Fitting Bayesian Regression Models

Math 315, Fall 2019

Example

Barberan and Leff (2019) published data on dust samples taken from the ledges above doorways in the continental US. Bioinformatics processing detects the presence or absence of 763 species (technically operational taxonomic units) of fungi. The log of the number of fungi species present in the sample, which is a measure of species richness. In this example, our objective is to determine which factors influence a home's species richness. For each home, eight explanatory variables (i.e. covariates) are included in this example:

Variable	Description
`lnspecies`	natural log of the number of fungi species present in the sample
`long`	longitude
`lat`	latitude
`temp`	annual mean temperature of the location
`precip`	annual mean precipitation of the location
`NPP`	net primary productivity (rate at which all the autotrophs in an ecosystem produce net useful chemical energy)
`elev`	elevation
`house`	indicator that the house is a single-family home
`bedrooms`	number of bedrooms in the home

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

The Bayesian regression model

Regardless of whether a frequentist or Bayesian approach is utilized, the core goals of regression remain the same. Namely, we wish to model the response variable as a linear combination of the explanatory variables. Thus, for a normally distributed response, the "base model" for Y_i is the same:

$$Y_i|x_1, \ldots, x_p \overset{\text{iid}}{\sim} \mathcal{N}(\beta_0 + \beta_1 x_{i1}, + \cdots + \beta_p x_{ip},\ \sigma^2)$$

For a Bayesian analysis, we also need to place prior distributions on each regression coefficient, β_i, and the variance, σ^2. We will talk about a variety of approaches to specify the prior distributions, but for today you will focus on an approximate reference prior.

Fitting a regression model in JAGS

Your task today is to fit a Bayesian regression model in JAGS and explore the fitted model. To keep your coding tasks shorter, we'll focus only on two predictors: temperature and precipitation.

Standardizing predictors

For the purpose of setting priors it is helpful to standardize both the response and each predictor to have mean zero and variance one. While this is not required, it allows you to always think about variables on the same scale. This standardization is quickly implemented using the `scale()` function:

```
homes.std <- scale(homes)
```

JAGS data format

Next, you should separate the response vector from the matrix of predictors for use in JAGS.

```
Y <- homes.std[,1]                    # response vector
X <- homes.std[,c("temp", "precip")] # matrix with predictors
```

Then, store the necessary summary statistics in a list to pass to JAGS and specify some other key settings for MCMC.

```
n        <- length(Y)    # sample size
p        <- ncol(X) + 1  # no. predictors + intercept
X        <- cbind(1, X)

data   <- list(Y = Y, X = X, n = n, p = p) # data list
params <- c("beta", "sigma")               # parameters to track

burn     <- 10000 # burn-in period
n.iter   <- 20000 # no. posterior draws
thin     <- 10    # thin posterior by k = 10
n.chains <- 1     # run 1 chain
```

Notice that we add a column of 1's to the X matrix to represent the intercept.

JAGS model string

Now that you have standardized the data, you can fit a Bayesian regression model. To begin, consider fitting the objective Bayesian regression model with Jefferys' prior (p. 125):

$$Y_i|x_1, \ldots, x_p \overset{\text{iid}}{\sim} \mathcal{N}(\beta_0 + \beta_1 x_{i1}, + \cdots + \beta_p x_{ip},\ \sigma^2)$$
$$\pi(\boldsymbol{\beta}, \sigma^2) \propto \left(\sigma^2\right)^{-\frac{p}{2}-1}$$

However, JAGS does not have an improper "flat" prior function, so we must approximate a flat prior using an available density. One option is to use a uniform distribution; however, this is very clunky. A better option is to use:

$$b_j \overset{\text{iid}}{\sim} \mathcal{N}(0, b)$$
$$\sigma^2 \sim \text{InvGamma}(c, c)$$

where b is a large integer, say 10^6 and c is a small value, say 10^{-3}. To implement this in JAGS, you can use the following model string:

```
library(rjags)
reference_string <- textConnection("model{
  # Likelihood
  for(i in 1:n) {
    Y[i] ~ dnorm(inprod(X[i,], beta[]), taue)
  }

  # Priors
  for(j in 1:p) {
    beta[j] ~ dnorm(0, 1/pow(10, 3))
  }
  taue ~ dgamma(0.001, 0.001)
```

```
  sigma <- 1 / sqrt(taue)
}")
```

Remarks:

- the `inprod()` function performs an inner product. Here, it is calculating `X[i, 1] * beta[1] + X[i, 2] * beta[2] + X[i, 3] * beta[3]`.
- Remember that JAGS parameterizes the normal distribution using the precision, $1/\sigma^2$; thus, we are specifying $\beta_j \sim \mathcal{N}(0, 10^6)$ as our prior. This is also why we are using a gamma prior for τ_e rather than an inverse gamma prior for σ^2 (the two are equivalent).

Your turn:

1. Create scatterplots of `Y` against each predictor variable and summarize your findings.
2. Run the JAGS model specified in `jeffreys_string` and do the following:
   (a) Report a table with the posterior mean, SD, and 95% credible interval for each slope term. What did you learn about the relationship between the response and each predictor?
   (b) Report the fitted model equation.
3. Fit the frequentist regression model using the `lm()` function. How do the parameter estimates, their uncertainty estimates, and intervals compare to the Bayesian model?

Plotting the fitted model

While creating a table giving the mean, SD, and 95% interval for each regression coefficient is informative, often plots of the fitted model are necessary to communicate your findings to a non-technical audience.

Calculating fitted values

With only one predictor, plotting the fitted line equates to obtaining fitted values for a grid of x values and plotting the fitted values as a line, and credible intervals for each x on the grid as bounds. However, with two predictor variables plotting the fitted model requires a little more thought.

To communicate the fitted model with two predictors we can create two *counterfactual plots* (sometimes called effect plots), where we plot predictions over a grid of one variable while holding the other constant at its mean.

Let's first create a counterfactual plot for `temp` holding `precip = 0` (its mean). To begin, create a grid over `temp`:

```
temp_grid<- seq(-3, 3, by = .01)
```

Next, calculate the fitted values for each set of regression coefficients in your posterior samples, which I will assume are called `post_samples`. To do this, extract the regression coefficient posterior draws and store them in `beta_draws`. Then, create a `mu_link()` function that calculates the fitted values given a value of `temp` and 'precip.

```
beta_draws <- post_samples[[1]][,-4]
mu_link <- function(temp, precip) beta_draws[,1] + beta_draws[,2] * temp + beta_draws[,3] * precip
```

To calculate the fitted values, use `sapply()` to evaluate `mu_link()` at each value on the temperature grid. Notice that the additional argument `precip = 0` specifies the mean value of `precip`.

```
yhat_draws <- sapply(temp_grid, mu_link, precip = 0)
```

The result is a matrix, `yhat_draws`, with rows corresponding to each posterior draw and columns for each value on the grid. To calculate the mean fitted value, calculate column means. One way to so this is with the `apply()` function. Here, `2` denotes columns.

```
yhat_mean <- apply(yhat_draws, 2, mean) # vector of avg. yhat values
```

Similarly, we can calculate credible intervals for each temperature using `apply()`. In this example, we calculate HPDIs, but you could use `quantile()` to calculate standard equal-tail credible intervals:

```
library(HDInterval)
yhat_hpdi <- apply(yhat_draws, 2, hdi)
```

Finally, we can construct the counterfactual plot:

```
plot(yhat_mean ~ temp_grid, type = 'l', xlim = c(-3, 3), ylim = c(-3, 3), xlab = "temperature", ylab = "lo
lines(yhat_hpdi[1,] ~ temp_grid, lty = 2)
lines(yhat_hpdi[2,] ~ temp_grid, lty = 2)
```

Notice that we do not plot the observed data because this is a hypothetical plot of what the model predicts if precipitation was held at its mean. In our data set precipitation is not held constant at the mean, so adding the data in the background will not aid in the interpretation.

Your task:

1. Adapt the provided code to create a counterfactual plot for `precip`, holding `temp` at 0 (its mean).

Making Predictions

In addition to understanding the relationship between the predictors and the response, you may also want to predict the species richness. To properly account for uncertainty in our predictions, you should utilize the *posterior predictive distribution (PPD)*.

Predictions in R

To create the PPD entirely within R, you need posterior draws for each β_i and σ. With these quantities in hand, you can simulate data corresponding to the predictor values of interest using the fact that

$$Y|x_1, x_2 \overset{\text{iid}}{\sim} \mathcal{N}\left(\beta_0 + \beta_1 x_1 + \beta_2 x_2, \sigma^2\right)$$

For example, to make predictions for a home in a region with average temperature 1 standard deviation below the average (`temp = -1`) and precipitation at the average (`precip = 0`) we can use the following code to create the PPD:

```
# Let's us mu_link to calc. the mean again
preds <- rnorm(2000, mean = mu_link(temp = -1, precip = 0), sd = post_samples[[1]][,"sigma"])
```

Remarks:

- Here we ask for 2000 simulated values, one for each draw from the posterior. This will change based on your posterior draws.
- If you prefer working with data frames, your can convert `post_samples` to a data frame and then work with it. With a single chain, `as.data.frame(post_samples[[1]])` works. With multiple chains, you will need to bind the elements of the list together into a single data frame.
- You can generalize this code to add a prediction interval to your counterfactual plots.
- Once you have the PPD draws, you can plot them or calculate estimates as before.

Predictions in JAGS

While you can simulated the PPD in R, sometimes you may wish to draw these samples in JAGS. To do this, you need to append prediction code to the model string:

```
reference_string <- textConnection("model{
  # Likelihood
  for(i in 1:n) {
    Y[i] ~ dnorm(inprod(X[i,], beta[]), taue)
  }

  # Priors
  for(j in 1:p) {
    beta[j] ~ dnorm(0, 1/pow(10, 6))
  }
  taue ~ dgamma(0.001, 0.001)

  # Predictions
  for(i in 1:n_pred) {
    Y_pred[i] ~ dnorm(inprod(X_pred[i,], beta[]), taue)
  }
}")
```

Notice we must pass additional data to JAGS to make predictions:

- `X_pred`: a matrix where each row specifies the values of the predictors that should be used for prediction.
- `n_pred`: the number of predictions requested.

Suppose we wish to make predictions for a home in a region with average temperature 1 standard deviation below the average (`temp = -1`) and precipitation at the average (`precip = 0`); and a home in a region with `temp = 1` and `precip = 1`. The matrix `X_pred` is created using the code given below:

```
X_pred <- matrix(c(1, -1, 0, 1, 1, 1), ncol = 3, byrow = TRUE)
```

Your turn:

1. Fit a regression model using the new `reference_string` and draw samples from the posterior distribution and PPD. Remember to add `X_pred` and `n_pred` to the list passed to the `data` argument of `jags.model()`, and to update the `variable.names` vector to include `Y_pred`.
2. Give and interpret a 95% prediction interval for each home.

Residual plots

Regardless of the paradigm, it is important to check your model to see whether you have adequately captured the important features in the data set to detect potential model deficiencies. The usual suspects from Math 245 should still be checked:

1. A plot of the residuals vs. fitted values.
2. A plot of the residuals vs. each predictor.
3. A normal Q-Q plot of the residuals.

Let's begin with a plot of the residuals against the fitted values. Recall that the residuals are $e_i = y - \widehat{y}$. These can be calculated quickly using the summary of your posterior draws and a little matrix algebra. First, extract the beta coefficients from the summary:

```
post_sum <- summary(post_samples)$statistics
betas <- post_sum[1:3,"Mean"]
```

With the posterior means in hand, you can calculate the fitted values as $\widehat{\mathbf{Y}} = \mathbf{X}\widehat{\boldsymbol{\beta}}$. Thus, the residuals are given by:

```
y_hat <- X %*% betas
.resid <- Y - y_hat
```

Now we can create any of the above residual plots using familiar code:

```
plot(.resid ~ y_hat, xlab = "Fitted values", ylab = "Residuals")
abline(h = 0, col = "blue", lty = 2)
```

Your turn:

1. Create the residual plots outlined above and determine whether the model is adequate.

References

Barberan, Albert; Leff, Jonathan (2019): Homes_USA. figshare. Dataset. https://doi.org/10.6084/m9.figshare.1270900.v10