Apuntes LAB3

Probabilidad

En este laboratorio vamos a aprender a estimar probabilidades utilizando simulación. Aprenderemos el uso de bucles `for` y otras estructuras de control, especialmente el uso de `if` para estudiar la probabilidad condicionada.

También introduciremos las variables aleatorias y sus distribuciones, con especial interés en las distribuciones binomial, normal y de Poisson.

Por último, comprobaremos la ley de los grandes números y el teorema central del límite.

Estimar probabilidades por simulación

La simulación permite, de forma muy intuitiva, comprender la definición frecuentista de la probabilidad: proporción de veces que ocurre un resultado de interés si el fenómeno aleatorio pudiera repetirse infinitamente. Con R, es posible realizar simulaciones con suficientes repeticiones de una manera rápida y fácil.

Muestrear aleatoriamente utilizando `sample()`

En anteriores laboratorios hemos utilizado la función `sample()` para seleccionar muestras aleatorias, donde todos los elementos tienen la misma probabilidad de ser elegidos. Sin embargo, en ocasiones es necesario simular experimentos en los que no todos los resultados son igualmente probables. En general, la estructura de `sample()` es,

```
sample(x, size = , replace = FALSE, prob = NULL)
```

donde el argumento `prob` permite introducir, en forma de vector, probabilidades diferentes para que cada elemento de `x` sea elegido. Cuando se omite este argumento, todos los elementos de `x` tendrán la misma probabilidad de ser elegidos.

El siguiente código simula 10 lanzamientos de una moneda sesgada, con probabilidad de cara (valor 1) 0.6.

```
# establecemos la semilla para reproducir los resultados
set.seed(2020)
res <- sample(c(0, 1), size = 10, prob = c(0.4, 0.6), replace = TRUE)
res
[1] 0 1 0 1 1 1 1 1 1 0
```

El primer elemento de x, 0, será elegido con probabilidad 0.4 y el segundo (1) con probabilidad 0.6.

La función sum()

La función sum() se va a utilizar habitualmente para devolver la suma de un vector que almacena 0's y 1's.

```
sum(res) # nº de 1's (caras)
[1] 7
10 - sum(res) # nº de 0's (cruces)
[1] 3
```

Es muy habitual utilizar esta función combinada con vectores lógicos.

```
sum(res == 1) # nº de 1's (caras)
[1] 7
sum(res == 0) # nº de 0's (cruces)
[1] 3
```

Esto nos proporciona mucha flexibilidad a la hora de trabajar con vectores que pueden tener más de dos valores distintos. Por ejemplo, el siguiente código identifica el número de lanzamientos de un dado de 6 lados, de entre 20, cuyo resultado es 1 o mayor que 4.

```
# establecemos la semilla para obtener la misma muestra
set.seed(2020)
res.dado <- sample(1:6, size = 20, replace = TRUE)
res.dado
[1] 4 4 6 1 1 4 2 6 1 5 2 2 6 5 2 3 2 5 4 2
sum(res.dado == 1 | res.dado > 4)
[1] 9
```

Además, esta forma de trabajar va a permitir poder utilizar sum() con vectores no-numéricos.

```
# establecemos la semilla para obtener la misma muestra
set.seed(2020)
res <- sample(c("X","C"),size = 10,prob = c(0.4, 0.6),replace = TRUE)
res
[1] "X" "C" "X" "C" "C" "C" "C" "C" "C" "X"
sum(res == "C") # nº de caras
[1] 7
```

Bucles for

Un bucle permite que un conjunto de código se repita bajo un conjunto específico de condiciones. El bucle for es uno de los bucles disponibles en R.

Un bucle `for` tiene la estructura básica `for(contador) { instrucciones }`. En el siguiente ejemplo vamos a calcular los cuadrados de todos los enteros del 1 al 5. Aspectos relevantes:

- Antes de ejecutar el bucle, creamos un vector vacío, `squares`, para almacenar los resultados. Se utiliza el comando `vector( )`. Este paso se conoce, en general, como inicialización.
- El contador es un índice que realiza el seguimiento de cada iteración del bucle; el índice suele ser denotado por una letra como i, j o k, pero puede ser cualquier secuencia. Conceptualmente, este índice es similar al índice i utilizado en un sumatorio $\sum_{i=1}^{n}$. En nuestro código, leeríamos "por cada k en la secuencia del 1 al 5, repita las siguientes instrucciones".
- Las instrucciones se incluyen entre `{ }`. Para cada iteración, el valor k^2 debe almacenarse en el elemento k-ésimo del vector `squares`.
- Para la primera iteración, se establece k = 1, se calcula 1^2 y se rellena el primer elemento de `squares` con el resultado. En la siguiente iteración, k = 2, se calcula 2^2 y se rellena el segundo elemento de `squares` con el resultado. El proceso se repite hasta que k = 5 y 5^2 se almacena en el quinto elemento de `squares`.

```
# Inicialización: creamos el vector squares
squares <- vector("numeric", 5)
# bucle for
for(k in 1:5){
   squares[k] = k^2
}
# resultados almacenados en squares
squares
```

```
[1] 1 4 9 16 25
```

En este caso tan sencillo, podríamos haber obtenido el mismo resultado sin utilizar este bucle.

```
(1:5)^2
```

```
[1] 1 4 9 16 25
```

Por norma general, y siempre que sea posible, preferiremos utilizar otras alternativas a los bucles, ya que nuestros programas serán más eficientes.

Probabilidad condicionada

La estructura de control `if` puede utilizarse dentro de bucles para la estimación de probabilidades condicionadas. La estructura de control `if` tiene la estructura `if (`

`condicion ) { intrucciones }`. Si se satisface la `condicion` se llevarán a cabo las `instrucciones`.

En la estimación de probabilidades condicionadas, esta estructura puede utilizarse para contar "éxitos" y para simular poblaciones basadas en el conocimiento de ciertas condiciones. Vamos a ver un ejemplo de cada uno de estos usos.

Contar "éxitos"

Una urna contiene 3 bolas rojas y 3 blancas. Se extraen dos bolas de las urnas de forma que la primera bola no es devuelta a la urna antes de extraer la segunda. Queremos estimar la probabilidad de extraer primero una bola blanca y después una roja.

En este caso el experimento será la extracción de dos bolas de la urna sin reemplazamiento, y el suceso de interés extraer la secuencia Blanca-Roja. Vamos a simular los resultados de 10 experimentos.

La función `rep(x,times)` se puede utilizar para replicar los elementos del vector `x` `times` veces. En este caso, podemos utilizar `rep()` para crear el vector que contiene todas las bolas que tenemos dentro de la urna.

```
bolas <- rep(c("R", "B"), c(3, 3))
bolas
```

```
[1] "R" "R" "R" "B" "B" "B"
```

```
# definimos el resto de parámetros
n.extracciones <- 2 # extraemos dos bolas en nuestro experimento
n.replicas <- 10 # repetimos el experimento 10 veces
# creamos un vector vacío para almacenar los resultados
exitos <- vector("numeric", n.replicas)
# establecemos la semilla para poder replicar resultados
set.seed(2020)
# simulamos las extracciones
for(k in 1:n.replicas){
   extrae <- sample(bolas, size = n.extracciones, replace = FALSE)
   if(extrae[1] == "B" & extrae[2] == "R"){
      exitos[k] <- 1
   } # fin de if
} # fin de for
# vemos los resultados
exitos
```

```
[1] 0 1 0 0 1 0 0 0 1 0
```

```
table(exitos)
```

```
exitos
0 1
7 3
```

La estructura `if` está anidada dentro del bucle `for`. Notad que,

- En la estructura `if`, la condición requiere que la primera extracción sea una bola blanca, "B", y la segunda una bola roja, "R".
- Si se cumple la condición en la k-ésima réplica se almacena un 1 en el k-ésimo elemento del vector de resultados `exitos`.

Por lo tanto, la probabilidad que nos piden se podrá calcular dividiendo el número de éxitos por el total de réplicas.

```
# estimar la probabilidad
sum(exitos)/n.replicas
```

```
[1] 0.3
```

Simular poblaciones para estimar probabilidades

En una determinada población se sabe que, aproximadamente el 20% de los hombres y el 3% de las mujeres tienen una altura mayor que 180 cm. Asumiendo que la mitad de la población son hombres y la otra mitad mujeres,

- ¿Cuál es la probabilidad de encontrar una persona que, midiendo más de 180 cm, sea mujer?
- ¿Cuál es la probabilidad de encontrar una persona que mida más de 180 cm?

Una posibilidad de estimar probabilidades condicionadas es simular grandes poblaciones que contengan la información conocida. En este caso, la idea es asignar, a cada individuo y de manera aleatoria, el sexo y el suceso "medir más de 180 cm", respetando la información conocida. Para estimar las probabilidades simplemente tendremos que contar cuantos individuos en dicha población representa un "éxito" cumpliendo las condiciones especificadas. El siguiente código simula una población de 10000 individuos para este ejemplo.

```
# definimos los parámetros
p.mujer <- 0.50 # probabilidad de ser mujer
p.alto.si.mujer <- 0.03 # probabilidad de ser alto condicionada a ser
mujer
p.alto.si.hombre <- 0.20 # probabilidad de ser alto condicionada a ser
hombre
n.pob <- 10000 # tamaño de la población
# establecemos la semilla para poder replicar resultados
set.seed(2020)
# creamos dos vectores vacíos, uno para el sexo y otro para la altura
sexo <- vector("numeric", n.pob)
alto <- vector("numeric", n.pob)
# asignamos el sexo: cada individuo será mujer (1) con probabilidad
p.mujer
```

```
sexo <- sample(c(0,1), size = n.pob, prob = c(1 - p.mujer,
p.mujer),replace = TRUE)
```

Para asignar la altura tenemos que tener en cuenta que los datos que tenemos son las probabilidades de ser alto condicionadas al sexo. Por lo tanto, para hacer esta asignación, en cada individuo tendremos que tener en cuenta si es hombre o mujer, puesto que las probabilidades serán diferentes en cada caso.

Una vez asignado el sexo, utilizamos un bucle for que recorrerá todos los individuos. Cada ciclo del bucle se corresponde con un individuo, y para cada uno de ellos, utilizamos if para comprobar si es hombre o mujer y asignarle la altura según la probabilidad condicionada correspondiente.

```
# asignamos altura: cada individuo valor 1 si mide más de 180cm
for (k in 1:n.pob){
   if (sexo[k] == 0) { # si el individuo k es hombre
      alto[k] <- sample(c(0,1), prob = c(1 - p.alto.si.hombre,
p.alto.si.hombre), size = 1, replace = TRUE)
   } # fin hombre
   if (sexo[k] == 1) { # si el individuo k es mujer
      alto[k] <- sample(c(0,1), prob = c(1 - p.alto.si.mujer,
p.alto.si.mujer), size = 1, replace = TRUE)
   } # fin mujer
} # fin for
# Resultados: tabla de contingencia
addmargins(table(sexo, alto))
```

```
     alto
sexo      0     1   Sum
  0    3941   964  4905
  1    4947   148  5095
  Sum  8888  1112 10000
```

Esta población simulada puede utilizarse para estimar diferentes probabilidades, tanto conjuntas como condicionadas, sin más que calcular las frecuencias correspondientes.

```
# probabilidad de ser mujer y alta
sum(sexo == 1 & alto == 1)/n.pob
```

```
[1] 0.0148
```

```
# probabilidad de ser mujer condicionada a ser alto
sum(sexo == 1 & alto == 1)/sum(alto == 1)
```

```
[1] 0.1330935
```

```
# probabilidad de ser alto condicionada a ser mujer
sum(sexo == 1 & alto == 1)/sum(sexo == 1)
```

```
[1] 0.02904809
```

```
# notad que ~0.03 la probabilidad condicionada del enunciado
# probabilidad de ser alto
sum(alto == 1)/n.pob
```

```
[1] 0.1112
```

Podéis comprobar lo cerca que están estos valores de las probabilidades calculadas algebraicamente.

Variables aleatorias y sus distribuciones

R tienen implementadas funciones que permiten trabajar con múltiples distribuciones. Concretamente vamos a trabajar con las distribuciones binomial, Normal y de Poisson.

La distribución binomial

La distribución binomial está caracterizada por dos parámetros: el número de ensayos de Bernoulli independientes n, y la probabilidad de éxito p, $X \to bin(n,p)$.

La probabilidad de que ocurran exactamente k éxitos en n ensayos independientes es,

$$P(X=k) = \binom{n}{k} p^k (1-p)^{n-k}$$

Podemos utilizar la función `dbinom()` para calcular $P(X=k)$ cuando $X \to bin(n,p)$. La estructura de esta función es `dbinom(x, size, prob)`, donde x is *k*, `size` es el número de ensayos de Bernouilli independientes *n*, y `prob` es la probabilidad de éxito *p*.

Los comandos siguientes muestran como calcular la probabilidad *P* (*X* = 5) para $X \to bin(10,0.35)$.

```
# Probabilidad de que X es exactamente igual que 5
dbinom(x = 5, size = 10, prob = 0.35)
```

```
[1] 0.1535704
```

```
# No es necesario especificar el nombre de los argumentos,
# basta con respetar el orden
dbinom(5, 10, 0.35)
```

```
[1] 0.1535704
```

La función `pbinom()` calcula las probabilidades $P(X \leq k)$o $P(X > k)$ relacionadas con la variable aleatoria $X \to bin(n,p)$. Su estructura es `pbinom(q, size, prob, lower.tail = TRUE)`, donde q is *k*, `size` es el número de ensayos de Bernouilli independientes *n*, y `prob` es la probabilidad de éxito *p*. Por defecto, R calcula la probabilidad $P(X \leq k)$. Para calcular $P(X > k)$ tenemos que especificar el argumento `lower.tail = FALSE`.

Los comandos siguientes muestran como calcular la probabilidad $P(X \leq 5)$ y $P(X > 5)$ para $X \to bin(10,0.35)$.

```
# probabilidad X menor o igual que 5
pbinom(5, 10, 0.35)
```

```
[1] 0.9050659
# probabilidad X mayor (estrictamente) que 5
pbinom(5, 10, 0.35, lower.tail = FALSE)
[1] 0.09493408
```

La distribución normal

La distribución normal está caracterizada por dos parámetros: la media (μ), y la desviación típica (σ), y es denotada por $N(\mu, \sigma)$. La distribución normal estándar es aquella con media 0 y desviación típica 1. Una variable aleatoria que sigue una normal estándar se denota habitualmente por Z.

El z-score de una observación cuantifica cuantas desviaciones típicas se aleja de su media. El z-score de una observación x de una variable aleatoria $X \to N(\mu, \sigma)$ se calcula como,

$$z = \frac{x - \mu}{\sigma}$$

La variable X estandarizada, $Z = \frac{X-\mu}{\sigma}$ tendrá distribución normal estándar.

La función `pnorm()` calcula las probabilidades $P(X \leq k)$o $P(X > k)$ relacionadas con la variable aleatoria $X \to N(\mu, \sigma)$. Su estructura es `pnorm(q, mean=0, sd=1, lower.tail = TRUE)`, donde `q` is *k*, `mean` es el parámetro media μ, y `sd` es el parámetro desviación típica σ. Por defecto, R calcula la probabilidad $P(X \leq k)$ con el argumento `lower.tail = TRUE`, y asume que la media y la desviación estándar son 0 y 1 respectivamente (normal estándar). Para calcular $P(X > k)$ tenemos que especificar el argumento `lower.tail = FALSE`.

Los comandos siguientes muestran como calcular la probabilidad $P(X \leq 105)$ y $P(X > 105)$ para $X \to N(100,5)$.

```
# probabilidad X menor o igual que 105
pnorm(105, 100, 5)
[1] 0.8413447
# probabilidad X mayor que 105
pnorm(105, 100, 5, lower.tail = FALSE)
[1] 0.1586553
```

Los comandos siguientes muestran como calcular la probabilidad $P(Z \leq 1)$ y $P(Z > 1)$ para $X \to N(0,1)$.

```
# probabilidad Z menor o igual que 1
pnorm(1)
[1] 0.8413447
# probabilidad Z mayor que 1
pnorm(1, lower.tail = FALSE)
[1] 0.1586553
```

Podemos utilizar la función `qnorm()` para identificar la observación que se corresponde con una probabilidad concreta, es decir $k\ tal\ que\ P(X \leq k) = p\ con\ X \rightarrow N(\mu, \sigma)$. La estructura de esta función es `qnorm(p, mean=0, sd=1, lower.tail = TRUE)`, donde p is *p*, `mean` es el parámetro media μ, y `sd` es el parámetro desviación típica σ. Por defecto, R identifica la observación que deja una probabilidad p por debajo, en la cola izquierda con el argumento `lower.tail = TRUE`, y asume que la media y la desviación estándar son 0 y 1 respectivamente (normal estándar). Para calcular $k\ tal\ que\ P(X > k) = p\ con\ X \rightarrow N(\mu, \sigma)$ tenemos que especificar el argumento `lower.tail = FALSE`.

Los comandos siguientes muestran como calcular el valor de observación, tanto en la distribución no estandarizada, $N(100,5)$, como en la estandarizada, que deja una probabilidad de 0.841 por debajo de ella, y la observación que deja un área 0.159 por encima.

```
# valor de x que deja una probabilidad 0.841 por debajo
qnorm(0.841, 100, 5)
[1] 104.9929
# valor de x que deja una probabilidad 0.159 por arriba
qnorm(0.159, 100, 5, lower.tail = FALSE)
[1] 104.9929
# valor de z que deja una probabilidad 0.841 por debajo
qnorm(0.841)
[1] 0.9985763
# valor de z que deja una probabilidad 0.159 por arriba
qnorm(0.159, lower.tail = FALSE)
[1] 0.9985763
```

La distribución de Poisson

La distribución de Poisson está caracterizada por el parámetro λ, que representa la tasa de sucesos que ocurren por unidad de tiempo, $X \rightarrow \mathcal{p}(\lambda)$. Se verifica que $E(X) = \lambda$ y $Var(X) = \lambda$

La probabilidad de que ocurran exactamente k sucesos en t unidades de tiempo es,

$$P(Y = k) = \frac{e^{-\lambda t}(\lambda t)^k}{k!}\ \ ya\ que\ Y \rightarrow \mathcal{p}(\lambda t)$$

Podemos utilizar la función `dpois()` para calcular $P(X = k)$ cuando $X \rightarrow \mathcal{p}(\lambda)$. La estructura de esta función es `dpois(x, lambda)`, donde `x` is *k*, y `lambda` es la tasa de sucesos por unidad de tiempo λ.

Los comandos siguientes muestran como calcular la probabilidad *P* (*X* = 5) para $X \rightarrow \mathcal{p}(3)$.

```
# Probabilidad de que X es exactamente igual que 5
dpois(5, 3)
[1] 0.1008188
```

La función `ppois()` calcula las probabilidades $P(X \leq k)$o $P(X > k)$ relacionadas con la variable aleatoria $X \to \mathcal{p}(\lambda)$. Su estructura es `ppois(q, lambda, lower.tail = TRUE)`, donde q is *k*, y `lambda` es la tasa de sucesos por unidad de tiempo λ. Por defecto, R calcula la probabilidad $P(X \leq k)$. Para calcular $P(X > k)$ tenemos que especificar el argumento `lower.tail = FALSE`.

Los comandos siguientes muestran como calcular la probabilidad $P(X \leq 5)$ y $P(X > 5)$ para $X \to \mathcal{p}(3)$.

```
# probabilidad X menor o igual que 5
ppois(5, 3)
[1] 0.9160821
# probabilidad X mayor (estrictamente) que 5
ppois(5, 3, lower.tail = FALSE)
[1] 0.08391794
```

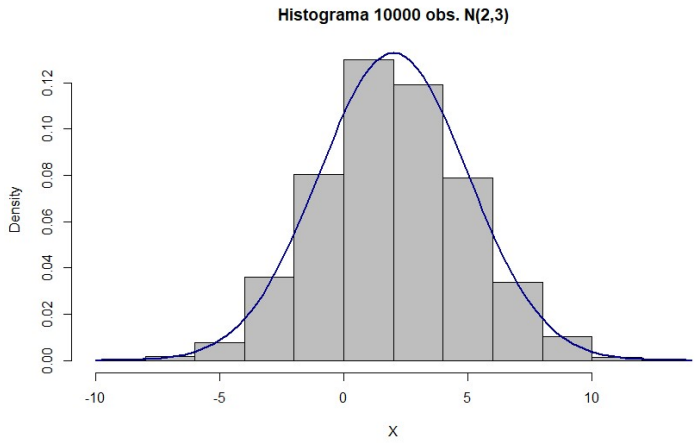
Simular observaciones de una distribución

En R podemos generar números aleatorios procedentes de una distribución concreta. Existen muchas distribuciones disponibles y, en general, el comando utilizado es `r` y el nombre de la distribución. La estructura para las distribuciones que usamos en este curso se resumen en la siguiente tabla,

Distribución	Estructura	Parámetros
Binomial	`rbinom(n, size, prob)`	`size = n, prob = p`
Exponencial	`rexp(n, rate)`	`rate = λ`
Gamma	`rgamma(n, shape, rate)`	`shape = r, rate = λ`
Geométrica	`rgeom(n, prob)`	`prob = p`
Normal	`rnorm(n, mean ,sd)`	`mean = μ, sd = σ`
Pascal	`rnbinom (n, size ,prob)`	`size = r, prob = p`
Poisson	`rpois(n, lambda)`	`lambda = λ`
Uniforme	`runif (n, min ,max)`	`min = a, max = b`

En todos los casos `n` es el número de observaciones a generar. El siguiente código genera 10000 observaciones de una variable aleatoria N(2,3).

```
# establecemos la semilla para poder replicar resultados
set.seed(2020)
# simulamos las observaciones normales
X <- rnorm(10000, 2, 3)
# hacemos un histograma
hist(X, main = "Histograma 10000 obs. N(2,3)",col = "grey", prob =
TRUE)
# añadimos la distribución teórica
curve(dnorm (x, mean = 2, sd = 3), col = "darkblue", lwd = 2, add =
TRUE)
```



Cuando hacemos el histograma utilizamos el argumento `prob=TRUE` para representar la densidad en lugar de las frecuencias absolutas. Hemos añadido la curva teórica de la distribución N(2,3) utilizando la instrucción `curve(dnorm(x, mean=2, sd=3), col="darkblue", lwd=2, add=TRUE)`. Notad que utilizamos la función `dnorm()` con el primer argumento, `x`, sin asignar a ningún objeto.

Para distribuciones discretas podemos construir un diagrama de frecuencias,

```
# establecemos la semilla para poder replicar resultados
set.seed(2020)
# simulamos las observaciones Pois(2)
X <- rpois(100000, 2)
# hacemos un histograma
hist(X, main = "Histograma 1000000 obs. Pois(2)",col = "grey", prob = TRUE)
# añadimos la distribución teórica: frecuencias
points(0:10,dpois (0:10, lambda = 2), col = "darkblue", pch = 20, cex = 2)
lines(0:10, dpois (0:10, lambda = 2), col = "darkblue", lwd = 2)
```