Fitting multiparameter models

Math 315, Adam Loy

For a small number of parameters, the grid approximation can be used to approximate the posterior distribution in a multiparameter model. Below, we work through the steps to fit the informative model for heights discussed in class:

$$\begin{aligned} Y_1, \ldots, Y_n &\overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma) \\ \pi(\mu, \sigma) &= \pi(\mu)\pi(\sigma) \\ \mu &\sim \mathcal{N}(178,\ 20) \\ \sigma^2 &\sim \text{Unif}(0, 50) \end{aligned}$$

Work through the steps of this handout carefully, so that you understand both the R code involved and what you are accomplishing with each code chunk.

0. Load any necessary packaages We'll use a couple functions from `dplyr` and I use `ggplot2` to make plots, so let's load the `tidyverse`:

```
library(tidyverse)
```

1. Load the data Now, load the data:

```
adults <- read.csv("http://aloy.rbind.io/data/adults.csv")
```

2. Explore the implications of the prior specification Next, explore the prior predictive distribution for this setting by

i. simulating $S = 1000$ $\mu^{(i)}$ draws from the prior for μ,
ii. simulating $S = 1000$ $\sigma^{2^{(i)}}$ draws from the prior for σ^2, and
iii. simulating $S = 1000$ $Y^{(i)}$ draws from $\mathcal{N}(\mu^{(i)}, \sigma^{(i)})$.

Does the prior specification appear reasonable considering the world's tallest person ever was 272 cm tall.

3. Choose a grid that covers the posterior density To use grid approximation, we need to choose a reasonably large grid to ensure that non-zero density is placed over the region where the posterior will reside. Here, the Cartesian product of $\mu \in [118, 238]$ and $\sigma \in [0.1, 50]$ seems reasonable. In R, the `expand.grid()` command returns a data frame containing all combinations of the two grids:

```
param_grid <- expand.grid(
  mu = seq(from = 118, to = 238, length.out = 1000),
  sigma = seq(from = 0.1, to = 50, length.out = 1000)
)
```

Check the number of rows and columns, does this match your intuition?

4. Evaluate the log prior density over the grid Once you have the grid in hand, you can evaluate the joint prior density over the grid, taking advantage of R's vectorized operations. As we discussed in class, it is wise to evaluate the prior, likelihood, and posterior on the log scale to avoid rounding error erroneously forcing the posterior probabilities to zero.

```
param_grid <-  param_grid %>%
  mutate(
    log_prior = dnorm(param_grid$mu, 178, 20, log = TRUE) +
```

```
      dunif(param_grid$sigma, 0, 50, log = TRUE)
  )
```

5. Evaluate the log likelihood density over the grid To evaluate the log likelihood using R's built-in `dnorm()` command, we either need to use a for loop, or to use the `Vectorize()` command to extend `dnorm()` for use with vectors of μs and σs. The vectorized code is *much* faster, so that's what is discussed below.

To begin, you need to write a log likelihood function that will be vectorized:

```
# log likelihood function
log_lik_norm <- function(x, mu, sigma) {
  sum(dnorm(x, mean = mu, sd = sigma, log = TRUE))
}
```

Next, we use the `Vectorize()` function and specify which arguments we need to allow to be vectorized. Here, we want `\mu` and `sigma` to be vectorized.

```
log_lik_norm <- Vectorize(log_lik_norm, vectorize.args = c("mu", "sigma"))
```

Once we have the vectorized function in hand, we can use `mutate()` to add the necessary columns for our posterior computation to `param_grid`.

```
posterior <- param_grid %>%
  mutate(
    log_lik = log_lik_norm(adults$height, mu = mu, sigma = sigma), # Step 1
    log_post = log_prior + log_lik,                                # Step 2
    unstd_post = exp(log_post - max(log_post)),                    # Step 3
    post = unstd_post / sum(unstd_post)                            # Step 4
  )
```

The above code completes the following steps:

1. Calculates the log likelihood for each μ, σ combination, storing the results in the `log_lik` column.
2. Calculates the log posterior, storing it in the `log_post` column.
3. Calculates the unstandardized posterior, storing it in the `unstd_post` column. (The formula here might seem a little funny, but subtracting the mean increases the computational stability.)
4. Standardizes the posterior (by dividing by the sum) and stores the result in the `post` column.

6. Use Monte Carlo sampling to draw $(\mu^{(i)}, \sigma^{(i)})$ pairs from the posterior We can use Monte Carlo sampling to draw samples from our grid-approximate posterior distribution. Since we are drawing pairs of parameters rather than single values, I recommend using the `sample_n()` command in the **dplyr** R package so that you can draw samples from the rows of a data frame.

```
posterior_draws <- sample_n(posterior, size = 1e4, replace = TRUE, weight = post)
```

7. Analyze the joint or marginal posteriors Now that you have the joint posterior distribution in hand you can conduct inference as desired. Complete the following steps to analyze the posterior in this setting:

a. Plot the joint posterior density as: (i) a scatterplot where you make the points transparent (say `alpha = 0.2`), and (ii) a contour plot. I would recommend using the **ggplot2** R package for both of these tasks, using either the `geom_point()` or `geom_density2d()`. You can fill in the code below to achieve this:

```
# Fill in code for a scatterplot
ggplot(data = ___, aes(x = ___, y = ___)) +
  geom_point(alpha = 0.2) +
```

```
  labs(x = expression(mu),
       y = expression(sigma))

# Fill in code for a contour plot
ggplot(data = ___, aes(x = ___, y = ___)) +
  geom_density2d() +
  labs(x = expression(mu),
       y = expression(sigma))
```

b. Plot the marginal posterior density of μ, the mean height.

c. Calculate and interpret a 93% credible interval for the mean height in this population.