Insumos de estimación

Nicolás Rivera Garzón

Abril, 2021

Problemas Algorítmicos: MLE

- Las estimaciones de máxima verosimilitud pueden no estar dadas explícitamente por fórmulas, sino implícitamente como soluciones de sistemas de ecuaciones no lineales.
- En el modelo de regresión clásico con matriz de diseño de rango completo, la fórmula del parámetro es fácil de escribir simbólicamente pero no es fácil de evaluar si el rango es grande

Método de Bisección

Si la verosimilitud es diferenciable y la ubicación del máximo global está en el interior del espacio de parámetros, encontrar la solución se reduce a encontrar el resultado de las ecuaciones normales de verosimilitud. Los pasos del método son:

1. Encontrar dos puntos, uno en el que la derivada está por debajo de cero y otro que está por encima.
2. Tomar la mitad del intervalo formado por estos puntos y cortarlo a la mitad.
3. Evaluar el valor de la derivada en el medio y reemplazar por el punto medio uno de los puntos originales que tiene el mismo signo del valor de la derivada.
4. Continuar hasta que la longitud del intervalo tenga la precisión deseada o se alcance el cero antes de eso.

Suponga que la función de verosimilitud es:

$$L(\theta) = \theta^{789}(1-\theta)^{211}$$

Y su log-verosimilitud es:

$$l(\theta) = 789\log\theta + 211\log(1-\theta)$$

Siguiendo el proceso manual de máxima verosimilitud el parámetro es $\hat{\theta} = 0,789$. Ahora bien, el proceso con el método de bisección se vería de la siguiente forma:

1. Definir la función de log-verosimilitud:

```
log_lik_0 <- function(theta) {
    l <- 789 * log(theta) + 211 * log(1 - theta)
    return(l)}
```

2. Definir la función para implementar el método de bisección:

```
## función para calcular el primer coeficiente diferencial
df <- function(f, x) {
    ## Args: f = función para ser diferenciada
    ##       x = ubicación donde se diferencia f
    ## Return: d = coeficiente diferencial
    h <- 0.0001
    d <- (f(x + h) - f(x)) / h
    return(d)
}
## Método de Bisección
mle_bisection <- function(theta_l, theta_u, ll, criterion = 0.1^10){
    ## Función para obtener MLE por bisección
    ## Args: theta_l = límite inferior de MLE
    ##       theta_u = límite superior de MLE
    ##       ll = función logarítmica de verosimilitud
    ##       criterion = criterio de convergencia
    ## Return: theta_hat = MLE

    counter <- 1  ## contador de iteraciones
    while (TRUE) {
        theta_m <- (theta_l + theta_u) / 2
        l_lower <- ll(theta_l)
        l_upper <- ll(theta_u)
        l_med <- ll(theta_m)
        first <- df(ll, theta_m)

        if (first == 0) {
            l_max <- l_med
            break
        } else if (first > 0) {
            theta_l <- theta_m
            epsilon <- l_med - l_upper
        } else {
            theta_u <- theta_m
            epsilon <- l_med - l_lower
        }

        ## Muestra el máximo de log-verosimilitud encontrado hasta ahora
        l_max <- max(c(l_lower, l_med, l_upper))
        cat("Iteration", counter, ": log likelihood =", l_max, "\n")

        ## romper el bucle si convergieron
        if (abs(epsilon) < criterion) break
        counter <- counter + 1
    }

    ## Encuentra el MLE y devuélvelo
    if (l_lower == l_max) theta_hat <- theta_l
    else if (l_med == l_max) theta_hat <- theta_m
    else theta_hat <- theta_u
    return(theta_hat)
}
```

3. Encontrar el parámetro con el supuesto inicial de $\hat{\theta} : 0,1 < \hat{\theta} < 0,9$:

```
mle_bisection(theta_l = .1, theta_u = .9, ll = log_lik_0)
```

```
## Iteration 1 : log likelihood = -568.9749
## Iteration 2 : log likelihood = -535.4548
## Iteration 3 : log likelihood = -515.6517
## Iteration 4 : log likelihood = -515.6517
## Iteration 5 : log likelihood = -515.6517
## Iteration 6 : log likelihood = -515.2853
## Iteration 7 : log likelihood = -515.2853
## Iteration 8 : log likelihood = -515.2853
## Iteration 9 : log likelihood = -515.2786
## Iteration 10 : log likelihood = -515.2786
## Iteration 11 : log likelihood = -515.2786
## Iteration 12 : log likelihood = -515.2786
## Iteration 13 : log likelihood = -515.2786
## Iteration 14 : log likelihood = -515.2786
## Iteration 15 : log likelihood = -515.2786
## Iteration 16 : log likelihood = -515.2786
## Iteration 17 : log likelihood = -515.2786
## Iteration 18 : log likelihood = -515.2786
## Iteration 19 : log likelihood = -515.2786
## Iteration 20 : log likelihood = -515.2786
## Iteration 21 : log likelihood = -515.2786
## Iteration 22 : log likelihood = -515.2786
## Iteration 23 : log likelihood = -515.2786
## Iteration 24 : log likelihood = -515.2786
## Iteration 25 : log likelihood = -515.2786
## Iteration 26 : log likelihood = -515.2786
## Iteration 27 : log likelihood = -515.2786
## Iteration 28 : log likelihood = -515.2786
## Iteration 29 : log likelihood = -515.2786
## Iteration 30 : log likelihood = -515.2786
## Iteration 31 : log likelihood = -515.2786
## Iteration 32 : log likelihood = -515.2786
```

```
## [1] 0.78895
```

Fueron necesarias 32 iteracciones para encontrar el estimador de máxima verosimilitud.

Algoritmo Newton-Raphson

- Es un algoritmo para encontrar aproximaciones de los ceros o raíces de una función real. También puede ser usado para encontrar el máximo o mínimo de una función, encontrando los ceros de su primera derivada.
- El método de Newton es un método abierto, en el sentido de que no está garantizada su convergencia global. La única manera de alcanzar la convergencia es seleccionar un valor inicial lo suficientemente cercano a la raíz buscada. Así, se ha de comenzar la iteración con un valor razonablemente cercano al cero (denominado punto de arranque o valor supuesto). Suponga que la misma función de verosimilitud anterior:

$$L(\theta) = \theta^{789}(1-\theta)^{211}$$

Y su log-verosimilitud:

$$l(\theta) = 789 \log \theta + 211 \log(1-\theta)$$

Con lo que el código para implementar NR es:

```
## Función para calcular el segundo coeficiente diferencial
ddf <- function(f, x){
    h <- -0.0001
    return((df(f, x+h) - df(f, x)) / h)
}
newton_mle <- function(ll, theta_0, delta = 0.1^12) {
    h <- 0.001
    theta_bar <- theta_0
    counter <- 1
    while (TRUE) {
        first <- df(ll, theta_bar)
        second <- ddf(ll, theta_bar)
        theta_0 <- theta_bar - first / second

        ## Muestra el máximo encontrado hasta ahora
        l_max <- ll(theta_0)
        cat("Iteration", counter, ": log likelihood =", l_max, "\n")

        ## romper el bucle si convergieron
        if (abs(theta_0 - theta_bar) < delta) break

        theta_bar <- theta_0
        counter = counter + 1
    }
    ## Encuentra el MLE y devuélvelo
    return(theta_0)
}
```

Ahora, para encontrar el parámetro, el supuesto inicial es $\hat{\theta} = 0,5$:

```
newton_mle(log_lik_0, theta_0 = .5)
```

```
## Iteration 1 : log likelihood = -515.2786
## Iteration 2 : log likelihood = -515.2786
## Iteration 3 : log likelihood = -515.2786
```

```
## [1] 0.78895
```

Mismo resultado con solo tres interacciones.

Algoritmo EM

Se usa en estadística para encontrar estimadores de máxima verosimilitud de parámetros en modelos probabilísticos que dependen de variables no observables. El algoritmo EM alterna pasos de esperanza (paso E), donde se computa la esperanza de la verosimilitud mediante la inclusión de variables latentes como si fueran observables, y un paso de maximización (paso M), donde se computan estimadores de máxima verosimilitud de los parámetros mediante la maximización de la verosimilitud esperada del paso E. Los parámetros que se encuentran en el paso M se usan para comenzar el paso E siguiente, y así el proceso se repite.

Suponga que tiene los datos $y_1, \ldots, y_n$ que se sacan de manera independiente de una combinación de normales con la densidad:

$$f(y \mid \theta) = \lambda\varphi(y \mid \mu_1, \sigma_1^2) + (1-\lambda)\varphi(y \mid \mu_2, \sigma_2^2)$$

donde el vector de parámetros es $\theta = (\mu_1, \mu_2, \sigma_1^2, \sigma_2^2, \lambda)$ y la log-verosimilitud es:

$$\log f(y_1, \ldots, y_n \mid \theta) = \log \sum_{i=1}^{n} \lambda\varphi(y_i \mid \mu_1, \sigma_1) + (1-\lambda)\varphi(y_i \mid \mu_2, \sigma_2)$$

Los datos son generados por el siguiente proceso:

```
mu1 <- 1
s1 <- 2
mu2 <- 4
s2 <- 1
n <- 100
lambda0 <- 0.4
z <- rbinom(n, 1, lambda0) # Porción S
y <- rnorm(n, mu1 * z + mu2 * (1-z), s1 * z + (1-z) * s2)
```

Ahora, asuma que μ_2, μ_2, σ_1 y σ_2 son conocidos, entonces se puede visualizar la función de log-verosimilitud como una función de $\lamda$:

```
## Función de verosimilitud
f <- function(y, lambda) {
        lambda * dnorm(y, mu1, s1) + (1-lambda) * dnorm(y, mu2, s2)
}

## Función de log-verosimilitud
loglike <- Vectorize(
        function(lambda) {
                sum(log(f(y, lambda)))
        }
)
```

La verosimilitud observada es relativamente sencilla, su solución se puede encontrar con la maximización directa, esto con el objetivo de comparar la precisión del algoritmo EM:

```
op <- optimize(loglike, c(0.01, 0.95), maximum = TRUE, tol = 1e-8)
op$maximum
```

```
## [1] 0.4750039
```

Ahora bien, el algoritmo para la estimación de lambda es el siguiente:

```
# Distribución
make_pi <- function(lambda, y, mu1, mu2, s1, s2) {
        lambda * dnorm(y, mu1, s1) / (lambda * dnorm(y, mu1, s1) +
                                              (1 - lambda) * (dnorm(y, mu2, s2)))
}

# Iteración individual del algoritmo EM
M <- function(lambda0) {
        pi.est <- make_pi(lambda0, y, mu1, mu2, s1, s2)
        mean(pi.est)
}
```

Con un 6 bucles se ejecuta el algoritmo EM y con el supuesto inicial de $\lambda = 1$:

```
lambda0 <- 0.1 # Supuesto inicial
lambda0star <- 0.1 # Supuesto inicial
iter <- 6 # Número de iteraciones
EM <-  numeric(iter)
for(i in 1:iter) {
        pihat <- make_pi(lambda0, y, mu1, mu2, s1, s2)
        lambda1 <- M(lambda0)
        EM[i] <- lambda1
        lambda0 <- lambda1
}
results <- data.frame(EM = EM, errorEM = abs(EM - op$maximum))
print(results)
```

```
##          EM     errorEM
## 1 0.3212639 0.153739965
## 2 0.4158530 0.059150872
## 3 0.4515759 0.023427933
## 4 0.4655686 0.009435210
## 5 0.4711759 0.003827957
## 6 0.4734460 0.001557834
```

Los errores van disminuyendo en cada iteracción.