STA2201H Methods of Applied Statistics II

Monica Alexander

Week 5: Bayesian regression and Stan

Annoucements

- Assignment 2 coming soon (Bayesian inference)

Where are we at

- Bayesian inference revolves around inference based on the posterior
- Posterior usually hard to write down in closed form
- But as long as we can get a set of samples from posterior, we can do inference
- For most problems, we can construct an MCMC algorithm that can be used to generate samples from posterior distributions

Running MCMC in R

Running MCMC in R

Good news:

- lots of standard software to run MCMC so that we (usually) don't have to code it ourselves
- easy to implement (m)any Bayesian models as long as you can write down the model specification
- The user does not need to worry about the MCMC algorithm
- you just have to check MCMC output for convergence/mixing (and understand why it might not be working)

BUGS/JAGS

- BUGS: Bayesian Inference Using Gibbs Sampling (includes WinBUGS, OpenBUGS)
- JAGS: Just Another Gibbs Sampler (BUGS but not platform dependent).
- Very common from late 2000s, still main tool used in many applied fields (particularly WinBUGS in epidemiological research where spatial modeling is common)
- Run models by writing a ‘model’ (as text) which specifies your likelihood and priors.

Stan

- Increasingly common to use on a wide range of estimation problems
- Estimates using a version of HMC (No-U-Turn-Sampler)
- Stan consists of a language for defining probabilistic models, a compiler to transform Stan code into C++, and math and algorithm libraries that help run models
- We will be running Stan through R using RStan; can also run in CMD or Python

Running Stan: overview

- Write a Stan program (in a .stan file), specifying data, parameters, model, and potentially other things
- Input data, run model
- Output: an approximate sample from the posterior distribution, summaries of the run which can help us diagnose problems.

Writing a model in Stan

Like C++, in Stan:

- Variables need to be declared and strongly typed
- Each statement must end with a semi-colon

A Stan program is divided into coding blocks. Essential:

- data
- parameters
- model

Optional:

- transformed parameters
- generated quantities
- ...

A Stan file

save as `whatever.stan`

```
data {
 Declare the data that will be given as an input.
}

parameters {
 Declare the parameters we want to sample.
}

model {
 Compute the log joint distribution.
}
```

Back to Kid's test scores

Recall model set up

$$
\begin{aligned}
y_i|\mu, \sigma &\sim N\left(\mu, \sigma^2\right) \\
\mu &\sim N\left(\mu_0, \sigma^2_{\mu 0}\right) \\
1/\sigma^2 &\sim \text{Gamma}\left(\nu_0/2, \nu_0/2 \cdot \sigma^2_{y0}\right)
\end{aligned}
$$

```
data {
  int<lower=0> N;            // number of kids
  vector[N] y;               // scores
  real mu0;                  // mean of mu prior
  real<lower=0> sigma0;     //  variance of mu prior
  real<lower=0> nu0;         // shape of precision prior
}
parameters {
  real mu;
  real<lower=0> tau;
}
transformed parameters {
   real<lower=0> sigma = sqrt(1/tau);
}
model {
  //priors
  mu ~ normal(mu0, sigma0);
  tau ~ gamma(nu0/2, nu0/2*sigma0^2);

  //target += normal_lpdf(y | mu, sigma);
  //equivalent:
  y ~ normal(mu, sigma);
}
```

Kid's example

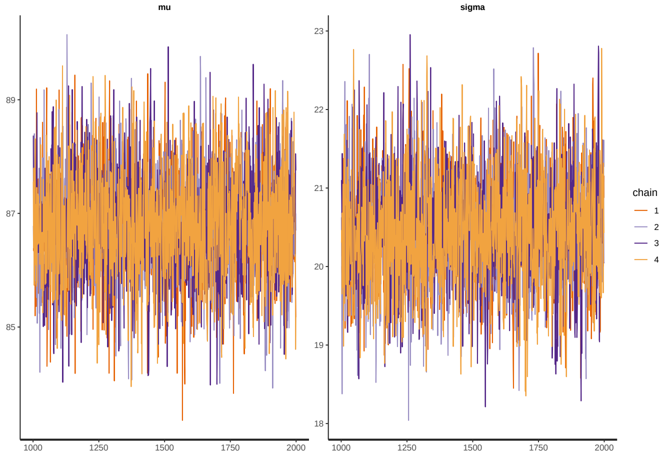
```r
library(rstan)
data <- list(y = y, N = length(y),
             mu0 = mu0,
             sigma0 = sigma0,
             nu0 = 1)
fit <- stan(file = "kids.stan",
            data = data)
```

```r
fit
```

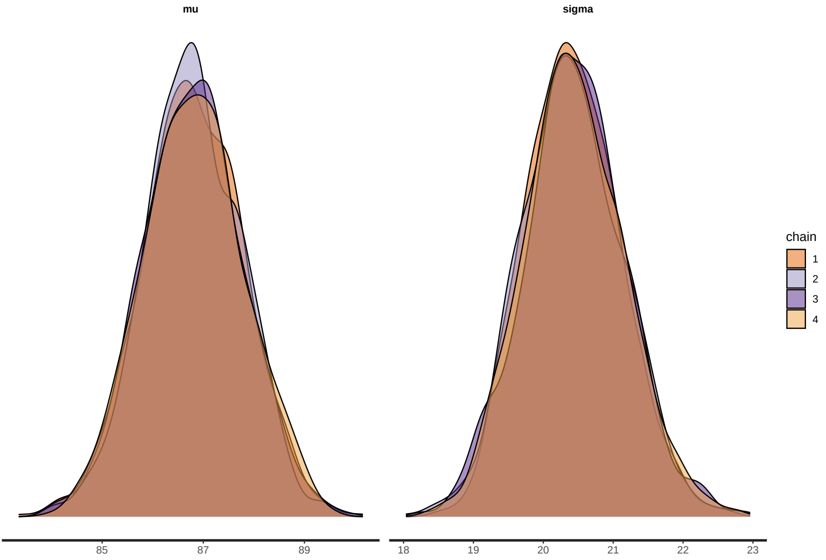
```
## Inference for Stan model: kids.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##           mean se_mean   sd     2.5%      25%      50%      75%    97.5% n_eff
## mu       86.80    0.02 0.97    84.92    86.15    86.80    87.46    88.70  3466
## tau       0.00    0.00 0.00     0.00     0.00     0.00     0.00     0.00  3125
## sigma    20.43    0.01 0.71    19.11    19.96    20.41    20.90    21.88  3111
## lp__  -1529.99    0.02 0.97 -1532.68 -1530.38 -1529.71 -1529.29 -1529.01  1791
##       Rhat
## mu       1
## tau      1
## sigma    1
## lp__     1
##
## Samples were drawn using NUTS(diag_e) at Wed Feb  9 08:22:06 2022.
## 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).
```

Some graphical checks

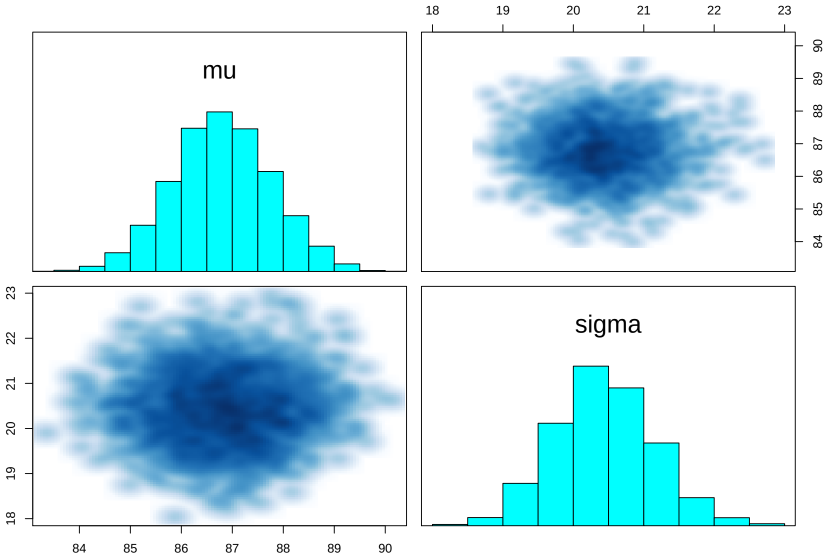
```
pars = c("mu", "sigma")
traceplot(fit, pars = pars)
```



```r
stan_dens(fit, separate_chains = TRUE, pars = pars)
```



```r
pairs(fit, pars = pars)
```



Bayesian inference for regression models

Kid's scores

A reasonable model to consider is

$$y_i|\mu_i, \sigma \sim N\left(\mu, \sigma^2\right)$$
$$\mu_i = \alpha + \beta x_i$$

where X_i is mother's IQ score. This is a simple linear regression model. We are primarily interested in obtaining estimates for the regression coefficients, α and β.

- OLS or MLE finds estimates of the parameters that best fit the data
- Bayesian inference incorporates prior information about the parameters
- In Bayesian inference, the estimates are a compromise between the prior info and the data

Bayesian inference for linear regression

What does Bayesian inference get us that MLE doesn't?

- **Inclusion of prior information**:
  - we usually know something
  - makes inferences more stable, as the estimates are typically somewhere between the prior and what would be obtained from the data alone
- **Prorogation of uncertainty**:
  - least squares gives us a point estimate
  - in Bayesian inference, we can summarize uncertainty using simulations from the posterior distribution

Kid's scores

$$y_i|\mu_i, \sigma \sim N\left(\mu, \sigma^2\right)$$
$$\mu_i = \alpha + \beta x_i$$

We need to put priors on σ (as before) but also α and β. Let's put

$$\alpha \sim N(0, 100^2)$$
$$\beta \sim N(0, 10^2)$$
$$\sigma \sim \text{Half-Normal}(0, 10)$$

In Stan

```
data {
  int<lower=0> N;           // number of kids
  int<lower=0> K;           // number of covariates
  vector[N] y;              // scores
  matrix[N, K] X;            // design matrix
}
parameters {
  real alpha;
  vector[K] beta;
  real<lower=0> sigma;
}
transformed parameters {
}
model {
  //priors
  alpha ~ normal(0, 100);
  beta ~ normal(0, 10);
  sigma ~ normal(0,10);

  //likelihood
  y ~ normal(alpha + X*beta, sigma);
}
```

Fits comparison

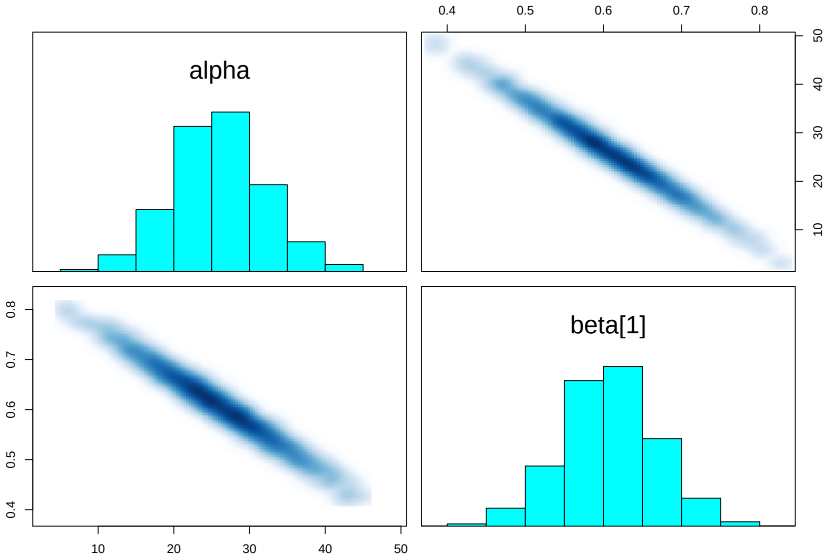
```
summary(fit)$summary[c("alpha", "beta[1]"),]
```

```
##                mean     se_mean         sd       2.5%        25%        50%
## alpha   25.9663164 0.170400111 6.10878490 13.9766899 22.0636107 25.9135726
## beta[1]  0.6084974 0.001683857 0.06032798  0.4869285  0.5698391  0.6090426
##                 75%      97.5%    n_eff     Rhat
## alpha   29.8466529 38.1250541 1285.198 1.002340
## beta[1]  0.6476261  0.7264963 1283.592 1.002338
```

```
summary(lm(kid_score~mom_iq, data = kidiq))
```

```
##
## Call:
## lm(formula = kid_score ~ mom_iq, data = kidiq)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -56.753 -12.074   2.217  11.710  47.691
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 25.79978    5.91741    4.36 1.63e-05 ***
## mom_iq       0.60997    0.05852   10.42  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 18.27 on 432 degrees of freedom
## Multiple R-squared:  0.201,  Adjusted R-squared:  0.1991
## F-statistic: 108.6 on 1 and 432 DF,  p-value: < 2.2e-16
```

```r
pairs(fit, pars = c("alpha", "beta[1]"))
```



What do we get

```r
post_samples <- extract(fit)
length(post_samples)
```

```
## [1] 4
```

```r
names(post_samples)
```

```
## [1] "alpha" "beta"  "sigma" "lp__"
```

What do we get

```r
dim(post_samples[["alpha"]])
```

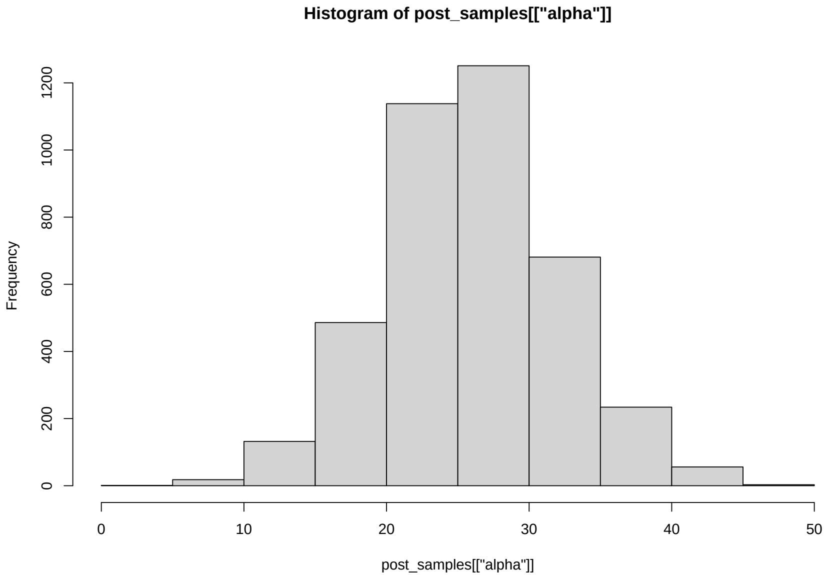
```
## [1] 4000
```

```r
post_samples[["alpha"]][1:5]
```

```
## [1] 21.42580 22.90771 19.09068 27.44277 43.81041
```

What do we get

```
hist(post_samples[["alpha"]])
```



Tidy version

```
library(tidybayes)
fit %>%
  gather_draws(alpha)
```

```
## # A tibble: 4,000 x 5
## # Groups:   .variable [1]
##    .chain .iteration .draw .variable .value
##     <int>      <int> <int> <chr>      <dbl>
##  1      1          1     1 alpha       31.9
##  2      1          2     2 alpha       32.4
##  3      1          3     3 alpha       34.3
##  4      1          4     4 alpha       31.1
##  5      1          5     5 alpha       31.6
##  6      1          6     6 alpha       30.1
##  7      1          7     7 alpha       27.7
##  8      1          8     8 alpha       27.9
##  9      1          9     9 alpha       24.7
## 10      1         10    10 alpha       24.2
## # ... with 3,990 more rows
```

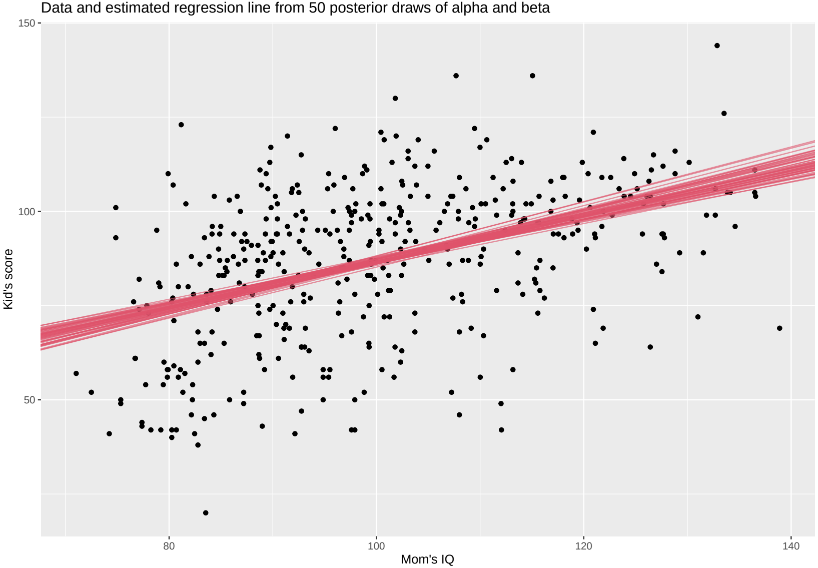
What can we do

- The data and model are combined to form a posterior distribution, which we typically summarize by a set of simulations of the parameters in the model
- We can propagate uncertainty in this distribution, that is, we can get simulation-based prediction for unobserved or future outcomes that accounts for uncertainty in the model parameters

With simulations, we can

- Visualize uncertainty in the regression line
- Get uncertainty for functions of parameters
- Make predictions based on new data points

Uncertainty in the regression coefficients and implied uncertainty in the regression line



Uncertainty about a function of parameters

For example, posterior samples for the ratio of α and β

Making predictions

Consider making a prediction of kid's score with a new observation of mother's IQ, x^{new}. We have

- the point prediction $\hat{\alpha} + \hat{\beta} x^{\text{new}}$
- the linear predictor with uncertainty $\alpha + \beta x^{\text{new}}$
  - propagates uncertainty in regression coefficients
  - represents the distribution of uncertainty about the expected value of y for new data points x^{new}
- the predictive distribution for a new observation $\alpha + \beta x^{\text{new}} +$ error
  - represents uncertainty about a new observation y with predictor x^{new}

Predictions

Consider a new mother with an IQ of 110.

Point prediction: use medians of posterior samples for $\hat{\alpha}$ and $b\hat{e}ta$

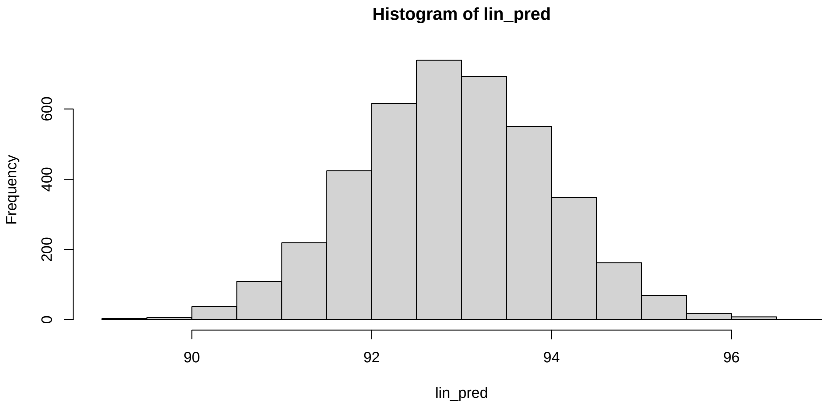
```
x_new <- 110
alpha_hat <- median(post_samples[["alpha"]])
beta_hat <- median(post_samples[["beta"]])

alpha_hat + beta_hat*x_new
```

```
## [1] 92.90826
```

Predictions

Linear predictor with uncertainty:

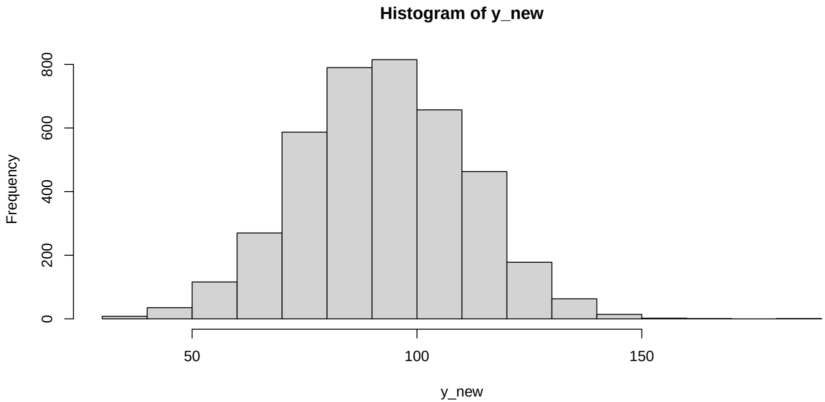
```
alpha <- post_samples[["alpha"]]
beta <- post_samples[["beta"]][,1]

lin_pred <- alpha + beta*x_new
hist(lin_pred)
```



Predictions

Predictive distribution for new observation:

```
sigma <- post_samples[["sigma"]]
y_new <- rnorm(n = length(sigma),mean = lin_pred, sd = sigma)
hist(y_new)
```



Can also do this within Stan

Can get posterior predictive distribution samples using the generated quantities block:

```
generated quantities{
  real y_new[1];
  y_new = normal_rng(alpha + x_new*beta, sigma);
}
```

Posterior predictive distribution

$$p(\tilde{y}|y) = \int_{\Theta} p(\tilde{y}|\theta, y)p(\theta|y)d\theta$$

- After we have seen the data and obtained the posterior distributions of the parameters, we can now use the posterior distributions to generate new data from the model.
- Given the posterior distributions of the parameters of the model, the posterior predictive distribution gives us some indication of what new data might look like, given the data and model.
- We can avoid performing the integration explicitly by generating samples from the posterior predictive distribution.

Posterior predictive distributions also important for model checking. More next week.

Posterior predictive distribution

Posterior predictive distribution for new $\tilde{y}$

$$p(\tilde{y}|y) = \int_{\Theta} p(\tilde{y}|\theta, y)p(\theta|y)d\theta$$

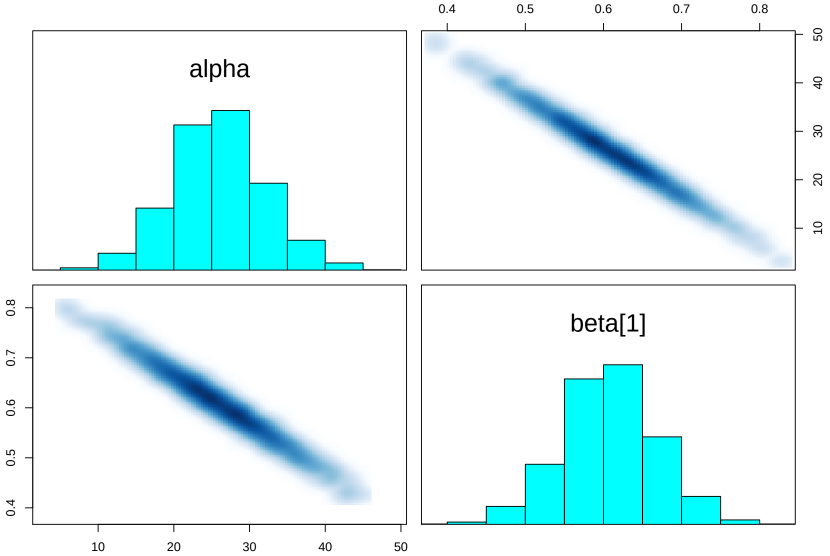
To obtain samples from this distribution, we need to

- Get posterior samples of our parameters $\theta^{(s)}$ (MCMC output!)
- For each posterior sample, we obtain one replicated dataset $\tilde{y}^{(s)}$ by sampling from the likelihood $p(\tilde{y}|\theta^{(s)})$. Can do this in R or within Stan.

Centering predictors to improve posterior geometries

Remember this

```
pairs(fit, pars = c("alpha", "beta"))
```



Centering

```
data <- list(y = y,
             N = length(y),
             K = 1,
             X = as.matrix(kidiq$mom_iq - mean(kidiq$mom_iq)))
fit2 <- stan(file = "kids3.stan",
            data = data)
```

Summary of fit

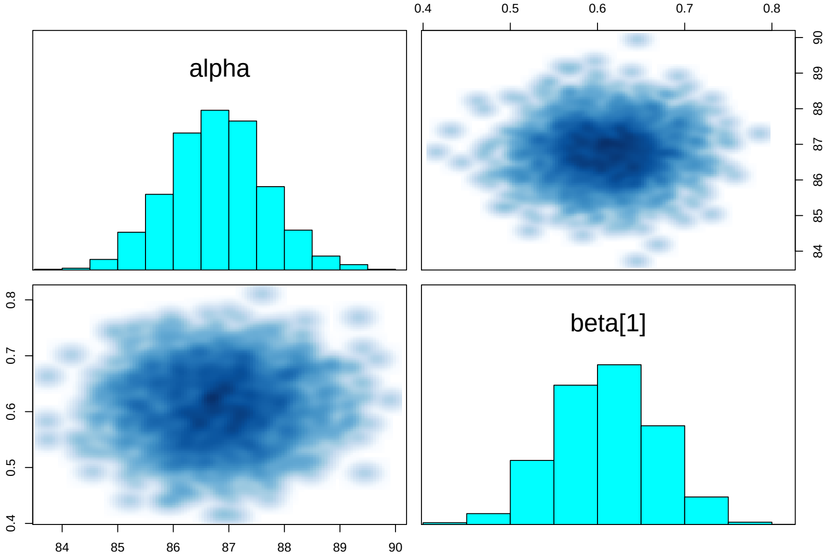
```
summary(fit2)$summary[c("alpha", "beta[1]"),]
```

```
##                mean      se_mean         sd       2.5%        25%
## alpha    86.7922708 0.0133140856 0.87522474 85.1051603 86.2002044 86.
## beta[1]   0.6102857 0.0008872342 0.05774695  0.5002678  0.5706817  0.
##                 75%       97.5%   n_eff      Rhat
## alpha    87.3572998 88.5519113 4321.32 0.9999881
## beta[1]   0.6510715  0.7195984 4236.25 1.0000424
```

What's different? What's the same?

Now look at joint posteriors

```
pairs(fit2, pars = c("alpha", "beta"))
```



What do you notice? Why does this matter?

Centering predictors

- When the mean of the predictors is far away from zero, changes in the slope induce the opposite change in the intercept
- Hard to interpret what intercepts mean
- Harder to sample: reducing correlation may speed up convergence

Changing prior information

Changing prior information

What if we knew with relative certainty that there's a 1:1 correspondence between kid's score and mother's IQ? How would we encode this information?

Changing prior information

$$\beta \sim N(1, 0.01^2)$$

Let's fit this:

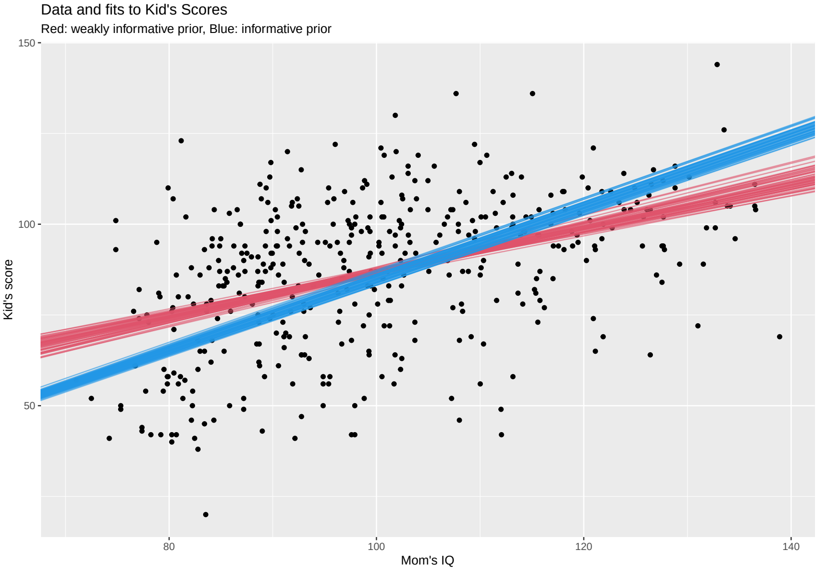
```
data <- list(y = y,
             N = length(y),
             K = 1,
             X = as.matrix(kidiq$mom_iq - mean(kidiq$mom_iq)))
fit3 <- stan(file = "kids5.stan",
            data = data)
```

Summary of fit

```
summary(fit3)$summary[c("alpha", "beta[1]"),]
```

```
##                 mean      se_mean          sd      2.5%        25%        50%
## alpha   86.7774869 0.0117989571 0.736762941 85.3314606 86.2998637 86.7811162
## beta[1]  0.9844801 0.0001462363 0.009745722  0.9653347  0.9777679  0.9845242
##               75%     97.5%    n_eff      Rhat
## alpha   87.251367 88.223460 3899.135 0.9993667
## beta[1]  0.990951  1.003995 4441.375 0.9998250
```

Comparison with weakly informative priors



Comments

- Okay, maybe this was a bad decision in this context, but when might we want to consider more informative priors?
- Measurement error?
- Less data?
- Previous evidence?

Summary

Bayesian inference for linear regression

- Focus on simulation-based inference and prediction, rather than point estimates
- Easy to propagate uncertainty to predictions, functions of estimated parameters

Lab: practice with kids dataset