Introduction to Probablistic Programming

Stan, Hierarchical Models, and Bayesian Inference

Jordan Love

2021-03-08

Workshop Outline

- What is Probabilistic Programming?
- What is Stan?
- Syntax of Stan
- Stan Examples
- Hierarchical Modeling
- Bayesian Inference

If you are interested in following along, slides can be found on our github page:

Probablistic Programming

We typically write code using **static** variables.

- We assume once a variable is assigned a value, it represents only that value until explicitly altered.
- In order to implement a probabilistic procedure, we typically define a vector or matrix to hold multiple resulting realizations
- While possible, the resulting code can be lengthy, unorganized, and difficult to read depending on implementation

Probabilistic Programming

Probabilistic programming allows variables to be represented by **probability distributions**.

- This allows us to write far more concise code which is only understood within a probablistic context
- Probablistic programming languages define what sort of method is used to perform optimization / inference
- Replaces the focus (hopefully) on *implementation* with *conceptualization*.
- Typically used within a Bayesian Inference context

Probabilistic Languages

There are many different types of probabilistic languages which offer different benefits.

- BUGS
- JAGS
- Nimble
- Stan

What is Stan?

Stan is among the most accessible and well-supported probabilistic programming languages.

- Promoted heavily by Andrew Gelman et al, a prominent bayesian statistician
- **Stan** operates as a separate language and is accessible through R, Python, and more
- Built in C++, and uses many data types from C++ to be computationally efficient

As an example, let's consider the case where we want to estimate a simple mode:

$$y_i \sim N(\mu, \sigma)$$

Syntax in Stan

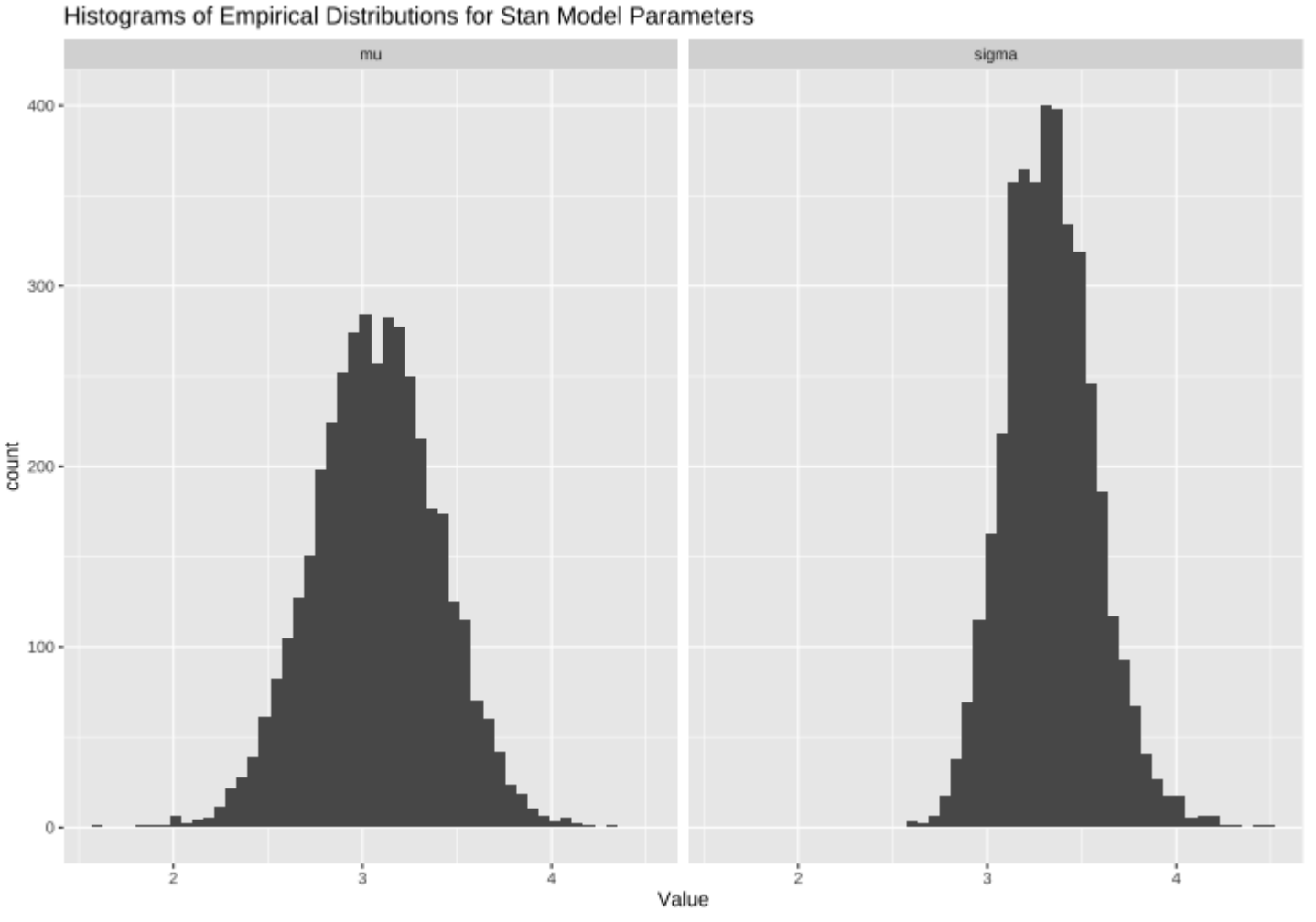
```
data {
  int N;
  vector[N] y;
}

parameters {
  real<lower=0> sigma;
  real mu;
}

model {
  for(i in 1:N){
    y[i] ~ normal(mu, sigma);
  }
}
```

Let's try running this model using simulated data.

Stan Output

```
## Inference for Stan model: 0b7974cfab2815607160caf0e96e2d8e.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd    2.5%     25%     50%     75%   97.5% n_eff Rh
## sigma    3.33    0.00 0.24    2.91    3.16    3.32    3.49    3.85  2870
## mu       3.07    0.01 0.33    2.43    2.85    3.07    3.29    3.72  3661
## lp__  -168.27    0.02 1.02 -171.10 -168.64 -167.97 -167.57 -167.29  1883
##
## Samples were drawn using NUTS(diag_e) at Tue Mar  9 12:42:46 2021.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Stan Output, Visualized



More Stan Examples

$$y_i \sim N(X_i\beta, \sigma)$$

```
data {
  int N; // Number of Observations
  int K; // Number of Covariates
  vector[N] y;
  matrix[N, K] X;
}

parameters {
  real<lower=0> sigma;
  vector[K] beta;
}

model {
  for(i in 1:N) {
    y ~ normal(X[i] * beta, sigma);
  }
}
```

More Stan Examples

$$y_i \in 0, 1$$

$$y_i \sim \text{Bernoulli}(p)$$

```
data {
  int N;
  int y[N];
}

parameters {
  real<lower=0, upper=1> p;
}

model {
  for(i in 1:N){
    y[i] ~ bernoulli(p);
  }
}
```

Program Blocks in Stan

```
  functions {
    // ... function declarations and definitions ...
  }
  data {
    // ... declarations ...
  }
  transformed data {
     // ... declarations ... statements ...
  }
  parameters {
     // ... declarations ...
  }
  transformed parameters {
     // ... declarations ... statements ...
  }
  model {
     // ... declarations ... statements ...
  }
  generated quantities {
     // ... declarations ... statements ...
}
```

Getting Started in Stan

Stan is one of the most well documented probabilistic programming languages available. Depending on what field you are in, you can often find a tutorial using an example with which you are familiar or interested in.

- PyStan provides an interface for Python Users
- RStan provides an interface for R Users

Those who work primarily in applied statistical contexts will benefit most from Stan. However, Stan can be a great tool to be aware of depending on the careers trajectory you are interested in.

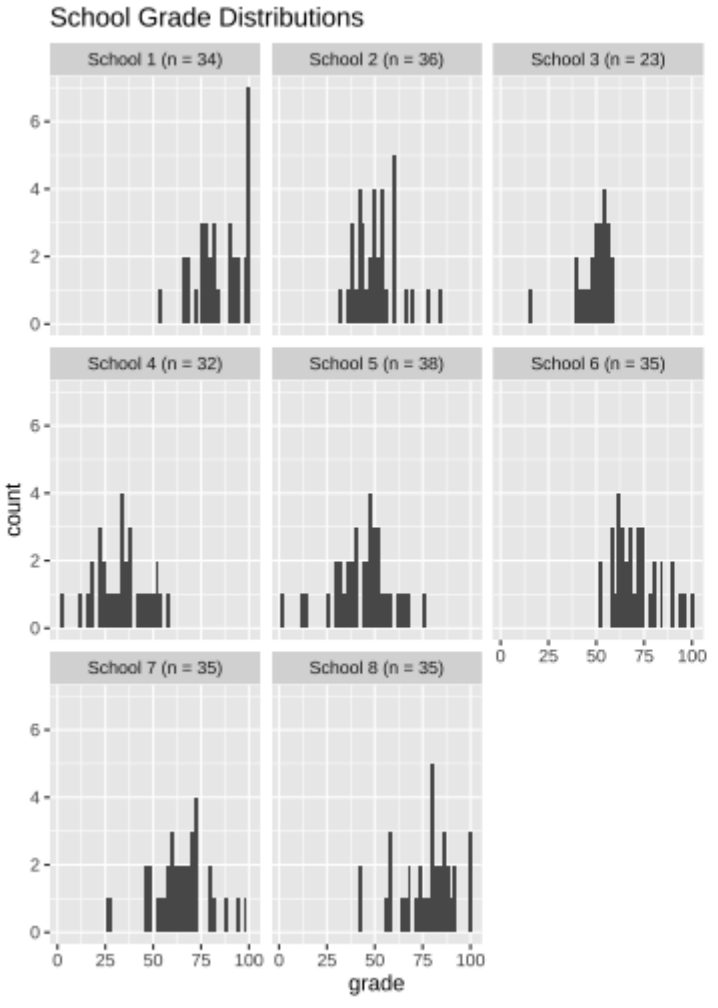
- Stan is easier to share than other, custom made statistical models
- Stan provides a common language to implement complex models
- Stan shines in Hierarchical modeling contexts

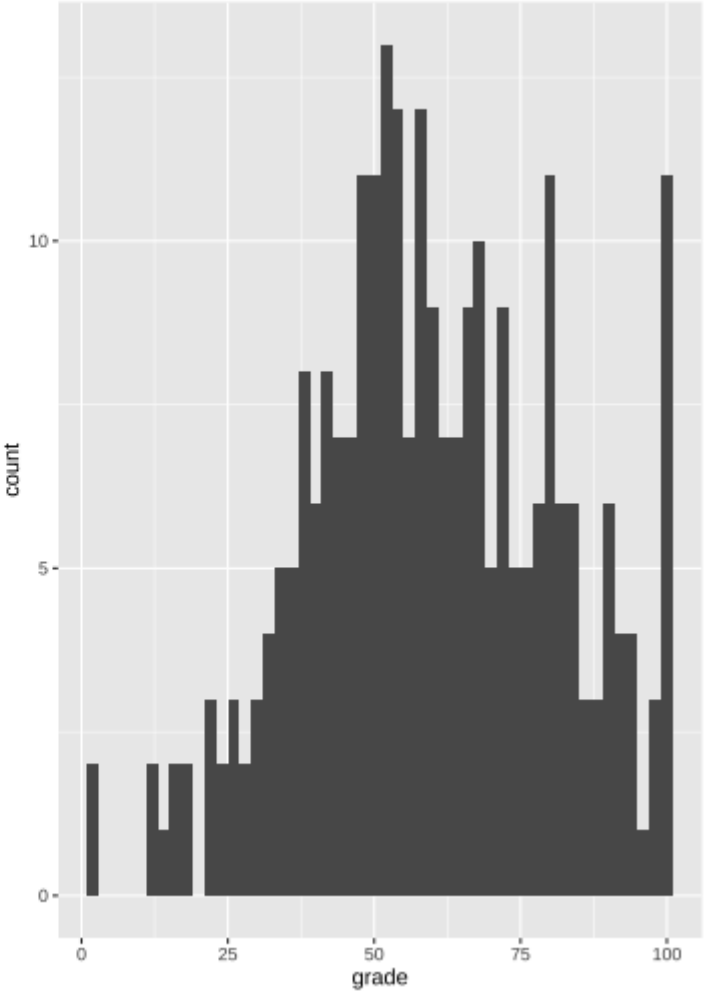
Questions or Discussion on Stan / Probabilistic Programming?

A Modeling Riddle

- We are interested in predicting student grades for a new school being built using data from 8 different schools existing schools
- Our data is given by a student id, a students score, and the students school
- The number of students at each school differs, so some schools may have more or less student information than others
- How should we use the data from each of the existing schools to predict student grades for the new school?

School Data


School Grade Distributions
School 1 (n = 34)
School 2 (n = 36)
School 3 (n = 23)
School 4 (n = 32)
School 5 (n = 38)
School 6 (n = 35)
School 7 (n = 35)
School 8 (n = 35)
count
grade
0
2
4
6
0
25
50
75
100

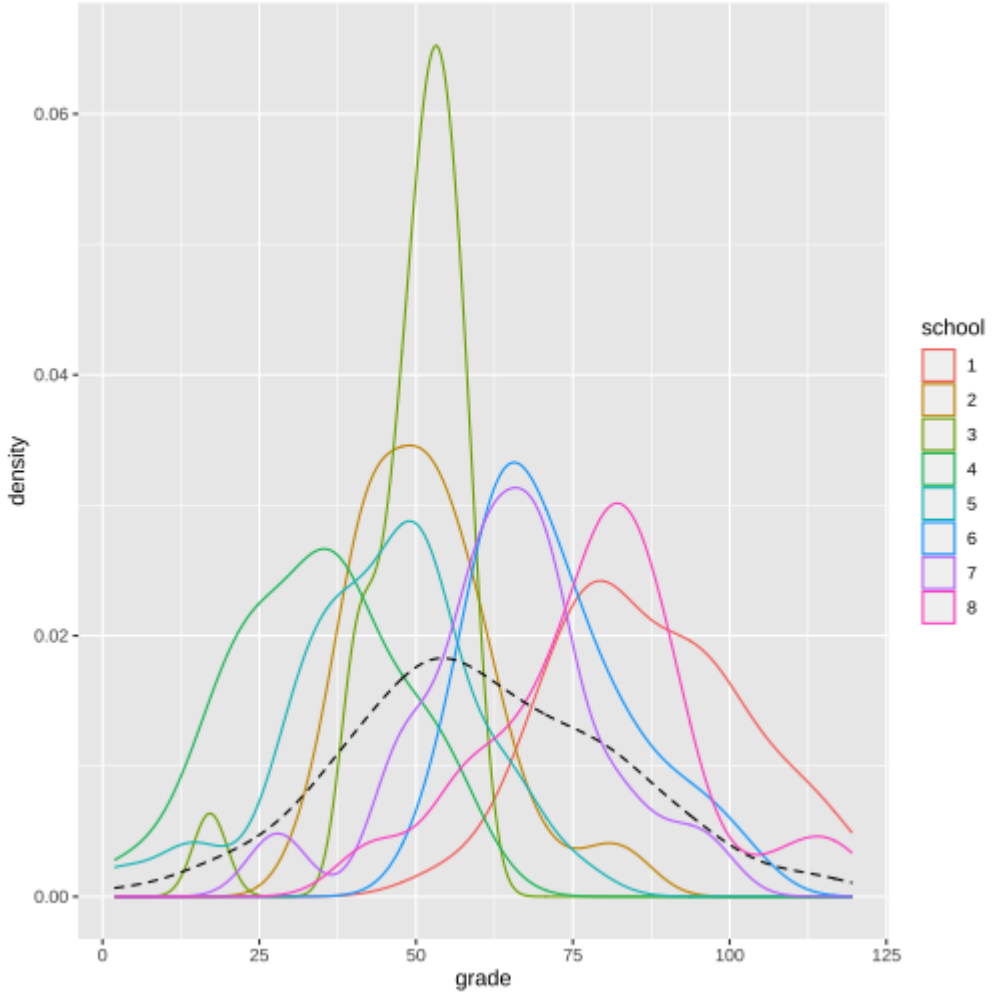

count
grade
0
5
10
0
25
50
75
100

Sharing Data

Hierarchical Modeling is a compromise to typical linear modeling techniques which default to either considering all schools to be the same (complete pooling), or all schools to be distinct (no pooling).

- The core concept is to consider how each school's data should contribute to the overall prediction of the new school
- In the scenario in which we effectively ignore school distinctiveness, we obtain one large distribution of student grades
- In the scenario in which we consider each school to be entirely distinct, we obtain eight separate distributions of student grades
- Hierarchical modeling says we should intelligently weight each school in the prediction by considering how much information it provides through a "hierarchical" distribution.

Partially Pooled Data



Hierarchical Modeling

$$\mu_i \sim N(\mu_{overall}, \sigma_{school})$$

$$y_{i,j} \sim N(\mu_i, \sigma_{student})$$

```
data {
  int N;
  int num_schools;
  vector[N] y;
  int school_id[N];
}

parameters {
  real overall_mu;
  real<lower=0> sigma_school;

  vector[num_schools] school_mu;
  real<lower=0> sigma_student;
}

model {
  for(i in 1:num_schools) {
    school_mu[i] ~ normal(overall_mu, sigma_school);
```

Hierarchical Modeling

Hierarchical modeling is a big idea which can take time to sink in conceptually. It also goes by many different names:

- Random Effects or Mixed Effects Modeling in classical statistical literature
- Hierarchical Modeling or Multilevel Modeling in Bayesian statistical literature

If you are interested in learning more, I encourage you to read Gelman & Hill's Data Analysis Using Regression and Multilevel/Hierarchical Models.

Bayesian Inference

Underlying all of the Stan code we have implemented has been an ultimately Bayesian modeling technique. While it is not necessary to be explicitly Bayesian for Stan to work, it can be helpful to understand how a "Bayesian analysis" would be performed.

- For some parameters, we did not specify a probability distribution in the `model` block. Typically, we only defined distributions for our data using our defined parameters.
- Defining a probability distribution on a parameter other than the data is called a **Prior distribution** and is used to quantify the information we have about the parameter before performing analysis.
- Priors have both philosophical and practical components. In data science, it is common to think about Priors as a way to obtain regularization to avoid overfitting. Both Ridge Regression and LASSO are use penalty terms which can be interpreted as specific prior distributions in a linear regression context.

Questions, Comments, Discussion?