Stan Lab

CS 109b Staff 2/21/18 - 2/22/18

Motivation

• How do we normalize posteriors?

Normalization Constant Example:

Given a dataset $D = [x_1, x_2, \dots, x_n]$

Bayes Rule tells us that:

 $P(\theta \mid D) = \frac{P(D|\theta)P(\theta)}{P(D)}, \text{ where}$ $P(\theta) \text{ is our prior distribution,}$ $P(D \mid \theta) \text{ is our likelihood function,}$ $P(\theta \mid D) \text{ is our posterior distribution}$

Problem: How do we calculate P(D)?

 $P(D) = \int P(D, \theta) d\theta$, which can be very high-dimensional and difficult to compute.

Motivation

- How do we normalize posteriors?
 - Conjugate priors
 - Metropolis-Hastings
 - Other forms of MCMC
- Strong developments in both sampling algorithms and computational power have made Bayesian models more feasible

Background

- Stan was developed by Andrew Gelman (PhD from Harvard in 1990) and his lab at Columbia University
 - BDA
- Many predecessors to Stan such as Bugs, Jags, etc.
- Landmark paper: The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo
 - Inspired by ideas from physics

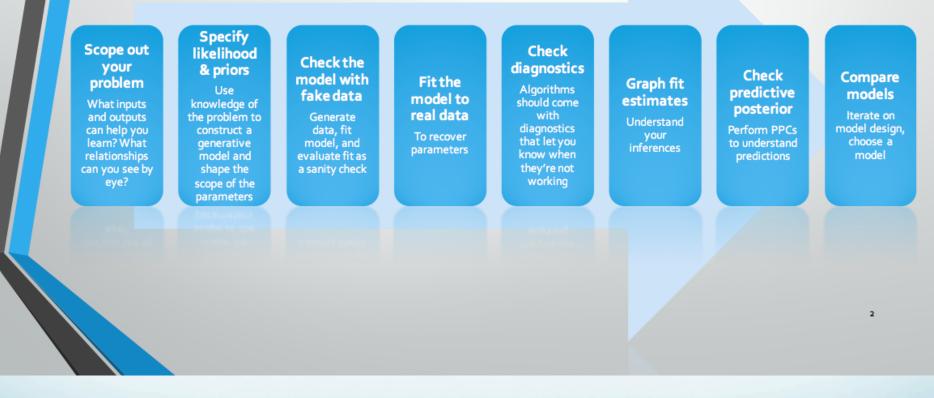
Bayesian Workflow

"How to structure the process of your analysis to maximise [sic] the odds that you build useful models."

-Jim Savage

Sean Talts Core Stan Developer

Bayesian Workflow



 $height \sim N(\alpha + \beta * weight, \sigma^2)$

- In Stan, code is structured in blocks
 - Functions
 - Data
 - Transformed Data (optional)
 - Parameters
 - Transformed Parameters (optional)
 - Model
 - Generated Quantities (optional)

- In Stan, code is structured in blocks
 - Functions
 - Data

```
data {
    int num_people;
    vector<lower=0>[num_people] weights;
    vector<lower=0> heights[num_people];
}
```

Notice how variables are statically-typed, rather than the dynamically-typed format you've used in R and in Python. You'll also have to specify whenever you want to end a line with a semicolon.

Stan also allows you to bound both data and parameters.

- In Stan, code is structured in blocks
 - Functions
 - Data
 - Parameters

```
parameters {
    real beta;
    real alpha;
    real<lower=0> sigma;
}
```

- In Stan, code is structured in blocks
 - Functions
 - Data
 - Parameters
 - Model

```
model {
    heights ~ normal(beta * weights + alpha, sigma);
}
```

- In Stan, code is structured in blocks
 - Functions
 - Data
 - Parameters
 - Model

```
model {
    heights ~ normal(beta * weights + alpha, sigma);
    beta ~ normal(0, 10); // cm/kg
    alpha ~ normal(50, 50); // avg cm for 0 kg
    sigma ~ normal(0, 5); // variation from average
}
```

- After we fit our model, we can perform a sanity check
 - 1) Draw parameter values from the posterior
 - 2) Generate data based on those parameter values
 - 3) Fit model to generated data
 - 4) Check if fit is reasonable

```
generated quantities {
   real<lower=0> heights[N];
   real beta = normal_rng(0, 10);
   real alpha = normal_rng(50, 50);
   real sigma = fabs(normal_rng(0, 5));
   for (n in 1:N)
        heights[n] = normal_rng(beta * weights[n] + alpha, sigma);
}
```

Your Turn!

- Open a blank R script and source the file "count_data.R", which is provided in the Lab materials
- Make sure you have the "rstan", "ggplot2", and "bayesplot" packages installed
- Try your best to fit the model described by the instructor