TP Analyse de données - Apprentissage supervisé (Classification)

Introduction

Nous allons utiliser le logiciel `R` (documentation ; possibilité d'utiliser Rstudio). Dans un dossier *AnalyseDeDonnees* créer deux sous dossiers :

- Code - dans lequel vous placerez vos fichiers de code
- Data - dans lequel vous placerez les fichiers du répertoire "Data" (à télécharger via Moodle)

Les 3 TP *Analyse de données* présentent différentes méthodes d'analyse de données : le but n'est pas de finir les TP le plus vite possible mais d'analyser les résultats ! Un rapport de 1 page **maximum** vous est demandé pour chacun des 3 TP : ne choisir que les résultats les plus intéressants et les **commenter**.

Ne pas hésiter à utiliser l'aide de `R` grâce à la commande :

```
help(...)
```

Classification - Approches de type prototype

0. Télécharger le cours sur la classification (seconde partie du cours).

Méthode des k-plus proches voisins

1. Créer un fichier *classification-kppv.R* et copier les lignes suivantes :

```
# Adresse du dossier où vous travaillez
setwd("/Users/.../TP/TP/Code")
# Packages utilisés dans la suite
library(class)
library(caret)
library(ROCR)
# Supprimer toutes les variables
rm(list=ls(all=TRUE))
# Supprimer tous les graphiques déjà  présents
graphics.off()
```

2. Charger les données (synthétiques) d'**apprentissage** et les séparer en 2 variables : données d'entrée et données de sortie. L'objectif est de déterminer si y vaut 1 ou 2 connaissant x_1 et x_2.

```
# Lecture des données d'apprentissage
data_train <- read.table("../Data/synth_train.txt",header=T,sep="\t");
print(data_train)
# Séparation des données et de la sortie
data_train_x <- data.frame(x1=data_train$x1,x2=data_train$x2)
```

3. Charger les données **test** et les séparer aussi en 2 variables :

```
# Lecture des données test
data_test <- read.table("../Data/synth_test.txt",header=T,sep="\t");
print(data_test)
# Séparation des données et de la sortie
data_test_x <- data.frame(x1=data_test$x1,x2=data_test$x2)
```

4. Tracer les données (colorées par la sortie y)

```
# Graphique des données (colorées par la sortie y)
plot(data_train$x1,data_train$x2,col=data_train$y,pch=16)
```

5. Appliquer les k-plus proches voisins sur les données d'**apprentissage** :

```
# nombre de voisins (par ex proche de la racine carré du nombre d'obs)
num_of_neigh <- 10
data_train_predict <- knn(train=data_train_x,test=data_train_x,
                          cl=data_train$y,k=num_of_neigh)
# Affichage des résultats (étoile)
par(new=T)
plot(data_train$x1,data_train$x2,col=data_train_predict,pch=8)
# Calcul du taux d'erreur
error_rate <- mean(data_train_predict != data_train$y)
cat("error_rate using train data = ",error_rate)
```

6. Appliquer les k-plus proches voisins sur les données **test** :

```
data_test_predict <- knn(train=data_train_x,test=data_test_x,
                          cl=data_train$y,k=num_of_neigh)
# Affichage des données (cercle)
plot(data_train$x1,data_train$x2,col=data_train$y,pch=16)
par(new=T)
# Affichage des résultats (étoile)
plot(data_test$x1,data_test$x2,col=data_test_predict,pch=8)
# Affichage des vraies valeurs (triangle)
par(new=T)
plot(data_test$x1,data_test$x2,col=data_test$y,pch=2)
# Calcul du taux d'erreur
error_rate <- mean(data_test_predict != data_test$y)
cat("error_rate using test data = ",error_rate)
```

7. Afficher la matrice de confusion (définir les vrais pos./nég. et les faux pos./nég.). Imaginons que $y = 1$ (resp. $y = 2$) signifie que le patient est malade (resp. sain) ? Est-ce une bonne méthode ? Est-ce qu'il vaut mieux dans ce cas là avoir des faux positifs ou des faux négatifs ?

```
# Matrice de confusion
confmat = table(data_test_predict,data_test$y)
print("Confusion Matrix")
print(confmat)
# vrais positifs + vrais negatifs + faux positifs + faux négatifs
TP = confmat[1,1]; TN = confmat[2,2]; FP = confmat[1,2]; FN = confmat[2,1];
```

8. Définir la sensibilité, la spécificité, la précision et la prévalence. Les étudier en considérant l'exemple de la question précédente.

```
# Sensibilité (sensitivity ; TPR = true positive rate)
TPR = TP/(TP+FN)
cat("TPR",TPR,"\n")TP
# Spécificité (specificity ; TNR = true negative rate)
TNR = TN/(TN+FP)
cat("TNR",TNR,"\n")
# Précision (precision ; positive predictive value)
PPV = TP/(TP+FP)
cat("PPV",PPV,"\n")
# se compare à la prévalence (prevalence)
cat("Prev =",length(data_test$y[data_test$y==1])/length(data_test$y),"\n")
```

9. Calculer la précision avec la F-measure (dit F-score).

```
cat("F-score = ",2 * TPR * PPV / (TPR+PPV),"\n")
```

10. Nous allons maintenant étudier sur ce petit exemple l'importance du nombre de voisins. Pour cela, nous allons représenter graphiquement la frontière de décision pour $k = 1, 5, 10, 15, 20, 30$ voisins. Commencer par construire une grille de points.

```
# Explication visuelle de l'importance de la valeur de k (nb de voisins)
# Construction de la grille
gridx1 <- seq(from=min(data_train$x1),to=max(data_train$x1),length.out=50)
gridx2 <- seq(from=min(data_train$x2),to=max(data_train$x2),length.out=50)
grid <- expand.grid(x1 = gridx1, x2 = gridx2)
data_grid_x <- data.frame(x1=grid[,1],x2=grid[,2])
```

11. Appliquer la méthode des k-plus proches voisins pour $k = 1, 5, 10, 15, 20, 30$ sur la grille. Que pensez vous des résultats ? Que se passe t-il au niveau de la frontière ? Réfléchir à un moyen de trouver comment choisir k (en utilisant que les données d'apprentissage !).

```
# k plus proches voisins avec application sur les données de la grille
num_of_neigh_grid <- c(1,5,10,15,20,30)
par(mfrow=c(2,length(num_of_neigh_grid)/2))
for (i in 1:length(num_of_neigh_grid))
{
  num_of_n <- num_of_neigh_grid[i]
  data_g_pr <- knn(train=data_train_x,test=data_grid_x,
                           cl=data_train$y,k=num_of_n)
  plot(data_train$x1,data_train$x2,col=data_train$y,pch=16)
  title(paste("num of neighbours = ",toString(num_of_n)))
  par(new=T)
  plot(data_grid_x$x1,data_grid_x$x2,col=data_g_pr,pch=8,cex=0.5,ann=FALSE)
}
```

12. Avec la méthode des k-plus proches voisins il est très facile de déterminer la probabilité associée au résultat. En effet supposons un nouveau point dont les valeurs de y pour les 7-plus proches voisins sont :

$$y = 1,\ y = 2,\ y = 2,\ y = 2,\ y = 1,\ y = 2,\ y = 1.$$

Comme il y a 4 voisins sur 7 avec $y = 2$, on va classer ce nouveau point à $y = 2$ mais on peut aussi très facilement déterminer la probabilité de bon classement : $4/7 \sim 0.6$.

```
# k plus proches voisins avec les probas
data_test_predict_with_proba <- knn(train=data_train_x,test=data_test_x,
                        cl=data_train$y,k=num_of_neigh,prob=TRUE)
```

13. À partir de cette probabilité p et du résultat ($y = 1$ ou $y = 2$), il est possible de déterminer un score, c'est-à-dire de déterminer quelle est la probabilité que le patient soit malade (c'est-à-dire $y = 1$). Si $y = 1$ (resp. $y = 2$) on va considérer p (resp. $1 - p$).

```
# Calcul du score
score <- attr(data_test_predict_with_proba, "prob")
score <- ifelse(data_test_predict_with_proba == "1", 1-score, score)
```

14. À partir de ce score il est possible de tracer une courbe ROC. La droite $y = x$ est aussi tracée : celle-ci correspond au classifieur aléatoire.

```
pred_knn <- prediction(score, data_test$y)
perf <- performance(pred_knn, "tpr", "fpr")
plot(perf,colorize=TRUE)
par(new=T)
plot(c(0,1),c(0,1),type="l",ann=FALSE)
```

15. Pour connaître la performance de ce classifieur, il est possible de calculer l'aire sous la courbe.

```
# Aire sous la courbe
AUC <- performance(pred_knn, "auc")@y.values[[1]]
cat("AUC = ", AUC)
```

16. Il est possible de déterminer le meilleur seuil. Le repérer aussi sur la courbe ROC.

```
# Choix du seuil
result <- NULL
threshold <- seq(0,1,len=11)
for (s in threshold)
{
  test <- as.integer(score>=s)+1
  result <- c(result,1-mean(test != data_test$y))
}
plot(threshold,result,type="l")
cat("Meilleur seuil ", threshold[which.max(result)])
```

Arbres de classification

1. Créer un fichier *classification-CART.R* et copier les lignes suivantes :

```
# Adresse du dossier où vous travaillez
setwd("/Users/.../TP/TP/Code")
# Packages utilisés dans la suite
library(rpart)
# Supprimer toutes les variables
rm(list=ls(all=TRUE))
# Supprimer tous les graphiques déjà  présents
graphics.off()
```

2. Les données utilisées sont les mêmes que précédemment.

```
# Lecture des données d'apprentissage
data_train <- read.table("../Data/synth_train.txt",header=T,sep="\t");
plot(data_train$x1, data_train$x2, pch=data_train$y, col=data_train$y,
                                main="Training set")
legend("topleft", legend=c("classe1", "classe2"), pch=1:2, col=1:2)

# Création de la sortie (à  mettre sous le format facteur sinon
# un modèle de régression est créé)
data_train$y <- as.factor(data_train$y)
```

3. Créer l'arbre en utilisant les données d'apprentissage (méthode CART).

```
tree <- rpart(y~., data = data_train)
```

4. Afficher les informations sur l'arbre créé. Interpréter les résultats.

```
print(tree)
summary(tree)
```

5.Tracer l'arbre dans un graphique.

```
plot(tree)
text(tree)
```

6. Définir la variable correspond à la valeur de séparation sur x_1.

```
# Valeur de séparation de x1
split_x1 <- tree$splits[1,4]
```

7. Lire les données test et faire un graphique avec la droite de séparation pour les données d'apprentissage et les données test.

```
data_test <- read.table("../Data/synth_test.txt",header=T,sep="\t");
par(mfrow=c(1:2))
plot(data_train$x1, data_train$x2, pch=data_train_y, col=data_train$y,
                                                    main="Training set")
legend("topleft", legend=c("classe1", "classe2"), pch=1:2, col=1:2)
abline(v=split_x1,lty=2,col=4)
plot(data_test$x1, data_test$x2, pch=data_test$y, col=data_test$y,
                                                  main="Test set")
legend("topleft", legend=c("classe1", "classe2"), pch=1:2, col=1:2)
abline(v=split_x1,lty=2,col=4)
```

8. Évaluer la performance de l'arbre de classification sur les données test.

```
# Prediction sur les données test
data_test_x <- data.frame(x1=data_test$x1,x2=data_test$x2)
tree_predict_data_test <- predict(tree, newdata=data_test_x, type="class")

# Comparaison des valeurs prédites et des valeurs observées
table(tree_predict_data_test, data_test$y)

# Calcul du taux d'erreur
error_rate <- mean(tree_predict_data_test != data_test$y)
cat("error_rate using test data = ",error_rate)
```

9. Dans la méthode CART, l'arbre est élagué. Il est possible de construire l'arbre complet en modifiant les valeurs par défaut de la fonction *rpart*.

```
# Arbre de longueur maximale
tree_max <- rpart(y~., data=data_train, minsplit=2,cp=0)
# Tracer l'arbre maximal
par(mfrow=c(1,1))
plot(tree_max)
text(tree_max,use.n = TRUE,all=TRUE)
```

10. En affichant l'arbre complet, interpréter l'élagage précédent.

```
# Affichage de l'arbre
plotcp(tree_max)
```

Forêts aléatoires

1. Créer un fichier *classification-randomforest.R* et copier les lignes suivantes :

```
# Adresse du dossier où vous travaillez
setwd("/Users/.../TP/TP/Code")
# Packages utilisés dans la suite
library(randomForest)
# Supprimer toutes les variables
rm(list=ls(all=TRUE))
# Supprimer tous les graphiques déjà  présents
graphics.off()
```

2. Les données utilisées sont les mêmes que précédemment.

```
# Lecture des données d'apprentissage
data_train <- read.table("../Data/synth_train.txt",header=T,sep="\t");
# Séparation des données et de la sortie !!
data_train_x <- data.frame(x1=data_train$x1,x2=data_train$x2)
# Création de la sortie (à  mettre sous le format facteur sinon
# un modèle de régression est créé)
data_train_y <- as.factor(data_train$y)
```

3. Appliquer l'algorithme des forêts aléatoires.

```
# Forêts aléatoires
rf <- randomForest(x = data_train_x, y = data_train_y)

# Évolution de l'erreur en fonction du nombre d'arbres
# Ici ntree est fixé à la valeur par défaut = 500
plot(rf$err.rate[,1], type="l")

# Affichage des résultats
print(rf)
```

4. Plusieurs paramètres peuvent être modifiés (ils sont fixés pour le moment à leur valeur par défaut) :

- le nombre d'arbres considérés *ntree* (par défaut à 500)
- la taille de l'échantillon *sampsize* (par défaut pas fixé)
- la limite maximale du nombre de noeuds de chaque arbre *maxnodes* (par défaut non fixé)
- quand plus de 2 variables d'entrée, le nombre de variables testé à chaque division *mtry*

Modifier les 3 premiers paramètres pour tester (sur cet exemple très simple les changements ne sont pas majeurs).

5. Afficher l'importance de chaque variable.

```
# Importance des variables
rf$importance
varImpPlot(rf)
```

6. Tester l'algorithme de forêt aléatoire sur les données test. Est-ce mieux que les 2 précédentes méthodes (k-plus proches voisins et CART) ?

```
# Lecture des données test
data_test <- read.table("../Data/synth_test.txt",header=T,sep="\t");
# Séparation des données et de la sortie !!
data_test_x <- data.frame(x1=data_test$x1,x2=data_test$x2)

# Prediction sur les données test
rf_predit_data_test <- predict(rf, newdata=data_test_x)

# Comparaison des valeurs prédites et des valeurs observées
table(rf_predit_data_test, data_test$y)

# Calcul du taux d'erreur
error_rate <- mean(rf_predit_data_test != data_test$y)
cat("error_rate using test data = ",error_rate)
```

7. Pour comparer les résultats à la méthode des k-plus proches voisins nous allons appliquer les résultats sur une grille pour obtenir la frontière :

```
# Résultat de la méthode des forêts aléatoires
gridx1 <- seq(from=min(data_train$x1), to=max(data_train$x1), length.out=50)
gridx2 <- seq(from=min(data_train$x2), to=max(data_train$x2), length.out=50)
grid <- expand.grid(x1 = gridx1, x2 = gridx2)
data_grid_x <- data.frame(x1=grid[,1],x2=grid[,2])

rf_predit_data_grid <- predict(rf, newdata=data_grid_x)

plot(data_train$x1,data_train$x2,col=data_train$y,pch=16)
par(new=T)
plot(data_grid_x$x1,data_grid_x$x2,col=rf_predit_data_grid,pch=8,cex=0.5,
                                                        ann=FALSE)
```

0. Continuer à travailler sur le cours sur la classification (première partie du cours).

Analyse discriminante linéaire et quadratique

1. Créer un fichier *classification-analysediscriminante.R* et copier les lignes suivantes :

```
# Adresse du dossier où vous travaillez
setwd("/Users/.../TP/TP/Code")
# Packages utilisés dans la suite
library(MASS)
# Supprimer toutes les variables
rm(list=ls(all=TRUE))
# Supprimer tous les graphiques déjà  présents
graphics.off()
```

2. Lire les données.

```
# Lecture des données d'apprentissage
data_train <- read.table("../Data/synth_train.txt",header=T,sep="\t");
data_train_y <- data_train$y
data_train$y <- as.factor(data_train$y)
```

3. Appliquer la méthode d'analyse discriminante linéaire.

```
# Analyse discriminante LINEAIRE
lin_disc_an <- lda(y~., data = data_train)
```

4. Lire les données test et les prédire.

```
# Lecture des données test
data_test <- read.table("../Data/synth_test.txt",header=T,sep="\t");
data_test_x <- data.frame(x1=data_test$x1,x2=data_test$x2)
# Prédiction sur les données test
test_LDA_predict <- predict(lin_disc_an, newdata=data_test_x, type="class")
```

5. Évaluer la méthode sur le jeu de données test.

```
# Comparaison des valeurs prédites et des valeurs observées
table(test_LDA_predict$class, data_test$y)
# Calcul du taux d'erreur
lda_error_rate <- mean(test_LDA_predict$class != data_test$y)
cat("error rate using test data (LDA) = ", lda_error_rate)
```

6. Appliquer cette fois ci la méthode d'analyse discriminante quadratique.

```
# Analyse discriminante QUADRATIQUE
quad_disc_an <- qda(y~., data = data_train)
# Prédiction sur les données test
test_QDA_predict <- predict(quad_disc_an, newdata=data_test_x, type="class")
```

7. Évaluer la méthode sur le jeu de données test.

```
# Comparaison des valeurs prédites et des valeurs observées
table(test_QDA_predict$class, data_test$y)
# Calcul du taux d'erreur
qda_error_rate <- mean(test_QDA_predict$class != data_test$y)
cat("error rate using test data (QDA) = ", qda_error_rate)
```

8. Comparer les 2 méthodes en affichant leurs frontières sur une grille.

```
# Frontières LDA et QDA
gridx1 <- seq(from=min(data_train$x1), to=max(data_train$x1), length.out=40)
gridx2 <- seq(from=min(data_train$x2), to=max(data_train$x2), length.out=40)
grid <- expand.grid(x1 = gridx1, x2 = gridx2)
data_grid_x <- data.frame(x1=grid[,1],x2=grid[,2])

test_LDA_predict_grid <- predict(lin_disc_an, newdata=data_grid_x, type="class")
test_QDA_predict_grid <- predict(quad_disc_an, newdata=data_grid_x, type="class")

par(mfrow=c(1:2))
plot(data_train$x1,data_train$x2,col=data_train_y,pch=16)
par(new=T)
plot(data_grid_x$x1,data_grid_x$x2,col=test_LDA_predict_grid$class,pch=8,cex=0.5,
                                                                ann=FALSE)

plot(data_train$x1,data_train$x2,col=data_train_y,pch=16)
par(new=T)
plot(data_grid_x$x1,data_grid_x$x2,col=test_QDA_predict_grid$class,pch=8,cex=0.5,
                                                                ann=FALSE)
```

Bayésien naif

1. Créer un fichier *classification-bayesiennaif.R* et copier les lignes suivantes.

```
# Adresse du dossier où vous travaillez
setwd("/Users/.../TP/Code")
# Packages utilisés dans la suite
library(e1071)
# Supprimer toutes les variables
rm(list=ls(all=TRUE))
# Supprimer tous les graphiques déjà  présents
graphics.off()
```

2. Lire les données d'apprentissage.

```
# Lecture des données d'apprentissage
data_train <- read.table("../Data/synth_train.txt",header=T,sep="\t");
data_train_y <- data_train$y
data_train$y <- as.factor(data_train$y)
```

3. Appliquer la méthode du Bayésien naïf.

```
# Bayesien Naif
naiv_bayes <- naiveBayes(y~., data = data_train)
```

4. Lire les données test et prédire la sortie.

```
# Lecture des données test
data_test <- read.table("../Data/synth_test.txt",header=T,sep="\t");
data_test_x <- data.frame(x1=data_test$x1,x2=data_test$x2)
# Prédiction sur les données test
naiv_bayes_predict <- predict(naiv_bayes, newdata=data_test_x, type="class")
```

5. Évaluer la méthode sur le jeu de données test.

```
# Comparaison des valeurs prédites et des valeurs observées
table(naiv_bayes_predict, data_test$y)
# Calcul du taux d'erreur
naiv_bayes_error_rate <- mean(naiv_bayes_predict != data_test$y)
cat("error rate using test data (Naive bayes) = ", naiv_bayes_error_rate)
```

6. Afficher la frontière de cette méthode.

```
gridx1 <- seq(from=min(data_train$x1), to=max(data_train$x1), length.out=50)
gridx2 <- seq(from=min(data_train$x2), to=max(data_train$x2), length.out=50)
grid <- expand.grid(x1 = gridx1, x2 = gridx2)
data_grid_x <- data.frame(x1=grid[,1],x2=grid[,2])
naiv_bayes_predict_grid <- predict(naiv_bayes, newdata=data_grid_x, type="class")
plot(data_train$x1,data_train$x2,col=data_train_y,pch=16)
par(new=T)
plot(data_grid_x$x1,data_grid_x$x2,col=naiv_bayes_predict_grid,pch=8,cex=0.5,
                                                              ann=FALSE)
```

Régression logistique

1. Créer un fichier *classification-reglogistique.R* et copier les lignes suivantes.

```
# Adresse du dossier où vous travaillez
setwd("/Users/.../TP/Code")
# Supprimer toutes les variables
rm(list=ls(all=TRUE))
# Supprimer tous les graphiques déjà  présents
graphics.off()
```

2. Lire les données d'apprentissage.

```
# Lecture des données d'apprentissage
data_train <- read.table("../Data/synth_train.txt",header=T,sep="\t");
data_train_y <- data_train$y
data_train$y <- as.factor(data_train$y)
```

3. Appliquer la méthode de régression logistique sur les données d'apprentissage.

```
# Régression logistique
glm_train <- glm(y~., data = data_train, family=binomial())
```

4. Lire les données test et prédire la sortie.

```
# Lecture des données test
data_test <- read.table("../Data/synth_test.txt",header=T,sep="\t");
data_test_x <- data.frame(x1=data_test$x1,x2=data_test$x2)
# Prédiction sur les données test
glm_train_predict <- predict(glm_train, newdata=data_test_x, type="response")
result_glm_train_predict <- (glm_train_predict > 0.5)+1
```

5. Évaluer la méthode sur le jeu de données test.

```
# Comparaison des valeurs prédites et des valeurs observées
table(result_glm_train_predict, data_test$y)
# Calcul du taux d'erreur
glm_error_rate <- mean(result_glm_train_predict != data_test$y)
cat("error rate using test data (Logistic regression) = ", glm_error_rate)
```

6. Afficher la frontière de cette méthode.

```
# Frontières LDA et QDA
gridx1 <- seq(from=min(data_train$x1), to=max(data_train$x1), length.out=50)
gridx2 <- seq(from=min(data_train$x2), to=max(data_train$x2), length.out=50)
grid <- expand.grid(x1 = gridx1, x2 = gridx2)
data_grid_x <- data.frame(x1=grid[,1],x2=grid[,2])
glm_train_predict_grid <- predict(glm_train,newdata=data_grid_x,type="response")
result_glm_train_predict_grid <- (glm_train_predict_grid > 0.5)+1
plot(data_train$x1,data_train$x2,col=data_train_y,pch=16)
par(new=T)
plot(data_grid_x$x1,data_grid_x$x2,col=result_glm_train_predict_grid,pch=8,
                                                    cex=0.5,ann=FALSE)
```