

# 11 God Spiked the Integers

---

The cold of space is named Kelvin, about 3 degrees Kelvin, or 3 degrees centigrade above absolute zero. Kelvin is also the name of a river in Scotland, near Glasgow. The same river gave its name to William Thomson, the Lord Kelvin (1824–1907), the first scientist in the United Kingdom to be granted a noble title. Thomson studied thermodynamics in his laboratory in Glasgow, and now the cold of space bears the name of a Scottish river.

Lord Kelvin befittingly also researched water. He invented several tide prediction engines. These were essentially mechanical computers that calculated the tides (FIGURE 11.1). All the gears and cables comprised a set of oscillators that produced accurate tide predictions. But when you look at such a machine, most of it is internal states, not the predictions. It would be quite hard to inspect any one of the gears at the bottom and know when to expect the tide, because the predictions emerge from the combination of internal states.

**GENERALIZED LINEAR MODELS** (GLMs) are a lot like these early mechanical computers. The moving pieces within them, the parameters, interact to produce non-obvious predictions. But we can't read the parameters directly to understand the predictions. This is quite different than the Gaussian linear models of previous chapters, where individual parameters had clear meanings on the prediction scale. Mastering GLMs requires a little more attention. They are always confusing, when you first try to grasp how they operate.

The most common and useful generalized linear models are models for counts. Counts are non-negative integers—0, 1, 2, and so on. They are the basis of all mathematics, the first bits that children learn. But they are also intoxicatingly complicated to model—hence the apocryphal slogan that titles this chapter<sup>173</sup>. The essential problem is this: When what we wish to predict is a count, the scale of the parameters is never the same as the scale of the outcome. A count golem, like a tide prediction engine, has a whirring machinery underneath that doesn't resemble the output. Keeping the tide engine in mind, you can master these models and use them responsibly.

We will engineer complete examples of the two most common types of count model. **BINOMIAL REGRESSION** is the name we'll use for a family of related procedures that all model a binary classification—alive/dead, accept/reject, left/right—for which the total of both categories is known. This is like the marble and globe tossing examples from Chapter 2. But now you get to incorporate predictor variables. **POISSON REGRESSION** is a GLM that models a count with an unknown maximum—number of elephants in Kenya, number of applications to a PhD program, number of significance tests in an issue of *Psychological Science*. As described in Chapter 10, the Poisson model is a special case of binomial. At the end, the chapter describes some other count regressions.

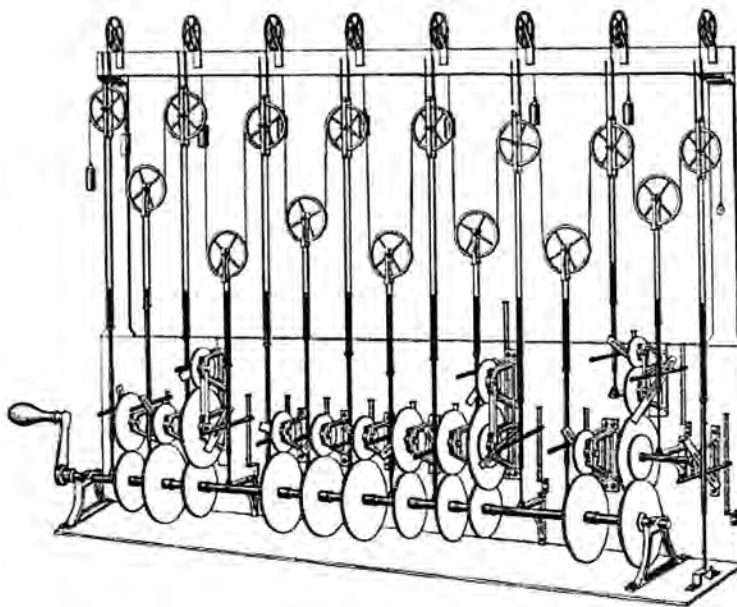


FIGURE 11.1. William Thomson's third tide prediction design. (Image source: [https://en.wikipedia.org/wiki/Tide-predicting\\_machine](https://en.wikipedia.org/wiki/Tide-predicting_machine))

All of the examples in this chapter, and the chapters to come, use all of the tools introduced in previous chapters. Regularizing priors, information criteria, and MCMC estimation are woven into the data analysis examples. So as you work through the examples that introduced each new type of GLM, you'll also get to practice and better understand previous lessons.

### 11.1. Binomial regression

Think back to the early chapters, the globe tossing model. That model was a binomial model. The outcome was a count of water samples. But it wasn't yet a generalized linear model, because there were no predictor variables to relate to the outcome. That's our work now—to mate observed counts to variables that are associated with different average counts.

The binomial distribution is denoted:

$$y \sim \text{Binomial}(n, p)$$

where  $y$  is a count (zero or a positive whole number),  $p$  is the probability any particular “trial” is a success, and  $n$  is the number of trials. As proved in the previous chapter, as the basis for a generalized linear model, the binomial distribution has maximum entropy when each trial must result in one of two events and the expected value is constant. There is no other pre-observation probability assumption for such a variable that will have higher entropy. It is the flattest data prior we can use, given the known constraints on the values.

There are two common flavors of GLM that use binomial probability functions, and they are really the same model, just with the data organized in different ways.

- (1) **LOGISTIC REGRESSION** is the common name when the data are organized into single-trial cases, such that the outcome variable can only take values 0 and 1.
- (2) When individual trials with the same covariate values are instead aggregated together, it is common to speak of an **AGGREGATED BINOMIAL REGRESSION**. In this case, the outcome can take the value zero or any positive integer up to  $n$ , the number of trials.

Both flavors use the same logit link function (page 316), so both may sometimes be called “logistic” regression, as the inverse of the logit function is the logistic. Either form of binomial regression can be converted into the other by aggregating (logistic to aggregated) or exploding (aggregated to logistic) the outcome variable. We’ll fully work an example of each.

Like other GLMs, binomial regression is never guaranteed to produce a nice multivariate Gaussian posterior distribution. So quadratic approximation is not always satisfactory. We’ll work some examples using `quap`, but we’ll also check the inferences against MCMC sampling, using `ulam`. The reason to do it both ways is so you can get a sense of both how often quadratic approximation works, even when in principle it should not, and why it fails in particular contexts. This is useful, because even if you never use quadratic approximation again, your Frequentist colleagues use it all the time, and you might want to be skeptical of their estimates.

**11.1.1. Logistic regression: Prosocial chimpanzees.** The data for this example come from an experiment<sup>174</sup> aimed at evaluating the prosocial tendencies of chimpanzees (*Pan troglodytes*). The experimental structure mimics many common experiments conducted on human students (*Homo sapiens studiensis*) by economists and psychologists. A focal chimpanzee sits at one end of a long table with two levers, one on the left and one on the right in **FIGURE 11.2**. On the table are four dishes which may contain desirable food items. The two dishes on the right side of the table are attached by a mechanism to the right-hand lever. The two dishes on the left side are similarly attached to the left-hand lever.

When either the left or right lever is pulled by the focal animal, the two dishes on the same side slide towards opposite ends of the table. This delivers whatever is in those dishes to the opposite ends. In all experimental trials, both dishes on the focal animal’s side contain food items. But only one of the dishes on the other side of the table contains a food item. Therefore while both levers deliver food to the focal animal, only one of the levers delivers food to the other side of the table.

There are two experimental conditions. In the *partner* condition, another chimpanzee is seated at the opposite end of the table, as pictured in **FIGURE 11.2**. In the control condition, the other side of the table is empty. Finally, two counterbalancing treatments alternate which side, left or right, has a food item for the other side of the table. This helps detect any handedness preferences for individual focal animals.

When human students participate in an experiment like this, they nearly always choose the lever linked to two pieces of food, the *prosocial* option, but only when another student sits on the opposite side of the table. The motivating question is whether a focal chimpanzee behaves similarly, choosing the prosocial option more often when another animal is present. In terms of linear models, we want to estimate the interaction between condition (presence or absence of another animal) and option (which side is prosocial).

Load the data from the `rethinking` package:

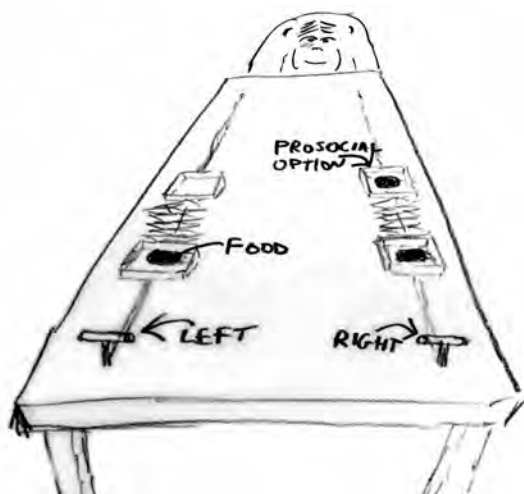


FIGURE 11.2. Chimpanzee prosociality experiment, as seen from the perspective of the focal animal. The left and right levers are indicated in the foreground. Pulling either expands an accordion device in the center, pushing the food trays towards both ends of the table. Both food trays close to the focal animal have food in them. Only one of the food trays on the other side contains food. The partner condition means another animal, as pictured, sits on the other end of the table. Otherwise, the other end was empty.

```
R code
11.1 library(rethinking)
      data(chimpanzees)
      d <- chimpanzees
```

Take a look at the built-in help, `?chimpanzees`, for details on all of the available variables. We're going to focus on `pulled_left` as the outcome to predict, with `prosoc_left` and `condition` as predictor variables. The outcome `pulled_left` is a 0 or 1 indicator that the focal animal pulled the left-hand lever. The predictor `prosoc_left` is a 0/1 indicator that the left-hand lever was (1) or was not (0) attached to the prosocial option, the side with two pieces of food. The `condition` predictor is another 0/1 indicator, with value 1 for the partner condition and value 0 for the control condition.

We'll want to infer what happens in each combination of `prosoc_left` and `condition`. There are four combinations:

- (1) `prosoc_left`= 0 and `condition`= 0: Two food items on right and no partner.
- (2) `prosoc_left`= 1 and `condition`= 0: Two food items on left and no partner.
- (3) `prosoc_left`= 0 and `condition`= 1: Two food items on right and partner present.
- (4) `prosoc_left`= 1 and `condition`= 1: Two food items on left and partner present.

The conventional thing to do here is use these dummy variables to build a linear interaction model. We aren't going to do that, for the reason discussed back in [Chapter 5](#): Using dummy variables makes it hard to construct sensible priors. So instead let's build an index variable containing the values 1 through 4, to index the combinations above. A very quick way to do this is:

```
R code
11.2 d$treatment <- 1 + d$prosoc_left + 2*d$condition
```

Now `treatment` contains the values 1 through 4, matching the numbers in the list above. You can verify by using cross-tabs:

```
R code
11.3 xtabs( ~ treatment + prosoc_left + condition , d )
```

The output isn't shown. There are many ways to construct new variables like this, including mutant helper functions. But often all you need is a little arithmetic.

Now for our target model. Since this is an experiment, the structure tells us the model relevant to inference. The model implied by the research question is, in mathematical form:

$$\begin{aligned} L_i &\sim \text{Binomial}(1, p_i) \\ \text{logit}(p_i) &= \alpha_{\text{ACTOR}[i]} + \beta_{\text{TREATMENT}[i]} \\ \alpha_j &\sim \text{to be determined} \\ \beta_k &\sim \text{to be determined} \end{aligned}$$

Here  $L$  indicates the 0/1 variable `pulled_left`. Since the outcome counts are just 0 or 1, you might see the same type of model defined using a Bernoulli distribution:

$$L_i \sim \text{Bernoulli}(p_i)$$

This is just another way of saying  $\text{Binomial}(1, p_i)$ . Either way, the model above implies 7  $\alpha$  parameters, one for each chimpanzee, and 4 treatment parameters, one for each unique combination of the position of the prosocial option and the presence of a partner. In principle, we could specify a model that allows every chimpanzee to have their own 4 unique treatment parameters. If that sounds fun to you, I have good news. We'll do exactly that, in a later chapter.

I've left the priors above "to be determined." Let's determine them. I was trying to warm you up for prior predictive simulation earlier in the book. Now with GLMs, it is really going to pay off. Let's consider a runt of a logistic regression, with just a single  $\alpha$  parameter in the linear model:

$$\begin{aligned} L_i &\sim \text{Binomial}(1, p_i) \\ \text{logit}(p_i) &= \alpha \\ \alpha &\sim \text{Normal}(0, \omega) \end{aligned}$$

We need to pick a value for  $\omega$ . To emphasize the madness of conventional flat priors, let's start with something rather flat, like  $\omega = 10$ .

```
m11.1 <- quap(
  alist(
    pulled_left ~ dbinom( 1 , p ) ,
    logit(p) <- a ,
    a ~ dnorm( 0 , 10 )
  ) , data=d )
```

R code  
11.4

Now let's sample from the prior:

```
set.seed(1999)
prior <- extract.prior( m11.1 , n=1e4 )
```

R code  
11.5

One step remains. We need to convert the parameter to the outcome scale. This means using the **INVERSE-LINK FUNCTION**, as discussed in the previous chapter. In this case, the link function is `logit`, so the inverse link is `inv_logit`.

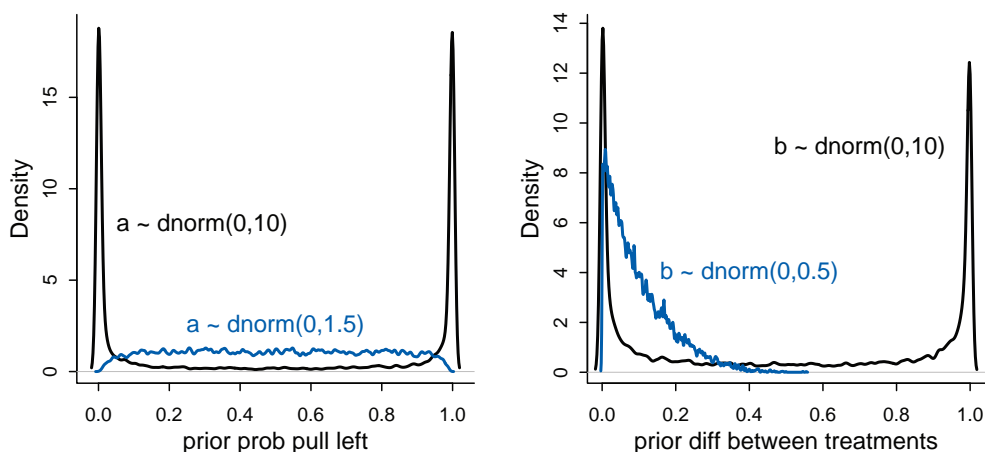


FIGURE 11.3. Prior predictive simulations for the most basic logistic regression. Black density: A flat Normal(0,10) prior on the intercept produces a very non-flat prior distribution on the outcome scale. Blue density: A more concentrated Normal(0,1.5) prior produces something more reasonable.

```
R code
11.6 p <- inv_logit( prior$a )
      dens( p , adj=0.1 )
```

I've displayed the resulting prior distribution in the left-hand plot of [FIGURE 11.3](#). Notice that most of the probability mass is piled up near zero and one. The model thinks, before it sees the data, that chimpanzees either never or always pull the left lever. This is clearly silly, and will generate unnecessary inference error. A flat prior in the logit space is not a flat prior in the outcome probability space. The blue distribution in the same plot shows the same model but now with  $\omega = 1.5$ . You can modify the code above to reproduce this. Now the prior probability on the outcome scale is rather flat. This is probably much flatter than is optimal, since probabilities near the center are more plausible. But this is better than the default priors most people use most of the time. We'll use it.

Now we need to determine a prior for the treatment effects, the  $\beta$  parameters. We could default to using the same Normal(0,1.5) prior for the treatment effects, on the reasoning that they are also just intercepts, one intercept for each treatment. But to drive home the weirdness of conventionally flat priors, let's see what Normal(0,10) looks like.

```
R code
11.7 m11.2 <- quap(
      alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a + b[treatment] ,
        a ~ dnorm( 0 , 1.5 ) ,
        b[treatment] ~ dnorm( 0 , 10 )
      ) , data=d )
      set.seed(1999)
```

```
prior <- extract.prior( m11.2 , n=1e4 )
p <- sapply( 1:4 , function(k) inv_logit( prior$a + prior$b[,k] ) )
```

The code just above computes the prior probability of pulling left for each treatment. We are interested in what the priors imply about the prior *differences* among treatments. So let's plot the absolute prior difference between the first two treatments.

```
dens( abs( p[,1] - p[,2] ) , adj=0.1 )
```

R code  
11.8

I show this distribution on the right in [FIGURE 11.3](#). Just like with  $\alpha$ , a flat prior on the logit scale piles up nearly all of the prior probability on zero and one—the model believes, before it sees that data, that the treatments are either completely alike or completely different. Maybe there are contexts in which such a prior makes sense. But they don't make sense here. Typical behavioral treatments have modest effects on chimpanzees and humans alike.

The blue distribution in the same figure shows the code above repeated using a Normal(0,0.5) prior instead. This prior is now concentrated on low absolute differences. While a difference of zero has the highest prior probability, the average prior difference is:

```
m11.3 <- quap(
  alist(
    pulled_left ~ dbinom( 1 , p ) ,
    logit(p) <- a + b[treatment] ,
    a ~ dnorm( 0 , 1.5 ) ,
    b[treatment] ~ dnorm( 0 , 0.5 )
  ) , data=d )
set.seed(1999)
prior <- extract.prior( m11.3 , n=1e4 )
p <- sapply( 1:4 , function(k) inv_logit( prior$a + prior$b[,k] ) )
mean( abs( p[,1] - p[,2] ) )
```

R code  
11.9

```
[1] 0.09838663
```

About 10%. Extremely large differences are less plausible. However this is not a strong prior. If the data contain evidence of large differences, they will shine through. And keep in mind the lessons of [Chapter 7](#): We want our priors to be skeptical of large differences, so that we reduce overfitting. Good priors hurt fit to sample but are expected to improve prediction.

Finally, we have our complete model and are ready to add in all the individual chimpanzee parameters. Let's turn to Hamiltonian Monte Carlo to approximate the posterior, so you can get some practice with it. `quap` will actually do a fine job with this posterior, but only because the priors are sufficiently regularizing. In the practice problems at the end of the chapter, you'll compare the two engines on less regularized models. First prepare the data list:

```
# trimmed data list
dat_list <- list(
  pulled_left = d$pulled_left,
  actor = d$actor,
  treatment = as.integer(d$treatment) )
```

R code  
11.10

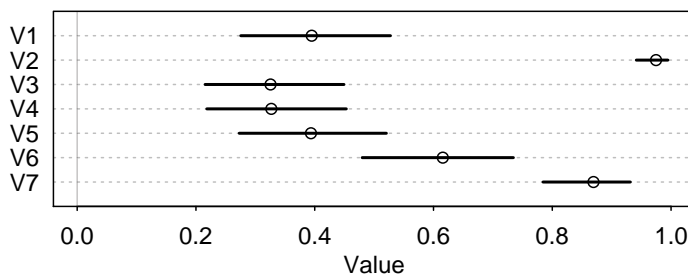
Now we can start the Markov chain. I'll add `log_lik=TRUE` to the call, so that `ulam` computes the values necessary for PSIS and WAIC. There is an Overthinking box at the end that explains this in great detail.

```
R code
11.11 m11.4 <- ulam(
      alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + b[treatment] ,
        a[actor] ~ dnorm( 0 , 1.5 ) ,
        b[treatment] ~ dnorm( 0 , 0.5 )
      ) , data=dat_list , chains=4 , log_lik=TRUE )
precis( m11.4 , depth=2 )
```

	mean	sd	5.5%	94.5%	n_eff	Rhat
a[1]	-0.45	0.32	-0.95	0.04	690	1
a[2]	3.86	0.73	2.78	5.09	1417	1
a[3]	-0.75	0.33	-1.28	-0.23	765	1
a[4]	-0.74	0.33	-1.26	-0.21	887	1
a[5]	-0.44	0.32	-0.94	0.10	743	1
a[6]	0.48	0.32	-0.02	1.00	894	1
a[7]	1.95	0.40	1.32	2.63	882	1
b[1]	-0.04	0.28	-0.51	0.40	669	1
b[2]	0.48	0.28	0.04	0.92	675	1
b[3]	-0.38	0.28	-0.83	0.06	768	1
b[4]	0.37	0.27	-0.07	0.79	666	1

This is the guts of the tide prediction engine. We'll need to do a little work to interpret it. The first 7 parameters are the intercepts unique to each chimpanzee. Each of these expresses the tendency of each individual to pull the left lever. Let's look at these on the outcome scale:

```
R code
11.12 post <- extract.samples(m11.4)
p_left <- inv_logit( post$a )
plot( precis( as.data.frame(p_left) ) , xlim=c(0,1) )
```



Each row is a chimpanzee, the numbers corresponding to the values in `actor`. Four of the individuals—numbers 1, 3, 4, and 5—show a preference for the right lever. Two individuals—numbers 2 and 7—show the opposite preference. Number 2's preference is very strong indeed. If you inspect the data, you'll see that actor 2 never once pulled the right lever in any trial or treatment. There are substantial differences among the actors in their baseline tendencies. This is exactly the kind of effect that makes pure experiments difficult in the

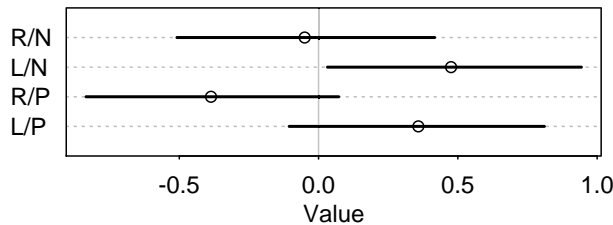


behavioral sciences. Having repeat measurements, like in this experiment, and measuring them is very useful.

Now let's consider the treatment effects, hopefully estimated more precisely because the model could subtract out the handedness variation among actors. On the logit scale:

```
labs <- c("R/N","L/N","R/P","L/P")
plot( precis( m11.4 , depth=2 , pars="b" ) , labels=labs )
```

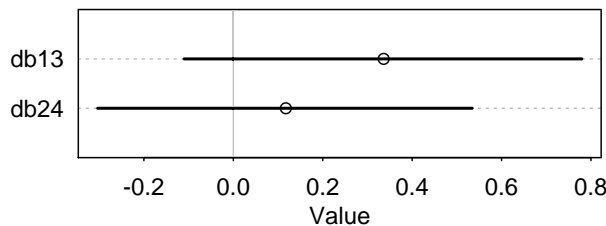
R code  
11.13



I've added treatment labels in place of the parameter names. L/N means “prosocial on left / no partner.” R/P means “prosocial on right / partner.” To understand these distributions, it'll help to consider our expectations. What we are looking for is evidence that the chimpanzees choose the prosocial option more when a partner is present. This implies comparing the first row with the third row and the second row with the fourth row. You can probably see already that there isn't much evidence of prosocial intention in these data. But let's calculate the differences between no-partner/partner and make sure.

```
diffs <- list(
  db13 = post$b[,1] - post$b[,3],
  db24 = post$b[,2] - post$b[,4] )
plot( precis(diffs) )
```

R code  
11.14



These are the **CONSTRASTS** between the no-partner/partner treatments. The scale is log-odds of pulling the left lever still. Remember the tide engine! db13 is the difference between no-partner/partner treatments when the prosocial option was on the right. So if there is evidence of more prosocial choice when partner is present, this will show up here as a larger difference, consistent with pulling right more when partner is present. There is indeed weak evidence that individuals pulled left more when the partner was absent, but the compatibility interval is quite wide. db24 is the same difference, but for when the prosocial option was on the left. Now negative differences would be consistent with more prosocial choice when partner is present. Clearly that is not the case. If anything, individuals chose prosocial more when partner was absent. Overall, there isn't any compelling evidence of prosocial choice in this experiment.

Now let's consider a posterior prediction check. Let's summarize the proportions of left pulls for each actor in each treatment and then plot against the posterior predictions. First, to calculate the proportion in each combination of actor and treatment:

```
R code
11.15 pl <- by( d$pullled_left , list( d$actor , d$treatment ) , mean )
      pl[1,]
```

```

      1      2      3      4
0.3333333 0.5000000 0.2777778 0.5555556
```

The result `pl` is a matrix with 7 rows and 4 columns. Each row is an individual chimpanzee. Each column is a treatment. And the cells contain proportions of pulls that were of the left lever. Above is the first row, showing the proportions for the first actor. The model will make predictions for these values, so we can see how the posterior predictions look against the raw data. Remember that we don't want an exact match—that would mean overfitting. But we would like to understand how the model sees the data and learn from any anomalies.

I've displayed these values, against the posterior predictions, in [FIGURE 11.4](#). The top plot is just the raw data. You can reproduce it with this code:

```
R code
11.16 plot( NULL , xlim=c(1,28) , ylim=c(0,1) , xlab="" ,
      ylab="proportion left lever" , xaxt="n" , yaxt="n" )
axis( 2 , at=c(0,0.5,1) , labels=c(0,0.5,1) )
abline( h=0.5 , lty=2 )
for ( j in 1:7 ) abline( v=(j-1)*4+4.5 , lwd=0.5 )
for ( j in 1:7 ) text( (j-1)*4+2.5 , 1.1 , concat("actor ",j) , xpd=TRUE )
for ( j in (1:7)[-2] ) {
  lines( (j-1)*4+c(1,3) , pl[j,c(1,3)] , lwd=2 , col=rangi2 )
  lines( (j-1)*4+c(2,4) , pl[j,c(2,4)] , lwd=2 , col=rangi2 )
}
points( 1:28 , t(pl) , pch=16 , col="white" , cex=1.7 )
points( 1:28 , t(pl) , pch=c(1,1,16,16) , col=rangi2 , lwd=2 )
yoff <- 0.01
text( 1 , pl[1,1]-yoff , "R/N" , pos=1 , cex=0.8 )
text( 2 , pl[1,2]+yoff , "L/N" , pos=3 , cex=0.8 )
text( 3 , pl[1,3]-yoff , "R/P" , pos=1 , cex=0.8 )
text( 4 , pl[1,4]+yoff , "L/P" , pos=3 , cex=0.8 )
mtext( "observed proportions\n" )
```

There are a lot of visual embellishments in this plot, so the code is longer than it really needs to be. It is just plotting the points in `pl` and then dressing them up. The open points are the non-partner treatments. The filled points are the partner treatments. Then the first point in each open/filled pair is prosocial on the right. The second is prosocial on the left. Each group of four point is an individual actor, labeled at the top.

The bottom plot in [FIGURE 11.4](#) shows the posterior predictions. We can compute these using `link`, just like you would with a quap model in earlier chapters:

```
R code
11.17 dat <- list( actor=rep(1:7,each=4) , treatment=rep(1:4,times=7) )
      p_post <- link( m11.4 , data=dat )
      p_mu <- apply( p_post , 2 , mean )
```

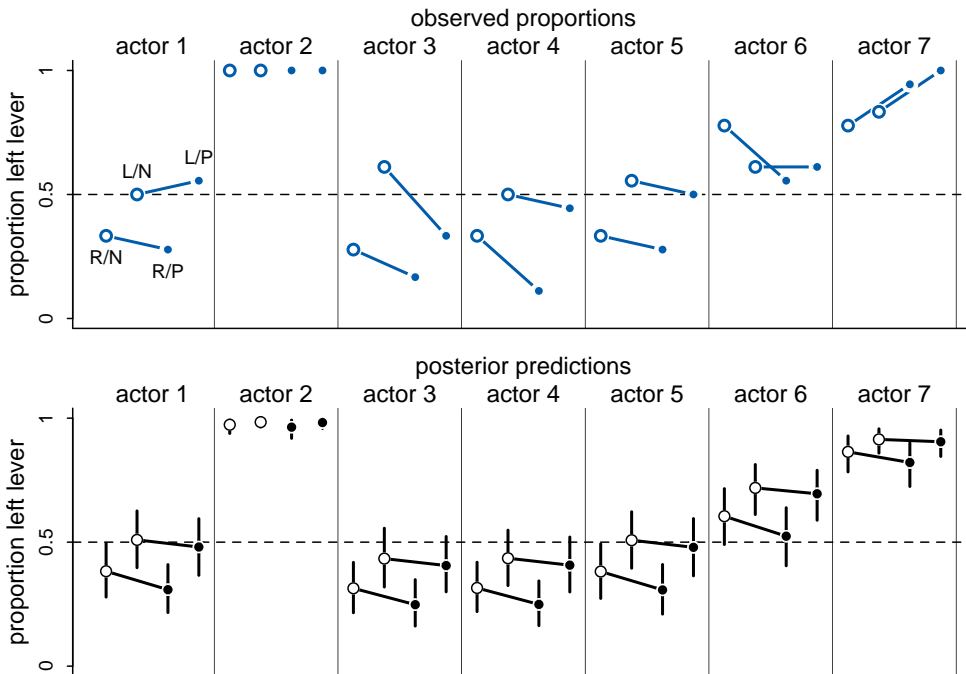


FIGURE 11.4. Observed data (top) and posterior predictions (bottom) for the chimpanzee data. Data are grouped by actor. Open points are no-partner treatments. Filled points are partner treatments. The right R and left L sides of the prosocial option are labeled in the top figure. Both left treatments and both right treatments are connected by a line segment, within each actor. The bottom plot shows 89% compatibility intervals for each proportion for each actor.

```
p_ci <- apply( p_post , 2 , PI )
```

The model expects almost no change when adding a partner. Most of the variation in predictions comes from the actor intercepts. Handedness seems to be the big story of this experiment.

The data themselves show additional variation—some of the actors possibly respond more to the treatments than others do. We might consider a model that allows each unique actor to have unique treatment parameters. But we'll leave such a model until we arrive at multilevel models, because we'll need some additional tricks to do the model well.

We haven't considered a model that splits into separate index variables the location of the prosocial option and the presence of a partner. Why not? Because the driving hypothesis of the experiment is that the prosocial option will be chosen more when the partner is present. That is an interaction effect—the effect of the prosocial option depends upon a partner being present. But we could build a model without the interaction and use PSIS or WAIC to compare it to `m11.4`. You can guess from the posterior distribution of `m11.4` what would happen: The simpler model will do just fine, because there doesn't seem to be any evidence of an interaction between location of the prosocial option and the presence of the partner.

To confirm this guess, here are the new index variables we need:

```
R code
11.18 d$side <- d$prosoc_left + 1 # right 1, left 2
      d$cond <- d$condition + 1 # no partner 1, partner 2
```

And now the model. Again, we add `log_lik=TRUE` to the call, so we can compare the two models with PSIS or WAIC.

```
R code
11.19 dat_list2 <- list(
      pulled_left = d$pulled_left,
      actor = d$actor,
      side = d$side,
      cond = d$cond )
m11.5 <- ulam(
      alist(
        pulled_left ~ dbinom( 1 , p ) ,
        logit(p) <- a[actor] + bs[side] + bc[cond] ,
        a[actor] ~ dnorm( 0 , 1.5 ) ,
        bs[side] ~ dnorm( 0 , 0.5 ) ,
        bc[cond] ~ dnorm( 0 , 0.5 )
      ) , data=dat_list2 , chains=4 , log_lik=TRUE )
```

Comparing the two models with PSIS:

```
R code
11.20 compare( m11.5 , m11.4 , func=PSIS )
```

	PSIS	SE	dPSIS	dSE	pPSIS	weight
m11.5	530.6	19.13	0.0	NA	7.6	0.68
m11.4	532.1	18.97	1.5	1.29	8.5	0.32

WAIC produces almost identical results. As we guessed, the model without the interaction is really no worse, in expected predictive accuracy, than the model with it. You should inspect the posterior distribution for `m11.5` to make sure you can relate its parameters to those of `m11.4`. They tell the same story.

Do note that model comparison here is for the sake of understanding how it works. We don't need the model comparison for inference in this example. The experiment and hypothesis tell us which model to use (`m11.4`). Then the posterior distribution is sufficient for inference.

---

**Overthinking: Adding log-probability calculations to a Stan model.** When we add `log_lik=TRUE` to an `ulam` model, we are adding a block of code to the Stan model that calculates for each observed outcome the log-probability. These calculations are returned as samples in the posterior—there will be one log-probability for each observation and each sample. So we end up with a matrix of log-probabilities that has a column for each observation and a row for each sample. You won't see this matrix by default in `precis` or `extract.samples`. You can extract it by telling `extract.samples` that `clean=FALSE`:

```
R code
11.21 post <- extract.samples( m11.4 , clean=FALSE )
      str(post)
```

```
List of 4
$ log_lik: num [1:2000, 1:504] -0.53 -0.381 -0.441 -0.475 -0.548 ...
$ a      : num [1:2000, 1:7] -0.3675 0.0123 -0.8544 -0.2473 -0.762 ...
$ b      : num [1:2000, 1:4] 0.00915 -0.78079 0.26441 -0.25036 0.44651 ...
$ lp__   : num [1:2000(1d)] -270 -273 -270 -268 -268 ...
```

The `log_lik` matrix at the top contains all of the log-probabilities needed to calculate WAIC and PSIS. You can see the code that produces them by calling `stancode(m11.4)`. Let's review each piece of the model, so you can relate it to the `ulam` formula. First, there is the data block, naming and defining the size of each observed variable:

```
data{
  int pulled_left[504];
  int treatment[504];
  int actor[504];
}
```

Next comes the parameters block, which does the same for unobserved variables:

```
parameters{
  vector[7] a;
  vector[4] b;
}
```

Now the model block, which calculates the log-posterior. The log-posterior is used in turn to compute the shape of the surface that the Hamiltonian simulations glide around on. Note that this block executes in order, from top to bottom. The values of `p` must be computed before they are used in `binomial( 1 , p )`. This is unlike BUGS or JAGS where the lines can be in any order.

```
model{
  vector[504] p;
  b ~ normal( 0 , 0.5 );
  a ~ normal( 0 , 1.5 );
  for ( i in 1:504 ) {
    p[i] = a[actor[i]] + b[treatment[i]];
    p[i] = inv_logit(p[i]);
  }
  pulled_left ~ binomial( 1 , p );
}
```

Finally, the reason we are here, the **GENERATED QUANTITIES** block. This is an optional block that lets us compute anything we'd like returned in the posterior. It executes only after a sample is accepted, so it doesn't slow down sampling much. This is unlike the model block, which is executed many times during each path needed to produce a sample.

```
generated quantities{
  vector[504] log_lik;
  vector[504] p;
  for ( i in 1:504 ) {
    p[i] = a[actor[i]] + b[treatment[i]];
    p[i] = inv_logit(p[i]);
  }
  for ( i in 1:504 ) log_lik[i] = binomial_lpmf( pulled_left[i] | 1 , p[i] );
}
```

The log-probabilities are stored in a vector of the same length as the number of observations—504 here. The linear model needs to be calculated again, because while the parameters are available in this block, any variables declared inside the model block, like `p`, are not. So we do all of that again. There is a trick for writing the `p` code only once, using another optional block called **TRANSFORMED PARAMETERS**, but let's not make things too complicated yet. Finally, we loop over the observations and calculate the binomial probability of each, conditional on the parameters. The helper functions PSIS and WAIC expect to see this `log_lik` matrix in the posterior samples. You can write a raw Stan model, include these calculations, and still use PSIS and WAIC as before. To run this model without

using `ulam`, you just need to put the Stan model code above into a character vector and then call `stan`:

R code  
11.22

```
m11.4_stan_code <- stancode(m11.4)
m11.4_stan <- stan( model_code=m11.4_stan_code , data=dat_list , chains=4 )
compare( m11.4_stan , m11.4 )
```

	WAIC	SE	dWAIC	dSE	pWAIC	weight
m11.4	531.6	18.87	0.0	NA	8.2	0.66
m11.4_stan	532.9	18.92	1.3	0.15	8.7	0.34

Warning message:  
In compare(m11.4\_stan, m11.4) : Not all model fits of same class.  
This is usually a bad idea, because it implies they were fit by different algorithms.  
Check yourself, before you wreck yourself.

They are the same model, as indicated by the identical (within sampling error) WAIC values. Note also the warning message. The `compare` function checks the types of the model objects. If there is more than one class, it carries on but with this warning. In this case, it is a false alarm—both models used the same algorithm. Model `m11.4` is of class `ulam`, which is just a wrapper for a `stanfit` class object. In general, it is a bad idea to compare models that approximate the posterior using different algorithms. Any difference could just be a product of the difference in algorithms. In the often quoted words of the American philosopher O'Shea Jackson, check yourself before you wreck yourself.

**11.1.2. Relative shark and absolute deer.** In the analysis above, I mostly focused on changes in predictions on the outcome scale—how much difference does the treatment make in the probability of pulling a lever? This view of posterior prediction focuses on **ABSOLUTE EFFECTS**, the difference a counter-factual change in a variable might make on an absolute scale of measurement, like the probability of an event.

It is more common to see logistic regressions interpreted through **RELATIVE EFFECTS**. Relative effects are proportional changes in the odds of an outcome. If we change a variable and say the odds of an outcome double, then we are discussing relative effects. You can calculate these **PROPORTIONAL ODDS** relative effect sizes by simply exponentiating the parameter of interest. For example, to calculate the proportional odds of switching from treatment 2 to treatment 4 (adding a partner):

R code  
11.23

```
post <- extract.samples(m11.4)
mean( exp(post$b[,4]-post$b[,2]) )
```

```
[1] 0.9206479
```

On average, the switch multiplies the odds of pulling the left lever by 0.92, an 8% reduction in odds. This is what is meant by proportional odds. The new odds are calculated by taking the old odds and multiplying them by the proportional odds, which is 0.92 in this example.

The risk of focusing on relative effects, such as proportional odds, is that they aren't enough to tell us whether a variable is important or not. If the other parameters in the model make the outcome very unlikely, then even a large proportional odds like 5.0 would not make the outcome frequent. Consider for example a rare disease which occurs in 1 per 10-million people. Suppose also that reading this textbook increased the odds of the disease 5-fold. That would mean approximately 4 more cases of the disease per 10-million people. So only 5-in-10-million chance now. The book is safe.

But we also shouldn't forget about relative effects. Relative effects are needed to make causal inferences, and they can be conditionally very important, when other baseline rates change. Consider for example the parable of relative shark and absolute deer. People are very afraid of sharks, but not so afraid of deer. But each year, deer kill many more people than sharks do. In this comparison, absolute risks are being compared: The lifetime risk of death from deer vastly exceeds the lifetime risk of death from shark bite.

However, this comparison is irrelevant in nearly all circumstances, because deer and sharks don't live in the same places. When you are in the water, you want to know instead the relative risk of dying from a shark attack. Conditional on being in the ocean, sharks are much more dangerous than deer. The relative shark risk is what we want to know, for those rare times when we are in the ocean.

Neither absolute nor relative risk is sufficient for all purposes. Relative risk can make a mostly irrelevant threat, like death from deer, seem deadly. For general advice, absolute risk often makes more sense. But to make general predictions, conditional on specific circumstances, we still need relative risk. Sharks are absolutely safe, while deer are relatively safe. Both are important truths.

---

**Overthinking: Proportional odds and relative risk.** Why does exponentiating a logistic regression coefficient compute the proportional odds? Consider the formula for the odds in a logistic regression:

$$p_i/(1 - p_i) = \exp(\alpha + \beta x_i)$$

The proportional odds of the event is the number we need to multiply the odds by when we increase  $x_i$  by 1 unit. Let  $q$  stand for the proportional odds. Then it is defined by:

$$q = \frac{\exp(\alpha + \beta(x_i + 1))}{\exp(\alpha + \beta x_i)} = \frac{\exp(\alpha) \exp(\beta x_i) \exp(\beta)}{\exp(\alpha) \exp(\beta x_i)} = \exp(\beta)$$

It's really that simple. So if  $q = 2$ , that means a unit increase in  $x_i$  generates a doubling of the odds. This a relative risk, because if the intercept  $\alpha$ , or any combination of other predictors, makes the event very unlikely or almost certain, then a doubling of the odds might not change the probability  $p_i$  much. Suppose for example that the odds are  $p_i/(1 - p_i) = 1/100$ . Doubling this to  $2/99$  moves  $p_i$  from approximately 0.01 to approximately 0.02. Similarly, if the odds are  $p_i/(1 - p_i) = 100/1$ , the doubling moves  $p_i$  from about 0.99 to 0.995.

---

**11.1.3. Aggregated binomial: Chimpanzees again, condensed.** In the chimpanzees data context, the models all calculated the likelihood of observing either zero or one pulls of the left-hand lever. The models did so, because the data were organized such that each row describes the outcome of a single pull. But in principle the same data could be organized differently. As long as we don't care about the order of the individual pulls, the same information is contained in a count of how many times each individual pulled the left-hand lever, for each combination of predictor variables.

For example, to calculate the number of times each chimpanzee pulled the left-hand lever, for each combination of predictor values:

```
data(chimpanzees)
d <- chimpanzees
d$treatment <- 1 + d$prosoc_left + 2*d$condition
d$side <- d$prosoc_left + 1 # right 1, left 2
d$cond <- d$condition + 1 # no partner 1, partner 2
```

R code  
11.24

```
d_aggregated <- aggregate(
  d$pulled_left ,
  list( treatment=d$treatment , actor=d$actor ,
        side=d$side , cond=d$cond ) ,
  sum )
colnames(d_aggregated)[5] <- "left_pulls"
```

Here are the results for the first two chimpanzees:

	treatment	actor	side	cond	left_pulls
1	1	1	1	1	6
2	1	2	1	1	18
3	1	3	1	1	5
4	1	4	1	1	6
5	1	5	1	1	6
6	1	6	1	1	14
7	1	7	1	1	14
8	2	1	2	1	9

The `left_pulls` column on the right is the count of times each actor pulled the left-hand lever for trials in each treatment. Recall that actor number 2 always pulled the left-hand lever. As a result, the counts for actor 2 are all 18—there were 18 trials for each animal for each treatment. Now we can get exactly the same inferences as before, just by defining the following model:

R code  
11.25

```
dat <- with( d_aggregated , list(
  left_pulls = left_pulls,
  treatment = treatment,
  actor = actor,
  side = side,
  cond = cond ) )

m11.6 <- ulam(
  alist(
    left_pulls ~ dbinom( 18 , p ) ,
    logit(p) <- a[actor] + b[treatment] ,
    a[actor] ~ dnorm( 0 , 1.5 ) ,
    b[treatment] ~ dnorm( 0 , 0.5 )
  ) , data=dat , chains=4 , log_lik=TRUE )
```

Take note of the 18 in the spot where a 1 used to be. Now there are 18 trials on each row, and the likelihood defines the probability of each count `left_pulls` out of 18 trials. Inspect the `precis` output. You'll see that the posterior distribution is the same as in model `m11.4`.

However, the PSIS (and WAIC) scores are very different between the 0/1 and aggregated models. Let's compare them:

R code  
11.26

```
compare( m11.6 , m11.4 , func=PSIS )
```

Some Pareto `k` values are high ( $>0.5$ ).

```
PSIS    SE dPSIS  dSE pPSIS weight
```



```

m11.6 113.5  8.41   0.0   NA   8.1     1
m11.4 532.1 18.97 418.6 9.44   8.5     0
Warning message:
In compare(m11.6, m11.4, func = PSIS) :
  Different numbers of observations found for at least two models.
Model comparison is valid only for models fit to exactly the same observations.
Number of observations for each model:
m11.6 28
m11.4 504

```

There's a lot of output here. But let's take it slowly, top to bottom. First, the PSIS summary table shows very different scores for the two models, even though they have the same posterior distribution. Why is this? The major reason is the aggregated model, `m11.6`, contains an extra factor in its log-probabilities, because of the way the data are organized. When calculating `dbinom(6,9,0.2)`, for example, the `dbinom` function contains a term for all the orders the 6 successes could appear in 9 trials. You've seen this term before:

$$\Pr(6|9, p) = \frac{6!}{6!(9-6)!} p^6 (1-p)^{9-6}$$

That ugly fraction in front is the multiplicity that was so important in the first half of the previous chapter. It just counts all the ways you could see 6 successes in 9 trials. When we instead split the 6 successes apart into 9 different 0/1 trials, like in a logistic regression, there is no multiplicity term to compute. So the joint probability of all 9 trials is instead:

$$\Pr(1, 1, 1, 1, 1, 1, 0, 0, 0|p) = p^6 (1-p)^{9-6}$$

This makes the aggregated probabilities larger—there are more ways to see the data. So the PSIS/WAIC scores end up being smaller. Go ahead and try it with the simple example here:

```

# deviance of aggregated 6-in-9
-2*dbinom(6,9,0.2,log=TRUE)
# deviance of dis-aggregated
-2*sum(dbern(c(1,1,1,1,1,1,0,0,0),0.2,log=TRUE))

```

R code  
11.27

```

[1] 11.79048
[1] 20.65212

```

But this difference is entirely meaningless. It is just a side effect of how we organized the data. The posterior distribution for the probability of success on each trial will end up the same, either way.

Continuing with the `compare` output, there are two warnings. The first is just to flag the fact that the two models have different numbers of observations. Never compare models fit to different sets of observations. The other warning is the Pareto  $k$  message at the top:

Some Pareto  $k$  values are high ( $>0.5$ ).

This is the Pareto  $k$  warning you met way back in [Chapter 7](#). The value in these warnings is more that they inform us about the presence of highly influential observations. Observations with these high Pareto  $k$  values are usually influential—the posterior changes a lot when they are dropped from the sample. As with the example from [Chapter 7](#), looking at individual points is very helpful for understanding why the model behaves as it does. And the penalty terms from WAIC contain similar information about relative influence of each observation.

Before looking at the Pareto  $k$  values, you might have noticed already that we didn't get a similar warning before in the disaggregated logistic models of the same data. Why not? Because when we aggregated the data by actor-treatment, we forced PSIS (and WAIC) to imagine cross-validation that leaves out all 18 observations in each actor-treatment combination. So instead of leave-one-out cross-validation, it is more like leave-eighteen-out. This makes some observations more influential, because they are really now 18 observations.

What's the bottom line? If you want to calculate WAIC or PSIS, you should use a logistic regression data format, not an aggregated format. Otherwise you are implicitly assuming that only large chunks of the data are separable. There are times when this makes sense, like with multilevel models. But it doesn't in most ordinary binomial regressions. If you dig into the Stan code that computes the individual log-likelihood terms, you can aggregate at any level you like, computing effect scores that are relevant to the level you want to predict at, whether that is 0/1 events or rather new individuals with many 0/1 events.

**11.1.4. Aggregated binomial: Graduate school admissions.** In the aggregated binomial example above, the number of trials was always 18 on every row. This is often not the case. The way to handle this is to insert a variable from the data in place of the "18". Let's work through an example. First, load the data:

R code  
11.28

```
library(rethinking)
data(UCBadmit)
d <- UCBadmit
```

This data table only has 12 rows, so let's look at the entire thing:

	dept	applicant.gender	admit	reject	applications
1	A	male	512	313	825
2	A	female	89	19	108
3	B	male	353	207	560
4	B	female	17	8	25
5	C	male	120	205	325
6	C	female	202	391	593
7	D	male	138	279	417
8	D	female	131	244	375
9	E	male	53	138	191
10	E	female	94	299	393
11	F	male	22	351	373
12	F	female	24	317	341

These are graduate school applications to 6 different academic departments at UC Berkeley.<sup>175</sup> The `admit` column indicates the number offered admission. The `reject` column indicates the opposite decision. The `applications` column is just the sum of `admit` and `reject`. Each application has a 0 or 1 outcome for admission, but since these outcomes have been aggregated by department and gender, there are only 12 rows. These 12 rows however represent 4526 applications, the sum of the `applications` column. So there is a lot of data here—counting the rows in the data table is no longer a sensible way to assess sample size. We could split these data apart into 0/1 Bernoulli trials, like in the original chimpanzees data. Then there would be 4526 rows in the data.

Our job is to evaluate whether these data contain evidence of gender bias in admissions. We will model the admission decisions, focusing on applicant gender as a predictor variable. So we want to fit a binomial regression that models `admit` as a function of each applicant's

gender. This will estimate the association between gender and probability of admission. This is what the model looks like, in mathematical form:

$$\begin{aligned} A_i &\sim \text{Binomial}(N_i, p_i) \\ \text{logit}(p_i) &= \alpha_{\text{GID}[i]} \\ \alpha_j &\sim \text{Normal}(0, 1.5) \end{aligned}$$

The variable  $N_i$  indicates `applications[i]`, the number of applications on row  $i$ . The index variable `GID[i]` indexes gender of an applicant. 1 indicates male, and 2 indicates female. We'll construct it just before fitting the model, like this:

```
dat_list <- list(
  admit = d$admit,
  applications = d$applications,
  gid = ifelse( d$applicant.gender=="male" , 1 , 2 )
)
m11.7 <- ulam(
  alist(
    admit ~ dbinom( applications , p ) ,
    logit(p) <- a[gid] ,
    a[gid] ~ dnorm( 0 , 1.5 )
  ) , data=dat_list , chains=4 )
precis( m11.7 , depth=2 )
```

R code  
11.29

```
      mean    sd  5.5% 94.5% n_eff Rhat
a[1] -0.22 0.04 -0.29 -0.16 1232    1
a[2] -0.83 0.05 -0.91 -0.75 1323    1
```

The posterior for male applicants, `a[1]`, is higher than that of female applicants. How much higher? We need to compute the contrast. Let's calculate the contrast on the logit scale (shark) as well as the contrast on the outcome scale (absolute deer):

```
post <- extract.samples(m11.7)
diff_a <- post$a[,1] - post$a[,2]
diff_p <- inv_logit(post$a[,1]) - inv_logit(post$a[,2])
precis( list( diff_a=diff_a , diff_p=diff_p ) )
```

R code  
11.30

```
'data.frame': 2000 obs. of 2 variables:
      mean    sd  5.5% 94.5% histogram
diff_a 0.61 0.07 0.50 0.71 
diff_p 0.14 0.01 0.12 0.16
```

The log-odds difference is certainly positive, corresponding to a higher probability of admission for male applicants. On the probability scale itself, the difference is somewhere between 12% and 16%.

Before moving on to speculate on the cause of the male advantage, let's plot posterior predictions for the model. We'll use the default posterior validation check function, `postcheck`, and then dress it up a little by adding lines to connect data points from the same department.

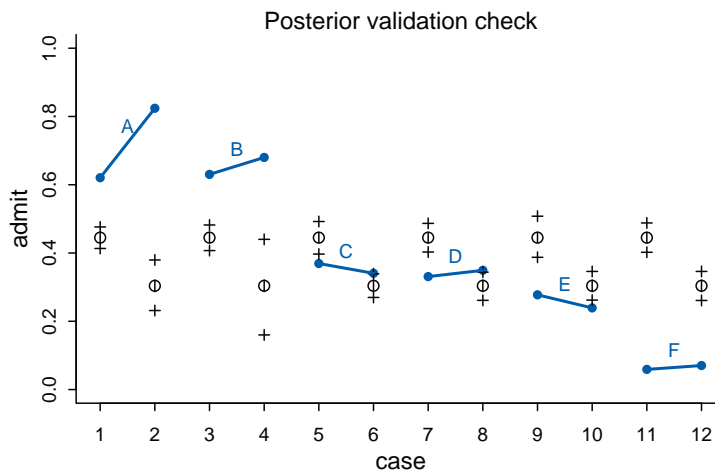


FIGURE 11.5. Posterior check for model `m11.7`. Blue points are observed proportions admitted for each row in the data, with points from the same department connected by a blue line. Open points, the tiny vertical black lines within them, and the crosses are expected proportions, 89% intervals of the expectation, and 89% interval of simulated samples, respectively.

R code  
11.31

```
postcheck( m11.7 )
# draw lines connecting points from same dept
for ( i in 1:6 ) {
  x <- 1 + 2*(i-1)
  y1 <- d$admit[x]/d$applications[x]
  y2 <- d$admit[x+1]/d$applications[x+1]
  lines( c(x,x+1) , c(y1,y2) , col=rangi2 , lwd=2 )
  text( x+0.5 , (y1+y2)/2 + 0.05 , d$dept[x] , cex=0.8 , col=rangi2 )
}
```

The result is shown as [FIGURE 11.5](#). Those are pretty terrible predictions. There are only two departments in which women had a lower rate of admission than men (C and E), and yet the model says that women should expect to have a 14% lower chance of admission.

Sometimes a fit this bad is the result of a coding mistake. In this case, it is not. The model did correctly answer the question we asked of it: *What are the average probabilities of admission for women and men, across all departments?* The problem in this case is that men and women did not apply to the same departments, and departments vary in their rates of admission. This makes the answer misleading. You can see the steady decline in admission probability for both men and women from department A to department F. Women in these data tended not to apply to departments like A and B, which had high overall admission rates. Instead they applied in large numbers to departments like F, which admitted less than 10% of applicants.

So while it is true overall that women had a lower probability of admission in these data, it is clearly not true within most departments. And note that just inspecting the posterior

distribution alone would never have revealed that fact to us. We had to appeal to something outside the fit model. In this case, it was a simple posterior validation check.

Instead of asking “What are the average probabilities of admission for women and men across all departments?” we want to ask “What is the average difference in probability of admission between women and men within departments?” In order to ask the second question, we estimate unique female and male admission rates in each department. Here’s a model that asks this new question:

$$\begin{aligned} A_i &\sim \text{Binomial}(N_i, p_i) \\ \text{logit}(p_i) &= \alpha_{\text{GID}[i]} + \delta_{\text{DEPT}[i]} \\ \alpha_j &\sim \text{Normal}(0, 1.5) \\ \delta_k &\sim \text{Normal}(0, 1.5) \end{aligned}$$

where DEPT indexes department in  $k = 1..6$ . So now each department  $k$  gets its own log-odds of admission,  $\delta_k$ , but the model still estimates universal adjustments, which are the same in all departments, for male and female applications.

Fitting this model is straightforward. We’ll use the indexing notation again to construct an intercept for each department. But first, we also need to construct a numerical index that numbers the departments 1 through 6. The function `coerce_index` can do this for us, using the `dept` factor as input. Here’s the code to construct the index and fit both models:

```
dat_list$dept_id <- rep(1:6,each=2)
m11.8 <- ulam(
  alist(
    admit ~ dbinom( applications , p ) ,
    logit(p) <- a[gid] + delta[dept_id] ,
    a[gid] ~ dnorm( 0 , 1.5 ) ,
    delta[dept_id] ~ dnorm( 0 , 1.5 )
  ) , data=dat_list , chains=4 , iter=4000 )
precis( m11.8 , depth=2 )
```

R code  
11.32


	mean	sd	5.5%	94.5%	n_eff	Rhat
a[1]	-0.54	0.52	-1.40	0.27	763	1
a[2]	-0.44	0.53	-1.29	0.38	768	1
delta[1]	1.12	0.53	0.31	1.98	772	1
delta[2]	1.08	0.53	0.25	1.94	782	1
delta[3]	-0.14	0.53	-0.97	0.72	767	1
delta[4]	-0.17	0.53	-0.99	0.69	767	1
delta[5]	-0.62	0.53	-1.44	0.25	789	1
delta[6]	-2.17	0.54	-3.03	-1.30	812	1

The intercept for male applicants, `a[1]`, is now a little smaller on average than the one for female applicants. Let’s calculate the contrasts against, both on relative (shark) and absolute (deer) scales:

```
post <- extract.samples(m11.8)
diff_a <- post$a[,1] - post$a[,2]
diff_p <- inv_logit(post$a[,1]) - inv_logit(post$a[,2])
precis( list( diff_a=diff_a , diff_p=diff_p ) )
```

R code  
11.33

```
'data.frame': 10000 obs. of 2 variables:
      mean   sd  5.5% 94.5%      histogram
diff_a -0.10 0.08 -0.22  0.03
diff_p -0.02 0.02 -0.05  0.01
```



If male applicants have it worse, it is only by a very small amount, about 2% on average.

Why did adding departments to the model change the inference about gender so much? The earlier figure gives you a hint—the rates of admission vary a lot across departments. Furthermore, women and men applied to different departments. Let's do a quick tabulation to show that:

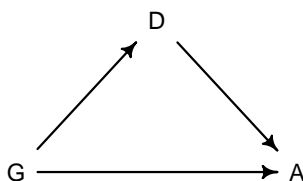
R code  
11.34

```
pg <- with( dat_list , sapply( 1:6 , function(k)
  applications[dept_id==k]/sum(applications[dept_id==k]) ) )
rownames(pg) <- c("male","female")
colnames(pg) <- unique(d$dept)
round( pg , 2 )
```

	A	B	C	D	E	F
male	0.88	0.96	0.35	0.53	0.33	0.52
female	0.12	0.04	0.65	0.47	0.67	0.48

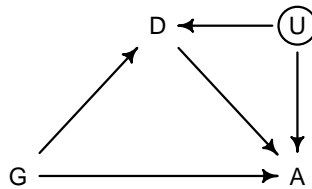
These are the proportions of all applications in each department that are from either men (top row) or women (bottom row). Department A receives 88% of its applications from men. Department E receives 33% from men. Now look back at the `delta` posterior means in the `precis` output from `m11.8`. The departments with a larger proportion of women applicants are also those with lower overall admissions rates.

Department is probably a confound, in the sense that it misleads us about the direct causal effect. But it is not a confound, in the sense that it is probably a genuine causal path through gender influences admission. Gender influences choice of department, and department influences chance of admission. Controlling for department reveals a more plausible direct causal influence of gender. In DAG form:



The variable  $G$  is gender,  $D$  is department, and  $A$  is acceptance. There is an indirect causal path  $G \rightarrow D \rightarrow A$  from gender to acceptance. So to infer the direct effect  $G \rightarrow A$ , we need to condition on  $D$  and close the indirect path. Model `m11.8` does that. If you inspect `postcheck(m11.8)`, you'll see that the model lines up much better now with the variation among departments. This is another example of a **MEDIATION** analysis.

Don't get too excited however that conditioning on department is sufficient to estimate the direct causal effect of gender on admissions. What if there are unobserved confounds influencing both department and admissions? Like this:



What could  $U$  be? How about academic ability. Ability could influence choice of department and probability of admission. In that case, conditioning on department is conditioning on a collider, and it opens a non-causal path between gender and admissions,  $G \rightarrow D \leftarrow U \rightarrow A$ . I'll ask you to explore some possibilities like this in the practice problems at the end.

As a final note, you might have noticed that model `m11.8` is over-parameterized. We don't actually need one of the parameters, either `a[1]` or `a[2]`. Why? Because the individual `delta` parameters can stand for the acceptance rate of one of the genders in each department. Then we just need an average deviation across departments. If this were a non-Bayesian model, it wouldn't work. But this kind of model is perfectly fine for us. The standard deviations are inflated, because there are many combinations of the `a` and `delta` parameters that can match the data. If you look at `pairs(m11.8)`, you'll see high posterior correlations among all of the parameters. But on the outcome scale, the predictions are much tighter, as you can see if you invoke `postcheck(m11.8)`. It's all good.

Why might we want to over-parameterize the model? Because it makes it easier to assign priors. If we made one of the genders baseline and measured the other as a deviation from it, we would stumble into the issue of assuming that the acceptance rate for one of the genders is pre-data more uncertain than the other. This isn't to say that over-parameterizing a model is always a good idea. But it isn't a violation of any statistical principle. You can always convert the posterior, post sampling, to any alternative parameterization. The only limitation is whether the algorithm we use to approximate the posterior can handle the high correlations. In this case, it can, and I bumped up the number of iterations to make sure.

**Rethinking: Simpson's paradox is not a paradox.** This empirical example is a famous one in statistical teaching. It is often used to illustrate a phenomenon known as [SIMPSON'S PARADOX](#).<sup>176</sup> Like most paradoxes, there is no violation of logic, just of intuition. And since different people have different intuition, Simpson's paradox means different things to different people. The poor intuition being violated in this case is that a positive association in the entire population should also hold within each department. Overall, females in these data did have a harder time getting admitted to graduate school. But that arose because females applied to the hardest departments for anyone, male or female, to gain admission to.

Perhaps a little more paradoxical is that this phenomenon can repeat itself indefinitely within a sample. Any association between an outcome and a predictor can be nullified or reversed when another predictor is added to the model. And the reversal can reveal a true causal influence or rather just be a confound, as occurred in the grandparents example in [Chapter 6](#). All that we can do about this is to remain skeptical of models and try to imagine ways they might be deceiving us. Thinking causally about these settings usually helps.<sup>177</sup>

## 11.2. Poisson regression

Binomial GLMs are appropriate when the outcome is a count from zero to some known upper bound. If you can analogize the data to the globe tossing model, then you should use a binomial GLM. But often the upper bound isn't known. Instead the counts never get

close to any upper limit. For example, if we go fishing and return with 17 fish, what was the theoretical maximum? Whatever it is, it isn't in our data. How do we model the fish counts?

It turns out that the binomial model works here, provided we squint at it the right way. When a binomial distribution has a very small probability of an event  $p$  and a very large number of trials  $N$ , then it takes on a special shape. The expected value of a binomial distribution is just  $Np$ , and its variance is  $Np(1 - p)$ . But when  $N$  is very large and  $p$  is very small, then these are approximately the same.

For example, suppose you own a monastery that is in the business, like many monasteries before the invention of the printing press, of copying manuscripts. You employ 1000 monks, and on any particular day about 1 of them finishes a manuscript. Since the monks are working independently of one another, and manuscripts vary in length, some days produce 3 or more manuscripts, and many days produce none. Since this is a binomial process, you can calculate the variance across days as  $Np(1 - p) = 1000(0.001)(1 - 0.001) \approx 1$ . You can simulate this, for example over 10,000 ( $1e5$ ) days:

R code  
11.35

```
y <- rbinom(1e5, 1000, 1/1000)
c( mean(y) , var(y) )
```

```
[1] 0.9968400 0.9928199
```

The mean and the variance are nearly identical. This is a special shape of the binomial. This special shape is known as the **POISSON DISTRIBUTION**, and it is useful because it allows us to model binomial events for which the number of trials  $N$  is unknown or uncountably large. Suppose for example that you come to own, through imperial drama, another monastery. You don't know how many monks toil within it, but your advisors tell you that it produces, on average, 2 manuscripts per day. With this information alone, you can infer the entire distribution of numbers of manuscripts completed each day.

To build models with a Poisson distribution, the model form is even simpler than it is for a binomial or Gaussian model. This simplicity arises from the Poisson's having only one parameter that describes its shape, resulting in a data probability definition like this:

$$y_i \sim \text{Poisson}(\lambda)$$

The parameter  $\lambda$  is the expected value of the outcome  $y$ . It is also the expected variance of the counts  $y$ .

We also need a link function. The conventional link function for a Poisson model is the log link, as introduced in the previous chapter ([page 318](#)). To embed a linear model, we use:

$$y_i \sim \text{Poisson}(\lambda_i)$$

$$\log(\lambda_i) = \alpha + \beta(x_i - \bar{x})$$

The log link ensures that  $\lambda_i$  is always positive, which is required of the expected value of a count outcome. But as mentioned in the previous chapter, it also implies an exponential relationship between predictors and the expected value. Exponential relationships grow very quickly, and few natural phenomena remain exponential for long. So one thing to always check with a log link is whether it makes sense at all ranges of the predictor variables. Priors on the log scale also scale in surprising ways. So prior predictive simulation is again helpful.

**11.2.1. Example: Oceanic tool complexity.** The island societies of Oceania provide a natural experiment in technological evolution. Different historical island populations possessed tool kits of different size. These kits include fish hooks, axes, boats, hand plows, and many





FIGURE 11.6. Locations of societies in the Kline data. The Equator and International Date Line are shown.

other types of tools. A number of theories predict that larger populations will both develop and sustain more complex tool kits. So the natural variation in population size induced by natural variation in island size in Oceania provides a natural experiment to test these ideas. It's also suggested that contact rates among populations effectively increase population size, as it's relevant to technological evolution. So variation in contact rates among Oceanic societies is also relevant.

We'll use this topic to develop a standard Poisson GLM analysis. And then I'll pivot at the end and also do a non-standard, but more theoretically motivated, Poisson model. The data we'll work with are counts of unique tool types for 10 historical Oceanic societies:<sup>178</sup>

```
library(rethinking)
data(Kline)
d <- Kline
d
```

R code  
11.36

	culture	population	contact	total_tools	mean_TU
1	Malekula	1100	low	13	3.2
2	Tikopia	1500	low	22	4.7
3	Santa Cruz	3600	low	24	4.0
4	Yap	4791	high	43	5.0
5	Lau Fiji	7400	high	33	5.0
6	Trobriand	8000	high	19	4.0
7	Chuuk	9200	high	40	3.8
8	Manus	13000	low	28	6.6
9	Tonga	17500	high	55	5.4
10	Hawaii	275000	low	71	6.6

That's the entire data set. You can see the location of these societies in the Pacific Ocean in FIGURE 11.6. Keep in mind that the number of rows is not clearly the same as the “sample size” in a count model. The relationship between parameters and “degrees of freedom” is not simple, outside of simple linear regressions. Still, there isn't a lot of data here, because there just aren't that many historic Oceanic societies for which reliable data can be gathered. We'll want to use regularization to damp down overfitting, as always. But as you'll see, a lot can still be learned from these data. Any rules you've been taught about minimum sample sizes for inference are just non-Bayesian superstitions. If you get the prior back, then the data aren't enough. It's that simple.

The `total_tools` variable will be the outcome variable. We'll model the idea that:

- (1) The number of tools increases with the log population size. Why log? Because that's what the theory says, that it is the order of magnitude of the population that matters, not the absolute size of it. So we'll look for a positive association between `total_tools` and `log population`. You can get some intuition for why a linear impact of population size can't be right by thinking about mechanism. We'll think about mechanism more at the end.
- (2) The number of tools increases with the contact rate among islands. No nation is an island, even when it is an island. Islands that are better networked may acquire or sustain more tool types.
- (3) The impact of population on tool counts is moderated by high contact. This is to say that the association between `total_tools` and `log population` depends upon contact. So we will look for a positive interaction between `log population` and `contact rate`.

Let's build now. First, we make some new columns with the standardized log of population and an index variable for contact:

R code  
11.37

```
d$P <- scale( log(d$population) )
d$contact_id <- ifelse( d$contact=="high" , 2 , 1 )
```

The model that conforms to the research hypothesis includes an interaction between log-population and contact rate. In math form, this is:

$$\begin{aligned}
 T_i &\sim \text{Poisson}(\lambda_i) \\
 \log \lambda_i &= \alpha_{\text{CID}[i]} + \beta_{\text{CID}[i]} \log P_i \\
 \alpha_j &\sim \text{to be determined} \\
 \beta_j &\sim \text{to be determined}
 \end{aligned}$$

where  $P$  is population and CID is `contact_id`.

We need to figure out some sensible priors. As with binomial models, the transformation of scale between the scale of the linear model and the count scale of the outcome means that something flat on the linear model scale will not be flat on the outcome scale. Let's consider for example just a model with an intercept and a vague  $\text{Normal}(0,10)$  prior on it:

$$\begin{aligned}
 T_i &\sim \text{Poisson}(\lambda_i) \\
 \log \lambda_i &= \alpha \\
 \alpha &\sim \text{Normal}(0, 10)
 \end{aligned}$$

What does this prior look like on the outcome scale,  $\lambda$ ? If  $\alpha$  has a normal distribution, then  $\lambda$  has a log-normal distribution. So let's plot a log-normal with these values for the (normal) mean and standard deviation:

R code  
11.38

```
curve( dlnorm( x , 0 , 10 ) , from=0 , to=100 , n=200 )
```

The distribution is shown in [FIGURE 11.7](#) as the black curve. I've used a range from 0 to 100 on the horizontal axis, reflecting the notion that we know all historical tool kits in the Pacific were in this range. For the  $\alpha \sim \text{Normal}(0, 10)$  prior, there is a huge spike right around zero—that means zero tools on average—and a very long tail. How long? Well the mean of

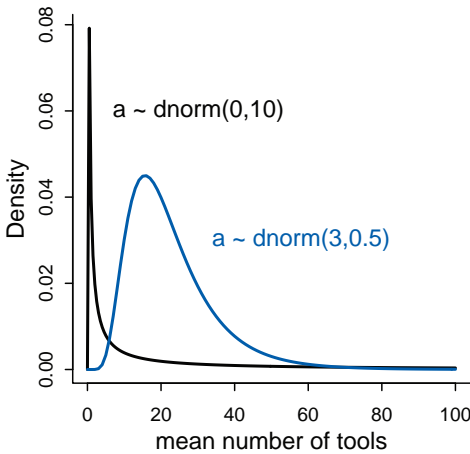


FIGURE 11.7. Prior predictive distribution of the mean  $\lambda$  of a simple Poisson GLM, considering only the intercept  $\alpha$ . A flat conventional prior (black) creates absurd expectations on the outcome scale. The mean of this distribution is  $\exp(50) \approx$  stupidly large. It is easy to do better by shifting prior mass above zero (blue).

a log-normal distribution is  $\exp(\mu + \sigma^2/2)$ , which evaluates to  $\exp(50)$ , which is impossibly large. If you doubt this, just simulate it:

```
a <- rnorm(1e4,0,10)
lambda <- exp(a)
mean( lambda )
```

R code  
11.39

```
[1] 9.622994e+12
```

That's a lot of tools, enough to cover an entire island. We can do better than this.

I encourage you to play around with the curve code above, trying different means and standard deviations. The fact to appreciate is that a log link puts half of the real numbers—the negative numbers—between 0 and 1 on the outcome scale. So if your prior puts half its mass below zero, then half the mass will end up between 0 and 1 on the outcome scale. For Poisson models, flat priors make no sense and can wreck Prague. Here's my weakly informative suggestion:

```
curve( dlnorm( x , 3 , 0.5 ) , from=0 , to=100 , n=200 )
```

R code  
11.40

I've displayed this distribution as well in [FIGURE 11.7](#), as the blue curve. The mean is now  $\exp(3 + 0.5^2/2) \approx 20$ . We haven't looked at the mean of the `total_tools` column, and we don't want to. This is supposed to be a prior. We want the prior predictive distribution to live in the plausible outcome space, not fit the sample.

Now we need a prior for  $\beta$ , the coefficient of log population. Again for dramatic effect, let's consider first a conventional flat prior like  $\beta \sim \text{Normal}(0, 10)$ . Conventional priors are even flatter. We'll simulate together with the intercept and plot 100 prior trends of standardized log population against total tools:

```
N <- 100
a <- rnorm( N , 3 , 0.5 )
b <- rnorm( N , 0 , 10 )
plot( NULL , xlim=c(-2,2) , ylim=c(0,100) )
```

R code  
11.41

```
for ( i in 1:N ) curve( exp( a[i] + b[i]*x ) , add=TRUE , col=grau() )
```

I display this prior predictive distribution as the top-left plot of [FIGURE 11.8](#). The pivoting around zero makes sense—that’s just the average log population. The values on the horizontal axis are z-scores, because the variable is standardized. So you can see that this prior thinks that the vast majority of prior relationships between log population and total tools embody either explosive growth just above the mean log population size or rather catastrophic decline right before the mean. This prior is terrible. Of course you will be able to confirm, once we start fitting models, that even 10 observations can overcome these terrible priors. But please remember that we are practicing for when it does matter. And in any particular application, it could matter.

So let’s try something much tighter. I’m tempted actually to force the prior for  $\beta$  to be positive. But I’ll resist that temptation and let the data prove that to you. Instead let’s just dampen the prior’s enthusiasm for impossibly explosive relationships. After some experimentation, I’ve settled on  $\beta \sim \text{Normal}(0, 0.2)$ :

R code  
11.42

```
set.seed(10)
N <- 100
a <- rnorm( N , 3 , 0.5 )
b <- rnorm( N , 0 , 0.2 )
plot( NULL , xlim=c(-2,2) , ylim=c(0,100) )
for ( i in 1:N ) curve( exp( a[i] + b[i]*x ) , add=TRUE , col=grau() )
```

This plot is displayed in the top-right of [FIGURE 11.8](#). Strong relationships are still possible, but most of the mass is for rather flat relationships between total tools and log population.

It will also help to view these priors on more natural outcome scales. The standardized log population variable is good for fitting. But it is bad for thinking. Population size has a natural zero, and we want to keep that in sight. Standardizing the variable destroys that. First, here are 100 prior predictive trends between total tools and un-standardized log population:

R code  
11.43

```
x_seq <- seq( from=log(100) , to=log(200000) , length.out=100 )
lambda <- sapply( x_seq , function(x) exp( a + b*x ) )
plot( NULL , xlim=range(x_seq) , ylim=c(0,500) , xlab="log population" ,
      ylab="total tools" )
for ( i in 1:N ) lines( x_seq , lambda[i,] , col=grau() , lwd=1.5 )
```

This plot appears in the bottom-left of [FIGURE 11.8](#). Notice that 100 total tools is probably the most we expect to ever see in these data. While most of the trends are in that range, some explosive options remain. And finally let’s also view these same curves on the natural population scale:

R code  
11.44

```
plot( NULL , xlim=range(exp(x_seq)) , ylim=c(0,500) , xlab="population" ,
      ylab="total tools" )
for ( i in 1:N ) lines( exp(x_seq) , lambda[i,] , col=grau() , lwd=1.5 )
```

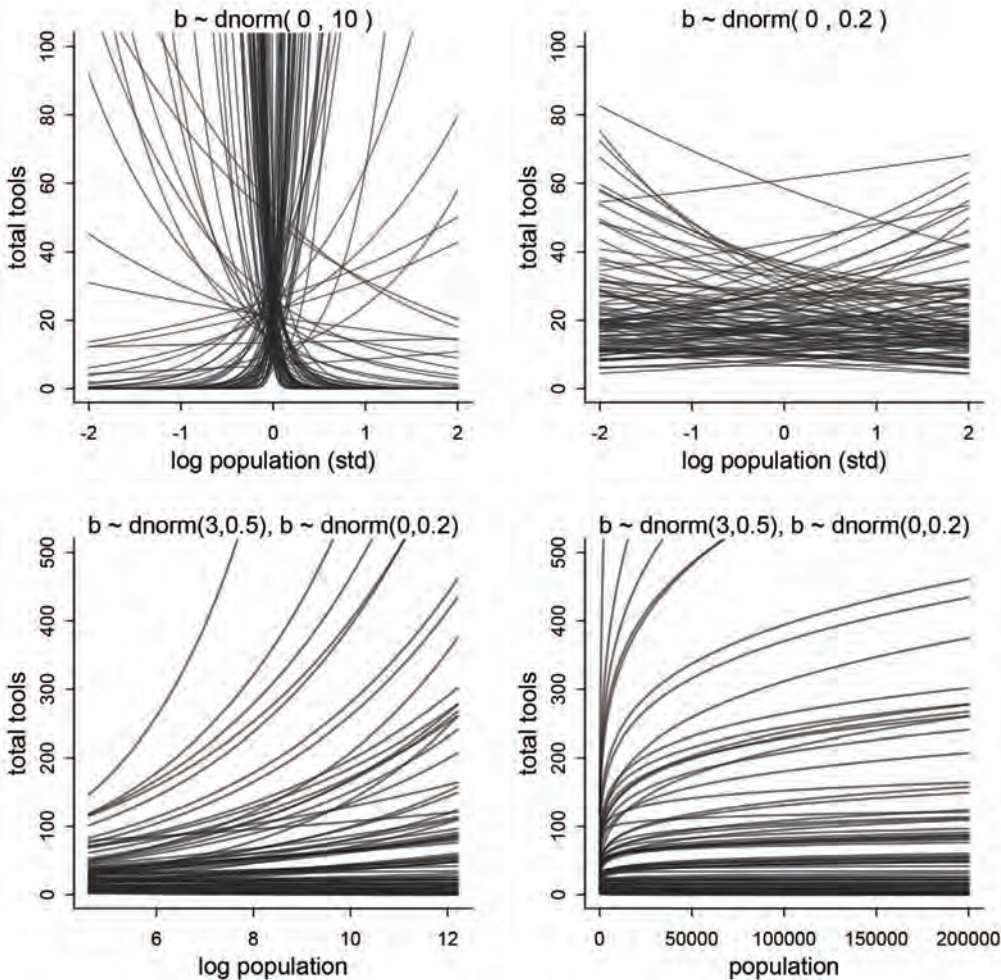


FIGURE 11.8. Struggling with slope priors in a Poisson GLM. Top-left: A flat prior produces explosive trends on the outcome scale. Top-right: A regularizing prior remains mostly within the space of outcomes. Bottom-left: Horizontal axis now on unstandardized scale. Bottom-right: Horizontal axis on natural scale (raw population size).

This plot lies in the bottom-right of [FIGURE 11.8](#). On the raw population scale, these curves bend the other direction. This is the natural consequence of putting the log of population inside the linear model. Poisson models with log links create **LOG-LINEAR** relationships with their predictor variables. When a predictor variable is itself logged, this means we are assuming diminishing returns for the raw variable. You can see this by comparing the two plots in the bottom of [FIGURE 11.8](#). The curves on the left would be linear if you log them. On the natural population scale, the model imposes diminishing returns on population: Each additional person contributes a smaller increase in the expected number of tools. The curves bend down and level off. Many predictor variables are better used as logarithms, for this reason. Simulating prior predictive distributions is a useful way to think through these issues.

Okay, finally we can approximate some posterior distributions. I’m going to code both the interaction model presented above as well as a very simple intercept-only model. The intercept-only model is here because I want to show you something interesting about Poisson models and how parameters relate to model complexity. Here’s the code for both models:

```
R code
11.45 dat <- list(
      T = d$total_tools ,
      P = d$P ,
      cid = d$contact_id )

# intercept only
m11.9 <- ulam(
  alist(
    T ~ dpois( lambda ),
    log(lambda) <- a,
    a ~ dnorm( 3 , 0.5 )
  ), data=dat , chains=4 , log_lik=TRUE )

# interaction model
m11.10 <- ulam(
  alist(
    T ~ dpois( lambda ),
    log(lambda) <- a[cid] + b[cid]*P,
    a[cid] ~ dnorm( 3 , 0.5 ),
    b[cid] ~ dnorm( 0 , 0.2 )
  ), data=dat , chains=4 , log_lik=TRUE )
```

Let’s look at the PSIS model comparison quickly, just to flag two important facts.

```
R code
11.46 compare( m11.9 , m11.10 , func=PSIS )
```

Some Pareto  $k$  values are high ( $>0.5$ ).

	PSIS	SE	dPSIS	dSE	pPSIS	weight
m11.10	84.6	13.24	0.0	NA	6.6	1
m11.9	141.8	33.78	57.2	33.68	8.5	0

First, note that we get the Pareto  $k$  warning again. This indicates some highly influential points. That shouldn’t be surprising—this is a small data set. But it means we’ll want to take a look at the posterior predictions with that in mind. Second, while it’s no surprise that the intercept-only model `m11.9` has a worse score than the interaction model `m11.10`, it might be very surprising that the “effective number of parameters” `pPSIS` is actually *larger* for the model with fewer parameters. Model `m11.9` has only one parameter. Model `m11.10` has four parameters. This isn’t some weird thing about PSIS—WAIC tells you the same story. What is going on here?

The only place that model complexity—a model’s tendency to overfit—and parameter count have a clear relationship is in a simple linear regression with flat priors. Once a distribution is bounded, for example, then parameter values near the boundary produce less overfitting than those far from the boundary. The same principle applies to data distributions. Any count near zero is harder to overfit. So overfitting risk depends both upon structural details of the model and the composition of the sample.

In this sample, a major source of overfitting risk is the highly influential point flagged by PSIS. Let's plot the posterior predictions now, and I'll scale and label the highly influential points with their Pareto  $k$  values. Here's the code to plot the data and superimpose posterior predictions for the expected number of tools at each population size and contact rate:

```
k <- PSIS( m11.10 , pointwise=TRUE )$k
plot( dat$P , dat$T , xlab="log population (std)" , ylab="total tools" ,
      col=rangi2 , pch=ifelse( dat$cid==1 , 1 , 16 ) , lwd=2 ,
      ylim=c(0,75) , cex=1+normalize(k) )

# set up the horizontal axis values to compute predictions at
ns <- 100
P_seq <- seq( from=-1.4 , to=3 , length.out=ns )

# predictions for cid=1 (low contact)
lambda <- link( m11.10 , data=data.frame( P=P_seq , cid=1 ) )
lmu <- apply( lambda , 2 , mean )
lci <- apply( lambda , 2 , PI )
lines( P_seq , lmu , lty=2 , lwd=1.5 )
shade( lci , P_seq , xpd=TRUE )

# predictions for cid=2 (high contact)
lambda <- link( m11.10 , data=data.frame( P=P_seq , cid=2 ) )
lmu <- apply( lambda , 2 , mean )
lci <- apply( lambda , 2 , PI )
lines( P_seq , lmu , lty=1 , lwd=1.5 )
shade( lci , P_seq , xpd=TRUE )
```

R code  
11.47

The result is shown in [FIGURE 11.9](#). Open points are low contact societies. Filled points are high contact societies. The points are scaled by their Pareto  $k$  values. The dashed curve is the low contact posterior mean. The solid curve is the high contact posterior mean.

This plot is joined on its right by the same predictions shown on the natural scale, with raw population sizes on the horizontal. The code to do that is very similar, but you need to convert the `P_seq` to the natural scale, by reversing the standardization, and then you can just replace `P_seq` with the converted sequence in the `lines` and `shade` commands.

```
plot( d$population , d$total_tools , xlab="population" , ylab="total tools" ,
      col=rangi2 , pch=ifelse( dat$cid==1 , 1 , 16 ) , lwd=2 ,
      ylim=c(0,75) , cex=1+normalize(k) )

ns <- 100
P_seq <- seq( from=-5 , to=3 , length.out=ns )
# 1.53 is sd of log(population)
# 9 is mean of log(population)
pop_seq <- exp( P_seq*1.53 + 9 )

lambda <- link( m11.10 , data=data.frame( P=P_seq , cid=1 ) )
lmu <- apply( lambda , 2 , mean )
lci <- apply( lambda , 2 , PI )
```

R code  
11.48



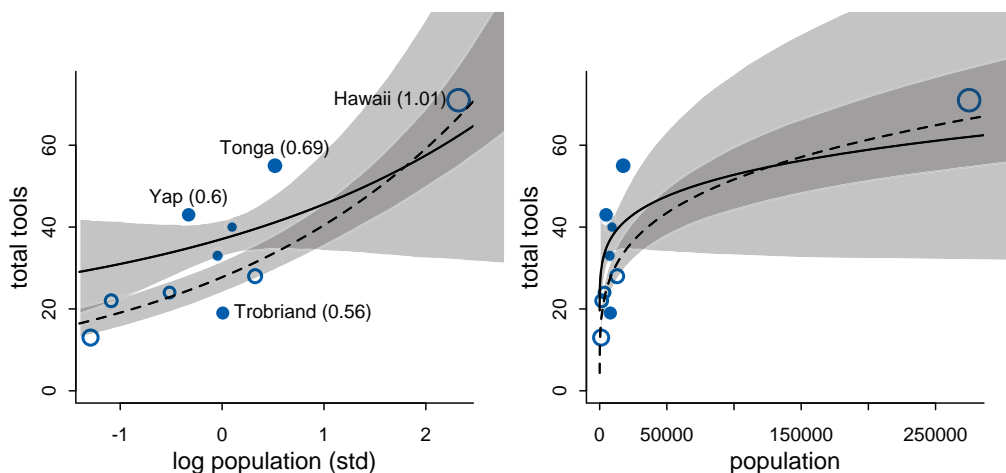


FIGURE 11.9. Posterior predictions for the Oceanic tools model. Filled points are societies with historically high contact. Open points are those with low contact. Point size is scaled by relative PSIS Pareto  $k$  values. Larger points are more influential. The solid curve is the posterior mean for high contact societies. The dashed curve is the same for low contact societies. 89% compatibility intervals are shown by the shaded regions. Left: Standardized log population scale, as in the model code. Right: Same predictions on the natural population scale.

```
lines( pop_seq , lmu , lty=2 , lwd=1.5 )
shade( lci , pop_seq , xpd=TRUE )

lambda <- link( m11.10 , data=data.frame( P=P_seq , cid=2 ) )
lmu <- apply( lambda , 2 , mean )
lci <- apply( lambda , 2 , PI )
lines( pop_seq , lmu , lty=1 , lwd=1.5 )
shade( lci , pop_seq , xpd=TRUE )
```

Hawaii ( $k = 1.01$ ), Tonga ( $k = 0.69$ ), Yap ( $k = 0.6$ ), and the Trobriand Islands ( $k = 0.56$ ) are highly influential points. Most are not too influential, but Hawaii is very influential. You can see why in the figure: It has extreme population size and the most tools. This is most obvious on the natural scale. This doesn't mean Hawaii is some "outlier" that should be dropped from the data. But it does mean that Hawaii strongly influences the posterior distribution. In the practice problems at the end of the chapter, I'll ask you to drop Hawaii and see what changes. For now, let's do something much more interesting.

Look at the posterior predictions in [FIGURE 11.9](#). Notice that the trend for societies with high contact (solid) is higher than the trend for societies with low contact (dashed) when population size is low, but then the model allows it to actually be smaller. The means cross one another at high population sizes. Of course the model is actually saying it has no idea where the trend for high contact societies goes at high population sizes, because there are no high population size societies with high contact. There is only low-contact Hawaii. But it is still a silly pattern that we know shouldn't happen. A counter-factual Hawaii with the same



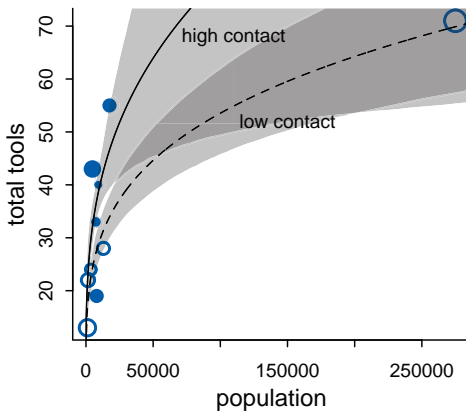


FIGURE 11.10. Posterior predictions for the scientific model of the Oceanic tool counts. Compare to the right-hand plot in [FIGURE 11.9](#). Since this model forces the trends to pass through the origin, as it must, its behavior is more sensible, in addition to having parameters with meaning outside a linear model.

population size but high contact should theoretically have at least as many tools as the real Hawaii. It shouldn't have fewer.

The model can produce this silly pattern, because it lets the intercept be a free parameter. Why is this bad? Because it means there is no guarantee that the trend for  $\lambda$  will pass through the origin where total tools equals zero and the population size equals zero. When there are zero people, there are also zero tools! As population increases, tools increase. So we get the intercept for free, if we stop and think.

Let's stop and think. Instead of the conventional GLM above, we could use the predictions of an actual model of the relationship between population size and tool kit complexity. By "actual model," I mean a model constructed specifically from scientific knowledge and hypothetical causal effects. The downside of this is that it will feel less like statistics—suddenly domain-specific skills are relevant. The upside is that it will feel more like science.

What we want is a dynamic model of the cultural evolution of tools. Tools aren't created all at once. Instead they develop over time. Innovation adds them to a population. Processes of loss remove them. The simplest model assumes that innovation is proportional to population size, but with diminishing returns. Each additional person adds less innovation than the previous. It also assumes that tool loss is proportional to the number of tools, without diminishing returns. These forces balance to produce a tool kit of some size.

The Overthinking box below presents the mathematical version of this model and shows you the code to build it in `ulam`. The model ends up in `m11.11`. Let's call this the *scientific model* and the previous `m11.10` the *geocentric model*. [FIGURE 11.10](#) shows the posterior predictions for the scientific model, on the natural scale of population size. Comparing it with the analogous plot in [FIGURE 11.9](#), notice that the trend for high contact societies always trends above the trend for low contact societies. Both trends always pass through the origin now, as they must. The scientific model is still far from perfect. But it provides a better foundation to learn from. The parameters have clearer meanings now. They aren't just bits of machinery in the bottom of a tide prediction engine.

You might ask how the scientific model compares to the geocentric model. The expected accuracy out of sample, whether you use PSIS or WAIC, is a few points better than the geocentric model. It is still tugged around by Hawaii and Tonga. We'll return to these data in a later chapter and approach contact rate a different way, by taking account of how close these societies are to one another.

---

**Overthinking: Modeling tool innovation.** Taking the verbal model in the main text above, we can write that the change in the expected number of tools in one time step is:

$$\Delta T = \alpha P^\beta - \gamma T$$

where  $P$  is the population size,  $T$  is the number of tools, and  $\alpha$ ,  $\beta$ , and  $\gamma$  are parameters to be estimated. To find an equilibrium number of tools  $T$ , just set  $\Delta T = 0$  and solve for  $T$ . This yields:

$$\hat{T} = \frac{\alpha P^\beta}{\gamma}$$

We're going to use this inside a Poisson model now. The noise around the outcome will still be Poisson, because that is still the maximum entropy distribution in this context—`total_tools` is a count with no clear upper bound. But the linear model is gone:

$$T_i \sim \text{Poisson}(\lambda_i)$$

$$\lambda_i = \alpha P_i^\beta / \gamma$$

Notice that there is no link function! All we have to do to ensure that  $\lambda$  remains positive is to make sure the parameters are positive. In the code below, I'll use exponential priors for  $\beta$  and  $\gamma$  and a log-Normal for  $\alpha$ . Then they all have to be positive. In building the model, we also want to allow some or all of the parameters to vary by contact rate. Since contact rate is supposed to mediate the influence of population size, let's allow  $\alpha$  and  $\beta$ . It could also influence  $\gamma$ , because trade networks might prevent tools from vanishing over time. But we'll leave that as an exercise for the reader. Here's the code:

R code  
11.49

```
dat2 <- list( T=d$total_tools, P=d$population, cid=d$contact_id )
m11.11 <- ulam(
  alist(
    T ~ dpois( lambda ),
    lambda <- exp(a[cid])*P^b[cid]/g,
    a[cid] ~ dnorm(1,1),
    b[cid] ~ dexp(1),
    g ~ dexp(1)
  ), data=dat2 , chains=4 , log_lik=TRUE )
```

I've invented the exact priors behind the scenes. Let's not get distracted with those. I encourage you to play around. The lesson here is in how we build in the predictor variables. Using prior simulations to design the priors is the same, although easier now that the parameters mean something. Finally, the code to produce posterior predictions is no different than the code in the main text used to plot predictions for `m11.10`.

---

**11.2.2. Negative binomial (gamma-Poisson) models.** Typically there is a lot of unexplained variation in Poisson models. Presumably this additional variation arises from unobserved influences that vary from case to case, generating variation in the true  $\lambda$ 's. Ignoring this variation, or *rate heterogeneity*, can cause confounds just like it can for binomial models. So a very common extension of Poisson GLMs is to swap the Poisson distribution for something called the **NEGATIVE BINOMIAL** distribution. This is really a Poisson distribution in disguise, and it is also sometimes called the **GAMMA-POISSON** distribution for this reason. It is a Poisson in disguise, because it is a mixture of different Poisson distributions. This is the Poisson analogue of the Student-t model, which is a mixture of different normal distributions. We'll work with mixtures in the next chapter.

**11.2.3. Example: Exposure and the offset.** The parameter  $\lambda$  is the expected value of a Poisson model, but it's also commonly thought of as a rate. Both interpretations are correct, and realizing this allows us to make Poisson models for which the **EXPOSURE** varies across cases  $i$ . Suppose for example that a neighboring monastery performs weekly totals of completed manuscripts while your monastery does daily totals. If you come into possession of both sets of records, how could you analyze both in the same model, given that the counts are aggregated over different amounts of time, different exposures?

Here's how. Implicitly,  $\lambda$  is equal to an expected number of events,  $\mu$ , per unit time or distance,  $\tau$ . This implies that  $\lambda = \mu/\tau$ , which lets us redefine the link:

$$y_i \sim \text{Poisson}(\lambda_i)$$

$$\log \lambda_i = \log \frac{\mu_i}{\tau_i} = \alpha + \beta x_i$$

Since the logarithm of a ratio is the same as a difference of logarithms, we can also write:

$$\log \lambda_i = \log \mu_i - \log \tau_i = \alpha + \beta x_i$$

These  $\tau$  values are the “exposures.” So if different observations  $i$  have different exposures, then this implies that the expected value on row  $i$  is given by:

$$\log \mu_i = \log \tau_i + \alpha + \beta x_i$$

When  $\tau_i = 1$ , then  $\log \tau_i = 0$  and we're back where we started. But when the exposure varies across cases, then  $\tau_i$  does the important work of correctly scaling the expected number of events for each case  $i$ . So you can model cases with different exposures just by writing a model like:

$$y_i \sim \text{Poisson}(\mu_i)$$

$$\log \mu_i = \log \tau_i + \alpha + \beta x_i$$

where  $\tau$  is a column in the data. So this is just like adding a predictor, the logarithm of the exposure, without adding a parameter for it. There will be an example later in this section. You can also put a parameter in front of  $\log \tau_i$ , which is one way to model the hypothesis that the rate is not constant with time.

For the last Poisson example, we'll look at a case where the exposure varies across observations. When the length of observation, area of sampling, or intensity of sampling varies, the counts we observe also naturally vary. Since a Poisson distribution assumes that the rate of events is constant in time (or space), it's easy to handle this. All we need to do, as explained above, is to add the logarithm of the exposure to the linear model. The term we add is typically called an *offset*.

We'll simulate for this example, both to provide another example of dummy-data simulation as well as to ensure we get the right answer from the offset approach. Suppose, as we did earlier, that you own a monastery. The data available to you about the rate at which manuscripts are completed is totaled up each day. Suppose the true rate is  $\lambda = 1.5$  manuscripts per day. We can simulate a month of daily counts:

```
num_days <- 30
y <- rpois( num_days , 1.5 )
```

R code  
11.50

So now  $y$  holds 30 days of simulated counts of completed manuscripts.

Also suppose that your monastery is turning a tidy profit, so you are considering purchasing another monastery. Before purchasing, you'd like to know how productive the new monastery might be. Unfortunately, the current owners don't keep daily records, so a head-to-head comparison of the daily totals isn't possible. Instead, the owners keep weekly totals. Suppose the daily rate at the new monastery is actually  $\lambda = 0.5$  manuscripts per day. To simulate data on a weekly basis, we just multiply this average by 7, the exposure:

```
R code
11.51 num_weeks <- 4
      y_new <- rpois( num_weeks , 0.5*7 )
```

And new `y_new` holds four weeks of counts of completed manuscripts.

To analyze both `y`, totaled up daily, and `y_new`, totaled up weekly, we just add the logarithm of the exposure to linear model. First, let's build a data frame to organize the counts and help you see the exposure for each case:

```
R code
11.52 y_all <- c( y , y_new )
      exposure <- c( rep(1,30) , rep(7,4) )
      monastery <- c( rep(0,30) , rep(1,4) )
      d <- data.frame( y=y_all , days=exposure , monastery=monastery )
```

Take a look at `d` and confirm that there are three columns: The observed counts are in `y`, the number of days each count was totaled over are in `days`, and the new monastery is indicated by `monastery`.

To fit the model, and estimate the rate of manuscript production at each monastery, we just compute the log of each exposure and then include that variable in linear model. This code will do the job:


```
R code
11.53 # compute the offset
      d$log_days <- log( d$days )

      # fit the model
      m11.12 <- quap(
        alist(
          y ~ dpois( lambda ),
          log(lambda) <- log_days + a + b*monastery,
          a ~ dnorm( 0 , 1 ),
          b ~ dnorm( 0 , 1 )
        ), data=d )
```

To compute the posterior distributions of  $\lambda$  in each monastery, we sample from the posterior and then just use the linear model, but without the offset now. We don't use the offset again, when computing predictions, because the parameters are already on the daily scale, for both monasteries.

```
R code
11.54 post <- extract.samples( m11.12 )
      lambda_old <- exp( post$a )
      lambda_new <- exp( post$a + post$b )
      precis( data.frame( lambda_old , lambda_new ) )
```

```
'data.frame': 10000 obs. of 2 variables:
      mean   sd 5.5% 94.5%      histogram
lambda_old 1.34 0.21 1.03  1.70
lambda_new  0.52 0.14 0.33  0.77
```



The new monastery produces about half as many manuscripts per day. So you aren't going to pay that much for it.

### 11.3. Multinomial and categorical models

The binomial distribution is relevant when there are only two things that can happen, and we count those things. In general, more than two things can happen. For example, recall the bag of marbles from way back in [Chapter 2](#). It contained only blue and white marbles. But suppose we introduce red marbles as well. Now each draw from the bag can be one of three categories, and the count that accumulates is across all three categories. So we end up with a count of blue, white, and red marbles.

When more than two types of unordered events are possible, and the probability of each type of event is constant across trials, then the maximum entropy distribution is the **MULTINOMIAL DISTRIBUTION**. You already met the multinomial, implicitly, in [Chapter 10](#) when we tossed pebbles into buckets as an introduction to maximum entropy. The binomial is really a special case of this distribution. And so its distribution formula resembles the binomial, just extrapolated out to three or more types of events. If there are  $K$  types of events with probabilities  $p_1, \dots, p_K$ , then the probability of observing  $y_1, \dots, y_K$  events of each type out of  $n$  total trials is:

$$\Pr(y_1, \dots, y_K | n, p_1, \dots, p_K) = \frac{n!}{\prod_i y_i!} \prod_{i=1}^K p_i^{y_i}$$

The fraction with  $n!$  on top just expresses the number of different orderings that give the same counts  $y_1, \dots, y_K$ . It's the famous multiplicity from the previous chapter.

A model built on a multinomial distribution may also be called a **CATEGORICAL** regression, usually when each event is isolated on a single row, like with logistic regression. In machine learning, this model type is sometimes known as the **MAXIMUM ENTROPY CLASSIFIER**. Building a generalized linear model from a multinomial likelihood is complicated, because as the event types multiply, so too do your modeling choices. And there are two different approaches to constructing the likelihoods, as well. The first is based directly on the multinomial likelihood and uses a generalization of the logit link. I'll show you an example of this approach, which I'll call the *explicit* approach. The second approach transforms the multinomial likelihood into a series of Poisson likelihoods, oddly enough. I'll introduce that approach after I introduce Poisson GLMs.

The conventional and natural link in this context is the **MULTINOMIAL LOGIT**, also known as the **SOFTMAX** function. This link function takes a vector of *scores*, one for each of  $K$  event types, and computes the probability of a particular type of event  $k$  as:

$$\Pr(k | s_1, s_2, \dots, s_K) = \frac{\exp(s_k)}{\sum_{i=1}^K \exp(s_i)}$$

The rethinking package provides this link as the `softmax` function. Combined with this conventional link, this type of GLM may be called **MULTINOMIAL LOGISTIC REGRESSION**.

The biggest issue is what to do with the multiple linear models. In a binomial GLM, you can pick either of the two possible events and build a single linear model for its log odds.

The other event is handled automatically. But in a multinomial (or categorical) GLM, you need  $K - 1$  linear models for  $K$  types of events. One of the outcome values is chosen as a “pivot” and the others are modeled relative to it. In each of the  $K - 1$  linear models, you can use any predictors and parameters you like—they don’t have to be the same, and there are often good reasons for them to be different. In the special case of two types of events, none of these choices arise, because there is only one linear model. And that’s why the binomial GLM is so much easier.

There are two basic cases: (1) predictors have different values for different values of the outcome, and (2) parameters are distinct for each value of the outcome. The first case is useful when each type of event has its own quantitative *traits*, and you want to estimate the association between those traits and the probability each type of event appears in the data. The second case is useful when you are interested instead in features of some entity that produces each event, whatever type it turns out to be. Let’s consider each case separately and talk through an empirically motivated example of each. You can mix both cases in the same model. But it’ll be easier to grasp the distinction in pure examples of each.

I’m going to build the models in this section with pure Stan code. We could make the models with `quap` or `ulam`. But using Stan directly will provide some additional clarity about the data structures needed to manage multiple, simultaneous linear models. It will also make it easier for you to modify these models for your purposes, including adding varying effects and other gizmos later on.<sup>179</sup>

**11.3.1. Predictors matched to outcomes.** For example, suppose you are modeling choice of career for a number of young adults. One of the relevant predictor variables is expected income. In that case, the same parameter  $\beta_{\text{INCOME}}$  appears in each linear model, in order to estimate the impact of the income trait on the probability a career is chosen. But a different income value multiplies the parameter in each linear model.

Here’s a simulated example in R code. This code simulates career choice from three different careers, each with its own income trait. These traits are used to assign a *score* to each type of event. Then when the model is fit to the data, one of these scores is held constant, and the other two scores are estimated, using the known income traits. It is a little confusing. Step through the implementation, and it’ll make more sense. First, we simulate career choices:

R code  
11.55

```
# simulate career choices among 500 individuals
N <- 500                # number of individuals
income <- c(1,2,5)      # expected income of each career
score <- 0.5*income      # scores for each career, based on income
# next line converts scores to probabilities
p <- softmax(score[1],score[2],score[3])

# now simulate choice
# outcome career holds event type values, not counts
career <- rep(NA,N)     # empty vector of choices for each individual
# sample chosen career for each individual
set.seed(34302)
for ( i in 1:N ) career[i] <- sample( 1:3 , size=1 , prob=p )
```

To fit the model to these fake data, we use the `dcategorical` likelihood, which is the multinomial logistic regression distribution. It works when each value in the outcome variable,

here `career`, contains the individual event types on each row. To convert all the scores to probabilities, we'll use the multinomial logit link, which is called `softmax`. Then each possible career gets its own linear model with its own features. There are no intercepts in the simulation above. But if income doesn't predict career choice, you still want an intercept to account for differences in frequency. Here's the code:

```
code_m11.13 <- "
data{
  int N; // number of individuals
  int K; // number of possible careers
  int career[N]; // outcome
  vector[K] career_income;
}
parameters{
  vector[K-1] a; // intercepts
  real<lower=0> b; // association of income with choice
}
model{
  vector[K] p;
  vector[K] s;
  a ~ normal( 0 , 1 );
  b ~ normal( 0 , 0.5 );
  s[1] = a[1] + b*career_income[1];
  s[2] = a[2] + b*career_income[2];
  s[3] = 0; // pivot
  p = softmax( s );
  career ~ categorical( p );
}
"
```

R code  
11.56

Then we set up the data list and invoke `stan`:

```
dat_list <- list( N=N , K=3 , career=career , career_income=income )
m11.13 <- stan( model_code=code_m11.13 , data=dat_list , chains=4 )
precis( m11.13 , 2 )
```

R code  
11.57

	mean	sd	5.5%	94.5%	n_eff	Rhat
a[1]	0.06	0.21	-0.31	0.37	423	1
a[2]	-0.49	0.38	-1.19	0.04	435	1
b	0.27	0.19	0.02	0.61	460	1

You might have gotten some divergent transitions above. Can you figure out why?

Be aware that the estimates you get from these models are extraordinarily difficult to interpret. Since the parameters are relative to the pivot outcome value, they could end up positive or negative, depending upon the context. In the example above, I chose the last outcome type, the third career. If you choose another, you'll get different estimates, but the same predictions on the outcome scale. It really is a tide prediction engine. So you absolutely must convert them to a vector of probabilities to make much sense of them. However, in this case, it's clear that the coefficient on career income `b` is positive. It's just not clear at all how big the effect is.

To conduct a counterfactual simulation, we can extract the samples and make our own. The goal is to compare a counterfactual career in which the income is changed. How much does the probability change, in the presence of these competing careers? This is a subtle kind of question, because the probability change always depends upon the other choices. So let's imagine doubling the income of career 2 above:

R code  
11.58


```
post <- extract.samples( m11.13 )

# set up logit scores
s1 <- with( post , a[,1] + b*income[1] )
s2_orig <- with( post , a[,2] + b*income[2] )
s2_new <- with( post , a[,2] + b*income[2]*2 )

# compute probabilities for original and counterfactual
p_orig <- sapply( 1:length(post$b) , function(i)
  softmax( c(s1[i],s2_orig[i],0) ) )
p_new <- sapply( 1:length(post$b) , function(i)
  softmax( c(s1[i],s2_new[i],0) ) )

# summarize
p_diff <- p_new[2,] - p_orig[2,]
precis( p_diff )
```

'data.frame': 4000 obs. of 1 variables:

	mean	sd	5.5%	94.5%	histogram
p_diff	0.13	0.09	0.01	0.29	

So on average a 13% increase in probability of choosing the career, when the income is doubled. Note that value is conditional on comparing to the other careers in the calculation. These models do not produce predictions independent of a specific set of options. That's not a bug. That's just how choice works.

**11.3.2. Predictors matched to observations.** Now consider an example in which each observed outcome has unique predictor values. Suppose you are still modeling career choice. But now you want to estimate the association between each person's family income and which career they choose. So the predictor variable must have the same value in each linear model, for each row in the data. But now there is a unique parameter multiplying it in each linear model. This provides an estimate of the impact of family income on choice, for each type of career.

R code  
11.59

```
N <- 500
# simulate family incomes for each individual
family_income <- runif(N)
# assign a unique coefficient for each type of event
b <- c(-2,0,2)
career <- rep(NA,N) # empty vector of choices for each individual
for ( i in 1:N ) {
  score <- 0.5*(1:3) + b*family_income[i]
  p <- softmax(score[1],score[2],score[3])
  career[i] <- sample( 1:3 , size=1 , prob=p )
}
```



```

}

code_m11.14 <- "
data{
  int N; // number of observations
  int K; // number of outcome values
  int career[N]; // outcome
  real family_income[N];
}
parameters{
  vector[K-1] a; // intercepts
  vector[K-1] b; // coefficients on family income
}
model{
  vector[K] p;
  vector[K] s;
  a ~ normal(0,1.5);
  b ~ normal(0,1);
  for ( i in 1:N ) {
    for ( j in 1:(K-1) ) s[j] = a[j] + b[j]*family_income[i];
    s[K] = 0; // the pivot
    p = softmax( s );
    career[i] ~ categorical( p );
  }
}
"

dat_list <- list( N=N , K=3 , career=career , family_income=family_income )
m11.14 <- stan( model_code=code_m11.14 , data=dat_list , chains=4 )
precis( m11.14 , 2 )

```

	mean	sd	5.5%	94.5%	n_eff	Rhat
a[1]	-1.41	0.28	-1.88	-0.97	2263	1
a[2]	-0.64	0.20	-0.96	-0.33	2163	1
b[1]	-2.72	0.60	-3.69	-1.79	2128	1
b[2]	-1.72	0.39	-2.32	-1.10	2183	1

Again, computing implied predictions is the safest way to interpret these models. They do a great job of classifying discrete, unordered events. But the parameters are on a scale that is very hard to interpret. In this case,  $b[2]$  ended up negative, because it is relative to the pivot, for which family income has a positive effect. If you produce posterior predictions on the probability scale, you'll see this.

**11.3.3. Multinomial in disguise as Poisson.** Another way to fit a multinomial/categorical model is to refactor it into a series of Poisson likelihoods.<sup>180</sup> That should sound a bit crazy. But it's actually both principled and commonplace to model multinomial outcomes this way. It's principled, because the mathematics justifies it. And it's commonplace, because it is usually computationally easier to use Poisson rather than multinomial likelihoods. Here I'll give an example of an implementation. For the mathematical details of the transformation, see the Overthinking box at the end.

I appreciate that this kind of thing—modeling the same data different ways but getting the same inferences—is exactly the kind of thing that makes statistics maddening for scientists. So I’ll begin by taking a binomial example from earlier in the chapter and doing it over as a Poisson regression. Since the binomial is just a special case of the multinomial, the approach extrapolates to any number of event types. Think again of the UC Berkeley admissions data. Let’s load it again:

```
R code
11.60 library(rethinking)
      data(UCBadmit)
      d <- UCBadmit
```

Now let’s use a Poisson regression to model both the rate of admission and the rate of rejection. And we’ll compare the inference to the binomial model’s probability of admission. Here are both the binomial and Poisson models:

```
R code
11.61 # binomial model of overall admission probability
      m_binom <- quap(
        alist(
          admit ~ dbinom(applications,p),
          logit(p) <- a,
          a ~ dnorm( 0 , 1.5 )
        ), data=d )

      # Poisson model of overall admission rate and rejection rate
      # 'reject' is a reserved word in Stan, cannot use as variable name
      dat <- list( admit=d$admit , rej=d$reject )
      m_pois <- ulam(
        alist(
          admit ~ dpois(lambda1),
          rej ~ dpois(lambda2),
          log(lambda1) <- a1,
          log(lambda2) <- a2,
          c(a1,a2) ~ dnorm(0,1.5)
        ), data=dat , chains=3 , cores=3 )
```

Let’s consider just the posterior means, for the sake of simplicity. But keep in mind that the entire posterior is what matters. First, the inferred binomial *probability* of admission, across the entire data set, is:

```
R code
11.62 inv_logit(coef(m_binom))
```

```

      a
0.3877596
```

And in the Poisson model, the implied probability of admission is given by:

$$p_{\text{ADMIT}} = \frac{\lambda_1}{\lambda_1 + \lambda_2} = \frac{\exp(a_1)}{\exp(a_1) + \exp(a_2)}$$

In code form:

```
k <- coef(m_pois)
a1 <- k['a1']; a2 <- k['a2']
exp(a1)/(exp(a1)+exp(a2))
```

```
[1] 0.3872366
```

That's the same inference as in the binomial model. These days, you can just as easily use a categorical distribution, as in the previous section. But sometimes this Poisson factorization is easier. And you might encounter it elsewhere. So it's good to know that it's not insane.

**Overthinking: Multinomial-Poisson transformation.** The Poisson distribution was introduced earlier in this chapter. The Poisson probability of  $y_1$  events of type 1, assuming a rate  $\lambda_1$ , is given by:

$$\Pr(y_1|\lambda_1) = \frac{e^{-\lambda_1} \lambda_1^{y_1}}{y_1!}$$

I'll show you a magic trick for extracting this expression from the multinomial probability expression. The multinomial probability is just an extrapolation of the binomial to more than two types of events. So we'll work here with the binomial distribution, but in multinomial form, just to make the derivation a little easier. The probability of counts  $y_1$  and  $y_2$  for event types 1 and 2 with probabilities  $p_1$  and  $p_2$ , respectively, out of  $n$  trials, is:

$$\Pr(y_1, y_2|n, p_1, p_2) = \frac{n!}{y_1! y_2!} p_1^{y_1} p_2^{y_2}$$

We need some definitions now. Let  $\Lambda = \lambda_1 + \lambda_2$ ,  $p_1 = \lambda_1/\Lambda$ , and  $p_2 = \lambda_2/\Lambda$ . Substituting these into the binomial probability:

$$\Pr(y_1, y_2|n, \lambda_1, \lambda_2) = \frac{n!}{y_1! y_2!} \left(\frac{\lambda_1}{\Lambda}\right)^{y_1} \left(\frac{\lambda_2}{\Lambda}\right)^{y_2} = \frac{n!}{\Lambda^{y_1} \Lambda^{y_2}} \frac{\lambda_1^{y_1}}{y_1!} \frac{\lambda_2^{y_2}}{y_2!} = \frac{n!}{\Lambda^n} \frac{\lambda_1^{y_1}}{y_1!} \frac{\lambda_2^{y_2}}{y_2!}$$

Now we simultaneously multiply and divide by both  $e^{-\lambda_1}$  and  $e^{-\lambda_2}$ , then perform some strategic rearrangement:

$$\begin{aligned} \Pr(y_1, y_2|n, \lambda_1, \lambda_2) &= \frac{n!}{\Lambda^n} \frac{e^{-\lambda_1}}{e^{-\lambda_1}} \frac{\lambda_1^{y_1}}{y_1!} \frac{e^{-\lambda_2}}{e^{-\lambda_2}} \frac{\lambda_2^{y_2}}{y_2!} = \frac{n!}{\Lambda^n e^{-\lambda_1} e^{-\lambda_2}} \frac{e^{-\lambda_1} \lambda_1^{y_1}}{y_1!} \frac{e^{-\lambda_2} \lambda_2^{y_2}}{y_2!} \\ &= \frac{n!}{\underbrace{e^{-\Lambda} \Lambda^n}_{\Pr(n)^{-1}}} \underbrace{\frac{e^{-\lambda_1} \lambda_1^{y_1}}{y_1!}}_{\Pr(y_1)} \underbrace{\frac{e^{-\lambda_2} \lambda_2^{y_2}}{y_2!}}_{\Pr(y_2)} \end{aligned}$$

The final expression is the product of the Poisson probabilities  $\Pr(y_1)$  and  $\Pr(y_2)$ , divided by the Poisson probability of  $n$ ,  $\Pr(n)$ . It makes sense that the product is divided by  $\Pr(n)$ , because this is a conditional probability for  $y_1$  and  $y_2$ . All of this means that if there are  $k$  event types, you can model multinomial probabilities  $p_1, \dots, p_k$  using Poisson rate parameters  $\lambda_1, \dots, \lambda_k$ . And you can recover the multinomial probabilities using the definition  $p_i = \lambda_i / \sum_j \lambda_j$ .

## 11.4. Summary

This chapter described some of the most common generalized linear models, those used to model counts. It is important to never convert counts to proportions before analysis, because doing so destroys information about sample size. A fundamental difficulty with these models is that parameters are on a different scale, typically log-odds (for binomial) or log-rate (for Poisson), than the outcome variable they describe. Therefore computing implied predictions is even more important than before.

## 11.5. Practice

Problems are labeled Easy (E), Medium (M), and Hard (H).

**11E1.** If an event has probability 0.35, what are the log-odds of this event?

**11E2.** If an event has log-odds 3.2, what is the probability of this event?

**11E3.** Suppose that a coefficient in a logistic regression has value 1.7. What does this imply about the proportional change in odds of the outcome?

**11E4.** Why do Poisson regressions sometimes require the use of an *offset*? Provide an example.

**11M1.** As explained in the chapter, binomial data can be organized in aggregated and disaggregated forms, without any impact on inference. But the likelihood of the data does change when the data are converted between the two formats. Can you explain why?

**11M2.** If a coefficient in a Poisson regression has value 1.7, what does this imply about the change in the outcome?

**11M3.** Explain why the logit link is appropriate for a binomial generalized linear model.

**11M4.** Explain why the log link is appropriate for a Poisson generalized linear model.

**11M5.** What would it imply to use a logit link for the mean of a Poisson generalized linear model? Can you think of a real research problem for which this would make sense?

**11M6.** State the constraints for which the binomial and Poisson distributions have maximum entropy. Are the constraints different at all for binomial and Poisson? Why or why not?

**11M7.** Use `quap` to construct a quadratic approximate posterior distribution for the chimpanzee model that includes a unique intercept for each actor, `m11.4` (page 330). Compare the quadratic approximation to the posterior distribution produced instead from MCMC. Can you explain both the differences and the similarities between the approximate and the MCMC distributions? Relax the prior on the actor intercepts to `Normal(0,10)`. Re-estimate the posterior using both `ulam` and `quap`. Do the differences increase or decrease? Why?

**11M8.** Revisit the `data(Kline)` islands example. This time drop Hawaii from the sample and refit the models. What changes do you observe?

**11H1.** Use WAIC or PSIS to compare the chimpanzee model that includes a unique intercept for each actor, `m11.4` (page 330), to the simpler models fit in the same section. Interpret the results.

**11H2.** The data contained in `library(MASS); data(eagles)` are records of salmon pirating attempts by Bald Eagles in Washington State. See `?eagles` for details. While one eagle feeds, sometimes another will swoop in and try to steal the salmon from it. Call the feeding eagle the “victim” and the thief the “pirate.” Use the available data to build a binomial GLM of successful pirating attempts.

(a) Consider the following model:

$$\begin{aligned} y_i &\sim \text{Binomial}(n_i, p_i) \\ \text{logit}(p_i) &= \alpha + \beta_P P_i + \beta_V V_i + \beta_A A_i \\ \alpha &\sim \text{Normal}(0, 1.5) \\ \beta_P, \beta_V, \beta_A &\sim \text{Normal}(0, 0.5) \end{aligned}$$

where  $y$  is the number of successful attempts,  $n$  is the total number of attempts,  $P$  is a dummy variable indicating whether or not the pirate had large body size,  $V$  is a dummy variable indicating whether or not the victim had large body size, and finally  $A$  is a dummy variable indicating whether or not

the pirate was an adult. Fit the model above to the eagles data, using both `quap` and `ulam`. Is the quadratic approximation okay?

(b) Now interpret the estimates. If the quadratic approximation turned out okay, then it's okay to use the `quap` estimates. Otherwise stick to `ulam` estimates. Then plot the posterior predictions. Compute and display both (1) the predicted **probability** of success and its 89% interval for each row (*i*) in the data, as well as (2) the predicted success **count** and its 89% interval. What different information does each type of posterior prediction provide?

(c) Now try to improve the model. Consider an interaction between the pirate's size and age (immature or adult). Compare this model to the previous one, using WAIC. Interpret.

**11H3.** The data contained in `data(salamanders)` are counts of salamanders (*Plethodon elongatus*) from 47 different 49-m<sup>2</sup> plots in northern California.<sup>181</sup> The column `SALAMAN` is the count in each plot, and the columns `PCTCOVER` and `FORESTAGE` are percent of ground cover and age of trees in the plot, respectively. You will model `SALAMAN` as a Poisson variable.

(a) Model the relationship between density and percent cover, using a log-link (same as the example in the book and lecture). Use weakly informative priors of your choosing. Check the quadratic approximation again, by comparing `quap` to `ulam`. Then plot the expected counts and their 89% interval against percent cover. In which ways does the model do a good job? A bad job?

(b) Can you improve the model by using the other predictor, `FORESTAGE`? Try any models you think useful. Can you explain why `FORESTAGE` helps or does not help with prediction?

**11H4.** The data in `data(NWOGrants)` are outcomes for scientific funding applications for the Netherlands Organization for Scientific Research (NWO) from 2010–2012 (see van der Lee and Ellemers (2015) for data and context). These data have a very similar structure to the `UCBAdmit` data discussed in the chapter. I want you to consider a similar question: What are the total and indirect causal effects of gender on grant awards? Consider a mediation path (a pipe) through `discipline`. Draw the corresponding DAG and then use one or more binomial GLMs to answer the question. What is your causal interpretation? If NWO's goal is to equalize rates of funding between men and women, what type of intervention would be most effective?

**11H5.** Suppose that the NWO Grants sample has an unobserved confound that influences both choice of discipline and the probability of an award. One example of such a confound could be the career stage of each applicant. Suppose that in some disciplines, junior scholars apply for most of the grants. In other disciplines, scholars from all career stages compete. As a result, career stage influences discipline as well as the probability of being awarded a grant. Add these influences to your DAG from the previous problem. What happens now when you condition on discipline? Does it provide an un-confounded estimate of the direct path from gender to an award? Why or why not? Justify your answer with the backdoor criterion. If you have trouble thinking this through, try simulating fake data, assuming your DAG is true. Then analyze it using the model from the previous problem. What do you conclude? Is it possible for gender to have a real direct causal influence but for a regression conditioning on both gender and discipline to suggest zero influence?

**11H6.** The data in `data(Primates301)` are 301 primate species and associated measures. In this problem, you will consider how brain size is associated with social learning. There are three parts.

(a) Model the number of observations of `social_learning` for each species as a function of the log brain size. Use a Poisson distribution for the `social_learning` outcome variable. Interpret the resulting posterior. (b) Some species are studied much more than others. So the number of reported instances of `social_learning` could be a product of research effort. Use the `research_effort` variable, specifically its logarithm, as an additional predictor variable. Interpret the coefficient for log `research_effort`. How does this model differ from the previous one? (c) Draw a DAG to represent how you think the variables `social_learning`, `brain`, and `research_effort` interact. Justify the DAG with the measured associations in the two models above (and any other models you used).