Introduction to Bayesian linear regression with brms — Part III

Stefano Coretta

26/06/2020

Contents


```
pilot <- list.files(
  path = "./data/perceptual/",
  pattern = "*.csv",
  full.names = TRUE
) %>%
  map_df(~read_csv(., col_types = cols(.default = "c"))) %>%
  mutate(
    key_resp_2.keys = ifelse(key_resp_2.keys == "z", "d", "t"),
    key_resp_8.keys = ifelse(key_resp_8.keys == "z", "t", "d"),
    response = coalesce(key_resp_2.keys, key_resp_8.keys),
    response = factor(response, levels = c("d", "t")),
    burst = as.numeric(burst),
    voicing = as.numeric(voicing),
    response_n = as.numeric(response) - 1,
    vowel = factor(vowel, levels = c("a", "i", "u"))
  ) %>%
  filter(!is.na(response))

contrasts(pilot$vowel) <- "contr.sum"

burst <- filter(pilot, condition == "burst") %>%
  select(burst, vowel, participant, response, response_n)
voicing <- filter(pilot, condition == "voicing")
```

1 Quick recap

We collect a sample of observations $y = (y_1, ..., y_n)$ generated by a **random variable** Y, i.e. we sample values generated by a random event.

We use the sample y to obtain an estimate of Y, i.e. we perform statistical inference.

Y is both *unknown* (we don't know the value it takes) and *uncertain* (we can't be sure about the value it takes). Uncertainty can be expressed by probability distributions.

A **probability distribution** is a list of values along with its corresponding probability. Probabilities take on values between 0 (impossible event) and 1 (certain event).

Probability distributions can be summarised using **parameters**. The *normal distribution* (aka Gaussian distribution) has two parameters: the mean μ and the standard deviation σ (or the variance σ^2). Parameters are random variables themselves, i.e. they are uncertain. We can express distribution parameters with probability distributions (which have *hyperparameters*, or in other words parameters on parameters). We can estimate the hyperparameters from the sample y.

For example:

$$y_i \sim Normal(\mu, \sigma)$$
$$\mu \sim Normal(\mu_1, \sigma_1)$$
$$\sigma \sim HalfCauchy(x_0, \gamma)$$

y is our outcome variable (our sample of values). We believe that the values of y were generated by a normal distribution which has parameters μ and σ (this is the *likelihood*, or simply the *probability distribution of the outcome variable*). We believe that the value of μ comes from a normal distribution with μ_1 and σ_1 and that the value of σ comes from a half Cauchy distribution with parameters x_0 and γ.

We convey our prior beliefs of what μ and σ might be by specifying probability distributions (the **prior probability distributions**, or *priors*). In other words, we specify the hyperparameters μ_1, σ_1, x_0, γ.

These hyperparameters are updated by the Bayesian model with the information provided by the sample, and we thus obtain the **posterior probability distributions** (or *posteriors*) of μ and σ.

2 Discrete variables

In Part I and II we focussed on *continuous variables* and continous probability distributions, among which the normal (Gaussian) distribution is the most commonly employed for statistical analyses. A continuous variable is a variable that can take on any value between two specific values (numbers). For example, height, weight, vowel formants, segment duration, and reaction times are all continuous variables.

Some events, on the other hand, generate **discrete variables**, i.e. variables that can only take specific values and not values in between. For example, binary outcomes like flips of a coin, yes/no questions, and success/failure are all discrete variables. Counts are discrete as well (if you count how many fish there are in a pond, you can't have 5.5 fish).

Probability distributions that generate discrete variables are discrete probability distributions.

In this tutorial, I will discuss how to model binary outcomes using binomial/Bernoulli distributions.

3 Binary outcomes: binomial/Bernoulli

Binary outcomes (yes/no questions, binary choices, successes/failures) follow the **binomial distribution**. The binomial distribution has two parameters: the number of trials n and the probability p of getting one of the two oucomes in each trial (p = probability of getting A, $1 - p$ = probability of getting B).

When fitting a model with the binomial family, the data needs to be in a special format. Let's assume you asked a yes/no question and participants answered either "yes" (= 1) or "no" (= 0). You have to indicate the total number of trials (for example, 20 participants), and the number of trials in which the participants have answered "yes" (= 1). For example, 15/20 participants have answered "yes".

However, binary outcome variables can also be formalised through the **Bernoulli distribution**. The Bernoulli distribution is a special case of the binomial, where the total number of trials n is 1, and p is the probability of one of the two oucomes (for example, "yes", 1). With the Bernoulli distribution, the data is generally entered as we normally do, i.e. for each observation we state whether the answer was 0 or 1 ("yes" or "no", etc…). In this format, we don't have to specify the total number of trials.

If you used `glm[er]()` with `family = binomial` before, note that the model automatically uses the Bernoulli distribution when you don't specify the total number of trials and the data is in the "long" format (one trial per row, specifying what the outcome was). On the other hand, `brms()` does not do this, so you have to explicitly use the `bernoulli()` family when you want to fit "binomial regressions".

4 Example: perception of voicing in Italian

To show hot to fit a binomial/Bernoulli regression, we will use data from a pilot study of the perception of voicing in Italian stops (note that the design of the experiment was very limited, so take this as toy data).

In this pilot study, I asked listeners to say whether they heard a voiced or a voiced stop after listening to synthetic nonce words (*pata*, *piti*, *putu*). The duration of the burst of the /t/ was manipulated in 5 steps (details don't matter), where 0 was the shorter duration and 5 the longest (so a total of 6 degrees of duration). The /t/ was voiceless throughout its closure in all tokens.

Here is how the data frame looks like (`burst` is the burst duration step between 0 and 5, and `response` is the listener response, either voiced "d" or voiceless "t"):

```
burst
```

```
## # A tibble: 360 x 5
##    burst vowel participant response response_n
##    <dbl> <fct> <chr>       <fct>         <dbl>
##  1     3 u     f001        t                 1
##  2     2 u     f001        d                 0
##  3     4 u     f001        t                 1
##  4     0 a     f001        d                 0
##  5     1 u     f001        t                 1
##  6     5 i     f001        t                 1
##  7     2 a     f001        t                 1
##  8     3 a     f001        t                 1
##  9     3 i     f001        t                 1
## 10     1 a     f001        d                 0
## # ... with 350 more rows
```

What we want to model here is the probability of choosing "voiceless" with increasing burst duration.

We can fit a Bernoulli distribution to the outcome variable (`response`) since it's binary ("voiceless" or "voiced"). The code of a simple model for the binary outcome `response` looks like this:

```
brm(
  response ~
    burst +
    (burst | participant),
  data = burst,
  family = bernoulli()
)
```

As usual, the first thing is to set up our priors. Let's check which ones we need to specify.

```
get_prior(
  response ~
    burst +
    (burst | participant),
  data = burst,
  family = bernoulli()
)
```

```
##                  prior     class      coef       group resp dpar nlpar bound
## 1                              b
## 2                              b     burst
## 3               lkj(1)       cor
## 4                            cor           participant
## 5 student_t(3, 0, 2.5) Intercept
## 6 student_t(3, 0, 2.5)        sd
## 7                              sd           participant
## 8                              sd     burst participant
## 9                              sd Intercept participant
```

So, nothing unusual. There's an `Intercept`, `b` for burst, `cor` because we included a random slope, and `sd` for the random intercept.

There's a catch though. Since what we are modelling here is a probability, which is a number between 0 and 1, we have to use a "link function" that maps (transforms) the numbers between 0 and 1 of the outcome variable to a continuous scale of unbounded numbers. These numbers on a continuous scale are used to estimate the coefficients of the model. (I won't explain link functions in details, and you might be familiar with the concept already if you fitted binomial regressions before).

The link function of Bernoulli models is the logistic unit function, or "logit" function for short (the link function for Gaussian/normal models is the "identity" function, which means that the outcome variable is not transformed). When using the logit function, the coefficients returned by the model are in log-odds, not probabilities. So priors need to be specified on that same scale, namely log-odds, which makes things a bit less straightforward.

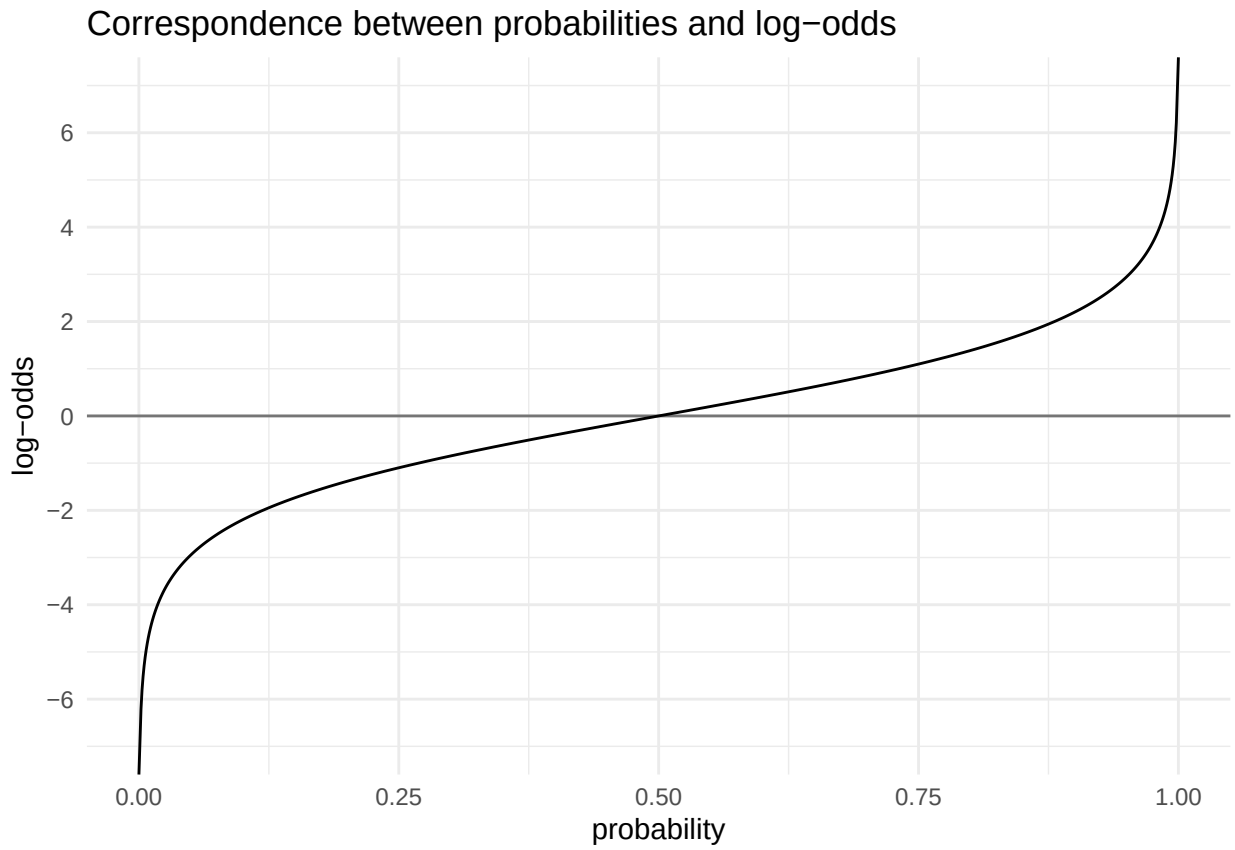
4.0.1 Priors for models using Bernoulli distributions

Priors for models that use the Bernoulli distribution for the outcome variable have to be defined on the log-odd scale. It is thus helpful to know how percentages (numbers between 0 and 1) map onto log-odds.

The following plot shows percentages on the x-axis and the corresponding log-odds on the y-axis. You will notice that a 50% probability ($p = 0.5$) corresponds to a log-odd of 0. Probabilities smaller than 0.5 map onto negative log-odds, while probabilities greater than 0.5 map onto positive log-odds. Finally, when approaching probabilities 0 and 1, the change in log-odds decreases asymptotically (ad infinitum), so that most of the

variation in log-odds is within the range -7 and +7. Since log-odds increase/decrease asymptotically, they are on a continuous scale and can be used in a regression model.

```
tibble(
  p = seq(0, 1, by = 0.001),
  log_odds = qlogis(p)
) %>%
  ggplot(aes(p, log_odds)) +
  geom_hline(aes(yintercept = 0), alpha = 0.5) +
  geom_line() +
  scale_y_continuous(breaks = seq(-6, 6, by = 2)) +
  labs(
    title = "Correspondence between probabilities and log-odds",
    x = "probability",
    y = "log-odds"
  )
```



A quick way to know which log-odd corresponds to which probability (and vice versa) is to use the functions `plogis()` and `qlogis()` (`logis` stands for "logistic function").

```
# log-odds = 0; get probability
plogis(0)
```

```
## [1] 0.5
```

```
# p = 0.5; get log-odds
qlogis(0.5)
```

```
## [1] 0
```

I suggest playing with these two functions to familiarise yourself better with the logit function.

We can start now with the prior for the intercept of our model of voicing perception. In our model, the intercept is the probability of obtaining a "voiceless" response when the burst duration is 0. A very weakly informative prior would be one which includes the whole range of probabilities, but in which a probability of 0 or 1 are less likely than probabilities around 0.5. Such prior can be expressed by a normal distribution with mean 0 and standard deviation 5 (`normal(0, 5)`).

To see why, we can do our usual maths to get the 95% credible interval: mean - 2 times the standard deviation = 0 - 2 x 5 = -10, mean + 2 times the standard deviation = 0 + 2 * 5 = +10. So, the 95% credible interval for `normal(0, 5)` is [-10, +10]. This interval is in log-odds and to get the respective probabilities we can use the `plogis()` function.

```
plogis(-10)
```

```
## [1] 4.539787e-05
```

```
plogis(10)
```

```
## [1] 0.9999546
```

The output shows that a log-odd of -10 corresponds approximately to a probability of 0.000045, while a log-odd of 10 corresponds almost to a probability of 1. In sum, a `normal(0, 5)` prior corresponds to a belief that the probability of obtaining a "voiceless" response when the burst duration is 0 is anything between 0 and 1 at 95% confidence. This is a very weakly informative prior (anything goes).

Now, for the prior of the effect of `burst` we can use the prior `normal(0, 2.5)`. The 95% CI of this prior is [-5, +5]. Since `burst` is a numeric variable, the effect of `burst` on the probability of getting a "voiceless" response is based on the unit increase of `burst`, which means that the effect of `burst` tells us what is the change in probability when increasing the duration of the burst by 1 unit relative to the intercept. Probabilities can also be expressed in odds and the change in probability can be expressed as change in odds.

Odds (or odd-ratios) can be calculated from log-odds by exponentiating the log-odds with the `exp()` function (this will come handy when we'll look at Poisson models). For example, a log-odd of 0 corresponds to an odd of 1 ($exp(0) = 1$), i.e. there is no change. A log-odd of 5 is 148 in odds, meaning that there is an increase in odds by a magnitude of 148, which is quite a big effect. So a `normal(0, 2.5)` prior is a very weakly informative prior.

4.1 Bernoulli model

We can now define our priors (we would also normally run prior predictive checks, but I will skip it here to save time).

```
priors <- c(
  prior(normal(0, 5), class = Intercept),
  prior(normal(0, 2.5), class = b)
)
```

We fit the model.

```
burst_bm <- brm(
  response ~
    burst +
    (burst | participant),
  data = burst,
  family = bernoulli(),
  prior = priors,
  file = "./cache/burst_bm"
)
```

You would normally proceed by doing posterior predictive checks and sensitivity analysis. If you are using Bayes Factors, remember to compute them with models that have increasingly informative priors.

The summary:

```
burst_bm
```

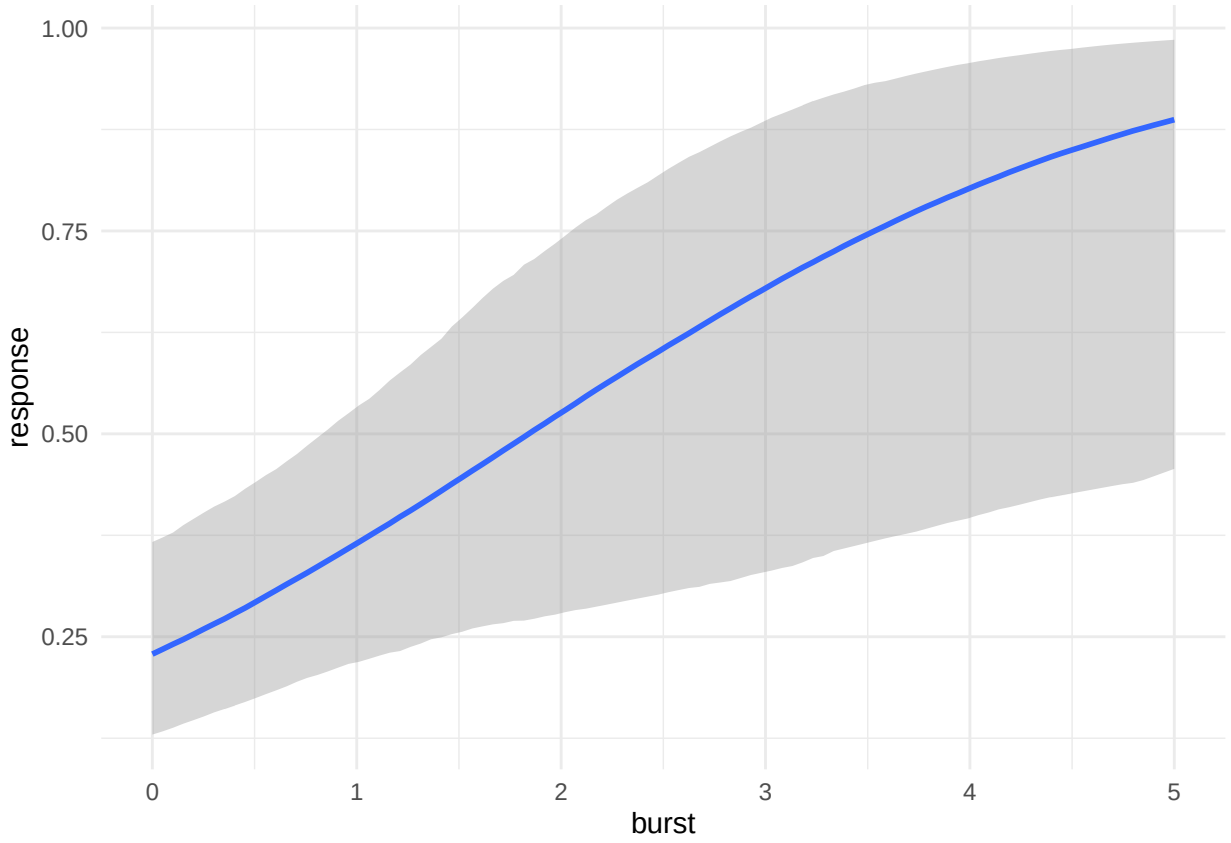
```
## Warning: There were 25 divergent transitions after warmup. Increasing
## adapt_delta above 0.8 may help. See http://mc-stan.org/misc/
## warnings.html#divergent-transitions-after-warmup

##  Family: bernoulli
##   Links: mu = logit
## Formula: response ~ burst + (burst | participant)
##    Data: burst (Number of observations: 360)
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
##          total post-warmup samples = 4000
##
## Group-Level Effects:
## ~participant (Number of levels: 4)
##                     Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
## sd(Intercept)           0.45      0.48     0.01     1.71 1.00     1173
## sd(burst)               0.37      0.27     0.04     1.11 1.01      398
## cor(Intercept,burst)   -0.02      0.59    -0.94     0.96 1.01      631
##                     Tail_ESS
## sd(Intercept)           1288
## sd(burst)                414
## cor(Intercept,burst)    1731
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept    -1.22      0.34    -1.91    -0.55 1.01     1407     1640
## burst         0.66      0.21     0.21     1.11 1.00      479      297
##
## Samples were drawn using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

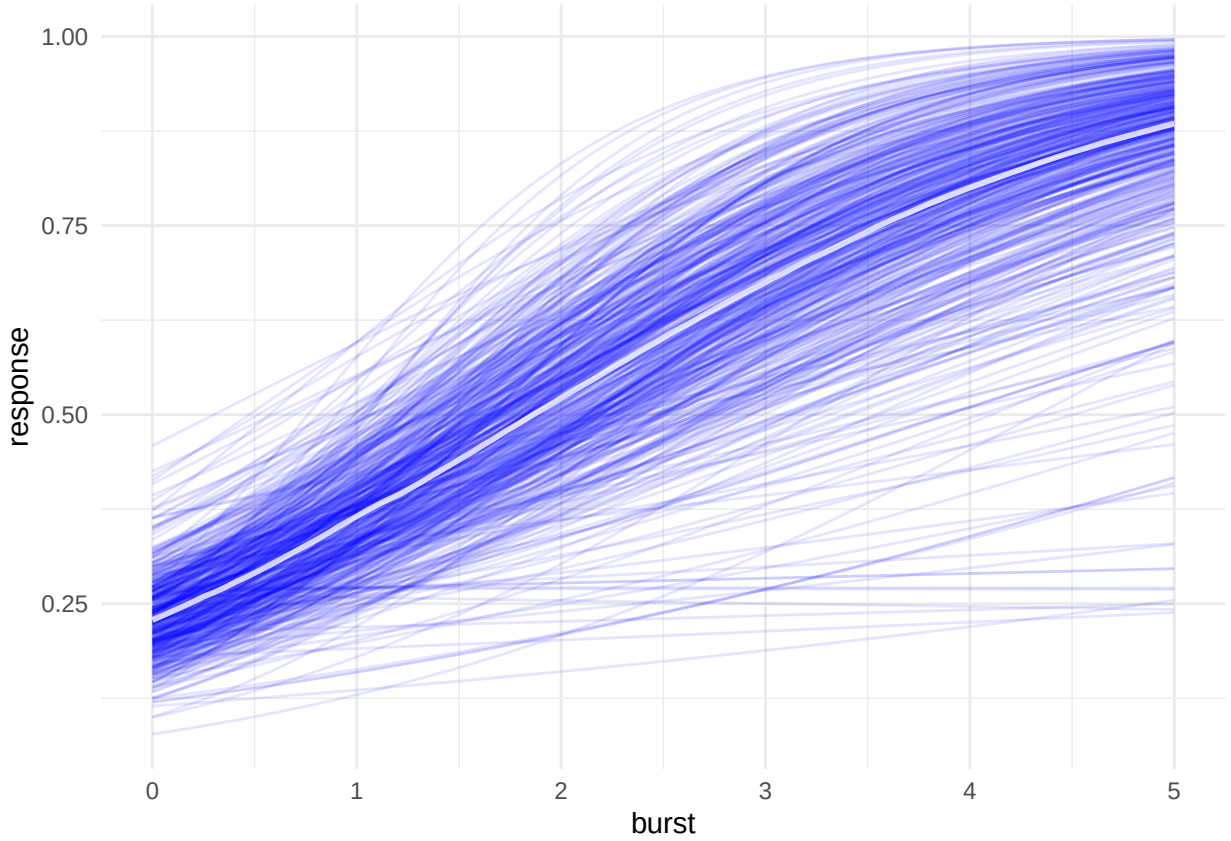
A warning is issued that says there were 25 divergent transitions. Even if the model converged (the $\hat{R}$ values are fine), divergent transitions are not a good sign, and it is generally better to increase the `adapt_delta` value as suggested. This can be done by running the model again with an extra argument in the call: `control = list(adapt_delta = 0.999)`. (To save time, I won't rerun the model here.)

For a quick plot of the results:

```
conditional_effects(burst_bm)
```




```
conditional_effects(burst_bm, spaghetti = TRUE, nsamples = 500)
```



As you can see, there's a lot of uncertainty (i.e. large CIs). Not surprising, given the sample size (which is incredibly small).

4.2 Extra

Note that a GLMER model with the same random specification as the model `burst_bm` does not converge.

```
burst_glm <- glmer(
  response ~
    burst +
    (burst | participant),
  data = burst,
  family = binomial()
)
```

```
## boundary (singular) fit: see ?isSingular
```

A simplified model converges and it returns very low p-values, which are clearly anti-conservative values (compare with the high degree of uncertainty in results of `burst_bm`).

```
burst_glm_simple <- glmer(
  response ~
    burst +
    (1 | participant),
```

```
  data = burst,
  family = binomial()
)

summary(burst_glm_simple)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: response ~ burst + (1 | participant)
##    Data: burst
##
##      AIC      BIC   logLik deviance df.resid
##    416.6    428.3   -205.3    410.6      357
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.1998 -0.7790  0.3703  0.6895  2.3385
##
## Random effects:
##  Groups      Name        Variance Std.Dev.
##  participant (Intercept) 0.1516   0.3893
## Number of obs: 360, groups:  participant, 4
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.17930    0.29034  -4.062 4.87e-05 ***
## burst        0.62154    0.07882   7.885 3.14e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##       (Intr)
## burst -0.614
```