Computational Bayesian data analysis

Stan language

Bruno Nicenboim / Shravan Vasishth

2020-03-11

Stan is (mostly) written in C++ and can be accessed through several interfaces:

- R
- Python
- Matlab
- Stata
- bash
- etc

A Stan program

- usually saved as a `.stan` file
- accessed through R (or other interfaces)
- organized into a sequence of optional and obligatory blocks, which must be written in order.

A Stan program

```
functions {
// This is an optional block used for functions that can be used in other blocks.
}
data {
// Obligatory block that specifies the required data for the model.
}
transformed data {
// Optional block if we want to manipulate the data.
}
parameters {
// Obligatory block that specifies the model's parameters.
}
transformed parameters {
// Optional block if we want to manipulate the parameters (re-parametrize the model).
}
model {
// Obligatory block that specifies the model's likelihood and priors.
}
generated quantities {
// Optional block if we want to manipulate the output of our model.
}
```

A Stan program

- every variable used needs to be declared first with its type (real, integer, vector, matrix, etc.).
- there must be a semi-colon (;) at the end of each line.

Some examples of data type below (but check the Stan reference manual for more):

- Variable `mu` contains a real number, either positive or negative:

```
real mu;
```

- Variable `X` contains a real number that is bounded between two numbers (or by only one). Suppose `X` is some type of measurement that can only be between 0 and 1000. (We can add lower and/or upper to any type.)

```
real<lower = 0, upper = 1000>  X;
```

- Variable `N` contains integers, such as the number of observations (which should be > 0):

```
int<lower = 0>  N;
```

- For vectors, row_vectors and matrices, we must define the number of elements (in each margin) that they will contain. This number can be defined earlier (such as `N`, the number of observations, in our example). (In Stan, the values inside a vectors and matrices are always *real*.)

```
vector<lower = 0> [N] Y;
```

```
row_vector<lower = 0> [10] Y;
```

```
matrix<upper = 0> [3, J] Rho;
```

- Any type can be converted into an array of as many dimensions as we want, even vectors and matrices. It's worthwhile to take a look at Array Data Types section of the Stan reference manual.

```
real mu[2];  // one dimension, two places

int<lower = 0>  N[x,y,z]; // 3 dimensions, with x, y, z places.

vector<lower = 0> [N] Y[2]; // array of one dimensions that
contain 2 vectors of N places
```

Example: Cloze probability with Stan (Binomial likelihood)

We want to derive the posterior distribution of the Cloze probability of *"umbrella"*, θ:

- Data: a word (e.g., *"umbrella"*) was answered 80 out of 100 times,
- Likelihood: a binomial distribution
- Prior for θ: $Beta(a = 4, b = 4)$

Example: Cloze probability with Stan (Binomial likelihood)

```
data {
  int<lower = 1> N;  // Total number of answers
  int<lower = 0, upper = N> k;  // Number of times "umbrella" was answered
}
parameters {
  // theta is a probability, it has to be constrained between 0 and 1
  real<lower = 0, upper = 1> theta;
}
model {
  // Prior on theta:
  target += beta_lpdf(theta | 4, 4);
  // Likelihood:
  target += binomial_lpmf(k | N, theta);
}
```

(*) every statement with `target +=` increments the unnormalized *log* posterior probability.

- Save the previous model as `stan_models/binomial.stan` (don't run it in R).
- Use the following code to call the model from R:

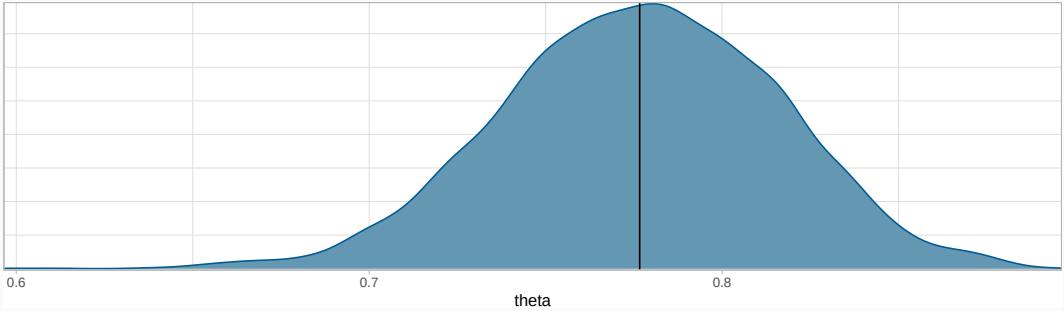
```
library(rstan)
options(mc.cores = parallel::detectCores())
lst_cloze_data <- list(k = 80, N = 100)
# Fit the model with the default values of number
# of chains and iterations (chains = 4,iter = 2000)
fit_cloze <- stan(
  file = "stan_models/binomial_cloze.stan",
  data = lst_cloze_data
)
```

```
fit_cloze
```

```
## Inference for Stan model: binomial_cloze.
## 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%   98%
## theta  0.78    0.00 0.04  0.69  0.75  0.78  0.8  0.85
## lp__  -5.05    0.02 0.73 -7.15 -5.23 -4.77 -4.6 -4.54
##       n_eff Rhat
## theta  1412    1
## lp__   1922    1
##
## Samples were drawn using NUTS(diag_e) at Wed Mar 11 15:18:52 2020.
## 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).
```

`bayesplot` (Gabry and Mahr 2019) is a wrapper around `ggplot2` (Wickham et al. 2019) and has several convenient functions to plot the samples (see their vignette).

```
library(bayesplot)
# We need to convert the fit to plot it with bayesplot functions
df_fit_cloze <- as.data.frame(fit_cloze)
# Bayes plot functions start with mcmc_
mcmc_dens(df_fit_cloze, pars = "theta") +
  geom_vline(xintercept = mean(df_fit_cloze$theta))
```



A more complex Stan model

$$
\begin{aligned}
rt_n &\sim LogNormal(\mu, \sigma) \\
\mu &\sim Normal(6, 1.5) \\
\sigma &\sim Normal_+(0, 1)
\end{aligned}
\tag{1}
$$

A more complex Stan model

```
data {
  int<lower=1> N_obs;
  vector[N_obs] rt;
}
parameters {
  real<lower=0> sigma;
  real mu;
}
model {
  target += normal_lpdf(mu | 6, 1.5);
  target += normal_lpdf(sigma | 0, 1)  -
    normal_lccdf(0 | 0, 1);
  target += lognormal_lpdf(rt | mu, sigma);
}
generated quantities {
  vector[N_obs] rt_sim;
  // posterior predictive
  for(i in 1:N_obs) {
    rt_sim[i] = lognormal_rng(mu, sigma);
  }
```

```r
library(dplyr)
library(readr)
df_noreading_data <-
  read_csv("./data/button_press.csv")
lst_noreading <- list(
  N_obs = nrow(df_noreading_data),
  rt = df_noreading_data$rt
)

fit_lognormal_reading <- stan("stan_models/lognormal.stan",
  data = lst_noreading
)
```

```r
print(fit_lognormal_reading, pars = c("sigma", "mu"))
```

```
## Inference for Stan model: lognormal.
## 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%  98% n_eff
## sigma 0.13       0 0.01 0.13 0.13 0.13 0.14 0.15  3081
## mu    5.12       0 0.01 5.10 5.11 5.12 5.12 5.13  4482
##       Rhat
## sigma    1
## mu       1
##
## Samples were drawn using NUTS(diag_e) at Wed Mar 11 15:46:49 2020.
## 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).
```

Extracting Stan code from brms

```
library(brms)
fit_press_ln <- brm(rt ~ 1,
  data = df_noreading_data,
  family = lognormal(),
  prior = c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = sigma)
  )
)
```

Extracting code when the model was run:

```
stancode(fit_press_ln)
```

```
## // generated with brms 2.12.0
##  functions {
##  }
##  data {
##    int<lower=1> N;  // number of observations
##    vector[N] Y;  // response variable
##    int prior_only;  // should the likelihood be ignored?
##  }
##  transformed data {
##  }
##  parameters {
##    real Intercept;  // temporary intercept for centered predictors
##    real<lower=0> sigma;  // residual SD
##  }
##  transformed parameters {
##  }
##  model {

•••
```

```
## }
##  model {
##    // initialize linear predictor term
##    vector[N] mu = Intercept + rep_vector(0, N);
##    // priors including all constants
##    target += normal_lpdf(Intercept | 6, 1.5);
##    target += normal_lpdf(sigma | 0, 1)
##      - 1 * normal_lccdf(0 | 0, 1);
##    // likelihood including all constants
##    if (!prior_only) {
##      target += lognormal_lpdf(Y | mu, sigma);
##    }
##  }
##  generated quantities {
##    // actual population-level intercept
##    real b_Intercept = Intercept;
##  }
```

Extracting code before the model was run:

```
make_stancode(rt ~ 1,
  data = df_noreading_data,
  family = lognormal(),
  prior = c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = sigma)
  )
)
```

Extracting the data when the model was run:

```
ls_stan <- standata(fit_press_ln)
ls_stan %>% str()
```

```
## List of 5
##  $ N         : int 361
##  $ Y         : num [1:361(1d)] 141 138 128 132 126 134 163 149 133 110 ...
##  $ K         : int 1
##  $ X         : num [1:361, 1] 1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:361] "1" "2" "3" "4" ...
##   .. ..$ : chr "Intercept"
##   ..- attr(*, "assign")= int 0
##  $ prior_only: int 0
##  - attr(*, "class")= chr "standata"
```

Extracting the data before the model was run:

```
ls_stan <- make_standata(rt ~ 1,
  data = df_noreading_data,
  family = lognormal(),
  prior = c(
    prior(normal(6, 1.5), class = Intercept),
    prior(normal(0, 1), class = sigma)
  )
)
```

References

Gabry, Jonah, and Tristan Mahr. 2019. *Bayesplot: Plotting for Bayesian Models*. https://CRAN.R-project.org/package=bayesplot.

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, Kara Woo, and Hiroaki Yutani. 2019. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. https://CRAN.R-project.org/package=ggplot2.