Université Grenoble Alpes
Data Institute

TP : Prise en main de R - Classification

Objectifs : découvrir la base du logiciel R et des algorithmes simples de classification.

1 Premiers pas avec R et l'environnement R Studio :

RStudio est un logiciel libre et collaboratif de statistique. Nous utiliserons dans les TP l'interface Rstudio de ce logiciel, que l'on peut télécharger gratuitement depuis le site

`www.rstudio.com`,

en ayant au préalable installé le logiciel R lui même, disponible sur le site

`www.r-project.org`.

Attention, dans RStudio, une majuscule et une minuscule n'ont pas le même sens.

RStudio est séparé en 4 fenêtres graphiques :

- la console (en bas à gauche) : permet d'executer les instructions avec la touche Entrée ↩
- le script (en haut à gauche) : permet d'écrire l'ensemble des commandes (ou instructions) que l'on veut exécuter et de pouvoir les sauvegarder dans un fichier. Une instruction s'exécute dans la fenêtre en bas à gauche en appuyant sur Ctrl+Entrée. Pour ne pas perdre de temps, vous pouvez copier-coller les commandes qui sont données dans les énoncés de TP dans votre script.
- la fenêtre Files/Plots/Packages/Help (en bas à droite) : permet entre autres de visualiser les graphiques ou l'aide.
- la fenêtre Workspace/History (en haut à droite) : permet de voir l'ensemble des objets en mémoire et historique de toutes les instructions réalisées.

Avant de démarrer :

1. Créer un répertoire "TP_R" dans votre dossier "documents".
2. Pour se simplifier la vie, nous allons travailler dans R studio dans le répertoire de travail TP_R. Pour cela, aller dans l'onglet session de la barre supérieure du menu, choisir Set Working Directory (Répertoire de Travail) et sélectionner le répertoire TP_R nouvellement créé.

Exercice 1

Dans R, une fonction s'écrit toujours avec des parenthèses, dans lesquelles les paramètres de la fonction sont précisés. Pour connaitre l'utilité et les paramètres d'une fonction, vous pouvez vous reporter à l'aide (cf. question 7).

1. Dans le script, créer la suite de données $(1, 2, 3, 4, 5)$ avec l'instruction :
   `c(1,2,3,4,5)`
   `c(.)` est la concaténation qui "range" les valeurs ou caractères tapés les uns à la suite des autres dans un vecteur.
   Exécuter la ligne en cliquant sur "Run" ou avec Ctrl+R.
2. Le précédent vecteur n'a pas de nom. Donner lui un nom :

   ```
   x<-c(1,2,3,4,5)
   ```

   La flèche permet d'assigner des valeurs à l'objet x créé. RStudio ne retourne rien. Pour vérifier que le vecteur x contient les valeurs, exécuter l'instruction
   `x`
   Prenez l'habitude de vérifier les objets que vous créez, utilisez.
   A la place de `<-`, on peut aussi utiliser `=`.
3. Créer le vecteur y contenant les valeurs $(2, 4, 6, 8, 10)$.
4. Prenez l'habitude de sauvegarder votre script au fur et à mesure ! (cliquer sur le script et appuyer sur Ctrl+S).
5. Vérifier que les vecteurs x et y ont la même longueur (le même nombre de valeurs)
   `length(x)`
   `length(y)`
6. Tracer sur un graphique les points définis par les deux vecteurs (x, y) :
   `plot(x,y)`
7. Personnaliser votre graphique :
   `plot(x,y, type = "p", pch = 3) # change les symboles`
   `plot(x,y, type = "b") # ajoute une ligne`
   `plot(x,y, col = "red") # change la couleur`
   On peut aussi rajouter des titres
   `plot(x,y,main="y selon x", type="p",xlab="abscisse",ylab="ordonnée") # ajoute un titre (paramètre main) et des légendes sur chaque axe (parametres xlab et ylab)`
   Créer votre propre graphique en changeant les couleurs, les symboles, les titres.
   Toutes les fonctions sont décrites dans l'aide. Par exemple pour plot, taper :
   `help(plot)`
   ou
   `?plot`

Exercice 2

1. Opérations basiques : comprendre les opérations suivantes
   `x/5`
   `x+5`
   `sum(x)`
   `cumsum(x)`
   `sqrt(x)`
   `x ^ 3`
2. Rajouter des valeurs à la suite du vecteur x
   `c(x,6)`
   Cette commande ne change pas x puisqu'on n'a pas utilisé la flèche ou `x`. Pour changer x, faire :

   ```
   x<-c(x,6)
   x
   ```

Pour reprendre les valeurs d'origine de x (dont on se sert ensuite)

```
x<-c(1,2,3,4,5)
z<- c(x,1,1,1,1,1)
c(x,rep(1,5))
c(x,seq(from=1, to=10, by=2))
c(x, 6:15)
```

Créer la suite de valeurs $(1, 4, 7, 10, \ldots, 61, 2, 2, 2, \ldots, 2, 1, 2, 3, \ldots, 20)$ où le nombre 2 a été répété 20 fois au milieu de la suite.

3. Dans R, on peut faire des tests logiques sur les éléments d'un vecteur. Comprendre les instructions suivantes :

```
(y>4)
(y!=4)
y==4)
(y>4)&(y<=6)
```

4. Dans R, les crochets permettent d'aller chercher des éléments d'un vecteur. Par exemple, pour extraire les deuxième et quatrième valeurs de y :

```
y[c(2,4)]
```

Comprendre les instructions suivantes :

```
y[1:4]
y[(y>4)]
```

Extraire les valeurs de y plus grandes strictement que 2 et plus petites ou égales à 8.

Extraire les valeurs de z différentes de 1.

Extraire les valeurs de x égales à 2.

5. Comprendre les opérations de base avec deux vecteurs :

```
x+y
x*y
x/y
```

6. Créer une table (ou une matrice) avec les deux vecteurs x et y

```
cbind(x,y) # matrice avec 5 lignes et 2 colonnes (cbind permet de coller des colonnes
les unes à la suite des autres)
rbind(x,y) # matrice avec 2 lignes et 5 colonnes (rbind permet de coller des lignes
les unes à la suite des autres)
```

7. De la même manière que pour les vecteurs, les crochets permettent d'aller chercher des éléments d'une matrice ou d'un tableau. Il y a alors 2 paramètres à définir, le premier pour les lignes et le second pour les colonnes à sélectionner (les 2 séparés par une virgule). Si rien n'est préciser pour le premier paramètre, cela signifie que l'on prend toutes les lignes. Si rien n'est préciser pour le deuxième paramètre, cela signifie que l'on prend toutes les colonnes.

```
M<-rbind(x,y,y,x) # matrice avec 4 lignes et 5 colonnes
M # toute la matrice
M[3,2] # élément de la 3ieme ligne 2ieme colonne
M[ ,1] # élément de la premiere colonne
```

Extraire les élements de la deuxieme et troisieme lignes.

Extraire les éléments de la première et troisième colonnes.

2 Classification

Exercice 3

On travaille sur la base de données "Spam" qui continent 4601 emails, qui ont été classés manuellement comme Spam ou Non-Spam.
Cette base contient également pour chaque email 57 caractéristiques (fréquence des mots "money", "free", "credit", etc). L'objectif est de developper un algorithme de classification automatique, qui, à partir des 57 caractéristiques, pourra décider si un nouveau mail doit être classé comme Spam ou comme Non-Spam.

1. La base est téléchargeable ici : `https://archive.ics.uci.edu/ml/datasets/spambase`
   Enregistrer les fichiers dans votre répertoire de travail.
2. Pour l'importer dans RStudio,
   ```
   spambase<-read.csv("spambase.data")
   names(spambase)<-c(read.table("spambase.names", skip=33, sep=":", stringsAsFactors
   = FALSE)[[1]], "spam")
   ```
3. Afficher les premières lignes de la base
   ```
   head(spambase)
   ```
4. Représenter graphiquement la base de données, par exemple avec un graphique de ce type :
   ```
   library(ggplot2)
   ggplot(spambase, aes(x = char_freq_dollar, y = word_freq_hp)) +
   geom_point(aes(colour=spam, shape=spam), size = 3) +
   xlab("frequence of dollar") +
   ylab("frequence of hp") +
   ggtitle("Spam vs frequences of free and hp")
   ```

5. On va appliquer une regression logistique pour prédire la probabilité qu'un email soit un spam. L'algorithme qui permet de calculer les coefficients du modele logistique est programmé dans la fonction `glm` sous R. On rentre la variable à expliquer (`spam`) en fonction de variables explicatives.

   (a) Avec deux variables explicatives, la syntaxe est
   ```
   res<-glm(as.factor(spam) word_freq_make+word_freq_address,data=spambase, family="binomial")
   summary(res)
   ```

   (b) Quand utilise l'ensemble des variables pour prédire la variable à expliquer, la syntaxe est
   ```
   resC<-glm(as.factor(spam) .,data=spambase, family="binomial")
   summary(resC)
   ```

   (c) On applique ensuite une méthode de sélection de variables automatique pour ne garder que celles qui sont pertinentes :
   ```
   final<-step(resC)
   summary(final)
   ```

   Les variables sélectionnées peuvent être interprêtées par l'expert.

6. On peut aussi appliquer un algorithme de perceptron pour réaliser la classification.

   (a) Télécharger le code du perceptron et l'enregistrer dans votre répertoire. Il faut ensuite le "sourcer" sous R :
   ```
   source("perceptron.R")
   ```

   (b) Pour appliquer le perceptron, on sépare la variable à expliquer (qui doit etre rentrée de facon numérique) et la matrice des variables explicatives.

```
x<-spambase[,1:57]
y<-as.numeric(spambase[,58])
```

(c) On peut ensuite appliquer le perceptron et evaluer le taux d'erreur en fonction du nombre d'iterations.

```
err <- perceptron(x, y)
```