Programación y Estadística con R

Resumen

Este curso es una introducción rápida a un «entorno para la computación estadística y los gráficos», que proporciona una amplia variedad de técnicas estadísticas y gráficas: modelización lineal y no lineal, pruebas estadísticas, análisis de series temporales, clasificación, agrupación, etc. Prácticamente todos los análisis estadísticos que se realizan en Bioinformática se pueden llevar a cabo con R. Además, la «minería de datos» está bien cubierta en R: el clustering (a menudo llamado «análisis no supervisado») en muchas de sus variantes (jerárquico, k-means y familia, modelos de mezcla, fuzzy, etc), bi-clustering, clasificación y discriminación (desde el análisis discriminante a los árboles de clasificación, bagging, máquinas de vectores soporte, etc), todos tienen muchos paquetes en R. Así, tareas como la búsqueda de subgrupos homogéneos en conjuntos de genes/sujetos, la identificación de genes que muestran una expresión diferencial (con ajuste para pruebas múltiples), la construcción de algoritmos de predicción de clases para separar a los pacientes de buen y mal pronóstico en función del perfil genético, o la identificación de regiones del genoma con pérdidas/ganancias de ADN (alteraciones del número de copias) pueden llevarse a cabo en R de forma inmediata.

Sandra Mingo Ramírez

UAM - 2024/25

10 de octubre de 2024 11:26

Universidad Autónoma de Madrid
Bioinformática y Biología Computacional

Código en Github

Índice general

Capítulo I

Introducción en R y estadística

I.1. RStudio y primeras nociones

En RStudio, se puede crear un nuevo fichero en File > New File > R script. Se abre un nuevo fichero en el que se puede programar. En R, la asignación de variables se realiza con <-. En la parte superior derecha, se pueden ver todas las variables que se han asignado en la sesión, los datos y las funciones.

```
x <- 9
y <- matrix(1:20, ncol = 4)
```

En la parte inferior derecha hay una pestaña para poder visualizar los gráficos. Desde ese menú, se puede guardar, pero esto no es recomendable, ya que el gráfico se ajusta al tamaño de la pantalla y luego eso no es reproducible. En otra pestaña aparece un listado de todos los paquetes instalados en el disco duro, aunque luego haya que cargarlos en cada script en el que se desee usar. Al pulsar en el nombre de un paquete, se va a la página de ayuda del mismo. También es posible acceder con:

```
help(rnorm)
```

La mayor parte del trabajo «real» con R requerirá la instalación de paquetes. Los paquetes proporcionan funcionalidad adicional. Los paquetes están disponibles en muchas fuentes diferentes, pero posiblemente las principales ahora son CRAN y BioConductor. Si un paquete está disponible en CRAN, puedes hacer lo siguiente:

```
install.packages("nombre-paquete") # 1 paquete
install.packages(c("paquete1", "paquete2")) # varios paquetes
```

En Bioinformática, BioConductor es una fuente bien conocida de muchos paquetes diferentes. Los paquetes de BioConductor pueden instalarse de varias maneras, y existe una herramienta semiautomatizada que permite instalar conjuntos de paquetes BioC. Implican hacer algo como

```
BiocManager::install("nombre-paquete")
```

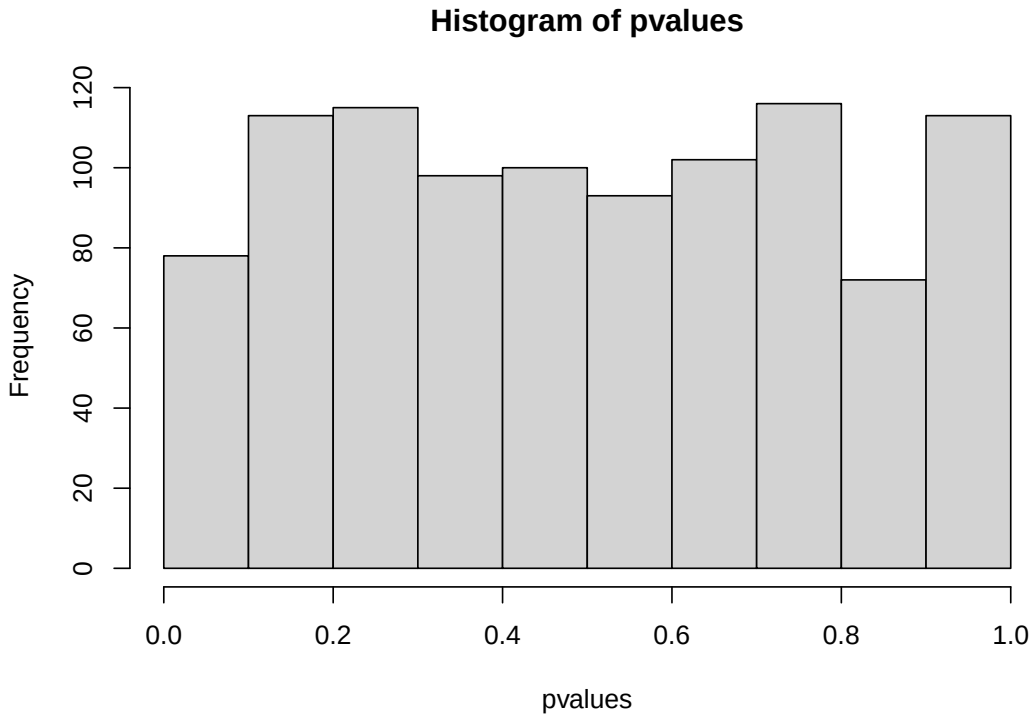
A veces los paquetes dependen de otros paquetes. Si este es el caso, por defecto, los mecanismos anteriores también instalarán las dependencias. Con algunas interfaces gráficas de usuario (en algunos sistemas operativos) también puede instalar paquetes desde una entrada de menú. Por ejemplo, en Windows, hay una entrada en la barra de menú llamada Paquetes, que permite instalar desde Internet, cambiar los repositorios, instalar desde archivos zip locales, etc. Del mismo modo, desde RStudio hay una entrada para instalar paquetes (en «Herramientas»). Los paquetes también están disponibles desde otros lugares (RForge, github, etc); a menudo encontrarás instrucciones allí.

Siempre puedes simplemente matar RStudio; pero eso no es agradable. En todos los sistemas escribir q() en el símbolo del sistema debería detener R/RStudio. También habrá entradas de menú (por ejemplo, «Salir de RStudio» en «Archivo», etc). A continuación sale la pregunta de si se debe guardar el workspace, y en general querremos decir que no.

I.2. Ejemplo: problema de las pruebas múltiples

Es posible que hayamos oído hablar del problema de las pruebas múltiples con los microarrays: si observamos los p-valores de un gran número de pruebas, podemos ser inducidos a pensar erróneamente que está ocurriendo algo (es decir, que hay genes expresados de forma diferencial) cuando, en realidad, no hay absolutamente ninguna señal en los datos. A nosotros esto nos convence. Pero tienes un colega testarudo que no lo está. Ha decidido utilizar un ejemplo numérico sencillo para mostrarle el problema. Este es el escenario ficticio: 50 sujetos, de los cuales 30 tienen cáncer y 20 no. Medimos 1000 genes, pero ninguno de los genes tiene diferencias reales entre los dos grupos; para simplificar, todos los genes tienen la misma distribución. Haremos una prueba t por gen, mostrará un histograma de los valores p e informaremos del número de genes «significativos» (genes con $p < 0{,}05$). Este es el código R:

```
randomdata <- matrix(rnorm(50 * 1000), ncol = 50)
class <- factor(c(rep("NC", 20), rep("cancer", 30)))
pvalues <- apply(randomdata, 1,
                 function(x) t.test(x ~ class)$p.value)
hist(pvalues)
```




```
sum(pvalues < 0.05)

## [1] 48
```

En un test de la t, tenemos una hipótesis nula (H_0) que normalmente es lo contrario de lo que queremos ver (es decir, si queremos ver si hay diferencias entre dos muestras, la hipótesis nula es que son iguales). Después, se aplica la fórmula de la t y se obtiene un valor, que es el que se quiere ver cómo se distribuye si H_0 es cierta. A continuación se mira cómo de probable es que lo que se observa se observe desde H_0. Eso produce el p-valor, que indica si algo bajo H_0 es probable o improbable (evidencia contra la hipótesis nula).

```
randomdata[1, ]
randomdata[, 1]
randomdata[2, ]
randomdata[, 2]
randomdata[2, 3]

class
as.numeric(class)

pvalues[1]

t.test(randomdata[1, ] ~ class)

t.test(randomdata[1, ] ~ class)$p.value
```

```
pvalues[1:10] < 0.05

sum(c(TRUE, TRUE, FALSE))

hist(c(1, 2, 7, 7, 7, 8, 8))
```

```
## For ease
rd2 <- randomdata[1:10, ]

## Where we will store results
pv2 <- vector(length = 10)

for(i in 1:10) {
    pv2[i] <- t.test(rd2[i, ] ~ class)$p.value
}

pv2

## Compare with
pvalues[1:10]
```

Ahora usamos apply. No lo hemos dicho explícitamente, pero cuando usamos apply estamos pasando una función (nuestra función anónima) a otra función. Esto es algo muy común y fácil en R: pasar funciones a otras funciones.

```
apply(rd2, 1, function(z) t.test(z ~ class)$p.value)
```

Or very verbose (this is unnecessarily verbose):

```
myfunction <- function(y, classfactor = class) {
    t.test(y ~ classfactor)$p.value
}

apply(rd2, 1, myfunction)
```

I.3. La consola de R para cálculos interactivos

Independientemente de cómo interactuemos con R, una vez que iniciemos una sesión interactiva de R, siempre habrá una consola, que es donde podemos introducir comandos para que sean ejecutados por R. En RStudio, por ejemplo, la consola suele estar situada en la parte inferior izquierda.Todos los prompts en la consola empiezan con >.

```
1 + 2
```

```
## [1] 3
```

Mira la salida. En este documento, los trozos de código, si muestran salida, mostrarán la salida precedida por ##. En R (como en Python), # es el carácter de comentario. En la consola, NO veremos el ## precediendo a la salida. Esto es sólo la forma en que está formateado en este documento (al igual que no se ve el > antes del comando). Fíjate también en que ves un [1], antes del 3. ¿Por qué? Porque la salida de esa operación es, en realidad, un vector de longitud 1, y R está mostrando su índice. Aquí no ayuda mucho, pero lo haría si imprimiéramos 40 números:

```
1:40
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
## [26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
```

Se puede asignar 1 + 2 a una variable mediante <-. También se puede utilizar =, pero no se aconseja. Para ver el valor de una variable, se puede escribir simplemente el nombre de la variable, utilizar print o hacer la asignación entre paréntesis (eso realiza la asignación y muestra el resultado por pantalla).

```
(v1 <- 1 + 2)
```

```
## [1] 3
```

```
print(v1)
```

```
## [1] 3
```

```
v1
```

```
## [1] 3
```