MASTER2- MLDS Année 2017-2018



**TP.2 Classification supervisée**

**Régression linéaire - K plus proches voisins – Bayésien naïf**

Rendu : TP à faire en binôme et à envoyer par mail à l’adresse suivante : [l.labiod@gmail.com](mailto:l.labiod@gmail.com) au plus tard le 30 .11.2011

**2 Données réelles (prostate)**

**2.1 Description des données prostate**

Récupérer le jeu de données prostate, les Données examinent la corrélation entre le niveau de l'antigène spécifique de la prostate et un certain nombre de mesures cliniques chez les hommes qui étaient sur le point de recevoir une prostatectomie radicale. La variable lpsa est la réponse au traitement qui sera discrétiser en deux catégories ; high si lpsa > median(lpsa) et 0 sinon. On veut construire un score de détection de la réponse applicable aux patients. Pour chaque patient on a mesuré une batterie de critères et finalement *p* = 3 critères ( lcavol, lweight et age) ont étés retenus pour construire le score.

**A. Charger le jeu de données dans R, décrire le jeu de données prostate**

- Description de la table prostate

library(ElemStatLearn)

data(prostate)

- Information sur la table prostate

?prostate

- attacher la table prostate

attach(prostate)

- Binariser la réponse lpsa ; high si lpsa > median(lpsa) et 0 sinon

g <- factor(ifelse(lpsa > median(lpsa), "high", "low"))

- utiliser les 3 variables explicatives lcavol, lweight and age pour modéliser les classes g et tracer le nuage des points de ces trois variables en indiquant les classes

library(lattice)

splom(~prostate[,1:3], groups=g)

- Ou utiliser la fonction pairs

pairs(prostate[,1:3], col=as.numeric(g))

**2.2 Régression linéaire**

**B. Pourquoi la régression linéaire n'est pas adaptée!?**

- régression linéaire sur une variable indicatrice (binaire)

- claculer le y binaire

y <- ifelse(g=="high", 1, 0)

- Calculer la régression linéaire

lm.fit <- lm(y~lcavol+lweight+age)

- coefficients de régrssion

lm.beta <- lm.fit$coef

- tracer le modèle estimé lcavol et age pour lweight moyen

b <- -lm.beta[2]/lm.beta[4]

a <- (0.5 - lm.beta[1] - lm.beta[3]\*mean(lweight))/lm.beta[4]

plot(lcavol, age, col=g)

abline(a,b)

- prédiction de y

yhat <- predict(lm.fit)

- prediction de la classe g

lm.ghat <- factor(ifelse(yhat > 0.5, "high", "low"))

- nombre d'exemples mal classés

sum(lm.ghat != g)

- erreur de classification

mean(lm.ghat != g)

- matrice de confusion

table(lm.ghat, g)

**2.3 KNN**

***C.* On s’intéresse d’abord à la méthodologie du choix de *k***

(1) Créer un jeu de données de données d’apprentissage (75% des données) et un jeu de données test (25% des données) avec le code suivant.

**set.seed**(30)

**X=cbind(g,** lcavol,lweight,age**)**

tr <- **sample**(1:**nrow**(X),72)

Xtrain <- X[tr,]

Xtest <- X[-tr,]

(2) Calculer les taux d’erreur sur les données test pour *k* variant de 1 à 100. Avec la fonction plot, représenter ce taux d’erreur test en fonction de *k* (contrôler que l’abscisse du graphique partde 0). Avec la fonction which.min, trouver le nombre de voisins qui donne la plus petite erreur test.

(3) Recommencer avec un autre découpage aléatoire apprentissage/test et représenter la courbe d’évolution du taux d’erreur test sur le même graphique qu’à la question précédente.

**library**(class)

kmax=100

err\_test <- **rep**(NA,kmax)

**for** (k **in** 1:kmax)

{

pred <- **knn**(Xtrain[,-1],Xtest[,-1],Xtrain[,1],,k)

err\_test[k] <- **sum**(pred!=Xtest[,1],)/**length**(Xtest[,1],)

}

lim <- **c**(0,**max**(err\_test))

**plot**(err\_test,type="l",ylim=lim,col=2,xlab="nombre de voisins",

ylab= "taux d'erreur")

**which.min**(err\_test)

*#Nouveau decoupage apprentissage/test*

**set.seed**(10)

tr <- **sample**(1:**nrow**(X),90)

train <- X[tr,]

test <- X[-tr,]

**for** (k **in** 1:kmax)

{

pred <- **knn**(Xtrain[,-1],Xtest[,-1], Xtrain [,1],k)

err\_test[k] <- **sum**(pred!=Xtest[,1])/**length**(Xtest[,1]))

}

**points**(err\_test,type="l",col=4)

**legend**("bottomright", legend=**c**("decoupage 1", "decoupage 2"), lty=1, col=**c**(2,4))

(4) Exécuter le code suivant et faire un choix pour *k*.

B<- 20

kmax <- 100

err\_test <- **matrix**(NA,kmax,B)

**for** (b **in** 1:B)

{

tr <- **sample**(1:**nrow**(X),90)

Xtrain <- X[tr,]

Xtest <- X[-tr,]

**for** (k **in** 1:kmax)

{

pred <- **knn**(Xtrain[,-1],Xtest[,-1], Xtrain [,1],k)

err\_test[k,b] <- **sum**(pred!= Xtest[,1])/**length**(Xtest[,1])

}

}

mean\_err\_test <- **apply**(err\_test,1,mean)

lim <-**c**(0,**max**(err\_test))

**matplot**(err\_test,type="l",lty=2,col=2,ylim=lim, xlab="nombre de voisins",ylab="taux d'erreur")

**matpoints**(mean\_err\_test,type="l",col=2,lwd=4)

**legend**("bottomright", legend=**c**("Erreur moyenne", "Erreurs conditionnelles"),

lty=**c**(1,3),lwd=**c**(4,2),col=**c**(2,2))

(5) Choisir maintenant le nombre *k* de voisin en utilisant par validation croisée (cross validation) leave-one-out (LOO) avec la fonction knn.cv.

?knn.cv

err\_test <- **rep**(NA,kmax)

**for** (k **in** 1:kmax)

{

pred <- **knn.cv**(X[,-1], X[,1],k)

err\_test[k] <- **sum**(pred!= X[,1])/**length**(X[,1])

}

lim <-**c**(0,**max**(err\_test))

**plot**(err\_test,type="l",col=2,ylim=lim,xlab="nombre de voisins", ylab="taux d'erreur")

**points**(mean\_err\_test,type="l",col=4,lwd=1)

**legend**("bottomright", legend=**c**("Erreur loo", "Erreur moyenne"), col=**c**(2,4),lty=1)

(6) Faire un petit bilan méthodologique concernant le choix du paramètre *k*.

**D. On veut maintenant non seulement choisir *k* mais également avoir une idée de l’erreur de prédiction de ce classifieur. Pour cela, il faut utiliser des données n’ayant jamais été utilisées. Les données doivent donc être découpées en trois parties : apprentissage/validation/test .**

(1) Couper aléatoirement les données des deux parties : un ensemble "apprentissage-validation" (75 % des données) et un ensemble test de taille (25% des données).

**set.seed**(30)

tr <- **sample**(1:**nrow**(X),72)

trainval <- X[tr,]

test <- X[-tr,]

(2) Utiliser la première approche pour choisir *k* sur l’ensemble "apprentissage-validation" :

i. Choisir *k* en découpant les 945 données de l’ensemble "apprentissage-validation" en deux

parties : une partie "apprentissage" (50% des données) et une partie "validation"

(25 % des données). Choisir *k* qui minimise le taux d’erreur moyen sur les

ensembles de validations de *B* = 25 découpages.

B <- 25

kmax <- 50

err\_valid <- **matrix**(NA,kmax,B)

**for** (b **in** 1:B)

{

tr <- **sample**(1:**nrow**(Xtrainval),36)

Xtrain <- Xtrainval[tr,]

Xvalid <- Xtrainval[-tr,]

**for** (k **in** 1:kmax)

{

pred <- **knn**(Xtrain[,-1],Xvalid[,-1],Xtrain[,1],k)

err\_valid[k,b] <- **sum**(pred!=Xvalid[,1])/**length**(Xvalid[,1])

}

}

mean\_err\_valid <- **apply**(err\_valid,1,mean)

**plot**(mean\_err\_valid,type="l")

ii. Constuire le classifieur avec ce nombre de voisins sur l’ensemble "apprentissage-validation" et calculer le taux d’erreur des données test.

pred <- **knn**(Xtrainval[,-1],Xtest[,-1],Xtrainval[,1],k=**which.min**(mean\_err\_valid))

**sum**(pred!=Xtest[,1])/**length**(Xtest[,1])

(3) Utiliser la seconde approche pour choisir *k* par validation croisée LOO sur l’ensemble "apprentissage validation". Calculer ensuite le taux d’erreur des données test.

err\_valid <- **rep**(NA,kmax)

**for** (k **in** 1:kmax)

{

pred <- **knn.cv**(Xtrainval[,-1],Xtrainval[,1],k)

err\_valid[k] <- **sum**(pred!=Xtrainval[,1])/**length**(Xtrainval[,1])

}

**which.min**(err\_valid)

pred <- **knn**(Xtrainval[,-1],Xtest[,-1],Xtrainval[,1],k=**which.min**(err\_valid))

**sum**(pred!=Xtest[,1])/**length**(Xtest[,1])

**E. Pour les courageux, on pourrait recommencer avec plusieurs découpages des données en deux parties "apprentissage-validation" et "test". Cela permettrait d’avoir une erreur test moyenne, et une idée de sa variabilié. C’est assez rapide à faire avec la méthode de LOO pour le choix de *k.***

B <- 10

kmax <- 50

err\_valid <- **rep**(NA,kmax)

err\_test <- **rep**(NA,B)

**for** (b **in** 1:B)

{

tr <- **sample**(1:**nrow**(X),72)

Xtrainval <- X[tr,]

Xtest <- X[-tr,]

**for** (k **in** 1:kmax)

{

pred <- **knn.cv**(Xtrainval[,-1],Xtrainval[,1],k)

err\_valid[k] <- **sum**(pred!=Xtrainval[,1])/**length**(Xtrainval[,1])

}

pred <- **knn**(Xtrainval[,-1],Xtest[,-1],Xtrainval[,1],k=**which.min**(err\_valid))

err\_test[b] <- **sum**(pred!=Xtest[,1])/**length**(Xtest[,1])

}

**boxplot**(err\_test,main="Erreurs test pour 50 decoupages")

**2.4 Bayésien naïf**

1. Appliquer le classifieur byésien naïf sur la table X.

prostate.d<-prostate[, -c(9,10)] # ensemble apprentissage

## Utiliser le pakcage e1071

library(e1071)

m <- naiveBayes(g ~ ., data = prostate.d)

## alternativement:

m <- naiveBayes(prostate.d, g)

m

table(predict(m, prostate.d), g)

2. Appliquer le classifieur bayésien naïf sur la table X dérivée de la table prostate

**3. Données Spam**

Le jeu de données SPAM est une base de données e-mail, avec 4601 observations et 58 variables descriptives.

1. Obtenir une description de la table spam (utiliser le package{kernlab} pour récupérer le jeu de données)

2. Effectuer une première analyse statistique univariée et bivariée de la table spam

3. Réaliser une étude comparative des méthodes de classification suivantes : régression linéaire, k plus proches voisins et le classifieur bayésien naïf sur le jeu de données spam.

4. Refaire la même étude sur le jeu de donnée spam après normalisation (utiliser la normalisation suivante : diviser chaque case nij de la table spam par la racine carrée du produit des sommes marginale ni. Et n.j )

5. Faire un petit bilan concernant les méthodes utilisées et l’impact de la normalisation sur les performances de ces méthodes.