EST-46115: Modelación Bayesiana

Profesor: Alfredo Garbuno Iñigo — Primavera, 2022 — Calibración basada en simulación .
Objetivo: En esta sección estudiaremos un mecanismo que puede ser utilizado para evaluar si un modelo está bien implementado. Esto aprovecha que la posterior es un procedimiento *calibrado* y recae en el hecho de que querríamos evaluar si nuestra implementación de un modelo es la correcta. Adicional, también querríamos identificar en qué parte del modelo se encuentra la deficiencia.
Lectura recomendada: Puedes consultar [2] el cual es una extensión de [1].

1. INTRODUCCIÓN

Decimos que un intervalo de probabilidad está `bien calibrado` si tiene la misma cobertura nominal con la que está construido. Digamos, si el intervalo es un intervalo de probabilidad del 80 % , entonces 4 de 5 veces éste contendrá al verdadero valor del parámetro.

Por definición, la distribución posterior está calibrada –si los datos son generados por el modelo–. Utilizando `calibración basada en simulación~` explotaremos esta propiedad.

La idea es: generar parámetros de la previa para generar datos condicionados en los parámetros simulados con el objetivo de estudiar la calibración de la posterior bajo escenarios independientes.

2. CALIBRACIÓN DE LA POSTERIOR

Supongamos que tenemos un modelo bayesiano donde tenemos una distribución previa $\pi(\theta)$ para los parámetros del mecanismo generador de datos (verosimilitud) $\pi(y|\theta)$.

Ahora, consideremos que generamos una simulación

$$\theta_{\mathsf{sim}} \sim \pi(\theta)\,, \tag{1}$$

con la cual generamos

$$y_{\mathsf{sim}} \sim \pi(y|\theta_{\mathsf{sim}})\,. \tag{2}$$

Por construcción lo que hemos hecho es

$$(y_{\mathsf{sim}}, \theta_{\mathsf{sim}}) \sim \pi(y, \theta)\,. \tag{3}$$

Ahora, la regla de Bayes nos dice que la distribución posterior es proporcional a la conjunta

$$\pi(\theta|y) \propto \pi(y, \theta)\,. \tag{4}$$

Por lo que también podríamos pensar que

$$\theta_{\mathsf{sim}} \sim \pi(\theta|y_{\mathsf{sim}})\,. \tag{5}$$

Si utilizamos nuestro muestreador favorito para generar

$$\theta_1, \ldots, \theta_M \sim \pi(\theta|y_{\mathsf{sim}})\,. \tag{6}$$

Entonces podríamos comparar las muestras simuladas con el parámetro que generó los datos. Esto es, por que θ_{sim} es una realización aleatoria de la posterior, y por lo tanto los estadísticos de orden de θ_{sim} deberían de ser uniformes con respecto a los de $\theta_1, \ldots, \theta_M$.

O dicho de otra manera,

$$\pi(\theta) = \int \pi(\theta|y_{\mathsf{sim}})\pi(y_{\mathsf{sim}}|\theta_{\mathsf{sim}})\pi(\theta_{\mathsf{sim}})\,\mathrm{d}y_{\mathsf{sim}}\,\mathrm{d}\theta_{\mathsf{sim}}\,. \tag{7}$$

Lo cual nos da un mecanismo para evaluar qué tan bien estamos utilizando nuestro algoritmo para generar muestras de la posterior [2].

3. CALIBRACIÓN BASADA EN CALIBRACIÓN (CBS)

Utilizaremos la propiedad discutida arriba para generar

$$y_{\mathsf{sim}}^{(n)}, \theta_{\mathsf{sim}}^{(n)} \sim \pi(y, \theta), \qquad n = 1, \ldots, N\,. \tag{8}$$

Para cada uno de los datos simulados $y_{\mathsf{sim}}^{(n)}$ utilizamos nuestro algoritmo para generar

$$\theta_1^{(n)}, \ldots, \theta_M^{(n)} \sim \pi(\theta|y_{\mathsf{sim}}^{(n)})\,. \tag{9}$$

Si consideramos los estadísticos de orden–el número de simulaciones de la posterior menores al parámetro simulado–en cada componente de nuestro vector de parámetros, tendremos que

$$r_n = \mathsf{orden}\left(\theta_{\mathsf{sim}}^{(n)}, \left\{\theta_1^{(n)}, \ldots, \theta_M^{(n)}\right\}\right) \tag{10}$$

$$= \sum_{m=1}^{M} 1[\theta_m^{(n)} < \theta_{\mathsf{sim}}^{(n)}]\,, \tag{11}$$

será un entero distribuido $\mathsf{Uniforme}\{0, \ldots, M\}$. Esto es, el estadístico de orden r_n tiene probabilidad $1/(M+1)$ de tomar valores entre $0, \ldots, M$.

> El artículo de Talts et al. [2] utiliza visualizaciones con histogramas. El artículo de Cook et al. [1] formuló la prueba de hipótesis que pone a prueba la uniformidad para poder identificar desviaciones de implementaciones de modelos.

3.1. Ejemplo: Modelo conjugado

Consideremos un modelo de una observación Gaussiana $y \sim \mathsf{N}(\mu, \sigma^2)$, en donde utilizamos el siguiente conocimiento previo para los parámetros

$$\mu \sim \mathsf{N}(0, 1)\,, \tag{12}$$

y asumimos una σ conocida.

Este modelo es un modelo conjugado `Normal-Normal` el cual tiene un distribución posterior

$$\mu|y \sim \mathsf{N}\left(\frac{y}{\sigma^2+1}, 1 + \frac{1}{\sigma^2}\right)\,. \tag{13}$$

Consideremos $\sigma^2 = 2$. Si realizamos una simulación de la previa tenemos el siguiente parámetro:

```
set.seed(108791)
sim ← list(mu = rnorm(1))
sim ▷ as.data.frame()
```

```
    mu
1 0.61
```

Con los cuales podemos simular un conjunto de datos:

```
data ← list(y = rnorm(1, sim$mu, sd = sqrt(2)))
data
```

```
$y
[1] 1.4
```

Y con estos datos simulamos de la posterior $M = 4$ iteraciones:

```
params ← tibble(mu = rnorm(4, data$y/3, sd = sqrt(2/3)))
params ▷ as.data.frame()
```

```
     mu
1 0.947
2 0.037
3 1.268
4 0.954
```

Hacemos las comparaciones contra $\mu_{\text{sim}} = 0.61$:

```
params ▷
  mutate(indicadora = ifelse(mu < sim$mu, 1, 0)) ▷
  as.data.frame()
```

```
     mu indicadora
1 0.947          0
2 0.037          1
3 1.268          0
4 0.954          0
```

Si calculamos el estadístico de rango, obtenemos una $r_{1,\mu} = 1$. El cual debería de estar uniformemente distribuido entre los enteros del 0 al 4. ¿lo ponemos a prueba?

```
experimento ← function(id){
  sim ← list(mu = rnorm(1))
  data ← list(y = rnorm(1, sim$mu, sd = sqrt(2)))
  mu ← rnorm(4, data$y/3, sd = sqrt(2/3))
  sum(mu < sim$mu)
}

resultados ← tibble(id = 1:100) ▷
   mutate(rank = map_dbl(id, experimento))
```

La idea es replicar el procedimiento de generación de parámetros y muestras sintéticas con la intención de observar un comportamiento uniforme en los histogramas (Fig. 1).

Para cada réplica $n = 1, \ldots, N$, podemos generar un número fijo de simulaciones de la posterior (M). Talts et al. [2] recomiendan simular tantas iteraciones de la posterior como se

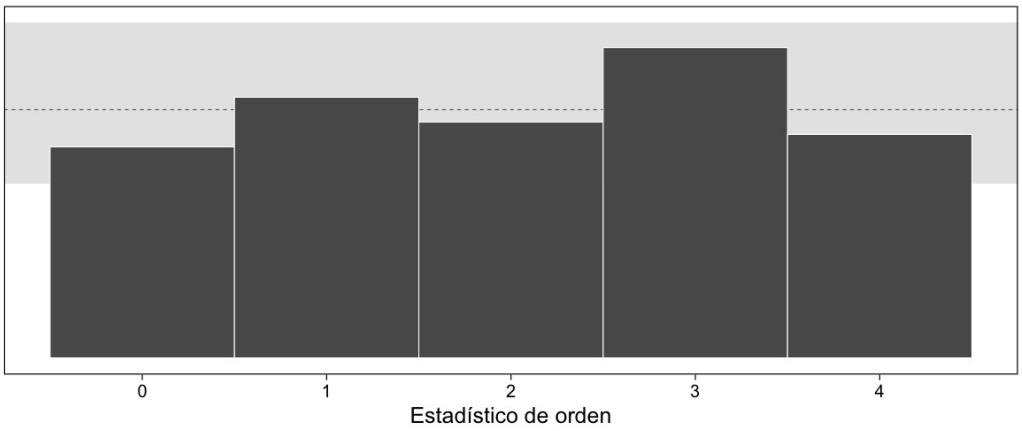

FIGURA 1. *Histogramas de estadisticas de orden con 4 simulaciones de la posterior . Construimos una línea de referencia (y bandas de confianza) bajo los supuestos de la distribución uniforme de los estadísticos de orden.*

requiera y *resumir* (agrupar) los resultados en 20 cubetas. De tal forma que podamos criticar un histograma de 20 barras. En la Fig. 2 observamos un histograma con 20 cubetas y la línea de referencia de un modelo uniforme con $M = 20$. Adicional, se muestran los intervalos de un experimento binomial con N réplicas con probabilidad $1/M$ de caer en cada cubeta.

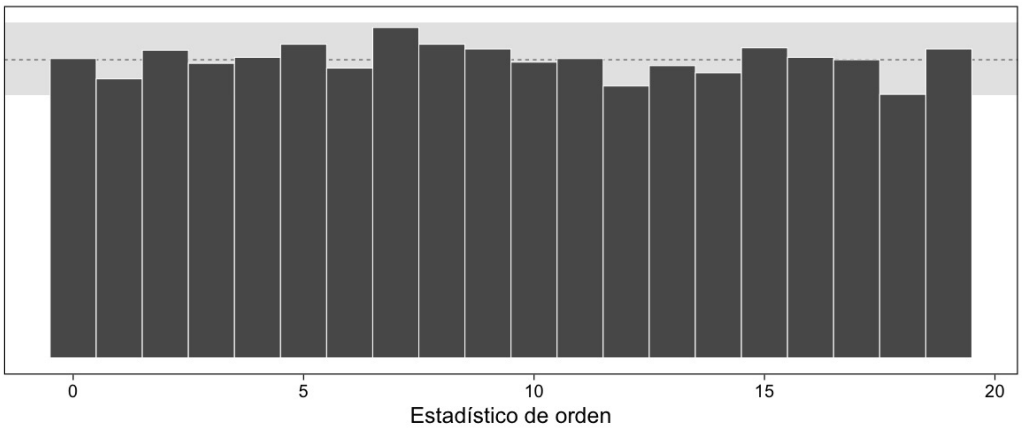

FIGURA 2. *Histogramas de estadisticas de orden con 19 simulaciones de la posterior. Construimos una línea de referencia (y bandas de confianza) bajo los supuestos de la distribución uniforme de los estadísticos de orden.*

El procedimiento descrito arriba nos permite evaluar de manera *visual* los histogramas. Alternativas a esta estrategia es poder evaluar la función de acumulación empírica (`ECDF`) contra el modelo uniforme. Esto también puede compararse de manera visual como se muestra en la Fig. 3, en donde estamos comparando contra la función de acumulación (`CDF`) de experimentos uniformes (panel izquierdo). Por otro lado, la comparación gráfica entre la `ECDF` y `CDF` se vuelve compleja en realizarse si el número de cubetas (M) es muy elevado. Por eso tendemos a comparar la diferencia, asumiendo una aproximación Gaussiana (panel derecho).

3.2. Cuando el modelo está mal especificado

Consideremos los errores típicos de una implementación de un modelo. Por ejemplo, tenemos un modelo que tiene una dispersión mas pequeña que la que debería. En estas situaciones tenemos un comportamiento de los histogramas en forma de $\cup$ como se muestra en la Fig. 4.

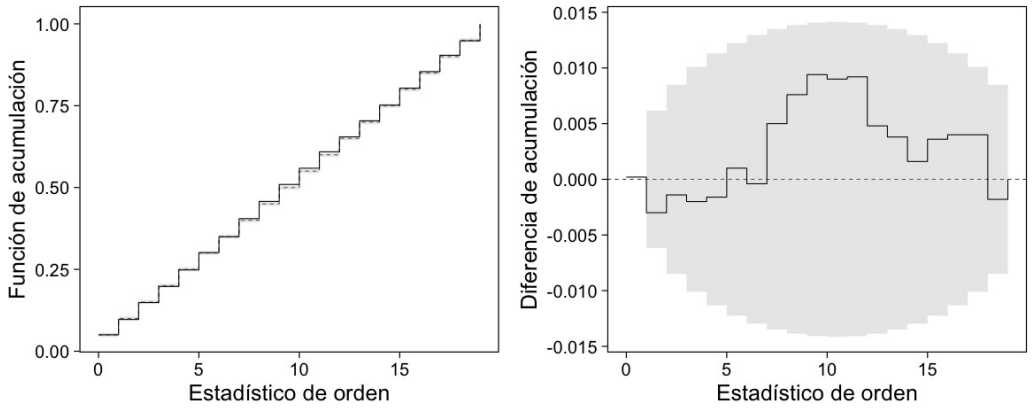

FIGURA 3. *Gráficos alternativos para evaluar la prueba uniforme.*

Esto corresponde a un modelo con una `incertidumbre baja` contra la que debería tener.

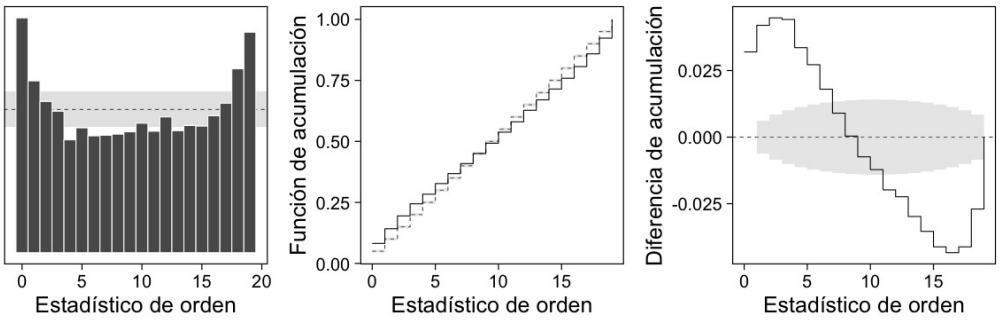

FIGURA 4. *Gráficos de comparación uniforme cuando la implementación está sub-dispersa.*

Cuando la implementación es de un modelo sobre-disperso tenemos un comportamiento en forma de ∩ como se muestra en la Fig. 5. Esto corresponde a un modelo con una `incertidumbre mayor` a la que debería corresponder.

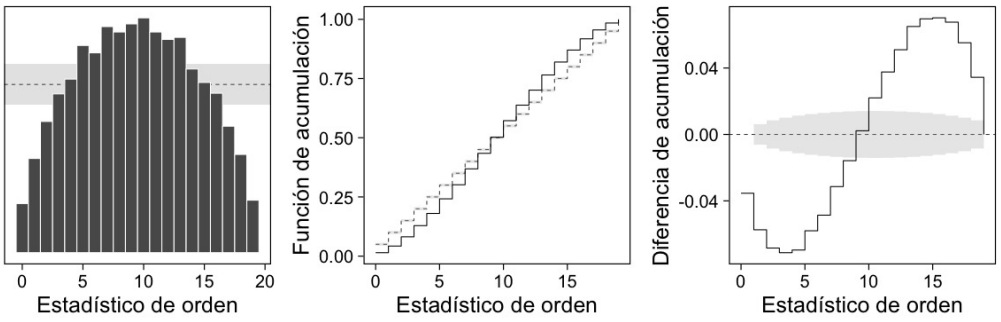

FIGURA 5. *Gráficos de comparación uniforme cuando la implementación está sobre-dispersa.*

Cuando la implementación es de un modelo con sesgo a la derecha tenemos un comportamiento como se muestra en la Fig. 6. Esto corresponde a un modelo que está `sobre-estimando` los resultados que debería tener.

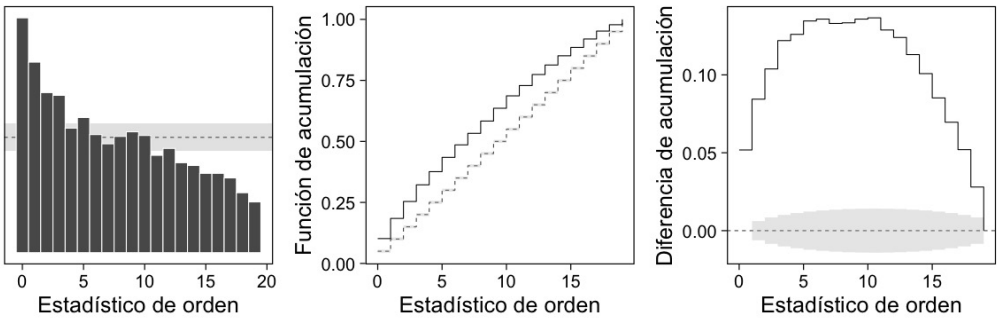

FIGURA 6. *Gráficos de comparación uniforme cuando la implementación tiene un sesgo a la derecha.*

El caso contrario (sesgo a la izquierda) representa un modelo que está `sub-estimando` las probabilidades.

3.3. Pruebas de uniformidad

Una manera de poder efectuar una prueba es considerar una χ^2 y verificar que los conteos en las cubetas corresponden, en promedio, a lo que esperaríamos con ordenes uniformes.

El estadístico de prueba sería

$$\hat{\chi}^2 = \sum_{m=1}^{M} \frac{(b_m - e_m)^2}{e_m}\,, \tag{14}$$

donde b_m denota el número de réplicas en la cubeta m ésima y e_m denota el número de réplicas que esperaríamos caigan en dicha cubeta.

La prueba radica en que los términos de la suma son potencias cuadradas de una normal estándar y por lo tanto

$$\hat{\chi}^2 \sim \chi^2_{M-1}\,, \tag{15}$$

de la cual podemos evaluar una prueba de hipótesis.

Nota la prueba de hipótesis definida anteriormente no tiene una potencia alta.

4. CBS EN `STAN`

La idea, como hemos mencionado antes, es poner a prueba si nuestra implementación de un modelo es la adecuada. Estas pruebas no están diseñadas para verificar que nuestro modelo es el adecuado.

Usaremos `Stan` para:

1. Simular datos.
2. Ajustar la distribución posterior.
3. Calcular los estadísticos de orden.

Esto implicará que tenemos que correr nuestro simulador varias veces para poder producir un histograma de estadísticos de orden que esperamos tenga una distribución de muestreo uniforme dentro de los rangos.

4.1. Implementación en `Stan`

Podemos utilizar un bloque `transformed data` para simular parámetros y datos para el modelo. Regresando a nuestro modelo Normal-Normal, tenemos un bloque que genera parámetros simulados.

```
transformed data {
  real mu_sim = normal_rng(0, 1);
  real y_sim  = normal_rng(mu_sim, sqrt(2));
}
```

Adicional, podemos utilizar un bloque `generated quantities` para calcular las indicadoras y los estadísticos de orden

```
generated quantities {
  int<lower=0, upper=1> lt_sim = { mu < mu_sim };
}
```

4.2. Consideración para métodos de MCMC

Utilizar técnicas de MCMC nos permite simular de la distribución objetivo. Esperaríamos que las muestras sean lo más cercanas a ser independientes. El diagnóstico N_{eff} nos puede dar una indicación de con cuántas muestras nos podemos quedar para realizar los histogramas.

4.3. Ejemplo

Regresaremos a nuestro ejemplo de las escuelas. Sabemos que el modelo puede tener problemas si no está bien parametrizado. Realizaremos un estudio numérico con $N = 500$ réplicas del proceso. En cada una simulamos de tal forma que `adelgazamos` la cadena de Markov cada 10 iteraciones. El número total de simulaciones se fija para recuperar $M = 100$ ordenes posibles. Los gráficos muestran histogramas con 20 cubetas.

> Nota que [2] proponen un algoritmo para poder aplicar `SBC` a muestras de un cadena de Markov. Dicha propuesta esta basada en estar revisando, por réplica, el número efectivo de simulaciones para poder generar una muestra que pueda ser adelgazada después. Sin embargo, el problema de las escuelas está tan bien identificado y sabemos que nuestra implementación del modelo será deficiente, que no será necesario pedir cadenas tan estables.

```
transformed data {
  real mu_sim = normal_rng(0, 5);
  real tau_sim = fabs(normal_rng(0, 5));
  int<lower=0> J = 8;
  array[J] real theta_sim = normal_rng(rep_vector(mu_sim, J), tau_sim);
  array[J] real<lower=0> sigma = fabs(normal_rng(rep_vector(0, J), 5));
  array[J] real y = normal_rng(theta_sim, sigma);
}
parameters {
  real mu;
  real<lower=0> tau;
  array[J] real theta;
}
model {
  mu ~ normal(0, 5);
  tau ~ normal(0, 5);
  theta ~ normal(mu, tau);
  y ~ normal(theta, sigma);
}
generated quantities {
  int<lower=0, upper=1> mu_lt_sim = mu < mu_sim;
  int<lower=0, upper=1> tau_lt_sim = tau < tau_sim;
```

```
  int<lower=0, upper=1> theta1_lt_sim = theta[1] < theta_sim[1];
}
```

Nota que el bloque de `transformed data` escribe el proceso generador de los datos. Primero, simulamos los parámetros poblacionales (μ, τ); después, los datos (y_j, σ_j).

Los resultados de esta implementación nos están advirtiendo que el modelo posterior tiene una distribución con sobre-dispersión para el parámetro θ_1. Además para $\log \tau$ parece también haber evidencia de cierto sesgo del modelo. Ver Fig. 7 y Fig. 8.

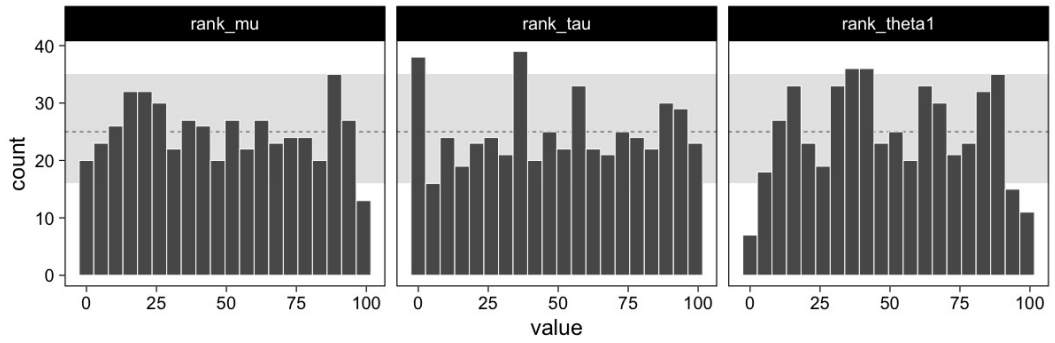

FIGURA 7. *Contraste de histogramas contra la distribución uniforme.*

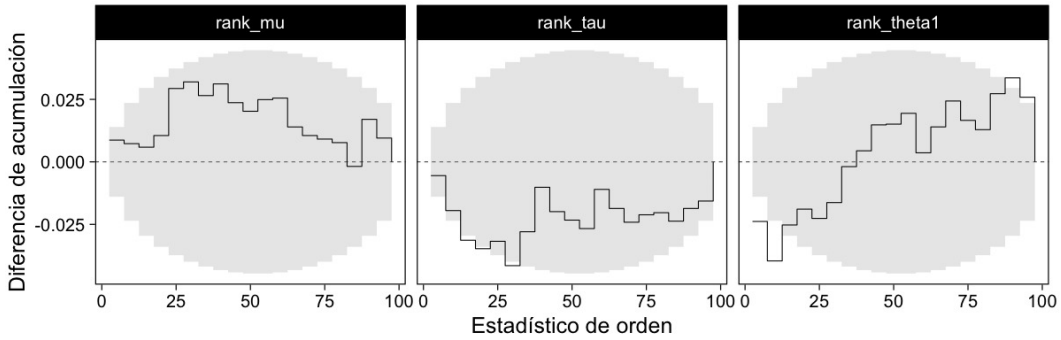

FIGURA 8. *Diferencia entre la `ECDF` y la `CDF` bajo un modelo uniforme de los estadísticos de orden.*

5. CASO PRÁCTICO

Consideraremos un modelo de mezclas

$$\pi(y|\theta, w) = \sum_{k=1}^{K} w_k \, \pi_k(y|\theta_k) \,, \tag{16}$$

donde $\sum_k w_k = 1$, θ es un vector de parámetros por bloques, y las densidades π_k pueden pertenecer a la misma familia.

En este caso consideraremos dos componentes $K = 2$, $\theta = (\mu_1, \mu_2)^\top$ y π_k la función de masa de probabilidad de una Poisson con media μ_k.

El modelo escrito en `Stan` queda como sigue. Nota que dejaremos en un ciclo externo la simulación de datos sintéticos, por lo tanto, no utilizaremos el bloque de `generated quantities`. Todo el procesamiento lo haremos fuera de `Stan`.

```
data {
  int<lower=0> N;
  int y[N];
}

parameters {
  real mu1;
  real mu2;
  real<lower=0, upper=1> omega;
}

model {
  target += log_mix(omega, poisson_log_lpmf(y | mu1), poisson_log_lpmf(y | mu2
      ));
  target += normal_lpdf(mu1 | 3, 1);
  target += normal_lpdf(mu2 | 3, 1);
}
```

La función para generar las simulaciones es el siguiente:

```
generate_poisson_mix ← function(N){
  ## Generamos parametros simulados
  mu1 ← rnorm(1, 3, 1)
  mu2 ← rnorm(1, 3, 1)
  omega ← runif(1)
  ## Generamos datos sinteticos
  y ← numeric(N)
  for(n in 1:N){
    if(runif(1) < omega){
      y[n] ← rpois(1, exp(mu1))
    } else {
      y[n] ← rpois(1, exp(mu2))
    }
  }
  ## Regresamos en lista
  sim ← within(list(), {
                mu ← c(mu1, mu2)
                omega ← omega
  })
  obs ← list(N = N, y = y)
  list(sim = sim, obs = obs)
}
```

El modelo tiene un poco de problemas en correr. Por ejemplo, algunas simulaciones tienen un número efectivo de simulaciones mucho menores de las que corremos (alrededor del 10 %). Así que hace sentido adelgazar la cadena para mitigar los efectos de correlación en los gráficos de diagnóstico.

Los resultados nos muestran histogramas que corresponden a un modelo sobre-disperso. Lo cual es consecuencia de un modelo posterior con mucho mayor incertidumbre de la que esperaríamos. Ver Fig. 9 y Fig. 10.

Por supuesto, esto lo pudimos haber diagnosticado observando una réplica de haber simulado de la posterior bajo un conjunto de datos hipotético. Sin embargo, bajo este enfoque (estudiar una sola réplica) siempre puede quedar la duda si lo que observamos es un artificio de una simulación (por ejemplo de fijar una semilla) o es un comportamiento generalizable.

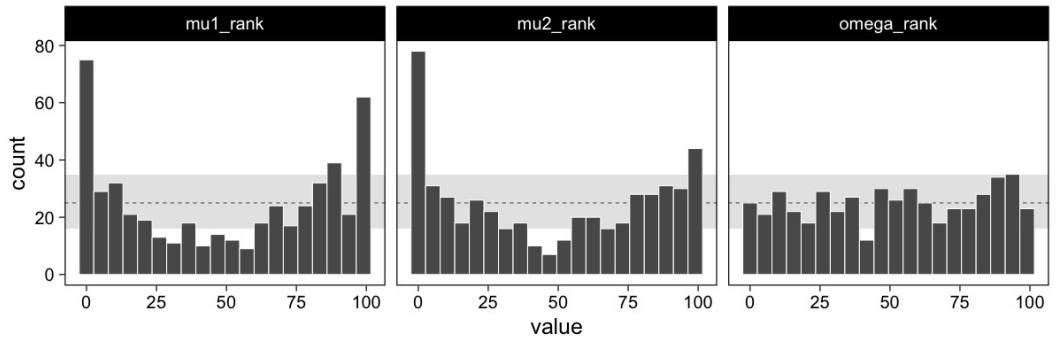

FIGURA 9. *Histogramas de los estadísticos de orden para el modelo de mezclas Poisson.*

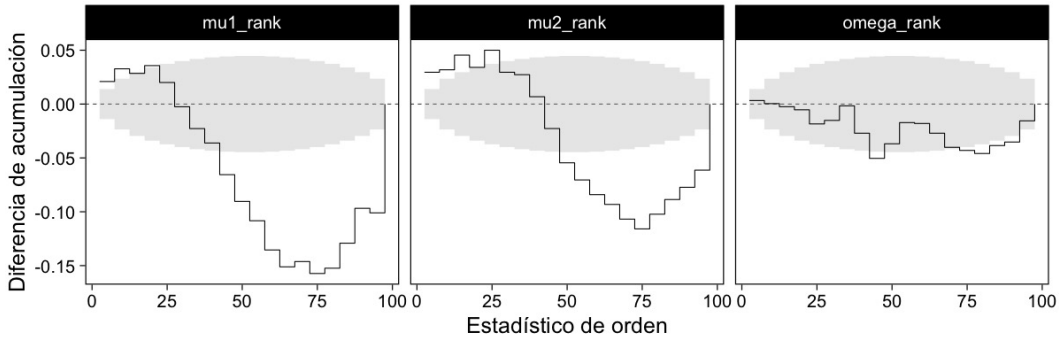

FIGURA 10. *Diferencia de los estadísticos de acumulación entre el empírico y el teórico.*

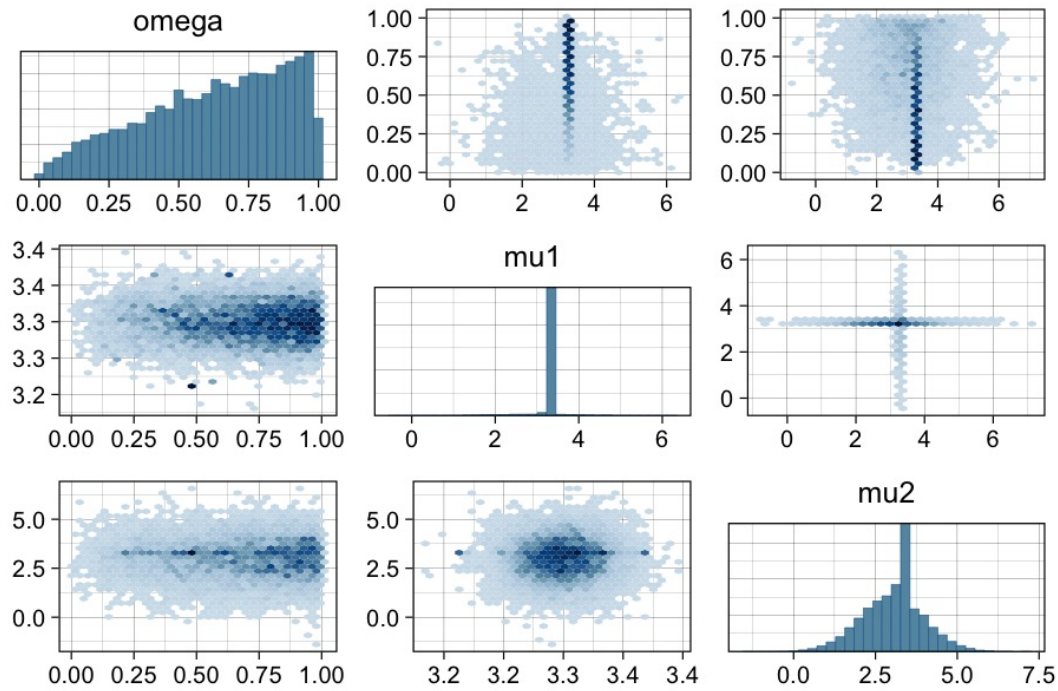

FIGURA 11. *Realización de un ajuste posterior con un modelo mal especificado para las muestras del modelo de mezcla Poisson.*

5.1. Re-implementando

El problema anterior se debe a que el modelo sobre-ajusta a un componente. Nota que el parámetro de peso no puede ser ajustado por el modelo. Revisando la documentación de `Stan` sobre modelos de mezclas, notamos que teníamos mal implementado el modelo para mas de una observación.

Lo que hicimos anteriormente asigna el mismo componente de la mezcla para todos los elementos de la muestra. Esto no tiene sentido, pues pensaríamos que nuestro modelo tiene datos que provienen de los dos componentes. El problema de inferencia es sobre con qué proporción vienen de cada uno y los parámetros que identifican a cada uno de los componentes.

```
data {
  int<lower=0> N;
  int y[N];
}

parameters {
  real mu1;
  real mu2;
  real<lower=0, upper=1> omega;
}

model {
  for(n in 1:N) {
    target += log_mix(omega,
                      poisson_log_lpmf(y[n] | mu1),
                      poisson_log_lpmf(y[n] | mu2));
  }
  target += normal_lpdf(mu1 | 3, 1);
  target += normal_lpdf(mu2 | 3, 1);
}
```

```
simulaciones ← tibble(id = 1:500) ▷
    mutate(results = map(id, replicate_experiment, modelo))
```

Los resultados con nuestra simulación (500 réplicas y utilizar muestras para identificar órdenes de hasta 100) nos brindan los siguientes gráficos.

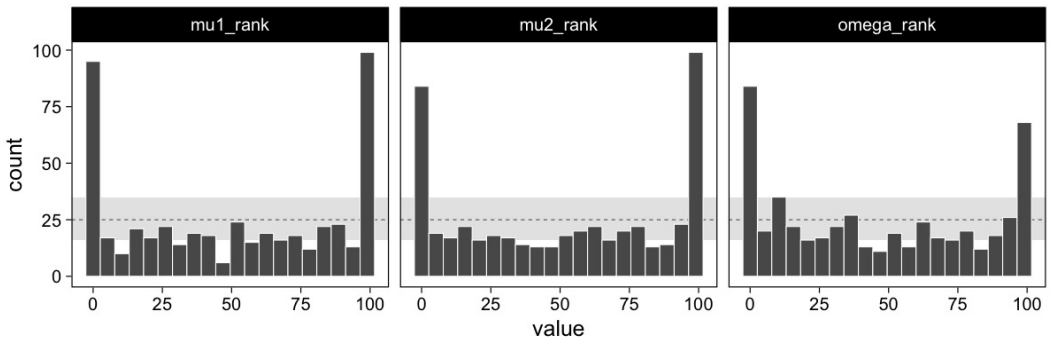

FIGURA 12. *Histogramas para los diagnósticos de orden.*

Después de observar Fig. 12 y Fig. 13, notamos que aún tenemos un modelo con muy poca incertidumbre. Al parecer hay todavía algo que no está bien en la implementación.

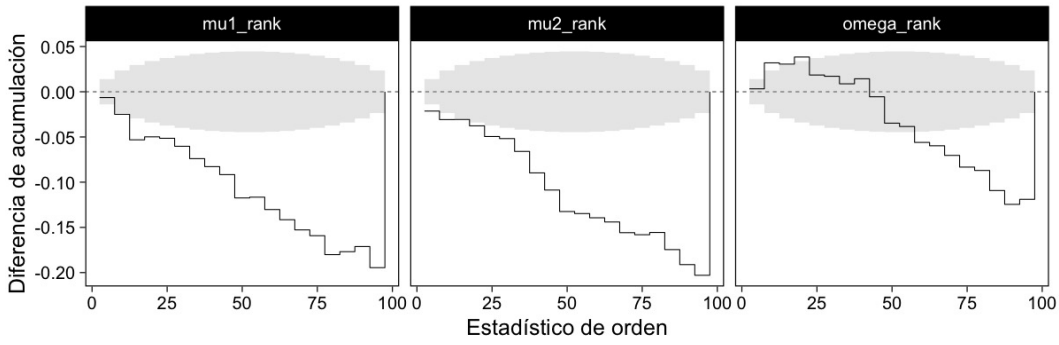

FIGURA 13. *Diferencia entre el estimador empírico de acumulación y el teórico.*

De nuevo exploramos una nueva simulación. En este caso utilizamos 4 cadenas para tratar de visualizar algún problema. La Fig. 14 nos muestra un comportamiento multi-modal.

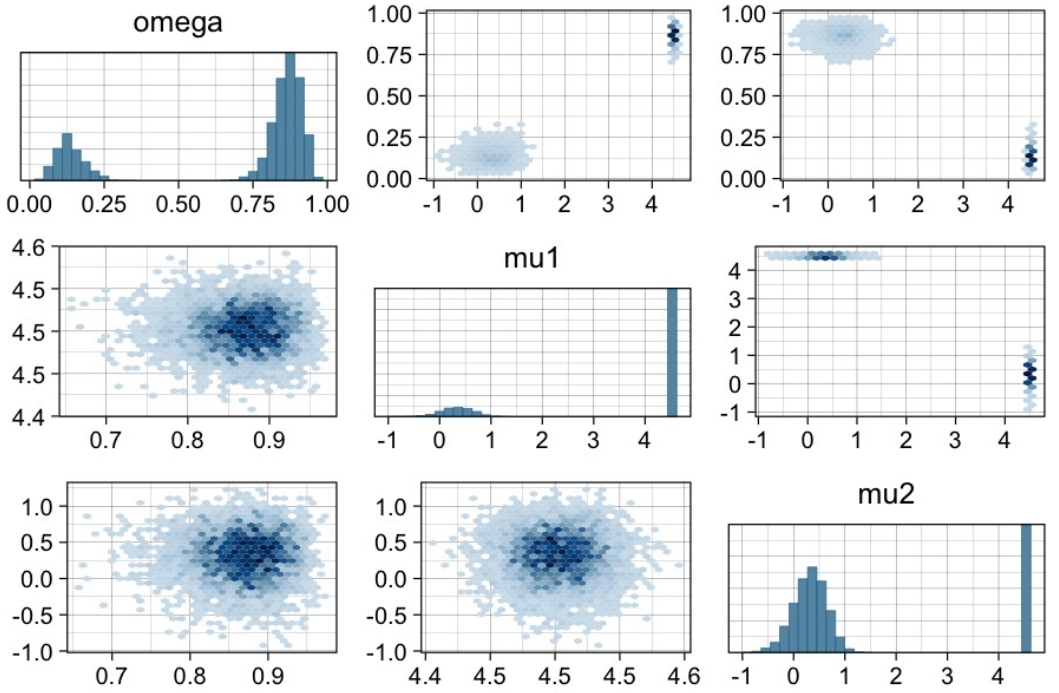

FIGURA 14. *Realización del modelo vemos un modelo con dos modas que no es fácilmente identificable.*

5.2. Arreglando problemas de identificabilidad

Para modelos de mezclas es usual no poder identificar cada componente de manera única. De hecho, no hay nada en el modelo anterior que limite de alguna manera el rol de cada uno de los componentes de la mezcla. Lo resolvemos con lo siguiente.

```
data {
  int<lower=0> N;
  int y[N];
}

parameters {
```

```
  ordered[2] mu;
  real<lower=0, upper=1> omega;
}

model {
  for(n in 1:N) {
    target += log_mix(omega,
                      poisson_log_lpmf(y[n] | mu[1]),
                      poisson_log_lpmf(y[n] | mu[2]));
  }
  target += normal_lpdf(mu | 3, 1);
}
```

Por supuesto, tenemos que cambiar nuestra implementación del mecanismo que genera datos del modelo generativo (la distribución conjunta de datos y parámetros).

```
generate_poisson_mix_ordered ← function(N){
  ## Generamos parametros simulados
  mu ← sort(rnorm(2, 3, 1))
  omega ← runif(1)
  ## Generamos datos sinteticos
  y ← numeric(N)
  for(n in 1:N){
    if(runif(1) < omega){
      y[n] ← rpois(1, exp(mu[1]))
    } else {
      y[n] ← rpois(1, exp(mu[2]))
    }
  }
  ## Regresamos en lista
  sim ← within(list(), {
                mu ← mu
                omega ← omega
  })
  obs ← list(N = N, y = y)
  list(sim = sim, obs = obs)
}
```

```
replicate_experiment_ordered ← function(id, modelo){
  data ← generate_poisson_mix_ordered(50)
  posterior ← modelo$sample(data$obs, chains = 1, refresh = 1000,
                            iter_sampling = 990, thin = 10)

  posterior$draws(format = "df") ▷
    as_tibble() ▷
    mutate(
      mu1_bool = `mu[1]` < data$sim$mu[1],
      mu2_bool = `mu[2]` < data$sim$mu[2],
      omega_bool = omega < data$sim$omega) ▷
    summarise(
      mu1_rank = sum(mu1_bool),
      mu2_rank = sum(mu2_bool),
      omega_rank = sum(omega_bool),
      )
}
```

```
simulaciones ← tibble(id = 1:500) ▷
    mutate(results = map(id, replicate_experiment_ordered, modelo))
```

Nuestros resultados se ilustran en las últimas figuras de esta sección.

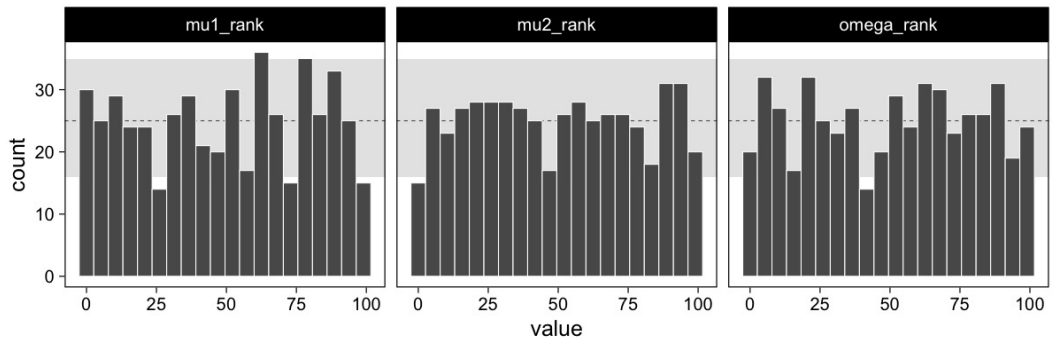

FIGURA 15. *Histogramas de los estadísticos de orden.*

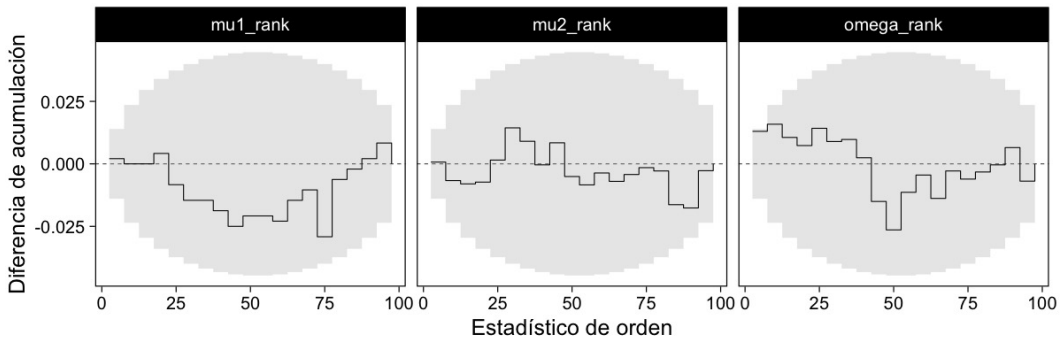

FIGURA 16. *Diferencia entre el estimador de acumulación empírico y el teórico.*

Lo que vemos es que nuestro modelo está bien implementado. Ya no sufre de los problemas que veíamos anteriormente y esto nos pone en una situación donde podremos utilizar nuestro modelo para ajustar datos. Por supuesto, esto no garantiza que el modelo será infalible cuando se enfrente a nuestras observaciones. Pero al menos podemos estar tranquilos que la implementación es la correcta.

6. CONCLUSIONES

En esta sección mostramos un mecanismo para identificar distribuciones bien calibradas. El mecanismo aprovecha que por definición hacer inferencia Bayesiana es un procedimiento bien calibrado. Es decir, siempre y cuando los datos sean generados por el modelo probabilístico nuestra cobertura de intervalos será igual a la nominal.

Existen alternativas para evaluar métodos de muestreo. Sin embargo, estos mecanismos son utilizados cuando hacemos alguna inferencia aproximada. Es decir, cuando estamos dispuestos a hacer una aproximación de la verosimilitud (usualmente el componente mas costoso) o de la posterior misma (que veremos rumbo al final del curso).

En el mismo espíritu de diagnósticos de MCMC, `SBC` es un mecanismo para evaluar y criticar la implementación de un modelo. No nos dice qué modelo tiene sentido bajo un conjunto de datos. Esto es, justo lo que estudiaremos en la sección siguiente.

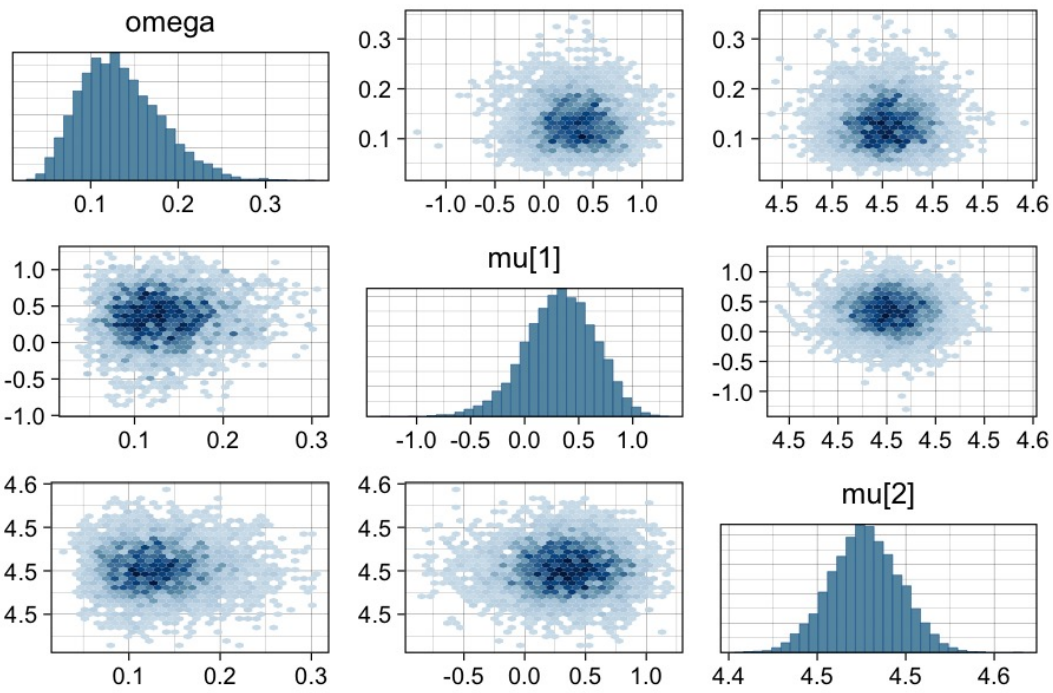

FIGURA 17. *Visualización de un ajuste posterior para el modelo de mezclas Poisson.*

REFERENCIAS

[1] S. R. Cook, A. Gelman, and D. B. Rubin. Validation of Software for Bayesian Models Using Posterior Quantiles. *Journal of Computational and Graphical Statistics*, 15(3):675–692, sep 2006. ISSN 1061-8600, 1537-2715. . 1, 2

[2] S. Talts, M. Betancourt, D. Simpson, A. Vehtari, and A. Gelman. Validating Bayesian Inference Algorithms with Simulation-Based Calibration. *arXiv:1804.06788 [stat]*, oct 2020. 1, 2, 3, 7