelle

Dans cette partie, nous allons mettre en oeuvre des méthodes nonparamétriques pour modéliser la variable Ozone en fonction de la Température.

```
plot(airquality_app$Temp,airquality_app$Ozone)
```

2.1 Splines

Nous illustrons ici la méthode développée au Chapitre 3 (Estimation sur des bases de splines)

```
Ozone=airquality_app$Ozone
Temp=airquality_app$Temp
```

```
plot(Temp,Ozone, main="Smoothing Splines")
Ozone.spl <- smooth.spline(Temp, Ozone)
Ozone.spl
lines(Ozone.spl, col = "blue")
lines(smooth.spline(Temp, Ozone, df=15), lty=2,
 col = "red")
legend(60,100,c(paste("default [C.V.] => df =",
round(Ozone.spl$df,1))," df = 15"),
 col = c("blue","red"), lty = 1:2)
```

Essayer d'autres valeurs de df : 2 puis 20.

Calcul de l'erreur d'apprentissage

```
pred.app.spl=predict(Ozone.spl,Temp)
sqrt(mean((pred.app.spl$y-Ozone)**2))
```

Calcul de l'erreur sur l'échantillon test :

```
Temp_test=airquality_test$Temp
Ozone_test=airquality_test$Ozone
pred.spl=predict(Ozone.spl,Temp_test)
sqrt(mean((pred.spl$y-Ozone_test)**2))
```

2.2 Estimateurs à noyaux

Nous illustrons ici la méthode développée au Chapitre 4.

```
Ozone.ker <- ksmooth(Temp, Ozone,kernel="normal",
bandwidth=5)
Ozone.ker
Ozone.ker2 <- ksmooth(Temp, Ozone,kernel="normal",
bandwidth=10)
plot(Temp,Ozone, main="Noyau gaussien")
lines(Ozone.ker,col="blue")
lines(Ozone.ker2,col="red",lty=2)
```

Faire la même chose avec le noyau "box".
Pas de fonction "predict" ! On programme la fonction qui définit l'estimateur.

```
 noyau=function(z,X,Y,h)
#z est la valeur en laquelle on calcule
#l'estimateur, h la fenêtre,
#(X,Y) l'échantillon d'apprentissage.
{
Vect=exp(-((X-z)**2)/(2*h**2))
noyau=(sum(Y*Vect))/(sum(Vect))
}
```

Calcul de l'erreur d'apprentissage

```
m=length(Temp)
pred.ker.app=c(rep(0,m))
for (j in 1:m){
pred.ker.app[j]=noyau(Temp[j],Temp,Ozone,5)}

sqrt(mean((pred.ker.app-Ozone)**2))
```

Calcul de l'erreur sur l'échantillon test :

```
p=length(Temp_test)
pred.ker.test=c(rep(0,p))
for (j in 1:p){
pred.ker.test[j]=noyau(Temp_test[j],Temp,Ozone,5)}

sqrt(mean((pred.ker.test-Ozone_test)**2))
```

2.3 Polynômes locaux

Nous illustrons ici la méthode développée au Chapitre 5 du cours : Estimation ponctuelle par des polynômes locaux.

```
plot(Temp,Ozone, main="Polynomes locaux")
polyloc=loess(Ozone~Temp,span=0.5,family="gaussian")
pred_app=
```

```
predict(polyloc,data.frame(Temp=sort(unique(Temp))))
lines(sort(unique(Temp)),pred_app,col="blue")
```

Modifier le paramètre span et le noyau.

```
pred=predict(polyloc,data.frame(Temp))
X11()
plot(Ozone,pred)
abline(0,1)
```

Calcul de l'erreur d'apprentissage

```
pred.app.loess=predict(polyloc,Temp)
sqrt(mean((pred.app.loess-Ozone)**2))
```

Calcul de l'erreur sur l'échantillon test :

```
pred.loess=predict(polyloc,Temp_test)
sqrt(mean((pred.loess-Ozone_test)**2))
```

Attention aux éventuelles données manquantes. Si nécessaire :

```
ind=complete.cases(pred.loess)
sqrt(mean((pred.loess[ind]-Ozone_test[ind])**2))
```

3 Modélisation multidimensionnelle

Nous illustrons ici la méthode développée au Chapitre 8 du cours : Modèles additifs généralisés. Nous allons ici prendre en compte l'ensemble des variables explicatives : température, rayonnement solaire, vitesse du vent.

3.1 Modèles Additifs Généralisés

```
b<-gam(Ozone~s(Solar.R)+s(Wind)+s(Temp),
data=airquality_app)
summary(b)
plot(b)
```

Calcul de l'erreur d'apprentissage

```
pred.app.gam=predict(b)
sqrt(mean((pred.app.gam-Ozone)**2))
```

Calcul de l'erreur sur l'échantillon test :

```
pred.gam=predict(b, newdata=airquality_test)
sqrt(mean((pred.gam-Ozone_test)**2))
```

4 Bases d'ondelettes

Nous allons utiliser le logiciel MATLAB.

4.1 Analyse d'un signal

On considère la fonction

$$f(t) = (-3t + 4)\sin(4\pi t) + 5 * (\sin(8\pi t)).\mathbf{1}_{t<1/4}.$$

On observe

$$Y_i = f(t_i) + \epsilon_i, i = 1, \ldots, 2^8$$

où les ϵ_i sont i.i.d. de loi $\mathcal{N}(0, (0.2)^2)$ et $t_i = i/2^8$.

Génération des données bruitées :

```
N=2^8
t=(1:N)/N
y=(-3*t+4).*sin(4*pi*t)+5*(sin(8*pi*t)).*(t<1/4)
plot(t,y,'r')
yb=y+0.2*randn(1,N)
hold on
plot(t,yb,'.')
dn=8;
```

Choix de la base :

```
qmf=MakeONFilter('Haar',0);
```

4.1.1 Approximation linéaire

```
for L=0:dn
    clf
    wc=FWT_PO(yb,L,qmf);
    wchat=zeros(size(wc))
    wchat(1:2^L)=wc(1:2^L);
    fhat=IWT_PO(wchat,L,qmf);
    plot(t,y,t,fhat,'r')
    pause
end;
```

Essayer d'autres types de bases :

```
qmf=MakeONFilter('Daubechies',4);
```

```
qmf=MakeONFilter('Symmlet',8);
```

Visualisation des coefficients d'ondelettes :

```
 L=3
 wc=FWT_PO(yb,L,qmf);
 figure(2)
 PlotWaveCoeff(wc,L,0)
figure(3)
 plot(abs(wc))
```

Avec un signal RMN (beaucoup plus irrégulier) :

```
y=ReadSignal('HochNMR')
n=length(ch);
figure(1)
plot(y)
dn=floor(log2(n));
L=3
 wc=FWT_PO(y,L,qmf);
 figure(2)
 PlotWaveCoeff(wc,L,0)
figure(3)
 plot(abs(wc))
```

4.1.2 Seuillage

On effectue un seuillage "dur" et on prend 5 valeurs possibles pour le seuil. On conserve l'approximation lineaire jusqu'au niveau L et on seuille les coefs des niveaux >L :

```
p=5;
lambda=linspace(0.01,2,p)
y=(-3*t+4).*sin(4*pi*t)+5*(sin(8*pi*t)).*(t<1/4)
yb=y+0.2*randn(1,N);
qmf=MakeONFilter('Symmlet',8);
L=3;
wcyb=FWT_PO(yb,L,qmf);

nb=zeros(size(lambda));
k=0;

for (thr=lambda)
  clf
  thr
  k=k+1;
  wcnonlin=wcyb;
  wcnonlin(abs(wcnonlin)<thr)=0;
  wcnonlin(1:2^L)= wcyb(1:2^L);
  fnonlin=IWT_PO(wcnonlin,L,qmf);
  nb(k)=sum(abs(wcnonlin)>0);
  figure(1)
  plot(t,y,t,fnonlin,'r')
  pause
end
nb
```

nb représente le nombre de coefficients d'ondelettes qui sont non nuls pour chacune des valeurs du seuil.

- Refaire la même étude sur le signal RMN
- Faire les modifications nécessaires dans le programme ci-dessus pour ob-

tenir un seuillage ”doux“.

4.2 Analyse d’images

4.2.1 Approximation linéaire

```
img=ReadImage('Lenna')
figure(1)
GrayImage(img)
qmf=MakeONFilter('Haar',0)

for L=1:8
  L
  wc=FWT2_PO(img,L,qmf);
  wchat=zeros(size(wc));
  wchat(1:2^L,1:2^L)= wc(1:2^L,1:2^L);
  imghat=IWT2_PO(wchat,L,qmf);
  figure(2)
  GrayImage(imghat)
  pause
end
```

4.2.2 Seuillage

```
 p=10;
lambda=linspace(10,50,p)

nb=zeros(size(lambda))
k=0;
L=3
for (thr=lambda)
  clf
  thr
  k=k+1;
   wcimg=FWT2_PO(img,L,qmf);
  wcnonlin=wcimg;
  wcnonlin(abs(wcnonlin)<thr)=0;
   wcnonlin(1:2^L,1:2^L)= wcimg(1:2^L,1:2^L);
   imgnonlin=IWT2_PO(wcnonlin,L,qmf);
  nb(k)=sum(sum(abs(wcnonlin)>0));

  figure(1)
  GrayImage(imgnonlin)
   pause
end
 nb
```