Fiche TD avec le logiciel R : tdr19

Algèbre relationnelle sous R

Pr Jean R. Lobry

Ceci n'est qu'un mémo pour les algébristes relationnels qui voudraient retrouver rapidement leurs petits sous R et procéder à des manipulations élémentaires directement sous R sans garantie aucune d'efficacité pour ce qui est du temps d'exécution, mais d'un confort humain certain dans le cadre des petits jeux de données.

Table des matières

1 Introduction

ATTENTION, R n'est pas un système de gestion de bases de données (SGBD), même s'il est capable de manipuler des jeux de données de taille respectable [1]. Pour des applications de forte volumétrie il faut envisager un couplage avec un véritable SGBD, il existe des paquets R qui fournissent une interface avec des SGBD classiques (*e.g.* `RMySQL`, `ROracle`) ou spécialisés [2]. Cette fiche n'est qu'un mémo pour ceux qui sont habitués aux notations de l'algèbre relationnelle pour faire des manipulations élémentaires sous R.

2 Relations

UN tuple est représenté par une ligne d'un `data.frame`. Une relation est représentée par un `data.frame` sans ligne dupliquée. Nous en construisons un ici à partir d'un jeu de données standard :

```
data(iris)
is.data.frame(iris)
```

```
[1] TRUE
```

```
any(duplicated(iris))
```

```
[1] TRUE
```

```
iris <- iris[!duplicated(iris), ]
any(duplicated(iris))
```

```
[1] FALSE
```

```
iris[1:5, ]
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

3 Sélection $\sigma_{\mathrm{p}}(\mathrm{R})$: `subset(R, p)`

3.1 $\sigma_{\mathrm{Species}=\mathrm{setosa}}(\mathrm{iris})$

On ne liste ici que les 5 premiers tuples :

```
subset(iris, Species == "setosa")[1:5, ]
```

```
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

1. Voir la fiche tdr16 *Installer des données volumineuses : les génomes bactériens*
2. Pour une interface avec les banques de séquences structurées sous ACNUC, voir la bibliothèque `seqinr`.

3.2 $\sigma_{\text{Sepal.Length}=7}(\text{iris})$

Danger :

```
x1 <- 0.5 - 0.3
x2 <- 0.3 - 0.1
x1 == x2                            # FALSE on most machines
```

```
[1] FALSE
```

```
identical(all.equal(x1, x2), TRUE) # TRUE everywhere
```

```
[1] TRUE
```

Donc, ne jamais utiliser l'opérateur `==` pour tester l'égalité de valeurs numériques. La fonction `all.equal()` permet de faire des comparaisons raisonnables par rapport au `.Machine$double.eps`[3]. La fonction `identical()` permet de tester l'égalité exacte entre deux objets, elle est *indispensable* parce que `NA` est une valeur logique admise dans R (logique ternaire).

```
subset(iris, sapply(Sepal.Length, function(x) identical(all.equal(x, 7), TRUE)))
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
51            7         3.2          4.7         1.4 versicolor
```

On peut définir son propre opérateur binaire infixe vectorisé :

```
"%=%" <- function(x,y) sapply(x, function(x) identical(all.equal(x, y), TRUE))
```

pour alléger les notations :

```
subset(iris, Sepal.Length %=% 7)
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
51            7         3.2          4.7         1.4 versicolor
```

3.3 $\sigma_{\text{Sepal.Length}>7.5}(\text{iris})$

```
subset(iris, Sepal.Length > 7.5)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
106          7.6         3.0          6.6         2.1 virginica
118          7.7         3.8          6.7         2.2 virginica
119          7.7         2.6          6.9         2.3 virginica
123          7.7         2.8          6.7         2.0 virginica
132          7.9         3.8          6.4         2.0 virginica
136          7.7         3.0          6.1         2.3 virginica
```

3.4 $\sigma_{\text{Species}\in\{\text{versicolor,virginica}\}}(\text{iris})$

On ne liste ici que les 5 premiers tuples :

```
subset(iris, Species %in% c("versicolor", "virginica"))[1:5, ]
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
51          7.0         3.2          4.7         1.4 versicolor
52          6.4         3.2          4.5         1.5 versicolor
53          6.9         3.1          4.9         1.5 versicolor
54          5.5         2.3          4.0         1.3 versicolor
55          6.5         2.8          4.6         1.5 versicolor
```

3. qui valait `2.22044604925031e-16` lors de la dernière compilation de ce document

3.5 $\sigma_{\text{Sepal.Length} > 5.5 \wedge \text{Species} = \text{setosa}}(\text{iris})$

```
subset(iris, Sepal.Length > 5.5 & Species == "setosa")
```

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
15          5.8         4.0          1.2         0.2  setosa
16          5.7         4.4          1.5         0.4  setosa
19          5.7         3.8          1.7         0.3  setosa
```

3.6 $\sigma_{\text{Sepal.Length} > 7.7 \vee \text{Sepal.Length} < 4.4}(\text{iris})$

```
subset(iris, Sepal.Length > 7.7 | Sepal.Length < 4.4)
```

```
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
14           4.3         3.0          1.1         0.1    setosa
132          7.9         3.8          6.4         2.0 virginica
```

4 Projection $\pi_{A_1,\ldots,A_n}(R)$: `proj(R, A)`

Je n'ai pas trouvé d'opérateur de projection (au sens de l'algèbre relationnelle) pré-défini dans R parce qu'il faut éliminer les tuples redondants. Ceci dit, c'est assez trivial de définir une fonction pour ça :

```
proj <- function(R, A)
{
  tmp <- R[ , A, drop = FALSE]
  return(tmp[ !duplicated(tmp), , drop = FALSE])
}
```

4.1 $\pi_{\text{Sepal.Length, Species}}(\text{iris})$

On ne liste ici que les 5 premiers tuples :

```
proj(iris, c("Sepal.Length","Species"))[1:5, ]
```

```
  Sepal.Length Species
1          5.1  setosa
2          4.9  setosa
3          4.7  setosa
4          4.6  setosa
5          5.0  setosa
```

On peut également utiliser le point de vue non étiqueté de l'algèbre relationnelle, ce qui est souvent assez commode en pratique. On ne liste ici que les 3 premiers tuples :

```
proj(iris, c(1,5))[1:3, ]
```

```
  Sepal.Length Species
1          5.1  setosa
2          4.9  setosa
3          4.7  setosa
```

4.2 $\pi_{\neg(\text{Sepal.Length, Species})}(\text{iris})$

Pour les relations avec beaucoup d'attributs, il est plus rapide d'exclure les attributs que d'inclure explicitement ceux que l'on veut retenir. C'est assez direct dans le point de vue non étiqueté (on ne liste ici que les 5 premiers tuples) :

```
proj(iris, -c(1,5))[1:5, ]
  Sepal.Width Petal.Length Petal.Width
1         3.5          1.4         0.2
2         3.0          1.4         0.2
3         3.2          1.3         0.2
4         3.1          1.5         0.2
5         3.6          1.4         0.2
```

Dans le mode étiqueté, il faut ruser un peu pour récupérer automatiquement le rang des attributs (on ne liste ici que les 7 premiers tuples) :

```
proj(iris, -sapply(c("Sepal.Length","Species"), function(x) which(x == names(iris))))[1:7, ]
  Sepal.Width Petal.Length Petal.Width
1         3.5          1.4         0.2
2         3.0          1.4         0.2
3         3.2          1.3         0.2
4         3.1          1.5         0.2
5         3.6          1.4         0.2
6         3.9          1.7         0.4
7         3.4          1.4         0.3
```

4.3 $\pi_{\text{f(iris)}}(\text{iris})$

La différence par rapport à un SGBD classique est peut-être que le choix des attributs peut lui même être le résultat d'un calcul statistique complexe. Dans cet exemple simple, on ne garde que l'attribut de moyenne minimale, mais on pourrait envisager quelque chose de bien plus élaboré :

```
proj(iris, which.min(colMeans(iris[,1:4])))
    Petal.Width
1           0.2
6           0.4
7           0.3
10          0.1
24          0.5
44          0.6
51          1.4
52          1.5
54          1.3
57          1.6
58          1.0
70          1.1
71          1.8
74          1.2
78          1.7
101         2.5
102         1.9
103         2.1
105         2.2
111         2.0
115         2.4
116         2.3
```

5 Renommage $\delta_{\text{A}_1\text{A}_2\ldots\text{A}_n \to \text{B}_1\text{B}_2\ldots\text{B}_n}(\text{R})$: `names<-`

5.1 $\delta_{\text{Species} \to \text{Especes}}(\text{iris})$

```
names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
names(iris)[which(names(iris) =="Species")] <- "Especes"
names(iris)
```

```
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Especes"
```

6 Produit cartésien $R_1 \times R_2$: `merge()`

Le produit cartésien est pré-défini dans R avec la fonction `merge()` quand son argument `by` est de longueur nulle :

```
data.frame(matrix(letters[1:6],3,2))->R1
names(R1)<-LETTERS[1:2]
R1
```

```
  A B
1 a d
2 b e
3 c f
```

```
data.frame(matrix(letters[(1:9)+2],3,3))->R2
names(R2)<-LETTERS[2:4]
R2
```

```
  B C D
1 c f i
2 d g j
3 e h k
```

```
merge(R1,R2, by = character(0))
```

```
  A B.x B.y C D
1 a   d   c f i
2 b   e   c f i
3 c   f   c f i
4 a   d   d g j
5 b   e   d g j
6 c   f   d g j
7 a   d   e h k
8 b   e   e h k
9 c   f   e h k
```

Ce n'est plus une véritable relation, il faudrait faire un renommage pour ne pas avoir d'attributs de même nom, c'est facile à imposer en utilisant le constructeur des `data.frame` :

```
data.frame(merge(R1,R2, by = character(0)))
```

```
  A B.x B.y C D
1 a   d   c f i
2 b   e   c f i
3 c   f   c f i
4 a   d   d g j
5 b   e   d g j
6 c   f   d g j
7 a   d   e h k
8 b   e   e h k
9 c   f   e h k
```

7 Jointure (naturelle) $R_1 \bowtie R_2$: `merge()`

Par défaut, la fonction `merge()` fait une jointure sur les attributs communs aux deux relations :

```
merge(R1, R2)
```

```
  B A C D
1 d a g j
2 e b h k
```

Si on a la chance d'avoir une clef d'identification commune aux deux relations on peut l'utiliser ainsi :

```
merge(R1, R2, by.x = "B", by.y = "B")
```

```
  B A C D
1 d a g j
2 e b h k
```

```
merge(R1, R2, by.x = "A", by.y = "B")
```

```
  A B C D
1 c f f i
```

8 Exercices

8.1 Auteurs de livres de statistiques

On considère la relation `livre` donnée dans la table 1 et la relation `auteur` donnée dans la table 2.

	auteur	titre	coauteur
1	Tukey	Exploratory Data Analysis	
2	Venables	Modern Applied Statistics	Ripley
3	Tierney	LISP-STAT	
4	Ripley	Spatial Statistics	
5	Ripley	Stochastic Simulation	
6	McNeil	Interactive Data Analysis	
7	R Core	An Introduction to R	Venables

TABLE 1 – La relation livre

	nom	nationalite	decede
1	Tukey	USA	oui
2	Venables	Australie	non
3	Tierney	USA	non
4	Ripley	UK	non
5	McNeil	Australie	non

TABLE 2 – La relation auteur

1. Entrez les relations sous la forme d'un `data.frame` et vérifiez qu'il s'agit bien de relations.

```
auteur <- data.frame(
    nom = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
    nationalite = c("USA", "Australie", "USA", "UK", "Australie"),
    decede = c("oui", rep("non", 4)))
livre <- data.frame(
    auteur = c("Tukey", "Venables", "Tierney",
               "Ripley", "Ripley", "McNeil", "R Core"),
    titre = c("Exploratory Data Analysis",
              "Modern Applied Statistics",
              "LISP-STAT",
              "Spatial Statistics", "Stochastic Simulation",
              "Interactive Data Analysis",
              "An Introduction to R"),
    coauteur = c(NA, "Ripley", NA, NA, NA, NA,
                 "Venables"))
any(duplicated(livre))
```

```
[1] FALSE
```

```
any(duplicated(auteur))
```

```
[1] FALSE
```

2. Quels sont les titres des livres dont l'auteur est Ripley ?

```
proj(subset(livre, auteur == "Ripley"), "titre")
```

```
                titre
4    Spatial Statistics
5 Stochastic Simulation
```

3. Quels sont les auteurs et titres des livres dont l'auteur ou le coauteur est Ripley ?

```
proj(subset(livre, auteur == "Ripley" | coauteur == "Ripley"), c("auteur","titre"))
```

```
   auteur                    titre
2 Venables Modern Applied Statistics
4   Ripley        Spatial Statistics
5   Ripley     Stochastic Simulation
```

4. Quels sont les auteurs qui sont également coauteurs ?

```
proj(subset(livre, auteur %in% coauteur), "auteur")
```

```
   auteur
2 Venables
4   Ripley
```

5. Quel est l'auteur qui a publié le plus de livres ?

```
proj( subset(livre, auteur == names(which.max(table(auteur)))), "auteur")
```

```
  auteur
4 Ripley
```

6. Quel est le nom et la nationalité de l'auteur dont la nationalité est la moins bien représentée ?

```
proj( subset(auteur, nationalite == names(which.min(table(nationalite)))), c("nom","nationalite"))
```

```
     nom nationalite
4 Ripley          UK
```

7. Quel est le nom des auteurs australiens encore vivants ?

```
proj( subset(auteur, nationalite == "Australie" & decede == "non"), "nom")
```

```
       nom
2 Venables
5   McNeil
```

8. Quelles sont les nationalités représentées ?

```
proj(auteur, "nationalite")
```

```
  nationalite
1         USA
2   Australie
4          UK
```

9. Quels sont les titres des ouvrages dont l'auteur est de nationalité australienne ?

```
M <- merge(livre, auteur, by.x = "auteur", by.y = "nom")
proj(subset(M, nationalite == "Australie"), "titre")
```

```
                    titre
1 Interactive Data Analysis
6 Modern Applied Statistics
```

10. Quels sont les titres des ouvrages dont l'auteur est de nationalité américaine et encore vivant ?

```
M <- merge(livre, auteur, by.x = "auteur", by.y = "nom")
proj(subset(M, nationalite == "USA" & decede == "non"), "titre")
```

```
      titre
4 LISP-STAT
```