Generalized Linear Models with the brms R Package

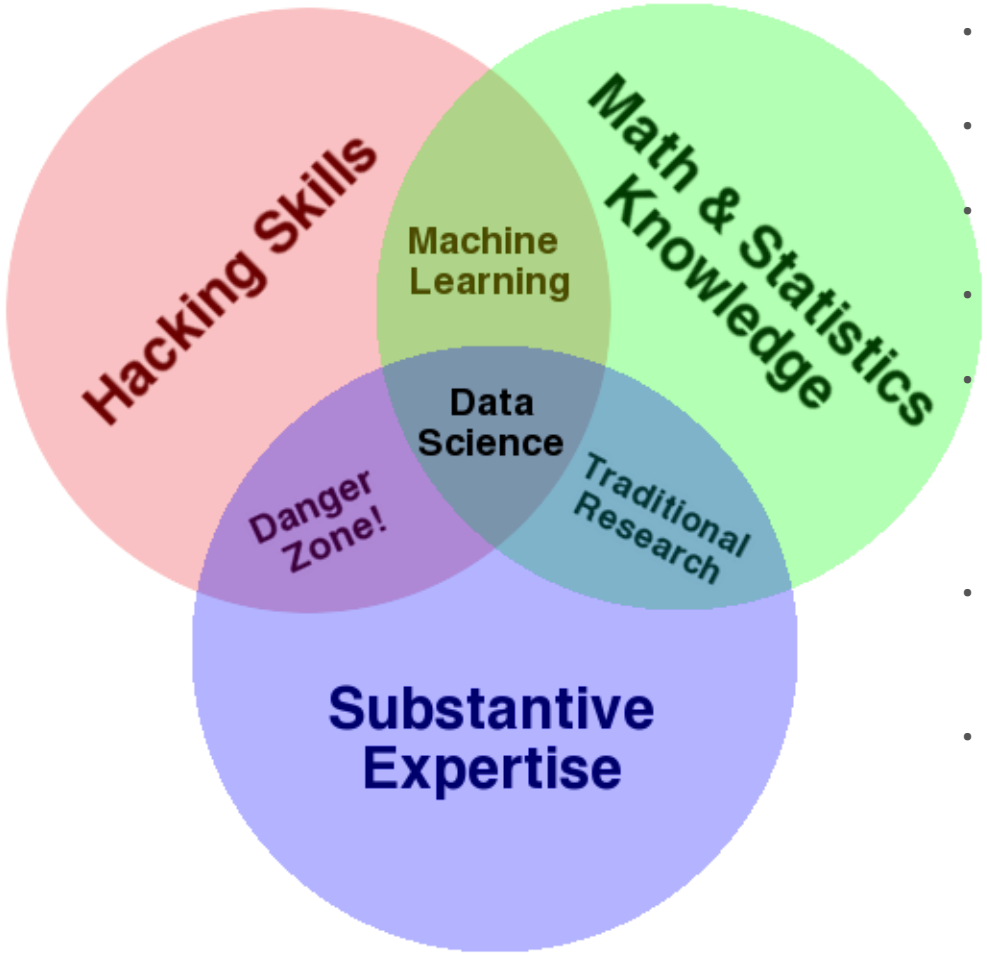
Ben Goodrich

April 11, 2022

Scoring Rules

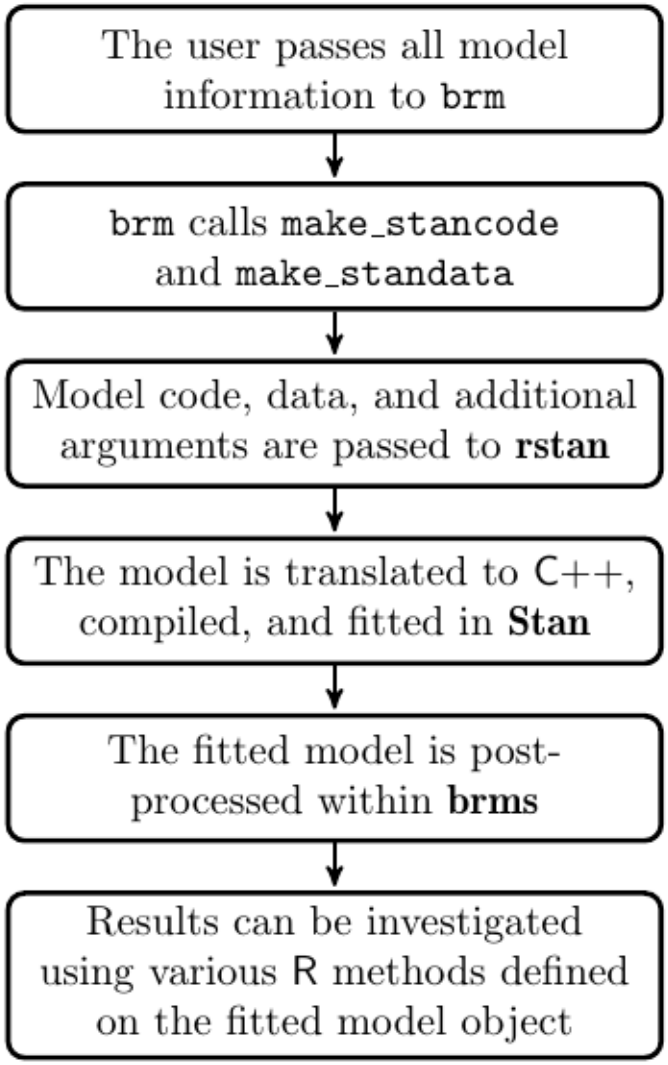
- The proportion of correct classifications is a classic example of an “improper” scoring rule due to the discontinuity at 0.5
- Bayesians (should, and typically do) use “strictly proper” scoring rules https://en.wikipedia.org/wiki/Scoring_rule , such as the ELPD, which are defined by the property that they are maximized / minimized by the truth
- If the model is correct, the posterior predictive distribution is expected to yield the best-calibrated predictions with respect to any strictly proper scoring rule
- Supervised learning presumes there is no truth, in which case, the justification for using a strictly proper scoring rule is less compelling. However, the choice of which improper scoring rule to use can determine which modeling procedure is best in the testing data

Data Science Venn Diagram



- What do you get if you equate . . .
- Probability
- Priors, DAGs, and Model Building
- Markov Chain Monte Carlo
- Since at least 1990, Bayesian estimation has been what data science purported to be but isn't
- Data science came along later but largely ignored Bayesian approaches
- In reality, data science programs are asymmetric with more emphasis on hacking skills and less on the others

The brms Workflow (Figure 1 in Bürkner 2016)



The brms workflow

The Arguments to `brm`

```
library(brms)
args(brm)
```

```
## function (formula, data, family = gaussian(), prior = NULL, autocor = NULL,
##     data2 = NULL, cov_ranef = NULL, sample_prior = "no", sparse = NULL,
##     knots = NULL, stanvars = NULL, stan_funs = NULL, fit = NA,
##     save_pars = NULL, save_ranef = NULL, save_mevars = NULL,
##     save_all_pars = NULL, inits = "random", chains = 4, iter = 2000,
##     warmup = floor(iter/2), thin = 1, cores = getOption("mc.cores",
##         1), threads = NULL, normalize = getOption("brms.normalize",
##         TRUE), control = NULL, algorithm = getOption("brms.algorithm",
##         "sampling"), backend = getOption("brms.backend", "rstan"),
##     future = getOption("future", FALSE), silent = 1, seed = NA,
##     save_model = NULL, stan_model_args = list(), file = NULL,
##     file_refit = "never", empty = FALSE, rename = TRUE, ...)
## NULL
```

The `formula` Argument to `brm`

- Everything to the right of the `~` is the same as in many other R functions
- In many cases, the thing to the left of the `~` is simply the outcome variable
- However, `brm` introduces a new possibility for this syntax like `y | fun(variable)`, where `fun` could be
  - `cens()` and `trunc()` to specify known censoring or truncation bounds
  - `weights()` and `disp()`, which should not be used with MCMC
  - `se()` to specify "known" standard errors in meta-analyses
  - `trials()`, which is used in binomial models only
  - `cat()` to specify the possible categories for ordinal models

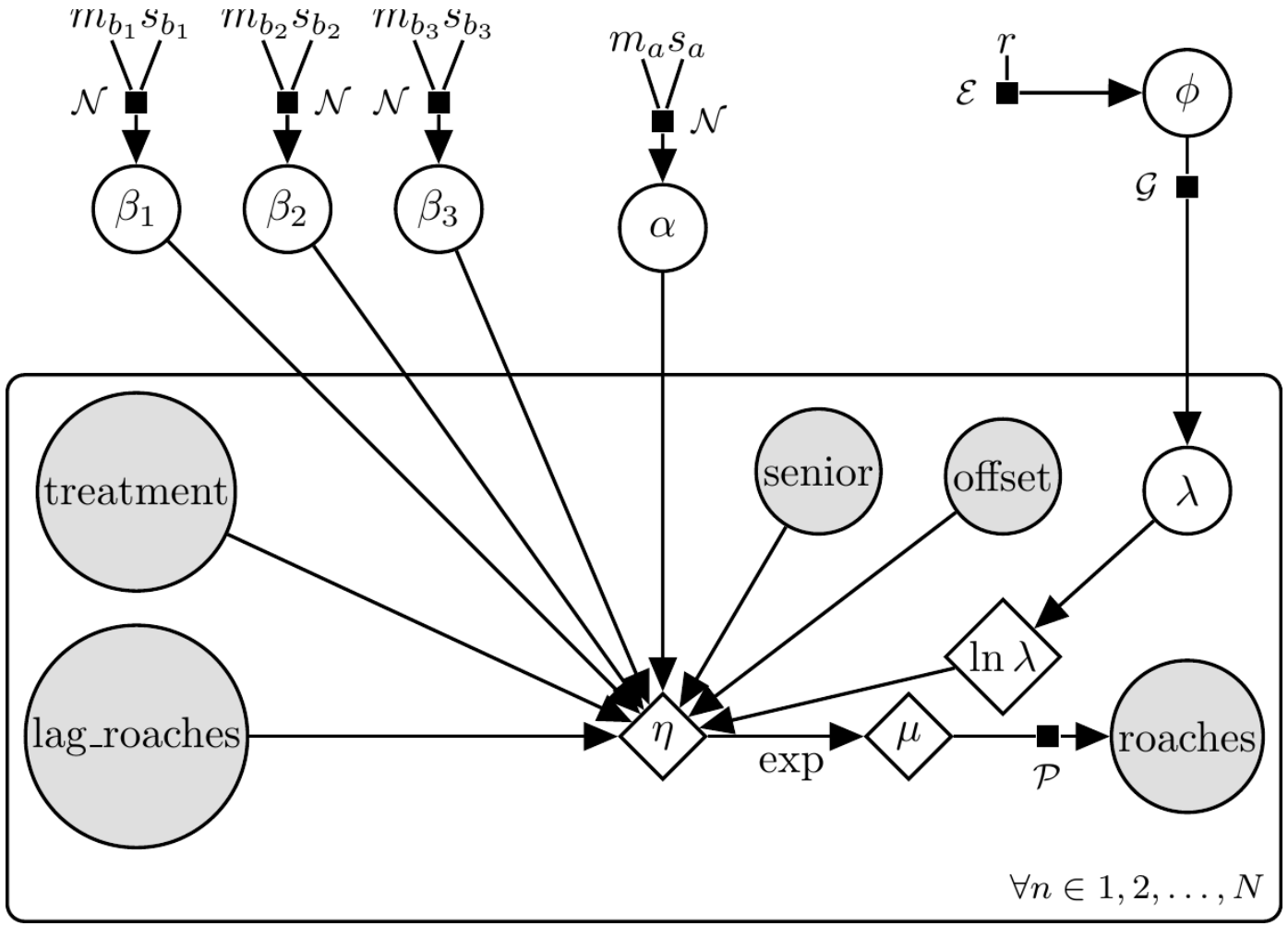
The `family` Argument to `brm`

The `family` argument can be any of the following functions, which also have a link argument that can be a variety of things depending on the family

```
gaussian; student; binomial; bernoulli; poisson; negbinomial; geometric; Gamma;
skew_normal; lognormal; shifted_lognormal; exgaussian; wiener; inverse.gaussian;
exponential; weibull; frechet; Beta; dirichlet; von_mises; asym_laplace;
gen_extreme_value; categorical; multinomial; cumulative; cratio; sratio; acat;
hurdle_poisson; hurdle_negbinomial; hurdle_gamma; hurdle_lognormal;
zero_inflated_binomial; zero_inflated_beta; zero_inflated_negbinomial;
zero_inflated_poisson; zero_one_inflated_beta
```

- The ones involving `hurdle_`, `zero_inflated_` and / or `negbinomial` are of particular interest in the social sciences

Prior Predictive Distribution for Roach Study



Roach Model

The `prior` Argument to `brm`

```
args(set_prior) # or usually just prior()
```

```
## function (prior, class = "b", coef = "", group = "", resp = "",
##     dpar = "", nlpar = "", lb = NA, ub = NA, check = TRUE)
## NULL
```

- `prior` is a character string (in the Stan language) such as `"normal(0,5)"` but you can omit the quotation marks if you instead call `prior`, which forwards to `set_prior`
- `class` indicates what parameters the call to `set_prior` pertains to
- `coef` is the name of the parameter in question
- `group` is the name of the grouping factor (if applicable)
- `resp` is the name of the response variable in multivariate models
- `dpar` is the name of the distribution parameter (if applicable)
- `nlpar` is the name of the non-linear parameter (if applicable)
- `lb` is the lower bound of the parameter (default $-\infty$)
- `ub` is the upper bound of the parameter (default ∞)
- `check` whether priors should be checked for validity

The `get_prior` Function

- Input the `formula`, `data`, and `family` and get back the possible prior choices (and defaults)

```
data(roaches, package = "rstanarm"); roaches <- roaches[roaches$roach1 > 0, ]
get_prior(y ~ log(roach1) + treatment + senior + offset(log(exposure2)),
          data = roaches, family = negbinomial)
```

```
##                  prior     class      coef group resp dpar nlpar bound       source
##                (flat)         b                                             default
##                (flat)         b logroach1                                (vectorized)
##                (flat)         b    senior                                (vectorized)
##                (flat)         b treatment                                (vectorized)
##  student_t(3, 1.9, 2.9) Intercept                                          default
##       gamma(0.01, 0.01)     shape                                          default
```

The `class` Argument to `set_prior`

- Refers to a type of parameter in the model
- Defaults to `"b"` which refers to (population-level) regression coefficients
- Other possible values are `"Intercept"`, `"sd"`, `"cor"`, `"sigma"` and others we may talk about later

```
my_prior <- prior(normal(0, 2), class = "b") + prior(normal(0, 5), class = "Intercept") +
  prior(exponential(1), class = "shape")
```

Example of `brm`

```
post <- brm(y ~ log(roach1) + treatment + senior + offset(log(exposure2)), data = roaches,
            family = negbinomial, prior = my_prior) # from previous slide
```

```
post
```

```
...
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept     1.33      0.26     0.84     1.85 1.00     4624     3083
## logroach1     0.70      0.06     0.57     0.82 1.00     4543     3167
## treatment    -0.62      0.21    -1.04    -0.21 1.00     4918     3137
## senior       -0.20      0.24    -0.65     0.28 1.00     4170     2915
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## shape     0.47      0.05     0.38     0.58 1.00     4694     3334
##
## 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).
...
```

Using the `loo` Function

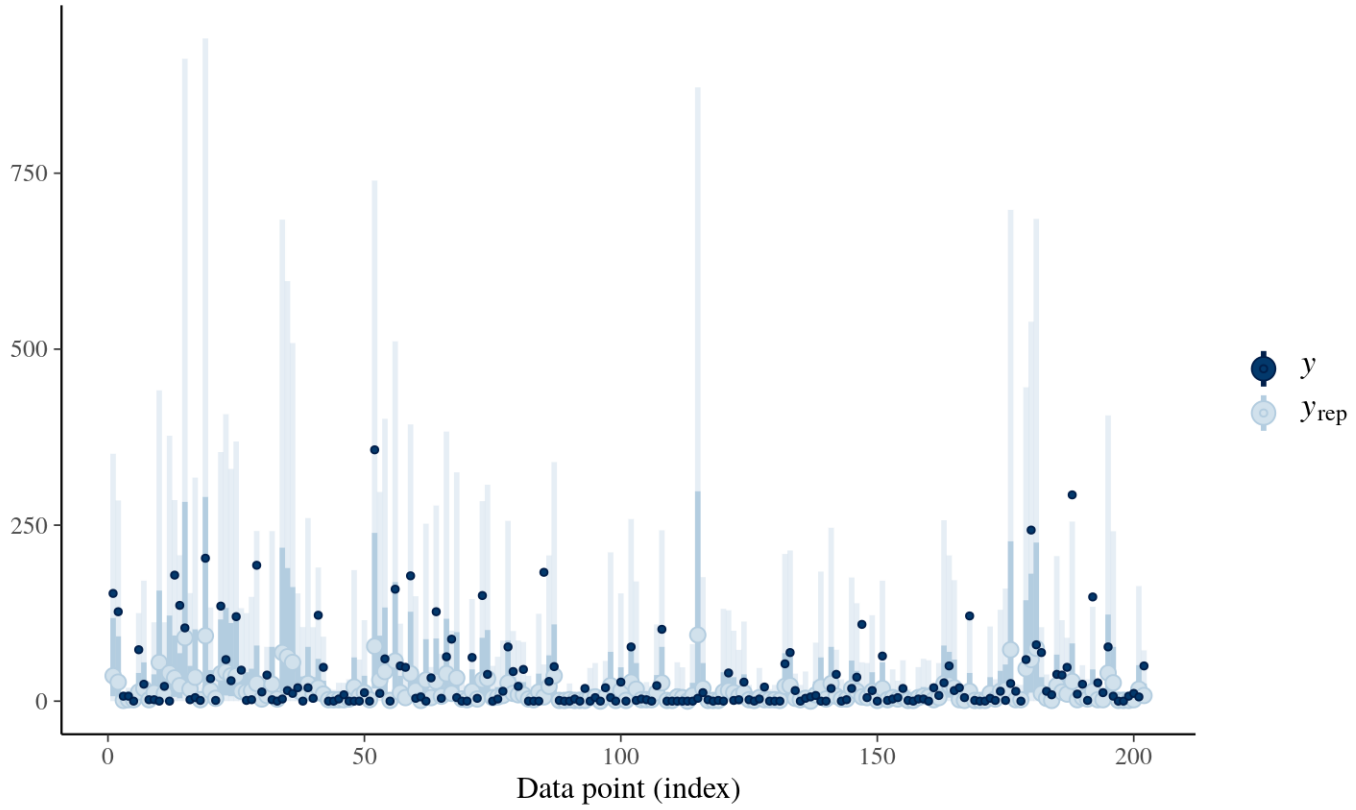
- McElreath cautions against using things like `loo` when λ_n is included
- If λ_n is integrated out of the posterior distribution by using a negative binomial likelihood, everything's fine (unless there are warnings, in which case `reloo`)

```
loo_post <- loo(post, reloo = TRUE) # observation 85 has a Pareto k > 0.7
loo_post
```

```
##
## Computed from 4000 by 202 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -759.2 28.3
## p_loo         5.2  1.5
## looic      1518.3 56.6
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     201   99.5%   1035
##  (0.5, 0.7]   (ok)         1    0.5%   380
```

Using the **pp_check** Function

```
pp_check(post, type = "loo_intervals") # type is the same as plotfun with rstanarm
```



Using the **hypothesis** Function

- To do this with **rstanarm**, you would have to first call `as.matrix`

```
args(brms:::hypothesis.brmsfit)
```

```
## function (x, hypothesis, class = "b", group = "", scope = c("standard",
##     "ranef", "coef"), alpha = 0.05, seed = NULL, ...)
## NULL
```

- Here `x` is the object produced by `brm` and `hypothesis` is a string, typically with an embedded `<` or `>`, such as

```
hypothesis(post, "treatment < 0")
```

```
## Hypothesis Tests for class b:
##        Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio Post.Prob Star
## 1 (treatment) < 0    -0.62      0.21    -0.97    -0.27     443.44         1    *
## ---
## 'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
## '*': For one-sided hypotheses, the posterior probability exceeds 95%;
## for two-sided hypotheses, the value tested against lies outside the 95%-CI.
## Posterior probabilities of point hypotheses assume equal prior probabilities.
```

Other Post-Estimation Methods

Many of the things you can do with an object produced by `brm` are analagous to **rstanarm**

```
##       [,1]                  [,2]                       [,3]
##  [1,] "add_criterion"       "loo_compare"              "posterior_predict"
##  [2,] "add_ic"              "loo_linpred"              "posterior_samples"
##  [3,] "as.array"            "loo_model_weights"        "posterior_smooths"
##  [4,] "as.data.frame"       "loo_moment_match"         "posterior_summary"
##  [5,] "as.matrix"           "loo_predict"              "pp_average"
##  [6,] "as.mcmc"             "loo_predictive_interval" "pp_check"
##  [7,] "autocor"             "loo_R2"                   "pp_mixture"
##  [8,] "bayes_factor"        "loo_subsample"            "predict"
##  [9,] "bayes_R2"            "loo"                      "predictive_error"
## [10,] "bridge_sampler"      "LOO"                      "predictive_interval"
## [11,] "coef"                "marginal_effects"         "prepare_predictions"
## [12,] "conditional_effects" "marginal_smooths"         "print"
## [13,] "conditional_smooths" "mcmc_plot"                "prior_samples"
## [14,] "control_params"      "model_weights"            "prior_summary"
## [15,] "cv_varsel"           "model.frame"              "ranef"
## [16,] "expose_functions"    "neff_ratio"               "reloo"
## [17,] "family"              "ngrps"                    "residuals"
## [18,] "fitted"              "nobs"                     "rhat"
## [19,] "fixef"               "nsamples"                 "stancode"
## [20,] "formula"             "nuts_params"              "standata"
## [21,] "get_refmodel"        "pairs"                    "stanplot"
## [22,] "getCall"             "parnames"                 "summary"
## [23,] "hypothesis"          "plot"                     "update"
## [24,] "kfold"               "post_prob"                "VarCorr"
## [25,] "launch_shinystan"    "posterior_average"        "varsel"
## [26,] "log_lik"             "posterior_epred"          "vcov"
## [27,] "log_posterior"       "posterior_interval"       "waic"
## [28,] "logLik"              "posterior_linpred"        "WAIC"
```

Hurdle Models with `brm`

There was an EdStem post about zero-inflated and hurdle models for count outcomes, which you can easily do with `brm`.

```
post_hurdle <- brm(brms::bf(y ~ log(roach1) + treatment + senior + offset(log(exposure2)),
                            hu ~ I(roach1 == 0) + treatment + senior), data = roaches,
                   family = hurdle_negbinomial, seed = 12345, prior = my_prior +
                     prior(normal(0, 2), class = "b", dpar = "hu"))
```

Results of Hurdle Model

```
post_hurdle
```

```
...
## Population-Level Effects:
##                     Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept               1.64      0.28     1.08     2.17 1.00     6921     2888
## hu_Intercept           -1.58      0.29    -2.18    -1.03 1.00     7451     3245
## logroach1               0.61      0.07     0.49     0.74 1.00     6875     3046
## treatment              -0.51      0.21    -0.93    -0.09 1.00     6898     3291
## senior                 -0.04      0.26    -0.54     0.48 1.00     7020     3009
## hu_Iroach1EQEQ0TRUE     0.01      2.00    -3.85     4.06 1.00     7683     2849
## hu_treatment            0.38      0.35    -0.30     1.10 1.00     7712     3188
## hu_senior               0.82      0.36     0.11     1.50 1.00     7513     3193
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## shape     0.64      0.11     0.43     0.87 1.00     4764     2555
##
## 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).
...
```

PSISLOOCV Comparison

```
loo(post, post_hurdle, reloo = TRUE)
```

```
## Output of model 'post':
##
## Computed from 4000 by 202 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -759.2 28.3
## p_loo         5.2  1.5
## looic      1518.3 56.6
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     201   99.5%   1035
##  (0.5, 0.7]   (ok)         1    0.5%   380
##    (0.7, 1]   (bad)        0    0.0%   <NA>
##    (1, Inf)   (very bad)   0    0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
##
## Output of model 'post_hurdle':
##
## Computed from 4000 by 202 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -772.9 28.0
## p_loo         8.0  1.5
## looic      1545.7 55.9
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     201   99.5%   2881
##  (0.5, 0.7]   (ok)         1    0.5%   184
##    (0.7, 1]   (bad)        0    0.0%   <NA>
##    (1, Inf)   (very bad)   0    0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
##
## Model comparisons:
##             elpd_diff se_diff
## post          0.0       0.0
## post_hurdle -13.7       4.9
```

Structured Nonlinear Model

- The `stan_gamm4` package in **rstanarm** can estimate effects that are arbitrary smooth functions of predictors, as can `brm`
- `brm` can do that and estimate many more models with particular non-linear forms (that are just pasted into a Stan program)
- Examples from a brms vignette
- It is a good idea to put the outcome variable in reasonable units

```
url <- "https://raw.githubusercontent.com/mages/diesunddas/master/Data/ClarkTriangle.csv"
loss <- readr::read_csv(url)
loss$cum <- loss$cum / 1000
fit_loss <- brm(brms::bf(cum ~ ult * (1 - exp(-(dev / theta) ^ omega)),
                         ult ~ 1, omega ~ 1, theta ~ 1, nl = TRUE),
                data = loss, family = gaussian(),
                prior = c(prior(normal(5, 1), nlpar = "ult"),
                          prior(normal(1, 2), nlpar = "omega"),
                          prior(normal(45, 10), nlpar = "theta")),
                control = list(adapt_delta = 0.9))
```

Resulting Nonlinear Plot

```
plot(conditional_effects(fit_loss), points = TRUE)
```