Package 'projpred'

August 19, 2022

Encoding UTF-8

Title Projection Predictive Feature Selection

Version 2.2.0

Date 2022-08-18

Description Performs projection predictive feature selection for generalized linear and additive models as well as for generalized linear and additive multilevel models (see Piironen, Paasiniemi and Vehtari, 2020, <doi:10.1214/20-EJS1711>; Catalina, Bürkner and Vehtari, 2020, <arXiv:2010.06994>). The package is compatible with the 'rstanarm' and 'brms' packages, but other reference models can also be used. See the documentation as well as the package vignette for more information and examples.

License GPL-3 | file LICENSE

URL https://mc-stan.org/projpred/, https://discourse.mc-stan.org

BugReports https://github.com/stan-dev/projpred/issues/

Depends R (>= 3.5.0)

Imports methods, dplyr, loo (>= 2.0.0), rstantools (>= 2.0.0), lme4, mvtnorm, ggplot2, Rcpp, utils, magrittr, mgcv, gamm4, rlang

Suggests rstanarm, brms, testthat, knitr, rmarkdown, glmnet, cmdstanr, bayesplot (>= 1.5.0), optimx, posterior, parallel, foreach, iterators, doParallel, future, future.callr, doFuture

LinkingTo Rcpp, RcppArmadillo

Additional_repositories https://mc-stan.org/r-packages/

LazyData TRUE

RoxygenNote 7.2.1

VignetteBuilder knitr, rmarkdown

NeedsCompilation yes

Author Juho Piironen [aut], Markus Paasiniemi [aut], Alejandro Catalina [aut],

Frank Weber [cre, aut],
Aki Vehtari [aut],
Jonah Gabry [ctb],
Marco Colombo [ctb],
Paul-Christian Bürkner [ctb],
Hamada S. Badr [ctb]

Maintainer Frank Weber <fweber144@protonmail.com>

Repository CRAN

Date/Publication 2022-08-19 08:40:02 UTC

R topics documented:

`projpred-package` *Projection predictive feature selection*

Description

projpred is an R package for performing a projection predictive variable (or "feature") selection for generalized linear models (GLMs), generalized linear multilevel (or "mixed") models (GLMMs), generalized additive models (GAMs), and generalized additive multilevel (or "mixed") models (GAMMs), with the support for additive models still being experimental. Note that the term "generalized" includes the Gaussian family as well.

The package is compatible with **rstanarm** and **brms**, but developers of other packages are welcome to add new `get_refmodel()` methods (which enable the compatibility of their packages with **projpred**). Custom reference models can also be used via `init_refmodel()`. It is via custom reference models that **projpred** supports the projection onto candidate models whose predictor terms are not a subset of the reference model's predictor terms. However, for **rstanarm** and **brms** reference models, **projpred** only supports the projection onto *submodels* of the reference model. For the sake of simplicity, throughout this package, we use the term "submodel" for all kinds of candidate models onto which the reference model is projected, even though this term is not always appropriate for custom reference models.

Currently, the supported families are `gaussian()`, `binomial()` (and—via `brms::get_refmodel.brmsfit()`—also `brms::bernoulli()`), as well as `poisson()`.

The projection of the reference model onto a submodel can be run on multiple CPU cores in parallel (across the projected draws). This is powered by the **foreach** package. Thus, you can use any parallel (or sequential) backend compatible with **foreach**, e.g., the backends from packages **doParallel**, **doMPI**, or **doFuture**. Using the global option `projpred.prll_prj_trigger`, you can modify the number of projected draws below which no parallelization is used (even if a parallel backend is registered). Such a "trigger" threshold exists because of the computational overhead of a parallelization which makes parallelization only useful for a sufficiently large number of projected draws. By default, parallelization is turned off, which can also be achieved by supplying `Inf` (or `NULL`) to option `projpred.prll_prj_trigger`. Note that we cannot recommend parallelizing the projection on Windows because in our experience, the parallelization overhead is larger there, causing a parallel run to take longer than a sequential run. Also note that the parallelization works well for GLMs, but for GLMMs, GAMs, and GAMMs, the fitted model objects are quite big, which—when running in parallel—may lead to an excessive memory usage which in turn may crash the R session. Thus, we currently cannot recommend the parallelization for GLMMs, GAMs, and GAMMs.

The vignettes (currently, there is only a single one) illustrate how to use the **projpred** functions in conjunction. Shorter examples are included here in the documentation.

Some references relevant for this package are given in section "References" below. See `citation(package = "projpred")` for details on citing **projpred**.

Functions

`init_refmodel()`, `get_refmodel()` For setting up a reference model (only rarely needed explicitly).

`varsel()`, `cv_varsel()` For variable selection, possibly with cross-validation (CV).

`summary.vsel()`, `print.vsel()`, `plot.vsel()`, `suggest_size.vsel()`, `solution_terms.vsel()` For post-processing the results from the variable selection.

`project()` For projecting the reference model onto submodel(s). Typically, this follows the variable selection, but it can also be applied directly (without a variable selection).

`as.matrix.projection()` For extracting projected parameter draws.

`proj_linpred()`, `proj_predict()` For making predictions from a submodel (after projecting the reference model onto it).

Author(s)

Maintainer: Frank Weber <fweber144@protonmail.com>

Authors:

- Juho Piironen <juho.t.piironen@gmail.com>
- Markus Paasiniemi
- Alejandro Catalina <alecatfel@gmail.com>
- Aki Vehtari

Other contributors:

- Jonah Gabry [contributor]
- Marco Colombo [contributor]
- Paul-Christian Bürkner [contributor]
- Hamada S. Badr [contributor]

References

Catalina, A., Bürkner, P.-C., and Vehtari, A. (2020). Projection predictive inference for generalized linear and additive multilevel models. *arXiv:2010.06994*. URL: https://arxiv.org/abs/2010.06994.

Dupuis, J. A. and Robert, C. P. (2003). Variable selection in qualitative models via an entropic explanatory power. *Journal of Statistical Planning and Inference*, **111**(1-2):77–94. doi:10.1016/S03783758(02)002860.

Goutis, C. and Robert, C. P. (1998). Model choice in generalised linear models: A Bayesian approach via Kullback–Leibler projections. *Biometrika*, **85**(1):29–37.

Piironen, J. and Vehtari, A. (2017). Comparison of Bayesian predictive methods for model selection. *Statistics and Computing*, **27**(3):711-735. doi:10.1007/s112220169649y.

Piironen, J., Paasiniemi, M., and Vehtari, A. (2020). Projective inference in high-dimensional problems: Prediction and feature selection. *Electronic Journal of Statistics*, **14**(1):2155-2197. doi:10.1214/20EJS1711.

See Also

Useful links:

- https://mc-stan.org/projpred/
- https://discourse.mc-stan.org
- Report bugs at https://github.com/stan-dev/projpred/issues/

`as.matrix.projection` *Extract projected parameter draws*

Description

This is the `as.matrix()` method for `projection` objects (returned by `project()`, possibly as elements of a `list`). It extracts the projected parameter draws and returns them as a matrix.

Usage

```
## S3 method for class 'projection'
as.matrix(x, nm_scheme = "auto", ...)
```

Arguments

`x`	An object of class `projection` (returned by `project()`, possibly as elements of a `list`).
`nm_scheme`	The naming scheme for the columns of the output matrix. Either `"auto"`, `"rstanarm"`, or `"brms"`, where `"auto"` chooses `"rstanarm"` or `"brms"` based on the class of the reference model fit (and uses `"rstanarm"` if the reference model fit is of an unknown class).
`...`	Currently ignored.

Value

An $S_{\text{prj}} \times Q$ matrix of projected draws, with S_{prj} denoting the number of projected draws and Q the number of parameters.

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Projection onto an arbitrary combination of predictor terms (with a small
  # value for `nclusters`, but only for the sake of speed in this example;
  # this is not recommended in general):
  prj <- project(fit, solution_terms = c("X1", "X3", "X5"), nclusters = 10,
                 seed = 9182)
  prjmat <- as.matrix(prj)
```

```
  ### For further post-processing (e.g., via packages `bayesplot` and
  ### `posterior`), we will here ignore the fact that clustering was used
  ### (due to argument `nclusters` above). CAUTION: Ignoring the clustering
  ### is not recommended and only shown here for demonstrative purposes. A
  ### better solution for the clustering case is explained below.
  # If the `bayesplot` package is installed, the output from
  # as.matrix.projection() can be used there. For example:
  if (requireNamespace("bayesplot", quietly = TRUE)) {
    print(bayesplot::mcmc_intervals(prjmat))
  }
  # If the `posterior` package is installed, the output from
  # as.matrix.projection() can be used there. For example:
  if (requireNamespace("posterior", quietly = TRUE)) {
    prjdrws <- posterior::as_draws_matrix(prjmat)
    print(posterior::summarize_draws(
      prjdrws,
      "median", "mad", function(x) quantile(x, probs = c(0.025, 0.975))
    ))
  }
  ### Better solution for post-processing clustered draws (e.g., via
  ### `bayesplot` or `posterior`): Don't ignore the fact that clustering was
  ### used. Instead, resample the clusters according to their weights (e.g.,
  ### via posterior::resample_draws()). However, this requires access to the
  ### cluster weights which is not implemented in `projpred` yet. This
  ### example will be extended as soon as those weights are accessible.
}
```

`break_up_matrix_term` *Break up matrix terms*

Description

Sometimes there can be terms in a formula that refer to a matrix instead of a single predictor. This function breaks up the matrix term into individual predictors to handle separately, as that is probably the intention of the user.

Usage

```
break_up_matrix_term(formula, data)
```

Arguments

`formula`	A `formula` for a valid model.
`data`	The original `data.frame` with a matrix as predictor.

Value

A `list` containing the expanded `formula` and the expanded `data.frame`.

cl_agg *Weighted averaging within clusters of parameter draws*

Description

This function aggregates S parameter draws that have been clustered into S_{cl} clusters by averaging across the draws that belong to the same cluster. This averaging can be done in a weighted fashion.

Usage

```
cl_agg(
  draws,
  cl = seq_len(nrow(draws)),
  wdraws = rep(1, nrow(draws)),
  eps_wdraws = 0
)
```

Arguments

draws	An $S \times P$ matrix of parameter draws, with P denoting the number of parameters.
cl	A numeric vector of length S, giving the cluster indices for the draws. Draws that should be dropped (e.g., by thinning) need to have an NA in cl.
wdraws	A numeric vector of length S, giving the weights of the draws. It doesn't matter whether these are normalized (i.e., sum to 1) or not because internally, these weights are normalized to sum to 1 within each cluster. Draws that should be dropped (e.g., by thinning) can (but must not necessarily) have an NA in wdraws.
eps_wdraws	A positive numeric value (typically small) which will be used to improve numerical stability: The weights of the draws within each cluster are multiplied by 1 - eps_wdraws. The default of 0 should be fine for most cases; this argument only exists to help in those cases where numerical instabilities occur (which must be detected by the user; this function will not detect numerical instabilities itself).

Value

An $S_{\text{cl}} \times P$ matrix of aggregated parameter draws.

Examples

```
set.seed(323)
S <- 100L
P <- 3L
draws <- matrix(rnorm(S * P), nrow = S, ncol = P)
# Clustering example:
S_cl <- 10L
cl_draws <- sample.int(S_cl, size = S, replace = TRUE)
draws_cl <- cl_agg(draws, cl = cl_draws)
# Clustering example with nonconstant `wdraws`:
```

```
w_draws <- rgamma(S, shape = 4)
draws_cl <- cl_agg(draws, cl = cl_draws, wdraws = w_draws)
# Thinning example:
S_th <- 50L
idxs_thin <- round(seq(1, S, length.out = S_th))
th_draws <- rep(NA, S)
th_draws[idxs_thin] <- seq_len(S_th)
draws_th <- cl_agg(draws, cl = th_draws)
# A thinning example with nonconstant `wdraws` doesn't make sense because in
# case of thinning, `wdraws` doesn't have any impact.
```

`cv-indices` *Create cross-validation folds*

Description

These are helper functions to create cross-validation (CV) folds, i.e., to split up the indices from 1 to `n` into `K` subsets ("folds") for K-fold CV. These functions are potentially useful when creating the `cvfits` and `cvfun` arguments for `init_refmodel()`. The return value is different for these two methods, see below for details.

Usage

```
cvfolds(n, K, seed = sample.int(.Machine$integer.max, 1))

cv_ids(
  n,
  K,
  out = c("foldwise", "indices"),
  seed = sample.int(.Machine$integer.max, 1)
)
```

Arguments

`n`	Number of observations.
`K`	Number of folds. Must be at least 2 and not exceed `n`.
`seed`	Pseudorandom number generation (PRNG) seed by which the same results can be obtained again if needed. Passed to argument `seed` of `set.seed()`, but can also be `NA` to not call `set.seed()` at all.
`out`	Format of the output, either `"foldwise"` or `"indices"`. See below for details.

Value

`cvfolds()` returns a vector of length `n` such that each element is an integer between 1 and `k` denoting which fold the corresponding data point belongs to. The return value of `cv_ids()` depends on the `out` argument. If `out = "foldwise"`, the return value is a `list` with `k` elements, each being a `list`

with elements `tr` and `ts` giving the training and test indices, respectively, for the corresponding fold. If `out = "indices"`, the return value is a `list` with elements `tr` and `ts` each being a `list` with `k` elements giving the training and test indices, respectively, for each fold.

Examples

```
n <- 100
set.seed(1234)
y <- rnorm(n)
cv <- cv_ids(n, K = 5, seed = 9876)
# Mean within the test set of each fold:
cvmeans <- sapply(cv, function(fold) mean(y[fold$ts]))
```

`cv_varsel` *Variable selection with cross-validation*

Description

Perform the projection predictive variable selection for GLMs, GLMMs, GAMs, and GAMMs. This variable selection consists of a *search* part and an *evaluation* part. The search part determines the solution path, i.e., the best submodel for each submodel size (number of predictor terms). The evaluation part determines the predictive performance of the submodels along the solution path. In contrast to `varsel()`, `cv_varsel()` performs a cross-validation (CV) by running the search part with the training data of each CV fold separately (an exception is explained in section "Note" below) and running the evaluation part on the corresponding test set of each CV fold.

Usage

```
cv_varsel(object, ...)

## Default S3 method:
cv_varsel(object, ...)

## S3 method for class 'refmodel'
cv_varsel(
  object,
  method = NULL,
  cv_method = if (!inherits(object, "datafit")) "LOO" else "kfold",
  ndraws = NULL,
  nclusters = 20,
  ndraws_pred = 400,
  nclusters_pred = NULL,
  refit_prj = !inherits(object, "datafit"),
  nterms_max = NULL,
  penalty = NULL,
  verbose = TRUE,
  nloo = NULL,
```

```
  K = if (!inherits(object, "datafit")) 5 else 10,
  lambda_min_ratio = 1e-05,
  nlambda = 150,
  thresh = 1e-06,
  regul = 1e-04,
  validate_search = TRUE,
  seed = sample.int(.Machine$integer.max, 1),
  search_terms = NULL,
  ...
)
```

Arguments

object	An object of class refmodel (returned by get_refmodel() or init_refmodel()) or an object that can be passed to argument object of get_refmodel().
...	Arguments passed to get_refmodel() as well as to the divergence minimizer (during a forward search and also during the evaluation part, but the latter only if refit_prj is TRUE).
method	The method for the search part. Possible options are "L1" for L1 search and "forward" for forward search. If NULL, then internally, "L1" is used, except if the reference model has multilevel or additive terms or if !is.null(search_terms). See also section "Details" below.
cv_method	The CV method, either "LOO" or "kfold". In the "LOO" case, a Pareto-smoothed importance sampling leave-one-out CV (PSIS-LOO CV) is performed, which avoids refitting the reference model nloo times (in contrast to a standard LOO CV). In the "kfold" case, a K-fold CV is performed.
ndraws	Number of posterior draws used in the search part. Ignored if nclusters is not NULL or in case of L1 search (because L1 search always uses a single cluster). If both (nclusters and ndraws) are NULL, the number of posterior draws from the reference model is used for ndraws. See also section "Details" below.
nclusters	Number of clusters of posterior draws used in the search part. Ignored in case of L1 search (because L1 search always uses a single cluster). For the meaning of NULL, see argument ndraws. See also section "Details" below.
ndraws_pred	Only relevant if refit_prj is TRUE. Number of posterior draws used in the evaluation part. Ignored if nclusters_pred is not NULL. If both (nclusters_pred and ndraws_pred) are NULL, the number of posterior draws from the reference model is used for ndraws_pred. See also section "Details" below.
nclusters_pred	Only relevant if refit_prj is TRUE. Number of clusters of posterior draws used in the evaluation part. For the meaning of NULL, see argument ndraws_pred. See also section "Details" below.
refit_prj	A single logical value indicating whether to fit the submodels along the solution path again (TRUE) or to retrieve their fits from the search part (FALSE) before using those (re-)fits in the evaluation part.
nterms_max	Maximum number of predictor terms until which the search is continued. If NULL, then min(19, D) is used where D is the number of terms in the reference model (or in search_terms, if supplied). Note that nterms_max does not count

the intercept, so use `nterms_max = 0` for the intercept-only model. (Correspondingly, `D` above does not count the intercept.)

`penalty` Only relevant for L1 search. A numeric vector determining the relative penalties or costs for the predictors. A value of `0` means that those predictors have no cost and will therefore be selected first, whereas `Inf` means those predictors will never be selected. If `NULL`, then `1` is used for each predictor.

`verbose` A single logical value indicating whether to print out additional information during the computations.

`nloo` **Caution:** Still experimental. Only relevant if `cv_method = "LOO"`. Number of subsampled LOO CV folds, i.e., number of observations used for the LOO CV (anything between 1 and the original number of observations). Smaller values lead to faster computation but higher uncertainty in the evaluation part. If `NULL`, all observations are used, but for faster experimentation, one can set this to a smaller value.

`K` Only relevant if `cv_method = "kfold"` and if the reference model was created with `cvfits` being `NULL` (which is the case for `get_refmodel.stanreg()` and `brms::get_refmodel.brmsfit()`). Number of folds in K-fold CV.

`lambda_min_ratio`
Only relevant for L1 search. Ratio between the smallest and largest lambda in the L1-penalized search. This parameter essentially determines how long the search is carried out, i.e., how large submodels are explored. No need to change this unless the program gives a warning about this.

`nlambda` Only relevant for L1 search. Number of values in the lambda grid for L1-penalized search. No need to change this unless the program gives a warning about this.

`thresh` Only relevant for L1 search. Convergence threshold when computing the L1 path. Usually, there is no need to change this.

`regul` A number giving the amount of ridge regularization when projecting onto (i.e., fitting) submodels which are GLMs. Usually there is no need for regularization, but sometimes we need to add some regularization to avoid numerical problems.

`validate_search`
Only relevant if `cv_method = "LOO"`. A single logical value indicating whether to cross-validate also the search part, i.e., whether to run the search separately for each CV fold (`TRUE`) or not (`FALSE`). We strongly do not recommend setting this to `FALSE`, because this is known to bias the predictive performance estimates of the selected submodels. However, setting this to `FALSE` can sometimes be useful because comparing the results to the case where this argument is `TRUE` gives an idea of how strongly the variable selection is (over-)fitted to the data (the difference corresponds to the search degrees of freedom or the effective number of parameters introduced by the search).

`seed` Pseudorandom number generation (PRNG) seed by which the same results can be obtained again if needed. Passed to argument `seed` of `set.seed()`, but can also be `NA` to not call `set.seed()` at all. Here, this seed is used for clustering the reference model's posterior draws (if `!is.null(nclusters)`), for subsampling LOO CV folds (if `nloo` is smaller than the number of observations), for sampling the folds in K-fold CV, and for drawing new group-level effects when predicting from a multilevel submodel (however, not yet in case of a GAMM).

search_terms Only relevant for forward search. A custom character vector of predictor term blocks to consider for the search. Section "Details" below describes more precisely what "predictor term block" means. The intercept ("1") is always included internally via union(), so there's no difference between including it explicitly or omitting it. The default search_terms considers all the terms in the reference model's formula.

Details

Arguments ndraws, nclusters, nclusters_pred, and ndraws_pred are automatically truncated at the number of posterior draws in the reference model (which is 1 for datafits). Using less draws or clusters in ndraws, nclusters, nclusters_pred, or ndraws_pred than posterior draws in the reference model may result in slightly inaccurate projection performance. Increasing these arguments affects the computation time linearly.

For argument method, there are some restrictions: For a reference model with multilevel or additive formula terms, only the forward search is available. Furthermore, argument search_terms requires a forward search to take effect.

L1 search is faster than forward search, but forward search may be more accurate. Furthermore, forward search may find a sparser model with comparable performance to that found by L1 search, but it may also start overfitting when more predictors are added.

An L1 search may select interaction terms before the corresponding main terms are selected. If this is undesired, choose the forward search instead.

The elements of the search_terms character vector don't need to be individual predictor terms. Instead, they can be building blocks consisting of several predictor terms connected by the + symbol. To understand how these building blocks works, it is important to know how **projpred**'s forward search works: It starts with an empty vector chosen which will later contain already selected predictor terms. Then, the search iterates over model sizes $j \in \{1, ..., J\}$. The candidate models at model size j are constructed from those elements from search_terms which yield model size j when combined with the chosen predictor terms. Note that sometimes, there may be no candidate models for model size j. Also note that internally, search_terms is expanded to include the intercept ("1"), so the first step of the search (model size 1) always consists of the intercept-only model as the only candidate.

As a search_terms example, consider a reference model with formula y ~ x1 + x2 + x3. Then, to ensure that x1 is always included in the candidate models, specify search_terms = c("x1", "x1 + x2", "x1 + x3", "x1 + x2 + x3"). This search would start with y ~ 1 as the only candidate at model size 1. At model size 2, y ~ x1 would be the only candidate. At model size 3, y ~ x1 + x2 and y ~ x1 + x3 would be the two candidates. At the last model size of 4, y ~ x1 + x2 + x3 would be the only candidate. As another example, to exclude x1 from the search, specify search_terms = c("x2", "x3", "x2 + x3").

Value

An object of class vsel. The elements of this object are not meant to be accessed directly but instead via helper functions (see the vignette or type ?projpred).

Note

The case `cv_method == "LOO" && !validate_search` constitutes an exception where the search part is not cross-validated. In that case, the evaluation part is based on a PSIS-LOO CV.

For all PSIS-LOO CVs, **projpred** calls `loo::psis()` with `r_eff = NA`. This is only a problem if there was extreme autocorrelation between the MCMC iterations when the reference model was built. In those cases however, the reference model should not have been used anyway, so we don't expect **projpred**'s `r_eff = NA` to be a problem.

References

Magnusson, M., Andersen, M., Jonasson, J., and Vehtari, A. (2019). Bayesian leave-one-out cross-validation for large data. In *Proceedings of the 36th International Conference on Machine Learning*, 4244–4253. URL: https://proceedings.mlr.press/v97/magnusson19a.html.

Vehtari, A., Gelman, A., and Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, **27**(5), 1413-1432. doi:10.1007/s1122201696964.

Vehtari, A., Simpson, D., Gelman, A., Yao, Y., and Gabry, J. (2021). Pareto smoothed importance sampling. *arXiv:1507.02646*. URL: https://arxiv.org/abs/1507.02646.

See Also

`varsel()`

Examples

```
# Note: The code from this example is not executed when called via example().
# To execute it, you have to copy and paste it manually to the console.
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Variable selection with cross-validation (with small values
  # for `nterms_max`, `nclusters`, and `nclusters_pred`, but only for the
  # sake of speed in this example; this is not recommended in general):
  cvvs <- cv_varsel(fit, nterms_max = 3, nclusters = 5, nclusters_pred = 10,
                    seed = 5555)
  # Now see, for example, `?print.vsel`, `?plot.vsel`, `?suggest_size.vsel`,
  # and `?solution_terms.vsel` for possible post-processing functions.
}
```

df_binom *Binomial toy example*

Description

Binomial toy example

Usage

```
df_binom
```

Format

A simulated classification dataset containing 100 observations.

y response, 0 or 1.

x predictors, 30 in total.

Source

https://web.stanford.edu/~hastie/glmnet/glmnetData/BNExample.RData

df_gaussian *Gaussian toy example*

Description

Gaussian toy example

Usage

```
df_gaussian
```

Format

A simulated regression dataset containing 100 observations.

y response, real-valued.

x predictors, 20 in total. Mean and SD are approximately 0 and 1, respectively.

Source

https://web.stanford.edu/~hastie/glmnet/glmnetData/QSExample.RData

extend_family *Extend a family*

Description

This function adds some internally required elements to a `family` object. It is called internally by `init_refmodel()`, so you will rarely need to call it yourself.

Usage

```
extend_family(family)
```

Arguments

`family`	A `family` object.

Value

The `family` object extended in the way needed by **projpred**.

extra-families *Extra family objects*

Description

Family objects not in the set of default `family` objects.

Usage

```
Student_t(link = "identity", nu = 3)
```

Arguments

`link`	Name of the link function. In contrast to the default `family` objects, this has to be a character string here.
`nu`	Degrees of freedom for the Student-t distribution.

Value

A family object analogous to those described in `family`.

Note

Support for the `Student_t()` family is still experimental.

mesquite *Mesquite data set*

Description

The mesquite bushes yields dataset from Gelman and Hill (2006) (http://www.stat.columbia.edu/~gelman/arm/).

Usage

```
mesquite
```

Format

The response variable is the total weight (in grams) of photosynthetic material as derived from actual harvesting of the bush. The predictor variables are:

diam1 diameter of the canopy (the leafy area of the bush) in meters, measured along the longer axis of the bush.

diam2 canopy diameter measured along the shorter axis.

canopy height height of the canopy.

total height total height of the bush.

density plant unit density (# of primary stems per plant unit).

group group of measurements (0 for the first group, 1 for the second group).

Source

http://www.stat.columbia.edu/~gelman/arm/examples/mesquite/mesquite.dat

References

Gelman, A. and Hill, J. (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press. doi:10.1017/CBO9780511790942.

plot.vsel *Plot summary statistics of a variable selection*

Description

This is the `plot()` method for `vsel` objects (returned by `varsel()` or `cv_varsel()`).

Usage

```
## S3 method for class 'vsel'
plot(
  x,
  nterms_max = NULL,
  stats = "elpd",
  deltas = FALSE,
  alpha = 0.32,
  baseline = if (!inherits(x$refmodel, "datafit")) "ref" else "best",
  thres_elpd = NA,
  ...
)
```

Arguments

x	An object of class vsel (returned by varsel() or cv_varsel()).
nterms_max	Maximum submodel size for which the statistics are calculated. Note that nterms_max does not count the intercept, so use nterms_max = 0 for the intercept-only model. For plot.vsel(), nterms_max must be at least 1.
stats	One or more character strings determining which performance statistics (i.e., utilities or losses) to calculate. Available statistics are: • "elpd": (expected) sum of log predictive densities. • "mlpd": mean log predictive density, that is, "elpd" divided by the number of observations. • "mse": mean squared error. • "rmse": root mean squared error. For the corresponding standard error, bootstrapping is used. • "acc" (or its alias, "pctcorr"): classification accuracy (binomial() family only). • "auc": area under the ROC curve (binomial() family only). For the corresponding standard error, bootstrapping is used.
deltas	If TRUE, the submodel statistics are estimated as differences from the baseline model (see argument baseline) instead of estimating the actual values of the statistics.
alpha	A number determining the (nominal) coverage 1 - alpha of the normal-approximation confidence intervals. For example, alpha = 0.32 corresponds to a coverage of 68%, i.e., one-standard-error intervals (because of the normal approximation).
baseline	For summary.vsel(): Only relevant if deltas is TRUE. For plot.vsel(): Always relevant. Either "ref" or "best", indicating whether the baseline is the reference model or the best submodel found (in terms of stats[1]), respectively.
thres_elpd	Only relevant if any(stats %in% c("elpd", "mlpd")). The threshold for the ELPD difference (taking the submodel's ELPD minus the baseline model's ELPD) above which the submodel's ELPD is considered to be close enough to the baseline model's ELPD. An equivalent rule is applied in case of the MLPD. See suggest_size() for a formalization. Supplying NA deactivates this.

`...` Arguments passed to the internal function which is used for bootstrapping (if applicable; see argument `stats`). Currently, relevant arguments are `B` (the number of bootstrap samples, defaulting to `2000`) and `seed` (see `set.seed()`, defaulting to `sample.int(.Machine$integer.max, 1)`, but can also be `NA` to not call `set.seed()` at all).

Details

As long as the reference model's performance is computable, it is always shown in the plot as a dashed red horizontal line. If `baseline = "best"`, the baseline model's performance is shown as a dotted black horizontal line. If `!is.na(thres_elpd)` and `any(stats %in% c("elpd", "mlpd"))`, the value supplied to `thres_elpd` (which is automatically adapted internally in case of the MLPD or `deltas = FALSE`) is shown as a dot-dashed gray horizontal line for the reference model and, if `baseline = "best"`, as a long-dashed green horizontal line for the baseline model.

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Variable selection (here without cross-validation and with small values
  # for `nterms_max`, `nclusters`, and `nclusters_pred`, but only for the
  # sake of speed in this example; this is not recommended in general):
  vs <- varsel(fit, nterms_max = 3, nclusters = 5, nclusters_pred = 10,
               seed = 5555)
  print(plot(vs))
}
```

`pred-projection` *Predictions from a submodel (after projection)*

Description

After the projection of the reference model onto a submodel, the linear predictors (for the original dataset or new data) based on that submodel can be calculated by `proj_linpred()`. The linear predictors can also be transformed to response scale. Furthermore, `proj_linpred()` returns the corresponding log predictive density values if the new dataset contains response values. The `proj_predict()` function draws from the predictive distribution of the submodel that the reference model has been projected onto. If the projection has not been performed yet, both functions call

project() internally to perform the projection. Both functions can also handle multiple submodels at once (for objects of class vsel or objects returned by a project() call to an object of class vsel; see project()).

Usage

```
proj_linpred(
  object,
  newdata = NULL,
  offsetnew = NULL,
  weightsnew = NULL,
  filter_nterms = NULL,
  transform = FALSE,
  integrated = FALSE,
  .seed = sample.int(.Machine$integer.max, 1),
  ...
)

proj_predict(
  object,
  newdata = NULL,
  offsetnew = NULL,
  weightsnew = NULL,
  filter_nterms = NULL,
  nresample_clusters = 1000,
  .seed = sample.int(.Machine$integer.max, 1),
  ...
)
```

Arguments

object	An object returned by project() or an object that can be passed to argument object of project().
newdata	Passed to argument newdata of the reference model's extract_model_data function (see init_refmodel()). Provides the predictor (and possibly also the response) data for the new (or old) observations. May also be NULL (see argument extract_model_data of init_refmodel()). If not NULL, any NAs will trigger an error.
offsetnew	Passed to argument orhs of the reference model's extract_model_data function (see init_refmodel()). Used to get the offsets for the new (or old) observations.
weightsnew	Passed to argument wrhs of the reference model's extract_model_data function (see init_refmodel()). Used to get the weights for the new (or old) observations.
filter_nterms	Only applies if object is an object returned by project(). In that case, filter_nterms can be used to filter object for only those elements (submodels) with a number of solution terms in filter_nterms. Therefore, needs to be a numeric vector or NULL. If NULL, use all submodels.

transform
: For `proj_linpred()` only. A single logical value indicating whether the linear predictor should be transformed to response scale using the inverse-link function (`TRUE`) or not (`FALSE`).

integrated
: For `proj_linpred()` only. A single logical value indicating whether the output should be averaged across the projected posterior draws (`TRUE`) or not (`FALSE`).

.seed
: Pseudorandom number generation (PRNG) seed by which the same results can be obtained again if needed. Passed to argument `seed` of `set.seed()`, but can also be `NA` to not call `set.seed()` at all. Here, this seed is used for drawing new group-level effects in case of a multilevel submodel (however, not yet in case of a GAMM) and for drawing from the predictive distribution of the submodel(s) in case of `proj_predict()`. If a clustered projection was performed, then in `proj_predict()`, `.seed` is also used for drawing from the set of the projected clusters of posterior draws (see argument `nresample_clusters`).

...
: Arguments passed to `project()` if `object` is not already an object returned by `project()`.

nresample_clusters
: For `proj_predict()` with clustered projection only. Number of draws to return from the predictive distribution of the submodel. Not to be confused with argument `nclusters` of `project()`: `nresample_clusters` gives the number of draws (*with* replacement) from the set of clustered posterior draws after projection (with this set being determined by argument `nclusters` of `project()`).

Value

Let S_{prj} denote the number of (possibly clustered) projected posterior draws (short: the number of projected draws) and N the number of observations. Then, if the prediction is done for one submodel only (i.e., `length(nterms) == 1 || !is.null(solution_terms)` in the call to `project()`):

- `proj_linpred()` returns a `list` with elements `pred` (predictions, i.e., the linear predictors, possibly transformed to response scale) and `lpd` (log predictive densities; only calculated if `newdata` contains response values). Both elements are $S_{\text{prj}} \times N$ matrices.
- `proj_predict()` returns an $S_{\text{prj}} \times N$ matrix of predictions where S_{prj} denotes `nresample_clusters` in case of clustered projection.

If the prediction is done for more than one submodel, the output from above is returned for each submodel, giving a named `list` with one element for each submodel (the names of this `list` being the numbers of solutions terms of the submodels when counting the intercept, too).

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
```

```
  QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
)

# Projection onto an arbitrary combination of predictor terms (with a small
# value for `nclusters`, but only for the sake of speed in this example;
# this is not recommended in general):
prj <- project(fit, solution_terms = c("X1", "X3", "X5"), nclusters = 10,
               seed = 9182)

# Predictions (at the training points) from the submodel onto which the
# reference model was projected:
prjl <- proj_linpred(prj)
prjp <- proj_predict(prj, .seed = 7364)
}
```

`predict.refmodel` *Predictions or log predictive densities from a reference model*

Description

This is the `predict()` method for `refmodel` objects (returned by `get_refmodel()` or `init_refmodel()`). It offers three types of output which are all based on the reference model and new (or old) observations: Either the linear predictor on link scale, the linear predictor transformed to response scale, or the log predictive density.

Usage

```
## S3 method for class 'refmodel'
predict(
  object,
  newdata = NULL,
  ynew = NULL,
  offsetnew = NULL,
  weightsnew = NULL,
  type = "response",
  ...
)
```

Arguments

`object`	An object of class `refmodel` (returned by `get_refmodel()` or `init_refmodel()`).
`newdata`	Passed to argument `newdata` of the reference model's `extract_model_data` function (see `init_refmodel()`). Provides the predictor (and possibly also the response) data for the new (or old) observations. May also be `NULL` (see argument `extract_model_data` of `init_refmodel()`). If not `NULL`, any `NA`s will trigger an error.

ynew	If not NULL, then this needs to be a vector of new (or old) response values. See section "Value" below.
offsetnew	Passed to argument orhs of the reference model's extract_model_data function (see init_refmodel()). Used to get the offsets for the new (or old) observations.
weightsnew	Passed to argument wrhs of the reference model's extract_model_data function (see init_refmodel()). Used to get the weights for the new (or old) observations.
type	Only relevant if is.null(ynew). The scale on which the predictions are returned, either "link" or "response" (see predict.glm() but note that predict.refmodel() does not adhere to the typical R convention of a default prediction on link scale). For both scales, the predictions are averaged across the posterior draws.
...	Currently ignored.

Details

Argument weightsnew is only relevant if !is.null(ynew).

Value

Either a vector of predictions (with the scale depending on argument type) or, if !is.null(ynew), a vector of log predictive densities evaluated at ynew.

---

print.vsel *Print results (summary) of variable selection*

Description

This is the print() method for vsel objects (returned by varsel() or cv_varsel()). It displays a summary of the results of the projection predictive variable selection by first calling summary.vsel() and then print.vselsummary().

Usage

```
## S3 method for class 'vsel'
print(x, ...)
```

Arguments

x	An object of class vsel (returned by varsel() or cv_varsel()).
...	Further arguments passed to summary.vsel() (apart from argument digits which is passed to print.vselsummary()).

Value

The output of summary.vsel() (invisible).

print.vselsummary *Print summary of variable selection*

Description

This is the `print()` method for summary objects created by `summary.vsel()`. It displays a summary of the results of the projection predictive variable selection.

Usage

```
## S3 method for class 'vselsummary'
print(x, digits = 1, ...)
```

Arguments

`x`	An object of class `vselsummary`.
`digits`	Number of decimal places to be reported.
`...`	Currently ignored.

Value

The output of `summary.vsel()` (invisible).

project *Projection onto submodel(s)*

Description

Project the posterior of the reference model onto the parameter space of a single submodel consisting of a specific combination of predictor terms or (after variable selection) onto the parameter space of a single or multiple submodels of specific sizes.

Usage

```
project(
  object,
  nterms = NULL,
  solution_terms = NULL,
  refit_prj = TRUE,
  ndraws = 400,
  nclusters = NULL,
  seed = sample.int(.Machine$integer.max, 1),
  regul = 1e-04,
  ...
)
```

Arguments

object	An object which can be used as input to get_refmodel() (in particular, objects of class refmodel).
nterms	Only relevant if object is of class vsel (returned by varsel() or cv_varsel()). Ignored if !is.null(solution_terms). Number of terms for the submodel (the corresponding combination of predictor terms is taken from object). If a numeric vector, then the projection is performed for each element of this vector. If NULL (and is.null(solution_terms)), then the value suggested by the variable selection is taken (see function suggest_size()). Note that nterms does not count the intercept, so use nterms = 0 for the intercept-only model.
solution_terms	If not NULL, then this needs to be a character vector of predictor terms for the submodel onto which the projection will be performed. Argument nterms is ignored in that case. For an object which is not of class vsel, solution_terms must not be NULL.
refit_prj	A single logical value indicating whether to fit the submodels (again) (TRUE) or to retrieve the fitted submodels from object (FALSE). For an object which is not of class vsel, refit_prj must be TRUE. Note that currently, refit_prj = FALSE requires some caution, see GitHub issues #168 and #211.
ndraws	Only relevant if refit_prj is TRUE. Number of posterior draws to be projected. Ignored if nclusters is not NULL or if the reference model is of class datafit (in which case one cluster is used). If both (nclusters and ndraws) are NULL, the number of posterior draws from the reference model is used for ndraws. See also section "Details" below.
nclusters	Only relevant if refit_prj is TRUE. Number of clusters of posterior draws to be projected. Ignored if the reference model is of class datafit (in which case one cluster is used). For the meaning of NULL, see argument ndraws. See also section "Details" below.
seed	Pseudorandom number generation (PRNG) seed by which the same results can be obtained again if needed. Passed to argument seed of set.seed(), but can also be NA to not call set.seed() at all. Here, this seed is used for clustering the reference model's posterior draws (if !is.null(nclusters)) and for drawing new group-level effects when predicting from a multilevel submodel (however, not yet in case of a GAMM).
regul	A number giving the amount of ridge regularization when projecting onto (i.e., fitting) submodels which are GLMs. Usually there is no need for regularization, but sometimes we need to add some regularization to avoid numerical problems.
...	Arguments passed to get_refmodel() (if get_refmodel() is actually used; see argument object) as well as to the divergence minimizer (if refit_prj is TRUE).

Details

Arguments ndraws and nclusters are automatically truncated at the number of posterior draws in the reference model (which is 1 for datafits). Using less draws or clusters in ndraws or nclusters than posterior draws in the reference model may result in slightly inaccurate projection performance. Increasing these arguments affects the computation time linearly.

Note that if project() is applied to output from cv_varsel(), then refit_prj = FALSE will take the results from the *full-data* search.

Value

If the projection is performed onto a single submodel (i.e., length(nterms) == 1 || !is.null(solution_terms)), an object of class projection which is a list containing the following elements:

dis Projected draws for the dispersion parameter.

kl The Kullback-Leibler (KL) divergence from the submodel to the reference model. Note that in case of the Gaussian family, this is not the actual KL divergence but merely a proxy.

weights Weights for the projected draws.

solution_terms A character vector of the submodel's predictor terms, ordered in the way in which the terms were added to the submodel.

submodl A list containing the submodel fits (one fit per projected draw).

p_type A single logical value indicating whether the reference model's posterior draws have been clustered for the projection (TRUE) or not (FALSE).

refmodel The reference model object.

If the projection is performed onto more than one submodel, the output from above is returned for each submodel, giving a list with one element for each submodel.

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Variable selection (here without cross-validation and with small values
  # for `nterms_max`, `nclusters`, and `nclusters_pred`, but only for the
  # sake of speed in this example; this is not recommended in general):
  vs <- varsel(fit, nterms_max = 3, nclusters = 5, nclusters_pred = 10,
               seed = 5555)

  # Projection onto the best submodel with 2 predictor terms (with a small
  # value for `nclusters`, but only for the sake of speed in this example;
  # this is not recommended in general):
  prj_from_vs <- project(vs, nterms = 2, nclusters = 10, seed = 9182)

  # Projection onto an arbitrary combination of predictor terms (with a small
  # value for `nclusters`, but only for the sake of speed in this example;
  # this is not recommended in general):
```

```
  prj <- project(fit, solution_terms = c("X1", "X3", "X5"), nclusters = 10,
                 seed = 9182)
}
```

---

refmodel-init-get *Reference model structure*

Description

Function `get_refmodel()` is a generic function for creating the reference model structure from a specific `object`. The methods for `get_refmodel()` usually call `init_refmodel()` which is the underlying workhorse (and may also be used directly without a call to `get_refmodel()`). Some arguments are for K-fold cross-validation (K-fold CV) only; see `cv_varsel()` for the use of K-fold CV in **projpred**.

Usage

```
get_refmodel(object, ...)

## S3 method for class 'refmodel'
get_refmodel(object, ...)

## S3 method for class 'vsel'
get_refmodel(object, ...)

## Default S3 method:
get_refmodel(object, formula, family = NULL, ...)

## S3 method for class 'stanreg'
get_refmodel(object, ...)

init_refmodel(
  object,
  data,
  formula,
  family,
  ref_predfun = NULL,
  div_minimizer = NULL,
  proj_predfun = NULL,
  extract_model_data,
  cvfun = NULL,
  cvfits = NULL,
  dis = NULL,
  cvrefbuilder = NULL,
  ...
)
```

Arguments

object	Object from which the reference model is created. For init_refmodel(), an object on which the functions from arguments extract_model_data and ref_predfun can be applied, with a NULL object being treated specially (see section "Value" below). For get_refmodel.default(), an object on which function family() can be applied to retrieve the family (if argument family is NULL), additionally to the properties required for init_refmodel(). For non-default methods of get_refmodel(), an object of the corresponding class.
...	For get_refmodel.default() and get_refmodel.stanreg(): arguments passed to init_refmodel(). For the get_refmodel() generic: arguments passed to the appropriate method. Else: ignored.
formula	Reference model's formula. For general information on formulas in R, see formula. For multilevel formulas, see also package **lme4** (in particular, functions lme4::lmer() and lme4::glmer()). For additive formulas, see also packages **mgcv** (in particular, function mgcv::gam()) and **gamm4** (in particular, function gamm4::gamm4()) as well as the notes in section "Formula terms" below.
family	A family object representing the observational model (i.e., the distributional family for the response). May be NULL for get_refmodel.default() in which case the family is retrieved from object.
data	Data used for fitting the reference model. Any contrasts attributes of the dataset's columns are silently removed.
ref_predfun	Prediction function for the linear predictor of the reference model, including offsets (if existing). See also section "Arguments ref_predfun, proj_predfun, and div_minimizer" below. If object is NULL, ref_predfun is ignored and an internal default is used instead.
div_minimizer	A function for minimizing the Kullback-Leibler (KL) divergence from a submodel to the reference model (i.e., for performing the projection of the reference model onto a submodel). The output of div_minimizer is used, e.g., by proj_predfun's argument fits. See also section "Arguments ref_predfun, proj_predfun, and div_minimizer" below.
proj_predfun	Prediction function for the linear predictor of a submodel onto which the reference model is projected. See also section "Arguments ref_predfun, proj_predfun, and div_minimizer" below.
extract_model_data	A function for fetching some variables (response, observation weights, offsets) from the original dataset (i.e., the dataset used for fitting the reference model) or from a new dataset. See also section "Argument extract_model_data" below.
cvfun	For K-fold CV only. A function that, given a fold indices vector, fits the reference model separately for each fold and returns the K model fits as a list. Each of the K model fits needs to be a list. If object is NULL, cvfun may be NULL for using an internal default. Only one of cvfits and cvfun needs to be provided (for K-fold CV). Note that cvfits takes precedence over cvfun, i.e., if both are provided, cvfits is used.
cvfits	For K-fold CV only. A list containing a sub-list called fits containing the K model fits from which reference model structures are created. The cvfits

list (i.e., the super-list) needs to have attributes K and folds: K has to be a single integer giving the number of folds and folds has to be an integer vector giving the fold indices (one fold index per observation). Each element of cvfits$fits (i.e., each of the K model fits) needs to be a list. Only one of cvfits and cvfun needs to be provided (for K-fold CV). Note that cvfits takes precedence over cvfun, i.e., if both are provided, cvfits is used.

dis — A vector of posterior draws for the dispersion parameter (if existing). May be NULL if the model has no dispersion parameter or if the model does have a dispersion parameter, but object is NULL (in which case 0 is used for dis). Note that for the gaussian() family, dis is the standard deviation, not the variance.

cvrefbuilder — For K-fold CV only. A function that, given a reference model fit for fold $k \in \{1, ..., K\}$ (this model fit is the k-th element of the return value of cvfun or the k-th element of cvfits$fits, extended by elements omitted (containing the indices of the left-out observations in that fold) and projpred_k (containing the integer k)), returns an object of the same type as init_refmodel() does. Argument cvrefbuilder may be NULL for using an internal default: get_refmodel() if object is not NULL and a function calling init_refmodel() appropriately (with the assumption dis = 0) if object is NULL.

Value

An object that can be passed to all the functions that take the reference model fit as the first argument, such as varsel(), cv_varsel(), project(), proj_linpred(), and proj_predict(). Usually, the returned object is of class refmodel. However, if object is NULL, the returned object is of class datafit as well as of class refmodel (with datafit being first). Objects of class datafit are handled differently at several places throughout this package.

Formula terms

For additive models (still an experimental feature), only mgcv::s() and mgcv::t2() are currently supported as smooth terms. Furthermore, these need to be called without any arguments apart from the predictor names (symbols). For example, for smoothing the effect of a predictor x, only s(x) or t2(x) are allowed. As another example, for smoothing the joint effect of two predictors x and z, only s(x, z) or t2(x, z) are allowed (and analogously for higher-order joint effects, e.g., of three predictors).

Arguments ref_predfun, proj_predfun, and div_minimizer

Arguments ref_predfun, proj_predfun, and div_minimizer may be NULL for using an internal default. Otherwise, let N denote the number of observations (in case of CV, these may be reduced to each fold), S_{ref} the number of posterior draws for the reference model's parameters, and S_{prj} the number of (possibly clustered) parameter draws for projection (short: the number of projected draws). Then the functions supplied to these arguments need to have the following prototypes:

- ref_predfun: ref_predfun(fit, newdata = NULL) where:
  - fit accepts the reference model fit as given in argument object (but possibly re-fitted to a subset of the observations, as done in K-fold CV).
  - newdata accepts either NULL (for using the original dataset, typically stored in fit) or data for new observations (at least in the form of a data.frame).

- `proj_predfun`: `proj_predfun(fits, newdata)` where:
  - `fits` accepts a `list` of length S_{prj} containing this number of submodel fits. This `list` is the same as that returned by `project()` in its output element `submodl` (which in turn is the same as the return value of `div_minimizer`, except if `project()` was used with an `object` of class `vsel` based on an L1 search as well as with `refit_prj = FALSE`).
  - `newdata` accepts data for new observations (at least in the form of a `data.frame`).
- `div_minimizer` does not need to have a specific prototype, but it needs to be able to be called with the following arguments:
  - `formula` accepts either a standard `formula` with a single response (if $S_{\text{prj}} = 1$) or a `formula` with $S_{\text{prj}} > 1$ response variables `cbind()`-ed on the left-hand side in which case the projection has to be performed for each of the response variables separately.
  - `data` accepts a `data.frame` to be used for the projection.
  - `family` accepts a `family` object.
  - `weights` accepts either observation weights (at least in the form of a numeric vector) or `NULL` (for using a vector of ones as weights).
  - `projpred_var` accepts an $N \times S_{\text{prj}}$ matrix of predictive variances (necessary for **projpred**'s internal GLM fitter).
  - `projpred_regul` accepts a single numeric value as supplied to argument `regul` of `project()`, for example.
  - `...` accepts further arguments specified by the user.

The return value of these functions needs to be:

- `ref_predfun`: an $N \times S_{\text{ref}}$ matrix.
- `proj_predfun`: an $N \times S_{\text{prj}}$ matrix.
- `div_minimizer`: a `list` of length S_{prj} containing this number of submodel fits.

Argument `extract_model_data`

The function supplied to argument `extract_model_data` needs to have the prototype

```
extract_model_data(object, newdata, wrhs = NULL, orhs = NULL, extract_y = TRUE)
```

where:

- `object` accepts the reference model fit as given in argument `object` (but possibly re-fitted to a subset of the observations, as done in K-fold CV).
- `newdata` accepts either `NULL` (for using the original dataset, typically stored in `object`) or data for new observations (at least in the form of a `data.frame`).
- `wrhs` accepts at least either `NULL` (for using a vector of ones) or a right-hand side formula consisting only of the variable in `newdata` containing the weights.
- `orhs` accepts at least either `NULL` (for using a vector of zeros) or a right-hand side formula consisting only of the variable in `newdata` containing the offsets.
- `extract_y` accepts a single logical value indicating whether output element `y` (see below) shall be `NULL` (`TRUE`) or not (`FALSE`).

The return value of `extract_model_data` needs to be a `list` with elements `y`, `weights`, and `offset`, each being a numeric vector containing the data for the response, the observation weights, and the offsets, respectively. An exception is that `y` may also be `NULL` (depending on argument `extract_y`) or a `factor`.

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Define the reference model explicitly:
  ref <- get_refmodel(fit)
  print(class(ref)) # gives `"refmodel"`
  # Now see, for example, `?varsel`, `?cv_varsel`, and `?project` for
  # possible post-processing functions. Most of the post-processing functions
  # call get_refmodel() internally at the beginning, so you will rarely need
  # to call get_refmodel() yourself.

  # A custom reference model which may be used in a variable selection where
  # the candidate predictors are not a subset of those used for the reference
  # model's predictions:
  ref_cust <- init_refmodel(
    fit,
    data = dat_gauss,
    formula = y ~ X6 + X7,
    family = gaussian(),
    extract_model_data = function(object, newdata = NULL, wrhs = NULL,
                                  orhs = NULL, extract_y = TRUE) {
      if (!extract_y) {
        resp_form <- NULL
      } else {
        resp_form <- ~ y
      }

      if (is.null(newdata)) {
        newdata <- dat_gauss
      }

      args <- projpred:::nlist(object, newdata, wrhs, orhs, resp_form)
      return(projpred::do_call(projpred:::.extract_model_data, args))
    },
    cvfun = function(folds) {
      kfold(
```

```
        fit, K = max(folds), save_fits = TRUE, folds = folds, cores = 1
      )$fits[, "fit"]
    },
    dis = as.matrix(fit)[, "sigma"]
  )
  # Now, the post-processing functions mentioned above (for example,
  # varsel(), cv_varsel(), and project()) may be applied to `ref_cust`.
}
```

`solution_terms` *Retrieve predictor solution path or predictor combination*

Description

This function retrieves the "solution terms" from an object. For `vsel` objects (returned by `varsel()` or `cv_varsel()`), this is the predictor solution path of the variable selection. For `projection` objects (returned by `project()`, possibly as elements of a `list`), this is the predictor combination onto which the projection was performed.

Usage

```
solution_terms(object, ...)

## S3 method for class 'vsel'
solution_terms(object, ...)

## S3 method for class 'projection'
solution_terms(object, ...)
```

Arguments

`object`	The object from which to retrieve the solution terms. Possible classes may be inferred from the names of the corresponding methods (see also the description).
`...`	Currently ignored.

Value

A character vector of solution terms.

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
```

```
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Variable selection (here without cross-validation and with small values
  # for `nterms_max`, `nclusters`, and `nclusters_pred`, but only for the
  # sake of speed in this example; this is not recommended in general):
  vs <- varsel(fit, nterms_max = 3, nclusters = 5, nclusters_pred = 10,
               seed = 5555)
  print(solution_terms(vs))

  # Projection onto an arbitrary combination of predictor terms (with a small
  # value for `nclusters`, but only for the sake of speed in this example;
  # this is not recommended in general):
  prj <- project(fit, solution_terms = c("X1", "X3", "X5"), nclusters = 10,
                 seed = 9182)
  print(solution_terms(prj)) # gives `c("X1", "X3", "X5")`
}
```

---

suggest_size *Suggest submodel size*

Description

This function can suggest an appropriate submodel size based on a decision rule described in section "Details" below. Note that this decision is quite heuristic and should be interpreted with caution. It is recommended to examine the results via `plot.vsel()` and/or `summary.vsel()` and to make the final decision based on what is most appropriate for the problem at hand.

Usage

```
suggest_size(object, ...)

## S3 method for class 'vsel'
suggest_size(
  object,
  stat = "elpd",
  pct = 0,
  type = "upper",
  thres_elpd = NA,
  warnings = TRUE,
  ...
)
```

Arguments

object	An object of class vsel (returned by varsel() or cv_varsel()).
...	Arguments passed to summary.vsel(), except for object, stats (which is set to stat), type, and deltas (which is set to TRUE). See section "Details" below for some important arguments which may be passed here.
stat	Performance statistic (i.e., utility or loss) used for the decision. See argument stats of summary.vsel() for possible choices.
pct	A number giving the relative proportion (*not* percents) between baseline model and null model utilities one is willing to sacrifice. See section "Details" below for more information.
type	Either "upper" or "lower" determining whether the decision is based on the upper or lower confidence interval bound, respectively. See section "Details" below for more information.
thres_elpd	Only relevant if stat %in% c("elpd", "mlpd"). The threshold for the ELPD difference (taking the submodel's ELPD minus the baseline model's ELPD) above which the submodel's ELPD is considered to be close enough to the baseline model's ELPD. An equivalent rule is applied in case of the MLPD. See section "Details" for a formalization. Supplying NA deactivates this.
warnings	Mainly for internal use. A single logical value indicating whether to throw warnings if automatic suggestion fails. Usually there is no reason to set this to FALSE.

Details

In general (beware of special extensions below), the suggested model size is the smallest model size $k \in \{0, 1, ..., \texttt{nterms\_max}\}$ for which either the lower or upper bound (depending on argument type) of the normal-approximation confidence interval (with nominal coverage 1 - alpha; see argument alpha of summary.vsel()) for $U_k - U_{\text{base}}$ (with U_k denoting the k-th submodel's true utility and U_{base} denoting the baseline model's true utility) falls above (or is equal to)

$$\texttt{pct} \cdot (u_0 - u_{\text{base}})$$

where u_0 denotes the null model's estimated utility and u_{base} the baseline model's estimated utility. The baseline model is either the reference model or the best submodel found (see argument baseline of summary.vsel()).

If !is.na(thres_elpd) and stat = "elpd", the decision rule above is extended: The suggested model size is then the smallest model size k fulfilling the rule above *or* $u_k - u_{\text{base}} > \texttt{thres\_elpd}$. Correspondingly, in case of stat = "mlpd" (and !is.na(thres_elpd)), the suggested model size is the smallest model size k fulfilling the rule above *or* $u_k - u_{\text{base}} > \frac{\texttt{thres\_elpd}}{N}$ with N denoting the number of observations.

For example (disregarding the special extensions in case of stat = "elpd" or stat = "mlpd"), alpha = 0.32, pct = 0, and type = "upper" means that we select the smallest model size for which the upper bound of the 68% confidence interval for $U_k - U_{\text{base}}$ exceeds (or is equal to) zero, that is, for which the submodel's utility estimate is at most one standard error smaller than the baseline model's utility estimate (with that standard error referring to the utility *difference*).

Note

Loss statistics like the root mean-squared error (RMSE) and the mean-squared error (MSE) are converted to utilities by multiplying them by `-1`, so a call such as `suggest_size(object, stat = "rmse", type = "upper")` finds the smallest model size whose upper confidence interval bound for the *negative* RMSE or MSE exceeds the cutoff (or, equivalently, has the lower confidence interval bound for the RMSE or MSE below the cutoff). This is done to make the interpretation of argument `type` the same regardless of argument `stat`.

The intercept is not counted by `suggest_size()`, so a suggested size of zero stands for the intercept-only model.

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Variable selection (here without cross-validation and with small values
  # for `nterms_max`, `nclusters`, and `nclusters_pred`, but only for the
  # sake of speed in this example; this is not recommended in general):
  vs <- varsel(fit, nterms_max = 3, nclusters = 5, nclusters_pred = 10,
               seed = 5555)
  print(suggest_size(vs))
}
```

`summary.vsel` *Summary statistics of a variable selection*

Description

This is the `summary()` method for `vsel` objects (returned by `varsel()` or `cv_varsel()`).

Usage

```
## S3 method for class 'vsel'
summary(
  object,
  nterms_max = NULL,
  stats = "elpd",
  type = c("mean", "se", "diff", "diff.se"),
```

```
  deltas = FALSE,
  alpha = 0.32,
  baseline = if (!inherits(object$refmodel, "datafit")) "ref" else "best",
  ...
)
```

Arguments

object	An object of class vsel (returned by varsel() or cv_varsel()).
nterms_max	Maximum submodel size for which the statistics are calculated. Note that nterms_max does not count the intercept, so use nterms_max = 0 for the intercept-only model. For plot.vsel(), nterms_max must be at least 1.
stats	One or more character strings determining which performance statistics (i.e., utilities or losses) to calculate. Available statistics are: • "elpd": (expected) sum of log predictive densities. • "mlpd": mean log predictive density, that is, "elpd" divided by the number of observations. • "mse": mean squared error. • "rmse": root mean squared error. For the corresponding standard error, bootstrapping is used. • "acc" (or its alias, "pctcorr"): classification accuracy (binomial() family only). • "auc": area under the ROC curve (binomial() family only). For the corresponding standard error, bootstrapping is used.
type	One or more items from "mean", "se", "lower", "upper", "diff", and "diff.se" indicating which of these to compute for each item from stats (mean, standard error, lower and upper confidence interval bounds, mean difference to the corresponding statistic of the reference model, and standard error of this difference, respectively). The confidence interval bounds belong to normal-approximation confidence intervals with (nominal) coverage 1 - alpha. Items "diff" and "diff.se" are only supported if deltas is FALSE.
deltas	If TRUE, the submodel statistics are estimated as differences from the baseline model (see argument baseline) instead of estimating the actual values of the statistics.
alpha	A number determining the (nominal) coverage 1 - alpha of the normal-approximation confidence intervals. For example, alpha = 0.32 corresponds to a coverage of 68%, i.e., one-standard-error intervals (because of the normal approximation).
baseline	For summary.vsel(): Only relevant if deltas is TRUE. For plot.vsel(): Always relevant. Either "ref" or "best", indicating whether the baseline is the reference model or the best submodel found (in terms of stats[1]), respectively.
...	Arguments passed to the internal function which is used for bootstrapping (if applicable; see argument stats). Currently, relevant arguments are B (the number of bootstrap samples, defaulting to 2000) and seed (see set.seed(), defaulting to sample.int(.Machine$integer.max, 1), but can also be NA to not call set.seed() at all).

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)

  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Variable selection (here without cross-validation and with small values
  # for `nterms_max`, `nclusters`, and `nclusters_pred`, but only for the
  # sake of speed in this example; this is not recommended in general):
  vs <- varsel(fit, nterms_max = 3, nclusters = 5, nclusters_pred = 10,
               seed = 5555)
  print(summary(vs))
}
```

`varsel` *Variable selection (without cross-validation)*

Description

Perform the projection predictive variable selection for GLMs, GLMMs, GAMs, and GAMMs. This variable selection consists of a *search* part and an *evaluation* part. The search part determines the solution path, i.e., the best submodel for each submodel size (number of predictor terms). The evaluation part determines the predictive performance of the submodels along the solution path.

Usage

```
varsel(object, ...)

## Default S3 method:
varsel(object, ...)

## S3 method for class 'refmodel'
varsel(
  object,
  d_test = NULL,
  method = NULL,
  ndraws = NULL,
  nclusters = 20,
  ndraws_pred = 400,
  nclusters_pred = NULL,
```

```
  refit_prj = !inherits(object, "datafit"),
  nterms_max = NULL,
  verbose = TRUE,
  lambda_min_ratio = 1e-05,
  nlambda = 150,
  thresh = 1e-06,
  regul = 1e-04,
  penalty = NULL,
  search_terms = NULL,
  seed = sample.int(.Machine$integer.max, 1),
  ...
)
```

Arguments

object	An object of class refmodel (returned by get_refmodel() or init_refmodel()) or an object that can be passed to argument object of get_refmodel().
...	Arguments passed to get_refmodel() as well as to the divergence minimizer (during a forward search and also during the evaluation part, but the latter only if refit_prj is TRUE).
d_test	A list of the structure outlined in section "Argument d_test" below, providing test data for evaluating the predictive performance of the submodels as well as of the reference model. If NULL, the training data is used.
method	The method for the search part. Possible options are "L1" for L1 search and "forward" for forward search. If NULL, then internally, "L1" is used, except if the reference model has multilevel or additive terms or if !is.null(search_terms). See also section "Details" below.
ndraws	Number of posterior draws used in the search part. Ignored if nclusters is not NULL or in case of L1 search (because L1 search always uses a single cluster). If both (nclusters and ndraws) are NULL, the number of posterior draws from the reference model is used for ndraws. See also section "Details" below.
nclusters	Number of clusters of posterior draws used in the search part. Ignored in case of L1 search (because L1 search always uses a single cluster). For the meaning of NULL, see argument ndraws. See also section "Details" below.
ndraws_pred	Only relevant if refit_prj is TRUE. Number of posterior draws used in the evaluation part. Ignored if nclusters_pred is not NULL. If both (nclusters_pred and ndraws_pred) are NULL, the number of posterior draws from the reference model is used for ndraws_pred. See also section "Details" below.
nclusters_pred	Only relevant if refit_prj is TRUE. Number of clusters of posterior draws used in the evaluation part. For the meaning of NULL, see argument ndraws_pred. See also section "Details" below.
refit_prj	A single logical value indicating whether to fit the submodels along the solution path again (TRUE) or to retrieve their fits from the search part (FALSE) before using those (re-)fits in the evaluation part.
nterms_max	Maximum number of predictor terms until which the search is continued. If NULL, then min(19, D) is used where D is the number of terms in the reference

model (or in search_terms, if supplied). Note that nterms_max does not count the intercept, so use nterms_max = 0 for the intercept-only model. (Correspondingly, D above does not count the intercept.)

verbose — A single logical value indicating whether to print out additional information during the computations.

lambda_min_ratio — Only relevant for L1 search. Ratio between the smallest and largest lambda in the L1-penalized search. This parameter essentially determines how long the search is carried out, i.e., how large submodels are explored. No need to change this unless the program gives a warning about this.

nlambda — Only relevant for L1 search. Number of values in the lambda grid for L1-penalized search. No need to change this unless the program gives a warning about this.

thresh — Only relevant for L1 search. Convergence threshold when computing the L1 path. Usually, there is no need to change this.

regul — A number giving the amount of ridge regularization when projecting onto (i.e., fitting) submodels which are GLMs. Usually there is no need for regularization, but sometimes we need to add some regularization to avoid numerical problems.

penalty — Only relevant for L1 search. A numeric vector determining the relative penalties or costs for the predictors. A value of 0 means that those predictors have no cost and will therefore be selected first, whereas Inf means those predictors will never be selected. If NULL, then 1 is used for each predictor.

search_terms — Only relevant for forward search. A custom character vector of predictor term blocks to consider for the search. Section "Details" below describes more precisely what "predictor term block" means. The intercept ("1") is always included internally via union(), so there's no difference between including it explicitly or omitting it. The default search_terms considers all the terms in the reference model's formula.

seed — Pseudorandom number generation (PRNG) seed by which the same results can be obtained again if needed. Passed to argument seed of set.seed(), but can also be NA to not call set.seed() at all. Here, this seed is used for clustering the reference model's posterior draws (if !is.null(nclusters)) and for drawing new group-level effects when predicting from a multilevel submodel (however, not yet in case of a GAMM).

Details

Arguments ndraws, nclusters, nclusters_pred, and ndraws_pred are automatically truncated at the number of posterior draws in the reference model (which is 1 for datafits). Using less draws or clusters in ndraws, nclusters, nclusters_pred, or ndraws_pred than posterior draws in the reference model may result in slightly inaccurate projection performance. Increasing these arguments affects the computation time linearly.

For argument method, there are some restrictions: For a reference model with multilevel or additive formula terms, only the forward search is available. Furthermore, argument search_terms requires a forward search to take effect.

L1 search is faster than forward search, but forward search may be more accurate. Furthermore, forward search may find a sparser model with comparable performance to that found by L1 search, but it may also start overfitting when more predictors are added.

An L1 search may select interaction terms before the corresponding main terms are selected. If this is undesired, choose the forward search instead.

The elements of the `search_terms` character vector don't need to be individual predictor terms. Instead, they can be building blocks consisting of several predictor terms connected by the `+` symbol. To understand how these building blocks works, it is important to know how **projpred**'s forward search works: It starts with an empty vector `chosen` which will later contain already selected predictor terms. Then, the search iterates over model sizes $j \in \{1, ..., J\}$. The candidate models at model size j are constructed from those elements from `search_terms` which yield model size j when combined with the `chosen` predictor terms. Note that sometimes, there may be no candidate models for model size j. Also note that internally, `search_terms` is expanded to include the intercept (`"1"`), so the first step of the search (model size 1) always consists of the intercept-only model as the only candidate.

As a `search_terms` example, consider a reference model with formula `y ~ x1 + x2 + x3`. Then, to ensure that `x1` is always included in the candidate models, specify `search_terms = c("x1", "x1 + x2", "x1 + x3", "x1 + x2 + x3")`. This search would start with `y ~ 1` as the only candidate at model size 1. At model size 2, `y ~ x1` would be the only candidate. At model size 3, `y ~ x1 + x2` and `y ~ x1 + x3` would be the two candidates. At the last model size of 4, `y ~ x1 + x2 + x3` would be the only candidate. As another example, to exclude `x1` from the search, specify `search_terms = c("x2", "x3", "x2 + x3")`.

Value

An object of class `vsel`. The elements of this object are not meant to be accessed directly but instead via helper functions (see the vignette or type `?projpred`).

Argument `d_test`

If not `NULL`, then `d_test` needs to be a `list` with the following elements:

- `data`: a `data.frame` containing the predictor variables for the test set.
- `offset`: a numeric vector containing the offset values for the test set (if there is no offset, use a vector of zeros).
- `weights`: a numeric vector containing the observation weights for the test set (if there are no observation weights, use a vector of ones).
- `y`: a numeric vector containing the response values for the test set.

See Also

`cv_varsel()`

Examples

```
if (requireNamespace("rstanarm", quietly = TRUE)) {
  # Data:
  dat_gauss <- data.frame(y = df_gaussian$y, df_gaussian$x)
```

```
  # The "stanreg" fit which will be used as the reference model (with small
  # values for `chains` and `iter`, but only for technical reasons in this
  # example; this is not recommended in general):
  fit <- rstanarm::stan_glm(
    y ~ X1 + X2 + X3 + X4 + X5, family = gaussian(), data = dat_gauss,
    QR = TRUE, chains = 2, iter = 500, refresh = 0, seed = 9876
  )

  # Variable selection (here without cross-validation and with small values
  # for `nterms_max`, `nclusters`, and `nclusters_pred`, but only for the
  # sake of speed in this example; this is not recommended in general):
  vs <- varsel(fit, nterms_max = 3, nclusters = 5, nclusters_pred = 10,
               seed = 5555)
  # Now see, for example, `?print.vsel`, `?plot.vsel`, `?suggest_size.vsel`,
  # and `?solution_terms.vsel` for possible post-processing functions.
}
```

Index