Bayesian CLT

Math 315, Fall 2019

In class, we discussed the univariate Bayesian CLT, but ran out of time while discussing the bivariate CLT. We were able to discuss the statement of the theorem, but did not talk about implementation. Since this approximation works well in problems with a small number of parameters and large sample size, it is worth knowing how to implement, but most statistical models require a larger number of parameters, so I don't want to dwell on it. With this in mind, please review this handout and ask any questions you have about the Bayesian CLT.

Review: One-parameter Bayesian CLT

In class, we discussed the one-parameter Bayesian CLT in detail.

Theorem. Under general conditions (which are discussed in *BSM* section 7.2.2), for large enough n the posterior is approximately

$$\theta|Y_1,\ldots,Y_n \dot{\sim} \mathcal{N}\left(\widehat{\theta}_{MAP}, \widehat{\Sigma}_{MAP}\right)$$

where $\widehat{\Sigma}_{MAP} = I_n(\theta)^{-1}$. Here, $I_n(\theta)^{-1}$ denotes the observed Fisher information in your random sample, which is given by

$$I_n(\theta) = n\left(-\frac{d^2}{d\theta^2}\log\left[p(\theta|Y_1,\ldots,Y_n)\right]\right).$$

To calculate $\widehat{\Sigma}_{MAP}$ you need to plug in $\widehat{\theta}_{MAP}$ for θ.

In class, we calculated $\widehat{\theta}_{MAP}$ and $\widehat{\Sigma}_{MAP}$ by hand, but you can also do this using the `laplace()` function in the **LearnBayes*** R package.

Example. In the beef patty example from class we considered the model

$$\begin{aligned} Y|\theta &\sim Binomial(n=16,\theta) \\ \theta &\sim Beta(1,1) \end{aligned}$$

Recall, that $Y = 13$ was observed. Let's use the `laplace()` function to approximate the posterior distribution. The `laplace()` function requires three key arguments:

1. A function providing the log posterior distribution,
2. an initial guess at the mode of the posterior, and
3. the observed data.

First, program a `logpost()` function that requires arguments for the parameter (`theta`) and the observed `data`:

```
logpost <- function(theta, data) {
  dbinom(x = data, size = 16, prob = theta, log = TRUE)
}
```

Next, pass your `logpost()` function and your initial guess at the MAP estimate into the `laplace()` function:

```
library(LearnBayes)
laplace(logpost, mode = 0.5, data = 13)
```

```
## $mode
## [1] 0.8125
##
## $var
##              [,1]
## [1,] 0.009521039
##
## $int
## [1] -2.801491
##
## $converge
## [1] TRUE
```

Tip: try a few initial guesses for the mode to see whether your estimates are sensitive to the initial guess.

Notice that `laplace()` returns a list. The key elements of this list are:

- `mode`: the MAP estimate
- `var`: the estimated variance

Putting the pieces together, the Bayesian CLT states that

$$\theta|Y_1, \ldots, Y_n \dot{\sim} \mathcal{N}\left(0.8125, 0.0095\right)$$

Multiparameter Bayesian CLT

In practice, the Bayesian CLT is used for multiparameter models. The multiparameter version is given below.

Theorem. Under general conditions (which are discussed in *BSM* section 7.2.2), for large enough n the posterior is approximately

$$\boldsymbol{\theta}|Y_1, \ldots, Y_n \dot{\sim} \mathcal{N}\left(\widehat{\boldsymbol{\theta}}_{MAP}, \widehat{\boldsymbol{\Sigma}}_{MAP}\right)$$

where

- $\boldsymbol{\theta}$ denotes a vector of parameters,
- $\widehat{\boldsymbol{\theta}}_{MAP}$ denotes the vector of MAP estimates, and
- $\widehat{\boldsymbol{\Sigma}}_{MAP}$ is the variance-covariance matrix for the multivariate normal distribution.

In addition, $\widehat{\boldsymbol{\Sigma}}_{MAP} = (-\mathbf{H})^{-\mathbf{1}}$, where the Hessian $\mathbf{H}$ is the matrix defined by

$$\frac{\partial^2}{\partial\theta_j \partial\theta_k} \log[p(\boldsymbol{\theta}|Y_1, \ldots, Y_n)]\Big|_{\boldsymbol{\theta}=\widehat{\boldsymbol{\theta}}_{MAP}}$$

Example. Let's clarify the statement of the Bayesian CLT by revisiting the rat radiation example. Recall that the model of interest is given by

$$\begin{aligned} Y_i|\alpha, \beta &\overset{\text{iid}}{\sim} \text{Gamma}(\alpha, \beta) \\ \alpha &\sim \text{Gamma}(5, 1) \\ \beta &\sim \text{Gamma}(2, 1) \end{aligned}$$

Yielding a the posterior

$$p(\alpha, \beta|\mathbf{Y}) \propto \beta^{n\alpha+4} e^{-\beta(\sum y_i+1)} \cdot \frac{\alpha}{[\Gamma(\alpha)]^n} e^{-\alpha} \left(\prod y_i\right)^{\alpha+1}$$

which is not a distribution we know. Let's use the Bayesian CLT to approximate this bivariate posterior density!

Note that here $\boldsymbol{\theta} = (\alpha, \beta)'$.

If we were planning on solving this by hand, then we would follow the steps outlined below:

Step 1. Find $\widehat{\boldsymbol{\theta}}_{MAP}$ by maximizing the log posterior density.

Step 2. Find $\widehat{\boldsymbol{\Sigma}}_{MAP}$ by calculating the Hessian, evaluating it at $(\widehat{\alpha}_{MAP}, \widehat{\beta}_{MAP})$, and inverting it. The Hessian is given below:

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2}{\partial \alpha^2} \log[p(\alpha, \beta|\mathbf{Y})] & \frac{\partial^2}{\partial \alpha \partial \beta} p(\alpha, \beta|\mathbf{Y}) \\ \frac{\partial^2}{\partial \alpha \partial \beta} p(\alpha, \beta|\mathbf{Y}) & \frac{\partial^2}{\partial \beta^2} p(\alpha, \beta|\mathbf{Y}) \end{bmatrix}$$

While this is a nice application of calculus and linear algebra, we'll use R to do the heavy lifting.

Step 0. Load your data set.

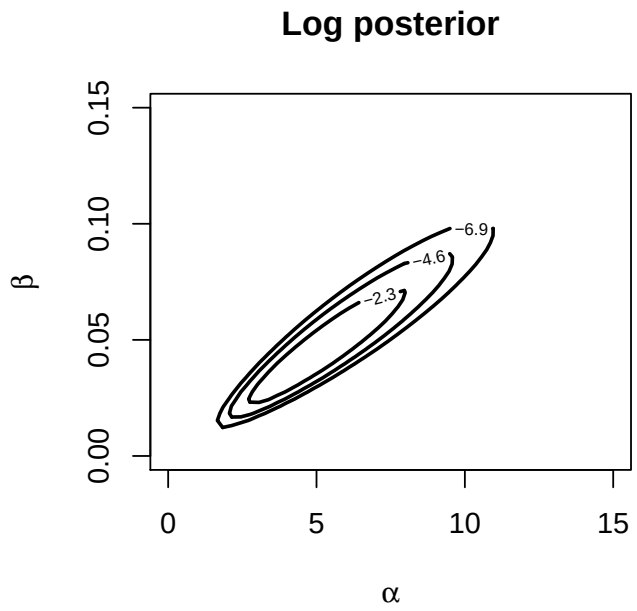
```
times <- c(152, 152, 115, 109, 137, 88, 94, 77, 160, 165,
           125, 40, 128, 123, 136, 101, 62, 153, 83, 69)
```

Step 1. Write a function for the log posterior. In general, I recommend first deriving the log posterior using pencil and paper before you start coding. This will reveal some simplifications. In this case, however, we can utilize the `dgamma()` function to implement the densities. Below is the log posterior function I wrote in R.

```
logpost <- function(theta, data) {
  # set up parameters
  alpha <- theta[1]
  beta  <- theta[2]

  loglike <- 0
  for(i in 1:length(data)) {
    loglike <- loglike + dgamma(data[i], alpha, beta, log = TRUE)
  }

  post <- loglike + dgamma(alpha, 5, 1, log = TRUE) + dgamma(alpha, 2, 1, log = TRUE)

  return(post)
}
```

You can visualize your log posterior using the `mycountour()` function in the **LearnBayes** package.

```
mycontour(
  logpost,                                          # log posterior function
  limits = c(xlo = 0, xhi = 15, ylo = 0, yhi = .15), # constraints on plotting window
  data = times                                      # data set
)
title(main = "Log posterior", xlab = expression(alpha), ylab = expression(beta))
```



Step 2. Next, pass your log posterior density function an initial guess for the MAP estimates and your data set into `laplace()`.

```
fitted_model <- laplace(
  logpost,          # log posterior density function
  mode = c(8, 1),   # initial guess for MAP estimates
  data = times      # observed data
)
```

To see the mean and variance-covariance matrix of this bivariate normal approximation, extract the `mode` and `var` from the `fitted_model` list:

```
fitted_model$mode
```

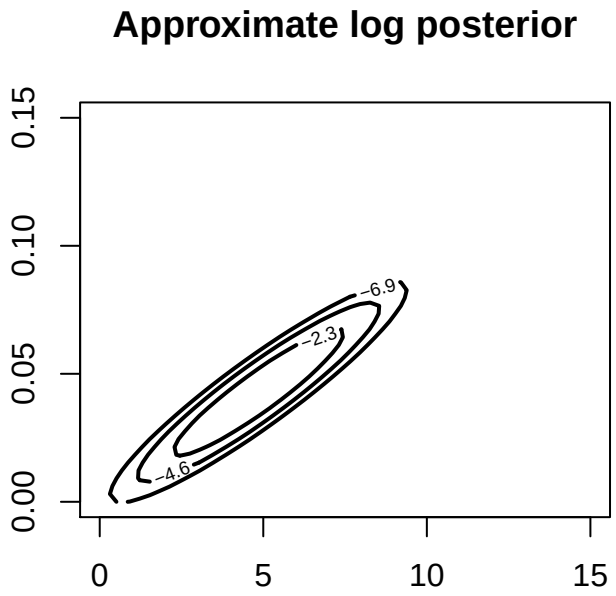
```
## [1] 4.85553691 0.04280149
```

```
fitted_model$var
```

```
##             [,1]         [,2]
## [1,] 1.49675294 0.0131818444
## [2,] 0.01318184 0.0001349361
```

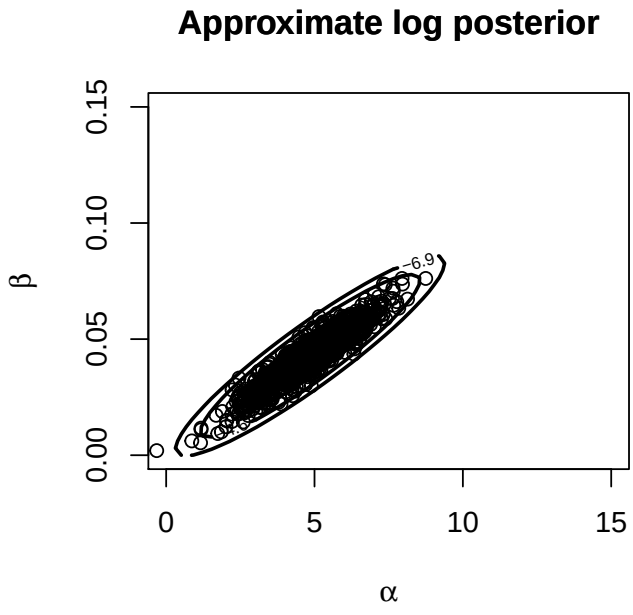
Again, we can plot the posterior density using tools in the **LearnBayes**. The `lbinorm()` function provides a log bivariate normal density function which requires the a list of parameters including the mean vector and variance-covariance matrix, in that order.

```
npar = list(m = fitted_model$mode, v = fitted_model$var) # list with mean, covariance
mycontour(
  lbinorm,                                              # approx log posterior density
  limits = c(xlo = 0, xhi = 15, ylo = 0, yhi = .15), # plotting limits
  npar,                                                 # parameters for lbinorm
  main = "Approximate log posterior"                    # plot title
)
```



Step 3. Conduct posterior inference using the bivariate normal approximation. The **MASS** package has an `mvrnorm()` function to simulate from a multivariate normal distribution. For example, we can simulate draws and plot them on the contour plot:

```
library(MASS)
post_draws <- mvrnorm(1000, mu = fitted_model$mode, Sigma = fitted_model$var)

mycontour(lbinorm, limits = c(xlo = 0, xhi = 15, ylo = 0, yhi = .15),
          npar, main = "Approximate log posterior")
points(post_draws[,1], post_draws[,2])
title(main = "Approximate log posterior", xlab = expression(alpha),
      ylab = expression(beta))
```



Another two-parameter example

Another two-parameter example attempts to model stomach cancer mortality data for 20 cities in Missouri. The `cancermortality` data set can be found in the **LearnBayes** package:

```
data("cancermortality", package = "LearnBayes")
str(cancermortality)
```

```
## 'data.frame':    20 obs. of  2 variables:
##  $ y: int  0 0 2 0 1 1 0 2 1 3 ...
##  $ n: int  1083 855 3461 657 1208 1025 527 1668 583 582 ...
```

Here `y` is the number of cancer deaths and `n` is the number of people at risk between 45 and 64 in the city.

At first, a Binomial model seems to make sense (the number of people who die of cancer out of the total number at risk); however, these data display more variation than the binomial model allows (i.e. they are overdispersed)

A beta-binomial model is a better idea for the likelihood:

$$f(y_i|\eta, K) = \binom{n_i}{y_i} \frac{\text{Beta}(K\eta + y_i,\ K(1-\eta) + (n_i - y_i))}{\text{Beta}(K\eta,\ K(1-\eta))}$$

where $\eta \in (0,1)$ is the mean, $K > 0$ is the precision, and $\text{Beta}(a,b) = \dfrac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$.

Suppose the researchers propose the vague prior:

$$\pi(\eta, K) \propto \frac{1}{\eta(1-\eta)} \cdot \frac{1}{(1+K)^2}.$$

So the posterior, $p(\eta, K|\mathbf{y})$, is proportional to

$$\frac{1}{\eta(1-\eta)} \cdot \frac{1}{(1+K)^2} \prod_{i=1}^{20} \frac{\text{Beta}(K\eta + y_i,\ K(1-\eta) + (n_i - y_i))}{\text{Beta}(K\eta,\ K(1-\eta))}$$

Our task is now to use the Bayesian CLT to approximate the posterior distribution.

Step 1. Define a log posterior density function.

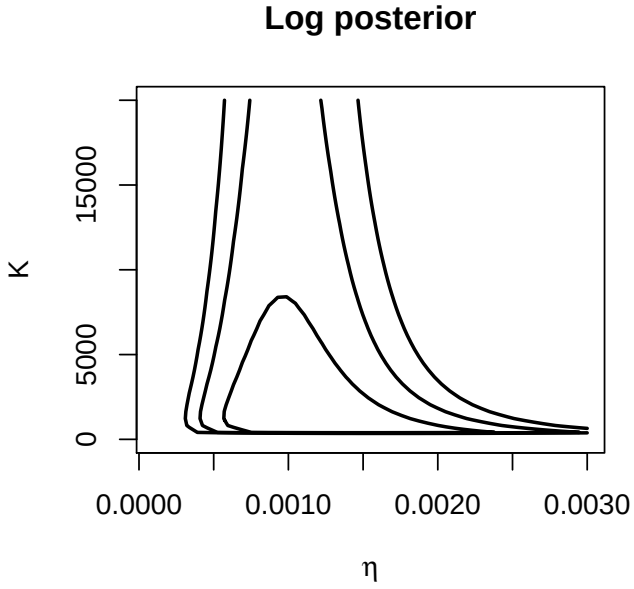
As before, we start by defining the log posterior:

```
lbetabinom_post0 <- function(theta, data) {
  eta <- theta[1]
  K   <- theta[2]

  y <- data[,1]
  n <- data[,2]

  N <- length(y)

  lpost <- 0
  for(i in 1:N) {
    lpost <- lpost +
      lbeta(K * eta + y[i], K * (1 - eta) + n[i] - y[i])
  }
  lpost <- lpost - N * lbeta(K * eta, K * (1 - eta))
```

```
  lpost <- lpost - log(eta) - log(1 - eta) - 2 * log(1 + K)

  return(lpost)
}
```

Note that the `lbeta()` function provides the natural log of the beta function, and is more efficient than nesting the functions. Let's take a look at a contour plot of this log posterior:

```
mycontour(lbetabinom_post0, limits = c(.0001, .003, 1, 20000),
          data = cancermortality, drawlabels = FALSE)
title(xlab = expression(eta), ylab = "K", main = "Log posterior")
```



The log posterior is not elliptical. This can pose challenges since the bivariate normal density is elliptical, so deriving the MAP estimate and Hessian from this parameterization will be fraught with error. In this type of situation, it's best to reparameterize the log posterior so that it is more elliptical. A general guideline is that you should transform your parameters to be real valued. In this example our parameters are not real valued:

- $\eta \in (0, 1)$
- $K > 0$

but we can find transformations of these parameters that are real valued:

- $\theta_1 = \log\left(\frac{\eta}{1-\eta}\right) \in \mathbb{R}$
- $\theta_2 = \log(K) \in \mathbb{R}$

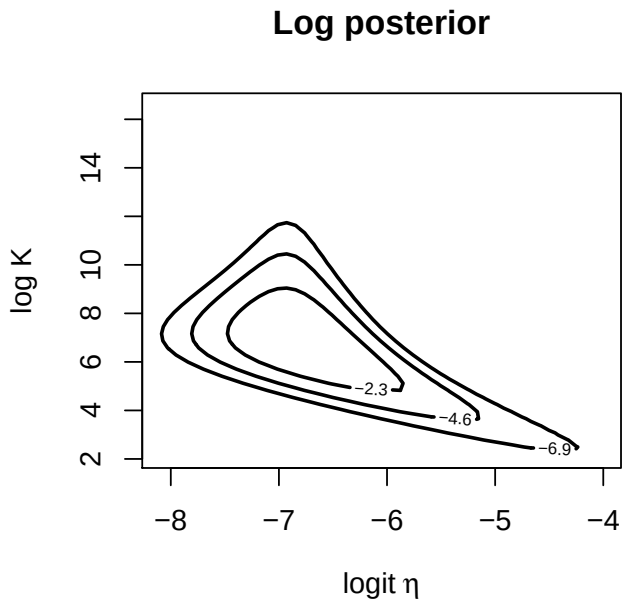
Now, we can recode the log posterior, defining η and K in terms of θ_1 and θ_2:

```
lbetabinom_post <- function(theta, data) {
  theta1 <- theta[1]
  theta2 <- theta[2]

  eta <- exp(theta1) / (1 + exp(theta1))
  K   <- exp(theta2)

  y <- data[,1]
  n <- data[,2]
```

```
  N <- length(y)

  lpost <- 0
  for(i in 1:N) {
    lpost <- lpost +
      lbeta(K * eta + y[i], K * (1 - eta) + n[i] - y[i])
  }
  lpost <- lpost - N * lbeta(K * eta, K * (1 - eta))
  lpost <- lpost - log(eta) - log(1 - eta) - 2 * log(1 + K)

  return(lpost)
}
```

This log posterior is much more elliptical, though certainly not "perfect."

```
mycontour(lbetabinom_post, limits = c(-8.1, -4, 2.2, 16.5), data = cancermortality)
title(xlab = bquote("logit" ~eta), ylab = "log K", main = "Log posterior")
```

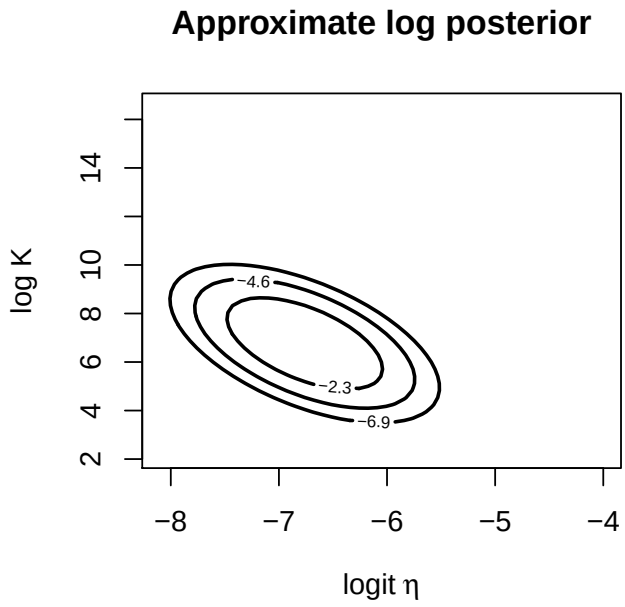


Step 2. Use `laplace()` to find the MAP estimates and variance-covariance matrix.

```
betabinom_fit <- laplace(lbetabinom_post, mode = c(-7,6),
                         data = cancermortality)
```

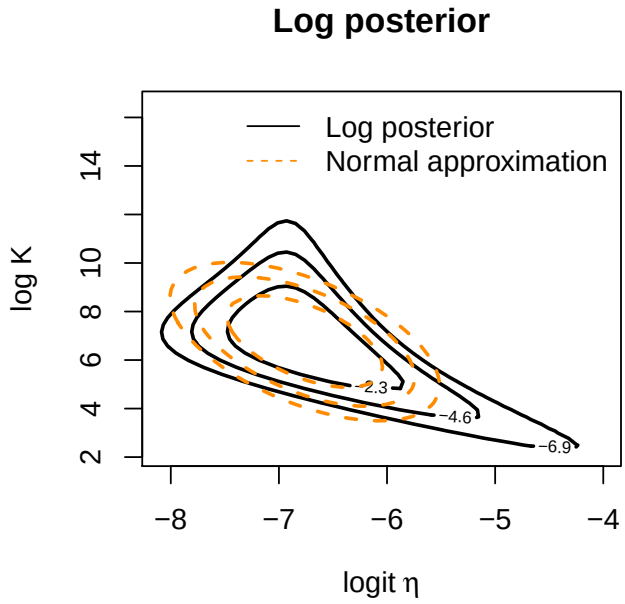
Again, we can plot the approximate posterior using the `mycountour()` function along with `lbinorm()`:

```
npar <- list(m = betabinom_fit$mode, v = betabinom_fit$var)
mycontour(lbinorm, c(-8.1, -4, 2.2, 16.5), npar)
title(xlab = bquote("logit" ~eta), ylab = "log K", main = "Approximate log posterior")
```



We can also overlay the true log posterior and the approximate log posterior to assess the fit:

```
npar <- list(m = betabinom_fit$mode, v = betabinom_fit$var)
mycontour(lbetabinom_post, limits = c(-8.1, -4, 2.2, 16.5), data = cancermortality)
mycontour(lbinorm, c(-8.1, -4, 2.2, 16.5), npar, add = TRUE,
          col= "darkorange", drawlabels = FALSE, lty =2)
title(xlab = bquote("logit" ~eta), ylab = "log K", main = "Log posterior")
legend("topright", c("Log posterior", "Normal approximation"),
       lty = c(1, 2), col = c("black", "darkorange"), bty = "n")
```



Step 3. Conduct posterior inference.

Again, we can simulate from the approximate posterior and conduct inference as usual. One important thing to keep in mind with a reparameterized posterior is that you will need to back-transform your parameters to the original scale if the interpretation relies on it.

```r
post_draws <- mvrnorm(1000, mu = betabinom_fit$mode, Sigma = betabinom_fit$var)

# transformed scale
theta1_draws <- post_draws[,1]
theta2_draws <- post_draws[,2]

# original parameters
eta_draws  <- exp(theta1_draws) / (1 + exp(theta1_draws))
k_draws    <- exp(theta2_draws)

# posterior analysis - eta
mean(eta_draws)
```

```
## [1] 0.001219557
```

```r
sd(eta_draws)
```

```
## [1] 0.0004196072
```

```r
quantile(eta_draws, probs = c(0.05, 0.95))
```

```
##           5%          95%
## 0.0006637702 0.0019722698
```

```r
# posterior analysis - K
mean(k_draws)
```

```
## [1] 1299.851
```

```r
sd(k_draws)
```

```
## [1] 1367.817
```

```r
quantile(k_draws, probs = c(0.05, 0.95))
```

```
##        5%       95%
##  209.4698 3785.0746
```