# HMMA303 Discrimination et scoring
# TP2 Analyse discriminante

*Jean-Michel Marin*

*Version du 21/09/2018*

Le but de ce TP est d'illustrer la règle de Bayes et le calcul d'une règle de classification à partir de l'analyse discriminante linéaire et l'analyse discriminante quadratique.

## 1 Simulation de données

Dans cette partie, on va simuler des données dans $\mathbb{R}^2$ issues d'un mélange de 2 distributions gaussiennes. On suppose que les probabilités a priori des classes sont $\pi_0$, $\pi_1$. Pour générer ces données, il suffit de simuler une variable de classe $Y$ qui prend les valeurs 0 ou 1 avec probabilités $\pi_0$, $\pi_1$ respectivement. Puis, conditionnellement à la valeur de $Y = k$, on simule un vecteur qui suit une loi normale en dimension 2 (i.e. si $Y = k$, on simule $X|Y = k \sim \mathcal{N}_2(\mu_k, \Sigma_k)$ pour $k = 0, 1$).
Utiliser le code ci-dessous pour générer $n = 200$ observations issues d'un mélange de vecteurs gaussiens.

- quelles sont les probabilités a priori, les moyennes et les matrices de variance des classes ?
- que permet de faire la fonction `sample` ?
- que permet de faire la fonction `rmvnorm` ?

```
n <- 200 ; pi0 <- 0.3 ; pi1 <- 0.7 ; mu0 <- c(3,9) ; mu1 <- c(5,5)
Sigma0 <- matrix(c(3,3*0.5,3*0.5,3),2,2)
Sigma1 <- matrix(c(5,-5*0.9,-5*0.9,5),2,2)

y <-sample(c(0,1),size=n,prob=c(pi0,pi1),replace=TRUE)
y <- sort(y)

n0 <- sum(y==0)
n1 <- sum(y==1)

library(mvtnorm)
x <- rbind(rmvnorm(n0,mu0,Sigma0),rmvnorm(n1,mu1,Sigma1))

x1 <- seq(min(x[,1]),max(x[,1]),length=50)
x2 <- seq(min(x[,2]),max(x[,2]),length=50)
z <- matrix(0,50,50)
for (i in 1:50) for (j in 1:50)
z[i,j] <- pi0*dmvnorm(c(x1[i],x2[j]),mu0,Sigma0)+
pi1*dmvnorm(c(x1[i],x2[j]),mu1,Sigma1)

contour(x1,x2,z,nlevels=30)
```

```
colo <- y
colo[y==0] <- heat.colors(12)[12]
colo[y==1] <- heat.colors(12)[1]
points(x,pch=21,cex=1,col="black",bg=colo)

persp(x1,x2,z)
library(rgl)
persp3d(x1,x2,z)
```

# 2 Classifieur de Bayes

Afin de représenter les frontières de décision données par le classifieur de Bayes, on discrètise le plan sous forme d'une grille de points régulièrement espacés dans le plan contenant les observations. Pour pouvoir classer chaque point $x$ de la grille, on calcule

$$\arg \max_{k \in \{0,1\}} \pi_k f_k(x)$$

où $f_k(x)$ est la densité de la loi $\mathcal{N}_2(m_k, \Sigma_k)$ pour $k = 0, 1$.
Afin de limiter le temps de calcul, on choisit une grille carrée de taille 50 sur 50 points (i.e. 2500 points) dont les extrémités sont données par les minima et les maxima des observations (en abscisse et ordonnée).
Utiliser le code ci-dessous pour colorier les zones de classification données par le classifieur de Bayes pour le mélange de vecteurs gaussiens de la question 1 :

— que permet de faire la fonction `dmvnorm` ?

```
pi0x=function(x)
{
c0 <- pi0*dmvnorm(x,mean=mu0,sigma=Sigma0)
c1 <- pi1*dmvnorm(x,mean=mu1,sigma=Sigma1)
c0/(c0+c1)
}

aff0 <- matrix(0,50,50)
for (i in 1:50) for (j in 1:50)
aff0[i,j] <- pi0x(c(x1[i],x2[j]))
image(x1,x2,aff0)
points(x,pch=21,cex=1,col="black",bg=colo)

contour(x1,x2,aff0,levels=0.5)
points(x,pch=21,cex=1,col="black",bg=colo)
```

# 3 Analyse discriminante quadratique

La fonction `qda` de la librairie `MASS` permet de réaliser l'analyse discriminante quadratique d'un jeu de données à partir de l'estimation empirique des probabilités a priori, des moyennes et des matrices de variance des classes.

```
library(MASS)
learning.set <- data.frame(x1=x[,1],x2=x[,2],y=as.factor(y))
model.qda <- qda(y~x1+x2,data=learning.set)
```

Les instructions suivantes permettent de calculer les affectations et les probabilités a posteriori des classes pour une ensemble de points d'un grille :

```
toaffect.set <- expand.grid(x1=x1,x2=x2)
affectations.qda <- predict(model.qda,toaffect.set)$class
postprob.qda <- predict(model.qda,toaffect.set)$posterior
contour(x1,x2,aff0,levels=0.5)
contour(x1,x2,matrix(postprob.qda[,1],50,50),levels=0.5,col="red",add=TRUE)
```

# 4 Analyse discriminante linéaire

La fonction `lda` de la librairie `MASS` permet de réaliser l'analyse discriminante linéaire d'un jeu de données à partir de l'estimation empirique des probabilités a priori, des moyennes et de la matrices de variance des classes.

```
model.lda <- lda(y~x1+x2,data=learning.set)
affectations.lda <- predict(model.lda,toaffect.set)$class
postprob.lda <- predict(model.lda,toaffect.set)$posterior
contour(x1,x2,aff0,levels=0.5)
contour(x1,x2,matrix(postprob.qda[,1],50,50),levels=0.5,col="red",add=TRUE)
contour(x1,x2,matrix(postprob.lda[,1],50,50),levels=0.5,col="green",add=TRUE)
```

# 5 Analyse discriminante non paramétrique

La fonction `NaiveBayes` de la librairie `klaR` permet de réaliser l'analyse discriminante non paramétrique d'un jeu de données.

```
library(klaR)
model.nb <- NaiveBayes(y~x1+x2,data=learning.set,usekernel=TRUE)
postprob.nb <- predict(model.nb,toaffect.set)$posterior
contour(x1,x2,aff0,levels=0.5)
contour(x1,x2,matrix(postprob.qda[,1],50,50),levels=0.5,col="red",add=TRUE)
contour(x1,x2,matrix(postprob.lda[,1],50,50),levels=0.5,col="green",add=TRUE)
contour(x1,x2,matrix(postprob.nb[,1],50,50),levels=0.5,col="orange",add=TRUE)
```

# 6 Validation croisée

Comparer par validation croisée à 10 ensembles les méthodes d'analyse discriminante linéaire, quadratique et non paramétrique pour le jeux de données précédemment généré.

```
library(caret)
model.lda <- train(y~x1+x2,data=learning.set,method="lda",metric="Accuracy",
trControl=trainControl(method="repeatedcv",number=10,repeats=50))
model.qda <- train(y~x1+x2,data=learning.set,method="qda",metric="Accuracy",
trControl=trainControl(method="repeatedcv",number=10,repeats=50))
model.nb <- train(y~x1+x2,data=learning.set,method="nb",metric="Accuracy",
trControl=trainControl(method="repeatedcv",number=10,repeats=50))
```

# 7 La validation croisée fonctionne-t-elle correctement ?

```
ntest <- 10000
ytest <- sample(c(0,1),size=ntest,prob=c(pi0,pi1),replace=TRUE)
ytest <- sort(ytest)
n0test <- sum(ytest==0)
n1test <- sum(ytest==1)
xtest <- rbind(rmvnorm(n0test,mu0,Sigma0),rmvnorm(n1test,mu1,Sigma1))
test.set <- data.frame(x1=xtest[,1],x2=xtest[,2],y=as.factor(ytest))

1-mean(predict(model.lda,test.set,type="raw")!=test.set$y)
1-mean(predict(model.qda,test.set,type="raw")!=test.set$y)
1-mean(predict(model.nb,test.set,type="raw")!=test.set$y)
```