MLE

Juan Manuel Morales

2015-09-08

Máxima Verosimilitud (Likelihood)

objetivos:

- Relacionar el "likelihood" de datos con parámetros de una distribución
- Familiarizarse con curvas de likelihood y log-likelihood
- Comprobar que la forma de la curva de likelihood depende del tamaño muestreal
- Aprender a usar herramientas para encontrar óptimos

Vimos que las funciones de densidad nos dan la probabilidad de observar un valor determinado de una variable aleatoria (para variables continuas nos dan la probabilidad alrededor de un valor). Por ejemplo, para una distribución de Poisson, la probabilidad para un valor y es $\lambda^y \exp(-\lambda)/y!$ (donde ! es el factorial de y, no que y sea copado...). Entonces, para una variable aleatoria que asumimos tiene una distribución de Poisson con $\lambda = 2$ la probabilidad de observar un cero ($y = 0$) es de $2^0 \exp(-2)/0! = \exp(-2)$. En `R` usamos la función `dpois` para hacer este tipo de cuentas. Para este ejemplo escribimos `dpois(0, lambda=2)`.

Normalmente tenemos un conjunto de datos, por ejemplo números de renovales por metro cuadrado de muestreo en un bosque: $y = 1, 0, 2, 1, 0, 0, 1, 1, 2, 2$. Si asumimos que estos datos se pueden describir como el resultado de un proceso aleatorio de tipo Poisson, y que además **las observaciones son independientes**, podemos calcular la probabilidad del conjunto de datos como el producto de las probabilidades de las observaciones:

$$p(y) = \prod_i^n \frac{\lambda^{y_i} \exp(-\lambda)}{y_i!}$$

Podemos ver que el producto ($\prod$) se hace usándo i como índice para cada observación del set de datos y, desde $i = 1$ hasta $i = n$ que es el tamaño de y. Esta probabilidad de observar los datos es lo que llamamos *likelihood* o *verosimilitud*. Para hacer la cuenta de arriba tenemos que definir además el parámetro λ. La probabilidad de los datos depende entonces no sólo de los valores presentes en el set de datos sino también de la distribución utilizada y el valor del o los parámetros de la distribución. De manera más general, la expresión es un **modelo de datos**, es decir una descripción probabilística de como llegar a los datos. Más adelante veremos como combinando procesos determinísticos y estocásticos podemos armar modelos de datos realistas y satisfactorios.

Veamos ahora como calcular la probabilidad de estos datos en R asumiendo $\lambda = 1$:

```
y <- c(1, 0, 2, 1, 0, 0, 1, 1, 2, 2)
lambda <- 1
p.y <- dpois(y, lambda = lambda)
p.y
```

```
##  [1] 0.3678794 0.3678794 0.1839397 0.3678794 0.3678794 0.3678794 0.3678794
##  [8] 0.3678794 0.1839397 0.1839397
```

```
prod(p.y)
```

```
## [1] 5.674991e-06
```

Como las probabilidades son números que van entre 0 y 1, el producto de varias probabilidades resulta en números muy pequeños. Vemos en este caso que la probabilidad del **conjunto** de datos es muy baja (`5.67e-06`, donde `e-06` es 10^{-6} o 0.000001) pero esto no debería sorprendernos porque esa es la probabilidad de encontrar exactamente esos datos. Entonces, para evitar problemas numéricos preferimos trabajar con la suma de los logaritmos de las probabilidades, es decir con el logaritmo del *likelihood*. En R, podemos usar la opción `log = TRUE` dentro de las funciones de densidad.

```
y <- c(1, 0, 2, 1, 0, 0, 1, 1, 2, 2)
lambda <- 1
ll.y <- dpois(y, lambda = lambda, log = TRUE)
sum(ll.y)
```

```
## [1] -12.07944
```

Una vez que logramos hacer estas cuentas, podemos preguntarnos cómo cambia la probabilidad de observar el set de datos si cambiamos el valor de λ. Probemos con $\lambda = 2$

```
y <- c(1, 0, 2, 1, 0, 0, 1, 1, 2, 2)
lambda <- 2
sum(dpois(y, lambda = lambda, log = TRUE))
```

```
## [1] -15.14797
```

Con un λ de 2 la *log likelihood* de los datos empeora, pero podemos seguir probando distintos valores de λ hasta encontrar el que hace más posible al set de datos. Esta es la idea detrás de los análisis de máxima verosimilitud (a falta de mejor nombre) o de *maximum likelihood* (si estamos dispuestos a usar palabras en inglés). Normalmente tenemos más de un parámetro en nuestro modelo de datos, y usamos aglún algoritmo de búsqueda para encontrar la combinación de parámetros que hace más probables a nuestros datos (por tradición, los algoritmos buscan minimizar el menos logaritmo de la verosimilitud de nuestros datos).

Ejemplo: Remoción de Frutos

Mariano Rodriguez-Cabal estudió la remoción de frutos de quintral por parte del monito del monte en bosque continuo y fragmentos en 3 sitios pareados con la idea de ver si la fragmentación afectabla la interacción entre este marsupial y el muérdago. El trabajo está dispoible aquí

Los datos son de número de frutos removidos de un total disponible. Podemos usar una distribución Binomial para capturar la variabilidad (estocasticidad) en el proceso de remoción de frutos. Es decir, asumimos que la distribución Binomial tiene sentido para representar el proceso que genera los datos.

```
quintral <- read.table("~/cursoMyD/Data/quintral.txt", header = TRUE)
```

Para ver como están organizados los datos podemos hacer:

```
str(quintral)
```

```
## 'data.frame':    70 obs. of  4 variables:
##  $ bosque   : Factor w/ 2 levels "c","f": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sitio    : Factor w/ 3 levels "Campanario","Llao-Llao",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ Removidos: int  30 40 11 33 35 25 48 13 25 31 ...
##  $ Frutos   : int  46 51 32 49 46 37 53 31 33 44 ...
```

Vemos que se trata de un *data frame* que es la manera preferida de `R` para organizar datos. Los data frames tienen a las observaciones en filas y a las variables en columnas. Las variables individuales se pueden acceder escribiendo el nombre del data frame seguido del signo $ y el nombre de la variable. Por ejemplo, si queremos ver los valores de frutos disponibles en las tres primeras observaciones hacemos:

```
quintral$Frutos[1:3]
```

```
## [1] 46 51 32
```

Ahora calculemos el likelihood para la primera observación del set de datos asumiendo que la probabilidad de remoción es 0.5 (si no nos acordamos cómo es la distribución Binomial o cómo usarla en `R` podemos pedir ayuda: `?dbinom`)

```
dbinom(quintral$Removidos[1], size = quintral$Frutos[1], prob = 0.5)
```

```
## [1] 0.01408998
```

Normalmente trabajamos con log-likelihoods:

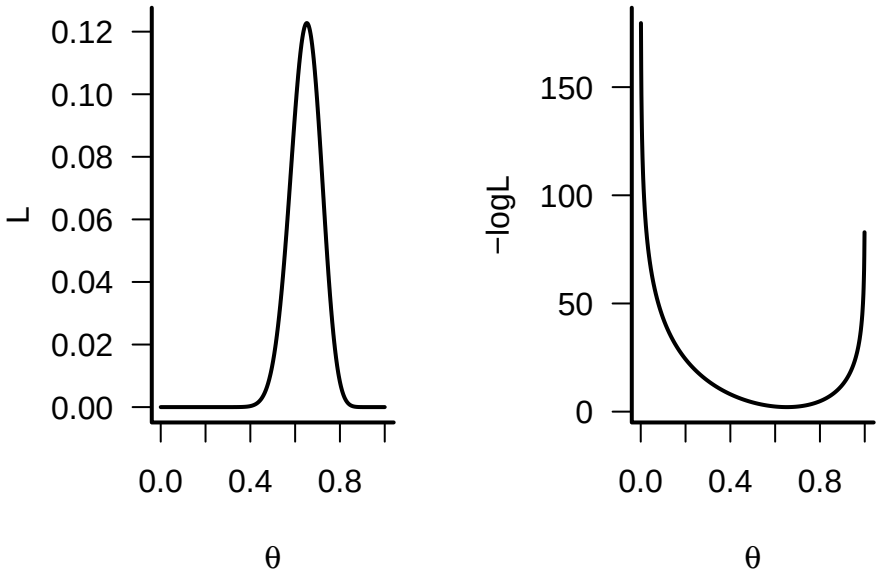
```r
dbinom(quintral$Removidos[1], size = quintral$Frutos[1], prob = 0.5, log = TRUE)
```

```
## [1] -4.262292
```

¿Cómo cambia la probabilidad de observar ese dato cuando cambiamos el parámetro de probabilidad de éxito (remoción) en la Binomial?

Como en este caso el likelihood depende de un solo parámetro, podemos ver esto gráficamente:

```
op <- par(mfrow = c(1, 2), lwd = 2, bty = "l", las = 1)
theta <- seq(0, 1, length = 1000)  # genera un vector de mil valores entre 0 y 1
plot(theta, dbinom(quintral$Removidos[1], size = quintral$Frutos[1], prob = theta,
    log = F), type = "l", ylab = "L", xlab = expression(theta))
plot(theta, -dbinom(quintral$Removidos[1], size = quintral$Frutos[1], prob = theta,
    log = T), type = "l", ylab = "-logL", xlab = expression(theta))
```

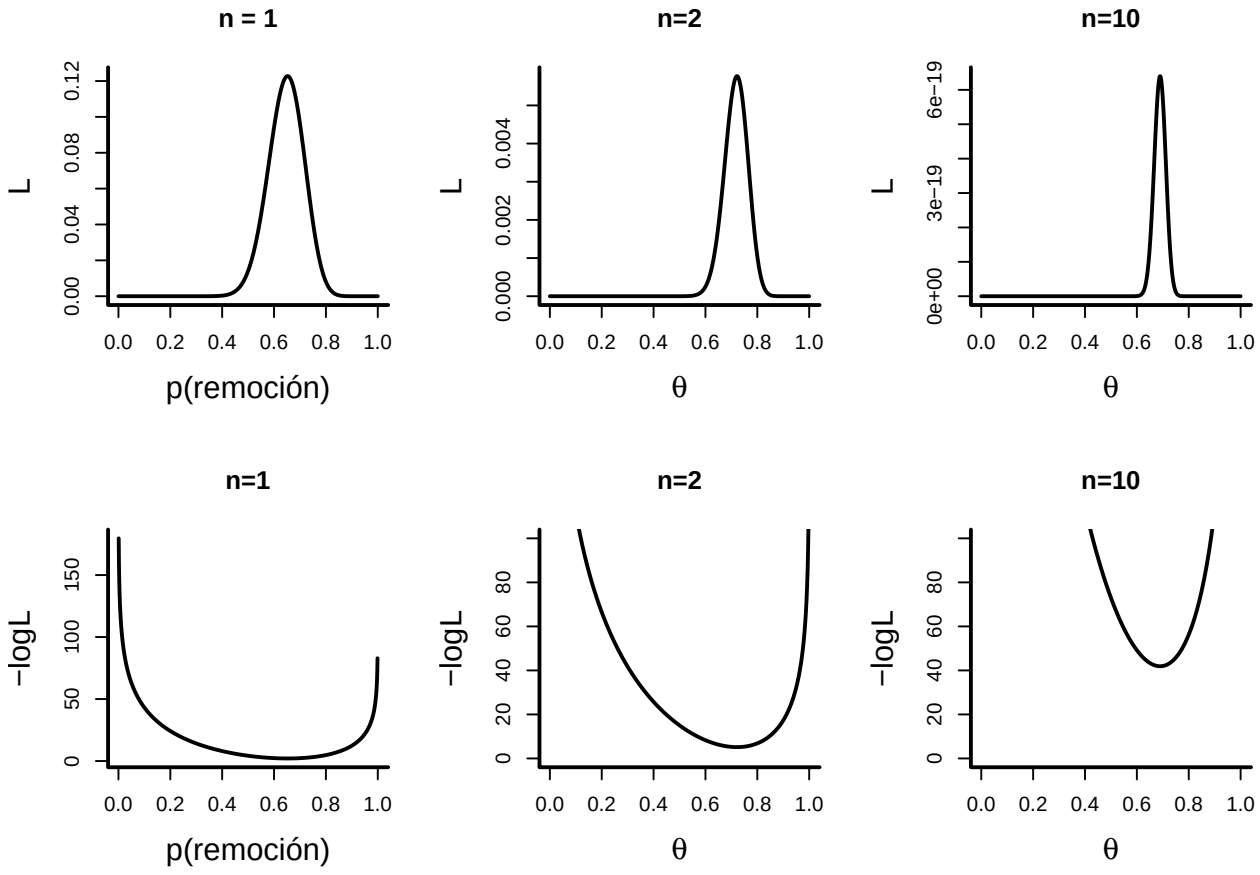



```
par(op)
```

También podemos ver cómo cambia el Likelihood cuando cambiamos el número de observaciones

```
op <- par(mfrow = c(2, 3), lwd = 2, bty = "l", cex.lab = 1.5)
theta <- seq(0, 1, length = 1000)
plot(theta, dbinom(quintral$Removidos[1], size = quintral$Frutos[1], prob = theta,
    log = F), type = "l", xlab = "p(remoción)", ylab = "L", main = "n = 1")
nLL2 <- array(0, length(theta))
for (i in 1:length(theta)) {
    nLL2[i] <- -sum(dbinom(quintral$Removidos[1:2], size = quintral$Frutos[1:2],
        prob = theta[i], log = TRUE))
}
plot(theta, exp(-nLL2), type = "l", xlab = expression(theta), ylab = "L", main = "n=2")

nLL10 <- array(0, length(theta))
```

```
for (i in 1:length(theta)) {
    nLL10[i] <- -sum(dbinom(quintral$Removidos[1:10], size = quintral$Frutos[1:10],
        prob = theta[i], log = TRUE))
}
plot(theta, exp(-nLL10), type = "l", xlab = expression(theta), ylab = "L", main = "n=10"
plot(theta, -dbinom(quintral$Removidos[1], size = quintral$Frutos[1], prob = theta,
    log = T), type = "l", xlab = "p(remoción)", ylab = "-logL", main = "n=1")
plot(theta, nLL2, type = "l", xlab = expression(theta), ylab = "-logL", main = "n=2",
    ylim = c(0, 100))
plot(theta, nLL10, type = "l", xlab = expression(theta), ylab = "-logL", main = "n=10",
    ylim = c(0, 100))
```




```
par(op)
```

Buscando el MLE

Queremos encontrar el valor de probabilidad de remoción por fruto que maximiza el likelihood (o minimiza el negative log-likelihood) del conjunto de datos. Una opción es hacerlo a lo bruto:

```r
p <- seq(0, 1, length = 100)  # defino un vector de probabilidades de remoción
nLL <- array(NA, length(p))  # genero un vector para guardar los valores de neg log li

for (i in 1:length(p)) {
    # loop sobre los valores de probabilidad de remoción
    nLL[i] <- -sum(dbinom(quintral$Removidos, size = quintral$Frutos, prob = p[i],
        log = TRUE))
}

# encontrar el valor de probabilidad de remoción que maximiza la likelihood
p[which(nLL == min(nLL))]
```

```
## [1] 0.5959596
```

En este caso, tenemos una solución analítica y podemos comparar:

```r
sum(quintral$Removidos)/sum(quintral$Frutos)
```

```
## [1] 0.5968072
```

Qué pasa si cambiamos la resolución en el vector de probabilidades de remoción?

Búsquedas más eficientes usando "optim" y "mle2" de Bolker

Primero tenemos que definir una función para nuestro 'negative log-likelihood' y luego usamos la función `optim` especificando los valores iniciales para la búsqueda, los datos y el método de búsqueda:

```r
binomNLL1 <- function(p, k, N) {
    -sum(dbinom(k, prob = p, size = N, log = TRUE))
    # k son los removidos, y N los frutos disponibles
}

O1 <- optim(fn = binomNLL1, par = c(p = 0.5), N = quintral$Frutos, k = quintral$Removido
    method = "BFGS")
```

Hay algunos "warnings" de NAs producidos en dbinom pero nada de qué preocuparse... El objeto de salida de `optim`, que en este caso se llama '`O1` contiene:

- en $par, el valor óptimo del parámetro "p"
- en $value, el valor del log likelihood para $par

- en $counts, hay algunas cosas crípticas acerca de cómo se desarrolló la búsqueda

Hay que asegurarse de que el algoritmo haya convergido ($convergence = 0)

La función `optim` en R es una función general de optimización, pero para MLE podemos usar `mle2` desarrollada por Ben Bolker. En `mle2` la función de negative log-likelihood se llama "minuslogl" en vez de "fn" (en `optim`).

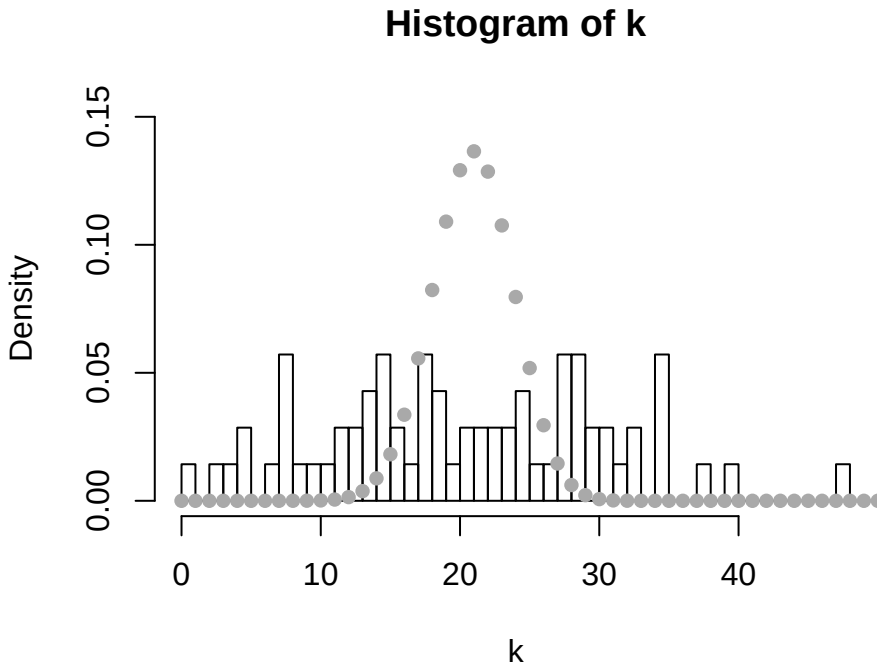
```
library(bbmle)  # cargar el paquete de MLE
`?`(mle2)

m1 <- mle2(minuslogl = binomNLL1, start = list(p = 0.5), data = list(N = quintral$Frutos
    k = quintral$Removidos))
```

En `m1` tenemos la función usada, los valores de los coeficientes y el log-likelihood. Podemos ver más detalles con `summary(m1)`

Ahora tenemos errores standares y valores de 'p' para los coeficientes y la devianza (-2 log-likelihood). Después veremos como usar éstas y otras salidas de `mle2` para obtener intervalos de confianza y hacer comparaciones entre modelos

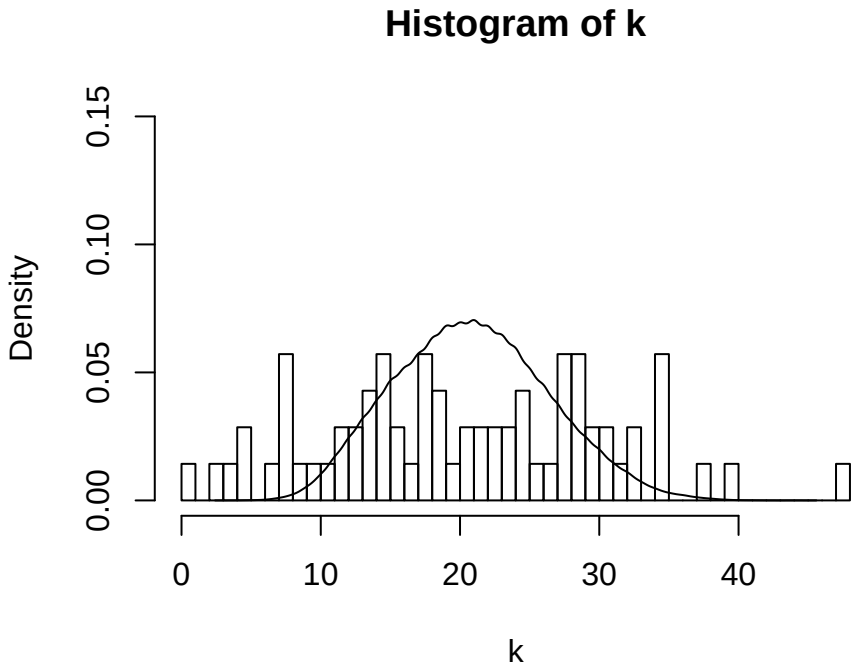
Comparación (cruda) con los datos

```
k <- quintral$Removidos
hist(k, breaks = 40, freq = F, ylim = c(0, 0.15))
points(0:50, dbinom(0:50, size = 35, prob = m1@coef), pch = 16, col = "darkgray")
```



Sin embargo, el número de frutos disponibles es variable y para una mejor comparación podemos usar un bootstrap (remuestreo)

```
res <- matrix(0, length(quintral$Frutos), 1000)
for (i in 1:1000) {
    n <- sample(quintral$Frutos, length(quintral$Frutos), replace = TRUE)
    res[, i] <- rbinom(length(quintral$Frutos), n, m1@coef)
}
hist(k, breaks = 40, freq = F, ylim = c(0, 0.15))
lines(density(res))
```



¿Qué se puede decir sobre el ajuste del modelo a los datos?

Ejercicio

1. Estimar las probabilidades de remoción de frutos por sitio y tipo de bosque.