Log link regression priors

Adam Smith

2022-12-22

```
library(tidyverse)
library(rethinking)
library(kableExtra)
library(patchwork)
library(scales)

birds <- read.csv("bird_data_bayes_stats.csv")
```

Priors in log-link regression models

When modeling counts, presence-absence or other data that are constrained to be non-negative, models often use a log-link. It is important to remember the link function when thinking about appropriate priors for parameters in the model.

```
sp_indicators <- birds %>% #creates a data frame that list each species indicator number
  filter(bird_guild == "Forest") %>%
  select(english) %>%
  distinct() %>% # selects just the unique rows
  mutate(species_ind = as.integer(factor(english))) %>%
  arrange(species_ind)


# forest cover compare partial pool with no pooling ------------------------------------------


birds_sub <- birds %>%
  mutate(route_ind = as.integer(factor(RouteName)),
         #species_ind = as.integer(factor(english)),
         guild_ind = as.integer(factor(bird_guild)),
         foot_scale = scale(human_footprint),
         change_foot_scale = scale(change_human_footprint),
         pforest_scale = (proportion_forest-0.5)) %>%
  inner_join(sp_indicators,
             by = c("english")) %>%
  select(count,species_ind,pforest_scale) # this removes all unnecessary columns to keep ulam() happy
```

Exploring priors in a simplified way

If the model is relatively simple, you can explore the priors distributions by transforming the priors (which are coded on the log-scale) into the original scale to see how they relate to the data.

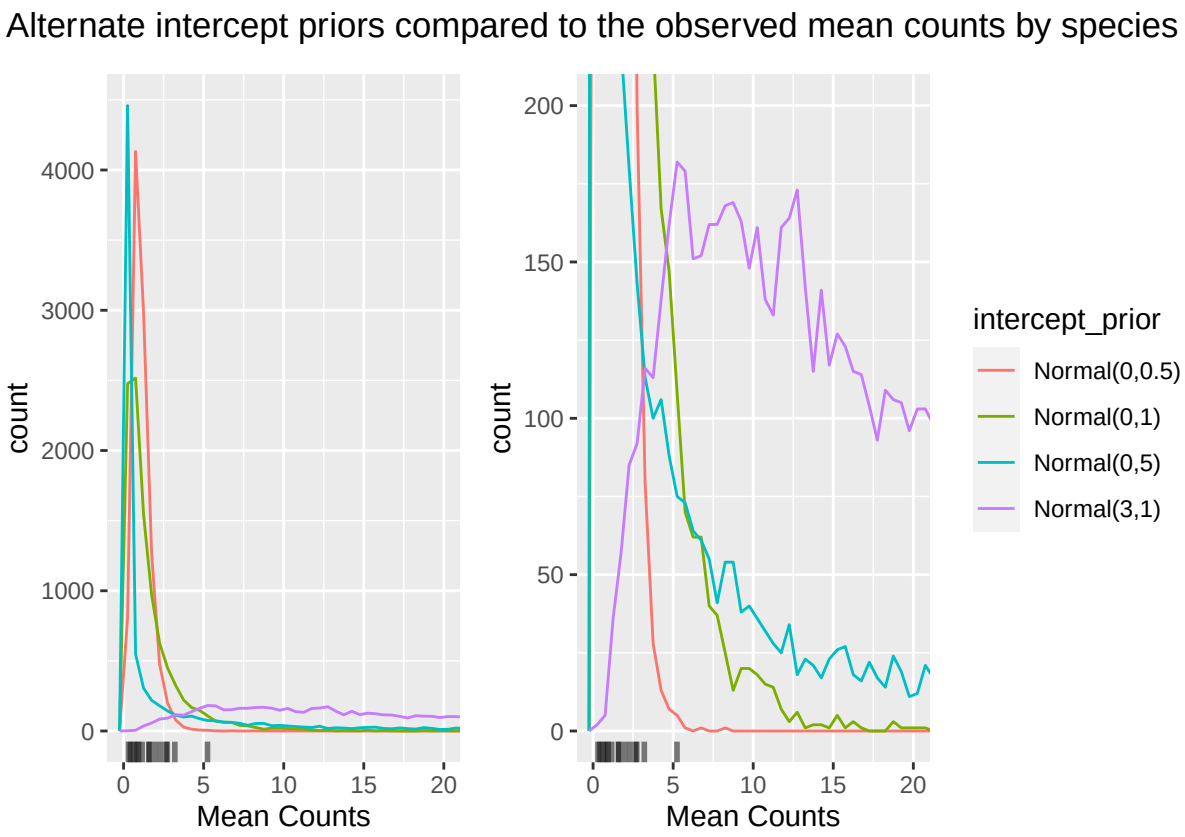
For example, here are 4 alternate priors for the intercept terms in a regression model estimating the abundance of forest birds. I've generated random draws from these priors, re-transformed them using the `exp()` function (`exp(log(a)) = a`), then added a rug-plot (dark lines along the x-axis) to indicate the observed means for each species. First the code, then the graphs.

```
obs_means <- birds_sub %>%
  group_by(species_ind) %>%
  summarise(mean = mean(count))

nn <- 10000#number of samples to draw from the prior
mean_priors = data.frame(prior = exp(c(rnorm(nn,0,1),
                                       rnorm(nn,3,1),
                                       rnorm(nn,0,0.5),
                                       rnorm(nn,0,5))),
                         intercept_prior = c(rep("Normal(0,1)",nn),
                                             rep("Normal(3,1)",nn),
                                             rep("Normal(0,0.5)",nn),
                                             rep("Normal(0,5)",nn)))
bks = seq(0,100,0.5)
prior_obs <- ggplot(data = mean_priors)+
  geom_freqpoly(aes(x = prior,colour = intercept_prior),breaks = bks,center = 0)+
  geom_rug(data = obs_means, aes(x = mean),inherit.aes = FALSE,
           size = 1,alpha = 0.5)+
  coord_cartesian(xlim = c(0,20))+
  xlab("Mean Counts")

prior_obs_zoom <- ggplot(data = mean_priors)+
  geom_freqpoly(aes(x = prior,colour = intercept_prior),breaks = bks,center = 0)+
  geom_rug(data = obs_means, aes(x = mean),inherit.aes = FALSE,
           size = 1,alpha = 0.5)+
  coord_cartesian(ylim = c(0,200),
                  xlim = c(0,20))+
  xlab("Mean Counts")

print(prior_obs + prior_obs_zoom + plot_layout(guides = "collect")
      + plot_annotation(title = "Alternate intercept priors compared to the observed mean counts by spe
```



The plot on the left shows most of prior distributions, the version on the right is the same plot, but zoomed-in to show detaily on the range of the observed means, which are all less than 10. So the priors with mean = 0 seem to capture the observed means relatively well, most mean counts are pretty small (< 3-5 birds). The prior with a positive mean value (`normal(3,1)`) doesn't seem to fit the data at all. This positive prior could seem perfectly intuitive if you had forgotten about the log-link in the model (counts are always positive), but it actually has relatively little prior probability mass at the most commonly observed mean counts (< 1) and has most of its prior probability mass at values greater than the highest observed mean count. If you studied birds a lot, you might also expect to see mean counts for some very common species that were on the order of 10 or more. In that case the `normal(0,0.5)` prior seems like it might be a little too narrow.

Combined interept and slope

You can also combine multiple priors to explore the combined effects on the expected values. As a next step, we can take what we learned above about a reasonable prior for the intercept, and combine with some possible priors for the slopes. A full exploration of the priors is possible using `ulam()` and `extract.priors()` from the `rethinking` package. However, I often find it faster to combine the priors from multiple parameters, just using the algebra from the model statement. So for example, in a basic log-link (Poisson or Negative Binomial) regression, the main line of code that defines the relationship between the response and the parameters/predictors is `log(lambda) = intercept + slope*predictor`, or the equivalent `lambda = exp(intercept + slope*predictor)`. `lambda` is the mean of the Poisson distribution, so it's scaling is relatively intuitive (mean count of birds). However, it's the log of lambda that is directly linked to the parameters and the predictors, so that complicates our intuition. In addition, the `slope*predictor` component means that the scale of the prior for the parameter will depend on the scale of the predictor as well. For example, if the predictor was percent of forest, then at its maximum value we'd be multiplying the parameter by 100. Whereas if the predictor was proportion of forest, at it's maximum value we'd multiply

the parameter by 1. But the number of birds hasn't changed, so the scale of the parameter will strongly depend on the scale of the predictor. So there are three aspects to consider: the link-function, the scale of the prior, and the scale of the predictor. We can explore different values of the prior and the scaling of the predictor below to demonstrate.

To start we set up a dataframe of forest cover values that span the available range and have a few different scalings: the raw proportion of forest in the surrounding landscape (100 values between 0 and 0.8); then a centered version, centered on the mean (ranging from -0.4 : 0.4); and a scaled version that is mean centered and has standard deviation = 1 `scale()`. For the first two a one-unit change represents more than the entire range of values, while for the scaled version a one-unit change represents one standard deviation.

```
#setting up a dataframe of forest cover values that span the available range
# and have a few different scalings
forest_range <- data.frame(proportion_forest = seq(0,0.8,length.out = 100)) %>%
  mutate(pforest_centered = proportion_forest-mean(proportion_forest),
         pforest_scaled = scale(proportion_forest))
```

The we'll draw 100 samples from each of three different priors for the slope parameters and pair them with random draws of the standard normal prior for the intercept. The three priors for the slope will be all be zero-mean normal distributions with different standard deviations: 1, 3, and 5. So the distributions will have the same shape and mean, but vary in their spread: larger values of SD allow greater variation in the range of slopes.

```
nn <- 100#number of samples to draw from the prior
priors_changes = data.frame(intercept = c(rnorm(nn*3,0,1)),
                            slope = c(rnorm(nn,0,1),
                                      rnorm(nn,0,3),
                                      rnorm(nn,0,5)),
                            slope_prior = c(rep("Normal(0,1)",nn),
                                            rep("Normal(0,3)",nn),
                                            rep("Normal(0,5)",nn))) %>%
  arrange(slope_prior,slope) %>%
  mutate(draw = rep(1:nn,times = 3)) # adding a unique integer to track the samples
```

Then we merge the priors with the values of forest cover, expanding the priors dataframe for every level of forest cover using `expand_grid()`. Finally, to calculate the expected values of lambda, we use the `exp(intercept+slope*proportion_forest)` algebra from the model description.

```
# calculating the expected counts for each
priors_changes <- priors_changes %>%
  expand_grid(.,forest_range) %>% #this replicates the prior_changes dataframe for every level of fores
  mutate(lambda_prop = exp(intercept + slope*proportion_forest),
         lambda_centered = exp(intercept + slope*pforest_centered),
         lambda_scaled = exp(intercept + slope*pforest_scaled)) %>%
  arrange(slope_prior,draw)
```

Then we plot the nine groups of expected values of lambda by the combinations of three priors and three predictors. The plots are scaled to reflect the expected counts on the y-axis and the proportion forest on the x-axis. All of the x-axes are scaled the same to facilitate the comparison, but each column of the plots is labeled to indicate the scaling of the predictor variable.

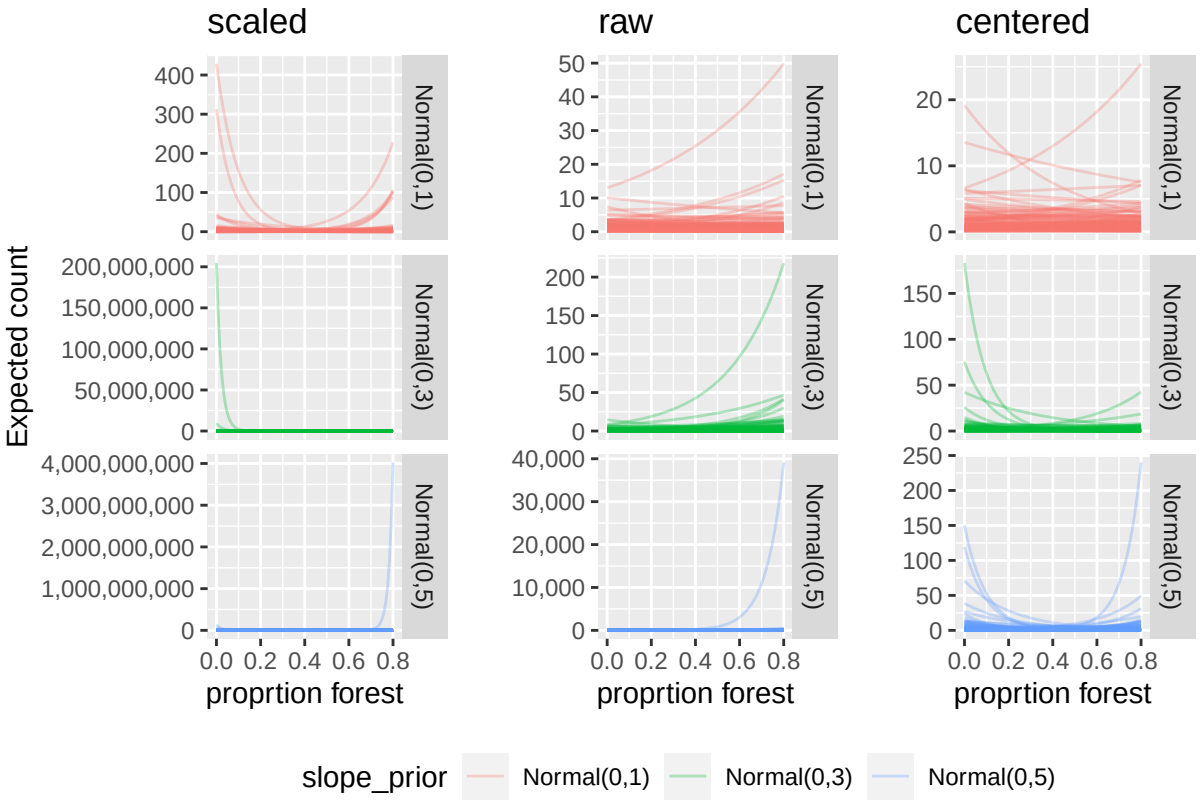
```
prior_slopes1 <- ggplot(data = priors_changes)+
  geom_line(aes(x = proportion_forest,y = lambda_prop,
```

```
                group = draw,
                colour = slope_prior),
            alpha = 0.3)+
  facet_grid(rows = vars(slope_prior),
             scales = "free_y")+
  scale_y_continuous(labels = comma)+
  labs(title = "raw")+
  ylab("")+
  xlab("proprtion forest")

prior_slopes2 <- ggplot(data = priors_changes)+
  geom_line(aes(x = proportion_forest,y = lambda_centered,
                group = draw,
                colour = slope_prior),
            alpha = 0.3)+
  facet_grid(rows = vars(slope_prior),
             scales = "free_y")+
  scale_y_continuous(labels = comma)+
  labs(title = "centered")+
  ylab("")+
  xlab("proprtion forest")

prior_slopes3 <- ggplot(data = priors_changes)+
  geom_line(aes(x = proportion_forest,y = lambda_scaled,
                group = draw,
                colour = slope_prior),
            alpha = 0.3)+
  facet_grid(rows = vars(slope_prior),
             scales = "free_y")+
  scale_y_continuous( labels = comma)+
  labs(title = "scaled")+
  ylab("Expected count")+
  xlab("proprtion forest")

print(prior_slopes3 + prior_slopes1 + prior_slopes2 +
        plot_layout(guides = "collect") &
        theme(legend.position = "bottom"))
```



The graphs show the expected relationships between number of birds for some hypothetical species and the proportion of forest in the landscape. For the scaled predictor values (left column) standard normal prior `normal(0,1)` does well, in that the number of birds observed might at an extreme go from a few to >100 birds, but most relationships seem more moderate. So this is a flexible prior that likely covers all of the relationships we see in our data (given the maximum values of observed counts), but doesn't constrain the estimates too much. By comparison using the same scaled predictor, the other priors generate ridiculous estimates of abundance (1 Billion birds!). For the raw proportion forest or the centered proportion forest, the standard normal prior seems pretty restrictive on the rates of change. The steepest lines appear to double in abundance across the range of forest amount. The same prior that fits the scaled predictor might have an unintended regularizing effect on the estimates. Many of the priors that were used in the assignment 3 examples showed this kind of regularizing behaviour. Many of you found that your partial pooling model's slope estimates were generally larger than the slope estimates from the non-pooling model. This was often because the non-pooling model had a prior that was too narrow and tended to shrink the slope estimates towards 0. For the non-scaled (centered and raw) proportion forest predictors, the `normal(0,3)` prior seems more reasonable. The `normal(0,5)` would probably be ok as well, but it does seem to suggest there's some prior probability of some pretty extreme overall changes.

Take-home Message

When thinking about appropriate priors, you need to consider:

1. The parameter of interest (intercept or slope)
2. The link-function
3. The scale of the predictor (if it's a slope parameter)