Data Mining 2: Modélisation statistique et apprentissage **G. Cohen**

Travaux pratiques 17 mars 2018

ETheme 2 Discriminants

1 Données du syndrome de Cushing

Trouble de l'hypertension associée à la sécrétion excessive de cortisol par la glande surrénale. Les observations sont les taux d'excrétion urinaire de deux métabolites de stéroïdes. Le data frame Cushings (dans library MASS) a 27 lignes et 3 colonnes:

- Tetrahydrocortisone: taux d'excrétion urinaire (mg/24hr).
- Pregnanetriol: taux d'excrétion urinaire (mg/24hr).
- Type: type du syndrome sous-jacent
  - (adenome)
  - (hyperplasie bilatérale)
  - (carcinome)
  - inconnu

```
rm(list = ls(all = TRUE))
library(rpart)
library(MASS)
```

Il y a trois types de symdromes de Cushings syndrome qui sont distingués, codés a, b, et c. On retire les observations du jeux de données avec le type u (unknown).

```
#Cushings data
data(Cushings)
cush <- Cushings[Cushings$Type!="u",]
cush[,3] <- factor(cush[,3],levels=c("a","b","c"))
cush.type<-cush[,3]
```

La fonction factor est utilisée pour créer un vecteur cush.type qui indique le type de syndrome de chaque observation dans le dataset reconstruit cush. La transformation log des variables continues assure que leur distribution soit plus symétrique.

```
boxplot(cush[,1:2])
cush[,1] <- log(cush[,1])
cush[,2] <- log(cush[,2])
boxplot(cush[,1:2])
```

On peut calculer les directions du discriminant linéaire. **lda()** trouvent ces combinaisons linéaires des données, et **predict()** transforme les données dans le système de coordonnées LDA. Les coordonnées LDA montrent une meilleure séparation des classes que dans les coordonnées d'origine.

```
#plot the data on the lda's
cush.lda<-lda(cush[,1:2], cush.type)
cush.pred<-predict(cush.lda, dimen=2)
device()
plot(cush.pred$x, type="n", xlab="premier discriminant linéaire",
ylab="second discriminant linéaire", main="LDA")
text(cush.pred$x, labels=as.character(cush.type), cex=0.8)
#plot the data
device()
plot(cush[,1:2],pch=as.character(cush.type))
```

Il est intéressant de tracer les frontières de discrimination LDA sur les données. C'est facile, puisque les données sont en 2D, on peut donc prendre un treillis de points dans les coordonnées originales, et demander pour une prédiction des classes des points du treillis. On réalise un tracé du contour des frontières entre les classes.

```
#tracer les forntières de décision
#prendre un treillis de points
#treillis de 100 par 100 - augmenter à 200 x 200 pour des lignes plus fines
L<-100
x<-seq(1,4,3/L)
y<-seq(-3,2,5/L)
z<-as.matrix(expand.grid(Tetrahydrocortisone=x,Pregnanetriol=y))
m<-length(x)
n<-length(y)
cush.ldp<-predict(cush.lda,z)$class
#les classes sont a,b,c =1,2,3 donc on met les contours à 1.5 et 2.5
contour(x,y,matrix(as.numeric(cush.ldp),m,n),levels=c(1.5,2.5),
add=T,d=F,lty=1,col=2)
```

On peut réaliser une analyse discriminante quadratique, et ainsi permettre des matrices de covariances inégales entre classes. En regardant les données , il apparait que c'est le cas. On utilise la même stratégie que précédemment pour tracer les frontières.

```
cush.qda <- qda(cush[,1:2], cush.type)
cush.qdp<-predict(cush.qda,z)$class
contour(x,y,matrix(as.numeric(cush.qdp),m,n),levels=c(1.5,2.5),
add=T,d=F,lty=1,col=3)
```

2 Implémentation de procédures LDA et QDA

On vient de voir que la bibliothèque MASS library(MASS) comporte les fonctions lda et qda respectivement pour l'analyse discriminante linéaire et quadratique. Toutefois pour des raisons d'illustration des méthodes et pour réaliser des discriminations faciles on a écrit ici des fonctions "faites maison".

Fonction *lda.fit* - calcul pour lda ou qda discrimination.

```
lda.fit <- function(y,X,prior,qda=FALSE,tol=1e-6) {

  N <- dim(X)[1]          # taille de l'échantillon
  n <- c(table(y))        # nombre de cas par classe
  p <- dim(X)[2]          # dimension (nombre de variables)
  K <- length(unique(y))  # nombre de classes

  # à priori
  if(missing(prior)) prior <- n/N
  logpi <- log(prior)

  # faire une liste des données df'apprentissage pour chaque classe
  G <- split(X,y)
  for (k in 1:K) G[[k]] <- matrix(G[[k]],ncol=p)

  # statistiques pour chaque classe (moyenne et variance)
  G.mean <- as.list(as.data.frame(sapply(G,apply,2,mean)))
  G.var <- as.list(as.data.frame(sapply(G,var)))
  for (k in 1:K) G.var[[k]] <- matrix(G.var[[k]],ncol=p)*(n[k]/(n[k]-1))

  # calcul de la matrice de variance/covariance combinée var/cov pour lda seulement
```

```
  if(!qda) {
    T.var <- G.var[[1]]*(n[1]-1)
    for (k in 2:K) T.var <- T.var + G.var[[k]]*(n[k]-1)
    T.var <- T.var/(N-K)
    for (k in 1:K) G.var[[k]] <- T.var
  }

  # fournit les ingrédients pour la fonction de discrimination;
  # Ici, on rappelle que pour le SVD S=UDU^T, on peut
  # calculer log(det(S)) = sum(log(d_{ii)), et
  # (x-mu)^TS^{-1}(x-mu) = [D^{-1/2}U^T(x-mu)]^T[D^{-1/2}U^T(x-mu)]

  logdet <- rep(NA,length=K)
  G.U <- G.var
  G.D2i <- G.var
  for (k in 1:K) {
    sv <- svd(G.var[[k]],nu=p,nv=0)
    rnk <- sum(sv$d>tol)
    if (rnk<p) {
      cat("Collinéarité dans le groupe",k,"...\n")
      stop("lda/qda ne peut pas continuer en raison de la collinéarité.")
    }
    logdet[k] <- sum(log(sv$d))
    G.U[[k]] <- sv$u
    G.D2i[[k]] <- diag(1/sqrt(sv$d))
  }

  return(list(logpi=logpi,logdet=logdet,mu=G.mean,D2i=G.D2i,U=G.U))

}
```

et la fonction *lda.pred* - qui prédit la classe d'appartenance à partir d'un objet *lda.fit*.

```
lda.pred <- function(X,ldaf) {

  offset <- -ldaf$logdet/2 + ldaf$logpi
  mu <- ldaf$mu
  D2i <- ldaf$D2i
  U <- ldaf$U
```

```
  p <- length(mu[[1]])
  X <- matrix(c(X),ncol=p)
  N <- dim(X)[1]
  K <- length(offset)

  delta <- matrix(NA,nrow=N,ncol=K)

  for (k in 1:K) {
    Xs <- sweep(X,2,mu[[k]])%*%U[[k]]%*%D2i[[k]]
    delta[,k] <- offset[k] - diag(Xs%*%t(Xs))/2
  }
  return(list(delta=delta,class=apply(-delta,1,order)[1,]))
}
```

Création d'un jeu de données artificiel

```
par(mfrow=c(1,1))
x1 <- rnorm(100,1,.5)
x2 <- rnorm(100,2,.25)
y1 <- rnorm(100,1,.5)
y2 <- rnorm(100,2,.25)
y <- c(rep(-1,100),rep(1,100))
X <- cbind(c(x1,x2),c(y1,y2))
minx <- min(X[,1])
maxx <- max(X[,1])
miny <- min(X[,2])
maxy <- max(X[,2])
xx <- seq(minx,maxx,length=50)
yy <- seq(miny,maxy,length=length(xx))

plot(X,col=(y+1)/2+2)
```

Discriminant linéaire LDA

```
X.fit <- lda.fit(y,X,prior=c(1,1)/2)
print((X.pred <- lda.pred(X,X.fit))$class)
print(sum((y+1)/2+1 != X.pred$class)/length(y))
```

```
plot(X,col=(y+1)/2+2,xlab="",ylab="",pch=15,main="lda, orig features")
title(xlab=paste("err rate =",sum((y+1)/2+1 != X.pred$class)/length(y)))

for (i in 1:length(xx)) {
  for (j in 1:length(yy)) {
    class <- lda.pred(c(xx[i],yy[j]),X.fit)$class
    points(xx[i],yy[j],col=class+1,pch="+")
  }
}
```

Discriminant quadratique QDA

```
Xq.fit <- lda.fit(y,X,prior=c(1,1)/2,qda=T)
print((Xq.pred <- lda.pred(X,Xq.fit))$class)
print(sum((y+1)/2+1 != Xq.pred$class)/length(y))

plot(X,col=(y+1)/2+2,xlab="",ylab="",pch=15,main="qda, orig features")
title(xlab=paste("err rate =",sum((y+1)/2+1 != Xq.pred$class)/length(y)))

for (i in 1:length(xx)) {
  for (j in 1:length(yy)) {
    class <- lda.pred(c(xx[i],yy[j]),Xq.fit)$class
    points(xx[i],yy[j],col=class+1,pch="+")
  }
}
```

On considère les vins du jeu de données wine. Il consiste en cent soixante-dix-huit vins provenant de trois régions d'Italie et décrits par treize variables quantitatives.

```
wine.fl <- "http://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data"
wine <- read.csv(wine.fl,header = F)
# Names of the variables
wine.names=c("Alcohol", "Malic acid", "Ash", "Alcalinity of ash", "Magnesium",
             "Total phenols", "Flavanoids", "Nonflavanoid phenols", "Proanthocyanins"
             "Color intensity", "Hue", "OD280/OD315 of diluted wines", "Proline")
colnames(wine)[2:14]=wine.names
colnames(wine)[1]="Class"
wine$Class <- as.factor(wine$Class)
library(MASS)
wine.lda <- lda(Class ~ ., data=wine)
```

Diagramme de dispersion des fonctions discriminantes
Comme il y a 3 classes, on peut projeter le résultat sur 2 axes (un de moins que le nombre de classes).

```
wine.lda.values <- predict(wine.lda)
plot(wine.lda.values$x[,1],wine.lda.values$x[,2]) # diagramme dispersion
text(wine.lda.values$x[,1],wine.lda.values$x[,2],Class,cex=0.7,pos=4,col="red") # eti
```

LD1 et LD2 sont les vecteurs portant les axes de la projections (vecteurs propres associées aux plus grandes valeurs propres).

Pour visualiser l'espace de décision, on peut utiliser la librairie klaR.

```
install.packages('klaR')
library(klaR)
partimat(Class ~ Alcohol + Ash + Proanthocyanins + Proline, data = wine, method = "ld
```