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

11 de octubre de 2024 23:43

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

I.2.1. Introducción al test de la t

En un test de la t, la hipótesis nula (H_0) suele representar lo contrario de lo que se desea demostrar. Por ejemplo, si nuestro objetivo es comprobar si hay diferencias entre dos muestras, la hipótesis nula establece que ambas son iguales. A continuación, se utiliza la fórmula de la t para obtener un valor estadístico, cuya distribución se examina bajo la suposición de que H_0 es cierta. Luego, se calcula la probabilidad de observar un resultado tan extremo o más extremo que el obtenido bajo H_0. Esta probabilidad se denomina p-valor, y su interpretación indica cuánta evidencia hay en contra de H_0: un p-valor bajo sugiere que lo observado es improbable bajo H_0.

$$t = \frac{x_A - x_B}{SD_{x_A, x_B}}$$

Es importante aclarar que el p-valor no representa la probabilidad de que H_0 sea cierta, ni la probabilidad de que H_0 o la hipótesis alternativa (H_1) se cumplan dado los datos. Lo que el p-valor señala es que, o bien H_0 es falsa, o ha ocurrido un evento tan improbable como el valor observado. No se "rechaza" H_0 de manera concluyente, sino que simplemente no se acepta si el p-valor es suficientemente bajo. En este análisis, se compara el resultado observado con todos aquellos más extremos, algo que es distinto de seleccionar el valor que hace los datos lo más probables posible (como se hace en la máxima verosimilitud).

Por ejemplo, una moneda perfectamente equilibrada tiene una probabilidad de 0.5^6 de que al lanzarla seis veces, salga exactamente tres veces cara y tres veces cruz. Aunque este número es pequeño, no implica que la hipótesis alternativa sea necesariamente más probable, ya que otros resultados también podrían ser igualmente o más improbables.

En la mayoría de los casos de comparación de medias, los datos no están restringidos a un único valor.

Cuando H_0 es cierta:

$$Pr(p - valor \leq 0,05) = 0,05$$

$$Pr(p - valor \leq 0,01) = 0,01$$

En muchos casos se comprueba más de una H_0. En un screening, se analizan 20.000 genes y se decide elegir todos aquellos que tengan un p-valor inferior a 0,05. Esa lista, sobre el total de los genes, la probabilidad de rechazar H_0 cuando es cierta, es muy superior al 5 %, aunque se cumpla para cada gen individual. Así, se debe trasladar la lógica al test múltiple, puesto que si no se va a rechazar H_0 en muchas ocasiones cuando no se debería.

I.2.2. Problema de las pruebas múltiples

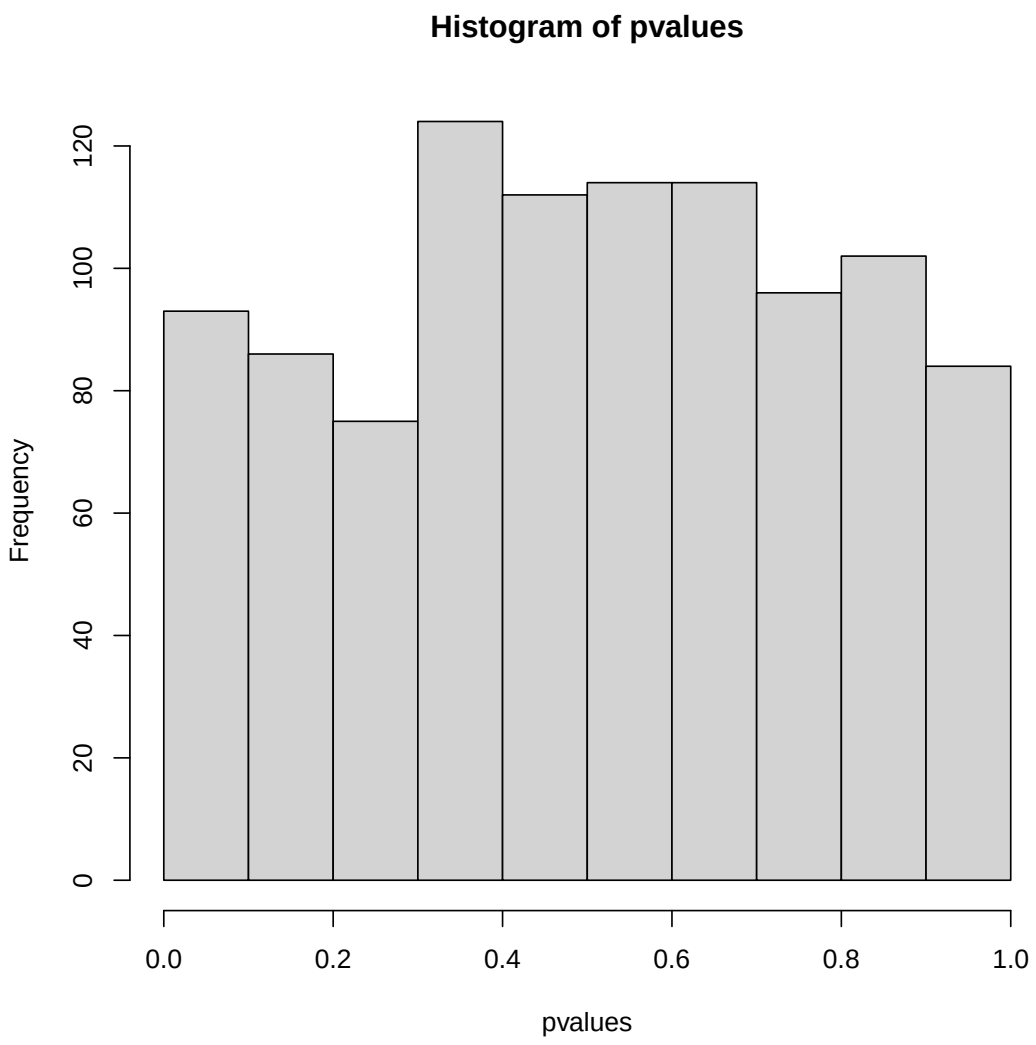
Es posible que hayamos oído hablar del problema de las pruebas múltiples con los microarrays: si observamos los p-valores de un gran número de pruebas, podemos ser inducidos a pensar erróneamente que está ocurriendo algo (es decir, que hay genes expresados de forma diferencial) cuando, en realidad, no 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 (una distribución normal). 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)
```

Para leer el código, se empieza por la función más interna, que en este caso es `rnorm`. Así, primero se generan 50.000 entradas de distribución normal (1000 genes por 50 personas) de los que se quiere realizar 1000 contrastes de hipótesis (uno por gen) y representar el aspecto de la distribución (que será uniforme). Todas las entradas se organizan en una matriz con 50 columnas. Después, se crean los dos grupos que se están analizando mediante repeticiones (función `rep`). El comando de `factor` crea las etiquetas. En R, se puede llamar al test de la t de varias maneras, siendo una estándar con la interfaz de tipo fórmula (x ∼ class), dividiendo así x en los distintos niveles que se han creado previamente. La sintaxis siempre es una variable que va cambiando (en este caso, las filas) antes de la virgulilla y una variable constante después de la virgulilla (los distintos niveles). La función `apply` permite aplicar una función a un objeto o conjunto de datos, evitando así tener que realizar un bucle for. El primer

argumento es el objeto, el segundo la dimensión del objeto a lo que se quiere aplicar (si se recorren filas, columnas, etc.), y el tercero la función que se va a aplicar. La función `t.test` devuelve objetos a los que se puede acceder, como el valor `t`, df, p-valor, la media de cada grupo, etc. Se puede acceder al nombre de todos los valores mediante `names(t.test(x ~ class))`. En nuestro caso, `x` es el valor que irá adquiriendo el número de filas a recorrer. En este caso, se define la función en el momento de llamarla, pero también se puede definir antes y utilizarla en el apply. En este caso se define dentro porque es una función corta que solo se utilizará en ese momento, por lo que no es necesario crearla fuera. Si por el contrario fuese una función a la que quisiéramos acceder posteriormente o que fuese compleja con varias líneas, se suele crear fuera. Por último, se accede a los p-valores y se guardan en la variable `pvalues`. Esos p-valores se pueden representar a continuación en un histograma y calcular todos aquellos que sean menores o iguales que 0,05.

```
hist(pvalues)
```



```
sum(pvalues <= 0.05)

## [1] 43
```

Al realizar la suma de una lógica booleana, se coercia para que los valores falsos se conviertan en 0 y los verdaderos en 1. Así, al sumarlos, el resultado es numérico.

En resumen, en este ejemplo hemos visto los siguientes objetos:

- Vectores: colección de uno o más datos del mismo tipo.
- Matrices: conjunto de datos indexados por filas y columnas del mismo tipo.
- Arrays: generalización de una matriz que no tiene límite de dimensiones (pero debe tener una estrucrtura rectangular).
- Data frames: estructura rectangular de dos dimensiones (filas y columnas) en la que cada columna puede ser de un tipo diferente.
- Listas: cajón desastre en el que se pueden meter muchas cosas de muchos tipos distintos. Muchas funciones devuelven listas u objetos que contienen listas.
- Factores: vectores de un tipo especial (variable categórica).
- Funciones: objetos que realizan una operación y devuelven algo.

En el siguiente código se muestran las distintas maneras de acceder a una matriz. La indexación funciona [filas, columnas], y si un campo está sin rellenar implica todos sus datos.

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

Al ejecutar la variable `class` creada anteriormente, no solo devuelve la lista de los elementos con las distintas etiquetas, si no que también muestra al final los distintos niveles. Como `factor` por detrás les asignó un valor entero que corresponda a la etiqueta dada, cuando se pide convertir en numérico, se devuelve el entero. La asignación de los valores se realiza por orden alfanumérico.

```
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)
```

Esta es otra forma de hacerlo, pero es más verbosa (quizás incluso innecesariamente verbosa):

```
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] 21 22 23 24 25 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
```