

Probabilità

14 ottobre 2024

Indice

1	Calcolo combinatorio	1
2	Simulazione	3
2.1	Impostazione del seme	3
2.2	Generazione di numeri casuali ed estrazioni da un'urna	3
2.2.1	Variabili casuali e funzioni rilevanti	3
2.2.2	L'uso di sample	4
2.3	L'impiego di replicate	5
2.4	Applicazioni varie	5
2.4.1	Simulazione di dati per un modello di regressione	5

1 Calcolo combinatorio

Fattoriale In R il fattoriale è ottenuto mediante la funzione **factorial**; poi nei casi di fattoriali di numeri molto grandi si può utilizzare **lfactorial** che calcola il logaritmo naturale del fattoriale di un numero, ossia $\log(n!)$. Nell'ultimo esempio di sotto si ha che approssimativamente $1000000! = 10^{5565709}$

```
factorial(4)

## [1] 24

factorial(1000000)

## [1] Inf

lfactorial(1000000)

## [1] 12815518

# convertiamo in logaritmo base 10
lfactorial(1000000)/log(10)

## [1] 5565709
```

Coefficiente binomiale Il coefficiente binomiale è implementato in R, mediante `choose`, mentre `lchoose` ne calcola il logaritmo:

```
choose(4,2)

## [1] 6

choose(1e06, 500)

## [1] Inf

lchoose(1e06, 500)

## [1] 4296.3
```

Permutazioni Per determinare tutte le permutazioni di un vettore si può utilizzare `permn` dal pacchetto `combinat`:

```
combinat::permn(1:3)

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

Combinazioni Per la determinazione delle combinazioni degli elementi da un vettore `x` presi `tot` (nell'esempio 2) alla volta si usa `combn`:

```
combn(letters[1:4], 2)

##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] "a"  "a"  "a"  "b"  "b"  "c"
## [2,] "b"  "c"  "d"  "c"  "d"  "d"
```

Nome variabile	varcas	Tipo v.c.
Binomiale	binom	discreta
Chi-squared	chisq	continua
F	f	continua
Ipergeometrica	hyper	discreta
Normale	norm	continua
Poisson	pois	discreta
T di Student	t	continua
Uniforme	unif	continua

Tabella 1: Variabili casuali standard in R

2 Simulazione

2.1 Impostazione del seme

Ogni volta che si simulano variabili casuali, è necessario impostare il seme della generazione con `set.seed`, al fine di assicurare riproducibilità su altre macchine della generazione effettuata.

Ogni volta che si reimposta il seme della generazione, la simulazione “riparte” ottenendo gli stessi risultati, come mostra il seguente esempio

```
set.seed(1)
rnorm(3)

## [1] -0.6264538  0.1836433 -0.8356286

rnorm(3)

## [1]  1.5952808  0.3295078 -0.8204684

set.seed(1)
rnorm(3)  #uguale alla prima sequenza generata

## [1] -0.6264538  0.1836433 -0.8356286
```

Sulla scelta del seme: meglio che sia casuale

2.2 Generazione di numeri casuali ed estrazioni da un'urna

2.2.1 Variabili casuali e funzioni rilevanti

In R si prevede che per la generica variabile casuale `varcas`, i cui principali tipi sono riportati in tabella 1 vi siano 4 funzioni principali:

- `dvarcas(x, ...)` genera la distribuzione di probabilità o la funzione di densità della v.c. nei valori `x`, con parametri della vc specificati al posto dei ...

- `pvarcas(x, ...)` genera la funzione di ripartizione della v.c. nei valori `x`, con parametri della vc specificati al posto dei ...
- `qvarcas(prob, ...)` genera i quantili della v.c. nei valori designati da `prob`, con parametri della vc specificati al posto dei ...
- `rdvarcas(n, ...)` genera `n` valori pseudo casuali per la vc, con parametri della vc specificati al posto dei ...

Ai fini della simulazione, le funzioni più utili sono quelle della famiglia `rvarcas`.

2.2.2 L'uso di `sample`

La funzione `sample` estrae casualmente da un set di oggetti scalari (es vettore).

```
str(sample)

## function (x, size, replace = FALSE, prob = NULL)
```

dove:

- `x` è l'oggetto (urna) dalla quale effettuare il sampling
- `size` è il numero di campioni estratti
- `replace` specifica se il campionamento deve esser fatto con reimmissione
- `prob` specifica la probabilità di estrazione di ogni elemento in `x` (qualora questa non sia uniforme)

Di default il sampling è effettuato senza replacement:

```
set.seed(2)
sample(1:10, 3)

## [1] 5 6 9

sample(1:10, 3)

## [1] 8 1 5

## permutazione
sample(1:5)

## [1] 1 4 5 3 2

sample(1:5)

## [1] 3 1 5 2 4

## campionamento con ripetizione
sample(1:4, 4, replace=T)

## [1] 4 2 3 3
```

La funzione `sample` permette di fatto di di **estrarre da una distribuzione arbitraria** (prima la specifichiamo nel vettore, poi estrairaiamo da essa) e questo può esser utile nelle applicazioni pratiche. Particolarmente utile è l'opzione `prob` che specifica la probabilità di estrazione di ogni singolo elemento. Alcuni esempi seguenti simulano un dado a 6 facce, prima equilibrato, poi truccato (con 6 molto più probabile)

```
## 5 lanci da dado equilibrato
sample(1:6, 10, replace=T)

## [1] 1 6 1 4 3 6 1 6 5 6

## 5 lanci da dado non equilibrato
sample(1:6, 10, replace=T, prob = c(rep(.1,5), .5) )

## [1] 4 6 1 6 6 6 1 5 1 6
```

2.3 L'impiego di replicate

Supponendo di volere simulare diversi campioni e su si essi dover applicare una funzione per analizzare la distribuzione dei risultati, possiamo utilizzare `replicate`

```
str(replicate)

## function (n, expr, simplify = "array")
```

Ad esempio per calcolare la media di 5 campioni composti ciascuno da 100 estrazioni da un'urna normale

```
replicate(5, mean(rnorm(100)))

## [1] -0.018463541 -0.003781908 0.101662525 0.231861455 -0.027696960
```

2.4 Applicazioni varie

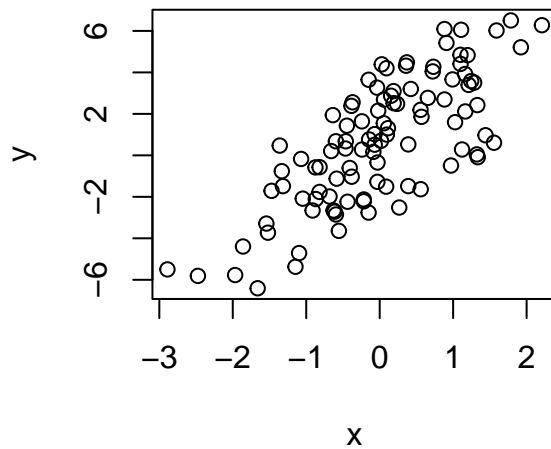
2.4.1 Simulazione di dati per un modello di regressione

Supponendo di voler simulare dal seguente modello lineare

$$y = \beta_0 + \beta_1 x + \epsilon$$

con $\epsilon \sim N(0, 4)$, assumendo $x \sim N(0, 1)$, $\beta_0 = 0.5$ e $\beta_1 = 2$

```
set.seed(20)
x <- rnorm(100)
e <- rnorm(100, 0, 2) # la varianza è 4 ma qui dobbiamo impostare sd
y <- 0.5 + 2*x + e
plot(x, y)
```



Se x è **binaria**, invece, usiamo `rbinom`

```
set.seed(10)
x <- rbinom(100, 1, 0.5)
e <- rnorm(100, 0, 2)
y <- 0.5 + 2*x + e
```

Supponiamo ora di voler simulare da un modello di poisson, con dati di conteggio, ed errore poissoniano dove

$$Y \sim \text{Poisson}(\mu) \quad (1)$$

$$\log \mu = \beta_0 + \beta_1 x \quad (2)$$

e dove $\beta_0 = 0.5$ e $\beta_1 = 0.3$. Dobbiamo impiegare la funzione `rpois`:

```
set.seed(1)
x <- rnorm(100)
log.mu <- 0.5 + 0.3*x # predittore lineare
y <- rpois(100, exp(log.mu))
plot(x, y)
```

