Practical 4 - Advanced hierarchical GLMs

Andrew Parnell

Introduction

In practical 4 we are going to:

- Fit a more complex hierarchical linear model
- Go through a full Bayesian analysis of some models on a new data set
- Perform some model comparison using WAIC with Stan
- Create some plots of the posterior predicted values with uncertainty

We are going to focus on the `prostate` data set in the `data` folder. We're going to use Stan throughout, though if you've stuck to JAGS so far you should try and translate the code into JAGS. Everything in this tutorial should still run for you.

The prostate data set

The `prostate` data set can be loaded in and explored with:

```
prostate = read.csv('https://raw.githubusercontent.com/andrewcparnell/bhm_course/master/data/prostate.cs
head(prostate)
```

```
##       lcavol  lweight age      lbph svi       lcp gleason pgg45       lpsa
## 1 -0.5798185 2.769459  50 -1.386294   0 -1.386294       6     0 -0.4307829
## 2 -0.9942523 3.319626  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 3 -0.5108256 2.691243  74 -1.386294   0 -1.386294       7    20 -0.1625189
## 4 -1.2039728 3.282789  58 -1.386294   0 -1.386294       6     0 -0.1625189
## 5  0.7514161 3.432373  62 -1.386294   0 -1.386294       6     0  0.3715636
## 6 -1.0498221 3.228826  50 -1.386294   0 -1.386294       6     0  0.7654678
##   train
## 1  TRUE
## 2  TRUE
## 3  TRUE
## 4  TRUE
## 5  TRUE
## 6  TRUE
```

The data contain information on 97 men with prostate cancer. The response variable here is `lpsa` - the log of the *prostate specific antigen*. We want to see whether this variable can be predicted from the others. The other key variables we will use are:

- `lcavol`, the log cancer volume
- `lweight`, the log of the weight of the patient
- `gleason`, the Gleason grade of the cancer (from 6 to 9). A higher value is more serious

---

Exercise 1

1. Create suitable plots (scatter plots, boxplots, etc) to get a sense of the relationships between these variables. Which variable(s) seems to be the most predictive?

Creating a first model

A first linear regression model I would suggest for these data is:

$$\log(psa) \sim N(\alpha + \beta_1 \text{lcavol}_i + \beta_2 \text{lweight}_i + \beta_3 \text{gleason}_i, \sigma^2)$$

Let's fit it in Stan. Here's the code with vague priors and no mean correction:

```
stan_code = '
data {
  int<lower=0> N;
  vector[N] lpsa;
  vector[N] lcavol;
  vector[N] lweight;
  vector[N] gleason;
}
parameters {
  real alpha;
  real beta_1;
  real beta_2;
  real beta_3;
  real<lower=0> sigma;
}
model {
  lpsa ~ normal(alpha + beta_1 * lcavol + beta_2 * lweight + beta_3 * gleason, sigma);
  alpha ~ normal(0, 10);
  beta_1 ~ normal(0, 10);
  beta_2 ~ normal(0, 10);
  beta_3 ~ normal(0, 10);
  sigma ~ cauchy(0, 10);
}
'
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
stan_run = stan(data = list(N = nrow(prostate),
                            lpsa = prostate$lpsa,
                            lcavol = prostate$lcavol,
                            lweight = prostate$lweight,
                            gleason = prostate$gleason),
                model_code = stan_code)
```

Exercise 2

1. This model isn't mean corrected. Mean correct it properly either in the `stan_code` object or in the data list provided to `stan`.
2. These are pretty poor priors. Try simulating data from these priors and this likelihood (once you've mean corrected it). Try and come up with your own better priors (pretend you are the data expert here)
3. Once you have come up with a model that you like, re-fit the model and interpret the output. Which variables are important?

Extracting the WAIC from a Stan model run

If you have a recent version of Stan (hopefully you installed it in the last few days), then you can create the WAIC value (a superior version of the DIC) but it's a bit fiddly at resent. You first need to re-run your model, adding in a `generated quantities` block which outputs the log likelihood.

```
stan_code = '
data {
  int<lower=0> N;
  vector[N] lpsa;
  vector[N] lcavol;
  vector[N] lweight;
  vector[N] gleason;
}
parameters {
  real alpha;
  real beta_1;
  real beta_2;
  real beta_3;
  real<lower=0> sigma;
}
model {
  lpsa ~ normal(alpha + beta_1 * lcavol + beta_2 * lweight + beta_3 * gleason, sigma);
  alpha ~ normal(0, 10);
  beta_1 ~ normal(0, 10);
  beta_2 ~ normal(0, 10);
  beta_3 ~ normal(0, 10);
  sigma ~ cauchy(0, 10);
}
generated quantities {
  vector[N] log_lik;
  for(i in 1:N) {
    log_lik[i] =  normal_log(lpsa[i], alpha + beta_1 * lcavol[i] + beta_2 * lweight[i] + beta_3 * gleas
  }
}
'
library(rstan)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
stan_run = stan(data = list(N = nrow(prostate),
                            lpsa = prostate$lpsa,
                            lcavol = prostate$lcavol,
                            lweight = prostate$lweight,
                            gleason = prostate$gleason),
                model_code = stan_code)
```

We can now use the `loo` package to get the WAIC (with uncertainty!)

```
library(loo)
log_lik = extract_log_lik(stan_run, parameter_name = 'log_lik')
waic(log_lik)
```

```
##
## Computed from 4000 by 97 log-likelihood matrix
##
```

```
##           Estimate   SE
## elpd_waic   -111.7  6.8
## p_waic         4.8  0.8
## waic         223.4 13.6
```

The `loo` package should be installed as part of `rstan` but if you find it's not there you can always run `install.packages('loo')`. The `extract_log_lik` line here gets the log likelihood scores for each iteration from the `stan_run` object. The `waic` command computers the WAIC and the effective number of parameters (`p_waic`).

This package is currently quite new and still seems to spit out some odd warnings but generally the results seem fast and useful, and it gives you uncertainty on the value! Besides the WAIC you can also get an estimate of the leave-one-out (LOO) cross validation error:

```
loo(log_lik)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
##
## Computed from 4000 by 97 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -111.7  6.8
## p_loo         4.9  0.8
## looic       223.5 13.6
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

These two will generally be pretty similar as the WAIC is designed to match the LOO except in the case of very influential observations. My current view is to stick with the WAIC as most people will be familiar with this type of measure from using AIC, BIC, DIC, etc.

Extending the model

One of way of extending the model is to have the intercept varying according to Gleason grade. Here is a model:

```
stan_code = '
data {
  int<lower=0> N;
  int<lower=0> N_gleason;
  vector[N] lpsa;
  vector[N] lcavol;
  vector[N] lweight;
  int gleason_ind[N];
}
parameters {
  vector[N_gleason] alpha;
  real beta_1;
  real beta_2;
  real<lower=0> sigma;
```

```
  real<lower=0> sigma_alpha;
}
model {
  for (i in 1:N)
    lpsa[i] ~ normal(alpha[gleason_ind[i]] + beta_1 * lcavol[i] + beta_2 * lweight[i], sigma);
  for (j in 1:N_gleason) {
    alpha[j] ~ normal(0, sigma_alpha);
  }
  beta_1 ~ normal(0, 10);
  beta_2 ~ normal(0, 10);
  sigma ~ cauchy(0, 10);
  sigma_alpha ~ cauchy(0, 10);
}
'
stan_run = stan(data = list(N = nrow(prostate),
                            N_gleason = 4,
                            lpsa = prostate$lpsa,
                            lcavol = prostate$lcavol,
                            lweight = prostate$lweight,
                            gleason_ind = prostate$gleason-5),
                model_code = stan_code)
```

```
## Warning: There were 14 divergent transitions after warmup. Increasing adapt_delta above 0.8 may help
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

Exercise 3

1. Try to calculate the WAIC value for this model (ask if you get stuck with the `generated quantities` value)
2. Improve the model with the priors you developed in the previous exercise
3. Think of two or three different ways to expand this model.
4. Try to fit each of the models and keep a note of the WAIC.
5. Find your 'best' model and interpret the parameters

Creating posterior predictive distributions in Stan

We can use another `generated quantities` block to include a sampling statement to generate posterior predictive distributions. We can do this with:

```
stan_code = '
data {
  int<lower=0> N;
  int<lower=0> N_gleason;
  vector[N] lpsa;
  vector[N] lcavol;
  vector[N] lweight;
  int gleason_ind[N];
}
parameters {
  vector[N_gleason] alpha;
```

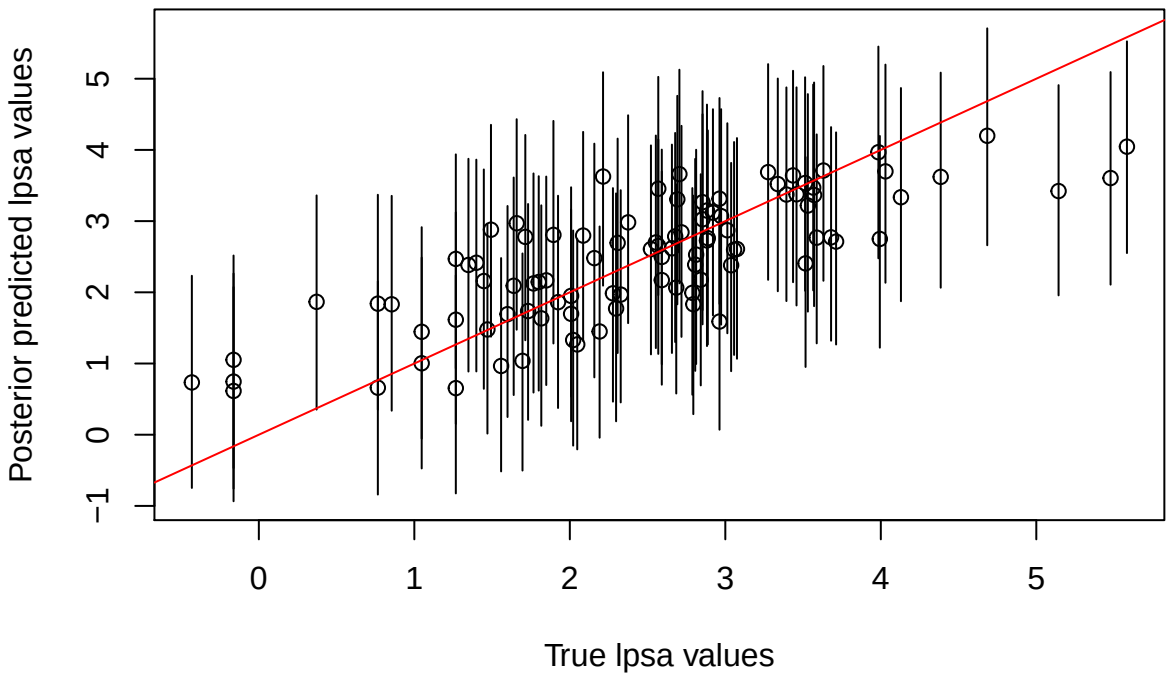
```
  real beta_1;
  real beta_2;
  real<lower=0> sigma;
  real<lower=0> sigma_alpha;
}
model {
  for (i in 1:N)
    lpsa[i] ~ normal(alpha[gleason_ind[i]] + beta_1 * lcavol[i] + beta_2 * lweight[i], sigma);
  for (j in 1:N_gleason) {
    alpha[j] ~ normal(0, sigma_alpha);
  }
  beta_1 ~ normal(0, 10);
  beta_2 ~ normal(0, 10);
  sigma ~ cauchy(0, 10);
  sigma_alpha ~ cauchy(0, 10);
}
generated quantities {
  vector[N] lpsa_sim;
  for (i in 1:N) {
    lpsa_sim[i] = normal_rng(alpha[gleason_ind[i]] + beta_1 * lcavol[i] + beta_2 * lweight[i], sigma);
  }
}
'
stan_run = stan(data = list(N = nrow(prostate),
                            N_gleason = 4,
                            lpsa = prostate$lpsa,
                            lcavol = prostate$lcavol,
                            lweight = prostate$lweight,
                            gleason_ind = prostate$gleason-5),
                model_code = stan_code)
```

```
## Warning: There were 138 divergent transitions after warmup. Increasing adapt_delta above 0.8 may hel
## http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
## Warning: Examine the pairs() plot to diagnose sampling problems
```

We can now plot the posterior predictive. Here I'm going to include uncertainty on the values:

```
lpsa_sim = extract(stan_run, pars = 'lpsa_sim')$lpsa_sim
lpsa_sim_summary = apply(lpsa_sim, 2, 'quantile', probs = c(0.025, 0.5, 0.975))
plot(prostate$lpsa, lpsa_sim_summary[2,],
     xlab = 'True lpsa values',
     ylab = 'Posterior predicted lpsa values',
     ylim = range(lpsa_sim_summary))
for(i in 1:ncol(lpsa_sim_summary)) {
  lines(c(prostate$lpsa[i],prostate$lpsa[i]),
        c(lpsa_sim_summary[1,i],lpsa_sim_summary[3,i]))
}
abline(a = 0, b = 1, col = 'red')
```



A pretty good fitting model!

Exercise 3

1. Create the posterior predictive for the best model you found above
2. Get the plot with uncertainties like the above. Did it fit the data well?
3. Interpret the parameters and performance of your model.

Congratulations, you have now completed the majority of a Bayesian analysis on these data!

Harder exercises

- Try incorporating some of the other variables in the data set that we have left out. Do they improve the model fit?
- Pick two or three of the best models that you tried, and run a full 5-fold cross validation. Does the 5-fold CV pick the same best model as the `waic` and `loo` function values?