Introduzione a R

Corso di Bioinformatica

Nicola Vitacolonna
Corso di Laurea in Biotecnologie

Perché R?

- Gratuito, open-source, multipiattaforma
- Potente (oltre 5200 pacchetti (http://cran.r-project.org/web/packages/))
- Flessibile
- Ottime capacità grafiche
- Ampiamente adottato (http://www.revolutionanalytics.com/what-is-open-source-r/companies-using-r.php) sia in ambito accademico sia nel mondo aziendale
- Analisi riproducibili

Perché non un foglio elettronico (leggi: Excel)?

- Analisi difficilmente riproducibili
  - Un foglio elettronico riporta dati, formule e risultati, ma non il procedimento seguito
- Visualizzazione dei dati non ottimale
- Difficile o impossibile operare con grandi quantità di dati
- Computazioni statistiche a volte non corrette (e comunque non è possibile ispezionare il codice sorgente)
- Di gran lunga meno ricco di funzionalità di R

L'importanza della riproducibilità

Nature Genetics **41**, 149 - 155 (2009)
Published online: 28 January 2008 | doi:10.1038/ng.295

Repeatability of published microarray gene expression analyses

See associated Correspondence: Baggerly, *Nature* **467**, 401 (September 2010)

John P A Ioannidis[1,2,3], David B Allison[4], Catherine A Ball[5], Issa Coulibaly[4], Xiangqin Cui[4], Aedín C Culhane[6,7], Mario Falchi[8,9], Cesare Furlanello[10], Laurence Game[11], Giuseppe Jurman[10], Jon Mangion[11], Tapan Mehta[4], Michael Nitzberg[5], Grier P Page[4,12], Enrico Petretto[11,13] & Vera van Noort[14]

Given the complexity of microarray-based gene expression studies, guidelines encourage transparent design and public data availability. Several journals require public data deposition and several public databases exist. However, not all data are publicly available, and even when available, it is unknown whether the published results are reproducible by independent scientists. Here we evaluated the replication of data analyses in 18 articles on microarray-based gene expression profiling published in *Nature Genetics* in 2005–2006. One table or figure from each article was independently evaluated by two teams of analysts. We reproduced two analyses in principle and six partially or with some discrepancies; ten could not be reproduced. The main reason for failure to reproduce was data unavailability, and discrepancies were mostly due to incomplete data annotation or specification of data processing and analysis. Repeatability of published microarray studies is apparently limited. More strict publication rules enforcing public data availability and explicit description of data processing and analysis should be considered.

http://www.nature.com/ng/journal/v41/n2/abs/ng.295.html
(http://www.nature.com/ng/journal/v41/n2/abs/ng.295.html)

Inserire i comandi

R ha un **prompt** interattivo: l'utente digita un'**espressione** che il sistema valuta quando si preme Invio. Il risultato in certi casi è stampato sullo schermo:

```
x <- 5  # non stampa niente
x  # stampa il valore corrente di x
```

```
## [1] 5
```

- `[1]` indica che il risultato è un vettore e `5` è il primo elemento del vettore
- # segnala un commento: tutto ciò che è scritto da # alla fine della riga è ignorato
- `x` è una **variabile**, cioè un nome assegnato a un oggetto. I nomi di variabile sono arbitrari, ma devono consistere solo di lettere, numeri, punti e caratteri di sottolineatura (*underscore*), e devono cominciare con una lettera. *Gli spazi sono vietati nei nomi di variabile!*
- Lettere maiuscole e minuscole *sono simboli distinti* (R è “case-sensitive”)

Convenzioni per i nomi di variabile

I nomi delle variabili devono essere brevi, ma descrittivi. Si raccomanda di usare uno degli stili seguenti, in modo consistente:

Camel caps

```
altezzaPersonaCM <- 188
```

Underscore

```
altezza_persona_cm <- 188
```

Dot separated

```
altezza.persona.cm <- 188
```

Ottenere aiuto

- R ha una documentazione online estesa
- **È fondamentale** saper richiedere e interpretare la documentazione di R
- Per cercare informazioni su un termine preciso usare `help()` oppure `?`
- Per cercare informazioni su un argomento specificando un termine potenzialmente approssimato usare `help.search()` oppure `??`

```
help("getwd")
?"getwd"; # come help()
help.search("titani")
??"titaic" # come help.search()
```

Oggetti

R tratta con cinque **classi** elementari di oggetti (**tipi di dato**):

1. caratteri (stringhe): `character`
2. numeri interi: `integer`
3. numeri reali (piú precisamente, “in virgola mobile” o *floating-point*): `numeric`
4. numeri complessi: `complex`
5. valori logici (vero/falso): `logical`

La struttura dati fondamentale è il **vettore**, che è una sequenza ordinata di oggetti dello stesso tipo

- Ad esempio, vettore di interi, vettore di caratteri, etc…

Manipolazione di oggetti

È possibile assegnare un oggetto a una variabile con l'operatore di **assegnamento** `<-`:

```
x <- 12345
```

Se si scrive il nome di una variabile e si preme il tasto Invio, viene stampato il valore della variabile

```
x
```

```
## [1] 12345
```

È possibile vedere la classe di un oggetto con la funzione `class()`:

```
class(x)
```

```
## [1] "numeric"
```

Caratteri (o stringhe)

- Racchiusi tra virgolette

```
frase <- "Mi piace la bioinformatica!"
frase
```

```
## [1] "Mi piace la bioinformatica!"
```

```
class(frase)
```

```
## [1] "character"
```

Numeri

```
y <- 1.6
class(y)
```

```
## [1] "numeric"
```

- R può operare come una calcolatrice

```
((1 + 4 * 3 - 2)/3)^2
```

```
## [1] 13.44
```

- `*` è la moltiplicazione, `/` è la divisione, `^` o `**` è l'elevamento a potenza
- Le parentesi tonde sono usate per raggruppare sotto-espressioni

Numeri

Due quantità speciali sono `Inf` (infinito) e `Nan` (*Not a Number*)

```
1/0
```

```
## [1] Inf
```

```
-1/0
```

```
## [1] -Inf
```

```
0/0
```

```
## [1] NaN
```

Numeri interi

In R, i numeri sono tipicamente trattati come numeri reali (classe `numeric`). Si può forzare l'interpretazione di un numero come intero giustapponendo `L` dopo il numero:

```
numeroDiFigli <- 2
class(numeroDiFigli)
```

```
## [1] "numeric"
```

```
numeroDiFigli <- 2L
class(numeroDiFigli)
```

```
## [1] "integer"
```

Valori logici (o booleani)

Le costanti di tipo logico sono `TRUE` (oppure `T`) e `FALSE` (oppure `F`)

```
esameSuperato <- TRUE
class(esameSuperato)
```

```
## [1] "logical"
```

```
esameSuperato
```

```
## [1] TRUE
```

Espressioni logiche

- Espressioni logiche elementari si costruiscono con gli **operatori di confronto**:
  - <, <=, >, >= per le disuguaglianze
  - == per l'uguaglianza e `!=` per la disuguaglianza
- Le espressioni logiche possono essere combinate mediante **connettivi logici**:
  - `&` per la congiunzione logica, | per la disgiunzione, `!` per la negazione

```
3 != 1 + 2
```

```
## [1] FALSE
```

```
(3 == 1 + 2) & !(4 <= 2)
```

```
## [1] TRUE
```

Vettori

Un vettore si costruisce con la funzione `c()` (`c` sta per “concatenare”)

```
altezze <- c(188.6, 177.4, 180.2)
valoriLogici <- c(TRUE, FALSE)
colori <- c("rosso", "giallo", "blu")
```

La classe di un vettore è la classe degli elementi del vettore

```
class(altezze)
```

```
## [1] "numeric"
```

Importante: i vettori sono **sempre** sequenze di elementi omogenei (stessa classe)!

Altri modi di costruire un vettore

Sequenza di numeri interi consecutivi:

```
v <- 3:10
v
```

```
## [1]  3  4  5  6  7  8  9 10
```

Sequenza di numeri equi-spaziati (funzione `seq()`):

```
v <- seq(from = 1, to = 10, by = 0.5)  # Si può scrivere anche seq(1, 10, 0.5)
v
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0
```

Altri modi di costruire un vettore

Ripetizione di un elemento (funzione `rep()`)

```
v <- rep("R è bello", 4)
v
```

```
## [1] "R è bello" "R è bello" "R è bello" "R è bello"
```

```
v <- c(rep("yes", 5), rep("no", 3))
v
```

```
## [1] "yes" "yes" "yes" "yes" "yes" "no"  "no"  "no"
```

Mischiare oggetti di classe diversa in un vettore

Che succede se si prova a creare un vettore con oggetti di classe diversa?

```
c(1.7, "a")
```

```
## [1] "1.7" "a"
```

```
c(TRUE, 2)
```

```
## [1] 1 2
```

```
c("a", TRUE)
```

```
## [1] "a"    "TRUE"
```

R non produce un errore, ma **converte** automaticamente i valori in modo che abbiano tutti la stessa classe

Conversione esplicita

È possibile, e talvolta necessario, convertire un oggetto da una classe ad un'altra in modo esplicito con le funzioni `as.*`:

```
x <- 0:6
class(x)
```

```
## [1] "integer"
```

```
as.logical(x)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5" "6"
```

Conversioni prive di significato

Quando una conversione è priva di significato, viene assegnato il valore `NA`:

```
x <- c("a", "b", "c")
as.numeric(x)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA NA NA
```

```
as.logical(x)
```

```
## [1] NA NA NA
```

Altri tipi di dato

- Matrici
- Liste
- Factor (variabili categoriali)
- Data frame (matrici dei dati)

Matrici

- Si costruisce con la funzione `matrix()` specificando:
  1. un vettore contenente tutti gli elementi della matrice *ordinati per colonne*
  2. il numero di righe oppure il numero di colonne (o entrambi)

```
matrix(1:6, nrow = 2)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
matrix(1:6, ncol = 2)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

Liste

- Simili ai vettori, ma possono contenere elementi di tipo eterogeneo
- Si costruiscono con la funzione `list()`

```
x <- list(1.8, "a", TRUE, c(3, 7, 2))
x
```

```
## [[1]]
## [1] 1.8
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 3 7 2
```

Factor (vettori categoriali)

- Usati per rappresentare variabili categoriali (sconnesse o ordinali)
- Si costruiscono con la funzione `factor()`

```
x <- factor(c("yes", "no", "yes", "yes", "no"))
x
```

```
## [1] yes no  yes yes no
## Levels: no yes
```

```
class(x)
```

```
## [1] "factor"
```

Factor (vettori categoriali)

- I **livelli** di un vettore categoriale si ottengono con la funzione `levels()`
- La **tabella di contingenza** dei livelli si ottiene con `table()`

```
levels(x)
```

```
## [1] "no"  "yes"
```

```
table(x)
```

```
## x
##  no yes
##   2   3
```

Conversione in un vettore categoriale

Si può forzare l'interpretazione di una variabile come categoriale con la funzione `as.factor()` (è un caso particolare di conversione esplicita, vista nelle slide precedenti)

```
x <- c(1, 2, 1, 3, 2, 2)
class(x)
```

```
## [1] "numeric"
```

```
x <- as.factor(x)
class(x)
```

```
## [1] "factor"
```

Variabili categoriali ordinali

Si può specificare un ordinamento dei livelli passando alla funzione `factor()` l'argomento `ordered = TRUE` e specificando un vettore con i livelli ordinati nel modo desiderato, come segue:

```
x <- factor(c("alto", "medio", "basso"), ordered = TRUE, levels = c("basso",
    "medio", "alto"))
x
```

```
## [1] alto  medio basso
## Levels: basso < medio < alto
```

```
is.ordered(x)
```

```
## [1] TRUE
```

Valori nulli

- Un valore mancante o non specificato è denotato con `NA` (o `NaN` se si tratta del risultato indefinito di un'operazione matematica)
- Si può verificare la presenza di valori nulli con la funzione `is.na()` (ovvero `is.nan()` per `NaN`)
- Il risultato di `is.na()` è un vettore logico

```
x <- c(1, 2, NA, 10, 3)
is.na(x)
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

Attenzione: `NA` **non** non va racchiuso tra virgolette (non è una stringa)!

Data frame (matrice dei dati)

- L'equivalente in R di una tabella
- A differenza delle matrici, i data frame possono contenere dati eterogenei
- Sono liste speciali i cui elementi sono vettori della stessa lunghezza
- Ciascun elemento della lista è una colonna della tabella
- La lunghezza di ciascun elemento della lista è il numero di righe della tabella
- Tipicamente, un data frame è costruito leggendo una tabella da un file esterno con le funzioni `read.table()` o `read.csv()`, oppure è costruito direttamente con la funzione `data.frame()`, oppure si ottiene per conversione da un'altra classe con `as.data.frame()`

Data frame: esempio

- La funzione `head()` consente di stampare le prime righe di un data frame
- La funzione `dim()` stampa le dimensioni di un oggetto (si può usare anche con le matrici)

```
data(iris)  # Carica in memoria un data set predefinito
head(iris, 3)
```

```
##   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
```

```
dim(iris)  # 5 variabili, 150 osservazioni
```

```
## [1] 150   5
```

Costruzione diretta di un data frame

È possibile costruire un data frame a partire da vettori *della stessa lunghezza* con la funzione `data.frame()`:

```
df <- data.frame(A = c(18, 20, 19, 18), B = c(18, 20, 20, 15), C = c(17, 15,
    17, 18))
df
```

```
##    A  B  C
## 1 18 18 17
## 2 20 20 15
## 3 19 20 17
## 4 18 15 18
```

Etichettare i vettori

In R, gli oggetti possono essere etichettati con le funzioni `names()`, `colnames()`, `rownames()`, `row.names()` (http://stackoverflow.com/questions/2281353/row-names-column-names-in-r) per rendere la loro interpretazione piú chiara

Gli elementi di un vettore possono essere etichettati con `names()`:

```
x <- 1:3
names(x) <- c("uno", "due", "tre")
x
```

```
## uno due tre
##   1   2   3
```

```
names(x)
```

```
## [1] "uno" "due" "tre"
```

Etichettare matrici

Le matrici possono essere etichettate con `rownames()` e `colnames()`:

```
M <- matrix(1:6, nrow = 2)
rownames(M) <- c("r1", "r2")
colnames(M) <- c("c1", "c2", "c3")
M
```

```
##    c1 c2 c3
## r1  1  3  5
## r2  2  4  6
```

```
rownames(M)
```

```
## [1] "r1" "r2"
```

```
colnames(M)
```

Etichettare data frame

I data frame possono essere etichettati con `names()` (per le variabili) e `rownames()` (per le osservazioni):

```
data(iris)
names(iris) <- c("SL", "SW", "PL", "PW", "S")
rownames(iris) <- as.character(101:250)
head(iris)
```

```
##      SL  SW  PL  PW      S
## 101 5.1 3.5 1.4 0.2 setosa
## 102 4.9 3.0 1.4 0.2 setosa
## 103 4.7 3.2 1.3 0.2 setosa
## 104 4.6 3.1 1.5 0.2 setosa
## 105 5.0 3.6 1.4 0.2 setosa
## 106 5.4 3.9 1.7 0.4 setosa
```

Per approfondire

- R tutorial (http://biochemistry.utoronto.ca/undergraduates/courses/BCH441H/wiki/index.php/R_tutorial)
- Quick-R (http://www.statmethods.net)
- An Introduction to R (http://cran.r-project.org/doc/manuals/r-release/R-intro.html)
- Cercate con Google e condividete risorse su R nel forum di discussione!