Meta-Analysis for the Binomial Model with Normal Random Effects

Heirarchical and Empirical Bayesian Approaches

Sophie Huebler

Overview

We consider a binomial model with normal random effects as an illustrative example for Bayesian inference. Two different approaches, hierarchical and empirical, are used to draw inference on the same set of data. The hierarchical approach utilizes a coded-from-scratch Gibbs-Metropolis-Hastings algorithm, and the empirical approach uses optimization of marginal likelihood found by simulation or numerical integration. Both approaches are computationally intensive, and thus efficiency improvements are possible through data table utilization, parallel computing, and C++ implementation through RCPP. The goal of this project is to write a package that efficiently performs these two analyses.

Section 1: Introduction

Sub-section 1.1: The model

The treatment effect of interest is the odds ratio of a binary event occurring in a member of a treatment group versus a control group. This can be modeled in the standard way by recording the number of events and sample size of both groups. However, if multiple studies are performed investigating this same treatment effect, we could not reasonably expect the exact same estimate from all of the studies. Instead, We assume that due to differing study protocol and random chance, each study is estimating a log odds ratio that is drawn from a normal distribution with center μ and variance τ^2. We will call this the random effect population prior $\pi(\theta_i|\mu, \tau)$. A meta-analysis will allow us to incorporate information from all of the studies, not just one, to better estimate the treatment effect.

Let k be the number of studies we want to incorporate into the meta-analysis, and let j be the treatment assignment. Then Y_{ij} is the number of successes recorded in trial i for the jth

group. We will draw inference on the log odds ratios θ_is which are drawn from the $N(\mu, \tau^2)$ distribution.

$$
\begin{aligned}
&i \in i, ..., k \quad \textit{and} \quad j \in \{C, T\} \\
&Y_{ij}|\pi_{ij}, n_{ij} \sim Bin(n_{ij}, \pi_{ij}) \\
&logit\pi_i^C|\gamma_i, \theta_i = \gamma_i - \theta_i/2 \\
&logit\pi_i^T|\gamma_i, \theta_i = \gamma_i + \theta_i/2 \\
&logit(\frac{\pi_{iT}}{\pi_{iC}}) = \theta_i|\mu, \tau^2 \sim N(\mu, \tau^2) \\
&\mu \sim N(a_1, b_1) \\
&1/\tau^2 \sim gamma(a_2, b_2)
\end{aligned}
$$

Sub-section 1.2: Hierarchical Approach

The Gibbs sampling algorithm with a metropolis-hastings procedure is a well known tool used in hierarchical Bayesian inference. The basis of the algorithm is breaking the joint distribution into full univariate conditional posterior densities and then generating Markov chains by iteratively updating estimates for each variable from its full conditional. When the full conditional does not have a closed form solution, the metropolis-hastings procedure is used to component wise update variables. In our case, the full joint distribution has the form:

$$
\begin{aligned}
f(Y, \theta, \gamma, \mu, \tau) &= f(Y|\theta, \gamma) f(\gamma) f(\theta|\mu, \tau^2) f(\mu) f(\tau^2) \\
f(r^T_{1,...,k}, r^C_{1,...,k}, \gamma_{1,...,k}, \theta_{1,...k}, \mu, \tau) &\propto \prod_{i=1}^{k} f(r_i^T|\gamma_i, \theta_i) f(r_i^C|\gamma_i, \theta_i) f(\gamma_i) f(\theta_i|\mu, \tau) f(\mu) f(\tau^2)
\end{aligned}
$$

which gives the following set of full conditionals:

$$
\begin{aligned}
f(\gamma_i|\theta, \gamma_{[i]}, \gamma_i, \mu, \tau^2, Y) &\propto \frac{e^{\gamma_i(r_i^T - r_i^C)}}{(1 + e^{\gamma_i - \theta_i/2})^{r_i^C} (1 + e^{\gamma_i + \theta_i/2})^{r_i^T}} e^{-\frac{1}{2*100}(\gamma_i)^2} \\
f(\theta_i|\theta_{[i]}, \gamma, \mu, \tau^2, Y) &\propto \frac{e^{\frac{\theta_i}{2}(r_i^T - r_i^C)}}{(1 + e^{\gamma_i - \theta_i/2})^{r_i^C} (1 + e^{\gamma_i + \theta_i/2})^{r_i^T}} \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{1}{2\tau^2}(\theta_i - \mu)^2} \\
f(\mu|\theta, \gamma, \tau^2, Y) &\sim N(\frac{\sigma \sum \theta}{k\sigma + 100}, \frac{1}{k\sigma + 100}) \\
f(\tau^2|\theta_i, \gamma_i, \mu, Y_i) &\sim InverseGamma(\frac{k}{2} + a_2, \frac{\sum(\theta_i - \mu)^2}{2} + b_2)
\end{aligned}
$$

where

$Y := (r^C_{1,\ldots,k}, r^T_{1,\ldots,k}, n^T_{1,\ldots,k}, n^C_{1,\ldots,k})$

$\theta_{[i]}$:= all θ_j such that $i \neq j$

$\gamma_{[i]}$:= all γ_j such that $i \neq j$

Note that only μ and τ^2 have closed forms. Metropolis-hastings must be used for the other variables. In this approach, hyperparameters are used as priors for the random effects. These hyperparamateters are given the following defaults:

$\mu \sim N(0, 100)$; $\frac{1}{\tau^2} \sim \text{Gamma}(0.001, 0.001)$

but functions are written flexibly so different hyperparameters can be chosen.

Sub-section 1.3: Empirical Approach

Under the empirical approach, rather than update the estimates for μ and τ^2 in the Markov chain, all other parameters can be integrated out of the full likelihood to find the marginal likelihood for only μ and τ^2. Optimization of that marginal likelihood can then be used to empirically estimate the μ and τ^2 values, leaving only the a single level to the model to perform bayesian inference on the θ_i values.

Optimizing the following gives $\hat{\mu}$ and $\hat{\tau^2}$.

$$\int_\theta \int_\gamma \binom{n_i^T}{r_i^T}\binom{n_i^C}{r_i^C} \frac{e^{(\gamma_i-\theta_i/2)r_i^C} e^{(\gamma_i+\theta_i/2)r_i^T}}{(1+e^{\gamma_i-\theta_i/2})^{r_i^C}(1+e^{\gamma_i+\theta_i/2})^{r_i^T}} \frac{1}{\sqrt{2\pi 100}} e^{-\frac{1}{2*100}(\gamma_i-0)^2}$$
$$\times \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{1}{2*\tau^2}(\theta_i-\mu)^2} \frac{0.001^{0.001}}{\Gamma(0.001)} (1/\tau^2)^{0.001-1} e^{\frac{-0.001}{\tau^2}} d\gamma d\theta$$

From there, the full conditionals for only the γs and θs reduce to

$$f(\gamma_i|Y,\theta) \propto \frac{e^{\gamma_i(r_i^T-r_i^C)}}{(1+e^{\gamma_i-\theta_i/2})^{r_i^C}(1+e^{\gamma_i+\theta_i/2})^{r_i^T}} e^{-\frac{1}{2*100}(\gamma_i)^2}$$

$$f(\theta_i|Y,\gamma) \propto \frac{e^{\theta_i/2(r_i^T-r_i^C)}}{(1+e^{\gamma_i-\theta_i/2})^{r_i^C}(1+e^{\gamma_i+\theta_i/2})^{r_i^T}} e^{-\frac{1}{2*\hat{\tau}^2}(\theta_i-\hat{\mu})^2}$$

The gibbs-metropolis-hastings algorithm can then be run on these simplified full conditionals.

Section 2: Solution Plan

Sub-section 2.1: Simulation and Timing

Currently, both analyses are written inefficiently. Improvements will be tested using the bench and microbenchmark packages in R. Of particular interest is the visualizations within microbenchmark to determine if the overhead cost of parallel programming is worth it for the Gibbs-Metropolis-Hastings algorithm. Since the algorithm is iterative, parallelization for each step of the sequence is not possible. However, within each step of the Metropolis-Hastings procedure there is a loop equal to the number of studies in the meta-analysis. This could be parallelized, but may not be worth it.

Sub-section 2.2: RCPP

The marginal likelihood function is currently written in R (see Functions/mlik_original.R), and it approximates numerical integration using simulation. However, as an alternative, the full likelihood will be written in C++. This offers an advantage because C++ has a GNU Scientific Library that has capabilities for numerical integration. The current approximation does not perform well if the choice for the starting estimates of the variables of interest are not close to the true value. This means that there is opprotunity for improving accuracy as well as efficiency.

Additionally, the full conditional posterior functions can be written in C++. This benefit may prove negligible because the functions are already pretty fast in their current form in R.

Sub-section 2.3: Parallelization

Within the R marginal likelihood function that uses simulation, the simulations can be run in parallel because the iterations are independent. This should be achievable with little overhead cost because only base R functions are being used.

Also, as mentioned in section 2.1, parallelization within the Metropolis-Hastings step will be implemented if simulations determine that the overhead cost is not prohibitive.

Sub-section 2.4: Data Table

The Gibbs-Metropolis-Hastings algorithm must store a large array of data. Currently, the data is being stored as a list at the end of each modularized function and then pieces are called in the next function (mh -> gibbs -> mh_gibbs). Even with pre-allocation of space, the conversion and appending of lists is inefficient. Changing the output to data tables in each step will allow smoother and faster transfering.

Section 3: Preliminary results

The repository is set up with a "Functions" folder that contains modularized functions and annotations including which goals are to be addressed within each step. The original codeas for the Gibbs-Metropolis-Hastings algorithm and marginal likelihood estimation is also stored within the functions folder so that comparisons in efficiency can be made.

Sub-section 3.1: The Data

The example data to be used comes from 7 sites investigating a treatment effect. Log odds ratios and their variances are presented for each study.

Table 1: Data

Trial	R_it	N_it	R_ic	N_ic	log(OR)	Var(log(OR))
Balcon	14	56	15	58	-0.05	0.19
Clausen	18	66	19	64	-0.12	0.15
Multicentre	15	100	12	95	0.20	0.17
Barber	10	52	12	47	-0.36	0.24
Norris	21	226	24	228	-0.14	0.10
Kahler	3	38	6	31	-1.03	0.57
Ledwich	2	20	3	20	-0.46	0.95

Subsection 3.2: Improvements to the Gibbs-Metropolis-Hastings Algorithm

Improvements to the Gibbs-Metropolis-Hastings algorithm have been made by eliminating unnecessary loops. The original construction was set up with 1 for-loop over the length of the chain which called 4 lapply statements over the number of studies incorporated. Then, still within the outer for-loop, vectorized computation was performed to identify which variables required updating during that iteration, and a second inner for-loop over the number of studies was called to make those updates. In total, there were 5 loops over the number of studies (4 lapply calls and 1 for-loop) nested within each step of the outer loop. This was rectified in the updated algorithm by reducing to only 1 for-loop over the number of studies within each step of the chain.

Below is a bench mark time comparison of the original and updated algorithms.

```
             min   median  itr/sec mem_alloc   gc/sec n_itr n_gc total_time
original 1.173255 1.173255 1.000000   1.66645 1.022796     1    6   2.948077
updated  1.000000 1.000000 1.173255   1.00000 1.000000     1    5   2.512734
```

As we can see, minor time improvements have been made even before the implementation of data table or parallelization.