Assignment5

Yung-Tang Chou

2019/5/16

Here is the rmarkdown file for the Multivariate Analysis Assignment 5.

< Question A >

The data is about salmon pirating attempts by Bald Eagles. Before we start let’s import some libraries needed.

library(rstan)

## Loading required package: ggplot2

## Loading required package: StanHeaders

## rstan (Version 2.18.2, GitRev: 2e1f913d3ca3)

## For execution on a local, multicore CPU with excess RAM we recommend calling  
## options(mc.cores = parallel::detectCores()).  
## To avoid recompilation of unchanged Stan programs, we recommend calling  
## rstan\_options(auto\_write = TRUE)

## For improved execution time, we recommend calling  
## Sys.setenv(LOCAL\_CPPFLAGS = '-march=native')  
## although this causes Stan to throw an error on a few processors.

library(rethinking)

## Loading required package: parallel

## rethinking (Version 1.88)

library(MASS)  
library(loo)

## This is loo version 2.1.0.  
## \*\*NOTE: As of version 2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' argument or set options(mc.cores = NUM\_CORES) for an entire session. Visit mc-stan.org/loo/news for details on other changes.

## \*\*NOTE for Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see https://github.com/stan-dev/loo/issues/94).

##   
## Attaching package: 'loo'

## The following object is masked from 'package:rethinking':  
##   
## compare

## The following object is masked from 'package:rstan':  
##   
## loo

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following object is masked from 'package:MASS':  
##   
## select

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(tidyr)

##   
## Attaching package: 'tidyr'

## The following object is masked from 'package:rstan':  
##   
## extract

library(skimr)

##   
## Attaching package: 'skimr'

## The following object is masked from 'package:stats':  
##   
## filter

The dataset itself contains categorical values in columns V, P, and A. Before building the model, let’s transform them into dummy variables.

data("eagles")  
d <- eagles; rm(eagles)  
d$V <- as.integer(ifelse(d$V == 'L', 1, 0))  
d$P <- as.integer(ifelse(d$P == 'L', 1, 0))  
d$A <- as.integer(ifelse(d$A == 'A', 1, 0))

A basic summary statistics is shown below.

## Skim summary statistics  
## n obs: 8   
## n variables: 5   
##   
## -- Variable type:integer --------------------------------------------------------------------------------------------------  
## variable missing complete n mean sd p0 p25 p50 p75 p100 hist  
## A 0 8 8 0.5 0.53 0 0 0.5 1 1 ▇▁▁▁▁▁▁▇  
## n 0 8 8 20 8.83 4 15 22 27.25 29 ▂▁▂▂▁▂▂▇  
## P 0 8 8 0.5 0.53 0 0 0.5 1 1 ▇▁▁▁▁▁▁▇  
## V 0 8 8 0.5 0.53 0 0 0.5 1 1 ▇▁▁▁▁▁▁▇  
## y 0 8 8 12.5 10.66 0 1 16 17.75 29 ▇▁▁▁▇▂▁▂

Let’s start by building the stan model. In this model we want to predict the relationship between successful pirating attemps and its body size, the victim’s body size, and adulthood. Since the success rate is between zero and one, we form a logistic model to compress the output ranging from zero to one.

The model is shown below.

model.01.code <- "  
data{  
 int n[8];  
int y[8];  
int V[8];  
int A[8];  
int P[8];  
}  
parameters{  
real a;  
real bP;  
real bA;  
real bV;  
}  
model{  
vector[8] p;  
bV ~ normal( 0 , 5 );  
bA ~ normal( 0 , 5 );  
bP ~ normal( 0 , 5 );  
a ~ normal( 0 , 10 );  
for ( i in 1:8 ) {  
p[i] = a + bP \* P[i] + bA \* A[i] + bV \* V[i];  
p[i] = inv\_logit(p[i]);  
}  
y ~ binomial( n , p );  
}  
generated quantities{  
vector[8] log\_lik;  
vector[8] p;  
for ( i in 1:8 ) {  
p[i] = a + bP \* P[i] + bA \* A[i] + bV \* V[i];  
p[i] = inv\_logit(p[i]);  
}  
for ( i in 1:8 ) log\_lik[i] = binomial\_lpmf( y[i] | n[i] , p[i] );  
}"

Note that within the generated quantities block, the log\_lik is also calculated.

Now we are going to run the stan model. Before doing so, to prevent stan model from crashing, we form a specific dataset for the stan model.

dat <- list(  
 N = NROW(d),  
 y = d$y,  
 n = d$n,  
 P = d$P,  
 A = d$A,  
 V = d$V)

For the stan model, we will use “dat” generated above as the input data, and use four markov chains to do the modeling. The model code is shown below.

model.01 <- stan(model\_code = model.01.code, data = dat, chains = 4)

##   
## SAMPLING FOR MODEL '85fba7139a15c5f5f0d368e239d14553' NOW (CHAIN 1).  
## Chain 1:   
## Chain 1: Gradient evaluation took 0 seconds  
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 1: Adjust your expectations accordingly!  
## Chain 1:   
## Chain 1:   
## Chain 1: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 1: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 1: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 1: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 1: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 1: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 1: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 1: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 1: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 1: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 1: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 1: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 1:   
## Chain 1: Elapsed Time: 0.075 seconds (Warm-up)  
## Chain 1: 0.074 seconds (Sampling)  
## Chain 1: 0.149 seconds (Total)  
## Chain 1:   
##   
## SAMPLING FOR MODEL '85fba7139a15c5f5f0d368e239d14553' NOW (CHAIN 2).  
## Chain 2:   
## Chain 2: Gradient evaluation took 0 seconds  
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 2: Adjust your expectations accordingly!  
## Chain 2:   
## Chain 2:   
## Chain 2: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 2: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 2: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 2: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 2: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 2: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 2: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 2: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 2: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 2: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 2: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 2: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 2:   
## Chain 2: Elapsed Time: 0.07 seconds (Warm-up)  
## Chain 2: 0.073 seconds (Sampling)  
## Chain 2: 0.143 seconds (Total)  
## Chain 2:   
##   
## SAMPLING FOR MODEL '85fba7139a15c5f5f0d368e239d14553' NOW (CHAIN 3).  
## Chain 3:   
## Chain 3: Gradient evaluation took 0 seconds  
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 3: Adjust your expectations accordingly!  
## Chain 3:   
## Chain 3:   
## Chain 3: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 3: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 3: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 3: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 3: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 3: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 3: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 3: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 3: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 3: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 3: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 3: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 3:   
## Chain 3: Elapsed Time: 0.069 seconds (Warm-up)  
## Chain 3: 0.068 seconds (Sampling)  
## Chain 3: 0.137 seconds (Total)  
## Chain 3:   
##   
## SAMPLING FOR MODEL '85fba7139a15c5f5f0d368e239d14553' NOW (CHAIN 4).  
## Chain 4:   
## Chain 4: Gradient evaluation took 0 seconds  
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 4: Adjust your expectations accordingly!  
## Chain 4:   
## Chain 4:   
## Chain 4: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 4: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 4: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 4: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 4: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 4: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 4: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 4: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 4: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 4: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 4: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 4: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 4:   
## Chain 4: Elapsed Time: 0.071 seconds (Warm-up)  
## Chain 4: 0.064 seconds (Sampling)  
## Chain 4: 0.135 seconds (Total)  
## Chain 4:

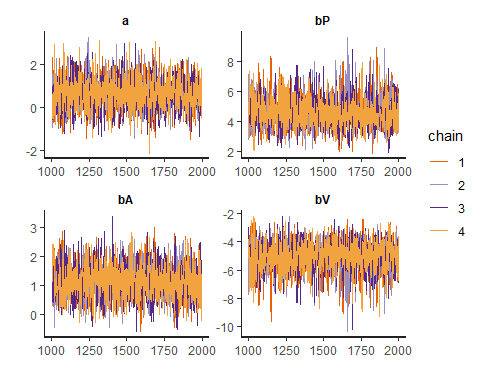
The descriptions of the model is shown below, using print statement.

print(model.01, include = F, pars = "log\_lik", probs = c(.1, .5, .9))

## Inference for Stan model: 85fba7139a15c5f5f0d368e239d14553.  
## 4 chains, each with iter=2000; warmup=1000; thin=1;   
## post-warmup draws per chain=1000, total post-warmup draws=4000.  
##   
## mean se\_mean sd 10% 50% 90% n\_eff Rhat  
## a 0.63 0.02 0.71 -0.29 0.62 1.53 2089 1  
## bP 4.68 0.03 1.01 3.49 4.58 6.00 1443 1  
## bA 1.15 0.01 0.56 0.42 1.14 1.87 2064 1  
## bV -5.07 0.03 1.07 -6.45 -4.97 -3.77 1424 1  
## p[1] 0.79 0.00 0.08 0.69 0.80 0.88 2874 1  
## p[2] 1.00 0.00 0.00 0.99 1.00 1.00 1509 1  
## p[3] 0.56 0.00 0.09 0.44 0.56 0.67 3486 1  
## p[4] 0.99 0.00 0.01 0.98 0.99 1.00 1889 1  
## p[5] 0.05 0.00 0.05 0.01 0.04 0.11 2093 1  
## p[6] 0.84 0.00 0.08 0.74 0.85 0.93 3192 1  
## p[7] 0.02 0.00 0.02 0.00 0.01 0.04 1605 1  
## p[8] 0.64 0.00 0.15 0.43 0.65 0.82 2009 1  
## lp\_\_ -48.21 0.04 1.49 -50.25 -47.88 -46.63 1435 1  
##   
## Samples were drawn using NUTS(diag\_e) at Thu May 16 14:16:22 2019.  
## For each parameter, n\_eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor on split chains (at   
## convergence, Rhat=1).

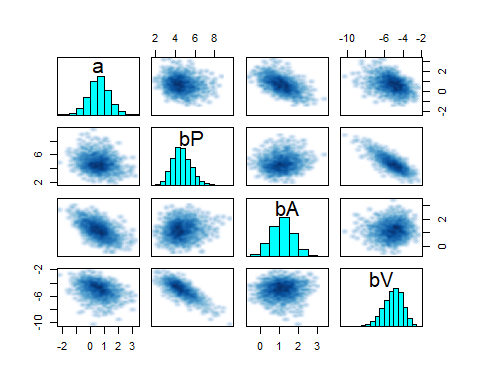
In order to identify whether the stan model is decent, we use traceplot to see if all the params lie in a specific range. The code is as below.

traceplot(model.01, pars = c('a','bP','bA','bV'))



Besides, let’s see the distribution of the model params using pairplot. The code is as below.

pairs(model.01, pars = c('a','bP','bA','bV'))



From the plots above, we can tell the following things. 1. The stan model converges properly. The traceplot shows that all the values lie in a specific range, which is the desired outcome. 2. The pairplot shows that the distribution of param bP is right-skewed, while the distribution of param bV is left-skewed.

Now let’s do a comparison with quadratic approximation, using quap function in the rethinking package. The code for the quadratic model is shown below.

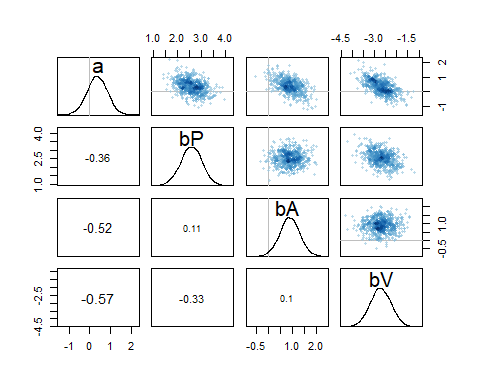
model.01.quap <- quap(  
 alist(  
 y ~ dbinom(n, p),  
 logit(p) <- a + bP \* P + bA \* A + bV \* V,  
 a ~ dnorm(0, 2),  
 bP ~ dnorm(0, 1),  
 bA ~ dnorm(0, 1),  
 bV ~ dnorm(0, 1)  
 ),  
 data = d  
)

precis(model.01.quap)

## mean sd 5.5% 94.5%  
## a 0.3671246 0.4909311 -0.4174781 1.151727  
## bP 2.5761871 0.4385449 1.8753076 3.277066  
## bA 0.8849650 0.4089999 0.2313041 1.538626  
## bV -2.7201825 0.4744372 -3.4784248 -1.961940

The pairplot of the quadratic model is shown below.

pairs(model.01.quap)



Comparing the two model, we can find out something interesting. 1. Amost all the means of the parameters in the quadratic model is larger, as is the standard deviation. 2. From the pairplot for quadratic approximation we can see that almost all the distribution is normal distributed, which is really different from what we’ve seen in the stan model.

We can tell from the stan model that the distribution may not be Gaussian. If we use quadratic approximation to fit the data, we don’t allow tong-tail distribution, which might in turn produces misleading distribution. In a nutshell, using stan model is a better idea since it allows distribution to skew, granting more flexibility rather than inposing a Gaussian distribution to a somehow perhaps not-Gaussian distributed dataset.

< Question B >

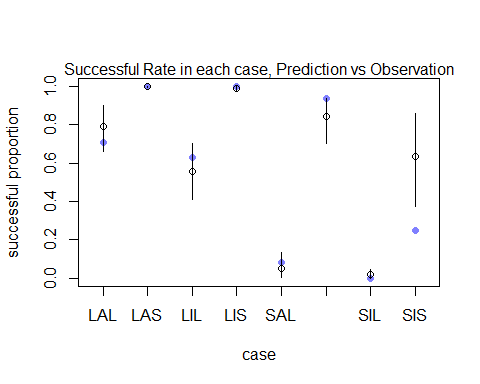
Now we are going to interpret the predictions. First we will see the predicted probability of the sucess rate and its 89% interval; then we will look at the predicted success count.

For success rate, we can directly grab the dataframe from the stan model. Then, calculate the mean and PI with apply functions. The code is shown below.

prob <- as.data.frame(model.01)[,13:20]  
prob.mean <- apply(prob, 2, mean)  
prob.PI <- apply(prob, 2, PI, prob=.89)

Let’s plot out to see the predictions.

d$success.proportion <- d$y / d$n  
plot(d$success.proportion, col=rangi2, xaxt = 'n', ylab='successful proportion', xlab='case',  
 xlim=c(0.75,8.25), ylim=c(0,1), pch=16)  
axis(1, at=1:8, labels = c('LAL','LAS','LIL','LIS','SAL','SAS','SIL','SIS'))  
points(1:8, prob.mean)  
for (i in 1:8) lines(c(i,i), prob.PI[,i])  
mtext('Successful Rate in each case, Prediction vs Observation')



The blue dots in the plot show the succesful rate calculated directly from the data. The hollow dots demonstrate the predicted mean of the successful rate. The line is where the percentile intervals are landed.

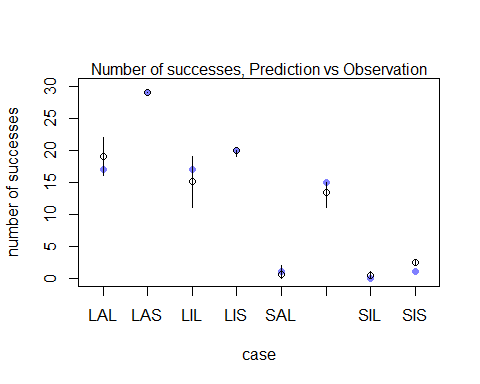
From the plot we can see almost all the cases follow our prediction, excep for case “SIS”. We can draw some conclusions: 1. Body size plays an important role in this prediction. If the size of the pirating bird is larger than that of the victim, the success rate is approximately 100%. If the size pirating bird is smaller than that of the victim, the success rate will drop tremendously. 2. The sample for the case “body size of both pirating bird and victim bird are small, in the meantime being immature” is small, therefore hard to predict. This might be the reason why the real data differs from the predicted data.

Then let’s move on to predict the successful attempts. We first calculate the successful attempts using the predictions generated from stan model along with the attempts in the original dataset. The code is as below.

prob <- as.data.frame(model.01)[,13:20]  
suc <- as.data.frame(round(prob[,1] \* d$n[1]))  
for (i in 2:8) {  
 suc <- cbind(suc, round(prob[,i] \* d$n[i]))  
}  
suc.mean <- apply(suc, 2, mean)  
suc.PI <- apply(suc, 2, PI, prob=.89)

Let’s plot out to see the predictions.

plot(d$y, col=rangi2, xaxt="n", ylab="number of successes", xlab="case",   
 xlim=c(0.75,8.25) , ylim = c(0, 30), pch=16)  
axis(1, at=1:8, labels=c( "LAL","LAS","LIL","LIS","SAL","SAS","SIL","SIS" ))  
points( 1:8 , suc.mean)  
for ( i in 1:8 ) lines( c(i, i), suc.PI[,i])  
mtext('Number of successes, Prediction vs Observation')



The prediction and the observation is similar to each other. It proves that the model is decent.

< Question C > Lastly, let’s consider the interaction between body size and age. Again we build the stan model first, but this time intercations are considered. The code is shown below.

model.02.code <- 'data{  
 int n[8];  
int y[8];  
int V[8];  
int A[8];  
int P[8];  
}  
parameters{  
real a;  
real bP;  
real bA;  
real bV;  
real bPA;  
}  
model{  
vector[8] p;  
bPA ~ normal( 0 , 2 );  
bV ~ normal( 0 , 2 );  
bA ~ normal( 0 , 2 );  
bP ~ normal( 0 , 2 );  
a ~ normal( 0 , 5 );  
for ( i in 1:8 ) {  
p[i] = a + bP \* P[i] + bA \* A[i] + bV \* V[i] + bPA \* P[i] \* A[i];  
p[i] = inv\_logit(p[i]);  
}  
y ~ binomial( n , p );  
}  
generated quantities{  
vector[8] log\_lik;  
vector[8] p;  
for ( i in 1:8 ) {  
p[i] = a + bP \* P[i] + bA \* A[i] + bV \* V[i] + bPA \* P[i] \* A[i];  
p[i] = inv\_logit(p[i]);  
}  
for ( i in 1:8 ) log\_lik[i] = binomial\_lpmf( y[i] | n[i] , p[i] );  
}'

Similar to what we have done with the first stan model, we set a new dataframe to prevent errors occured when modeling from stan model.

dat <- list(  
 N = NROW(d),  
 V = d$V,  
 P = d$P,  
 A = d$A,  
 y = d$y,  
 n = d$n)

model.02 <- stan(model\_code = model.02.code, data = dat, chains = 4)

##   
## SAMPLING FOR MODEL '59a4fd583919f30eba0cef4a3a4486f2' NOW (CHAIN 1).  
## Chain 1:   
## Chain 1: Gradient evaluation took 0 seconds  
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 1: Adjust your expectations accordingly!  
## Chain 1:   
## Chain 1:   
## Chain 1: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 1: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 1: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 1: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 1: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 1: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 1: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 1: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 1: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 1: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 1: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 1: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 1:   
## Chain 1: Elapsed Time: 0.107 seconds (Warm-up)  
## Chain 1: 0.102 seconds (Sampling)  
## Chain 1: 0.209 seconds (Total)  
## Chain 1:   
##   
## SAMPLING FOR MODEL '59a4fd583919f30eba0cef4a3a4486f2' NOW (CHAIN 2).  
## Chain 2:   
## Chain 2: Gradient evaluation took 0 seconds  
## Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 2: Adjust your expectations accordingly!  
## Chain 2:   
## Chain 2:   
## Chain 2: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 2: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 2: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 2: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 2: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 2: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 2: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 2: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 2: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 2: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 2: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 2: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 2:   
## Chain 2: Elapsed Time: 0.198 seconds (Warm-up)  
## Chain 2: 0.155 seconds (Sampling)  
## Chain 2: 0.353 seconds (Total)  
## Chain 2:   
##   
## SAMPLING FOR MODEL '59a4fd583919f30eba0cef4a3a4486f2' NOW (CHAIN 3).  
## Chain 3:   
## Chain 3: Gradient evaluation took 0 seconds  
## Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 3: Adjust your expectations accordingly!  
## Chain 3:   
## Chain 3:   
## Chain 3: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 3: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 3: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 3: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 3: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 3: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 3: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 3: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 3: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 3: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 3: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 3: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 3:   
## Chain 3: Elapsed Time: 0.18 seconds (Warm-up)  
## Chain 3: 0.124 seconds (Sampling)  
## Chain 3: 0.304 seconds (Total)  
## Chain 3:   
##   
## SAMPLING FOR MODEL '59a4fd583919f30eba0cef4a3a4486f2' NOW (CHAIN 4).  
## Chain 4:   
## Chain 4: Gradient evaluation took 0 seconds  
## Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0 seconds.  
## Chain 4: Adjust your expectations accordingly!  
## Chain 4:   
## Chain 4:   
## Chain 4: Iteration: 1 / 2000 [ 0%] (Warmup)  
## Chain 4: Iteration: 200 / 2000 [ 10%] (Warmup)  
## Chain 4: Iteration: 400 / 2000 [ 20%] (Warmup)  
## Chain 4: Iteration: 600 / 2000 [ 30%] (Warmup)  
## Chain 4: Iteration: 800 / 2000 [ 40%] (Warmup)  
## Chain 4: Iteration: 1000 / 2000 [ 50%] (Warmup)  
## Chain 4: Iteration: 1001 / 2000 [ 50%] (Sampling)  
## Chain 4: Iteration: 1200 / 2000 [ 60%] (Sampling)  
## Chain 4: Iteration: 1400 / 2000 [ 70%] (Sampling)  
## Chain 4: Iteration: 1600 / 2000 [ 80%] (Sampling)  
## Chain 4: Iteration: 1800 / 2000 [ 90%] (Sampling)  
## Chain 4: Iteration: 2000 / 2000 [100%] (Sampling)  
## Chain 4:   
## Chain 4: Elapsed Time: 0.089 seconds (Warm-up)  
## Chain 4: 0.108 seconds (Sampling)  
## Chain 4: 0.197 seconds (Total)  
## Chain 4:

print(model.02, include = F, pars = "log\_lik", probs = c(.1, .5, .9))

## Inference for Stan model: 59a4fd583919f30eba0cef4a3a4486f2.  
## 4 chains, each with iter=2000; warmup=1000; thin=1;   
## post-warmup draws per chain=1000, total post-warmup draws=4000.  
##   
## mean se\_mean sd 10% 50% 90% n\_eff Rhat  
## a 0.01 0.02 0.75 -0.93 0.02 0.96 1552 1  
## bP 4.22 0.02 0.80 3.21 4.19 5.30 1581 1  
## bA 1.85 0.02 0.77 0.85 1.84 2.85 1445 1  
## bV -3.85 0.02 0.71 -4.78 -3.83 -2.96 2253 1  
## bPA -1.18 0.02 0.90 -2.35 -1.19 -0.06 1545 1  
## p[1] 0.73 0.00 0.09 0.62 0.74 0.84 4539 1  
## p[2] 0.99 0.00 0.01 0.98 0.99 1.00 3087 1  
## p[3] 0.59 0.00 0.09 0.48 0.59 0.71 4221 1  
## p[4] 0.98 0.00 0.01 0.96 0.99 0.99 2453 1  
## p[5] 0.14 0.00 0.07 0.05 0.12 0.24 2702 1  
## p[6] 0.85 0.00 0.07 0.75 0.86 0.93 2986 1  
## p[7] 0.03 0.00 0.02 0.01 0.02 0.05 1799 1  
## p[8] 0.50 0.00 0.17 0.28 0.50 0.72 1555 1  
## lp\_\_ -51.16 0.04 1.56 -53.31 -50.85 -49.45 1788 1  
##   
## Samples were drawn using NUTS(diag\_e) at Thu May 16 14:17:30 2019.  
## For each parameter, n\_eff is a crude measure of effective sample size,  
## and Rhat is the potential scale reduction factor on split chains (at   
## convergence, Rhat=1).

Let’s compare the two stan model using waic. The first stan model doesn’t consider interactions, while the second one does. We first extract log\_lik and then evaluate by loo and waic. The code is shown below.

log\_lik\_01 <- extract\_log\_lik(model.01, merge\_chains = FALSE)  
r\_eff <- relative\_eff(exp(log\_lik\_01))  
loo\_01 <- loo(log\_lik\_01, r\_eff = r\_eff, cores = 2)

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.

## Warning in log(z): 產生了 NaNs

waic\_01 <- waic(log\_lik\_01)

## Warning: 4 (50.0%) p\_waic estimates greater than 0.4. We recommend trying  
## loo instead.

log\_lik\_02 <- extract\_log\_lik(model.02, merge\_chains = FALSE)  
r\_eff <- relative\_eff(exp(log\_lik\_02))  
loo\_02 <- loo(log\_lik\_02, r\_eff = r\_eff, cores = 2)

## Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for details.

waic\_02 <- waic(log\_lik\_02)

## Warning: 4 (50.0%) p\_waic estimates greater than 0.4. We recommend trying  
## loo instead.

loo::compare(waic\_01, waic\_02)

## elpd\_diff se   
## 2.5 1.7

Since the elpd\_diff is positive, we can conclude that the second model, which considers interactions, is better than the first model.