Trabajo Práctico 2: El Dibu de la vida

Introducción

El ozono (O) es un gas que se encuentra tanto en la atmósfera como en la superficie terrestre y cuya molécula está formada por tres átomos de oxígeno. Seguramente, lo han escuchado nombrar por su papel en la capa de ozono, una región de la estratófera que se encuentra entre unos 15 y 30 kilómetros sobre la superficie terrestre, donde se concentra este gas y filtra los rayos ultravioletas del sol.

La capa de ozono permite que exista la vida en la Tierra tal como la conocemos y que una escapada de verano a la isla no termine siendo solo un viaje de ida. En otras palabras, la capa de ozono es como un Dibu Martínez de la vida: evita que el sol nos clave un golazo de radiación y nos deje, literalmente, fritos.

Pero no todo lo que brilla es oro. A nivel del suelo, el ozono se convierte en un contaminante atmosférico dañino tanto para la salud humana como para el medio ambiente. Puede deteriorar la función pulmonar, desencadenar ataques de asma, causar irritación en los ojos, la nariz y la garganta, e incluso dañar la vegetación. Entonces, ¿todo mal con el ozono?

Quien se planteó esa pregunta fue Brian Tarkington, del *California Primate Research Center* de la Universidad de California en Davis. Preocupado por la presencia de ozono en el *smog* californiano y las posibles consecuencias en la salud humana, decidió llevar a cabo un estudio para evaluar el efecto del ozono en el crecimiento de ratas. Tomó un grupo de 46 ratas jóvenes y de la misma edad, las pesó, las dividió aleatoriamente en 2 grupos de igual tamaño y las dejó en 2 ambientes distintos, el primero rico en ozono y el segundo libre del mismo. Luego de 7 días las pesó nuevamente y registró la diferencia entre los pesos.

El archivo `tarkington.csv` contiene, para cada una de las ratas, el diferencial de peso en gramos y el grupo al que pertenece. Interesa conocer si la exposición prolongada a un ambiente contaminado con ozono se asocia a un deterioro en el desarrollo de las ratas, es decir, a un menor incremento de peso.

Modelización estadística

La Estadística ofrece una amplia variedad de técnicas para dar respuesta a la inquietud de Tarkington. En el contexto de esta materia, naturalmente, abordaremos el problema mediante el uso modelos bayesianos.

Y dado que el entusiasmo en esta materia es lo que sobra, aprovecharemos la oportunidad para utilizar no uno, sino dos modelos, sutilmente distintos. Esto nos permitirá no solo evaluar el efecto del ozono en el crecimiento de las ratas, sino también profundizar en esos pequeños detalles que hacen que la inferencia estadística sea tan interesante.

Modelo normal

El primer modelo que proponemos es el clásico caballito de batalla de innumerables análisis estadísticos: el viejo y confiable modelo normal. Aquí, el cambio en el peso de las ratas en cada grupo se modela mediante una distribución normal, permitiendo medias y varianzas específicas para cada grupo.

$$
\begin{aligned}
Y_{O,i} &\sim \text{Normal}(\mu_O, \sigma_O^2) \qquad i = 1, \ldots, N_O \\
Y_{C,j} &\sim \text{Normal}(\mu_C, \sigma_C^2) \qquad j = 1, \ldots, N_C \\
\mu_O, \mu_C &\sim \text{Normal}(0, 5^2) \\
\sigma_O, \sigma_C &\sim \text{Gamma}(\alpha = 6, \beta = 2)
\end{aligned}
$$

donde $Y_{O,i}$ representa el cambio en el peso de la i-ésima rata en el grupo expuesto al ozono e $Y_{C,j}$ representa el cambio en el peso de la j-ésima rata en el grupo control.

Modelo T de Student

El segundo modelo se parece demasiado al primero. La única —y, en principio, sutil— diferencia es que, en lugar de modelar la variable respuesta en cada grupo con una distribución normal, utilizamos una T de Student con 3 grados de libertad.

$$
\begin{aligned}
Y_{O,i} &\sim \text{Student-T}(\mu_O, \sigma_O^2, \nu = 3) \quad i = 1, \ldots, N_O \\
Y_{C,i} &\sim \text{Student-T}(\mu_C, \sigma_C^2, \nu = 3) \quad j = 1, \ldots, N_C \\
\mu_O, \mu_C &\sim \text{Normal}(0, 5^2) \\
\sigma_O, \sigma_C &\sim \text{Gamma}(\alpha = 6, \beta = 2)
\end{aligned}
$$

Un *deep dive* bien profundo en Metropolis-Hastings

Además de brindarle una respuesta al pobre de Brian Tarkington, que hace décadas que espera que alguien le ayude a resolver su problema, el objetivo principal de este trabajo práctico es profundizar en el uso de Metropolis-Hastings (MH) como algoritmo de inferencia bayesiana.

El algoritmo de Metropolis-Hastings permite generar muestras (pseudo-)aleatorias a partir de una distribución de probabilidad P que no necesariamente pertence a una familia de distribuciones conocida. El único requisito es que se pueda evaluar la función de densidad (o de masa de probabilidad) $p^*(\boldsymbol{\theta})$ en cualquier valor de $\boldsymbol{\theta}$, incluso cuando $p^*(\boldsymbol{\theta})$ sea impropia (es decir, incluso aunque sea desconocida la constante de normalización que hace que la integral en el soporte de la función sea igual a uno).

Algoritmo de Metropolis-Hastings

Se desea generar una muestra de valores $\{y^{(1)}, y^{(2)}, \cdots, y^{(n)}\}$ a partir de una distribución de probabilidad P con función de densidad p.

1. Seleccionar un punto inicial $y^{(1)}$.
2. Para cada $t \in \{1, \cdots, n\}$, repetir:

i. Realizar propuesta
Obtener un valor aleatorio y' de una variable Y' cuya distribución está dada por la distribución de propuesta Q, centrada en el valor de la última muestra obtenida:

$$Y' \sim Q(y^{(t)})$$

ii. Calcular la probabilidad de aceptación
Calcular el cociente entre la función de densidad en el punto propuesto y en el punto actual, ponderando por la densidad de la distribución de propuesta. La probabilidad de aceptación es igual a este cociente si es menor a 1, caso contrario es igual a 1.

$$\alpha = \min \left\{ 1, \frac{p(y')q(y^{(t)} \mid y')}{p(y)q(y' \mid y^{(t)})} \right\}$$

iii. Seleccionar el nuevo valor
Generar un valor aleatorio u de una distribución $\mathcal{U}(0,1)$ y determinar $y^{(t+1)}$ de la siguiente manera:

$$y^{(t+1)} = \begin{cases} y' & \text{si} \quad u \leq \alpha \\ y^{(t)} & \text{si} \quad u > \alpha \end{cases}$$

Metropolis-Hastings en espacios paramétricos acotados

La distribución normal suele ser una elección conveniente a la hora de proponer saltos, pero presenta desventajas cuando el espacio paramétrico es acotado: inevitablemente, algunas propuestas caerán fuera del dominio y rechazaremos más que de costumbre.

Para abordar este problema, existen dos estrategias principales. Una opción es utilizar distribuciones cuyo soporte coincida con el espacio objetivo. Sin embargo, encontrar parametrizaciones basadas en la media para todas ellas no es trivial.

La otra alternativa, más general, consiste en aplicar una transformación de variables para trabajar en un espacio no acotado, permitiendo así seguir utilizando una distribución de propuesta normal. Esta estrategia requiere conocer la densidad objetivo en el nuevo espacio, lo que implica calcular el determinante del jacobiano de la transformación. A su favor, muchas de las transformaciones más utilizadas permiten un cálculo sencillo, y este proceso puede automatizarse con herramientas de diferenciación automática existentes.

Transformación de variables aleatorias

Sean X e $Y = g(X)$ variables aleatorias continuas, donde g es una función uno a uno y X tiene función de densidad $f_X(x)$ conocida. Luego, la función de densidad de Y es:

$$f_Y(y) = f_X\left(g^{-1}(y)\right) \left| \frac{d}{dy} g^{-1}(y) \right|$$

donde $\left| \frac{d}{dy} g^{-1}(y) \right|$ es el determinante del jacobiano de la transformación. Por ejemplo, en el caso de la función $g(u) = \text{logit}(u)$, se tiene:

$$g : (0,1) \to \mathbb{R} \quad g(u) = \text{logit}(u) = \log\left(\frac{u}{1-u}\right)$$
$$g^{-1} : \mathbb{R} \to (0,1) \quad g(v) = \text{expit}(v) = \frac{1}{1+\exp(-v)}$$

y luego, utilizando X e Y definidas al principio:

$$f_Y(y) = f_X(\text{expit}(y)) \cdot \text{expit}(y) \cdot (1 - \text{expit}(y))$$

Metropolis-Hastings en escala logarítmica

Sin importar cuán moderna y potente sea nuestra computadora, siempre tendremos que lidiar con el talón de Aquiles del cálculo computacional: los problemas de subdesbordamiento y sobredesbordamiento (conocidos como *underflow* y *overflow* en inglés). Por ejemplo, si intentamos multiplicar 100 números del orden de `0.0001` en R, la computadora podría interpretar el resultado como 0, aunque matemáticamente esto no sea cierto.

El *underflow* es un problema frecuente en el cómputo estadístico. Un caso típico es el que se da al evaluar funciones de verosimilitud, donde se multiplican densidades correspondientes a cada observación. Dado que estas densidades suelen ser valores muy pequeños, es común enfrentarse

a un problema de subdesbordamiento. Este riesgo aumenta con la cantidad de observaciones y el número de dimensiones de la distribución objetivo.

El problema se agrava al utilizar algoritmos como Metropolis-Hastings, ya que la densidad objetivo se evalúa miles de veces en diferentes puntos del espacio paramétrico. Además, en inferencia bayesiana, esta función suele involucrar una función de verosimilitud, que puede implicar una multiplicación de gran cantidad de números pequeños, lo que incrementa aún más las chances de que el cómputo resulte en un *underflow*.

Si nuestro algoritmo de muestreo encuentra un problema de *underflow*, pueden ocurrir dos situaciones:

- Se genera un error que detiene la ejecución del programa.
- Se ignora la propuesta problemática y se realiza una nueva, pero nuestro programa ya no sigue la cadena de Markov deseada.

Entonces, ¿cómo solucionamos este problema? La respuesta es sencilla: realizar todos los cálculos en escala logarítmica. Las multiplicaciones se vuelven sumas y el cómputo se estabiliza.

Ahora sí, cerebro a la obra

La función `metropolis_hastings_log()` es una herramienta poderosa que, en teoría, permite obtener muestras de cualquier distribución *a posteriori* que se pueda implementar en R en escala logarítmica. Pero, ¿de qué sirve tener una herramienta tan potente si no está nada claro cómo dominarla?

A esta altura, intentar usar `metropolis_hastings_log()` para muestrear del *posterior* en los modelos propuestos inicialmente resultaría muy dificultoso. Sería como pasar de manejar un Golcito por las calles del pueblo a subirse a un Fórmula 1 en el medio de un Gran Premio.

Como bien dice el refrán, la práctica hace al maestro. Y quienes conocen de primera mano la verdad de estas palabras son, casualmente, los pilotos de Fórmula 1. No solo acumulan experiencia tras años de competencia en categorías menores, desde el karting hasta la Fórmula 2, sino que también, antes de cada Gran Premio, dan entre 400 y 500 vueltas al circuito… ¡pero en un simulador!

De este modo, en preparación para la aplicación final en este trabajo —nuestro propio Gran Premio— comenzaremos utilizando el algoritmo de Metropolis-Hastings en escenarios simples y controlados, aumentando gradualmente la complejidad. Esto nos permitirá practicar su uso con transformación de variables, en escala logarítmica y en *posteriors* de múltiples dimensiones. Y al final, como en toda buena carrera, llegará el Gran Premio.

1. En el simulador: ensayos con la distribución Gamma

El objetivo de esta sección es familiarizarnos con el uso de transformaciones para obtener muestras de distribuciones con soporte acotado. Inicialmente, utilizaremos la función `metropolis_hastings()` desarrollada en la práctica de la materia, que opera en escala original. Luego, utilizaremos la función `metropolis_hastings_log()` incluida más arriba, que trabaja en escala logarítmica.

Sea $X \sim \text{Gamma}(\alpha = 3, \beta = 8)$.

1. Obtenga y grafique la función de densidad de $Y = \log(X)$.
2. Utilice Metropolis-Hastings para obtener 10000 muestras de X. Para ello, obtenga muestras de Y utilizando una propuesta normal con $\sigma = 0.2$ y realice la transformación correspondiente para obtener las muestras de X. Para evaluar la bondad del método, calcule el tamaño de muestra efectivo, visualice la trayectoria de la cadena con un *traceplot* y compare la distribución empírica con la función de densidad teórica utilizando un histograma.

3. Implemente la función de densidad de Y en escala logarítmica y utilice `metropolis_hastings_log()` para obtener muestras de X. Calcule el tamaño efectivo de muestra efectivo y compárelo con el obtenido anteriormente. Finalmente, obtenga un histograma de las muestras y compárelo con la densidad empírica y con el histograma obtenido en el punto anterior.

2. *Free practice*: múltiples dimensiones

Al cómputo en escala logarítmica y el uso de transformaciones de variables, en esta etapa se le agrega que la distribución objetivo tiene múltiples dimensiones y debe ser obtenida mediante el uso de la regla de Bayes.

Dado el siguiente modelo:

$$\begin{aligned} Y_i &\sim \text{Normal}(\mu, \sigma^2) \\ \mu &\sim \text{Normal}(0, 1^2) \\ \sigma &\sim \text{Gamma}(\alpha = 2, \beta = 2) \end{aligned}$$

con $i = 1, \ldots, N$.

4. Implemente en R una función que permita calcular la densidad a *posteriori* en escala logarítmica, sin incluir la constante de normalización. Esta función tendrá 2 parámetros de entrada, uno para el vector de parámetros y otro para el vector de valores observados.
5. Utilice los valores de y_i en el archivo `precalentamiento-mh.txt` y la función `metropolis_hastings_log(` para obtener muestras del *posterior* de $\boldsymbol{\theta} = \{\mu, \sigma\}$. Corra dos cadenas de Markov. Calcule el tamaño efectivo de muestra y evalúe mezcla y convergencia de manera gráfica.

> Evaluación parcial de funciones
>
> En las líneas 41 y 42 del Listing 1 se puede ver que la función `logp` que se pasa a `metropolis_hastings_log()` debe funcionar correctamente al ser llamada con un único argumento, que es el vector de parámetros. Pero la función que implementamos en el punto 4 tiene 2 argumentos, uno para el vector de parámetros y otro para el vector de datos… ¿cómo hacemos?
> La función `partial()`, de la librería `purrr`, permite pre-llenar argumentos de una función. Toma como entrada una función, los argumentos con sus valores a pre-llenar, y devuelve una nueva función que hace lo mismo que la función original, pero con valores de argumentos ya especificados.
> En nuestro caso, podríamos hacer:
>
> ```
> library(purrr)
> logp <- partial(log_posterior, y = datos)
> ```
>
> Luego, la función `logp()` puede ser llamada solamente pasando el vector de parámetros. El valor de `y` será el que se pasó originalmente a la función `partial()`.

3. El Gran Prix: ¿qué le decimos a Brian?

A esta altura del partido (o de la carrera mejor dicho) ya estamos 'retequecontra' preparados para trabajar con los modelos presentados inicialmente y dar respuesta a la pregunta sobre el efecto del ozono en desarrollo de las ratas de Brian Tarkington. Llegó la hora de realizar el análisis estadístico.

6. Realice un análisis exploratorio de los datos y describa sus hallazgos.

7. Para una media y una varianza dada, ¿en qué se diferencia la distribución normal de la distribución T de Student con 3 grados de libertad?
8. Tanto para el modelo normal como para el basado en la T de Student:
   i. Escriba la función de densidad *a posteriori* en escala logarítmica, sin considerar la constante de normalización. De ser necesario, realice las transformaciones de variables que crea conveniente.
   ii. Implemente la función anterior en R.
   iii. Use `metropolis_hastings_log()` para obtener muestras del *posterior*. Utilice 4 cadenas de 5000 muestras cada una. Evalúe mezcla y convergencia mediante medidas numéricas y gráficas. Visualice las distribuciones *a posteriori* marginales.
   iv. Concluya en términos del problema.
9. ¿Llega a la misma conclusión con ambos modelos? ¿Por qué?

> **T de Student no centrada y escalada**
>
> La función `dt()` de R permite evaluar la función de densidad de una distribución T de Student estándar, con media 0 y desvío 1. Para evaluar la función de densidad con media y desvío arbitrario se puede utilizar la siguiente función de R:

```r
scaled_t_density <- function(x, df, mean = 0, sd = 1) {
  dt((x - mean) / sd, df) / sd
}
```

Listing 1 Implementación del algoritmo de Metropolis-Hastings en escala logarítmica.

```r
metropolis_hastings_log <- function(logp, x, n, sigma = NULL) {
  # Algortimo de Metropolis Hastings en escala logarítmica
  #
  # Parámetros
  #  ------------------------------------------------------------------------
  # | logp     | Función de densidad objetivo, normalizada o sin normalizar, |
  # |          | en escala logarítmica.                                      |
  # | x        | Posición inicial del algoritmo.                             |
  # | n        | Cantidad de muestras a obtener.                             |
  # | sigma    | Matriz de varianzas y covarianzas para la distribución de   |
  # |          | propuesta. Por defecto, es NULL y usa la matriz identidad.  |
  #  ------------------------------------------------------------------------
  #
  # Salida
  #  ------------------------------------------------------------------------
  # | muestras | Matriz con las muestras obtenidas.                          |
  #  ------------------------------------------------------------------------

  # Obtener dimensionalidad de la distribución objetivo a partir del punto inicial
  p <- length(x)

  # Inicializar matriz donde se guardan las muestras
  muestras <- matrix(NA, nrow = n, ncol = p)

  # Usar matriz diagonal unitaria para la distribución de propuesta cuando no se especifica
  if (is.null(sigma)) {
    sigma <- diag(p)
  }

  # Almacenar el punto inicial en la matriz de muestras
  muestras[1, ] <- x

  for (i in 1:(n - 1)) {
    # Obtener el valor de la muestra actual
    muestra_actual <- muestras[i, ]

    # Proponer un nuevo valor
    muestra_propuesta <- mvtnorm::rmvnorm(1, mean = muestra_actual, sigma = sigma)

    # Evaluar la log densidad en el valor actual y en el propuesto
    logp_propuesta <- logp(muestra_propuesta)
    logp_actual <- logp(muestra_actual)

    # Log densidad al pasar de muestra_propuesta a muestra_actual y viceversa
    logq_actual <- mvtnorm::dmvnorm(                                    ①
      muestra_actual, mean = muestra_propuesta, sigma = sigma, log = TRUE
    )
    logq_propuesta <- mvtnorm::dmvnorm(
      muestra_propuesta, mean = muestra_actual, sigma = sigma, log = TRUE
    )

    # Calcular log probabilidad de aceptación
    log_alpha <-                                                        ②

    # Simular aceptación o rechazo
```