Estimación de parámetros e inferencia bayesiana

Table of contents

Ideas Modelos Datos

0.1 Ajuste de modelos a datos

- Sin cuantificar incertidumbre: optimización.
- Cuantificando incertidumbre: inferencia estadística.
  - Frecuentista
  - Bayesiana

¿En qué contextos se usa cada enfoque?

En general, cuantificar incertidumbre es computacionalmente costoso y/o metodológicamente más desafiante que no hacerlo. Esto lleva a ignorar la incertidumbre cuando tenemos modelos muy complejos, como *deep neural networks*, o sets de datos muy grandes. Pero además, cuantificar o no incertidumbre en la estimación también depende del contexto. Cuando simplemente necesitamos un modelo que prediga bien y no nos preocupa mucho cómo lo hace, o cuando es difícil juzgar la performance de un modelo, cuantificar incertidumbre suele ser secundario. Por ejemplo, ¿cómo podríamos evaluar la incertidumbre de una respuesta de un modelo de lenguaje?; ¿cuánto nos preocupa la incertidumbre en la predicción de un modelo que genera sugerencias de contenido en Netflix?

0.2 Optimización

0.2.1 Minimización de funciones de costo

El enfoque más sencillo es minimizar una medida de discrepancia entre los datos y la predicción.
En general, llamamos a esta medida **función de costo** (*loss function*).

Consideremos un modelo muy simple:

$$\hat{y} = \lambda$$

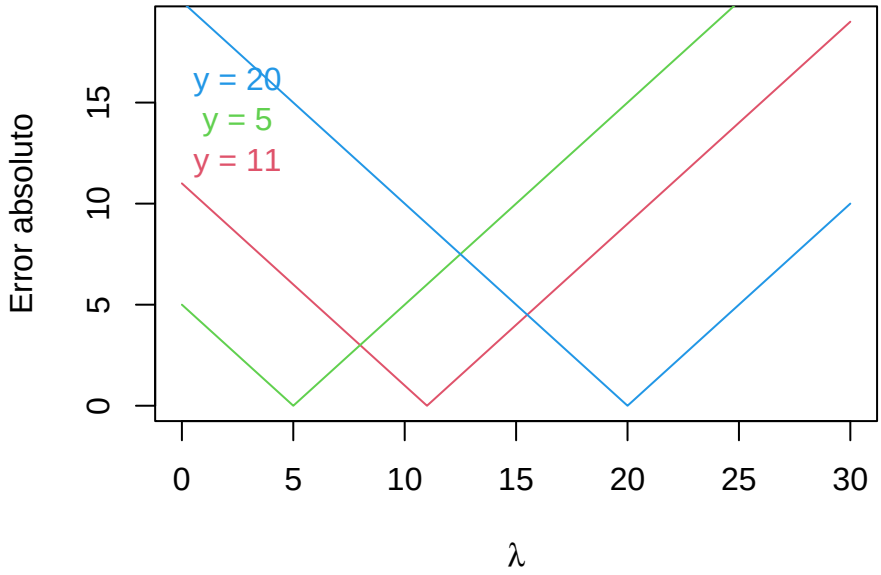
Con una sola observación, y_1, la discrepancia se minimiza en

$$\lambda = y_1$$

Tomamos como función de costo el error absoluto:
$|y_i - \hat{y}_i|$.

```
lambda_seq <- seq(0, 30, by = 0.25)

#  Error absoluto

y1 <- 11
d1 <- abs(y1 - lambda_seq)
plot(d1 ~ lambda_seq, type = "l", col = 2,
     xlab = expression(lambda), ylab = "Error absoluto")
text(2.5, 12, "y = 11", col = 2)
```

```r
y2 <- 5
d2 <- abs(y2 - lambda_seq)
lines(d2 ~ lambda_seq, col = 3)
text(2.5, 14, "y = 5", col = 3)

y3 <- 20
d3 <- abs(y3 - lambda_seq)
lines(d3 ~ lambda_seq, col = 4)
text(2.5, 16, "y = 20", col = 4)
```

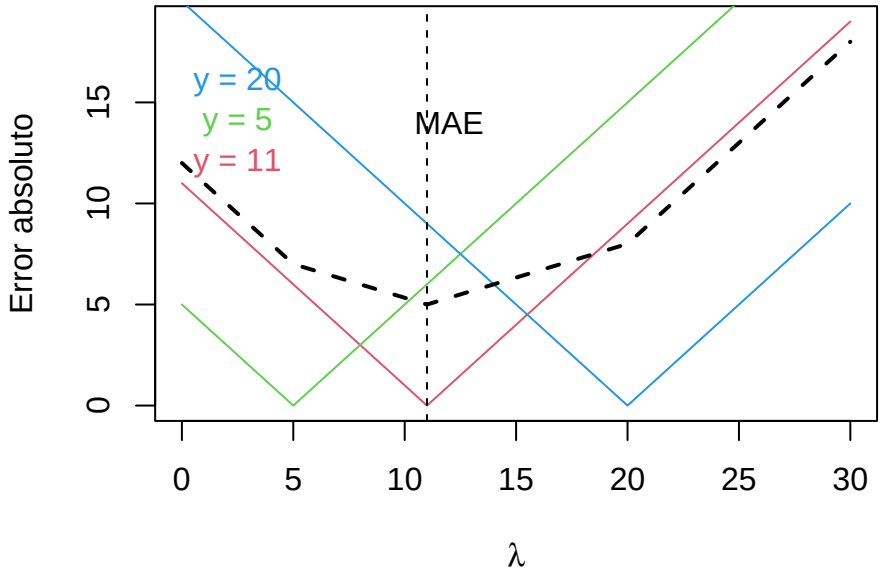


Con más de una observación, requerimos una métrica global.
Error Absoluto Medio (*MAE*):

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

```r
plot(d1 ~ lambda_seq, type = "l", col = 2,
     xlab = expression(lambda), ylab = "Error absoluto")
text(2.5, 12, "y = 11", col = 2)

lines(d2 ~ lambda_seq, col = 3)
text(2.5, 14, "y = 5", col = 3)

lines(d3 ~ lambda_seq, col = 4)
text(2.5, 16, "y = 20", col = 4)
```

```r
# MAE
mae <- rowMeans(cbind(d1, d2, d3))
lines(mae ~ lambda_seq, lwd = 2, lty = 2)
text(12, 14, "MAE")
abline(v = median(c(y1, y2, y3)), lty = 2)
```



¿Promedio o suma?

El promedio y la suma son operaciones equivalentes, ya que generalmente lo relevante es la forma de la función de costo, no su valor absoluto. El promedio es una transformación lineal de la suma (se multiplica por 1/N), y ninguna transformación lineal altera la forma. Es decir, podemos sumar los errores y multiplicarles cualquier constante o sumarles cualquier constante y no tendremos problemas.

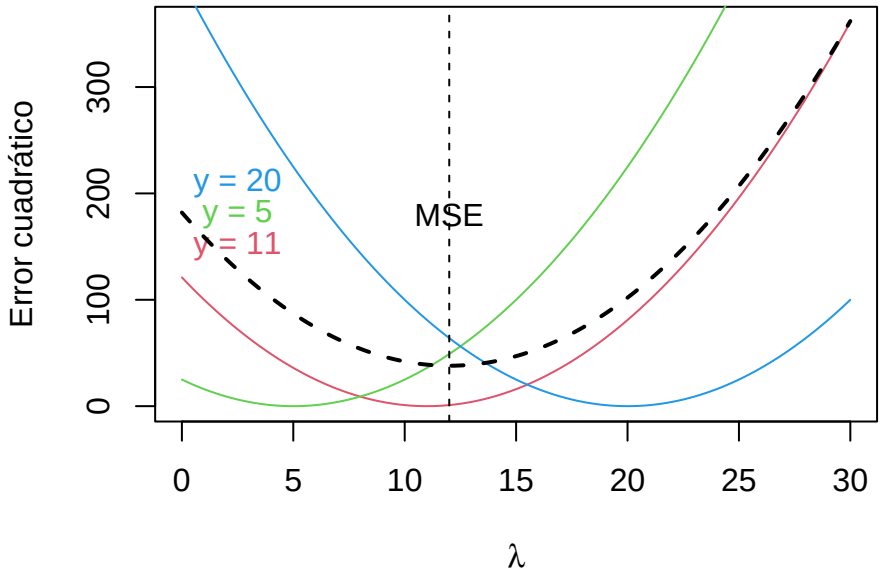
El MAE se minimiza en la mediana.
No es derivable (problemático).

Alternativa: error cuadrático

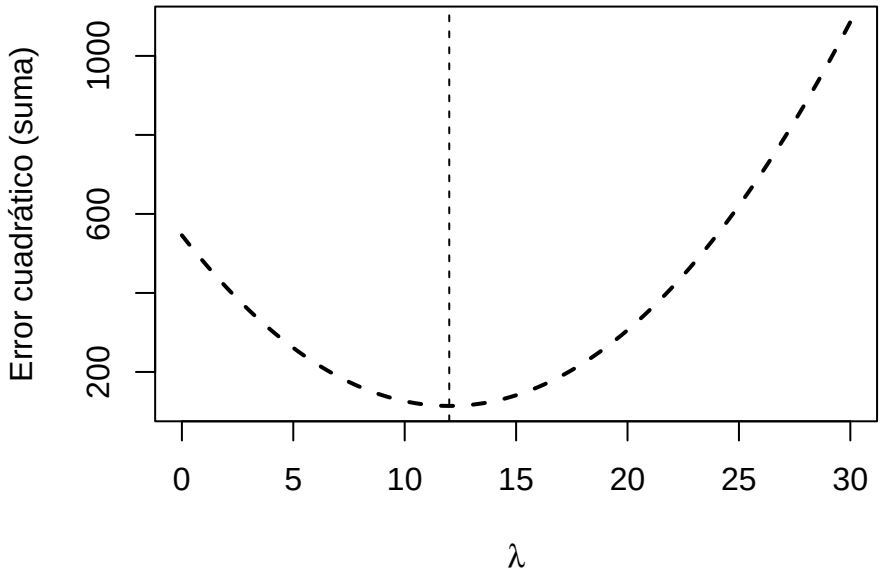
$$(y_i - \hat{y}_i)^2$$

```r
e1 <- (y1 - lambda_seq) ^ 2
plot(e1 ~ lambda_seq, type = "l", col = 2,
     xlab = expression(lambda), ylab = "Error cuadrático")
text(2.5, 150, "y = 11", col = 2)
```

```r
e2 <- (y2 - lambda_seq) ^ 2
lines(e2 ~ lambda_seq, col = 3)
text(2.5, 180, "y = 5", col = 3)

e3 <- (y3 - lambda_seq) ^ 2
lines(e3 ~ lambda_seq, col = 4)
text(2.5, 210, "y = 20", col = 4)

# error absoluto media
mse <- rowMeans(cbind(e1, e2, e3))
lines(mse ~ lambda_seq, lwd = 2, lty = 2)
text(12, 180, "MSE")
abline(v = mean(c(y1, y2, y3)), lty = 2)
```



```r
# Y da lo mismo si sumamos los errores en vez de promediar:
tse <- rowSums(cbind(e1, e2, e3))
plot(tse ~ lambda_seq, type = "l", lwd = 2, lty = 2,
     xlab = expression(lambda), ylab = "Error cuadrático (suma)")
abline(v = mean(c(y1, y2, y3)), lty = 2)
```



Detalles

El error cuadrático se minimiza en la media de los datos, y, al igual que con el MAE, da lo mismo si sumamos o promediamos los errores individuales.

0.2.2 Una función de costo elegante: la función de verosimilitud

Los modelos estadísticos definen de manera explícita la relación entre las predicciones y los datos.
Los datos son considerados realizaciones de una variable aleatoria, cuya distribución se modela. Ejemplo:

$$y_i \sim \text{Poisson}(\lambda)$$
$$y_1 = 11$$

- "y_i sigue una distribución Poisson con parámetro λ."
- "Definimos una función de verosimilitud Poisson."

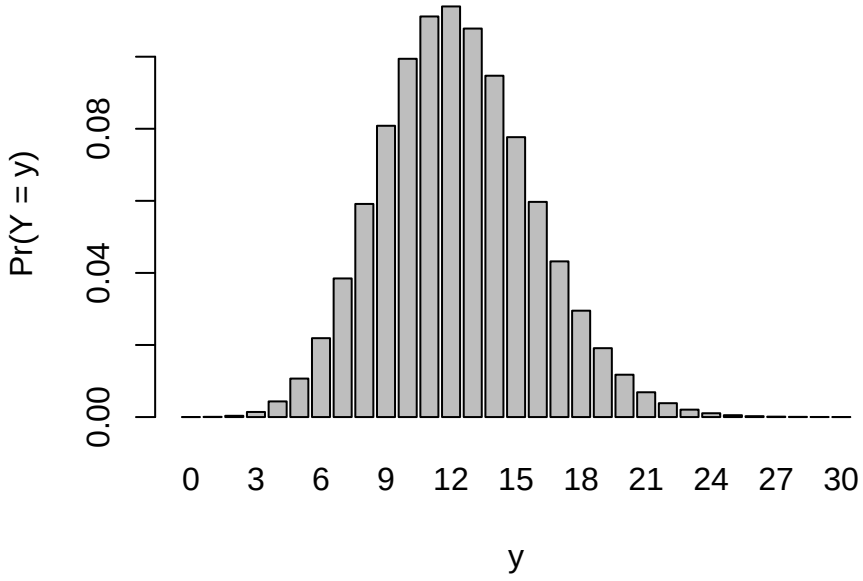
0.2.2.1 La verosimilitud no es una distribución

Función de masa de probabilidad de Poisson:

$$\Pr(Y = y) = \frac{\lambda^y e^{-\lambda}}{y!},$$
$$\text{con } y \in \{0, 1, 2, ...\}, \lambda \geq 0.$$

Tomando $\lambda = 12.3$, tiene esta forma:

```
y_seq <- 0:30
lambda <- 12.3
prob <- dpois(y_seq, lambda)
barplot(prob ~ y_seq,
        ylab = "Pr(Y = y)", xlab = "y",
        main = bquote("Función de masa de probabilidad para " * lambda == 12.3))
```



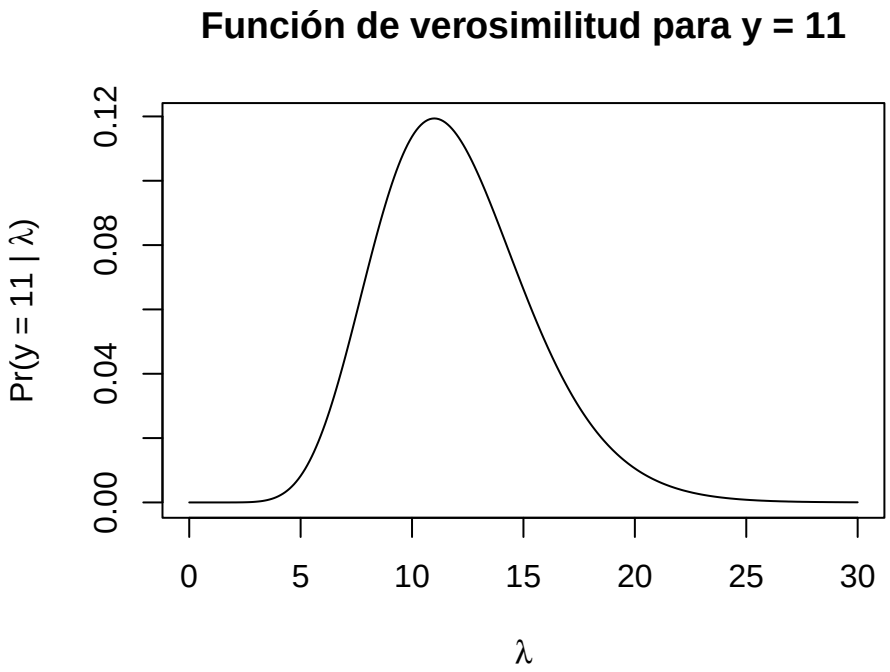
Nótese que la suma de las probabilidades tiende a 1:

```
sum(prob)
```

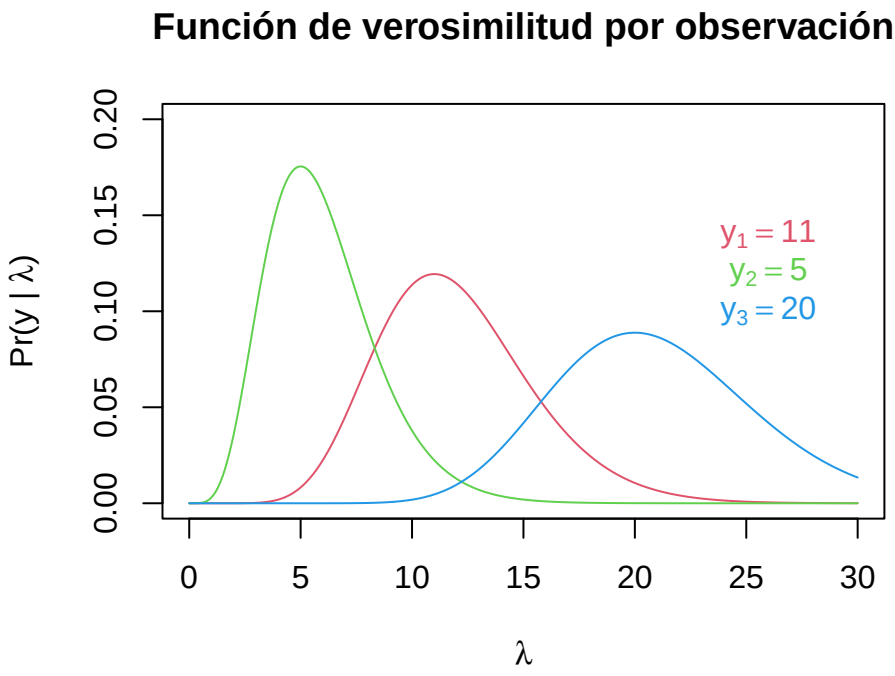
```
[1] 0.9999946
```

Usando la misma ecuación, podemos evaluar la probabilidad de algún valor puntual de y en función de λ:

```
y1 <- 11
lambda_seq <- seq(0, 30, by = 0.1)
l1 <- dpois(y1, lambda_seq)
plot(l1 ~ lambda_seq, type = "l",
     ylab = bquote("Pr(y = 11 | " * lambda * ")"), xlab = expression(lambda),
     main = "Función de verosimilitud para y = 11")
```



```r
y1 <- 11
lambda_seq <- seq(0, 30, by = 0.1)

l1 <- dpois(y1, lambda_seq)
plot(l1 ~ lambda_seq, type = "l", col = 2, ylim = c(0, 0.2),
     ylab = bquote("Pr(y | " * lambda * ")"), xlab = expression(lambda),
     main = "Función de verosimilitud por observación")
text(26, 0.14, expression(y[1] == 11), col = 2)

l2 <- dpois(y2, lambda_seq)
lines(l2 ~ lambda_seq, col = 3)
text(26, 0.12, expression(y[2] == 5), col = 3)

l3 <- dpois(y3, lambda_seq)
lines(l3 ~ lambda_seq, col = 4)
text(26, 0.10, expression(y[3] == 20), col = 4)
```



Con una sola observación, podemos visualizar ambas funciones a la vez. (Gracias, `plotly`)

```
lambda_seq_short <- seq(0, 15, by = 0.5)
y_max <- 20
d <- expand.grid(lambda = lambda_seq_short, y = 0:y_max)
d$prob <- dpois(d$y, d$lambda)
d$yfac <- factor(as.character(d$y), levels = as.character(0:y_max))

# Convert to wide format for z matrix
z_matrix <- acast(d, lambda ~ y, value.var = "prob")

# Step 2: Create interactive 3D surface with all requested tweaks
plot_ly(
  x = 0:y_max,
  y = lambda_seq_short,
  z = z_matrix,
  type = "surface",
  showscale = FALSE,  # (2) Remove the legend (color bar)
  colorscale = "Viridis",
  hovertemplate = paste(
    "y: %{x}<br>",
    "\u03BB: %{y}<br>",
```

```
    "Pr: %{z:.3f}<extra></extra>" # 3 decimales
  ),
  contours = list(
    # x = list(show = TRUE, highlight = TRUE, color = "black", width = 0.2),
    # y = list(show = TRUE, highlight = TRUE, color = "black", width = 0.2,
    #          start = 5, end = 15, size = 5)
    x = list(show = TRUE, highlight = F),
    y = list(show = TRUE, highlight = F,
             start = 5, end = 15, size = 5)
  )
) %>%
  layout(
    scene = list(
      xaxis = list(
        title = "y",
        tickmode = "linear",
        autorange = "reversed",
        showspikes = FALSE     # disable hover lines
      ),
      yaxis = list(
        title = "\u03BB",  # Greek lambda
        autorange = "reversed",
        showspikes = FALSE
      ),
      zaxis = list(title = "Pr(Y = y)", showspikes = FALSE)
    ),
    title = "Distribución Poisson:\nverosimilitud y función de masa de probabilidad"
  )
```

- Función de masa de probabilidad: se fija λ y varía y.
- Función de verosimilitud: se fija y y varía λ.

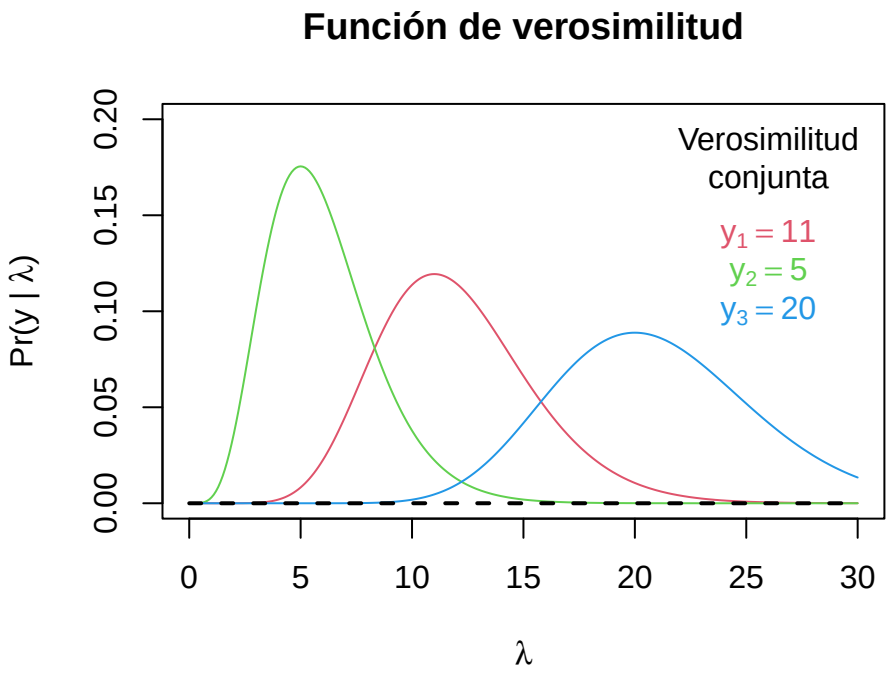
Verosimilitud conjunta asumiendo observaciones independientes:

$$L(y_1, y_2, y_3 \mid \lambda) = \Pr(y_1 \mid \lambda) \times \Pr(y_2 \mid \lambda) \times \Pr(y_2 \mid \lambda).$$

Y de forma más general,

$$L(y \mid \lambda) = \prod_{i=1}^{N} \Pr(y_i \mid \lambda).$$

```r
y1 <- 11
lambda_seq <- seq(0, 30, by = 0.1)

l1 <- dpois(y1, lambda_seq)
plot(l1 ~ lambda_seq, type = "l", col = 2, ylim = c(0, 0.2),
     ylab = bquote("Pr(y | " * lambda * ")"), xlab = expression(lambda),
     main = "Función de verosimilitud")
text(26, 0.14, expression(y[1] == 11), col = 2)

l2 <- dpois(y2, lambda_seq)
lines(l2 ~ lambda_seq, col = 3)
text(26, 0.12, expression(y[2] == 5), col = 3)

l3 <- dpois(y3, lambda_seq)
lines(l3 ~ lambda_seq, col = 4)
text(26, 0.10, expression(y[3] == 20), col = 4)

# verosimilitud conjunta:
lfull <- l1 * l2 * l3
lines(lfull ~ lambda_seq, lwd = 2, lty = 2)
text(26, 0.18, "Verosimilitud\nconjunta")
```

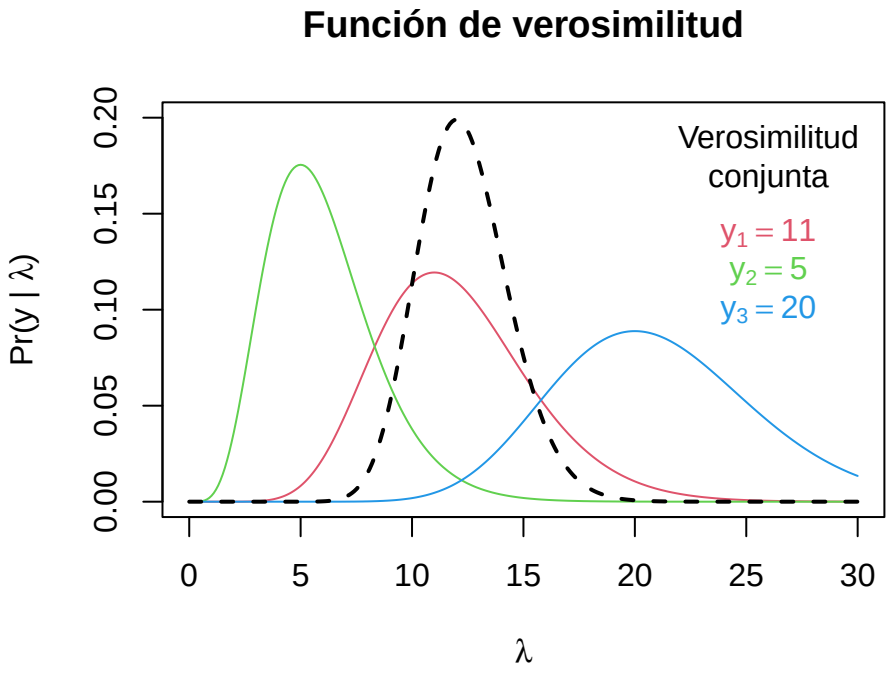


¡Es plana!

La escalamos

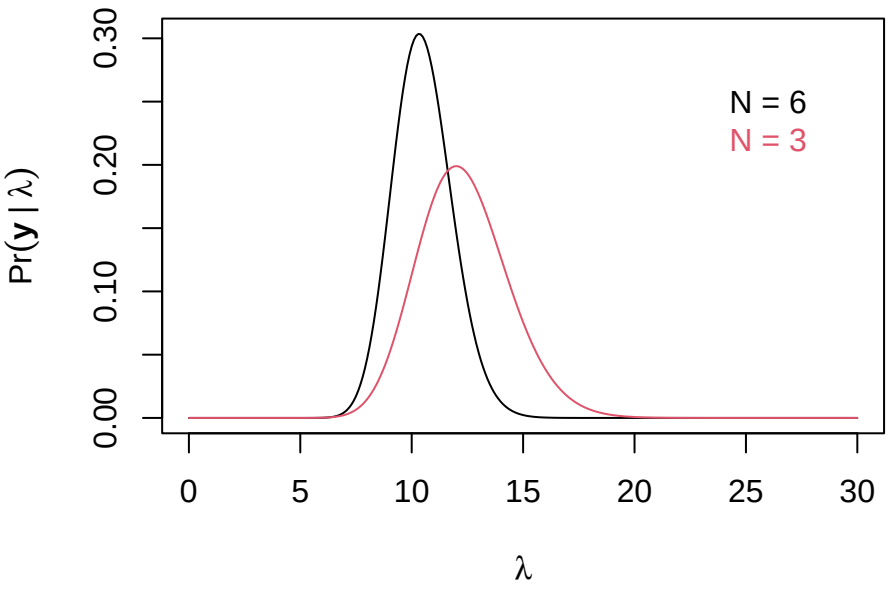
```
y1 <- 11
lambda_seq <- seq(0, 30, by = 0.1)

l1 <- dpois(y1, lambda_seq)
plot(l1 ~ lambda_seq, type = "l", col = 2, ylim = c(0, 0.2),
     ylab = bquote("Pr(y | " * lambda * ")"), xlab = expression(lambda),
     main = "Función de verosimilitud")
text(26, 0.14, expression(y[1] == 11), col = 2)

l2 <- dpois(y2, lambda_seq)
lines(l2 ~ lambda_seq, col = 3)
text(26, 0.12, expression(y[2] == 5), col = 3)

l3 <- dpois(y3, lambda_seq)
lines(l3 ~ lambda_seq, col = 4)
text(26, 0.10, expression(y[3] == 20), col = 4)

# verosimilitud conjunta:
lfull <- l1 * l2 * l3
a <- diff(lambda_seq)[1]
```

```r
lfull_norm <- normalize_dens(lfull, a) # sólo la escalamos, para que se vea
lines(lfull_norm ~ lambda_seq, lwd = 2, lty = 2)
text(26, 0.18, "Verosimilitud\nconjunta")
```



La función de verosimilitud se va volviendo más puntuda a medida que agregamos datos:

```r
y <- c(y1, y2, y3, 4, 10, 12) # lo extendemos

# vector con verosimilitud para cada valor de lambda_seq
l <- numeric(length(lambda_seq))
for (i in 1:length(lambda_seq)) {
  l[i] <- prod(dpois(y, lambda_seq[i]))
}

l_norm <- normalize_dens(l, diff(lambda_seq)[1])
plot(l_norm ~ lambda_seq, type = "l",
     xlab = expression(lambda), ylab = expression(Pr(bold(y) ~ "|" ~ lambda)))
lines(lfull_norm ~ lambda_seq, col = 2)
text(26, 0.25, "N = 6")
text(26, 0.22, "N = 3", col = 2)
```



Optimización:

Buscamos el valor de λ que maximice la verosimilitud:
estimador de máxima verosimilitud.
(*maximum likelihood estimate, MLE*)

Generalmente se trabaja con la log-verosimilitud.

Recordemos que

$$\log(a \times b) = \log(a) + \log(b).$$

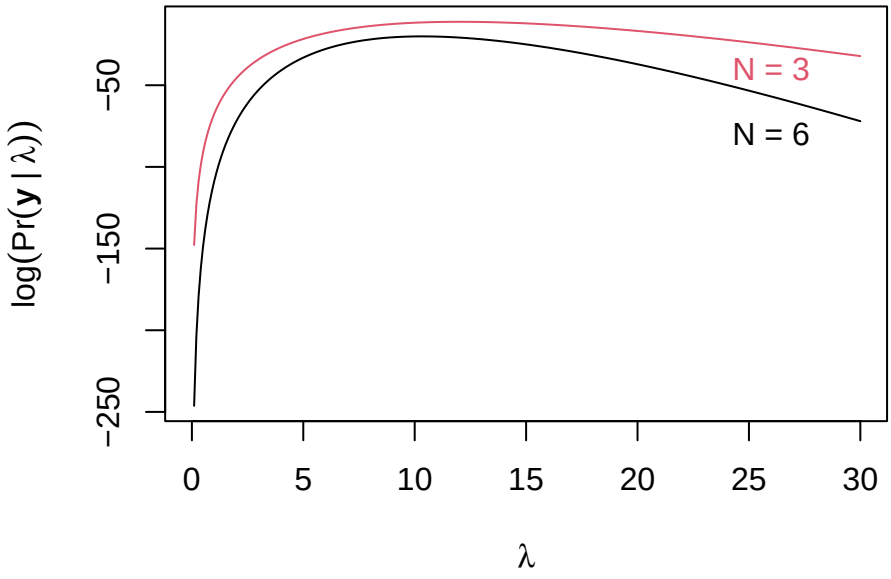
Entonces, la log-verosimilitud conjunta se define así:

$$\log[L(y_1, y_2, y_3 \mid \lambda)] = \log[\Pr(y_1 \mid \lambda)] + \log[\Pr(y_2 \mid \lambda)] + \log[\Pr(y_2 \mid \lambda)].$$

Y generalizando,

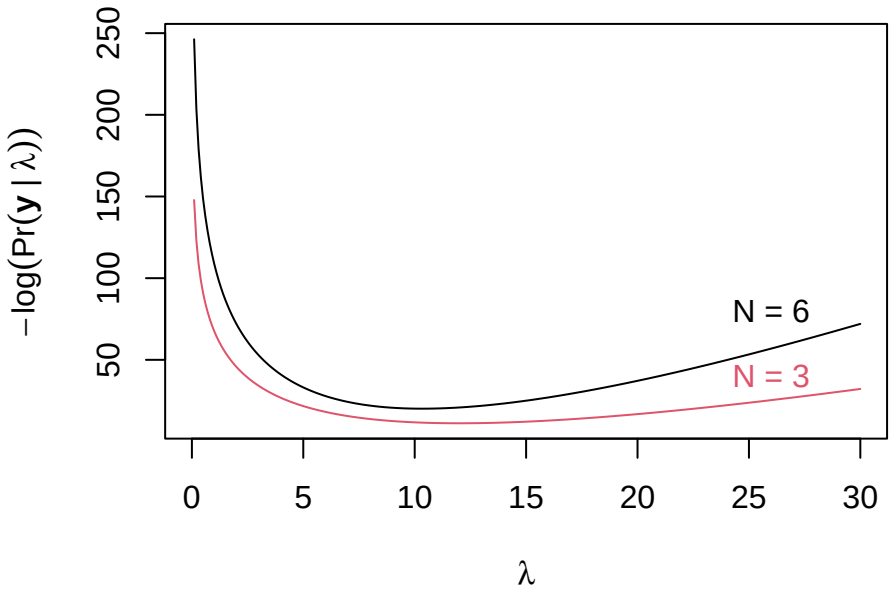
$$\log[L(y \mid \lambda)] = \sum_{i=1}^{N} \log[\Pr(y_i \mid \lambda)].$$

```r
ll3 <- log(lfull) # la que calculamos con 3 datos
ll6 <- log(l) # la que calculamos con 6 datos

rr <- c(ll3, ll6)
rr <- range(rr[is.finite(rr)])

plot(ll6 ~ lambda_seq, type = "l",
     xlab = expression(lambda),
     ylab = expression(log(Pr(bold(y) ~ "|" ~ lambda))),
     ylim = rr)
lines(ll3 ~ lambda_seq, col = 2)
text(26, -80, "N = 6")
text(26, -40, "N = 3", col = 2)
```



Y si la negamos, nos interesa minimizarla:

```r
plot(-ll6 ~ lambda_seq, type = "l",
     xlab = expression(lambda),
     ylab = expression(-log(Pr(bold(y) ~ "|" ~ lambda))),
     ylim = rr[2:1] * (-1))
lines(-ll3 ~ lambda_seq, col = 2)
text(26, 80, "N = 6")
text(26, 40, "N = 3", col = 2)
```



En un contexto de optimización, los modelos estadísticos son una forma elegante de elegir la función de costo, definida como la -log verosimilitud.

0.2.2.2 Máxima verosimilitud en 2D (ejemplo)

Para tener un ejemplo real, podemos modelar el número de incendios por verano en función de un índice de peligrosidad de incendios, llamado FWI (datos de Barberá et al. 2025).

$$y_i \sim \text{Poisson}(\lambda_i)$$
$$\lambda_i = \exp(\alpha + \beta\ FWI_i)$$

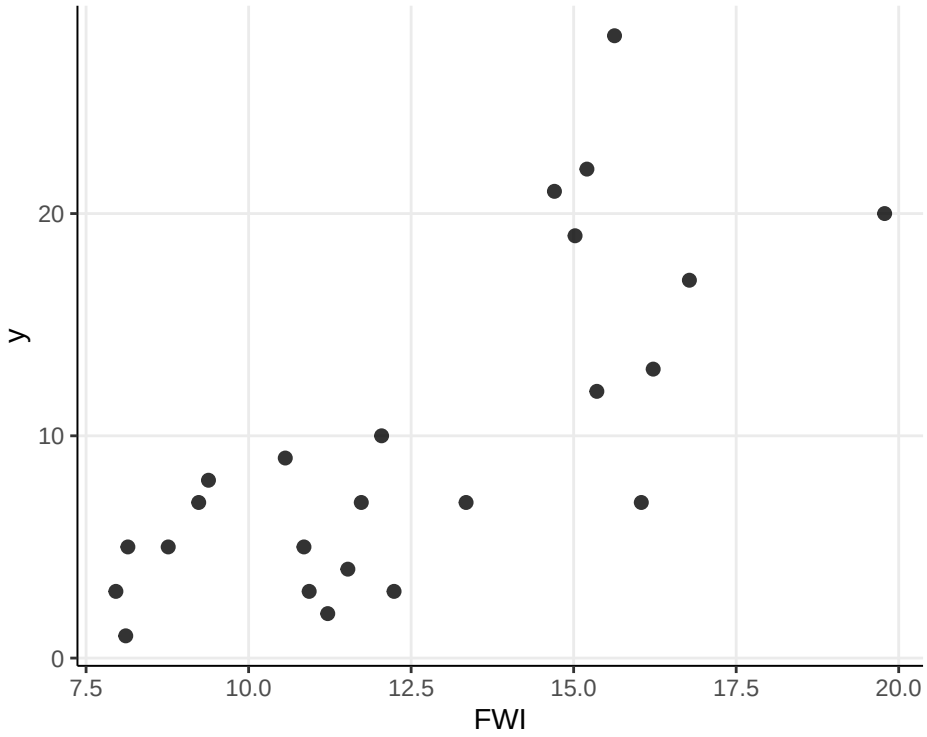
y_i : número de incendios $\geq$ 10ha en el verano i
FWI_i : Fire Weather Index en el verano i

λ no se estima, es una *cantidad derivada*.
Sólo necesitamos estimar α y β.

Cargamos datos y graficamos.

```
datos <- read.csv(here::here("datos", "barbera_data_fire_total_climate.csv"))
```

```r
ggplot(datos, aes(x = fwi, y = fires)) +
  geom_point(color = rgb(0, 0, 0, 0.8), size = 2) +
  ylab("y") + xlab("FWI") +
  nice_theme()
```



Función de verosimilitud, ahora depende de α y β
(uso interactivo, vaya a `R`).

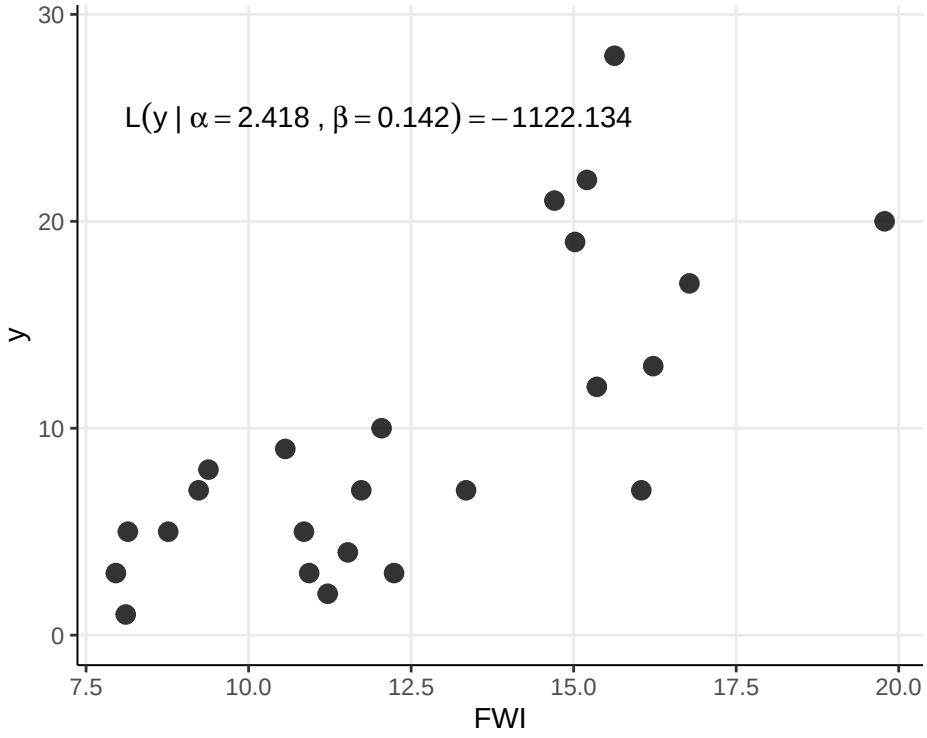
```r
like_fire <- function(alpha, beta, log = T) {
  # "like" por likelihood (sería raro decirle "vero");
  # "fire" porque es la función para estos datos de fuego, no cualquier
  # verosimilitud.

  # Calculamos la media, lambda:
  lambda <- exp(alpha + beta * datos$fwi)

  # usando lambda evaluamos la verosimilitud de cada observación,
  # en escala log:
  like_pointwise <- dpois(datos$fires, lambda, log = T)

  # la log-verosimilitud conjunta es la suma de las log-verosimilitudes por
```

```
  # observación:
  like <- sum(like_pointwise)

  if (log) return(like)
  else return(exp(like))
}
```

Ahora podemos darle valores a α y β y evaluar la verosimilitud.
También graficamos la curva definida por esos valores.
(Uso interactivo, vaya a R.)

```
## Podemos elegir valores a mano
# alpha <- log(2)
# beta <- 0.01

# O más interesante, muestrear en cierto rango
alpha <- runif(1, log(0.01), log(30))
beta <- runif(1, 0, 0.2)

like <- like_fire(alpha, beta)

# Label para el gráfico
label_text <- as.character(
  as.expression(
    bquote(L(y ~ "|" ~ alpha == .(round(alpha, 3)) ~ "," ~ beta == .(round(beta, 3))) == .(l
  )
)

# Plot de puntos + curva
ggplot(datos, aes(x = fwi, y = fires)) +
  geom_point(color = rgb(0, 0, 0, 0.8), size = 3) +
  stat_function(fun = function(x) exp(alpha + beta * x),
                color = "blue", linewidth = 1) +
  ylab("y") + xlab("FWI") +
  annotate("text", x = 12, y = 25,
           label = label_text,
           parse = TRUE, hjust = 0.5) +
  ylim(min(datos$fires - 1), max(datos$fires + 1)) +
  xlim(min(datos$fwi), max(datos$fwi)) +
  nice_theme()
```

```
Warning: Removed 101 rows containing missing values or values outside the scale range
(`geom_function()`).
```

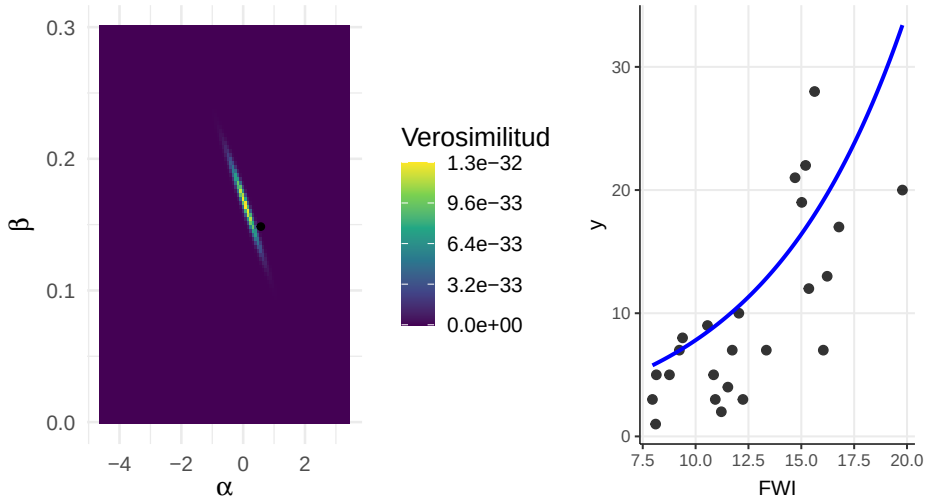


Mejor que probar valores al azar es buscar de forma más ordenada. El método más simple, aunque ineficiente, es buscar en una grilla.

```
# Creamos grilla de valores de alpha y beta.
side <- 100 # cantidad de valores por lado
grilla <- expand.grid(
  alpha = seq(log(0.01), log(30), length.out = side),
  beta = seq(0, 0.3, length.out = side)
)

# Evaluamos la verosimilitud para cada valor
grilla$loglike <- NA
size <- side ^ 2 # cantidad de valores en la grilla
for (i in 1:size) {
  grilla$loglike[i] <- like_fire(grilla$alpha[i], grilla$beta[i])
}
grilla$like <- exp(grilla$loglike)
```

Visualizamos la grilla y evaluamos puntos.

```r
# Elegimos un punto para evaluar
row <- sample(1:size, 1)
alpha <- grilla$alpha[row]
beta <- grilla$beta[row]

# Grilla
p1 <- ggplot(grilla, aes(x = alpha, y = beta, fill = like)) +
  geom_tile() +
  geom_point(data = grilla[row, , drop = F]) +
  scale_fill_viridis_c(option = "viridis", name = "Verosimilitud",
                       label = scientific_format(digits = 2)) +
  labs(x = expression(alpha), y = expression(beta)) +
  theme_minimal(base_size = 14)

# Plot de puntos + curva
p2 <- ggplot(datos, aes(x = fwi, y = fires)) +
  geom_point(color = rgb(0, 0, 0, 0.8), size = 2) +
  stat_function(fun = function(x) exp(alpha + beta * x),
                color = "blue", linewidth = 1) +
  ylab("y") + xlab("FWI") +
  nice_theme()

# Unimos
(p1 | p2)
```



Y como estimador puntual tenemos el valor con mayor verosimilitud:

```r
row_max <- which.max(grilla$loglike)
grilla[row_max, ]
```

```
          alpha     beta   loglike         like
5658 0.004556595 0.169697 -73.43519 1.280858e-32
```

Pero este resultado depende fuertemente de la resolución y extensión de la grilla. Una forma más robusta es utilizar un optimizador.

0.2.2.3 Optimización con `optim`

Esta función toma como argumento una función que evalúa un vector de parámetros x, así que creamos un wrapper para `like_fire`.

```r
like_fire_opt <- function(x) {
  alpha <- x[1]
  beta <- x[2]
  return(-like_fire(alpha, beta))
}
```

Devuelve la -log verosimilitud porque `optim` minimiza.

Corremos el optimizador

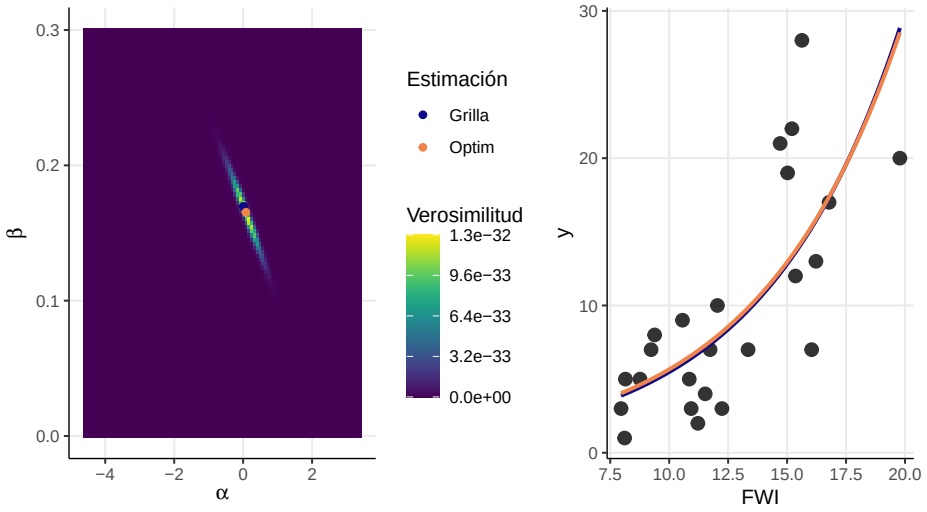
```r
opt <- optim(
  par = c(0, 0.1),    # vector inicial de parámetros, donde comienza la búsqueda
  fn = like_fire_opt, # función a optimizar
  method = "BFGS"
)
opt
```

```
$par
[1] 0.08094103 0.16529256

$value
[1] 73.39016

$counts
function gradient
      48        9

$convergence
[1] 0

$message
```

```
NULL
```

Comparamos las estimaciones

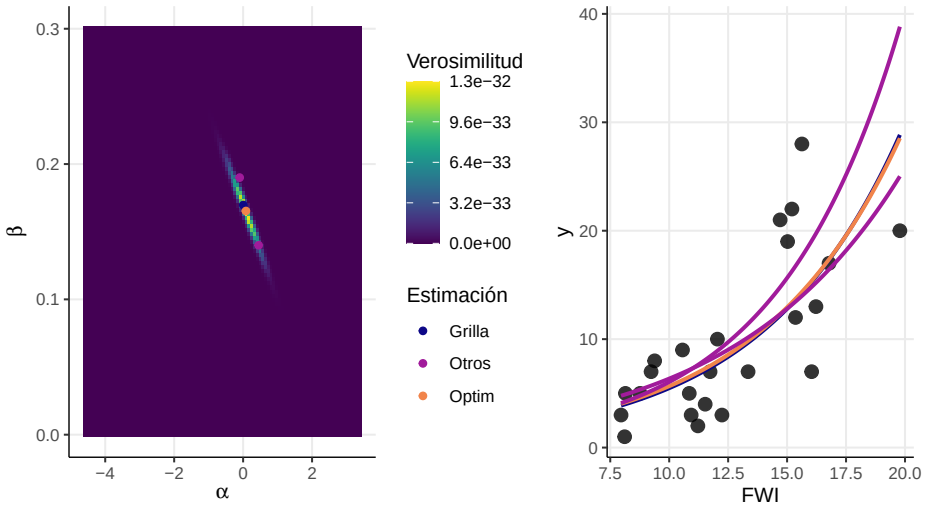
```r
# Tomamos los óptimos según la grilla y optim
b_hat <- grilla[c(row_max, row_max), c("alpha", "beta")]
b_hat[2, "alpha"]<- opt$par[1]
b_hat[2, "beta"] <- opt$par[2]
b_hat$estimacion <- c("Grilla", "Optim")

# Grilla con estimadores puntuales
p1 <- ggplot(grilla, aes(x = alpha, y = beta, fill = like)) +
  geom_tile() +
  scale_fill_viridis(option = "D", name = "Verosimilitud",
                     label = scientific_format(digits = 2)) +
  geom_point(data = b_hat,
             mapping = aes(x = alpha, y = beta, colour = estimacion),
             inherit.aes = F) +
  scale_color_viridis(option = "C", name = "Estimación", end = 0.7,
                      discrete = T) +
  labs(x = expression(alpha), y = expression(beta)) +
  nice_theme()

# Plot de puntos + curva
cc <- viridis(2, option = "C", end = 0.7) # colores

p2 <- ggplot(datos, aes(x = fwi, y = fires)) +
  geom_point(color = rgb(0, 0, 0, 0.8), size = 3) +
  stat_function(fun = function(x) exp(b_hat$alpha[1] + b_hat$beta[1] * x),
                color = cc[1], linewidth = 1) +
  stat_function(fun = function(x) exp(b_hat$alpha[2] + b_hat$beta[2] * x),
                color = cc[2], linewidth = 1) +
  ylab("y") + xlab("FWI") +
  nice_theme()

# Unimos, usando el paquete patchwork
(p1 | p2)
```



Pero hay otras combinaciones de parámetros que también andan más o menos bien:

```
# Tomamos los óptimos según la grilla y optim
b_hat2 <- b_hat
b_hat2$estimacion <- "Otros"

b_hat2$alpha <- c(-0.1, 0.45)
b_hat2$beta <- c(0.19, 0.14)

b_all <- rbind(b_hat, b_hat2)
b_all$estimacion <- factor(b_all$estimacion,
                           levels = c("Grilla", "Otros", "Optim"))

# Grilla con estimadores puntuales y otros
p1 <- ggplot(grilla, aes(x = alpha, y = beta, fill = like)) +
  geom_tile() +
  scale_fill_viridis(option = "D", name = "Verosimilitud",
                     label = scientific_format(digits = 2)) +
  geom_point(data = b_all,
             mapping = aes(x = alpha, y = beta, colour = estimacion),
             inherit.aes = F) +
  scale_color_viridis(option = "C", name = "Estimación", end = 0.7,
                      discrete = T) +
  labs(x = expression(alpha), y = expression(beta)) +
  nice_theme()

# Plot de puntos + curva
```

```r
cc <- viridis(3, option = "C", end = 0.7) # colores

p2 <- ggplot(datos, aes(x = fwi, y = fires)) +
  geom_point(color = rgb(0, 0, 0, 0.8), size = 3) +
  stat_function(fun = function(x) exp(b_all$alpha[1] + b_all$beta[1] * x),
                color = cc[1], linewidth = 1) +
  stat_function(fun = function(x) exp(b_all$alpha[2] + b_all$beta[2] * x),
                color = cc[3], linewidth = 1) +
  stat_function(fun = function(x) exp(b_all$alpha[3] + b_all$beta[3] * x),
                color = cc[2], linewidth = 1) +
  stat_function(fun = function(x) exp(b_all$alpha[4] + b_all$beta[4] * x),
                color = cc[2], linewidth = 1) +
  ylab("y") + xlab("FWI") +
  nice_theme()

# Unimos
(p1 | p2)
```



0.3 Trabajo Práctico 5.1

Actividades 1 a 6 (incluida).

0.4 Estimación cuantificando incertidumbre (estadística)

0.4.1 Frecuentismo

La probabilidad es una frecuencia de eventos cuando el número de repeticiones tiende a infinito.

0.4.2 Bayesianismo

La probabilidad es una medida de cuánto creemos en una idea o evento. Las distribuciones representan nuestro conocimiento.

0.5 Estimación cuantificando incertidumbre (estadística)

0.5.1 Frecuentismo

Utiliza la probabilidad para evaluar cómo se comportarían ciertas métricas si repitiéramos el estudio muchas veces.

0.5.2 Bayesianismo

Utiliza la probabilidad para representar nuestro conocimiento (o desconocimiento) sobre los parámetros.

0.6 Estimación cuantificando incertidumbre (estadística)

0.6.1 Frecuentismo

- Estimadores puntuales
- Distribución muestral alrededor del valor verdadero
- Intervalos de confianza y pruebas de hipótesis

0.6.2 Bayesianismo

- Distribución previa de los parámetros
- Distribución posterior de los parámetros

0.7 Estimación cuantificando incertidumbre (estadística)

0.7.1 Frecuentismo

$$\widehat{\theta} = \text{argmax}_{\theta}\ L(y|\theta)$$

0.7.2 Bayesianismo

$$p(\theta|y) \propto L(y|\theta)\ p(\theta)$$

De nuestro amigo Simon Wood:

La diferencia entre los enfoques puede parecer enorme, y ha habido mucho debate sobre cuál es el menos feo. Sin embargo, desde una perspectiva práctica, los enfoques tienen mucho en común, salvo quizás cuando se trata de la selección de modelos. En particular, si se aplican correctamente, suelen producir resultados que difieren menos entre sí de lo que probablemente difieren los modelos analizados con respecto a la realidad.

[Wood, Simon. 2015. Core Statistics. Cambridge University Press.]

0.8 Inferencia bayesiana

- Nuestro conocimiento sobre los parámetros se representa con distribuciones de probabilidad. Es decir, los tratamos como variables aleatorias.

- Para pensar en un modelo y la distribución de sus parámetros no necesitamos estrictamente tener datos.

- Si tenemos información nueva (datos), actualizamos nuestro conocimiento. En este caso, llamamos *previa* a la distribución que antecede a la evidencia, *posterior* a la distribución actualizada, y la evidencia nueva está codificada en la *verosimilitud*.

Teorema de Bayes:

$$\Pr(A \mid B) = \frac{\Pr(B \mid A) \times \Pr(A)}{\Pr(B)}$$

Aplicado a Estadística:

$$\Pr(\text{parámetros} \mid \text{datos}) = \frac{\Pr(\text{datos} \mid \text{parámetros}) \times \Pr(\text{parámetros})}{\Pr(\text{datos})}$$

---

posterior = verosimilitud $\times$ previa / probabilidad de los datos

$$p(\theta \mid y) = \frac{L(y \mid \theta)\ p(\theta)}{\int L(y \mid \theta)\ p(\theta)\ \mathrm{d}\theta}$$

posterior $\propto$ verosimilitud $\times$ previa

$$p(\theta \mid y) \propto L(y \mid \theta)\ p(\theta)$$

0.8.1 Ejemplo en 1D

$$y_i \sim \text{poisson}(\lambda)$$
$$\lambda \sim \text{gama}(\alpha, \beta)$$

$$y = \{11, 5, 20\}$$
$$\alpha = \ ...$$
$$\beta = \ ...$$

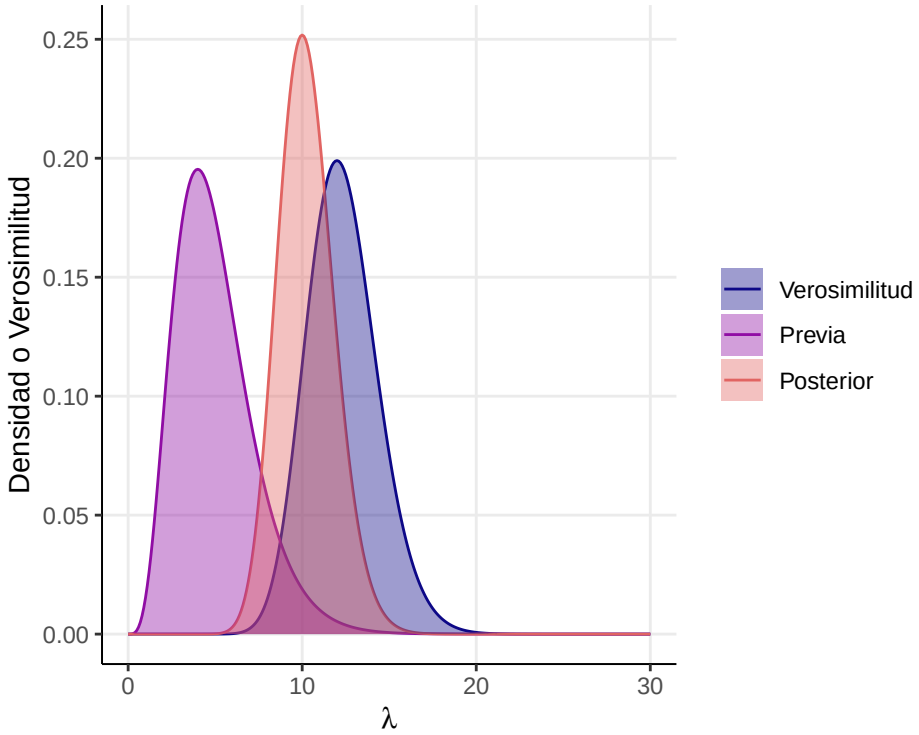
$$p(\lambda \mid y) \propto L(y \mid \lambda)\ p(\lambda)$$

$$p(\lambda \mid y) \propto \prod_i^N \text{poisson}(y_i \mid \lambda)\ \text{gama}(\lambda \mid \alpha, \beta)$$

[Uso interactivo, vaya a R.]

```
y <- c(11, 5, 20)
lambda_seq <- seq(0, 30, by = 0.1)
ns <- length(lambda_seq)

# Calcularemos verosimilitud (like), previa y posterior sobre esa secuencia
# de lambda
d0 <- data.frame(
  lambda = lambda_seq,
  like = NA,
  prior = NA,
  post = NA
)

# definimos parámetros de la previa
mu <- 5      # media
sigma2 <- 5  # varianza

# obtenemos alpha y beta, que son los parámetros que requiere dgamma
alpha <- mu ^ 2 / sigma2
beta <- mu / sigma2
```

```r
# calculamos verosimilitud, previa y posterior
for (i in 1:ns) {
  d0$like[i] <- prod(dpois(y, lambda_seq[i]))
  d0$prior[i] <- dgamma(lambda_seq[i], alpha, beta)
  #d0$prior[i] <- dunif(lambda_seq[i], 0, 15) # previa plana
}
d0$post <- d0$like * d0$prior

# Normalizamos para poder visualizar
d <- d0
a <- diff(lambda_seq)[1] # para normalizar
for (v in c("like", "prior", "post")) {
  d[, v] <- normalize_dens(d0[, v], a)
}
d$zero <- 0

# Elongamos d
cols <- which(names(d) %in% c("like", "prior", "post"))
dlong <- pivot_longer(d, all_of(cols), names_to = "variable",
                      values_to = "density")
dlong$variable <- factor(
  dlong$variable,
  levels = c("like", "prior", "post"),
  labels = c("Verosimilitud", "Previa", "Posterior")
)

# Graficamos
ggplot(dlong, aes(x = lambda, y = density, ymax = density, ymin = 0,
                  color = variable, fill = variable)) +
  geom_line() +
  geom_ribbon(alpha = 0.4, color = NA) +
  scale_color_viridis(discrete = T, option = "C", end = 0.6) +
  scale_fill_viridis(discrete = T, option = "C", end = 0.6) +
  theme(legend.title = element_blank()) +
  xlab(expression(lambda)) +
  ylab("Densidad o Verosimilitud") +
  nice_theme()
```



0.8.2 Ejemplo en 2D (incendios)

$$
\begin{aligned}
y_i &\sim \text{poisson}(\lambda_i) \\
\lambda_i &= \exp(\alpha + \beta\ FWI_i)
\end{aligned}
$$

$$
\begin{aligned}
\alpha &\sim \text{normal}(\mu_\alpha, \sigma_\alpha) \\
\beta &\sim \text{normal}(\mu_\beta, \sigma_\beta)
\end{aligned}
$$

$$
\begin{aligned}
\mu_\alpha &= \ ..., \sigma_\alpha = \ ... \\
\mu_\beta &= \ ..., \sigma_\beta = \ ...
\end{aligned}
$$

Densidad posterior no normalizada:

$$p(\alpha, \beta \mid y) \propto L(y \mid \alpha, \beta)\ p(\alpha)\ p(\beta)$$

$$p(\alpha, \beta \mid y) \propto \prod_i^N \text{poisson}(y_i \mid \alpha, \beta)\ \text{normal}(\alpha \mid \mu_\alpha, \sigma_\alpha)\ \text{normal}(\beta \mid \mu_\beta, \sigma_\beta)$$

Y en log:

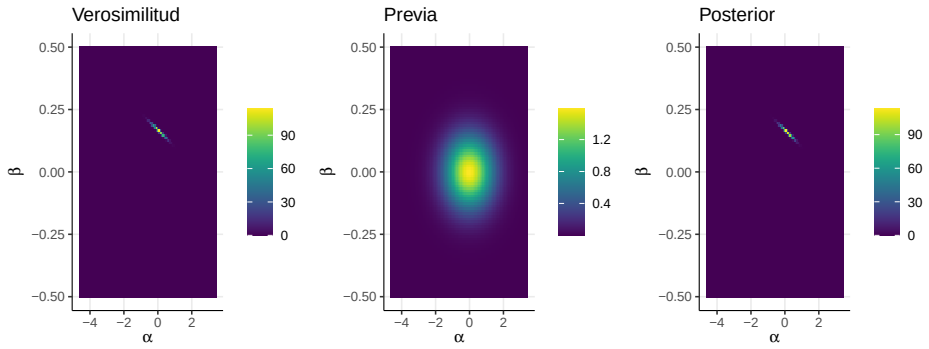
$$\log[p(\alpha, \beta \mid y)] \doteq \log[L(y \mid \alpha, \beta)] + \log[p(\alpha)] + \log[p(\beta)]$$

$$\log[p(\alpha, \beta \mid y)] \doteq \sum_{i}^{N} \log[\text{poisson}(y_i \mid \alpha, \beta)] + \log[\text{normal}(\alpha \mid \mu_\alpha, \sigma_\alpha)] + \log[\text{normal}(\beta \mid \mu_\beta, \sigma_\beta)]$$

[Uso interactivo, vaya a R.]

```r
# Creamos grilla de valores de alpha y beta.
side <- 100 # cantidad de valores por lado

alpha_seq <- seq(log(0.01), log(30), length.out = side)
beta_seq <- seq(-0.5, 0.5, length.out = side)

grilla <- expand.grid(
  alpha = alpha_seq,
  beta = beta_seq,
  loglike = NA,   # para llenar luego
  like = NA,
  prior = NA,
  post = NA
)

# Evaluamos la verosimilitud para cada valor
size <- side ^ 2 # cantidad de valores en la grilla
for (i in 1:size) {
  grilla$loglike[i] <- like_fire(grilla$alpha[i], grilla$beta[i])
}
grilla$like <- exp(grilla$loglike)

# Definimos y evaluamos la previa
mu_a <- 0; sigma_a <- 1
mu_b <- 0; sigma_b <- 0.1

for (i in 1:size) {
  grilla$prior[i] <-
    dnorm(grilla$alpha[i], mu_a, sigma_a) *
    dnorm(grilla$beta[i], mu_b, sigma_b)
}

# Posterior
grilla$post <- grilla$like * grilla$prior
```

```r
# Normalizamos para poder visualizar
g <- grilla
a <- diff(alpha_seq)[1] * diff(beta_seq)[1] # para normalizar
for (v in c("like", "prior", "post")) {
  g[, v] <- normalize_dens(grilla[, v], a)
}

# Graficamos por separado, para apreciar las distintas escalas
p1 <-
  ggplot(g, aes(x = alpha, y = beta, fill = like)) +
    geom_tile() +
    scale_fill_viridis() +
    theme(legend.title = element_blank()) +
    xlab(expression(alpha)) +
    ylab(expression(beta)) +
    ggtitle("Verosimilitud") +
    nice_theme()

p2 <-
  ggplot(g, aes(x = alpha, y = beta, fill = prior)) +
    geom_tile() +
    scale_fill_viridis() +
    theme(legend.title = element_blank()) +
    xlab(expression(alpha)) +
    ylab(expression(beta)) +
    ggtitle("Previa") +
    nice_theme()

p3 <-
  ggplot(g, aes(x = alpha, y = beta, fill = post)) +
    geom_tile() +
    scale_fill_viridis() +
    theme(legend.title = element_blank()) +
    xlab(expression(alpha)) +
    ylab(expression(beta)) +
    ggtitle("Posterior") +
    nice_theme()

(p1 | p2 | p3)# + plot_layout(nrow = 2)
```



¿Cómo definir en R una función que evalúe la densidad posterior?

```r
post_fire <- function(alpha, beta,
                      mu_a = 0, sigma_a = 1, # parámetros de la previa
                      mu_b = 0, sigma_b = 1,
                      log = T) {

  ## Log Verosimilitud (mismo código que en like_fire)

  # Calculamos la media, lambda:
  lambda <- exp(alpha + beta * datos$fwi)
  # usando lambda evaluamos la verosimilitud de cada observación,
  # en escala log:
  like_pointwise <- dpois(datos$fires, lambda, log = T)
  # la log-verosimilitud conjunta es la suma de las log-verosimilitudes por
  # observación:
  like <- sum(like_pointwise)

  ## Log Previas
  lprior_a <- dnorm(alpha, mean = mu_a, sd = sigma_a, log = T)
  lprior_b <- dnorm(beta, mean = mu_b, sd = sigma_b, log = T)

  ## Log Posterior
  lpost <- like + lprior_a + lprior_b

  if (log) return(lpost)
  else return(exp(lpost))
}
```

0.9 Trabajo Práctico 5.2

Actividades 7 a 11.

0.10 Ajuste frecuentista hegemónico a mano, usando `optim`

La práctica más usual para calcular valores P o intervalos de confianza en el enfoque frecuentista implica suponer que la función de log verosimilitud tiene forma cuadrática, centrada en el máximo. Esto se cumple si los parámetros no se encuentran cerca de un límite (e.g., una varianza muy cercana a cero) y si el N es grande en relación a la cantidad de parámetros.

La información sobre la curvatura de la log verosimilitud se encuentra en la matriz hessiana, que es la matriz de derivadas parciales de segundo orden de la log verosimilitud evaluadas en el máximo. `optim` la calcula si le indicamos `hessian = TRUE`:

```
opt <- optim(
  par = c(0, 0.1),    # vector inicial de parámetros, donde comienza la búsqueda
  fn = like_fire_opt, # función a optimizar
  method = "BFGS",
  hessian = TRUE
)
opt
```

```
$par
[1] 0.08094103 0.16529256

$value
[1] 73.39016

$counts
function gradient
      48        9

$convergence
[1] 0

$message
NULL

$hessian
          [,1]      [,2]
[1,]  238.0263  3394.741
[2,] 3394.7410 50913.778
```

Si repitiéramos el estudio (incluyendo este análisis) infinitas veces, siempre tomando un N grande, obtendríamos una muestra de estimadores de máxima verosimilitud (una por estudio), los cuales seguirían una Normal Multivariada alrededor del valor verdadero de los parámetros. A partir de esto, se desprende un método para construir intervalos de confianza y calcular valores P, que utilizamos a continuación:

```
I <- solve(opt$hessian) # obtenemos la inversa
I
```

```
             [,1]          [,2]
[1,]  0.085634146 -0.0057097657
[2,] -0.005709766  0.0004003469
```

Con esta matriz, podemos construir una tabla de resumen análoga a las que arrojan funciones como `glm`.

```
sds <- sqrt(diag(I)) # error estándar de los estimadores
zs <- opt$par / sds  # valor z
ps <- (1 - pnorm(abs(zs), mean = 0, sd = 1)) * 2 # valor P, a dos colas

summ_table <- data.frame(
  par = c("alpha", "beta"),
  est = opt$par,
  sd = sds,
  z = zs,
  p = ps
)

names(summ_table) <- c("Parameter", "Estimate", "Std. Error",
                       "z value",  "Pr(>|z|)")
summ_table
```

```
  Parameter   Estimate Std. Error   z value     Pr(>|z|)
1     alpha 0.08094103 0.29263313 0.2765956 7.820907e-01
2      beta 0.16529256 0.02000867 8.2610459 2.220446e-16
```

```
# Comparamos con glm
glm_fit <- glm(fires ~ fwi, data = datos, family = "poisson")
summary(glm_fit)
```

```
Call:
glm(formula = fires ~ fwi, family = "poisson", data = datos)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
```

```
-2.3930  -1.3925  -0.3374   0.8986   3.1816

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.08042    0.29267   0.275    0.783
fwi          0.16532    0.02001   8.261   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 122.571  on 23  degrees of freedom
Residual deviance:  54.035  on 22  degrees of freedom
AIC: 150.78

Number of Fisher Scoring iterations: 4
```

0.11 Máxima verosimilitud penalizada: casi una previa

Si en vez de concentrarnos en obtener la distribución posterior completa nos conformamos únicamente con obtener el modo, como estimador puntual, podríamos tomar la -log densidad posterior y minimizarla. La -log densidad posterior, es la suma de la -log verosimilitud, la -log previa y el -log de la constante normalizadora (C):

$$\begin{aligned} p(\theta \mid y) &= L(y \mid \theta) \times p(\theta) \times C \\ \log\left[p(\theta \mid y)\right] &= \log[L(y \mid \theta)] + \log[p(\theta)] + \log(C) \\ -\log\left[p(\theta \mid y)\right] &= -\log[L(y \mid \theta)] - \log[p(\theta)] - \log(C). \end{aligned}$$

De esta manera, nuestro estimador puntual puede ser

$$\hat{\theta}_{MAP} = \text{argmin}_{\theta}\{-\log[L(y \mid \theta)] - \log[p(\theta)]\},$$

En donde ignoramos C porque no varía en función de los parámetros, y el sufijo MAP viene de *maximum a posteriori*, el modo de la posterior.

Esta expresión es muy similar a un estimador de máxima verosimilitud penalizada (mínima -log verosimilitud), en donde se introduce un término de penalidad K:

$$\hat{\theta}_{PML} = \text{argmin}_{\theta}\{-\log[L(y \mid \theta)] + K\},$$

Con PML por *penalized maximum likelihood*.
La penalidad K suele utilizarse para evitar que un modelo muy complejo se

sobreajuste a los datos. En un contexto de regresión múltiple, con muchas predictoras, K se define como una función creciente del valor absoluto de los coeficientes (β_j). Por ejemplo, en LASSO (*Least Absolute Shrinkage and Selection Operator*), K se define así:

$$K_{LASSO} = \lambda \sum_{j=1}^{J} |\beta_j|,$$

siendo $\lambda > 0$ un parámetro de penalización usualmente escogido mediante validación cruzada. Esta forma de penalización también es conocida como L1. Desde una perspectiva bayesiana, introducir esta penalidad equivale a definir una previa de Laplace (doble exponencial) centrada en cero sobre los coeficientes:

$$\beta_j \sim \text{laplace}\left(0, \frac{1}{\lambda}\right).$$

De todos modos, recordemos que estimar simplemente el modo de la posterior no es una práctica usual en el enfoque bayesiano. Además de esto, la diferencia entre el estimador puntual bayesiano y la verosimilitud penalizada que en el primero la amplitud de la previa, definida por λ, se escoge sin mirar los datos. En cambio, en un contexto de optimización, se estima utilizando los datos (por ejemplo, mediante validación cruzada). Pero también hay bayesianos que escogen el valor de λ en base a los datos, y este enfoque se llama *Bayesianismo empírico*.

Otra forma de penalización es Ridge Regression, también llamada L2:

$$K_{Ridge} = \lambda \sum_{j=1}^{J} \beta_j^2,$$

equivalente a una previa

$$\beta_j \sim \text{normal}\left(0, \sqrt{\frac{1}{2\lambda}}\right).$$

Estas formas de penalización son muy utilizadas en *machine learning* para ajustar modelos de muchos parámetros previniendo el sobreajuste. También se suele usar la *elastic net*, una combinación lineal de Ridge y LASSO:

$$K_{EN} = \lambda \, [\alpha \sum_{j=1}^{J} \beta_j^2 + (1-\alpha) \sum_{j=1}^{J} |\beta_j|],$$

con $\alpha \in (0,1)$. Esto equivale a una previa definida como una mezcla entre una distribución de Laplace y una Normal, en donde α regula el peso de cada distribución.

Y otra forma de penalización es la Bridge:

$$K_{Bridge} = \lambda \sum_{j=1}^{J} |\beta_j|^p,$$

que se reduce a LASSO cuando $p = 1$ y a Ridge con $p = 2$.

La estrategia de agregar términos de penalización a la -log verosimilitud para evitar el sobreajuste también es conocida como *regularización*.