eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

COMENTARIOS A LA TAREA 1

1. Escribe una expresión que calcule Prob$\{X = x\}$ en la distribución binomial $b(p, n)$, es decir,

$$\binom{n}{x} p^x (1-p)^{n-x}$$

para valores $n = 10$, $x = 5$ y $p = 0{,}8$.

Respuesta: Trivial.

2. Escribe una función que haga lo anterior, admitiendo valores cualesquiera de n, x y p como argumentos.

Respuesta: Trivial.

3. Escribe una función que, dados n y p como argumentos, proporcione *todas* las probabilidades correspondientes a $x = 0, \ldots, n$.

Respuesta: Hay muchas posibilidades. Podríamos hacer, por ejemplo (en R),

```
bin1 <- function(n,p) {
          res <- rep(0,n+1)
          for (i in 0:n) {
            res[i+1] <- choose(n,i)*p^i*(1-p)^(n-i)
          }
          return(res)
        }
```

pero también (aprovechando que muchas de las funciones de R están vectorizadas, es decir, que dando un argumento vectorial proporcionan un vector de soluciones)

```
bin2 <- function(n,p) {
          res <- choose(n,0:n)*p^(0:n)*(1-p)^(n-0:n)
          return(res)
        }
```

4. Escribe una función bivalente que refunda las dos anteriores. Si se proporciona el argumento x, debe devolver sólo la probabilidad pedida; en otro caso, la tabla completa.

Respuesta: Esta es una solución (simplificada; no se hace, como en las anteriores, ninguna comprobación de validez y consistencia de los argumentos):

```
bin3 <-  function(n,p,x=NA) {
           if (is.na(x)) {
             res <- choose(n,0:n)*p^(0:n)*(1-p)^(n-0:n)
           }
           else {
             res <- choose(n,x)*p^(x)*(1-p)^(n-x)
           }
           return(res)
         }
```

Obsérvese que funcionará tanto si no se da argumento `x` como si éste es un escalar o un vector.

5. A partir de la identidad[1]

$$\binom{n}{x}p^x(1-p)^{n-x} = p\binom{n-1}{x-1}p^{x-1}(1-p)^{n-x+1} + (1-p)\binom{n-1}{x}p^x(1-p)^{n-x-1},$$

escribe una función como la indicada en el ejercicio 2, pero recursiva.

Respuesta: Las dos condiciones extremas bajo las cuales es fácil obtener los valores de la probabilidad binomial son $n = x$ y $x = 0$. Haciendo uso de este hecho podemos explotar la idea de recursividad así:

```
binrec <- function(n,p,x) {
            if (x==0) {
               return((1-p)^n)
            }
            else if (n==x) {
               return(p^x)
            }
            else {
               return( p*binrec(n-1,p,x-1)
                       + (1-p)*binrec(n-1,p,x) )
            }
          }
```

Esto tiene interés puramente didáctico: si ejecutáis con valores pequeños de los argumentos, obtenéis la respuesta correcta, como puede verse comparando con la respuesta de `dbinom` (la función de librería de R que hace lo propio)

```
> binrec(10,.43,5)
[1] 0.2229036
> dbinom(5,10,.43)
[1] 0.2229036
> binrec(20,.43,10)
[1]
 0.1445545
> dbinom(10,20,.43)
[1] 0.1445545
```

Con argumentos como $n = 20$ y $x = 10$, `binrec` ya es notoriamente más lenta. Y con argumentos algo crecidos, obtenéis un desbordamiento de la memoria de pila:

```
> binrec(200,.43,100)
Error in binrec(n - 1, p, x - 1) : evaluation is nested too deeply:
infinite recursion?
```

[1] En palabras: "La probabilidad de obtener x éxitos en n ensayos es la probabilidad de obtener $x-1$ éxitos en los primeros $n-1$ ensayos y un éxito en el n-ésimo más la probabilidad de obtener x éxitos en los primeros $n-1$ ensayos y un fracaso en el n-ésimo".

En general, la recursividad *es cara*, aunque a veces sea el modo más elegante, si no el único, de obtener un resultado. Esto es particularmente cierto en algoritmos de ordenación y búsqueda.

6. A menudo en el contraste estadístico de hipótesis, en lugar de fijar una región crítica por anticipado, se computa un valor crítico o *p-value*. Se razona así: *Supongamos* (hipótesis nula) que la variable aleatoria X se genera al muestrear una binomial $b(p,n)$, lo que denotamos por $X \sim b(p,n)$. Entonces, deberíamos observar la mayor parte de las veces un valor no muy alejado de np. Si el valor de X que observamos está "muy alejado" de np, deberemos entenderlo como evidencia contra nuestra suposición inicial (= hipótesis nula) de que $X \sim b(p,n)$.

   Se define el *p-value* como la probabilidad bajo la hipótesis nula de obtener un resultado igual o más raro que el obtenido. Si la hipótesis nula fuera, por ejemplo, que la probabilidad de "cara" de una moneda es $p = p_0$ y al realizar n lanzamientos obtenemos x "caras", el *p-value* sería:

$$\sum_{z \in \mathcal{C}} \binom{n}{z} p_0^z (1-p_0)^{n-z} \tag{1}$$

   donde $\mathcal{C}$ es el conjunto de posibles resultados cuya probabilidad es menor o igual[2] que

$$\binom{n}{x} p_0^x (1-p_0)^{n-x}$$

   Escribe una función que admitiendo como argumentos n, x y p_0 proporcione como resultado el *p-value* asociado a la hipótesis nula $H_0 : p = p_0$.

   Respuesta:

```
pvalue <- function(n,p0,x) {
            r <- bin2(n,p0)
            s <- bin3(n,p0,x)
            pobs <- sum(r[r<=s])
            return(pobs)
          }
```

   Observemos que `s` ha sido calculado en `r`, por lo que bastaría `s <- r[x+1]`.

7. Una operación que muy frecuentemente hay que realizar es evaluar un polinomio como $p(x) = a_0 x^n + a_1 x^{n-1} + \ldots + a_{n-1}x + a_n$. ¿Cuantas operaciones (sumas, productos, cocientes) necesitas para evaluarlo de la manera más directa?

   Puede hacerse de manera más simple (regla de Horner: explicada, por ej., en [1], y en cualquier libro de Análisis Numérico). Imagina un polinomio cúbico. Podrías reescribirlo así:

$$p(z) = ((a_0 z + a_1)z + a_2)z + a_3$$

   lo que sugiere un modo de evaluarlo sin tener que calcular potencias, sólo mediante sumas y productos. Escribe una función que admita como argumentos un vector de coeficientes $a_0, \ldots, a_n$ y el valor z y proporcione $p(z)$.

   Respuesta: Trivial.

[2] Se supone un contraste sin alternativa preespecificada: si la alternativa fuera unilateral, la definición se modificaría del modo obvio.

OTROS COMENTARIOS

1. En general, se suele evitar que una función de posible utilización en otras escriba directamente a la consola. Las funciones "finales" se encargan de hacerlo.

   En ocasiones, cuando se tienen funciones bivalentes, de posible uso como bloque constructivo en otras o directamente utilizables por el usuario, se recurre a incluir un argumento como `verbose=FALSE`. Si en una invocación se hace `verbose=TRUE`, condicionales en el código de la función imprimen mensajes de control o error, de otro modo se ejecuta silenciosamente.

2. Hay que evitar evaluar reiteradamente la misma expresión. En general, una expresión que no dependa del contador de un bucle, bastaría evaluarla una sola vez antes del comienzo del mismo. Por ejemplo,

```
pvalue <- function(n,p0,x) {
            pobs <- 0
            for (i in 0:n) {
               if (bin3(n,p0,i) < bin3(n,p0,x)) {
                   pobs <- pobs + bin3(n,p0,i)
               }
            }
            return(pobs)
          }
```

   calcula correctamente el p-value requerido, pero evalúa `bin3(n,p0,x)` $n+1$ veces (cuando sólo se requiere hacerlo una). Con la misma estructura de función,

```
pvalue <- function(n,p0,x) {
            pobs <- 0
            pref <- bin3(n,p0,x)
            for (i in 0:n) {
               p  <- bin3(n,p0,i)
               if (p < pref) {
                   pobs <- pobs + p
               }
            }
            return(pobs)
          }
```

   es bastante más eficiente. De hecho, cuando se emplean compiladores optimizadores, código fuente como el de la primera función se reescribiría automáticamente en código como el de la segunda, antes de ser convertido a lenguaje de máquina.

3. En un lenguaje interpretado, en general, y en R en particular, es casi siempre ventajoso sustituir los bucles por funciones vectorizadas, que operan con vectores o matrices a la vez. La solución proporcionada más arriba es mejor que cualquiera de las dos basadas en un bucle `for`.

4. Es costumbre hacer que las funciones devuelvan algún código de error cuando no han podido completar el cómputo requerido. De este modo, la función o programa que las llama puede hacer comprobación de errores.

Referencias

[1] G. Dahlquist and Åke Björck. *Numerical Methods*. Prentice Hall, Englewood Cliffs, N.J., 1974.

[2] R.A. Thisted. *Elements of Statistical Computing*. Chapman & Hall, New York, 1988.