Bayesian model checking

Stat 340, Fall 2021

Your turn 1

Suppose we want to fit a model with the following sampling model:

$$Y_i|\beta_0, \beta_1, \sigma \overset{ind}{\sim} \mathcal{N}(\mu_i, \sigma), \quad \mu_i = \beta_0 + \beta_1 \texttt{aroma}$$

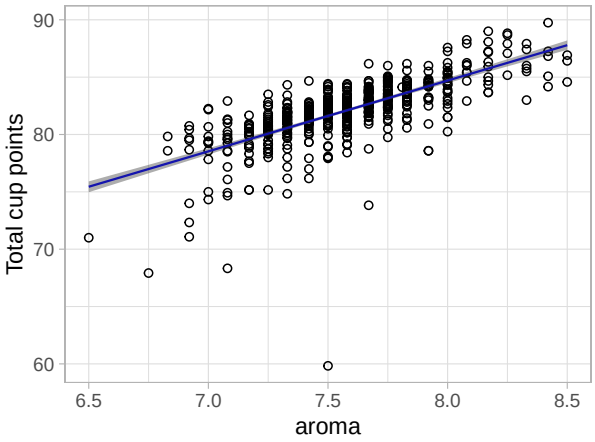
What assumptions/conditions does this model require?

Your turn 2

The coffee_ratings data includes ratings and features of 1339 different batches of beans grown on 571 different farms. Explain why using this data to model ratings by aroma likely violates the independence assumption.

Your turn 3

Based on this plot of the fitted model, do you see any issues with our assumptions?



Model setup and simulation

We're using this data set from `bayesrules` in model checking:

```
library(bayesrules)
set.seed(84735)
new_coffee <- coffee_ratings %>%
  group_by(farm_name) %>%
```

```
  sample_n(1) %>%
  ungroup()
```

We can fit a weakly informative model in JAGS:

```
slr_model <- "model{
# Sampling model
for(i in 1:n) {
  y[i] ~ dnorm(beta0 + beta1 * x[i], invsigma2)
}

# Priors
beta0 ~ dnorm(0, pow(100, -2))
beta1 ~ dnorm(0, pow(100, -2))
invsigma2 ~ dgamma(1, 1)
sigma <- pow(invsigma2, -1/2)
}"

slr_draws <- run.jags(
  slr_model,
  monitor = c("beta0", "beta1", "sigma"),
  data = list(y = new_coffee$total_cup_points,
              x = new_coffee$aroma,
              n = nrow(new_coffee)
              ),
  n.chains = 3,
  burnin = 5000,
  adapt = 1000,
  sample = 5000,
  thin = 80
)
```

Residual plots

- In previous courses, you used residual plots to diagnose linearity, constant variance assumptions
- $e_i = y - (\widehat{\beta}_0 + \widehat{\beta}_1 x_i)$
- To create a Bayesian analog, use posterior point estimates for $\widehat{\beta}_0$ and $\widehat{\beta}_1$

If you run multiple chains, `tidybayes::tidy_draws` makes it easy to combine them into a single data frame

```
post_draws <- tidybayes::tidy_draws(slr_draws$mcmc)
```

To calculate $\widehat{\beta}_0 + \widehat{\beta}_1 x_i$ and e_i, I find it easiest to add columns to the data

```
resids <- new_coffee %>%
  mutate(
    beta0 = median(post_draws$beta0),
    beta1 = median(post_draws$beta1),
    yhat  = beta0 + beta1 * aroma,
    resid = total_cup_points - yhat
  )
```

Now, plot the residuals:

```
ggplot(data = resids, aes(x = aroma, y = resid)) +
  geom_hline(yintercept = 0, linetype = 2, color = "blue") +
  geom_point()
```

Posterior prediction plots

Create a function to calculate μ_i given an x_i value

```
mu_link <- function(x) {
  post_draws[["beta0"]] + post_draws[["beta1"]] * x
}
```

Calculate S μ_i's for each values in the x vector (`new_coffee$aroma` here)

```
mu_draws<- sapply(new_coffee$aroma, mu_link)
```

Now, we have a $S \times n$ matrix, each column corresponds to an x_i

Simulate one $\tilde{y}_i$ for each μ_i in each column

```
S <- nrow(post_draws)
y_draws <- apply(mu_draws, 2, function(x) rnorm(S, x, post_draws[["sigma"]]))
```

Now, we have a $S \times n$ matrix of simulated observations

Calculate the mean and PI for each column

```
y_means <- colMeans(y_draws)
y_pis   <- apply(y_draws, 2, quantile, probs = c(0.05, 0.95))
```

To use `ggplot2`, let's combine the results into a data frame

```
post_pred_data <- data.frame(
  y = new_coffee$total_cup_points, # original y
  y_pred = y_means,                # avg. predicted response
  y_lo = y_pis[1,],                # lower bound of predicted response
  y_hi = y_pis[2,]                 # upper bound of predicted response
)
```

Render the plot

```
ggplot(post_pred_data, aes(x = y)) +
  geom_point(aes(y = y_means), alpha = 0.5) +
  geom_linerange(aes(ymin = y_lo, ymax = y_hi), alpha = 0.5) +
  geom_abline(slope = 1, intercept = 0, color = "dodgerblue")
```

Predictive residuals

We can start with our posterior prediction plot data set, then calculate the midpoint and endpoints of our residual intervals using those prediction intervals

```
pred_resids <- post_pred_data %>%
  mutate(
    resid = y - y_pred,
    resid_lo = y - y_lo,
    resid_hi = y - y_hi,
    x = jitter(new_coffee$aroma, factor = 2.5) # avoiding some overlap
)
```

Create the plot

```
ggplot(pred_resids, aes(x = x)) +
  geom_point(aes(y = resid), alpha = 0.5) +
  geom_linerange(aes(ymin = resid_lo, ymax = resid_hi), alpha = 0.5) +
  geom_hline(yintercept = 0, color = "dodgerblue")
```

Posterior predictive checks

We can also display the density of the response for the observed data and simulated data sets.

```
library(bayesplot)
ppc_dens_overlay(y = new_coffee$total_cup_points,
                 yrep = y_draws[1:100,])
```

Informative priors

Centering example

- On an average temperature day, say 65 or 70 degrees for D.C., there are typically around 5000 riders, though this average could be somewhere between 3000 and 7000.
- For every one degree increase in temperature, ridership typically increases by 100 rides, though this average increase could be as low as 20 or as high as 180.
- At any given temperature, daily ridership will tend to vary with a moderate standard deviation of 1250 rides.

Standardization example

How can we specify our priors for β_0 and β_1 based on the following prior information?

- On an average temperature day, say 65 or 70 degrees for D.C., there are typically around 5000 riders, though this average could be somewhere between 3000 and 7000.
- The correlation between ridership and temperature is expected to be of moderate strength, most likely between 0.45 and 0.75.

Conditional means example

How can we specify our priors for β_0 and β_1 based on the following prior information?

- On a 70 °F day there are typically around 4000 riders, though this average could be somewhere between 3000 and 5000.
- On a 50 °F day there are typically around 2000 riders, though this average could be somewhere between 1200 and 2800.

Inducing priors in JAGS

To use an induced prior, tell JAGS how to calculate the *original* parameter from the alternative parameterization (e.g., from μ_1 and μ_2)

```
modelString = "model{

# Sampling model
for(i in 1:n) {
  y[i] ~ dnorm(beta0 + beta1 * x[i], invsigma2)
}

# Prior models
mu1 ~ dnorm(m1, s1)                          # Specify prior info #<<
mu2 ~ dnorm(m2, s2)                          # Specify prior info #<<
beta1 <- (mu2 - mu1) / (x2 - x1)             # Calc. beta1 from mu1 & mu2 #<<
beta0 <- mu1 - x1 * (mu2 - mu1) / (x2 - x1) # Calc. beta0 from mu1 & mu2 #<<

invsigma2 ~ dgamma(1, 1)
sigma <- pow(invsigma2, -1/2)
}"
```