SY09 : Rapport TP4
Discrimination

Cathelain Valentin Marechal Anaig

22 juin 2016

Résumé

Ce nouveau TP nous propose d'apprendre à maîtriser de nouvelles méthodes supervisées : tout d'abord différentes méthodes d'analyse discriminante, puis des méthodes de régression logistique, et enfin nous nous intéresserons aux arbres binaires.
Nous allons voir que faire un choix parmi ces méthodes n'est pas évident mais que l'on peut tout de même avoir une intuition de la qualité des résultats de chacune, en fonctions des données de base. Ainsi, elles reposent parfois sur des hypothèses de distribution qui peuvent affecter la robustesse de la méthode. Certaines, plus robustes, sont également plus sensibles aux variations. Afin de comprendre ce sujet dans toute sa complexité, nous travaillerons dans un premier temps sur des données simulées, puis nous tacherons d'appliquer les méthodes sur des jeux de données réelles.

1 Programmation

1.1 Analyse discriminante

L'analyse discriminante se base sur l'hypothèse que le vecteur aléatoire étudié suit une loi normale. De là, on tente d'en déduire les différents paramètres de cette loi, l'espérance, la variance et la probabilité a priori des classes. Grâce à la règle de Bayes, on peut ensuite obtenir des règles de décisions grâce à ces estimations.
Nous avons programmé trois modèles d'analyse discriminantes, qui sont l'analyse discriminante quadratique, l'analyse discriminante linéaire et le classifieur bayésien naïf. Pour cela il faut comprendre comment estimer les paramètres :

	$\hat{\pi}_k$	$\hat{\mu}_k$	$\hat{\Sigma}_k$
ADQ	$\frac{n_k}{n}$	$\frac{\sum z_{ik} x_i}{n_k}$	$\frac{\sum_i z_{ik}(x_i-\hat{\mu}_k)(x_i-\hat{\mu}_k)^t}{n_k} = V_k$
ADL	$\frac{n_k}{n}$	$\frac{\sum z_{ik} x_i}{n_k}$	$\frac{\sum(n_k-1)}{n}$
NBA	$\frac{n_k}{n}$	$\frac{\sum z_{ik} x_i}{n_k}$	$diag(V_k)$

Nos fonctions se trouvent en annexe. Elles prennent en entrée la matrice d'apprentissage ainsi que ses étiquettes et renvoient les paramètres $\hat{\pi}$, $\hat{\mu}$ et $\hat{\Sigma}$. Ils sont respectivement stockés dans un vecteur de longueur k, une matrice de dimension (1×p), et un tableau à 3 dimensions (p×p×k).

La fonction suivante à développer calcule les probabilités a posteriori d'un ensemble de données. Cette fonction prend donc en entrée les estimations des paramètres précédemment calculés avec les différentes méthodes et un ensemble de données de test à classer. Elle retournera une structure

contenant la probabilité d'affectation de ces nouvelles données à un groupe et le classement associé. Pour ce faire, on utilise la formule de probabilités a posteriori :

$$\mathcal{P}(\omega_k|x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^{g} \pi_l f_l(x)} \tag{1}$$

1.2 Régression logistique

La suite du TP nous propose de programmer un modèle de régression logistique. Cette méthode, contrairement à l'analyse discriminante, ne fait pas d'hypothèse sur la distribution des données. Elle estime directement les probabilités d'appartenance aux classes. L'apprentissage se fait grâce à l'algorithme d'optimisation itératif de *Newton-Raphson*. Cette fonction prend en paramètre la matrice d'apprentissage et ses paramètres, un booléen permettant d'ajouter ou non un intercept et une valeur espilon permettant de règler le nombre d'itérations.
Pourquoi l'intercept est-il ici important ? La régression linéaire, comme pour l'analyse discriminante quadratique, présuppose que la frontière de décision est linéaire. En effet, $\mathcal{P}(\omega_1|x) = \mathcal{P}(\omega_2|x)$ ssi $\omega^t x = 0$. On obtient donc l'équation d'un hyperplan, et cela signifie que la frontière passera obligatoirement par l'origine. L'intercept permet donc une liberté supplémentaire en permettant à la frontière de ne pas nécessairement passer par l'origine.
Notre fonction log.app est visible en annexe. Lors de nos tests, nous avons pris soin de vérifier que la matrice Hessienne était bien définie négativement et que la log-vraisemblance augmente au fil des itérations (en effet, la fonction est concave).

A partir de Newton-Raphson, nous avons développé une deuxième fonction dont les arguments d'entrée sont les données de test et l'estimateur du maximum de vraisemblance, beta, retournée par log.app. En prenant soin d'ajouter un intercept aux données de test en fonction de beta, la fonction retourne les probabilités à posteriori grâce à la formule $\frac{exp(\beta^t x)}{1+exp(\beta^t x)}$ et le classement associé.

2 Application

2.1 Test sur données simulées

Afin de comparer les performances des différentes méthodes vues précédemment, on répète un même protocole 20 fois sur 3 jeux de données différents, en faisant d'abord l'apprentissage des données avant de classer des données de test et calculer le taux d'erreur.

	Synth1-1000	Synth2-1000	Synth3-1000
ADL	0.0381	0.0097	0.0228
ADQ	0.0271	0.0091	0.0117
NBA	0.0355	0.0152	0.0118
RL	0.0228	0.0107	0.0184
RLQ	0.0298	0.0116	0.0112
Arbre	0.0397	0.0197	0.0206

On trace également les frontières de décision pour l'analyse des résultats :

Frontières de décision pour synth1-1000

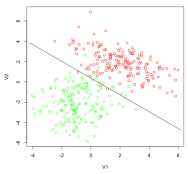

FIGURE 1 – ADL

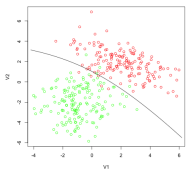

FIGURE 2 – ADQ

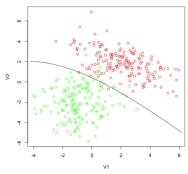

FIGURE 3 – NBA

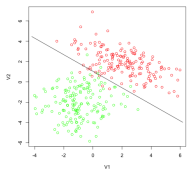

FIGURE 4 – RL

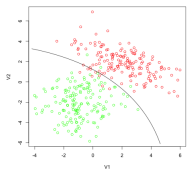

FIGURE 5 – RLQ

Frontières de décision pour synth2-1000

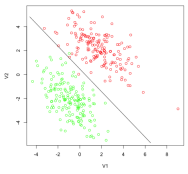

FIGURE 6 – ADL

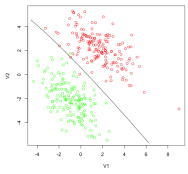

FIGURE 7 – ADQ

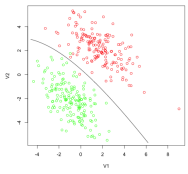

FIGURE 8 – NBA

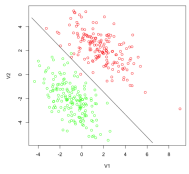

FIGURE 9 – RL

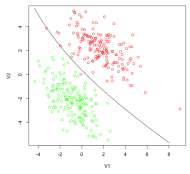

FIGURE 10 – RLQ

Frontières de décision pour synth3-1000

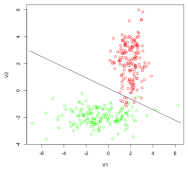

FIGURE 11 – ADL

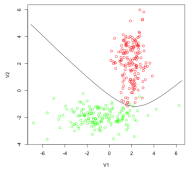

FIGURE 12 – ADQ

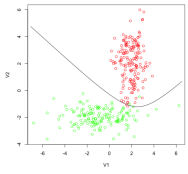

FIGURE 13 – NBA

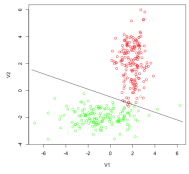

FIGURE 14 – RL

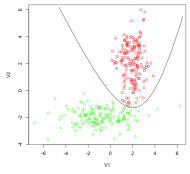

FIGURE 15 – RLQ

Observations et interprétation On remarque que globalement tous les modèles donnent des résultats corrects. Sachant que les données suivent dans chaque classe une loi normale multivariée, il semble normal que l'analyse discriminante fonctionne relativement bien car c'est l'hypothèse de départ de cette méthode. Les variations les plus visibles se situent au niveau du classifieur bayésien naïf et de l'analyse discriminante linéaire.
Au niveau du classifieur baysésien naïf, on note graphiquement qu'il marche particulièrement bien pour le dernier jeu de données, un peu moins bien pour les deux premiers (même si le taux d'erreur reste bon pour le deuxième).
On obtient de bons résultats au niveau de l'analyse discriminante linéaire, au niveau de la frontière de décision et du taux d'erreur, sur le jeu de données 2.
L'analyse discriminante linéaire et la régression logistique sont moins performantes que les autres méthodes pour le jeu 3, avec un taux d'erreur plus élevé. Une frontière de décision quadratique semble être plus adaptée pour ces données. On peut également s'en apercevoir en observant les répartitions des classes sur le graphe du jeu de données Synth3-1000.

En calculant les matrices de variance pour chaque classe des jeux de données, nous pouvons appuyer nos observations :

Synth1-1000 : $\Sigma_1 = \begin{pmatrix} 3.082769 & -1.447304 \\ -1.447304 & 1.843748 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 1.74283754 & -0.04648104 \\ -0.04648104 & 2.71414009 \end{pmatrix}$

Synth2-1000 : $\Sigma_1 = \begin{pmatrix} 3.385508 & -1.822969 \\ -1.822969 & 2.268692 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 2.044616 & -1.419177 \\ -1.822969 & 2.268692 \end{pmatrix}$

Synth3-1000 : $\Sigma_1 = \begin{pmatrix} 0.52559436 & 0.07576519 \\ 0.07576519 & 1.99854729 \end{pmatrix}, \Sigma_2 = \begin{pmatrix} 3.72682605 & 0.02707343 \\ 0.02707343 & 0.33236760 \end{pmatrix}$

On peut voir que le troisième jeu a des matrices de variance proches de matrices diagonales, c'est-à-dire que les coefficients en dehors de la diagonale sont très faibles. Cela explique le bon résultat du classifieur bayésien naïf sur ce jeu. Il est en effet basé sur une forte indépendance des variables. De plus, l'ADL présente de bons résultats pour le jeu 2 dont les matrices de variance se ressemblent relativement.

Nous avons également recalculé le taux d'erreur pour la régression logistique sans intercept afin de juger l'importance de ce facteur.

	Synth1-1000	Synth2-1000	Synth3-1000
RL inter=1	0.0261	0.0241	0.0202
RL inter=0	0.0388	0.0349	0.0238
RLQ inter=1	0.0249	0.0253	0.0156
RLQ inter=0	0.0324	0.0328	0.0127

Le taux d'errreur est sensiblement plus important pour les jeux 1 et 2. Cependant pour le 3ème jeu, le taux d'erreur est très correct sans intercept. On peut donc émettre l'hypothèse que la frontière de décision pour ce jeu passe près de l'origine. C'est ce que nous pouvons observer graphiquement :

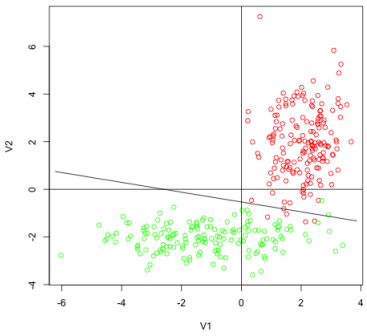

FIGURE 16 – Régression logistique sur le jeu 3 avec intercept

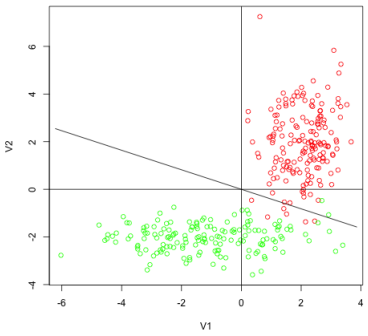

FIGURE 17 – Régression logistique sur le jeu 3 sans intercept

2.2 Arbres

Pour construire un arbre d'apprentissage, nous utilisons la fonction *tree* pour obtenir un arbre complet. La fonction est configurée de façon à ce que l'algorithme développe l'arbre jusqu'à obtenir des sous-ensembles d'apprentissage avec des régions parfaitement homogènes.

Malheureusement, cet arbre obtenu s'attache trop aux spécificités de l'arbre d'apprentissage considéré et présente de mauvaises capacité de généralisation. Il est donc nécessaire d'élaguer cet arbre en coupant certaines branches. Pour cela, nous utilisons d'abord la fonction *cv.tree*, qui nous permet de déterminer un nombre de feuilles optimal **k**.

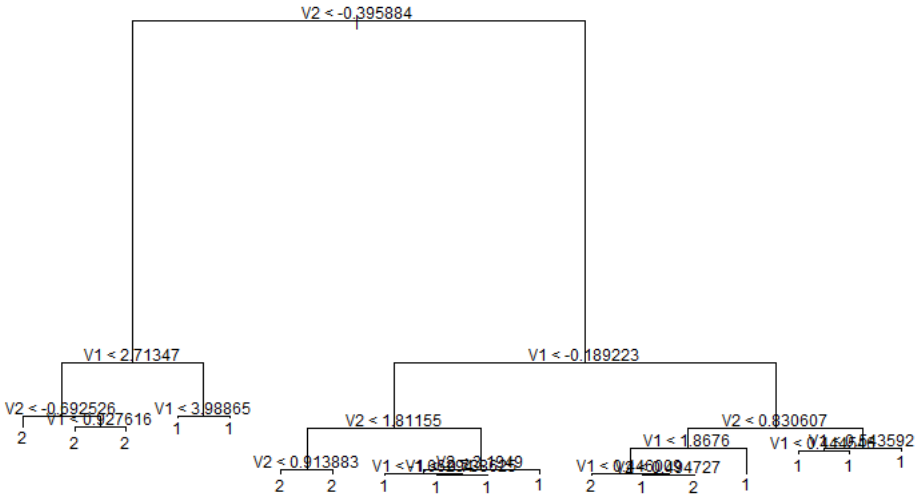

FIGURE 18 – Arbre obtenu

Dans notre exemple, nous utilisons le jeu de données Synth1-1000. Nous voyons que la déviance atteint sa valeur minimale dès $k = 8$. Nous choisirons donc ce nombre de noeuds terminaux pour effectuer l'élagage. Pour cela, nous utilisons la fonction *prune.misclass* avec en paramètre **k**.

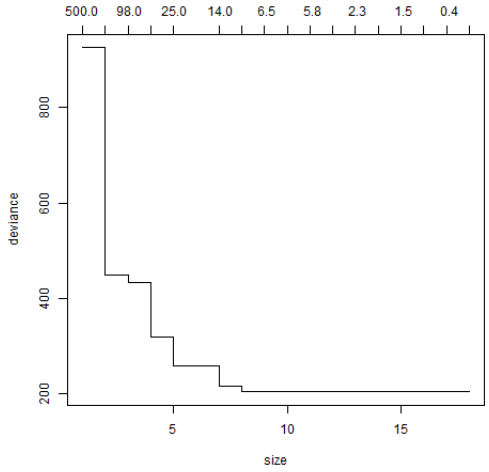

FIGURE 19 – Déviance en fonction de **k**

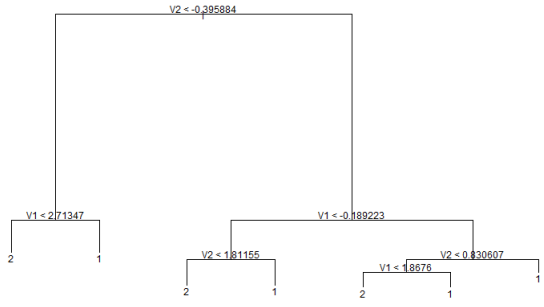

FIGURE 20 – Arbre élagué

3 Test sur données réelles

3.1 Données "Pima"

Dans cette partie nous traitons un échantillon de 532 individus, qui comportent chacun 7 attributs. L'objectif est de pouvoir prédir des cas de diabète à partir d'un ensemble d'apprentissage.

Méthode	Taux d'erreur
ADL	0.228
ADQ	0.241
NBA	0.242
RL	0.286
RLQ	0.231
Arbre	0.239

Les performances sont assez mauvaises pour toute les méthodes employées avec un taux d'erreur proche des 23%. Ces chiffes reflètent ce que l'on peut observer si l'on trace le graphe du jeu de données. On distingue difficilement les deux classes, ce qui montre que la différenciation des individus n'est pas aisée.

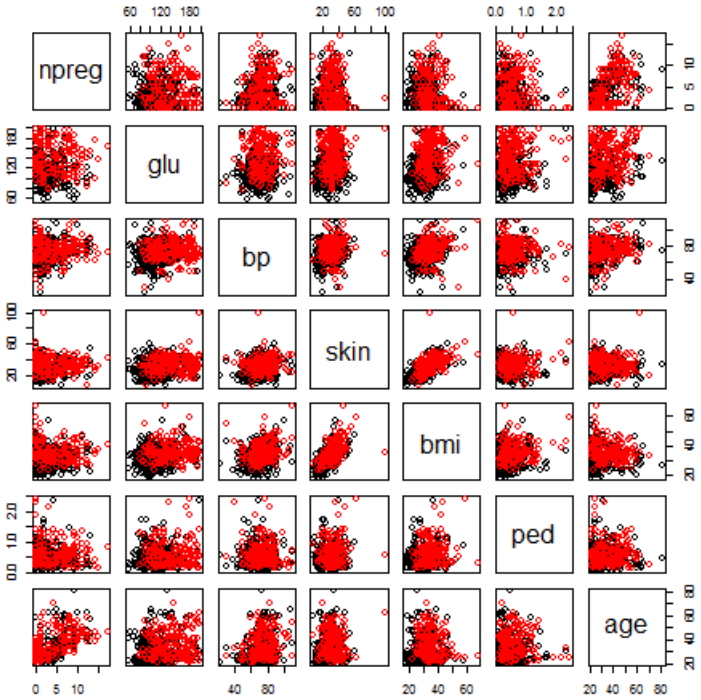

FIGURE 21 – Données Pima

3.2 Données "breast cancer Wisconsin"

	Taux d'erreur
ADL	0.0467
ADQ	0.0499
NBA	0.0386
RL intr=1	0.0405
RL intr=0	0.1555
Arbre	0.0515

L'analyse discriminante présente des résultats corrects, un peu plus élevés que lors de nos tests sur données simulées. Il se pourrait donc que les données suivent une loi normale multivariée. L'analyse discriminante quadratique est la méthode présentant le taux d'erreur le plus élevé. Or, c'est la méthode la plus sensible : comme le jeu présente une bonne quantité de données (683), on peut émettre l'hypothèse qu'il existe quelques données atypiques qui ont affaibli l'analyse discriminante quadratique. Les données ne suivent peut-être parfaitement une loi gaussienne. Le classifieur bayésien, lui, présente les meilleurs résultats. On peut émettre l'hypothèse que les données sont relativement indépendantes.
Quant à la régression linéaire, elle présente également de bons résultats, mais seulement en utilisant un intercept. La frontière de décision est donc éloignée de l'origine.
Le taux d'erreur pour l'arbre de décision est bon. Cela n'est pas étonnant car c'est une méthode très flexible qui marche dans de nombreux cas.

3.3 Challenge SPAM

Nous avons ici un jeu de données de taille plus importante (4601x58).

	Taux d'erreur
ADL	0.1112
ADQ	0.1611
NBA	0.1937
RL intr=1	0.0712
RL intr=0	0.0807
RLQ intr=1	0.4171
RLQ intr=0	0.4560
Arbre	0.0876

On remarque les taux d'erreur moyens sont globalement assez élevés avec ce jeu de données. Les méthode présentant les meilleures performances sont la classification par arbres et la régression logistique. Nous avons ici élagué les arbres obtenus en laissant uniquement $k = 17$ noeuds terminaux. Par ailleurs, les performances de la régression logistique sont meilleurs en utilisant un intercept ici aussi. Néanmoins la régression logistique quadratique donne de piètres résultats avec ces données. On sait que ce classifieur est moins robuste que le régression logistique classique et permet de contruire des frontières pus complexe. Ici, il y a certainement eu un sur-apprentissage et le modèle d'apprentissage ne convient pas à une généralisation sur des données dispersées.

4 Conclusion

Ce TP nous a permis de mettre en application de nouvelles méthodes d'apprentissage, sur des jeux de données plus ou moins conséquents. Nous avons pu mettre en avant le fait que les performances d'un classifieur dépendaient d'un certain nombre de paramètres liés au jeu de données.

Il n'y a malheureusement pas moyen de prédire quel classifieur nous permettra d'obtenir les meilleurs résultats face à un nouveau jeu, cependant nous avons vu au long de ce TP quels éléments pouvaient orienter notre décision. Les arbres de décisions, bien qu'ils ne donnent pas toujours le résultat optimal, semblent néanmoins être un outil sûr, renvoyant un résultat correct quel que soit le jeu car ils ne sont pas paramétriques, et ne reposent sur des hypothèses de distribution. Un jeu de données suivant une loi normale va lui nous orienter vers l'analyse discriminante qui peut être très performante si les hypothèses sont vérifiées, avec trois modèles plus ou moins robustes. De plus, le protocole de calcul du taux d'erreur pour chaque méthode est un bon moyen pour faire son choix lorsque le nombre de données nous le permet : nous l'avons appliqué sur des données de test tout d'abord, puis nous avons vu qu'il pouvait être utilisé efficacement sur des données réelles.

A Analyse discriminante

A.1 Analyse discriminante quadratique

```
    adq.app<-function(Xapp,zapp){

param<-NULL  #Variable qui contiendra les résultats
tablek<-table(zapp) #nb d'individus dans chaque classe
k = length(tablek)  #nb de classes
p<-dim(Xapp)[2] #nb param
pik<-vector(length=k)#probabilité à priori
mu<-matrix(nrow=k,ncol=p) #Espérance
vk<-array(dim=c(p,p,k)) #Covariance

for (i in 1:k){
pik[i]<-tablek[i]/length(zapp)
mu[,i]<-apply(Xapp[which(zapp==i),],2,mean)
vk[,,i]<-cov(Xapp[which(zapp==i),])

}
param$prop<-pik
param$moy<-mu
param$sig<-vk
(param)
}
```

A.2 Analyse discriminante linéaire

```
    adl.app<-function(Xapp,zapp){

param<-NULL #Variable qui contiendra les résultats
tablek<-table(zapp) #nb d'individus dans chaque classe
k = length(tablek) #nb de classes
n = length (zapp) #nb d'individus
p<-dim(Xapp)[2] #nb params
pik<-vector(length=k)#prob a priori
mu<-matrix(nrow=k,ncol=p)#Esperance
vk<-array(dim=c(p,p,k))#Covariance
x <-0

for (i in 1:k){
pik[i]<-tablek[i]/length(zapp)
mu[,i]<-apply(Xapp[which(zapp==i),],2,mean)
x <- x + (tablek[i]-1)*cov(Xapp[which(zapp==i),]) #calcul par recursivite de sum(nk-1)
}
for (i in 1:k)
vk[,,i]<- x/(n-k)
param$prop<-pik
param$moy<-mu
param$sig<-vk
(param)
```

```
}
```

A.3 Classifieur bayésien naïf

```
    nba.app <- function(Xapp,zapp){

param<-NULL #Variable qui contiendra les résultats
tablek<-table(zapp) #nb d'individus dans chaque classe
k = length(tablek) #nb de classes
p<-dim(Xapp)[2] #nb d'individus
pik<-vector(length=k)#nb params
mu<-matrix(nrow=k,ncol=p)#Esperance
vk<-array(dim=c(p,p,k))#Covariance

for (i in 1:k){
pik[i]<-tablek[i]/length(zapp)
mu[,i]<-apply(Xapp[which(zapp==i),],2,mean)
vk[,,i]<-cov(Xapp[which(zapp==i),])
vk[,,i]<-diag(diag(vk[,,i]))#On prend la diag de Vk et on en fait une matrice diagonale

}
param$prop<-pik
param$moy<-mu
param$sig<-vk
(param)
}
```

A.4 Probabilités a posteriori

```
    ad.val <- function(param, Xtst){
adval <- NULL
k = length(param$prop)
t <- array(dim=c(nrow(Xtst),k))
sum <- 0
for (i in 1:k){
dens<- mvdnorm(Xtst, param$moy[,i], param$sig[,,i])
t[,i] <- param$prop[i] * dens
sum <- sum + t[,i]
}
for (i in 1:k){
t[,i] <- t[,i] /sum
}

adval$prob <- t
adval$class <- (t[,1] > 0.5)*1
adval
}
```

B Régression logistique

B.1 Newton-Raphson

```
    log.app <- function(Xapp,zapp,intr,epsi){

Xapp <- as.matrix(Xapp)
n=dim(Xapp)[1]
p=dim(Xapp)[2]

if(intr==1){
one<-cbind(rep(1, n))
Xapp<-cbind(one,Xapp)
beta<- rep(0,p+1)
}else{
beta<- rep(0,p)
}
t <-(zapp==1)

niter <- 0
ndiff=epsi+1

while (ndiff>epsi){

pq <- post.pr(Xapp,beta)  #prob a posteriori
gradbeta <- t(Xapp)%*%(t-pq)  #gradient de la log-vraisemblance
matW <- diag(pq*(1-pq))
matH <- -t(Xapp)%*%matW%*%Xapp  #matrice hessienne
logL <- sum(t*log(pq)+(1-t)*log(1-pq))  #log-vraisemblance

betaold <- beta
beta <- beta-ginv(matH)%*%gradbeta

ndiff <- sum((betaold-beta)^2)

niter <- niter + 1}

log <- NULL
log$beta <- beta
log$niter <- niter
log$logL <- logL
log
}
```

B.2 Classement par probabilité a posteriori

```
    post.pr <- function(x,beta){

n <- dim(x)[1]
matbeta <- matrix(rep(beta,n),nrow=n,byrow=T)
prod <- rowSums(matbeta*x) #t(matbeta) ?
```

```
res <- exp(prod)/(1+exp(prod))

}

log.val<- function(Xtst,beta){

pbeta<-dim(beta)[1]
pXtst<-dim(Xtst)[2]
n=dim(Xtst)[1]

if (pbeta!=pXtst){
one<-cbind(rep(1, n))
Xtst<-cbind(one,Xtst)
}

prob<-post.pr(Xtst,beta)
probclass <- (prob>0.5)*1
probclass
}
```

C Arbre de décision

```
    arbre<-function(Xapp,zapp,Xtst,K){

#K dépend du jeu de données, à déterminer avec la fonction cv.tree

#apprentissage
zapp <-as.factor(zapp)
data.app<-cbind(Xapp,zapp)
treeapp<-tree(zapp~.,data=data.app,control=tree.control(nobs=dim(Xapp)[1],mindev = 0.0001))

#elagage
treeprune<-prune.misclass(treeapp,best=K)

#classification
pre<-predict(treeprune,Xtst)
zres=as.vector((pre[,1]>0.5)*1)
}
```