

Package: gasplit (via r-universe)

October 8, 2026

Title Flexible estimation of 2-category proportions

Author Mark Bravington <markb2@summerinsouth.net>

Description Like a GLM/GAM where the true category of each sample is unknown, but the response data is a continuous variable with known distributions for category-1 and category-2 samples.

Maintainer Mark Bravington <markb2@summerinsouth.net>

Imports mvbutils, atease, mgcv, RTMB

Suggests offarray

License GPL-3

Version 1.4.16

Repository <https://markbravington.r-universe.dev>

Date/Publication 2025-06-28 00:49:42 UTC

RemoteUrl <https://github.com/markbravington/gasplit>

RemoteRef HEAD

RemoteSha 986feff334f87082c44edce2aae79200e403e562

Contents

example_prep4missy	2
gasplit	3
make_fake_ppns	7
posterior	8

Index	11
--------------	-----------

example_prep4missy	<i>Schematic data-prep for 'gasplit_missy'</i>
--------------------	--

Description

example_prep4missy is a "template" showing how to prep a dataset with (one) sometimes-missing covariate (ie field) into a form suitable for [gasplit_missy](#) (qv). You could adapt example_prep4missy to your own situation.

There is an example dataset babydat just to demonstrate the prep (you can't use it to fit a model, coz the d1/d2 distributions are missing). The covariate in question is Sex, taking values "F", "M", or NA. Someone has already fitted a model which predicts the probability that a case will be Female ("F") based on its other covariates (but **not** on whatever the ultimate response variable will be!); that prediction has been stored in the field PrFem_y1. Cases where Sex is known are left untouched; cases where Sex is NA are duplicated (since there are only two possible values for this covariate), with one copy getting Sex="F", and the other Sex="M"; the field PROBIMP is set accordingly based on PrFem_y1. The MULTIMP field is used to link together the multiple versions of each case.

Remember that *you* have to develop whatever submodel for probabilities of the alternative values that *your* missing covariates might take! If you've got missing values on continuous covariate, rather than a discrete one like Sex, my recommendation is to add rows based on a few quantiles of the covariate (which will potentially be at different values for each case), and give them all equal weight (ie in the analog of PrFem_y1). I'd start by trying 5 quantiles (0.1; 0.3; ...; 0.9), but you could increase & see if it matters.

Usage

```
example_prep4missy( data)
```

Arguments

data	data.frame as per description.
------	--------------------------------

See Also

[gasplit_missy](#)

Examples

```
babydat      # with some missing values for 'Sex'  
example_prep4missy( babydat)
```

Description

Suppose your data each come from one or the other of two categories (say, Eastern or Western Atlantic). You don't know for sure which one, but you do have a "measurement" for each one (e.g. some kind of log-likelihood ratio, but it could be anything) which tends to be high for one category and low for the other, with an a-priori-known distribution for each category (say, based on other samples of known category).

Now you want to estimate the proportion of category-1 samples in your data, but stratified according to covariates of each sample (say, year and location)— and/or simply dependent on individual covariates, (say, size). Ideally, you'd like to be able to specify a flexible model formula just as you might in a GLM or GAM. Well, now you can! For example:

```
addmod <- gasplit( DISCRIM ~ factor(year) + factor(region) - 1,
  data= ew_data,
  d1= known_distro_of_measurement_for_category_1,
  d2= ditto_for_2
)
```

to fit a two-way model with no interactions, to a dataset with covariates "year" and "region" based on a measurement "DISCRIM".

Having fitted your model, you can make a subsequent call to predict the proportion for (presumably) new covariate values, via the `predict_from_previous` argument. And, provided that your "measurement" really is a log-likelihood ratio, you can also calculate a posterior probability (of being category-1) for each case, using [posterior](#).

If you want to use smoothers, i.e. `s()` terms a la mgcv (including non-sparse random effects handled via `s(bs="re")`), then you should call `gasplit2` instead. The latter requires package **RTMB**, in order to estimate smoothing parameters/random-effect variances. Results should be the same as `gasplit` if there are no smooth terms nor random effects; IDNK which routine is faster. NDIRC (Nor Do I Really Care.) If your covariates have missing values (MAR), you can do "full information maximum likelihood" using `gasplit_missy`, described next.

See **Examples** of [test_gasplit](#) (qv) for practical stuff. I should update that for smoothers and missy.

Missing covariates: `gasplit_missy` can handle Missing-At-Random (MAR) values in covariates via "full information maximum likelihood", though *you* have to have already done your homework by fitting sub-models that predict the probabilities of the different possible covariate values for each case. The covariate(s) should be intrinsically discrete, or reasonably discretizable; if they are continuous, a good approximation would be use a few quantiles of their conditional distribution.

Your data will be a copy of the original dataset with extra rows added (one or more extra rows each case with missing covariates), and two extra columns, by default assumed to be called MULTIMP and PROBIMP though you can change that. Their meaning (which is very simple) is explained in

the next paragraph. You might find it helpful to look at an example in [example_prep4missy](#) (qv), and the test dataset [babydat](#). (It's not for use in actually fitting `gasplit_missy`— just showing how the data prep works.)

MULTIMP—a misnomer really, since this is actually FIML not "multiple implementation"—should hold the "case number", so that a missing-covariate case will appear several times with the same value for some of its columns (including MULTIMP and the response variable), but different values for the missing covariate(s) in each appearance (which I might loosely call each "imputation"). The probabilities of those alternative covariate values are in PROBIMP, from that submodel(s) you've fitted (which will usually be conditional on the covariates that *are* observed). Cases with no (relevant) missing covariates should have a unique MULTIMP so they only appear once, and a PROBIMP of 1. MULTIMP does not have to contain numbers; any unique case-identifier will do. Fitted values are only returned for the first occurrence of each case, but take into account all "imputations" of that case. Predictions on new data don't allow for missing covariates, and the new data does not need MULTIMP or PROBIMP fields.

If there are cases where several covariates have missing values, then in principle you should produce extra rows for all combinations of possible values for that case. That could get messy for you, but fortunately it's not *my* problem; *you* might decide to do Multiple-Imputation instead (though it is less statistically efficient, and more fiddly).

Disclaimer: I've tried to support most `mgcv` features that I more-or-less understand. For example, I *think* `gasplit2` can cope with `s(..., id=...)`. However:

- I'm not sure how robust my solutions are;
- there are lots of things in `mgcv` that I *don't* understand (and haven't used) which may not work at all.

An example of something that might well misfire, is trying to fix some smoothing parameters at chosen values. Let me know how y'all go...

There's deffo no support for sparsity.

Usage

```
gasplit(
  formula,
  data,
  d1, d2,
  start=0.001,
  predict_from_previous= NULL,
  dbeta= FALSE
)
gasplit2(
  formula,
  data,
  d1, d2,
  predict_from_previous= NULL,
  dbeta= FALSE,
  ... # for gam()
)
gasplit_missy(
  formula,
```

```

    data,
    d1, d2,
    link_field = "MULTIMP", prob_field = "PROBIMP",
    predict_from_previous= NULL,
    dbeta= FALSE,
    ... # for gam()
)

```

Arguments

formula	a classic R model formula (eventually allowing mgcv extensions)
data	a dataframe...
d1, d2	these are <i>functions</i> giving the PDF of the response variable for category-1 and category-2 samples. You gotta pre-fit those to other data. See Examples .
start	(just gasplit, not gasplit2). Either a scalar (probably close to 0) which will be used for all coefficients, or a vector with length ncol(<model matrix>). TBH I'd leave it blank; the default will prolly work.
link_field, prob_field	what names the missing-data columns columns have. Probably simpler to make sure your dataset has them under the default names.
predict_from_previous	Optional result from a previous call to gasplit. If set, then no fitting will happen, but gasplit will make predictions at the (presumably new) values of data, based on the parameters of the previous fit.
dbeta	whether to calculate derivatives of each prediction WRT to the parameters (mainly for subsequent variance calculations).
...	for gasplit2 and gasplit_missy only, additional args to mgcv::gam(qv), eg knots.

Details

Given parameters β and a model-matrix X from the formula, the log-likelihood of observation i with response y_i is:

$$\log(\text{ppn}_i d_1(y_i) + (1 - \text{ppn}_i) d_2(y_i))$$

where $\text{ppn}_i = \text{inv.logit}(\sum_j \{X_{i,j} \beta_j\})$. The log-likelihood extracts the constant factor $d_2(y_i)$, to leave (in the non-missy case)

$$\log(1 + \text{ppn}_i (d_1/d_2 - 1)) + \text{const}$$

In a general setting, when a covariate z_i is missing for case i but other covariates x_i may be observed, the FIML log-lik of an observation y_i is

$$\log(\sum_z \{ \Pr[y_i | x_i, z] * \Pr[Z_i = z | x_i] \})$$

where the sum ranges over possible values z that the random variable Z_i might take, and the second prob is your "submodel". The point here is that the weighted sum is (and should be) applied to the *unlogged* probabilities, *before* the whole lot gets logged finally. That's why the code needs to be adapted.

For our particular application, this becomes:

$$\begin{aligned} & \log(\text{sum_z}\{ (\text{ppn_i}(z) * d1(y_i) + (1 - \text{ppn_i}(z)) * d2(y_i)) * \text{prob}(z_i = z) \} \\ & = \log(1 + (d1_i/d2_i - 1) * \text{sum_z}\{ \text{ppn_i}(z) * \text{prob}(z_i = z) \} \end{aligned}$$

with some obvious & harmless abuse of notation.

Smoothers and random-effects add "penalties" (ie priors) on the coefficients in the usual mgcv way, and RTMB computes the automated Laplace approximation required for consistent estimation of smoopars/RE-variances.

Value

If `predict_from_previous` is supplied (and data, of course), all three functions return predicted proportions at each row of data. If `dbeta==TRUE` then the result will be not a vector, but a matrix whose first column is the point estimate, the remaining columns being its derivatives WRTO the elements of `beta`. You can use that to get standard errors on the predictions, something like this:

```
t( ppn[, -1]) %***% V_beta %***% ppn[, -1]
```

Otherwise, the functions return a list, with various elements including at least:

`beta`, `SE_beta`, `V_beta`

for the raw parameter estimates;

`ppn` estimated proportions, casewise (or a matrix if `dbeta==TRUE`);

`G` setup for the underlying [gam](#), so eg `<obj>$G$X` is the design matrix

plus various convergence diagnostics. `gasplit2` and `gasplit_missy` also returns `$obj` which is the RTMB object (from `MakeADFun`) after fitting, plus the estimated "outer parameters" (smoothing parameters and/or random-effect variances) as `outerpars` (NB probably on log scale). The latter are treated as fixed when computing the coefficient Hessian, i.e. there's no Bayes-Empirical-Bayes correction. `gasplit_missy` also returns the `link_field` and `prob_field` arguments, plus a 3-column matrix `$split`, with columns "PPN", `<link>`, and `<prob>` (the latter two set by `link_field` and `prob_field`). This corresponds row-wise to the input dataframe (and to `$G$X`), so that a single case might be split into several imputations. It is used by [posterior](#). Note that the casewise proportions `$ppn` has just row per case.

See Also

`test_gasplit`, `example_prep4gasplit`

Examples

```
?test_gasplit
?example_prep4gasplit
```

make_fake_ppns

*Simulate dataset for 'gasplit'***Description**

make_fake_ppns simulates a dataset for fitting with [gasplit](#), using two factors Y and Z and giving a response variable DISCOSTAT; the strata comprise all combinations of Y and Z. You can control the number of factor-levels, distributions of DISCOSTAT for category-1 and category-2 observations (and thus the "power" of DISCOSTAT for assignment), the number of samples per stratum, the average strength of each Y-effect and Z-effect, the average strength of Y*Z interactions, etc. It requires the **offarray** package (purely for my programming convenience; it's not apparent in the output).

test_gasplit fits a [gasplit](#) model (with no interactions) to the output of make_fake_ppns, running the latter itself if required. The output is the (true and) fitted proportions per strata, which can be compared: see **Examples**. Note that it's a bit harsh to simulate with `interpow>0` and then test with [gasplit](#), coz the latter assumes there's no interactions. (Of course, you can fit a model with interactions if you want.)

Usage

```
make_fake_ppns(nyears = 4, nzones = 3, interpow = 0.1,
               df_t = 5, mean_nsamp = 100, meanE = 1, prange = 1, seed = 2)
test_gasplit(
  sim=NULL,
  formulalala= DISCOSTAT ~ Y+Z-1,
  use2= 's' %in% all.names( formula),
  ...
)
```

Arguments

nyears, nzones	number of factor levels
interpow	average strength of an interaction relative to a main effect
mean_nsamp	actually there are <i>exactly</i> this many total samples from each stratum
df_t, meanE	df_t degrees-of-freedom for the t-distributions of response by category, which will have variance 1 and means +meanE and -meanE. A meanE of 1 corresponds to pretty poor separation; 3 is pretty good.
prange	how strong the effects should be. If they are too strong, then
seed	random number seed. The system seed is restored on completion.
sim, ...	(test_gasplit) sim should come from a previous call to make_fake_ppns. If not supplied, then make_fake_ppns will be run first, using the ... arguments.
formulalala	what model to fit. The default corresponds to the simulation in make_fake_ppns. You can try putting smoothers in there, to test gasplit2.
use2	if TRUE, force test_gasplit to fit with gasplit2 rather than gasplit. Normally auto-deduced based on whether formulalala contains a call to s (but the test is not completely reliable, if you care). Even without a smoother, gasplit and gasplit2 should give the same result.

Value

`make_fake_ppns` returns a dataframe that has all the stuff you need, plus an attribute `truth` which is a list containing the real dope, as stratum-specific offarrays. Note that `truth$samp_ppnE` is the actual ppn of category-1 ("E") samples in each stratum, which will differ from the nominal value given by the factor strengths and interactions because of binomial sampling variability. `test_gasplit` returns a great long list of Stuff. See **Examples** and use yr noddle!

See Also

[gasplit](#)

Examples

```
if( require( offarray)){
sim2 <- make_fake_ppns(
  meanE= +2, # not-brilliant separation
  mean_nsamp=200, # reasonable samp sizes
  interpow=0) # no interactoins
hist( sim2$DISCOSTAT, nc=40) # to see how good the separation is
test2 <- test_gasplit( sim2)
with( test2, plot( tru_ppnE, fit_ppnE))
abline( 0,1)
test2b <- test_gasplit( sim2, use2=TRUE)
summary( test2$fit_ppnE - test2b$fit_ppnE) # close to 0...
# Make up some new data...
splodge <- with( test2, simpure[ 1:7,])
# ... and predict. The result of the original call to gasplit() is
# stored in test2$gg1
splodge_pred <- gasplit( data=splodge, predict_from_previous=test2$gg1)
# ... for normal non-test situations, do something more like this:
if( FALSE){
  pfit <- gasplit( myform, mydata)
  ppred <- gasplit( data=newmydata, predict_from_previous=pfit)
}
# See also *Examples* for 'posterior'
}
```

posterior

Posterior probability

Description

After fitting a [gasplit](#) model to estimate the *prior* probability of whateverness by covariates, you can also calculate a per-observation *posterior* probability.

The posteriors are also estimates, since they depend on the estimated model coefficients β . You can also get their derivatives WRT to the coefficients (useful for downstream variance propagation).

Missing data can be handled a la [gasplit_missy](#) (qv) by setting the `link_field` and `prob_field` arguments. This means the outputs are aggregated to case level with weighted sums over the imputations (which I think is the correct approach). It is most likely to be useful for extracting posteriors

from an existing model that was fitted by [gasplit_missy](#) (when it will be automatic), rather than for brand-new predictions. But, whatever floats your modeling boat.

Usage

```
posterior(
  object,
  newdata = NULL,
  dbeta = FALSE,
  link_field= object$link_field %||% NA,
  prob_field= object$prob_field %||% NA
)
```

Arguments

object	from previous gasplit etc call
newdata	default is to use the original data, but it can accept a dataframe with the necessary covariates. If newdata includes the columns named in link_field and prob_field, then results will be aggregated casewise as per gasplit_missy.
dbeta	if TRUE, return derivs of the posterior probs WRT to the params
link_field, prob_field	optional for "multiple imputation" with missing data; see gasplit_missy. (The funny-looking defaults circumvent R's perhaps unintuitive behaviour if there were no such fields in the original model.) Unlikely with de novo newdata but you never know.

Value

If dbeta==FALSE, a vector of estimated posterior probs, one per case. If not, then a matrix where the first column is the posteriors, and the remaining columns are the derivs WRT to each component of beta.

See Also

[gasplit](#), [gasplit_missy](#)

Examples

```
sim2 <- make_fake_ppns(
  meanE= +2, # not-brilliant separation
  mean_nsamp=200, # reasonable samp sizes
  interpow=0) # no interactoins
hist( sim2$DISCOSTAT, nc=40)
test <- test_gasplit( sim2)
with( test, plot( tru_ppnE, fit_ppnE))
abline( 0,1)
# Posteriors, and their derivs
posty <- posterior( test$gg1, dbeta=TRUE)
# Base-R does not define logit()--- super-annoying. Sigh (again).
plot( sim2$DISCOSTAT, mvbutils::logit( posty[,1]), pch='.')
```

```
abline( 0,1,col='red')
lm( posty[,1] ~ test$gg1$ppn) # 0, 1 pretty much
posty[1,2] # d postprob[1] / dparam[2]
# ... Points with high DISCOSTAT tend to pushed upwards in the posterior
# ... which makes sense, coz if you've got high DISCOSTAT, your neighbours prolly will too
# ... so your prior would be pushed up
```

Index

* **misc**

example_prep4missy, [2](#)

gasplit, [3](#)

make_fake_ppns, [7](#)

posterior, [8](#)

babydat, [4](#)

babydat (example_prep4missy), [2](#)

example_prep4missy, [2](#), [4](#)

gam, [6](#)

gasplit, [3](#), [7–9](#)

gasplit2 (gasplit), [3](#)

gasplit_missy, [2](#), [8](#), [9](#)

gasplit_missy (gasplit), [3](#)

make_fake_ppns, [7](#)

posterior, [3](#), [6](#), [8](#)

test_gasplit, [3](#)

test_gasplit (make_fake_ppns), [7](#)