Data Mining - TP classification

Julien JACQUES

14 avril, 2023

Données MNIST

Chargeons les images

```
#source("mnist.R")
load('MNIST.Rdata')
show_digit <- function(x, col=gray(12:1/12), title='image') {
image(matrix(x, nrow=28)[,28:1], col=col, main=title)
}
```

On extrait 5000 images d'apprentissage et 1000 de test

```
index=1:6000
app_x=train$x[index[1:5000],]
app_y=train$y[index[1:5000]]
test_x=train$x[index[5001:6000],]
test_y=train$y[index[5001:6000]]
```

K-plus proches voisins

On réalise un kNN avec k=10

```
library(class)
res=knn(app_x,test_x,app_y,k=10)
```

Comparons les prédictions obtenues pour l'échantillon test aux vraies étiquettes

```
table(res,test_y)
```

```
##    test_y
## res   0   1   2   3   4   5   6   7   8   9
##   0 110   0   2   1   0   0   0   0   1   1
##   1   0 106   6   2   4   3   0   4   2   0
##   2   0   0  78   0   0   0   0   0   1   0
##   3   0   0   3 109   0   1   0   0   2   2
##   4   1   0   0   0  80   1   1   0   0   2
##   5   1   0   0   3   0  73   1   0   1   0
##   6   1   0   1   0   1   0 105   0   3   0
##   7   0   1   3   0   0   0   0  90   1   2
##   8   0   0   0   0   0   0   0   0  78   0
##   9   0   1   0   0   3   2   0   7   0  99
```

```
cat('taux de bons classements :',mean(res==test_y))
```

```
## taux de bons classements : 0.928
```

arbre de décision

Effectuons maintenant la prédiction à l'aide d'un arbre de décision. Pour cela nous commençons par charger les packages nécessaires,

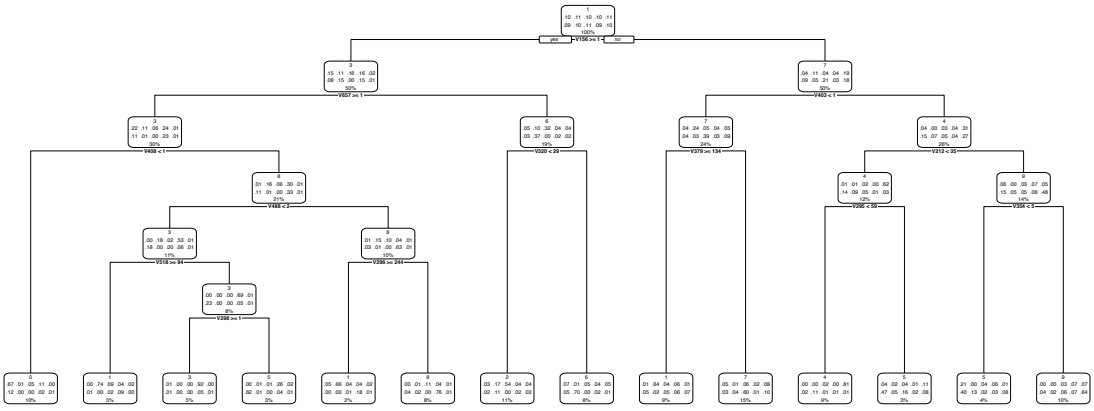
```
library('rpart')
library('rpart.plot')
```

puis à créer deux data.frame, contenant les données d'apprentissage et de test, en renomant la variable contenant les étiquettes afin de bien l'identifier, et en la définissant comme un facteur

```
app=data.frame(cbind(app_x,app_y))
names(app)[785]="y"
app$y=as.factor(app$y)
test=data.frame(cbind(test_x,test_y))
names(test)[785]="y"
test$y=as.factor(test$y)
```

On peut maintenant estimer l'arbre de décision

```
arbre=rpart(y~.,data=app)
rpart.plot(arbre)
```



On constate que l'arbre est très petit, très peu de pixels sont utilisés

Effectuons maintenant la prédiction sur l'echantillon test, et comparons la partition estimée à la partition réelle

```
res=predict(arbre,test,type="class")
table(res,test$y)
```

```
##
## res   0   1   2   3   4   5   6   7   8   9
##   0 80   1   6 19   0   6   0   0   0   2
##   1   0 83   6   8   2   2   1   4 12   5
##   2   3 19 43   6   0   1   9   0   3   3
##   3   2   1   3 47   0   1   0   0   6   0
##   4   0   0   1   0 52   3   9   0   0   3
##   5 14   0   2 14 16 53   7   9   2   3
##   6   2   1 10   6   5   5 73   0   4   0
##   7   9   1   5   4   8   1   7 77   3 25
##   8   2   0 13   2   1   1   1   0 49   0
##   9   1   2   4   9   4   7   0 11 10 65
```

```
cat('taux de bons classements :',mean(res==test_y))
```

```
## taux de bons classements : 0.622
```

Par contre cela ne fonctionne pas très bien, mais il est connu qu'un seul arbre a tendance a sur-apprendre l'échantillon d'apprentissage et à ne pas être très bon en prédiciton.

Forêts aléatoires

On va désormais réaliser la prédiction avec une forêt aléatoire de 200 arbres, effectuer la prédiction sur l'échantillon test et comparer les partitions réelles et estimées.

```
library(randomForest)
model <- randomForest(y~.,data=app,ntree=200)
pred<-predict(model,newdata=test,type="class")
table(pred,test$y)
```

```
##
## pred   0   1   2   3   4   5   6   7   8   9
##    0 110   0   2   0   0   0   0   0   1   1
##    1   0 104   0   1   0   0   0   0   0   1
##    2   0   1  84   0   0   0   0   0   1   1
##    3   0   0   3 110   0   1   0   0   1   2
##    4   1   1   1   0  84   1   1   1   0   4
##    5   0   0   0   2   0  76   1   0   1   0
##    6   0   0   1   0   2   1 104   0   4   0
##    7   0   1   1   0   0   0   0  92   1   0
##    8   2   0   1   2   0   0   1   0  80   0
##    9   0   1   0   0   2   1   0   8   0  97
```

```
cat('taux de bons classements :',mean(pred==test_y))
```

```
## taux de bons classements : 0.941
```

Les forêts aléatoires fonctionnent très bien.

On peut chercher à sélectionner les variables importantes en prenant les meilleurs suivants l'indicateur d'importance.

```
tmp=varImpPlot(model)
```