Stats 102C - Lecture 8-2

Miles Chen

Week 8 Wednesday

Section 1

Hierarchical Models

Gibbs Sampler

There is a target multivariate PDF $P(\mathbf{x})$

$\mathbf{x}$ is a d-dimensional vector $[x_1, x_2, \ldots, x_d]$

At time t, $\mathbf{x}_{(t)} = [x_{1(t)}, x_{2(t)}, \ldots, x_{d(t)}]$

To produce the next draw $\mathbf{x}_{(t+1)} = [x_{1(t+1)}, x_{2(t+1)}, \ldots, x_{d(t+1)}]$, we will draw from univariate conditional distributions. (Factoring a joint distribution into univariate conditional distributions can be hard.)

- $x_{1(t+1)} \sim P(x_1 | x_{2(t)}, \ldots, x_{d(t)})$
- $x_{2(t+1)} \sim P(x_2 | x_{1(t+1)}, x_{2(t)}, \ldots, x_{d(t)})$
- $x_{3(t+1)} \sim P(x_3 | x_{1(t+1)}, x_{2(t+1)}, x_{4(t)} \ldots, x_{d(t)})$
- $\ldots$
- $x_{i(t+1)} \sim P(x_i | x_{1(t+1)}, \ldots, x_{i-1(t+1)}, x_{i+1(t)} \ldots, x_{d(t)})$
- $\ldots$
- $x_{d(t+1)} \sim P(x_d | x_{1(t+1)}, \ldots, x_{d-1(t+1)})$

Hierarchical models

The Gibbs Sampler works well for Hierarchical models.

Hierarchical models are set up using conditional distributions, so deriving the conditional distribution will not be necessary.

Let's revisit the Bayesian Beta-Binomial Baseball example.

We said each player has some random variable θ which is their batting average. The batting average is the probability of getting a hit at an at-bat.

We said there is a prior distribution for θ. We chose a Beta distribution with shape parameters 81 and 219. This produced a distribution where almost all values are above 0.2 and below 0.35.

The choice of 81 and 219 is a bit arbitrary. We chose these because they produced a distribution with properties we like.

Hierarchical models

Based on Chapter 9 from Doing Bayesian Data Analysis

In a hierarchical model, we will not choose the parameters of the prior distribution (the beta distribution) arbitrarily, like we did when $\alpha = 81$ and $\beta = 219$.

In a hierarchical model, we treat the values of α and β as random variables themselves.

This means that the shape parameter α is a random variable and has its own random distribution. The same is true for β: it is a random variable with its own distribution.

α and β are parameters of the prior distribution and are called *hyperparameters*

Hyperprior distributions

The random distributions that produce α and β are known as hyperprior distributions because they produce the hyperparameters $alpha$ and $beta$.

The hyperprior distributions may also have its own parameters. These are called *hyperhyperparameters*. I'm not joking. We can choose to arbitrarily select the hyperhyperparameters ourselves, or we can also place another hyper-distribution on it, to infinity.

We generally try to stop having more than a couple layers of hyperdistributions.

Reexpressing α and β as the result of a random process

In the baseball example, we chose $\alpha = 81$ and $\beta = 219$. The numbers 81 and 219 are sometimes called pseudo-counts. Prior to seeing data from a player, we can pretend that the typical baseball player who qualifies for playing in the major leagues may have 81 "hits" and 219 "misses/outs". These are not actual data values, but imaginary counts of hits and outs.

These values ($\alpha = 81$ and $\beta = 219$) will produce a beta distribution with a peak (mode) of $\omega = \frac{81-1}{81+219-2} = 0.2685$, which seems to be a fairly reasonable value for a batting average.

We can arrive at the values 81 and 219 by taking $\omega = 0.2685$ and $K = 300$.

- $\alpha = \omega(K-2)+1 = 0.2685(298)+1 = 81$
- $(1-\omega)(K-2)+1 = 0.7315(298)+1 = 219.$

We treat K as an arbitrarily selected constant of 300.

We will treat ω as a random value drawn from a hyperprior distribution. The selection of ω will then determine the values of α and β.

A Hyperprior distribution for α and β

We can say the value ω itself comes from a beta distribution.

This Beta distribution is the hyperprior distribution of ω which is used to calculate α and β.

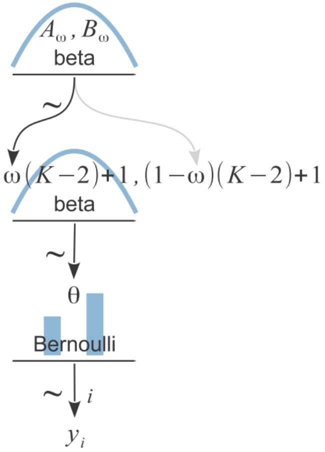
We'll need to select some values for shape parameters of the beta distribution used to generate ω. These will be A_ω and B_ω

The Hierarchical model for a Baseball Player

- $K = 300$
  - K is an arbitrarily selected constant
- $\omega \sim \text{Beta}(A_\omega, B_\omega)$
  - The mode ω is a random value drawn from the hyperprior distribution whose parameters are arbitrarily selected.
- $\alpha = \omega(K-2)+1, \beta = (1-\omega)(K-2)+1$
  - Shape parameters α, β are determined by ω and K
  - They determine the beta distribution used to generate θ
- $\theta \sim \text{Beta}(\alpha, \beta)$
  - The player's batting average θ is randomly drawn from this beta distribution.
- $x_i \sim \text{Bernoulli}(\theta)$
  - Whether the player gets a hit or out is a random outcome of the Bernoulli distribution

Hierarchical Diagram

From Doing Bayesian Data Analysis, Chapter 9



Using the Hierarchical model for multiple players

We can use the same hierarchical model to explain the data (hits and outs) of multiple players.

Each player has their own batting average. θ_s

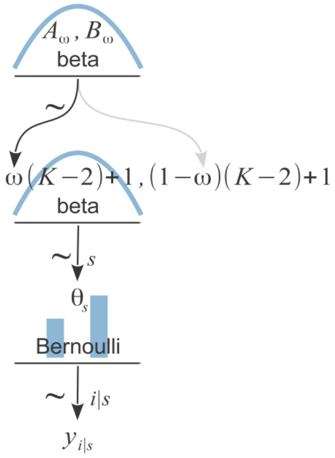
- Player 1 has batting average θ_1
- Player 2 has batting average θ_2
- Player s has batting average θ_s

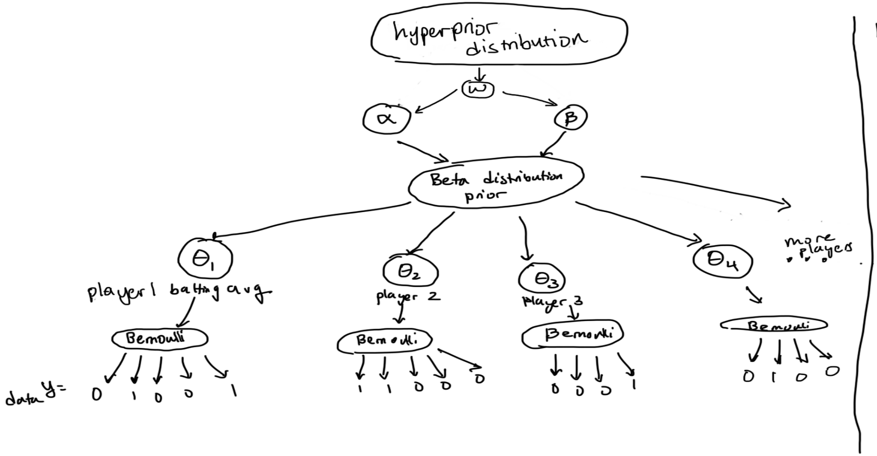
The Hierarchical model for a Baseball Player

- $K = 300$
  - K is an arbitrarily selected constant
- $\omega \sim \text{Beta}(A_\omega, B_\omega)$
  - The mode ω is a random value drawn from the hyperprior distribution whose parameters are arbitrarily selected.
- $\alpha = \omega(K-2)+1, \beta = (1-\omega)(K-2)+1$
  - Shape parameters α, β are determined by ω and K
  - They determine the beta distribution used to generate θ
- $\theta_s \sim \text{Beta}(\alpha, \beta)$
  - Each player's batting average θ_s is randomly drawn from this beta distribution.
- $y_i|s \sim \text{Bernoulli}(\theta_s)$
  - Whether we see a hit or out is a random outcome of the Bernoulli distribution that depends on the batting average θ_s of player s

Hierarchical Diagram

From Doing Bayesian Data Analysis, Chapter 9




hyperprior distribution
w
α
β
Beta distribution prior
θ1
player1 batting avg
Bernoulli
data y= 0 1 0 0 1
θ2
player 2
Bernoulli
1 1 0 0 0
θ3
player 3
Bernoulli
0 0 0 1
θ4
Bernoulli
0 1 0 0
more players
. . .

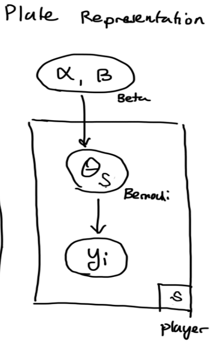


Plate Representation
α, β
Beta
θs
Bernoulli
yi
s
player

Bayesian Inference

Let's say we have a big matrix of data of all the hits/outs of 10 players.

Y_{si} is player the hit or out of player s

We want to get estimates of all the unknown parameters.

There are a total of 11 parameters to estimate:

- There are 10 θ_s values to estimate - one for each player
- We must also estimate ω which determine α and β in the beta prior distribution.
- The other parameters K and A_ω and B_ω have been arbitrarily selected and will not be inferred from the data.

The total join posterior distribution given all of the data is:

$$\begin{aligned}
P(\theta_1, \theta_2, \ldots, \theta_{10}, \omega | \mathbf{Y}, A_\omega, B_\omega, K) &= \frac{P(\mathbf{Y}|\theta_1, \theta_2, \ldots, \theta_{10}, \omega) P(\theta_1, \theta_2, \ldots, \theta_{10}, \omega | A_\omega, B_\omega, K)}{P(\mathbf{Y})} \\
&\propto P(\mathbf{Y}|\theta_1, \theta_2, \ldots, \theta_{10}, \omega) P(\theta_1, \theta_2, \ldots, \theta_{10}, \omega | A_\omega, B_\omega, K) \\
&\propto P(\mathbf{Y}, \theta_1, \theta_2, \ldots, \theta_{10}, \omega | A_\omega, B_\omega, K)
\end{aligned}$$

The last row above is just the joint distribution of everything. Because of the hierarchical nature of the data, the hits/outs of player 1 $\mathbf{y}_1 = [y_{11}, y_{12}, \ldots, y_{1i}]$ only depend on the batting average θ_1 of player 1.

This allows us to factor the joint distribution into many conditional distributions.

$$\begin{aligned}
&\propto P(\mathbf{Y}, \theta_1, \theta_2, \ldots, \theta_{10}, \omega | A_\omega, B_\omega, K) \\
&\propto P(\mathbf{y}_1|\theta_1) P(\mathbf{y}_2|\theta_2) \ldots P(\mathbf{y}_{10}|\theta_{10}) P(\theta_1|\omega, K) P(\theta_2|\omega, K) \ldots P(\theta_{10}|\omega, K) P(\omega|A_\omega, B_\omega) \\
&\propto P(\mathbf{y}_1|\theta_1) P(\theta_1|\omega, K) P(\mathbf{y}_2|\theta_2) P(\theta_2|\omega, K) \ldots P(\mathbf{y}_{10}|\theta_{10}) P(\theta_{10}|\omega, K) P(\omega|A_\omega, B_\omega) \\
&\propto P(\mathbf{y}_1|\theta_1) P(\theta_1|\alpha, \beta) P(\mathbf{y}_2|\theta_2) P(\theta_2|\alpha, \beta) \ldots P(\mathbf{y}_{10}|\theta_{10}) P(\theta_{10}|\alpha, \beta) P(\omega|A_\omega, B_\omega)
\end{aligned}$$

A closer look

$$\propto P(\mathbf{y}_1|\theta_1)P(\theta_1|\alpha,\beta)P(\mathbf{y}_2|\theta_2)P(\theta_2|\alpha,\beta)\dots P(\mathbf{y}_{10}|\theta_{10})P(\theta_{10}|\alpha,\beta)P(\omega|A_\omega,B_\omega)$$

We have factored the joint distribution of everything into distributions of each row in $\mathbf{Y}$ times distributions of each individual batting average θ_s and the distribution of ω.

- Each row in $\mathbf{Y}$ is the result of random draws from a Bernoulli distribution that depends only on the player's batting average θ_s
- Each players batting average θ_s is the result of a random draw from a Beta distribution that depends on α and β which are determined by the ω value and K
- The value ω is the result of a random draw from a Beta distribution where the values A_ω, B_ω, K are arbitrarily determined hyperhyperparameters.

We can create a Gibbs sampler that will take advantage of this structure of many conditional probabilities.

Sampling from the Posterior distributions

The posterior distribution that we want is a distribution over 11 variables: the ten θ values (one for each player) and the ω value which determines the prior distribution of θ.

Because of the hierarchical nature, when we want to sample a value for θ_s, the only values that are relevant are the ones that involve θ_s:

$$\text{Posterior of everything} \propto P(\mathbf{y}_1|\theta_1)P(\theta_1|\alpha,\beta)P(\mathbf{y}_2|\theta_2)P(\theta_2|\alpha,\beta)\ldots P(\mathbf{y}_{10}|\theta_{10})P(\theta_{10}|\alpha,\beta)P(\omega|A_\omega,B_\omega)$$

can be simplified to:

$$\text{Posterior of } \theta_1 \propto P(\mathbf{y}_1|\theta_1)P(\theta_1|\alpha,\beta)$$

Sampling from the Posterior distributions

We are familiar with the relationship between the posterior distribution of Bernoulli / Binomial data with a Beta distribution prior.

- $y_{[s,i]}$ are the individual 0 and 1 drawn from a Bernoulli for player s
- n_s is the number of trials for player s
- The posterior distribution of θ is:

$$\theta_s \sim \text{Beta}(\alpha + \sum y_{[s,i]}, \beta + n_s - \sum y_{[s,i]})$$

If we define k_s as the total number of hits for player s, $k_s = \sum y_{[s,i]}$, we have a Binomial likelihood.

In our Gibbs sampler, we will sample a random value θ_s from this posterior distribution.

$$\theta_s \sim \text{Beta}(\alpha + k_s, \beta + n_s - k_s))$$

Gibbs Sampler

```r
for(j in 1:iter){ # total number of iterations
  # Calculate alpha and beta based on current value of omega
  alpha <- omega * (K - 2) + 1
  beta <- (1 - omega) * (K - 2) + 1
  # Gibbs sampler
  # for each player, sample a random theta from the posterior distribution
  # k (vector of number of hits) exists in the global environment
  # n (vector of number of at bats) exists in global environment
  for(s in 1:n_players){
    thetas[j, s] <- rbeta(1, alpha + k[s], beta + n[s] - k[s])
  }
  # ... more code ...
}
```

Omega

Sampling from the conditional posterior distribution for ω is not as straightforward.

No problem!

We can use the Metropolis Algorithm.

For the Metropolis Algorithm, we just need a function that is proportional to the posterior distribution of ω.

From Bayes Rule, we know

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

Because we only need something proportional, there's no need to worry about the denominator in Bayes Rule!

Posterior distribution of omega

The hyperprior distribution for ω is Beta(A_ω, B_ω).

I arbitrarily select $A_\omega = 3$ and $B_\omega = 7$.

In our hierarchical model, ω determines the values of α and β in the prior distributions of θ_s.

$\alpha = \omega(K-2)+1$ and $\beta = (1-\omega)(K-2)+1$. K has been arbitrarily selected to be 300.

$$\text{posterior} \propto P(\mathbf{y}_1|\theta_1)P(\mathbf{y}_2|\theta_2)\ldots P(\mathbf{y}_{10}|\theta_{10})P(\theta_1|\alpha,\beta)P(\theta_2|\alpha,\beta)\ldots P(\theta_{10}|\alpha,\beta)P(\omega|A_\omega,B_\omega)$$

Posterior function

posterior $\propto P(\mathbf{y}_1|\theta_1)P(\mathbf{y}_2|\theta_2)\dots P(\mathbf{y}_{10}|\theta_{10})P(\theta_1|\alpha,\beta)P(\theta_2|\alpha,\beta)\dots P(\theta_{10}|\alpha,\beta)P(\omega|A_\omega,B_\omega)$

With our sampled values of θ_s and the current or proposed value of ω, we can calculate the posterior function easily:

- $P(\mathbf{y}_s|\theta_s)$ can be found using the binomial PMF.
  - `dbinom(k, n, theta)` where `k` is sum of $\mathbf{y}_s$ and `n` is the length of $\mathbf{y}_s$.
  - We find this for each player's y_s and take the product.
- $P(\theta_s|\alpha,\beta)$ can be found using the beta PDF.
  - We first must calculate $\alpha = \omega(K-2)+1$ and $\beta = (1-\omega)(K-2)+1$
  - We can then use `dbeta(theta, alpha, beta)`
  - We find this for each player's θ_s and take the product.
- $P(\omega|A_K,B_k)$ can be found using the beta PDF.
  - 'dbeta(omega, A_k, B_k)"
- To avoid computational issues with values near 0, we will take the log prob and use a sum.

Probability function for Metropolis:

The following function is proportional to the PDF of the posterior distribution of omega.

```r
logprob <- function(omega, K, theta){
  alpha <- omega * (K - 2) + 1
  beta <- (1 - omega) * (K - 2) + 1
  # little k and n are found in the global environment
  sum(dbinom(k, n, theta, log = TRUE) + dbeta(theta, alpha, beta, log = TRUE)
    dbeta(omega, alpha_omega, beta_omega, log = TRUE)
}
```

Proposal Function for Metropolis

For the Metropolis algorithm, we also need a function to propose a new value of omega based on the current value of omega.

```r
propose <- function(omega, sd = .05){
  omega_proposed <- rnorm(1, mean = omega, sd = sd)
  omega_proposed <- min(omega_proposed, 1) # to prevent value from being > 1
  omega_proposed <- max(omega_proposed, 0) # to prevent value from being < 0
  omega_proposed
}
```

Gibbs + Metropolis

```
for(j in 1:iter){ # total number of iterations
  alpha <- omega * (K - 2) + 1
  beta <- (1 - omega) * (K - 2) + 1
  # Gibbs sampler
  # for each player, sample a random theta from the posterior distribution
  for(s in 1:n_players){
    thetas[j, s] <- rbeta(1, alpha + k[s], beta + n[s] - k[s])
  }
  # the vector theta has all of the 'latest' values of the sampled thetas
  theta <- thetas[j, ]
  # metropolis algorithm to sample values of omega
  omega_proposed <- propose(omega, sd = 0.03)
  logratio <- logprob(omega_proposed, K, theta) - logprob(omega, K, theta)
  u <- runif(1)
  if(log(u) < logratio){
    omega <- omega_proposed
  }
  omegas[j] <- omega
}
```

Demo Code

Gibbs Sampling Demo

Section 2

JAGS

JAGS

JAGS is a program that can perform Gibbs sampling quickly for complex hierarchical models.

JAGS stands for "Just another Gibbs Sampler". It is developed using the BUGS modeling language (BUGS stands for Bayesian inference Using Gibbs Sampling).

You must download and install JAGS from http://mcmc-jags.sourceforge.net/

Once you have JAGS installed, you can interact with it using R. There are a few interfaces to JAGS from R and the one I will use is package `R2jags`.

```
library(R2jags)
```

The BUGS model specification language

Using JAGS does not require us to figure out the conditional distributions of each parameter.

Instead, we describe the hierarchical model using the BUGS language.

The BUGS language allows us to state that certain parameters come from certain distributions. It also allows us to specify how the distributions and parameters are related.

The BUGS language is reminiscent of R but there are distinctions and you will want to consult the documentation for what parameters are expected.

https://chjackson.github.io/openbugsdoc/Manuals/Contents.html

https://chjackson.github.io/openbugsdoc/Manuals/ModelSpecification.html

Recap of our Hierarchical model

Revisit Slides 12 - 14 to see our hierarchical model

Our model in the BUGS Language

```
bayes.mod <- function() {
  for(s in 1:n_players){
    theta[s] ~ dbeta(alpha, beta)
    for(j in 1:n_atbats[s]){
      Y[s, j] ~ dbern(theta[s])
    }
  }
  omega ~ dbeta(Aomega, Bomega)
  alpha <- omega * (K - 2) + 1
  beta <- (1 - omega) * (K - 2) + 1
  Aomega <- 3
  Bomega <- 7
  K <- 300
}
```

Line by line

```
bayes.mod <- function() {
# ...
}
```

When you specify the model in R2jags, you wrap it in a function. This is not a function that will every be run in R, but this is how the R2jags interface expects a model to be specified.

It is also possible to write a separate text file with the model specification.

Line by line

```
for(s in 1:n_players){
  theta[s] ~ dbeta(alpha, beta)
# ...
}
```

In our model, we write that for each player `s` has a value of `theta[s]` which is drawn from a beta distribution with parameters `alpha` and `beta`.

We haven't defined `alpha` and `beta` yet, but as long we define them later in the model, it will work.

Line by line

```
for(j in 1:n_atbats[s]){
  Y[s, j] ~ dbern(theta[s])
}
```

We reference the actual data `Y` a matrix with where each row represents a player. Each value in that row is a 0 or 1 representing a hit or miss for each at bat. Each of the values in the row are random draws from a Bernouli distribution, where the probability is `theta[s]` (which was specified earlier)

```
  omega ~ dbeta(Aomega, Bomega)
```

`omega` comes from a beta distribution. This is our hyperprior distribution and it depends on the hyperhyperparameters Aomega and Bomega.

```
alpha <- omega * (K - 2) + 1
beta <- (1 - omega) * (K - 2) + 1
Aomega <- 3
Bomega <- 7
K <- 300
```

These lines define values that are determined by other values. Note these values have `<-` to define them, while the values that came from random distributions used `~`.

`alpha` and `beta` are defined using `K` and the random value of `omega`.

`Aomega`, `Bomega`, and `K` are arbitrarily chosen values and defined here.

Other requirements for JAGS

```
data_jags <- list("Y", "n_atbats", "n_players")
bayes_mod_params <- c("theta", "omega")
```

Using JAGS requires a few lists and other functions to be passed.

You must specify the data that will be used. This is done by providing a list of character names of the data objects in the global environment.

You must also specify the parameters that must be estimated by the Gibbs sampler. In this case, `theta` is a vector of length 10, and `omega` is a single value.

```r
bayes_mod_inits <- function() {
  list("theta" = rbeta(n_players, 3, 7), "omega" = rbeta(1, 3, 7))
}
```

We must also provide a function that will initialize values of the parameter list.

This function is written in R.

You can use any of R's random generating functions. You must make sure that the random function will provide a valid number of the correct length. For example, the random function specifying `theta` must provide 10 numbers because there are 10 values of theta to estimate.

Also using `rnorm(1)` to specify `omega` could result in an error because `rnorm` might produce a negative value and that would cause problems in the model.

```r
bayes_mod_fit <- jags(data = data_jags, inits = bayes_mod_inits,
                      parameters.to.save = bayes_mod_params,
                      n.chains = 3, n.iter = 15000,
                      n.burnin = 1000, model.file = bayes.mod)

bayes_mod_fit

plot(bayes_mod_fit)
```