SMDA - TP1: SVM et modèles de mélanges

Maha ELBAYAD

10 Octobre 2015

Application I : classification

Preliminaries

```r
emails=read.table("spam.txt",header=T,sep=';');
#Nombre des observations et des variables:
str(emails)
```

```
## 'data.frame':    4601 obs. of  58 variables:
##  $ A.1 : num  0 0.21 0.06 0 0 0 0 0 0.15 0.06 ...
##  $ A.2 : num  0.64 0.28 0 0 0 0 0 0 0 0.12 ...
##  $ A.3 : num  0.64 0.5 0.71 0 0 0 0 0 0.46 0.77 ...
##  $ A.4 : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.5 : num  0.32 0.14 1.23 0.63 0.63 1.85 1.92 1.88 0.61 0.19 ...
##  $ A.6 : num  0 0.28 0.19 0 0 0 0 0 0 0.32 ...
##  $ A.7 : num  0 0.21 0.19 0.31 0.31 0 0 0 0.3 0.38 ...
##  $ A.8 : num  0 0.07 0.12 0.63 0.63 1.85 0 1.88 0 0 ...
##  $ A.9 : num  0 0 0.64 0.31 0.31 0 0 0 0.92 0.06 ...
##  $ A.10: num  0 0.94 0.25 0.63 0.63 0 0.64 0 0.76 0 ...
##  $ A.11: num  0 0.21 0.38 0.31 0.31 0 0.96 0 0.76 0 ...
##  $ A.12: num  0.64 0.79 0.45 0.31 0.31 0 1.28 0 0.92 0.64 ...
##  $ A.13: num  0 0.65 0.12 0.31 0.31 0 0 0 0 0.25 ...
##  $ A.14: num  0 0.21 0 0 0 0 0 0 0 0 ...
##  $ A.15: num  0 0.14 1.75 0 0 0 0 0 0 0.12 ...
##  $ A.16: num  0.32 0.14 0.06 0.31 0.31 0 0.96 0 0 0 ...
##  $ A.17: num  0 0.07 0.06 0 0 0 0 0 0 0 ...
##  $ A.18: num  1.29 0.28 1.03 0 0 0 0.32 0 0.15 0.12 ...
##  $ A.19: num  1.93 3.47 1.36 3.18 3.18 0 3.85 0 1.23 1.67 ...
##  $ A.20: num  0 0 0.32 0 0 0 0 0 3.53 0.06 ...
##  $ A.21: num  0.96 1.59 0.51 0.31 0.31 0 0.64 0 2 0.71 ...
##  $ A.22: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.23: num  0 0.43 1.16 0 0 0 0 0 0 0.19 ...
##  $ A.24: num  0 0.43 0.06 0 0 0 0 0 0.15 0 ...
##  $ A.25: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.26: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.27: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.28: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.29: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.30: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.31: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.32: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.33: num  0 0 0 0 0 0 0 0 0.15 0 ...
##  $ A.34: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.35: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.36: num  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ A.37: num  0 0.07 0 0 0 0 0 0 0 0 ...
##  $ A.38: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.39: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.40: num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ A.41: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.42: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.43: num  0 0 0.12 0 0 0 0 0 0.3 0 ...
##  $ A.44: num  0 0 0 0 0 0 0 0 0 0.06 ...
##  $ A.45: num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ A.46: num  0 0 0.06 0 0 0 0 0 0 0 ...
##  $ A.47: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.48: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.49: num  0 0 0.01 0 0 0 0 0 0 0.04 ...
##  $ A.50: num  0 0.132 0.143 0.137 0.135 0.223 0.054 0.206 0.271 0.03 ...
##  $ A.51: num  0 0 0 0 0 0 0 0 0 0 ...
##  $ A.52: num  0.778 0.372 0.276 0.137 0.135 0 0.164 0 0.181 0.244 ...
##  $ A.53: num  0 0.18 0.184 0 0 0 0.054 0 0.203 0.081 ...
##  $ A.54: num  0 0.048 0.01 0 0 0 0 0 0.022 0 ...
##  $ A.55: num  3.76 5.11 9.82 3.54 3.54 ...
##  $ A.56: int  61 101 485 40 40 15 4 11 445 43 ...
##  $ A.57: int  278 1028 2259 191 191 54 112 49 1257 749 ...
##  $ spam: Factor w/ 2 levels "email","spam": 2 2 2 2 2 2 2 2 2 2 ...
```

Les covariables A.1 … A.54 correspondent aux word_freq_WORD et aux char_freq_CHAR.

A.55 est le capital_run_length_average et A.56, A.57 sont les covariables entières capital_run_length_longest et capital_run_length_total.

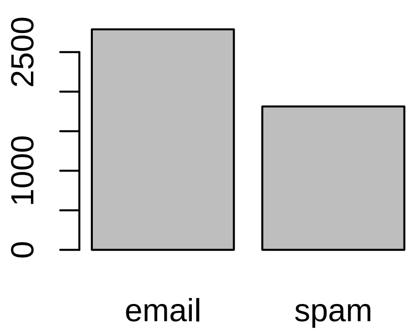
la dernière variable 'spam' est notre variable cible.

Variable cible

```
Y=emails$spam
levels(Y); table(Y); plot(Y)
```

```
## [1] "email" "spam"
```

```
## Y
## email  spam
##  2788  1813
```



```r
#proportions des spams/emails:
prop.table(table(Y))*100
```

```
## Y
##    email     spam
## 60.59552 39.40448
```

Classification par C-SVM - Noyau gaussien - C=1:

```r
Itrain=sample(1:nrow(emails),0.75*nrow(emails))
Xtrain=as.matrix(emails[Itrain,1:57])
Ytrain=as.factor(emails[Itrain,58])
Xtest=as.matrix(emails[-Itrain,1:57])
Ytest=as.factor(emails[-Itrain,58])

#Classification C-svm avec k-fold cross-validation.
classif=ksvm(x=Xtrain,y=Ytrain,type='C-svc', kernel='rbfdot',cross=4)
```

Performance sur la base d'apprentissage:

```r
prediction.train=predict(classif,Xtrain)
Confusion.train=table(pred=prediction.train, true=Ytrain)
Confusion.train
```

```
##        true
## pred    email spam
##   email  2051  102
##   spam     57 1240
```

```r
#Erreur de classification:
Err=(Confusion.train[1,2]+Confusion.train[2,1])/length(Ytrain)
#Faux positifs:(Spams classés comme emails)
FP=Confusion.train[1,2]/(Confusion.train[1,2]+Confusion.train[2,2])
#Faux négatifs (Emails classés comme spams)
FN=Confusion.train[2,1]/(Confusion.train[1,1]+Confusion.train[2,1])
```

AC (bonnes détections)=95.39%, Erreur=4.61%, FP=7.6% and FN=2.7%

Performance sur la base de test:

```r
prediction.test=predict(classif,Xtest)
Confusion.test=table(pred=prediction.test, true=Ytest)
Confusion.test
```

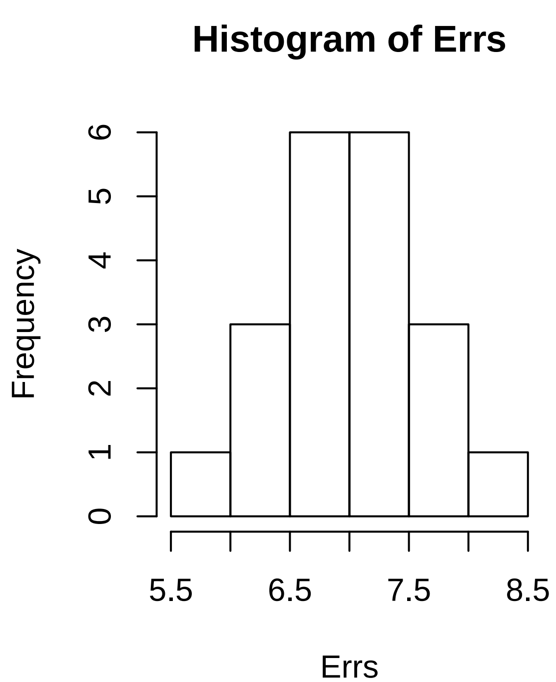
```
##        true
## pred    email spam
##   email   649   39
##   spam     31  432
```

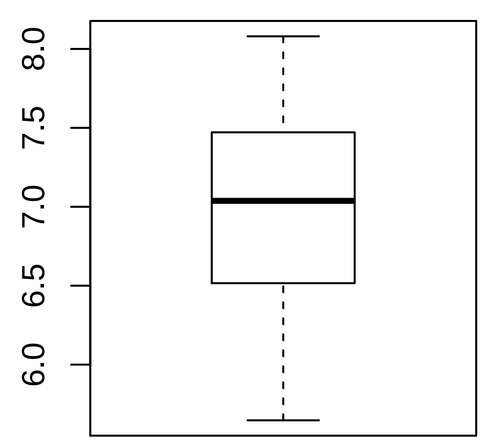
```
#Erreur de classification:
Err=(Confusion.test[1,2]+Confusion.test[2,1])/length(Ytest)
#Faux positifs:(Spams classés comme emails)
FP=Confusion.test[1,2]/(Confusion.test[1,2]+Confusion.test[2,2])
#Faux négatifs (Emails classés comme spams)
FN=Confusion.test[2,1]/(Confusion.test[1,1]+Confusion.test[2,1])
```

AC (bonnes détections)=93.92%, Erreur=6.08%, FP=8.28% and FN=4.56%

L'impact du choix aléatoire de la base d'apprentissage

```
Errs=rep(0,K)
set.seed(271)
for (k in 1:K){
  Itrain=sample(1:nrow(emails),0.75*nrow(emails))
  Xtrain=as.matrix(emails[Itrain,1:57])
  Ytrain=as.factor(emails[Itrain,58])
  Xtest=as.matrix(emails[-Itrain,1:57])
  Ytest=as.factor(emails[-Itrain,58])
  classif=ksvm(x=Xtrain,y=Ytrain,type='C-svc', kernel='rbfdot',cross=4)
  prediction.test=predict(classif,Xtest)
  Confusion.test=table(pred=prediction.test, true=Ytest)
  Errs[k]=(Confusion.test[1,2]+Confusion.test[2,1])/length(Ytest)*100
}
#Histogramme avec un nombre de classes selon la règle de Freedman-Diaconis.
par(mfrow=c(1,2))
hist(Errs,breaks="FD")
boxplot(Errs)
```





L'histogramme est identique à celui d'une distribution gaussienne et le corps de la boite à moustaches est très petit, ce qui signifie que l'erreur est homogène et que la séléction aléatoire de la base d'apprentissage a peu d'impact sur la performance du modèle SVM.

L'impact des noyaux:

On se propose de comparer les 3 noyaux suivants:

- vanilladot: Linear kernel function
- tanhdot: Hyperbolic tangent kernel function (sigmoid)
- polydot: Polynomial kernel function (d=2)

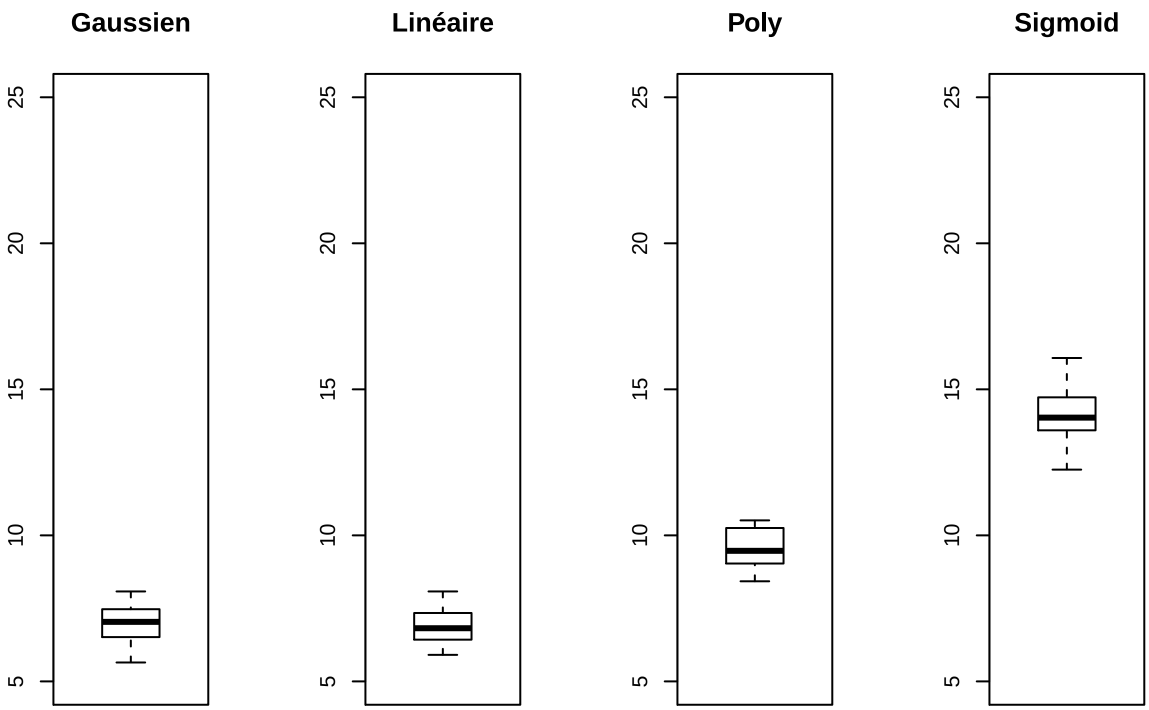
```
Errs.linear=rep(0,K)
Errs.sigmoid=rep(0,K)
Errs.poly=rep(0,K)


set.seed(1801)
for (k in 1:K){
  Itrain=sample(1:nrow(emails),0.75*nrow(emails))
  Xtrain=as.matrix(emails[Itrain,1:57])
  Ytrain=as.factor(emails[Itrain,58])
  Xtest=as.matrix(emails[-Itrain,1:57])
  Ytest=as.factor(emails[-Itrain,58])

  #Les modèles:
  classif.linear=ksvm(x=Xtrain,y=Ytrain,type='C-svc',
                      kernel='vanilladot',kpar=list(),cross=3)
  classif.sigmoid=ksvm(x=Xtrain,y=Ytrain,type='C-svc',
                       kernel='tanhdot',kpar=list(scale=0.001),cross=3)
  classif.poly=ksvm(x=Xtrain,y=Ytrain,type='C-svc',
                    kernel='polydot',kpar=list(degree=3,scale = 1, offset = 1),cross=3)
  #Les prédictions
  prediction.linear=predict(classif.linear,Xtest)
  prediction.sigmoid=predict(classif.sigmoid,Xtest)
  prediction.poly=predict(classif.poly,Xtest)
  #Les matrices de confusion et erreurs de prédiction:
  Confusion.linear=table(pred=prediction.linear, true=Ytest)
  Confusion.sigmoid=table(pred=prediction.sigmoid, true=Ytest)
  Confusion.poly=table(pred=prediction.poly, true=Ytest)

  Errs.linear[k]=(Confusion.linear[1,2]+Confusion.linear[2,1])/length(Ytest)*100
  Errs.sigmoid[k]=(Confusion.sigmoid[1,2]+Confusion.sigmoid[2,1])/length(Ytest)*100
  Errs.poly[k]=(Confusion.poly[1,2]+Confusion.poly[2,1])/length(Ytest)*100
}

  par(mfrow=c(1,4))
  boxplot(Errs,main="Gaussien",ylim=c(5,25))
  boxplot(Errs.linear,main="Linéaire",ylim=c(5,25))
  boxplot(Errs.poly,main="Poly",ylim=c(5,25))
  boxplot(Errs.sigmoid,main="Sigmoid",ylim=c(5,25))
```

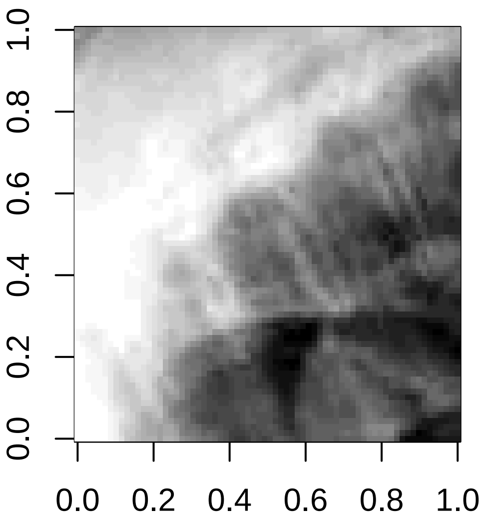


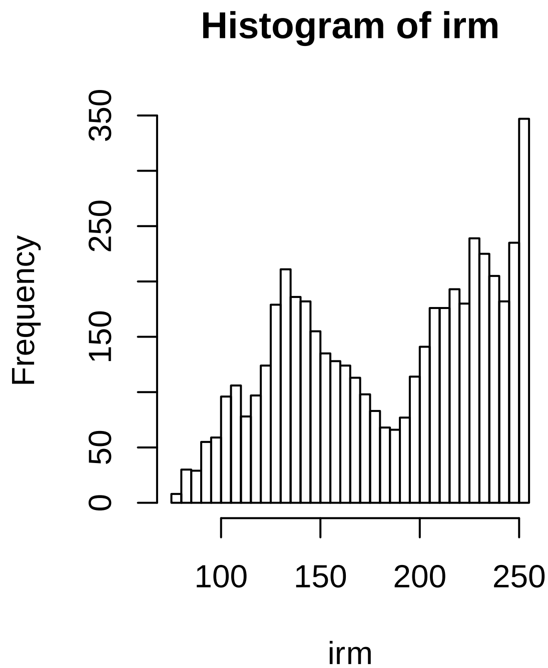
Les deux noyaux linéaire et gaussien ont de bonnes performances alors que le noyau *sigmoid* a une performance médiocre sans tuning.

Application II : Mélange de classifieurs, algorithme EM

Mélange de gaussiennes

```
irm=as.matrix(read.table("irm_thorax.txt",header=F,sep=';'))
par(mfrow=c(1,2))
image(irm,col= gray((0:32)/32))
hist=hist(irm,breaks=40)
```






```
X=c(irm)
```

A partir de l'histogramme on peut dire que la distribution des couleurs est un mélange de deux/trois gaussiennes.

Expectation-Maximization

Dans le cas d'un mélange de K gaussiennes $f(x) = \sum\limits_{k=1}^{K} p_k.f_k(x)$

Notre objectif est de maximiser $\ln \mathbb{P}(X|\theta)$ où $\theta = (p_1, ..., p_K, \mu_1, ..., \mu_K, \sigma_1, ..., \sigma_K)$.

On a donc:

$$\ln \mathbb{P}(X|\theta) = \sum_{i=1}^{n} \ln \left(\sum_{k=1}^{K} p_k.f_k(x_i) \right)$$

avec:

$$f_k(x) = \frac{1}{\sigma_k\sqrt{2\pi}} \exp\left[-\frac{(x-\mu_k)^2}{2\sigma_k^2} \right]$$

La condition d'optimalité par rapport à la variable μ_k donne:

$$\frac{\partial}{\partial \mu_k} \ln \mathbb{P}(X|\theta) = 0 = \sum_{i=1}^{n} \frac{p_k.f_k(x_i)}{f(x_i)}(x_i - \mu_k)$$

En posant

$$\eta_i^{(k)} = \frac{p_k.f_k(x_i)}{f(x_i)}$$

on a alors:

$$\mu_k = \frac{\sum\limits_{i=1}^{n} \eta_i^{(k)}.x_i}{\sum\limits_{i=1}^{n} \eta_i^{(k)}}$$

D'une façon similaire on dérive par rapport aux σ_k:

$$\sigma_k = \frac{\sum\limits_{i=1}^{n} \eta_i^{(k)}.(x_i - \mu_k)^2}{\sum\limits_{i=1}^{n} \eta_i^{(k)}}$$

On maximise ensuite selon les $(p_k)_k$ en tenant compte de la condition $\sum\limits_{k=1}^{K} p_k = 1$.

Ceci est équivalent à maximiser:

$$\ln \mathbb{P}(X|\theta) + \lambda(\sum_{k=1}^{K} p_k - 1)$$

où λ est un multiplicateur de Lagrange.

En dérivant par rapport à p_k on trouve:

$$\lambda + \sum_{i=1}^{n} \frac{f_k(x_i)}{f(x_i)} = 0$$

On multiplie les deux côtés par p_k et on somme suivant k en exploitant la contrainte du problème pour trouver $\lambda = -n$ Ainsi:

$$\sum_i \eta_i^{(k)} = p_k \sum_i \frac{f_k(x_i)}{f(x_i)} = -\lambda p_k = n p_k$$

d'où $p_k = \frac{1}{n} \sum_i \eta_i^{(k)}$

```
#Fonction auxiliaire: densité gaussienne
  gaussienne<-function(x,mu,sigma){1/sqrt(2*pi)/sigma*exp(-1/2*(x-mu)^2/sigma^2)}

EMG<-function(X,K,max_iter=500,tol=1e-10,verbose=FALSE){
  #X: les observations, K:le nombre de gaussiennes, max_iter: nombre d'itérations EM.
  #Initialisation:
  set.seed(1667)
  p=rep(1/K,K)
  sigma=rep(sd(X),K)
  mu=X[sample(1:length(X),K)]
  criterion=sqrt(sum(p^2))
  eta=matrix(rep(0,length(X)*K),nrow=K)
  iter=0
  while((iter<max_iter) & (criterion>tol)){
    iter=iter+1
    #Calcul des responsabilités (E-step):
    for (i in 1:length(X)){
      f=sum(p*gaussienne(X[i],mu,sigma))
      for(k in 1:K)
      eta[k,i]=p[k]*gaussienne(X[i],mu[k],sigma[k])/f
    }
    p.old=p
    p=rowMeans(eta)
    criterion=sqrt(sum((p-p.old)^2))
    if(verbose) {
      print(paste("iteration ",iter))
      print(p)
```

```r
   }

    #Calcul des moyennes/variances (M-step):
    mu=(eta%*%X)/rowSums(eta)
    for(k in 1:K)
      sigma[k]=sqrt(sum(eta[k,]*((X-mu[k])^2))/sum(eta[k,]))
  }
  if(iter==max_iter) warning('EM algorithm didn\'t converge')
  list(p=p,sigma=sigma,mu=mu,eta=eta,iter=iter)
}

## Deux gaussiennes:

  G2=EMG(X,2)
  G2$p
```

```
## [1] 0.4963509 0.5036491
```

```r
  G2$sigma
```

```
## [1] 18.32944 27.64159
```

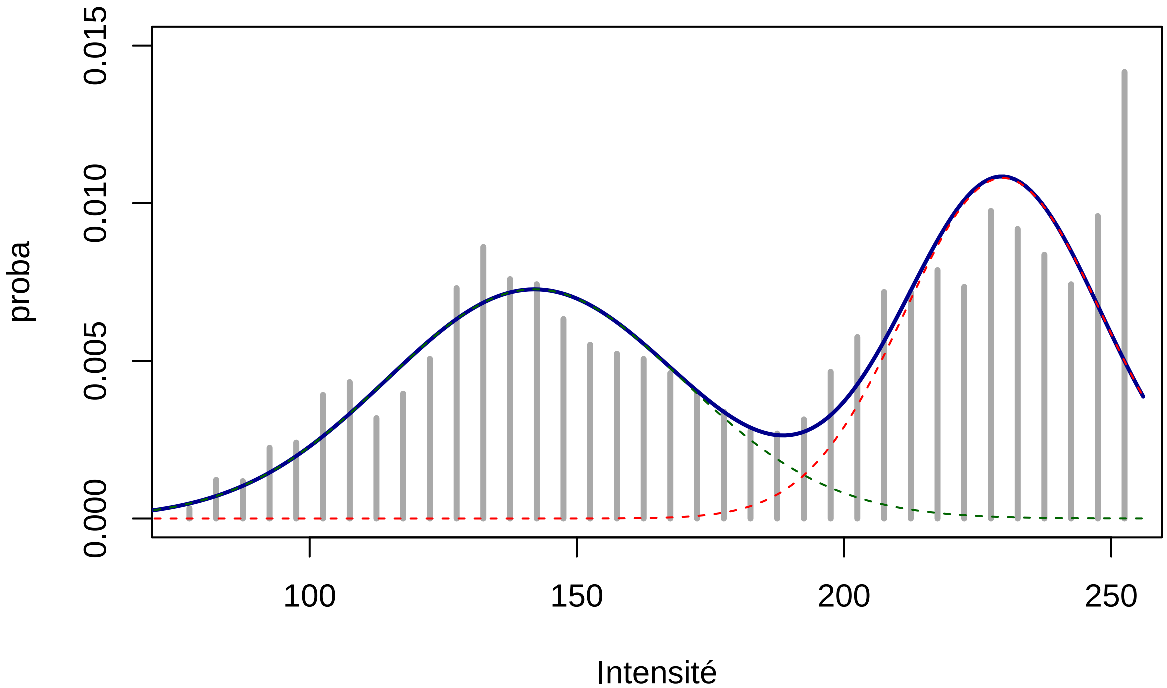
```r
  G2$mu
```

```
##           [,1]
## [1,] 228.7147
## [2,] 141.0565
```

```r
  G2$iter
```

```
## [1] 111
```

```r
  Ypred=G2$p[1]*gaussienne(0:255,G2$mu[1],G2$sigma[1])+
    G2$p[2]*gaussienne(0:255,G2$mu[2],G2$sigma[2])
  plot(hist$mids,hist$density,ylim=c(0,.015),xlab="Intensité",
       ylab="proba",lwd=3,type='h',col='darkgray')
  lines(Ypred,col="blue4",lwd=2)
  lines(G2$p[1]*gaussienne(0:255,G2$mu[1],G2$sigma[1]),col='red',lty=2)
  lines(G2$p[2]*gaussienne(0:255,G2$mu[2],G2$sigma[2]),col='darkgreen',lty=2)
```




```
## Trois gaussiennes:
  G3=EMG(X,3)
  G3$p
```

```
## [1] 0.09061959 0.40629932 0.50308109
```

```
  G3$sigma
```

```
## [1]  3.129073 16.128414 27.505720
```

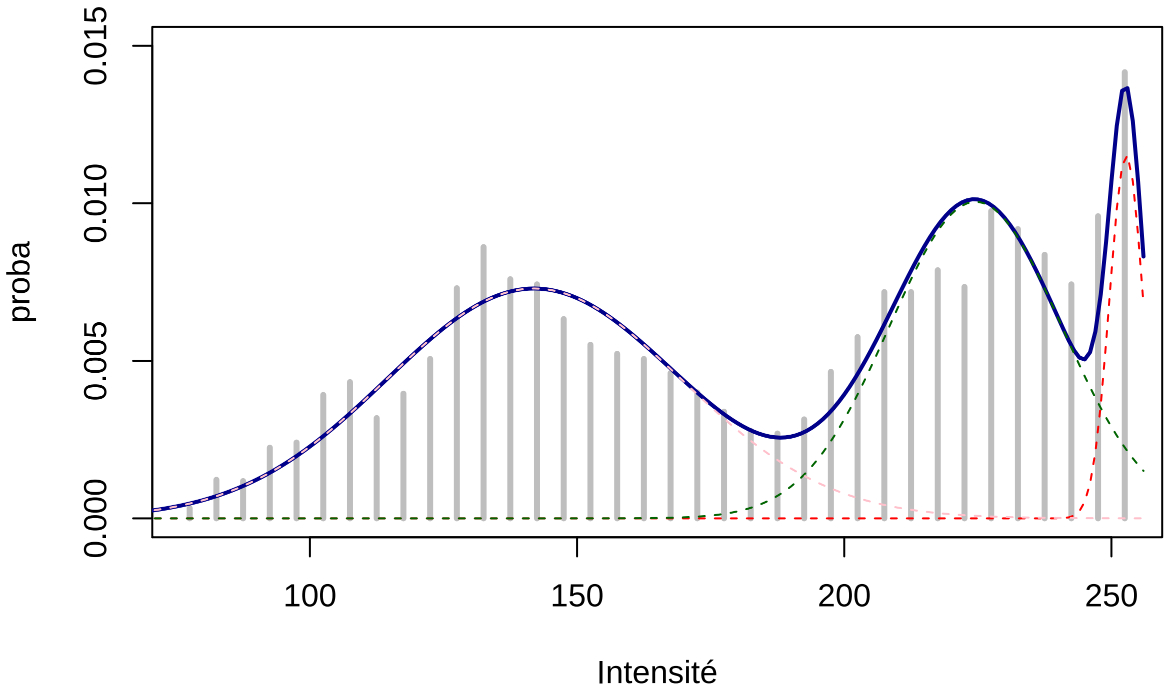
```
  G3$mu
```

```
##          [,1]
## [1,] 251.7788
## [2,] 223.5863
## [3,] 140.9448
```

```
  G3$iter
```

```
## [1] 363
```

```
  Ypred=G3$p[1]*gaussienne(0:255,G3$mu[1],G3$sigma[1])+
    G3$p[2]*gaussienne(0:255,G3$mu[2],G3$sigma[2])+
    G3$p[3]*gaussienne(0:255,G3$mu[3],G3$sigma[3])
  plot(hist$mids,hist$density,ylim=c(0,.015),xlab="Intensité",
       ylab="proba",lwd=3,type='h',col='gray')
  lines(Ypred,col="blue4",lwd=2)
  lines(G3$p[1]*gaussienne(0:255,G3$mu[1],G3$sigma[1]),col='red',lty=2)
  lines(G3$p[2]*gaussienne(0:255,G3$mu[2],G3$sigma[2]),col='darkgreen',lty=2)
  lines(G3$p[3]*gaussienne(0:255,G3$mu[3],G3$sigma[3]),col='pink',lty=2)
```




```
## Cinq gaussiennes:
  G5=EMG(X,5)
```

```
## Warning in EMG(X, 5): EM algorithm didn't converge
```

```
  G5$p
```

```
## [1] 0.03301901 0.17200957 0.38472335 0.27964528 0.13060280
```

```
  G5$sigma
```

```
## [1] 8.734104e-04 7.717982e+00 2.336935e+01 1.364996e+01 2.250391e+01
```

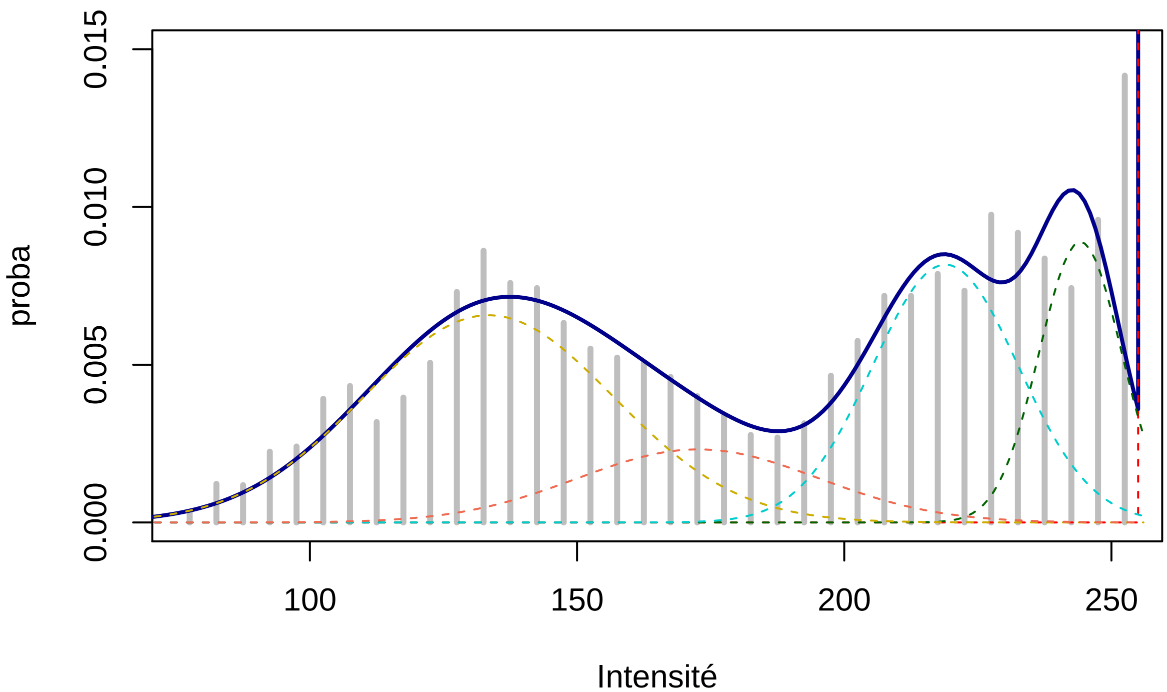
```
  G5$mu
```

```
##           [,1]
## [1,] 255.0000
## [2,] 243.1962
## [3,] 132.4344
## [4,] 217.9373
## [5,] 171.6507
```

```
  G5$iter
```

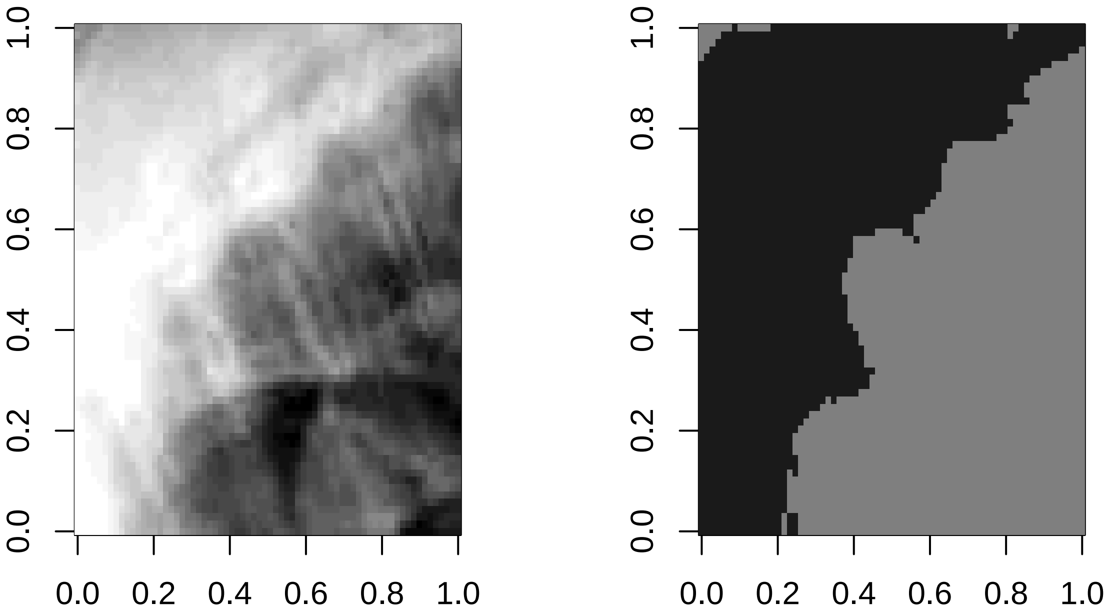
```
## [1] 500
```

```r
Ypred=G5$p[1]*gaussienne(0:255,G5$mu[1],G5$sigma[1])+
  G5$p[2]*gaussienne(0:255,G5$mu[2],G5$sigma[2])+
  G5$p[3]*gaussienne(0:255,G5$mu[3],G5$sigma[3])+
  G5$p[4]*gaussienne(0:255,G5$mu[4],G5$sigma[4])+
  G5$p[5]*gaussienne(0:255,G5$mu[5],G5$sigma[5])
plot(hist$mids,hist$density,ylim=c(0,.015),xlab="Intensité",
     ylab="proba",lwd=3,type='h',col='gray')
lines(Ypred,col="blue4",lwd=2)
lines(G5$p[1]*gaussienne(0:255,G5$mu[1],G5$sigma[1]),col='red',lty=2)
lines(G5$p[2]*gaussienne(0:255,G5$mu[2],G5$sigma[2]),col='darkgreen',lty=2)
lines(G5$p[3]*gaussienne(0:255,G5$mu[3],G5$sigma[3]),col='gold3',lty=2)
lines(G5$p[4]*gaussienne(0:255,G5$mu[4],G5$sigma[4]),col='cyan3',lty=2)
lines(G5$p[5]*gaussienne(0:255,G5$mu[5],G5$sigma[5]),col='coral2',lty=2)
```

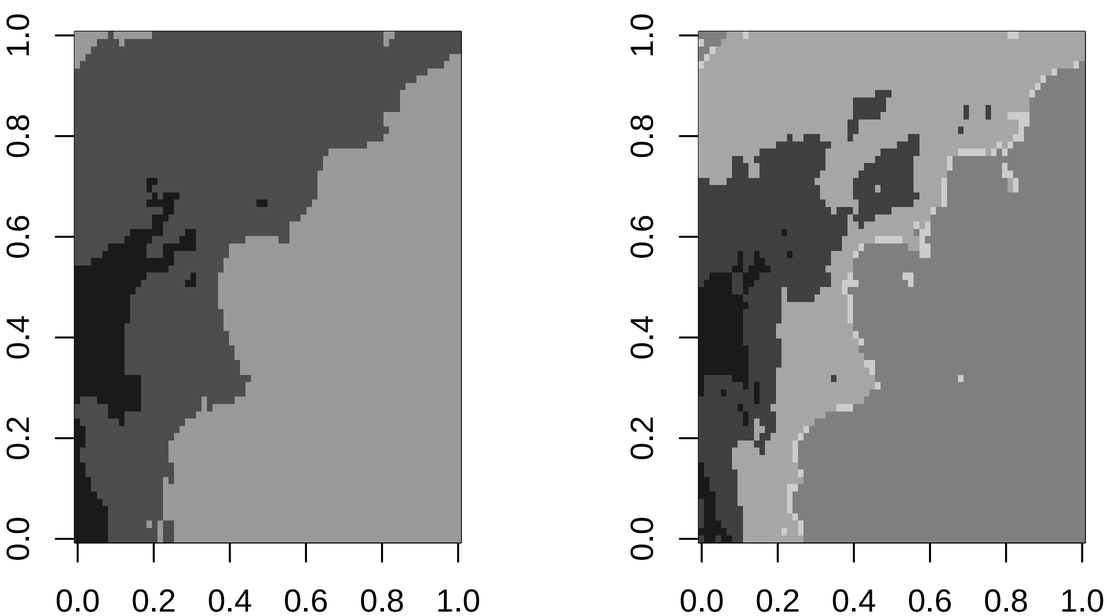


Segmentation de l'image:

```r
par(mfrow=c(1,2))
image(irm,col= gray((0:32)/32))
Classif2=apply(G2$eta*matrix(rep(G2$p,dim(G2$eta)[2]),nr=2), 2, which.max)
image(matrix(Classif2,nr=dim(irm)[1]),col=c('gray10','gray50'))
```



```
Classif3=apply(G3$eta*matrix(rep(G3$p,dim(G3$eta)[2]),nr=3), 2, which.max)
image(matrix(Classif3,nr=dim(irm)[1]),col=c('gray10','gray30','gray60'))
Classif5=apply(G5$eta*matrix(rep(G5$p,dim(G5$eta)[2]),nr=5), 2, which.max)
image(matrix(Classif5,nr=dim(irm)[1]),col=c('gray10','gray25','gray50','gray65','gray80'))
```



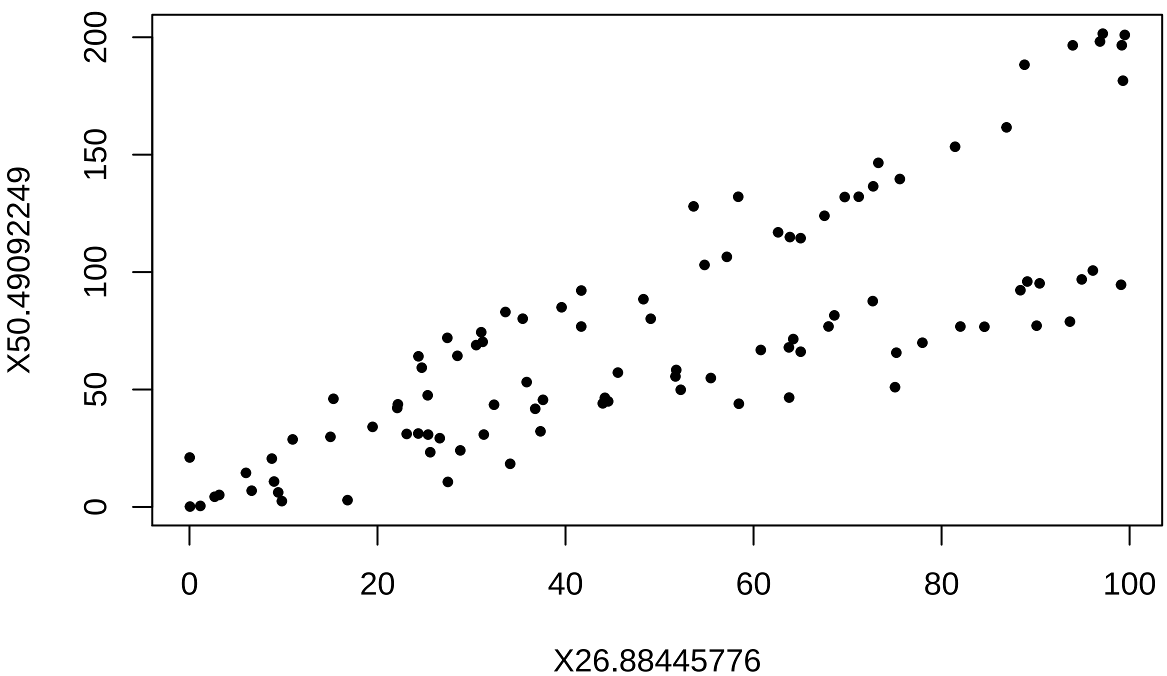
La segmentation semble de plus en plus pertinente en augmentant le nombre de gaussiennes et donc le nombre de classes. Mais à partir de 5 gaussiennes, l'image devient très fragmentée.

Mixture de régressions par l'algorithme EM:

```
reg.data=read.table("regression_double.txt",header=T,sep=';');
str(reg.data)
```

```
## 'data.frame':    99 obs. of  2 variables:
##  $ X26.88445776: num  64.22 58.37 75.05 9.83 65.02 ...
##  $ X50.49092249: num  71.48 132.08 51 2.48 66.09 ...
```

```
plot(reg.data,pch=19,cex=0.6)
```



On remarque qu'il ne peut exister une fonction qui mappe l'espace de la covariable dans celui de la variable cible, la même covariable peut avoir deux réponses différentes (d'où les deux droites). ceci est probablement dû au fait qu'on n'a pris compte d'autres variables explicatives. Une regression linéaire simple n'est pas suffisante dans ce cas.

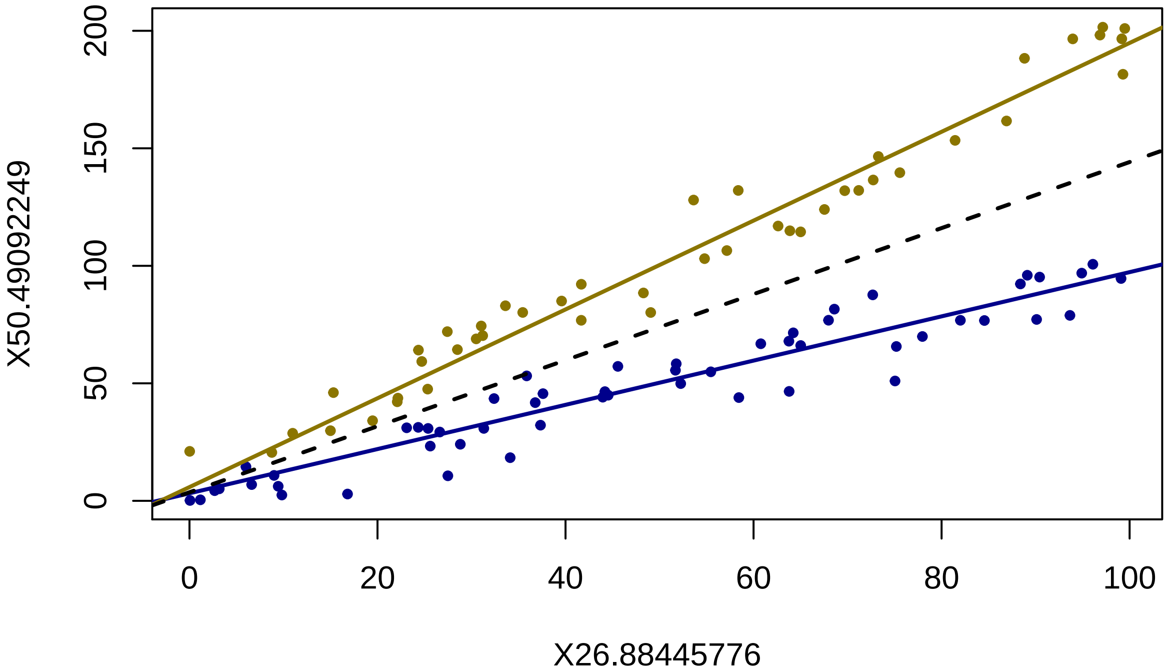
Deux régressions:

```
set.seed(1697)
reg.model=regmixEM(y=reg.data[,2],x=reg.data[,1],k=2)
```

```
## number of iterations= 14
```

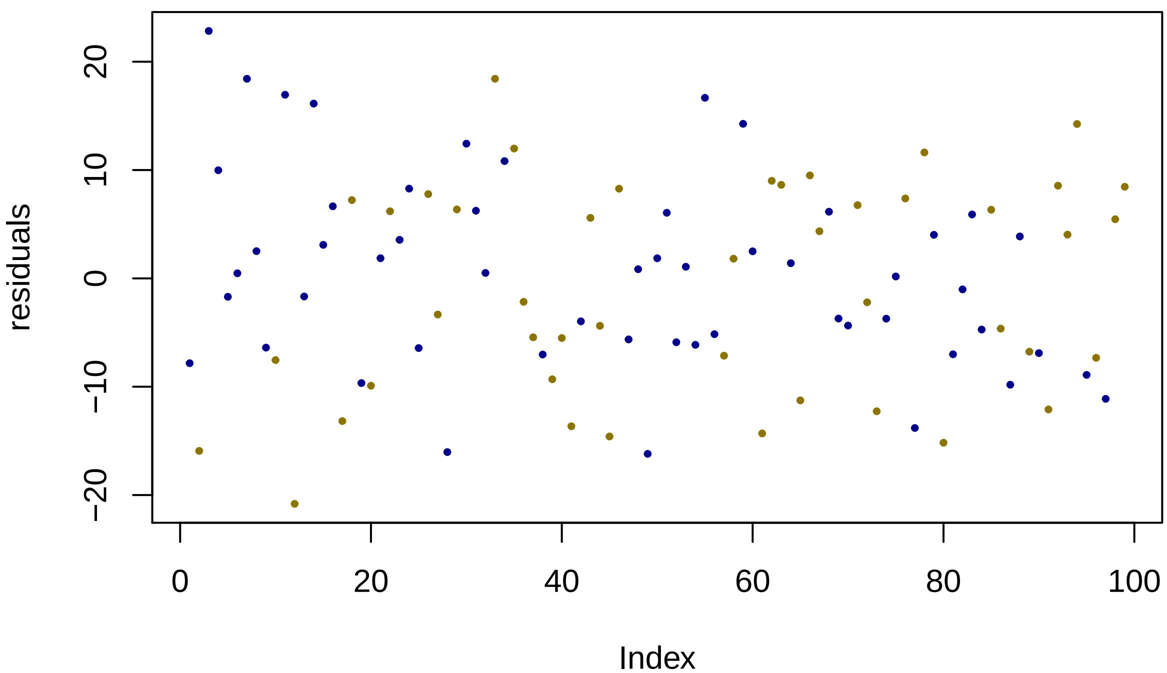
```
classify=apply(reg.model$posterior,1,which.max)
plot(reg.data,pch=19,cex=0.6,col=c("blue4","gold4")[classify])
abline(reg.model$beta[,1],lwd=2,col="blue4")
abline(reg.model$beta[,2],lwd=2,col="gold4")

#Régression linéaire simple:
lm1=lm(reg.data[,2]~reg.data[,1])
abline(lm1,lty=2,lwd=2)
```

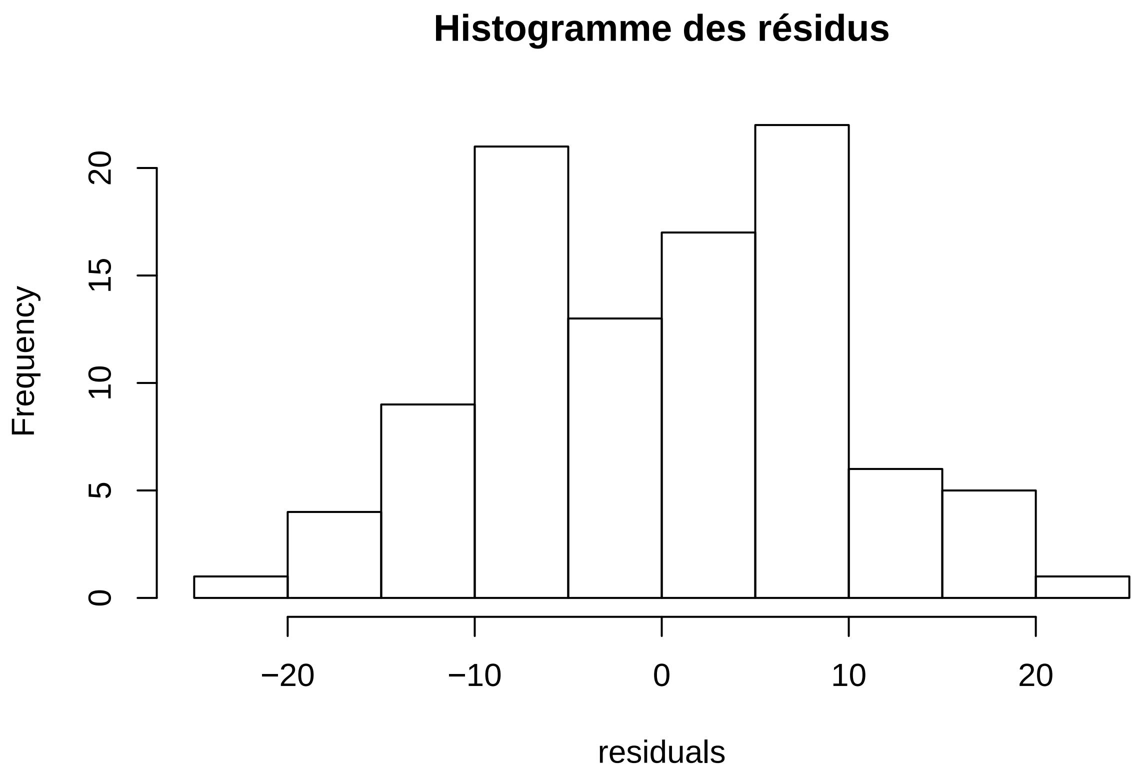



```
#Calcul des résidus:

residuals=reg.model$beta[1,classify]+reg.model$beta[2,classify]*reg.data[,1]-reg.data[,2]
plot(residuals,col=c("blue4","gold4")[classify],pch=19,cex=.4)
```




```
hist(residuals,breaks=15,main="Histogramme des résidus")
```




```
RSS=t(residuals)%*%residuals
RSS.1=t(lm1$residuals)%*%lm1$residuals
```

La somme des carrés des résidus (deux régressions linéaires) =8558.73

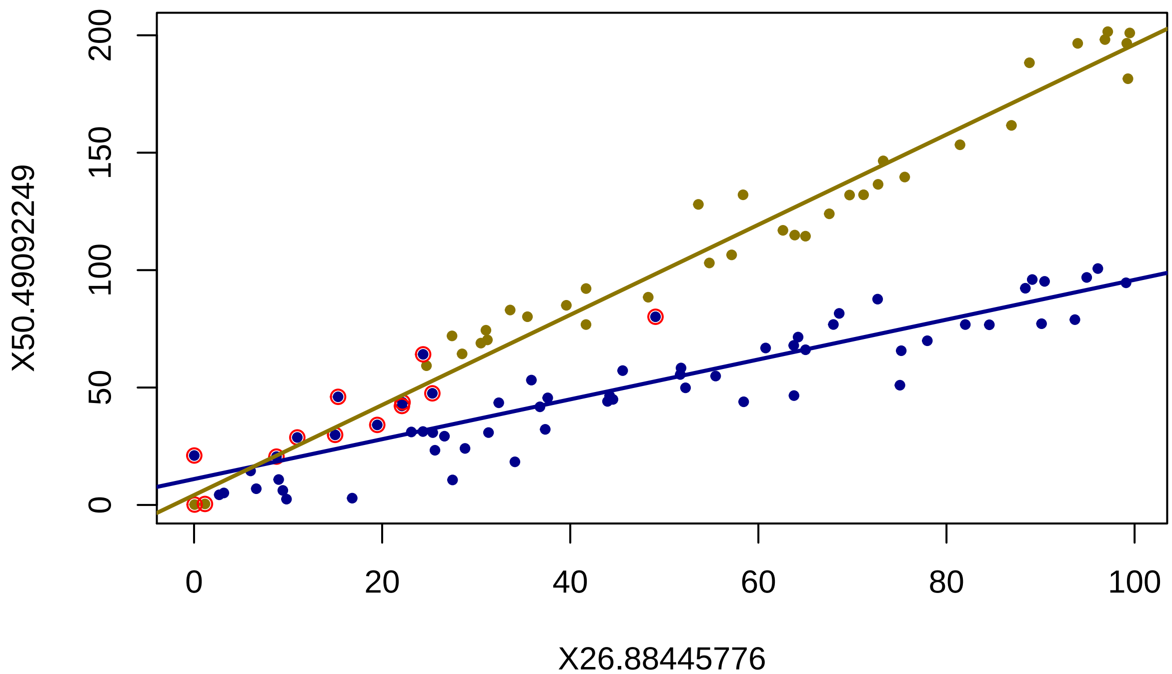
La somme des carrés des résidus (régression linéaire simple) =89122.1

Nombre d'itérations limité

```
#Une seule itération
set.seed(1697)
reg.model.1=regmixEM(y=reg.data[,2],x=reg.data[,1],k=2,maxit=1)
```

```
## WARNING! NOT CONVERGENT!
## number of iterations= 1
```

```
classify.1=apply(reg.model.1$posterior,1,which.max)
plot(reg.data,pch=19,cex=0.6,col=c("blue4","gold4")[classify.1])
points(reg.data[classify.1!=classify,],col='red')
abline(reg.model.1$beta[,1],lwd=2,col="blue4")
abline(reg.model.1$beta[,2],lwd=2,col="gold4")
```



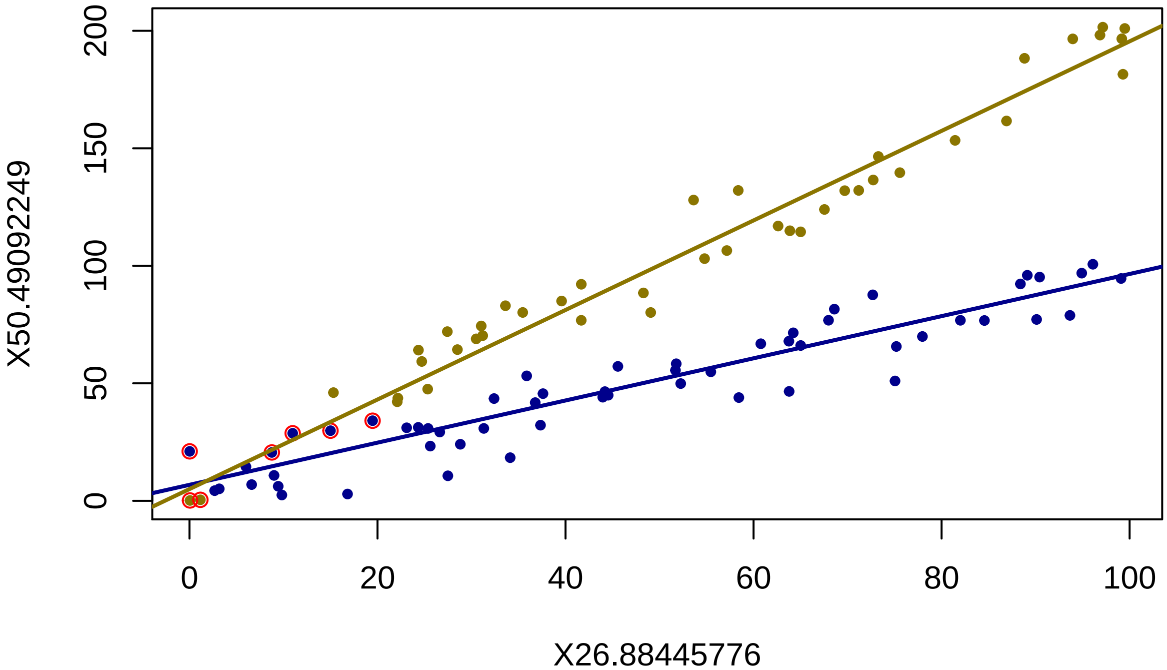

```
Misclass.1=sum(classify.1!=classify)
```

Avec une seule itération, 13 mauvaises prédictions.

```
#Deux itérations:
set.seed(1697)
reg.model.2=regmixEM(y=reg.data[,2],x=reg.data[,1],k=2,maxit=2)
```

```
## WARNING! NOT CONVERGENT!
## number of iterations= 2
```

```
classify.2=apply(reg.model.2$posterior,1,which.max)
plot(reg.data,pch=19,cex=0.6,col=c("blue4","gold4")[classify.2])
points(reg.data[classify.2!=classify,],col='red')
abline(reg.model.2$beta[,1],lwd=2,col="blue4")
abline(reg.model.2$beta[,2],lwd=2,col="gold4")
```



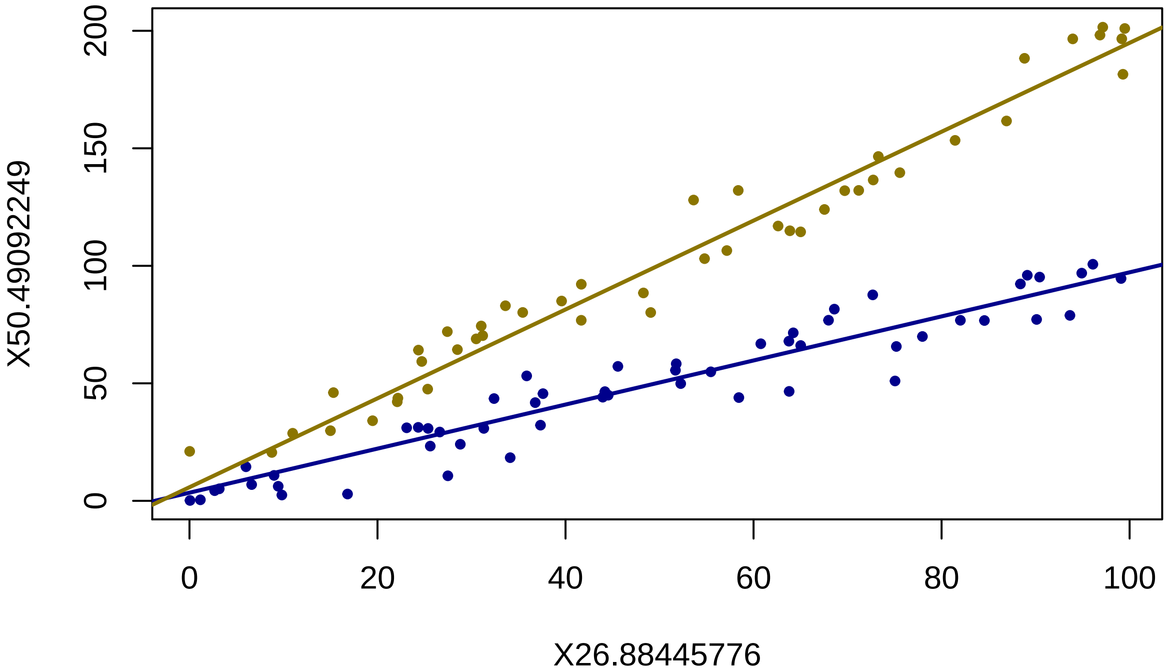

```
Misclass.2=sum(classify.2!=classify)
```

Avec deux itérations, 7 mauvaises prédictions.

```
#Cinq itérations:
set.seed(1697)
reg.model.5=regmixEM(y=reg.data[,2],x=reg.data[,1],k=2,maxit=5)
```

```
## WARNING! NOT CONVERGENT!
## number of iterations= 5
```

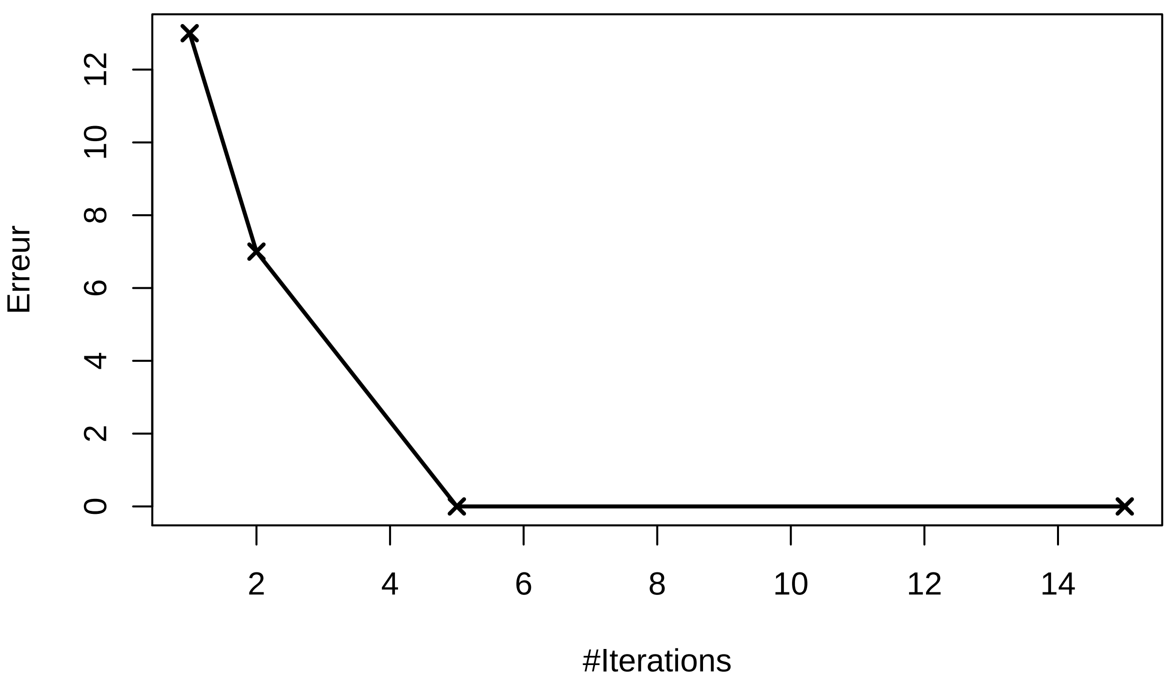
```
classify.5=apply(reg.model.5$posterior,1,which.max)
plot(reg.data,pch=19,cex=0.6,col=c("blue4","gold4")[classify.5])
points(reg.data[classify.5!=classify,],col='red')
abline(reg.model.5$beta[,1],lwd=2,col="blue4")
abline(reg.model.5$beta[,2],lwd=2,col="gold4")
```



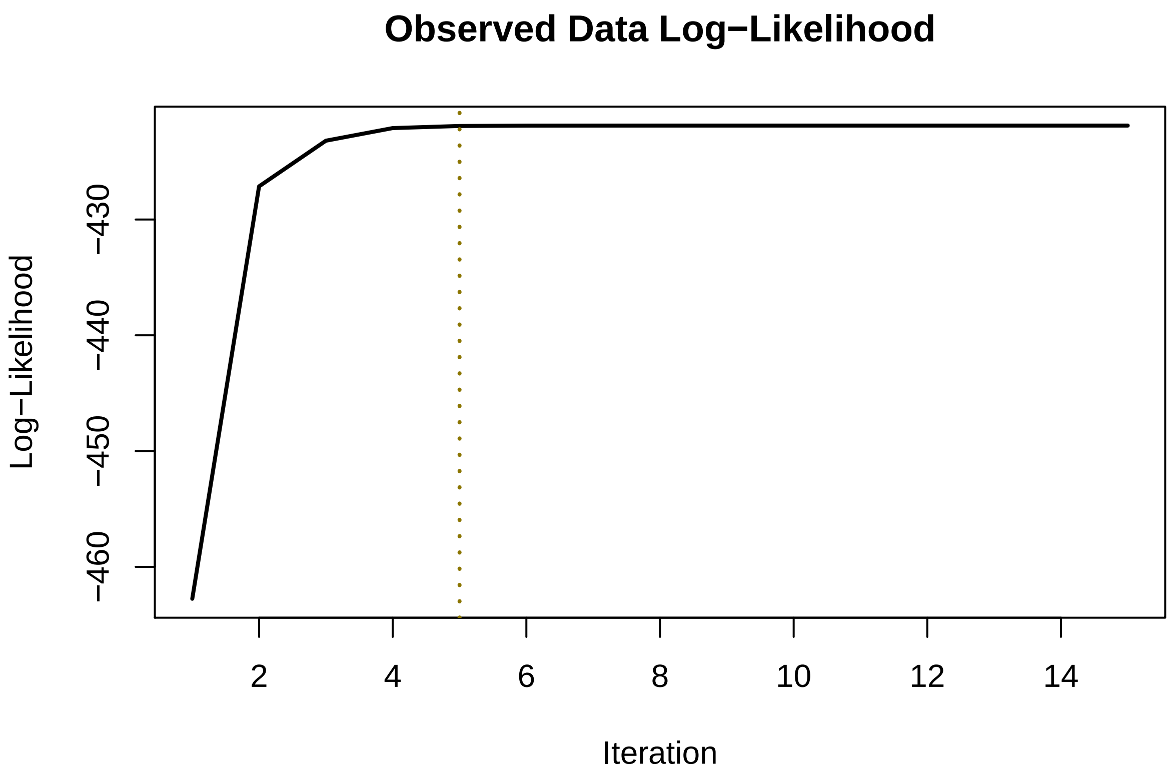

```
Misclass.5=sum(classify.5!=classify)
```

Avec cinq itérations, 0 mauvaises prédictions.

```
plot(c(1,2,5,length(reg.model$all.loglik)),
     c(Misclass.1,Misclass.2,Misclass.5,0),
     type="o",pch=4,lwd=2,xlab="#Iterations",ylab="Erreur")
```



```r
plot(reg.model)
abline(v=5,col='gold4',lty=3,lwd=2)
```



On constate qu'on arrive à bien classer les vecteurs à partir de la 5ème itération même si l'algorithme n'a pas encore convergé vu que la log-vraisemblance du modèle est "suffisante".