Package ‘vcrpart’

May 11, 2024

Type Package

Title Tree-Based Varying Coefficient Regression for Generalized Linear and Ordinal Mixed Models

Version 1.0-5

Date 2024-05-06

Author Reto Burgin [aut, cre] (<https://orcid.org/0000-0002-6212-1567>), Gilbert Ritschard [ctb] (<https://orcid.org/0000-0001-7776-0903>)

Maintainer Reto Burgin <rbuergin@gmx.ch>

Description Recursive partitioning for varying coefficient generalized linear models and ordinal linear mixed models. Special features are coefficient-wise partitioning, non-varying coefficients and partitioning of time-varying variables in longitudinal regression. A description of a part of this package was published by Burgin and Ritschard (2017) <doi:10.18637/jss.v080.i06>.

Depends R (>= 3.1.0), parallel, partykit

Imports stats, grid, graphics, methods, nlme (>= 3.1-123), rpart, formula.tools, numDeriv, ucminf, zoo, sandwich, strucchange

Suggests xtable, mlbench, Ecdat, RWeka

LazyLoad yes

NeedsCompilation yes

Repository CRAN

Date/Publication 2024-05-11 21:10:02 UTC

R topics documented:

`contr.wsum` *Contrast matrices*

Description

Returns a category-weighted contrast matrix

Usage

```
contr.wsum(x, weights = rep.int(1.0, length(x)), sparse = FALSE)
```

Arguments

`x`	a factor vector
`weights`	a vector of weights with the same length as `x`.
`sparse`	ogical indicating if the result should be sparse (of class `dgCMatrix`), using package **Matrix**.

Details

Computes a contrast matrix similar to `contr.sum`. The reference category is however weighted by the sum of weights of the other categories.

Value

A matrix with `nlevels(x)` rows and `nlevels(x)- 1` columns.

Author(s)

Reto Burgin

See Also

contr.sum

Examples

```
x <- factor(rep(LETTERS[1:3], c(10, 20, 30)))
contr.wsum(x) # standard call
contr.wsum(x, sparse = TRUE) # using a sparse matrix
```

---

fvcm *Bagging and Random Forests based on* tvcm

Description

Bagging (Breiman, 1996) and Random Forest (Breiman, 2001) ensemble algorithms for tvcm.

Usage

```
fvcm(..., control = fvcm_control())

fvcm_control(maxstep = 10, minsize = 10,
             folds = folds_control("subsampling", K = 100),
             mtry = 5, sctest = FALSE, alpha = 1.0,
             mindev = 0.0, verbose = TRUE, ...)

fvcolmm(..., family = cumulative(), control = fvcolmm_control())

fvcolmm_control(maxstep = 10, minsize = 20,
                folds = folds_control("subsampling", K = 100),
                mtry = 5, sctest = TRUE, alpha = 1.0,
                nimpute = 1, verbose = TRUE, ...)

fvcglm(..., family, control = fvcglm_control())

fvcglm_control(maxstep = 10, minsize = 10,
               folds = folds_control("subsampling", K = 100),
               mtry = 5, mindev = 0,
               verbose = TRUE, ...)
```

Arguments

`...`	for `fvcm`, `fvcolmm` and `fvcglm` arguments to be passed to `tvcm`. This includes at least the arguments `formula`, `data` and `family`, see examples below. For `fvcm_control` further control arguments to be passed to `tvcm_control`. For `fvcolmm_control` and `fvcglm_control` further control arguments to be passed to `fvcm_control`
.	
`control`	a list of control parameters as produced by `fvcm_control`.
`family`	the model family, e.g., `binomial` or `cumulative`.
`maxstep`	integer. The maximum number of steps for when growing individual trees.
`folds`	a list of parameters to control the extraction of subsets, as created by `folds_control`.
`mtry`	positive integer scalar. The number of combinations of partitions, nodes and variables to be randomly sampled as candidates in each iteration.
`sctest`	logical scalar. Defines whether coefficient constancy tests should be used for the variable and node selection in each iteration.
`mindev, alpha`	these parameters are merely specified to disable the default stopping rules for `tvcm`. See also `tvcm_control` for details.
`minsize, nimpute`	special parameter settings for `fvcolmm`. The minimum node size is set to the default of `tvcolmm`. The default `nimpute` deactivates the imputation procedure in cases of unbalanced data.
`verbose`	logical. Should information about the fitting process be printed to the screen?

Details

Implements the *Bagging* (Breiman, 1996) and *Random Forests* (Breiman, 2001) ensemble algorithms for `tvcm`. The method consist in growing multiple trees by using `tvcm` and aggregating the fitted coefficient functions in the scale of the predictor function. To enable bagging, use `mtry = Inf` in `fvcm_control`.

`fvcolmm` and `fvcglm` are the extensions for `tvcolmm` and `tvcglm`.

`fvcm_control` is a wrapper of `tvcm_control` and the arguments indicated specify modified defaults and parameters for randomizing split selections. Notice that, relative to `tvcm_control`, also the `cv` `prune` arguments are internally disabled. The default arguments for `alpha` and `maxoverstep` essentially disable the stopping rules of `tvcm`, where the argument `maxstep` (the number of iterations i.e. the maximum number of splits) fully controls the stopping. The parameter `mtry` controls the randomization for selecting combinations of partitions, nodes and variables for splitting. The default of `mtry = 5` is arbitrary.

Value

An object of class `fvcm`.

Author(s)

Reto Burgin

References

Breiman, L. (1996). Bagging Predictors. *Machine Learning*, **24**(2), 123–140.

Breiman, L. (2001). Random Forests. *Machine Learning*, **45**(1), 5–32.

Hastie, T., R. Tibshirani and J. Friedman (2001). *The Elements of Statistical Learning* (2 ed.). New York, USA: Springer-Verlag.

Burgin, R. A. (2015). Tree-based methods for moderated regression with application to longitudinal data. PhD thesis. University of Geneva.

See Also

fvcm-methods, tvcm, glm, olmm

Examples

```
## --------------------------------------------------------------------- #
## Dummy example:
##
## Bagging 'tvcm' on the artificially generated data 'vcrpart_3'. The
## true coefficient function is a sinus curve between -pi/2 and pi/2.
## The parameters 'maxstep = 3' and 'K = 5' are chosen to restrict the
## computations.
## --------------------------------------------------------------------- #

## simulated data
data(vcrpart_3)

## setting parameters
control <-
  fvcm_control(maxstep = 3,
               folds = folds_control("subsampling", K = 5, 0.5, seed = 3))

## fitting the forest
model <- fvcm(y ~ vc(z1, by = x1), data = vcrpart_3,
              family = gaussian(), control = control)

## plot the first two trees
plot(model, "coef", 1:2)

## plotting the partial dependency of the coefficient for 'x1'
plot(model, "partdep")
```

`fvcm-methods` *Methods for* `fvcm` *objects*

Description

Standard methods for computing on `fvcm` objects.

Usage

```
## S3 method for class 'fvcm'
oobloss(object, fun = NULL, ranef = FALSE, ...)

## S3 method for class 'fvcm'
plot(x, type = c("default", "coef",
           "simple", "partdep"),
     tree = NULL, ask = NULL, ...)

## S3 method for class 'fvcm'
predict(object, newdata = NULL,
        type = c("link", "response", "prob", "class", "coef", "ranef"),
        ranef = FALSE, na.action = na.pass, verbose = FALSE, ...)
```

Arguments

object, x	an object of class fvcm.
fun	the loss function. The default loss function is defined as the sum of the deviance residuals. For a user defined function fun, see the examples of oobloss.tvcm.
newdata	an optional data frame in which to look for variables with which to predict. If omitted, the training data are used.
type	character string indicating the type of plot or prediction. See plot.tvcm or predict.tvcm. "response" and "prob" are identical.
tree	integer vector. Which trees should be plotted.
ask	logical. Whether an input should be asked before printing the next panel.
ranef	logical scalar or matrix indicating whether predictions should be based on random effects. See predict.olmm.
na.action	function determining what should be done with missing values for fixed effects in newdata. The default is to predict NA: see na.pass.
verbose	logical scalar. If TRUE verbose output is generated during the validation.
...	further arguments passed to other methods.

Details

oobloss.fvcm estimates the out-of-bag loss based on predictions of the model that aggregates only those trees in which the observation didn't appear (cf. Hastie et al, 2001, sec. 15). The prediction error is computed as the sum of prediction errors obtained with fun, which are the deviance residuals by default.

The plot and the prediction methods are analogous to plot.tvcm resp. predict.tvcm. Note that the plot options mean and conf.int for type ="coef" are not available (and internally set to FALSE).

Further undocumented, available methods are fitted, print and ranef. All these latter methods have the same arguments as the corresponding default methods.

Value

The methods `fitted.fvcm` and `predict.fvcm` return an object of class `numeric` or `matrix`, depending on the used model or the specification of the argument `type`. See also `fitted.tvcm`.

The `oobloss.fvcm` method returns the output of the loss function defined by `fun`. This is a single numeric by default. See also `oobloss`.

The `plot.fvcm` method returns `NULL`.

The `ranef.fvcm` method returns an object of class `matrix` with values for the random effects. See also `ranef.olmm` and `ranef`.

Author(s)

Reto Burgin

References

Breiman, L. (1996). Bagging Predictors. *Machine Learning*, **24**(2), 123–140.

Breiman, L. (2001). Random Forests. *Machine Learning*, **45**(1), 5–32.

Hastie, T., R. Tibshirani and J. Friedman (2001). *The Elements of Statistical Learning* (2 ed.). New York, USA: Springer-Verlag.

See Also

`fvcm`, `tvcm-methods`

Examples

```
## ------------------------------------------------------------------- #
## Dummy example 1:
##
## Fitting a random forest tvcm on artificially generated ordinal
## longitudinal data. The parameters 'maxstep = 1' and 'K = 2' are
## chosen to restrict the computations.
## ------------------------------------------------------------------- #

## load the data

data(vcrpart_1)

## fit and analyse the model

control <-
  fvcolmm_control(mtry = 2, maxstep = 1,
                  folds = folds_control(type = "subsampling", K = 2, prob = 0.75))

model.1 <-
  fvcolmm(y ~ -1 + wave + vc(z3, z4, by = treat, intercept = TRUE) + re(1|id),
          family = cumulative(), subset = 1:100,
          data = vcrpart_1, control = control)
```

```
## estimating the out of bag loss
suppressWarnings(oobloss(model.1))

## predicting responses and varying coefficients for subject '27'
subs <- vcrpart_1$id == "27"

## predict coefficients
predict(model.1, newdata = vcrpart_1[subs,], type = "coef")

## marginal response prediction
predict(model.1, vcrpart_1[subs,], "response", ranef = FALSE)

## conditional response prediction
re <- matrix(5, 1, 1, dimnames = list("27", "(Intercept)"))
predict(model.1, vcrpart_1[subs,], "response", ranef = re)
predict(model.1, vcrpart_1[subs,], "response", ranef = 0 * re)

## predicting in-sample random effects
head(predict(model.1, type = "ranef"))

## fitted responses (marginal and conditional prediction)
head(predict(model.1, type = "response", ranef = FALSE))
head(predict(model.1, type = "response", ranef = TRUE))


## ------------------------------------------------------------------- #
## Dummy example 2:
##
## Fitting a random forest tvcm on artificially generated normally
## distributed data. The parameters 'maxstep = 3' and 'K = 3' are
## chosen to restrict the computations and 'minsize = 5' to obtain at
## least a few splits given the small sample size.
## ------------------------------------------------------------------- #

data(vcrpart_2)

## fit and analyse the model

control <- fvcm_control(mtry = 1L, minsize = 5, maxstep = 3,
                        folds_control("subsampling", K = 3, 0.75))

model.2 <- fvcglm(y ~ -1  + vc(z1, z2, by = x1, intercept = TRUE) + x2,
                  data = vcrpart_2,
                  family = gaussian(), subset = 1:50,control = control)

## estimating the out of bag loss
suppressWarnings(oobloss(model.2))

## predict the coefficient for individual cases
predict(model.2, vcrpart_2[91:100, ], "coef")
```

movie — *Movie critics*

Description

Movie critics of the Variety magazine. The data were previously used to fit adjacent-categories mixed models by Hartzl et al. (2001)

Usage

```
data(movie)
```

Format

A data frame with 372 observations on 93 movies. Three vectors contain information on

`movie` movie ID.

`critic` ordinal response on a 3 category scale, "Con" < "Mixed" < "Pro".

`review` critics, "Medved", "Ebert", "Siskel" and "Medved".

Source

The data are tabulated in Hartzel et al. (2001).

References

Hartzel, J., A. Agresti and B. Caffo (2001). Multinomial Logit Random Effect Models, *Statistical Modelling* **1**(2), 81–102.

olmm — *Fitting ordinal and nominal two-stage linear mixed models*

Description

Fits different types of two-stage linear mixed models for longitudinal (or clustered) ordinal (or multinomial) responses. O ne-stage models are also allowed. Random effects are assumed to be multivariate normal distributed with expectation 0. At the time being, cumulative link models with the logit, probit or cauchy link, the baseline-category logit and the adjacent-category logit model are implemented. Coefficients can be category-specific (i.e. non-proportional odds effects) or global (i.e. proportional odds, or parallel effects).

The function solves the score function for coefficients of the marginal likelihood by using Gauss-Hermite quadrature (e.g., Hedeker; 1994). Random effects are predicted by their expectation (see Hartzl et al.; 2001). Standard deviations of parameter estimates are, by default, based on the expected Fisher-information matrix.

Usage

```
cumulative(link = c("logit", "probit", "cauchy"))
adjacent(link = "logit")
baseline(link = "logit")

olmm(formula, data, family = cumulative(),
     weights, subset, na.action = na.omit,
     offset, contrasts, control = olmm_control(), ...)
```

Arguments

- `formula` a symbolic description of the model. This should be something like

  `y ~ ce(x1) + ge(x2) +re(1 + ge(w2) | id)`

  where `ce(x1)` specifies that the predictor `x1` has a category-specific i.e. non-proportional odds effect and `ge(x2)` that the predictor `x2` has global i.e. proportional odds fixed effect, see `ge`, resp. `ce`. Random effects are specified within the `re` term, where the variable `id` above behind the vertical bar `|` defines the subject i.e. cluster factor. Notice that only one subject factor is allowed. See details.
- `data` an optional data frame with the variables in `formula`. By default the variables are taken from the environment from which `olmm` is called.
- `family` an `family.olmm` object produced by `cumulative`, `adjacent` or `baseline`.
- `weights` a numeric vector of weights with length equal the number of observations. The weights should be constant for subjects.
- `offset` a matrix specifying the offset separately for each predictor equation, of which there are the number of categories of the response minus one.
- `subset, na.action, contrasts` further model specification arguments as in `lm`.
- `control` a list of control parameters produced by `olmm_control`.
- `link` character string. The name of the link function.
- `...` arguments to be passed to `control`.

Details

The function can be used to fit simple ordinal two-stage mixed effect models with up to 3-4 random effects. For models with higher dimensions on random effects, the procedure may not convergence (cf. Tutz; 1996). Coefficients for the adjacent-category logit model are extracted via coefficient transformation (e.g. Agresti; 2010).

The three implemented families are defined as follows: `cumulative` is defined as the link of the sum of probabilities of lower categories, e.g., for `link = "logit"`, the logit of the sum of probabilities of lower categories. `adjacent` is defined as the logit of the probability of the lower of two adjacent categories. `baseline` is defined as the logit of the probability of a category with reference to the highest category. Notice that the estimated coefficients of cumulative models may have the opposite sign those obtained with alternative software.

For alternative fitting functions, see for example the functions `clmm` of **ordinal**, `nplmt` of package **mixcat**, `DPolmm` of package **DPpackage**, `lcmm` of package **lcmm**, `MCMCglmm` of package **MCMCglmm** or `OrdinalBoost` of package **GMMBoost**.

The implementation adopts functions of the packages **statmod** (Novomestky, 2012) and **matrixcalc** (Smyth et al., 2014), which is not visible for the user. The authors are grateful for these codes.

The `formula` argument specifies the model to be fitted. Categorical regression models distinguish between global effects (or proportional-odds effects), which are defined with `ge` terms, and category-specific effects, which are defined by `ce` terms. For undefined terms, the function will use `ge` terms. Notice that this default does not necessarily yield interpretable outputs. For example, for the `baseline` model you may use only `ce` terms, which must be specified manually manually. See the example below. For `cumulative` models at present it is not possible to specifiy `ce` for the random effects component because the internal, unconstraint integration would yield unusable predictor values.

Value

`olmm` returns an object of class `olmm`. `cumulative`, `adjacent` and `baseline` yield an object of class `family.olmm`. The `olmm` class is a list containing the following components:

`env`	environment in which the object was built.
`frame`	the model frame.
`call`	the matched call to the function that created the object (class `"call"`).
`control`	a list of class `olmm_control` produced by `olmm_control`.
`formula`	the formula of the call.
`terms`	a list of `terms` of the fitted model.
`family`	an object of class `family.olmm` that specifies that family of the fitted model.
`y`	(ordered) categorical response vector.
`X`	model matrix for the fixed effects.
`W`	model matrix for the random effects.
`subject`	a factor vector with grouping levels.
`subjectName`	variable name of the subject vector.
`weights`	numeric observations weights vector.
`weights_sbj`	numeric weights vector of length N.
`offset`	numeric offset matrix
`xlevels`	(only where relevant) a list of levels of the factors used in fitting.
`contrasts`	(only where relevant) a list of contrasts used.
`dims`	a named integer of dimensions. Some of the dimensions are n is the number of observations, p is the number of fixed effects per predictor and q is the total number of random effects.
`fixef`	a matrix of fixed effects (one column for each predictor).
`ranefCholFac`	a lower triangular matrix. The cholesky decomposition of the covariance matrix of the random effects.

coefficients	a numeric vector of several fitted model parameters
restricted	a logical vector indicating which elements of the coefficients slot are restricted to an initial value at the estimation.
eta	a matrix of unconditional linear predictors of the fixed effects without random effects.
u	a matrix of orthogonal standardized random effects (one row for each subject level).
logLik_obs	a numeric vector of log likelihood value (one value for each observation).
logLik_sbj	a numeric vector of log likelihood values (one value for each subject level).
logLik	a numeric value. The log likelihood of the model.
score_obs	a matrix of observation-wise partial derivates of the marginal log-likelihood equation.
score_sbj	a matrix of subject-wise partial derivates of the marginal log-likelihood equation.
score	a numeric vector of (total) partial derivates of the log-Likelihood function.
info	the information matrix (default is the expected information).
ghx	a matrix of quadrature points for the Gauss-Hermite quadrature integration.
ghw	a matrix of weights for the Gauss-Hermite quadrature integration.
ranefElMat	a transformation matrix
optim	a list of arguments for calling the optimizer function.
control	a list of used control arguments produced by olmm_control.
output	the output of the optimizer (class "list").

Author(s)

Reto Burgin

References

Agresti, A. (2010). *Analysis of Ordinal Categorical Data* (2 ed.). New Jersey, USA: John Wiley & Sons.

Hartzel, J., A. Agresti and B. Caffo (2001). Multinomial Logit Random Effect Models, *Statistical Modelling* **1**(2), 81–102.

Hedeker, D. and R. Gibbons (1994). A Random-Effects Ordinal Regression Model for Multilevel Analysis, *Biometrics* **20**(4), 933–944.

Tutz, G. and W. Hennevogl (1996). Random Effects in Ordinal Regression Models, *Computational Statistics & Data Analysis* **22**(5), 537–557.

Tutz, G. (2012). *Regression for Categorical Data*. New York, USA: Cambridge Series in Statistical and Probabilistic Mathematics.

Novomestky, F. (2012). matrixcalc: Collection of Functions for Matrix Calculations. R package version 1.0-3. URL https://CRAN.R-project.org/package=matrixcalc

Smyth, G., Y. Hu, P. Dunn, B. Phipson and Y. Chen (2014). statmod: Statistical Modeling. R package version 1.4.20. URL https://CRAN.R-project.org/package=statmod

See Also

olmm-methods, olmm_control, ordered

Examples

```
## ------------------------------------------------------------------- #
## Example 1: Schizophrenia
##
## Estimating the cumulative mixed models of
## Agresti (2010) chapters 10.3.1
## ------------------------------------------------------------------- #

data(schizo)

model.10.3.1 <-
  olmm(imps79o ~ tx + sqrt(week) + re(1|id),
       data = schizo, family = cumulative())

summary(model.10.3.1)

## ------------------------------------------------------------------- #
## Example 2: Movie critics
##
## Estimating three of several adjacent-categories
## mixed models of Hartzl et. al. (2001)
## ------------------------------------------------------------------- #

data(movie)

## model with category-specific effects for "review"
model.24.1 <- olmm(critic ~ ce(review) + re(1|movie, intercept = "ce"),
                   data = movie, family = adjacent())

summary(model.24.1)
```

olmm-control — *Control parameters for* olmm.

Description

Various parameters that control aspects for olmm.

Usage

```
olmm_control(fit = c("nlminb", "ucminf", "optim"),
             doFit = TRUE, numGrad = FALSE,
             numHess = numGrad, nGHQ = 7L,
             start = NULL, restricted = NULL, verbose = FALSE, ...)
```

Arguments

fit	character string. The name of the function to be used for the optimization. Can be one of "nlminb", "ucminf", "optim"
doFit	logical scalar. When FALSE an unfitted olmm object is returned.
numGrad	logical scalar indicating whether the score function should be retrieved numerically.
numHess	logical scalar. Indicates whether the Hess matrix for the variance-covariance matrix should be estimated numerically, which is an approximation of the observed Fisher information. Must be TRUE if numGrad is TRUE. See details.
nGHQ	a positive integer specifying the number of quadrature points for the approximation of the marginal Likelihood by numerical integration.
start	a named numeric vector of initial values for the parameters. The parameter must be named in exactly in the way as they appear when the model is fitted.
restricted	a character vector of names of coefficients to be restricted to the initial values. The argument is ignored in case of adjacent category models.
verbose	logical scalar. If TRUE verbose output is generated during the optimization of the parameter estimates.
...	further arguments to be passed to fit.

Details

Initial values may decrease the computation time and avoid divergence. The start argument accepts a vector with named elements according to the column names of the model.matrix. At the time being, initial values for adjacent-categories models must be transformed into the baseline-category model form.

Notice that an additional argument control, e.g., control = list(trace = 1), can be passed access control parameters of the optimizers. For arguments, see ucminf, nlminb or optim.

Value

A list of class olmm_control containing the control parameters.

Author(s)

Reto Burgin

See Also

olmm

Examples

```
olmm_control(doFit = FALSE)
```

olmm-gefp *Methods for score processes of olmm objects*

Description

Methods to extract and pre-decorrelate the (negative) marginal maximum likelihood observation scores and compute the standardized cumulative score processes of a fitted olmm object.

Usage

```
olmm_estfun(x, predecor = FALSE, control = predecor_control(),
            nuisance = NULL, ...)

predecor_control(impute = TRUE, seed = NULL,
                 symmetric = TRUE, center = FALSE,
                 reltol = 1e-6,
                 maxit = 250L, minsize = 1L,
                 include = c("observed", "all"),
                 verbose = FALSE, silent = FALSE)

olmm_gefp(object, scores = NULL, order.by = NULL, subset = NULL,
          predecor = TRUE, parm = NULL, center = TRUE, drop = TRUE,
          silent = FALSE, ...)
```

Arguments

`x, object`	a fitted olmm object.
`predecor`	logical scalar. Indicates whether the within-subject correlation of the estimating equations should be removed by a linear transformation. See details.
`control`	a list of control parameter as produced by predecor_control.
`nuisance`	integer vector. Defines the coefficients which are regarded as nuisance and therefore omitted from the transformation.
`impute`	logical scalar. Whether missing values should be replaced using imputation.
`seed`	an integer scalar. Specifies the random number used for the `set.seed` call before the imputation. If set to `NULL`, `set.seed` is not processed.
`symmetric`	logical scalar. Whether the transformation matrix should be symmetric.
`minsize`	integer scalar. The minimum number of observations for which entries in the transformation should be computed. Higher values will lead to lower accuracy but stabilize the computation.
`reltol`	convergence tolerance used to compute the transformation matrix.
`maxit`	the maximum number of iterations used to compute the transformation matrix.
`silent`	logical scalar. Should the report of warnings be suppressed?

include	logical scalar. Whether the transformation matrix should be computed based on the scores corresponding to observations (option "observed") or on all scores (option "all"), including the imputed values.
verbose	logical scalar. Produces messages.
scores	a function or a matrix. Function to extract the estimating equations from object or a matrix representing the estimating equations. If NULL (default), the olmm_estfun function will be used with argument predecor and additional arguments from ....
order.by	a numeric or factor vector. The explanatory variable to be used to order the entries in the estimating equations. If set to NULL (the default) the observations are assumed to be ordered.
subset	logical vector. For extracts the subset of the estimating equations to be used.
parm	integer, logical or a character vector. Extracts the columns of the estimating equations.
center	logical scalar. TRUE subtracts, if necessary, the column means of the estimating equations.
drop	logical. Whether singularities should be handled automatically (otherwise singularities yield an error).
...	arguments passed to other functions. olmm_gefp passes these arguments to scores if scores is a function.

Details

Complements the estfun method of the package **sandwich** and the gefp function of the package **strucchange** for olmm objects. olmm_estfun allows to pre-decorrelate the intra-individual correlation of observation scores, see the argument predecor. The value returned by olmm_gefp may be used for testing coefficient constancy regarding an explanatory variable order.by by the sctest function of package **strucchange**, see the examples below.

If predecor = TRUE in olmm_estfun, a linear within-subject transformation is applied that removes (approximately) the intra-subject correlation from the scores. Backgrounds are provided by Burgin and Ritschard (2014a).

Given a score matrix produced by olmm_estfun, the empirical fluctuation process can be computed by olmm_gefp. See Zeileis and Hornik (2007). olmm_gefp provides with subset and parm arguments specifically designed for nodewise tests in the tvcm algorithm. Using subset extracts the partial fluctuation process of the selected subset. Further, center = TRUE makes sure that the partial fluctuation process (starts and) ends with zero.

Value

predecor_control returns a list of control parameters for computing the pre-decorrelation transformation matrix. olmm_estfun returns a matrix with the estimating equations and olmm_gefp a list of class class "gefp".

Author(s)

Reto Burgin

References

Zeileis A., Hornik K. (2007), Generalized M-Fluctuation Tests for Parameter Instability, *Statistica Neerlandica*, **61**(4), 488–508.

Burgin R. and Ritschard G. (2015), Tree-Based Varying Coefficient Regression for Longitudinal Ordinal Responses. *Computational Statistics & Data Analysis*, **86**, 65–80.

See Also

olmm

Examples

```
## ------------------------------------------------------------------- #
## Dummy example :
##
## Testing coefficient constancy on 'z4' of the 'vcrpart_1' data.
## ------------------------------------------------------------------- #

data(vcrpart_1)

## extract a unbalanced subset to show to the full functionality of estfun
vcrpart_1 <- vcrpart_1[-seq(1, 100, 4),]
subset <- vcrpart_1$wave != 1L ## obs. to keep for fluctuation tests
table(table(vcrpart_1$id))

## fit the model
model <- olmm(y ~ treat + re(1|id), data = vcrpart_1)

## extract and pre-decorrelate the scores
scores <- olmm_estfun(
  model, predecor = TRUE,
  control = predecor_control(verbose = TRUE))
attr(scores, "T") # transformation matrix

## compute the empirical fluctuation process
fp <- olmm_gefp(model, scores, order.by = vcrpart_1$z4)

## process a fluctuation test
library(strucchange)
sctest(fp, functional = catL2BB(fp))
```

---

olmm-methods *Methods for* olmm *objects*

Description

Standard methods for computing on olmm objects.

Usage

```
## S3 method for class 'olmm'
anova(object, ...)

## S3 method for class 'olmm'
coef(object, which = c("all", "fe"), ...)

## S3 method for class 'olmm'
fixef(object, which = c("all", "ce", "ge"), ...)

## S3 method for class 'olmm'
model.matrix(object, which = c("fe", "fe-ce", "fe-ge",
             "re", "re-ce", "re-ge"), ...)

## S3 method for class 'olmm'
neglogLik2(object, ...)

## S3 method for class 'olmm'
ranef(object, norm = FALSE, ...)

## S3 method for class 'olmm'
ranefCov(object, ...)

## S3 method for class 'olmm'
simulate(object, nsim = 1, seed = NULL,
         newdata = NULL, ranef = TRUE, ...)

## S3 method for class 'olmm'
terms(x, which = c("fe-ce", "fe-ge", "re-ce", "re-ge"), ...)

## S3 method for class 'olmm'
VarCorr(x, sigma = 1., ...)

## S3 method for class 'olmm'
weights(object, level = c("observation", "subject"), ...)
```

Arguments

object, x	an olmm object.
which	optional character string. For coef and fixef, it indicates whether "all" coefficients, the fixed effects "fe", the category-specific fixed effects "ce" (i.e. non-proportional odds) or the global fixed effects "ge" (i.e. proportional odds) should be extracted. For model.matrix it indicates whether the model matrix of the fixed- ("fe") or the random effects ("re") should be extracted.
level	character string. Whether the results should be on the observation level (level = "observation") or on the subject level (level = "subject").
norm	logical. Whether residuals should be divided by their standard deviation.

nsim	number of response vectors to simulate. Defaults to 1.
seed	an object specifying if and how the random number generator should be initialized. See simulate
newdata	a data frame with predictor variables.
ranef	either a logical or a matrix (see predict.olmm). Whether the simulated responses should be conditional on random effects. If TRUE, the newdata data frame must contain the subject identification variable. Further, if all subjects in newdata are in object, the simulation will be based on the estimated random effects as obtained with ranef. If any subject in newdata is not in object the random effects are simulated.
sigma	ignored but obligatory argument from original generic.
...	potential further arguments passed to methods.

Details

anova implements log-likelihood ratio tests for model comparisons, based on the marginal likelihood. At the time being, at least two models must be assigned.

neglogLik2 returns the marginal maximum likelihood of the fitted model times minus 2.

ranefCov extracts the variance-covariance matrix of the random effects. Similarly, VarCorr extracts the estimated variances, standard deviations and correlations of the random effects.

resid extracts the residuals of Li and Sheperd (2012). By default, the marginal outcome distribution is used to compute these residuals. The conditional residuals can be computed by assigning ranef = TRUE as a supplementary argument.

simulate simulates ordinal responses based on the input model.

Further, undocumented methods are deviance, extractAIC, fitted, formula, getCall, logLik, model.frame, nobs, update, vcov.

The anova implementation is based on codes of the **lme4** package. The authors are grateful for these codes.

Value

The anova.olmm method returns an object of class anova, see also anova.

The coef.olmm, coefficients.olmm, fixef, fixef.glm and fixef.olmm methods return named numeric vectors. See also coef and coefficients.

The deviance.olmm method returns a single numeric, see also deviance.

The formula.olmm method extracts the model formula, which is an object of class formula. See also formula.

The getCall.olmm method extracts the call for fitting the model, which is an object of class call. See also call.

The logLik.olmm method returns an object of class logLik, which is a single numeric with a few attributes. See also logLik.

The neglogLik2 and neglogLik2.olmm methods return a single numeric.

The model.frame.olmm and model.matrix.olmm methods return the model frame and the model matrix of the olmm object. See also model.frame and model.matrix.

The `ranef` and `ranef.olmm` methods return a matrix with the estimated random effects.

The `ranefCov` and `ranefCov.olmm` methods return an object of class `matrix`. The `VarCorr` and `VarCorr.olmm` methods return an object of class `VarCorr.olmm`. `print.VarCorr.olmm` returns an object of class `VarCorr.olmm`.

The `resid.olmm` and `residuals.olmm` methods return a numeric vector.

The `simulate.olmm` method returns a `data.frame` including simulated responses based on the input model.

The `terms.olmm` method returns an object of class `terms`. See also `terms`.

The `update.olmm` method will update and (by default) re-fit a model. It returns an object of class `olmm`. See also `update`.

The `vcov.olmm` method extracts a `matrix` with the variances and covariances of the fixed effects of the model. See also `vcov`.

The `weights.olmm` method extracts a `numeric` vector with the model weights. See also `weights`.

Author(s)

Reto Burgin

References

Agresti, A. (2010). *Analysis of Ordinal Categorical Data* (2 ed.). New Jersey, USA: John Wiley & Sons.

Tutz, G. (2012). *Regression for Categorical Data*. New York, USA: Cambridge Series in Statistical and Probabilistic Mathematics.

Li, C. and B. E. Sheperd (2012). A New Residual for Ordinal Outcomes, *Biometrika*, **99**(2), 437–480.

Bates, D., M. Maechler, B. M. Bolker and S. Walker (2015). Fitting Linear Mixed-Effects Models Using lme4, *Journal of Statistical Software*, **67**(1), 1–48.

See Also

`olmm`, `predict.olmm`, `olmm_gefp`

Examples

```
## --------------------------------------------------------- #
## Example: Schizophrenia (see also example of 'olmm')
## --------------------------------------------------------- #

data(schizo)

schizo <- schizo[1:181,]
schizo$id <- droplevels(schizo$id)

## anova comparison
## ----------------
```

```
## fit two alternative models for the 'schizo' data
model.0 <- olmm(imps79o ~ tx + sqrt(week) + re(1|id), schizo)
model.1 <- olmm(imps79o ~ tx + sqrt(week)+tx*sqrt(week)+re(1|id),schizo)
anova(model.0, model.1)

## simulate responses
## ------------------

## simulate responses based on estimated random effects
simulate(model.0, newdata = schizo[1, ], ranef = TRUE, seed = 1)
simulate(model.0, newdata = schizo[1, ], seed = 1,
         ranef = ranef(model.0)[schizo[1, "id"],,drop=FALSE])
## simulate responses based on simulated random effects
newdata <- schizo[1, ]
newdata$id <- factor("123456789")
simulate(model.0, newdata = newdata, ranef = TRUE)

## other methods
## -------------

coef(model.1)
fixef(model.1)
head(model.matrix(model.1, "fe-ge"))
head(weights(model.1))
ranefCov(model.1)
head(resid(model.1))
terms(model.1, "fe-ge")
VarCorr(model.1)
head(weights(model.1, "subject"))
```

---

olmm-predict *Predict outcome probabilities and responses for* olmm *objects*

Description

`fitted` and `predict` method for `olmm` objects. The function implements mainly the prediction methods of Skrondal and Rabe-Hesketh (2009).

Usage

```
## S3 method for class 'olmm'
fitted(object, ...)

## S3 method for class 'olmm'
predict(object, newdata = NULL,
        type = c("link", "response", "prob", "class", "ranef"),
        ranef = FALSE, na.action = na.pass, ...)
```

Arguments

`object`	a fitted `olmm` object.
`newdata`	data frame for which to evaluate predictions.
`type`	character string. `type = "response"` and `type = "prob"` yield response probabilities, `type = "class"` the response category with highest probability and `type = "link"` the linear predictor matrix. `type = "ranef"` yields the predicted random effects, see `ranef.olmm`.
`ranef`	logical or numeric matrix. See details.
`na.action`	function determining what should be done with missing values for fixed effects in `newdata`. The default is to predict `NA`: see `na.pass`.
`...`	optional additional parameters. Includes `offset` and `subset`.

Details

If `type = "link"` and `ranef = FALSE`, the fixed effects components are computed. The random effect components are ignored.

If `type = "link"` and `ranef = TRUE`, the fixed effect components plus the random effect components are computed. The function will look for whether random coefficients are available for the subjects (i.e. clusters) in `newdata`. If so, it extracts the corresponding random effects as obtained by `ranef`. For new subjects in `newdata` the random effects are set to zero. If `newdata` does not contain a subject vector, the random effects are set to zero.

If `type = "link"` and `ranef` is a matrix, the fixed effect components plus the random effect components with the random coefficients from the assigned matrix are computed. Notice that `newdata` should contain a subject vector to assign the random coefficients. This prediction method is, amongst others, proposed in Skrondal and Rabe-Hesketh (2009), Sec. 7.1.

The two options `type = "response"` and `type = "prob"` are identical and `type = "class"` extracts the response category with the highest probability. Hence, the prediction mechanism is the same for all three options.

Given `newdata` contains a subject vector, `type = "response"` combined with `ranef = FALSE` yields for new subjects the population-averaged response probabilities (Skrondal and Rabe-Hesketh, Sec. 7.2) and for existing subjects the cluster-averaged prediction (Skrondal and Rabe-Hesketh 2009, Sec. 7.3). If no subject vector is assigned the function assumes that all subjects are new and therefore yields the population-averaged response probabilities (Skrondal and Rabe-Hesketh 2009, Sec. 7.2).

The option `type = "response"` combined with `ranef = TRUE` works equivalent to `type = "link"` combined with `ranef = TRUE`.

If the model does not contain random effects, the argument `ranef` is ignored.

Value

A matrix or a vector of predicted values or response probabilities.

Note

The method can not yet handle new categories in categorical predictors and will return an error.

Author(s)

Reto Burgin

References

Skrondal, A., S. Rabe-Hesketh (2009). Prediction in Multilevel Generalized Linear Models. *Journal of the Royal Statistical Society A*, **172**(3), 659–687.

See Also

olmm, olmm-methods

Examples

```
## ------------------------------------------------------------------- #
## Example: Schizophrenia
## ------------------------------------------------------------------- #

data(schizo)

## omit subject 1103 and the last observations of 1104 and 1105
subs <- c(1:4, 8, 11)

dat.train <- schizo[-subs, ] # training data
dat.valid <- schizo[ subs, ] # test data

## fit the model
model <- olmm(imps79o ~ tx + sqrt(week) + tx:sqrt(week) + re(1|id), dat.train)

## prediction on the predictor scale
## ---------------------------------

## random effects are set equal zero
predict(model, newdata = dat.valid, type = "link", ranef = FALSE)

## .. or equally with self-defined random effects
ranef <- matrix(0, 3, 1)
rownames(ranef) <- c("1103", "1104", "1105")
predict(model, newdata = dat.valid, type = "link", ranef = ranef)

## use random effects for the subjects 1104 and 1105.
predict(model, newdata = dat.valid, type = "link", ranef = TRUE)

## prediction on the response scale
## --------------------------------

## use random effects for the subjects 1104 and 1105.
predict(model, newdata = dat.valid, type = "response", ranef = FALSE)
predict(model, newdata = dat.valid, type = "prob", ranef = FALSE) # .. or, equally
predict(model, newdata = dat.valid, type = "class", ranef = FALSE)

## treat all individuals as new (subject vector is deleted)
```

```
predict(model, newdata = dat.valid[,-1], type = "response", ranef = FALSE)

## use random effects for the subjects 1104 and 1105.
predict(model, newdata = dat.valid, type = "response", ranef = TRUE)

## use self defined random effects
ranef <- matrix(0, 3, 1)
rownames(ranef) <- c("1103", "1104", "1105")
predict(model, newdata = dat.valid, type = "response", ranef = ranef)

## predict random effects
## ----------------------

head(predict(model, type = "ranef"))
head(ranef(model)) # .. or, equally
```

`olmm-summary` *Printing and summarizing `olmm` objects*

Description

Generates summary results of a fitted `olmm` object.

Usage

```
## S3 method for class 'olmm'
summary(object, etalab = c("int", "char", "eta"),
        silent = FALSE, ...)

## S3 method for class 'olmm'
print(x, etalab = c("int", "char", "eta"), ...)
```

Arguments

`object, x`	a fitted `olmm` object.
`etalab`	character. Whether category-specific effects should be labeled by integers of categories (default), the labels of the categories or the index of the predictor.
`silent`	logical: should a warning be reported if the computation of the covariance matrix for the estimated coefficients failed.
`...`	additional arguments passed to print.

Value

The `summary` method returns a list of class `"summary.olmm"`.

Author(s)

Reto Burgin

See Also

olmm, olmm-methods

Examples

```
## ------------------------------------------------------------------- #
## Dummy example:
##
## Printing the summary of a model on artificially generated data.
## ------------------------------------------------------------------- #

data(vcrpart_1)

model <- olmm(y ~ wave + z4:treat + re(1|id), vcrpart_1, subset = 1:60)

print(model, digits = 2)

summary(model, digits = 2)
```

---

otsplot *Time-series plot for longitudinal ordinal data*

Description

Plots multiple ordinal sequences in a x (usually time) versus y (response variable) scatterplot. The sequences are displayed by jittered frequency-weighted parallel lines.

Usage

```
## Default S3 method:
otsplot(x, y, subject, weights, groups,
        control = otsplot_control(), filter = NULL,
        main, xlab, ylab, xlim, ylim, ...)

otsplot_control(cex = 1, lwd = 1/4, col = NULL,
                hide.col = grey(0.8), seed = NULL,
                lorder = c("background", "foreground") ,
                lcourse = c("upwards", "downwards"),
                grid.scale = 1/5, grid.lwd = 1/2,
                grid.fill =  grey(0.95), grid.col = grey(0.6),
                layout = NULL, margins = c(5.1, 4.1, 4.1, 3.1),
                strip.fontsize = 12, strip.fill =  grey(0.9),
                pop = TRUE, newpage = TRUE, maxit = 500L)

otsplot_filter(method = c("minfreq", "cumfreq", "linear"), level = NULL)
```

Arguments

`x`	a `numeric` or `factor` vector for the x axis, e.g. time.
`y`	an `ordered` factor vector for the y axis.
`subject`	a `factor` vector that identifies the subject, i.e., allocates elements in `x` and `y` to the subject i.e. observation unit.
`weights`	a numeric vector of weights of length equal the number of subjects.
`groups`	a `numeric` or `factor` vector of group memberships of length equal the number of subjects. When specified, one panel is generated for each distinct membership value.
`control`	control parameters produced by `otsplot_control`, such as line colors or the scale of translation zones.
`filter`	an `otsplot_filter` object which defines line coloring options. See details.
`main, xlab, ylab`	title and axis labels for the plot.
`xlim, ylim`	the x limits `c(x1, x2)` resp. y limits `(y1,y2)`.
`...`	additional undocumented arguments.
`cex`	expansion factor for the squared symbols.
`lwd`	expansion factor for line widths. The expansion is relative to the size of the squared symbols.
`col`	color palette vector for line coloring.
`hide.col`	Color for ordinal time-series filtered-out by the `filter` specification in `otsplot`.
`seed`	an integer specifying which seed should be set at the beginning.
`lorder`	line ordering. Either `"background"` or `"foreground"`.
`lcourse`	Method to connect simultaneous elements with the preceding and following ones. Either `"upwards"` (default) or `"downwards"`.
`grid.scale`	expansion factor for the translation zones.
`grid.lwd`	expansion factor for the borders of translation zones.
`grid.fill`	the fill color for translation zones.
`grid.col`	the border color for translation zones.
`strip.fontsize`	fontsize of titles in stripes that appear when a `groups` vector is assigned.
`strip.fill`	color of strips that appear when a `groups` vector is assigned.
`layout`	an integer vector `c(nr, nc)` specifying the number of rows and columns of the panel arrangement when the `groups` argument is used.
`margins`	a numeric vector `c(bottom, left, top, right)` specifying the space on the margins of the plot. See also the argument `mar` in `par`.
`pop`	logical scalar. Whether the viewport tree should be popped before return.
`newpage`	logical scalar. Whether `grid.newpage()` should be called previous to the plot.
`maxit`	maximal number of iteration for the algorithm that computes the translation arrangement.
`method`	character string. Defines the filtering function. Available are `"minfreq"`, `"cumfreq"` and `"linear"`.
`level`	numeric scalar between 0 and 1. The frequency threshold for the filtering methods `"minfreq"` and `"cumfreq"`.

Details

The function is a scaled down version of the seqpcplot function of the **TraMineR** package, implemented in the **grid** graphics environment.

The filter argument serves to specify filters to fade out less interesting patterns. The filtered-out patterns are displayed in the hide.col color. The filter argument expects an object produced by otsplot_filter.

otsplot_filter("minfreq", level = 0.05) colors patterns with a support of at least 5% (within a group). otsplot_filter("cumfreq", level = 0.75) highlight the 75% most frequent patterns (within group). otsplot_filter("linear") linearly greys out patterns with low support.

The implementation adopts a color palette which was originally generated by the **colorspace** package (Ihaka et al., 2013). The authors are grateful for these codes.

Value

otsplot returns an object of class otsplot.

otsplot_control returns an object of class otsplot_control and otsplot_filter an object of class otsplot_filter. Both these object types are specifically designed as input arguments of otsplot.

Author(s)

Reto Burgin and Gilbert Ritschard

References

Burgin, R. and G. Ritschard (2014). A Decorated Parallel Coordinate Plot for Categorical Longitudinal Data, *The American Statistician* **68**(2), 98–103.

Ihaka, R., P. Murrell, K. Hornik, J. C. Fisher and A. Zeileis (2013). colorspace: Color Space Manipulation. R package version 1.2-4. URL https://CRAN.R-project.org/package=colorspace.

Examples

```
## ------------------------------------------------------------------- #
## Dummy example:
##
## Plotting artificially generated ordinal longitudinal data
## ------------------------------------------------------------------- #

## load the data
data(vcrpart_1)
vcrpart_1 <- vcrpart_1[1:40,]

## plot the data
otsplot(x = vcrpart_1$wave, y = vcrpart_1$y, subject = vcrpart_1$id)

## using 'groups'
groups <- rep(c("A", "B"), each = nrow(vcrpart_1) / 2L)
otsplot(x = vcrpart_1$wave, y = vcrpart_1$y, subject = vcrpart_1$id,
        groups = groups)
```

```
## color series with supports over 30%
otsplot(x = vcrpart_1$wave, y = vcrpart_1$y, subject = vcrpart_1$id,
        filter = otsplot_filter("minfreq", level = 0.3))

## highlight the 50% most frequent series
otsplot(x = vcrpart_1$wave, y = vcrpart_1$y, subject = vcrpart_1$id,
        filter = otsplot_filter("cumfreq", level = 0.5))

## linearly grey out series with low support
otsplot(x = vcrpart_1$wave, y = vcrpart_1$y, subject = vcrpart_1$id,
        filter = otsplot_filter("linear"))

## subject-wise plot
otsplot(x = vcrpart_1$wave, y = vcrpart_1$y,
        subject = vcrpart_1$id, groups = vcrpart_1$id)
```

PL *Effect of parental leave policy*

Description

Data to analyze the effect of the 1990 Austrian parental leave reform on fertility and postbirth labor market careers. The data originate from the Austrian Social Security Database (ASSD) and where prepared by Lalive and Zweimueller (2009). The sample includes 6'180 women giving a childbirth (the first birth recorded in the ASSD data) between June and July 1990 and were eligible to benefit from the parental leave program.

Usage

```
data(PL)
```

Format

A data frame with 6'180 observations on the following variables

`uncb3` binary. Additional birth 0-36 months after child birth.

`uncb10` binary. Additional birth 0-120 months after child birth.

`uncj3` binary. Return-to-work 0-36 months after child birth.

`uncj10` numeric. Return-to-work 0-120 months after child birth.

`pbexp10` numeric. Employment (months/yr), 37-120 months after child birth.

`pbinc_tot10` numeric. Earnings (EUR/month), 37-120 months after child birth.

`pbexp3` numeric. Employment (months/yr), 0-36 months after child birth.

`pbinc_tot3` numeric. Earnings (EUR/month), 0-36 months after child birth.

`ikar3` numeric. Length of parental leave of the first year after birth.

`ikar4` numeric. Length of parental leave of the second year after birth.

`july` binary treatment variable. Indicates whether the child considered (the first recorded in the ASSD data) was born in June 1990 or in July 1990.

`bd` child's birthday.

`workExp` years in employment prior to birth.

`unEmpl` years in unemployment prior to birth.

`zeroLabEarn` factor. Whether women has earnings at birth.

`laborEarnings` numeric. Earnings at birth.

`employed` factor. Whether the woman was employed in 1989.

`whiteCollar` factor. Whether woman is white collar worker.

`wage` numeric. Daily 1989 earnings.

`age` ordered factor. Age.

`industry`, `industry.SL` factor. Industry where woman worked.

`region`, `region.SL` factor. The region where the woman lives.

Details

The data are described in Lalive and Zweimueller (2009).

Source

Austrian Social Security Database (ASSD). The data set is also available from https://sites.google.com/site/rafaellalive/research

References

Lalive, R. and J. Zweimueller (2009). Does Parental Leave Affect Fertility and Return-to-Work? Evidence from Two Natural Experiments. *The Quarterly Journal of Economics* **124**(3), 1363–1402.

`poverty` *Poverty in Switzerland*

Description

Poverty measurements of elderly people (older than the Swiss legal retirement age) in Switzerland. The data are the (complete) subsample of participants of the canton Valais of the Vivre-Leben-Vivere (VLV) survey data.

Usage

```
data(poverty)
```

Format

A data frame with 576 observations on the following variables

- `Poor` binary response variable on whether the person is considered as poor or not. 0 = no and 1 = yes.
- `Canton` the canton where the person lives. All individuals origin from the canton Wallis.
- `Gender` whether person is a male or a female.
- `AgeGroup` to which age group the person belongs to.
- `Edu` ordered 3-category measurement on the persons education.
- `CivStat` civil status.
- `NChild` number of children.
- `Working` whether the person is still working (even though all persons are in the legal retirement age).
- `FirstJob` 5-category classification of the person's first job.
- `LastJob` 5-category classification of the person's last job.
- `Origin` whether the person origins from Switzerland or a foreign country.
- `SocMob` whether and how the person has changed his social status over the life span.
- `RetirTiming` timing of the retirement relative to the legal retirement age.
- `ProfCar` 4-category classification of the professional carrier. Possible are `"full employment"`, `"missing / early retirement"`, `"start and stop"` and `"stop and restart"`. The classification was retrieved from a longitudinal cluster analysis on the professional carriers in Gabriel et. al. (2014).
- `Pension` 5-category classification of the pension plan. Number refer to the Swiss pension three-pillar system.
- `TimFirstChild` timing of first child relative to the average timing of the first child of the same age group.

Details

Poverty is defined by a threshold of 2400 Swiss francs per person in the household. Specifically, the `poverty` variable was retrieved from a self-rated ordinal variable with nine categories on household income and was adjusted by the OECD equivalence scales methodology (see https://www.oecd.org/economy/growth/OECD-Note-EquivalenceScales.pdf) to account for the household size.

The variables `Canton`, `Gender` and `AgeGroup` represent the stratification variables of the survey design.

The data include a significant number of missings, in particular for `Poor` and `RetirTiming`. The authors are grateful to Rainer Gabriel, Michel Oris and the *Centre interfacultaire de gerontologie et d'etudes des vulnerabilites* (CIGEV) at the University of Geneva for providing the prepared data set.

Source

VLV survey

References

Ludwig, C., S. Cavalli and M. Oris 'Vivre/Leben/Vivere': An interdisciplinary survey addressing progress and inequalities of ageing over the past 30 years in Switzerland. *Archives of Gerontology and Geriatrics*.

Gabriel, R., M. Oris, M. Studer and M. Baeriswyl (2015). The Persistance of Social Stratification? *Swiss Journal of Sociology*, **41**(3), 465–487.

`schizo` *National Institute of Mental Health shizophrenia study*

Description

Schizophrenia data from a randomized controlled trial with patients assigned to either drug or placebo group. "Severity of Illness" was measured, at weeks 0, 1, . . . , 6, on a four category ordered scale. Most of the observations where made on weeks 0, 1, 3, and 6.

Usage

```
data(schizo)
```

Format

A data frame with 1603 observations on 437 subjects. Five vectors contain information on

`id` patient ID.

`imps79` original response measurements on a numerical scale.

`imps79o` ordinal response on a 4 category scale, "normal or borderline mentally ill" < "mildly or moderately ill", "markedly ill", "severely or among the most extremely ill".

`tx` treatment indicator: 1 for drug, 0 for placebo.

`week` week.

Details

The documentation file was copied from the **mixcat** package and slightly modified.

Source

https://hedeker.people.uic.edu/ml.html

References

Hedeker, D. and R. Gibbons (2006). *Longitudinal Data Analysis*. New Jersey, USA: John Wiley & Sons.

tvcglm *Coefficient-wise tree-based varying coefficient regression based on generalized linear models*

Description

The tvcglm function implements the tree-based varying coefficient regression algorithm for generalized linear models introduced by Burgin and Ritschard (2017). The algorithm approximates varying coefficients by piecewise constant functions using recursive partitioning, i.e., it estimates the selected coefficients individually by strata of the value space of partitioning variables. The special feature of the provided algorithm is that it allows building for each varying coefficient an individual partition, which enhances the possibilities for model specification and to select partitioning variables individually by coefficient.

Usage

```
tvcglm(formula, data, family,
       weights, subset, offset, na.action = na.omit,
       control = tvcglm_control(), ...)

tvcglm_control(minsize = 30, mindev = 2.0,
               maxnomsplit = 5, maxordsplit = 9, maxnumsplit = 9,
               cv = TRUE, folds = folds_control("kfold", 5),
               prune = cv, fast = TRUE, center = fast,
       maxstep = 1e3, verbose = FALSE, ...)
```

Arguments

formula	a symbolic description of the model to fit, e.g., `y ~ vc(z1, z2, z3) + vc(z1, z2, by = x1) + vc(z2, z3, by = x2)` where the vc terms specify the varying fixed coefficients. The unnamed arguments within vc terms are interpreted as partitioning variables (i.e., moderators). The by argument specifies the associated predictor variable. If no such predictor variable is specified (e.g., see the first term in the above example formula), the vc term is interpreted as a varying intercept, i.e., an nonparametric estimate of the direct effect of the partitioning variables. For details, see vcrpart-formula. Note that the global intercept may be removed by a -1 term, according to the desired interpretation of the model.
family	the model family. An object of class family.
data	a data frame containing the variables in the model.
weights	an optional numeric vector of weights to be used in the fitting process.
subset	an optional logical or integer vector specifying a subset of 'data' to be used in the fitting process.
offset	this can be used to specify an a priori known component to be included in the linear predictor during fitting.

`na.action`	a function that indicates what should happen if data contain `NA`s. The default `na.action = na.omit` is listwise deletion, i.e., observations with missings on any variable are dropped. See `na.action`.
`control`	a list with control parameters as returned by `tvcglm_control`, or by `tvcm_control` for advanced users.
`minsize`	numeric (vector). The minimum sum of weights in terminal nodes.
`mindev`	numeric scalar. The minimum permitted training error reduction a split must exhibit to be considered of a new split. The main role of this parameter is to save computing time by early stopping. May be set lower for very few partitioning variables resp. higher for many partitioning variables.
`maxnomsplit, maxordsplit, maxnumsplit`	integer scalars for split candidate reduction. See `tvcm_control`
`cv`	logical scalar. Whether or not the `cp` parameter should be cross-validated. If `TRUE` `cvloss` is called.
`folds`	a list of parameters to create folds as produced by `folds_control`. Is used for cross-validation.
`prune`	logical scalar. Whether or not the initial tree should be pruned by the estimated `cp` parameter from cross-validation. Cannot be `TRUE` if `cv = FALSE`.
`fast`	logical scalar. Whether the approximative model should be used to search for the next split. The approximative search model uses only the observations of the node to split and incorporates the fitted values of the current model as offsets. Therewith the estimation is reduces to the coefficients of the added split. If `FALSE`, the accurate search model is used.
`center`	logical integer. Whether the predictor variables of update models during the grid search should be centered. Note that `TRUE` will not modify the predictors of the fitted model.
`maxstep`	integer. The maximum number of iterations i.e. number of splits to be processed.
`verbose`	logical. Should information about the fitting process be printed to the screen?
`...`	additional arguments passed to the fitting function `fit` or to `tvcm_control`.

Details

`tvcglm` processes two stages. The first stage, called partitioning stage, builds overly fine partitions for each `vc` term; the second stage, called pruning stage, selects the best-sized partitions by collapsing inner nodes. For details on the pruning stage, see `tvcm-assessment`. The partitioning stage iterates the following steps:

1. Fit the current generalized linear model
   `y ~ NodeA:x1 + ... + NodeK:xK`
   with `glm`, where `Nodek` is a categorical variable with terminal node labels for the k-th varying coefficient.
2. Search the globally best split among the candidate splits by an exhaustive -2 likelihood training error search that cycles through all possible splits.
3. If the -2 likelihood training error reduction of the best split is smaller than `mindev` or there is no candidate split satisfying the minimum node size `minsize`, stop the algorithm.

4. Else incorporate the best split and repeat the procedure.

The partitioning stage selects, in each iteration, the split that maximizes the -2 likelihood training error reduction, compared to the current model. The default stopping parameters are `minsize = 30` (a minimum node size of 30) and `mindev = 2` (the training error reduction of the best split must be larger than two to continue).

The algorithm implements a number of split point reduction methods to decrease the computational complexity. See the arguments `maxnomsplit`, `maxordsplit` and `maxnumsplit`.

The algorithm can be seen as an extension of CART (Breiman et. al., 1984) and PartReg (Wang and Hastie, 2014), with the new feature that partitioning can be processed coefficient-wise.

Value

An object of class `tvcm`

Author(s)

Reto Burgin

References

Breiman, L., J. H. Friedman, R. A. Olshen and C.J. Stone (1984). *Classification and Regression Trees*. New York, USA: Wadsworth.

Wang, J. C., Hastie, T. (2014), Boosted Varying-Coefficient Regression Models for Product Demand Prediction, *Journal of Computational and Graphical Statistics*, **23**(2), 361-382.

Burgin, R. and G. Ritschard (2017), Coefficient-Wise Tree-Based Varying Coefficient Regression with vcrpart. *Journal of Statistical Software*, **80**(6), 1–33.

See Also

`tvcm_control`, `tvcm-methods`, `tvcm-plot`, `tvcm-plot`, `tvcm-assessment`, `fvcglm`, `glm`

Examples

```
## --------------------------------------------------------------------- #
## Example: Moderated effect of education on poverty
##
## The algorithm is used to find out whether the effect of high
## education 'EduHigh' on poverty 'Poor' is moderated by the civil
## status 'CivStat'. We specify two 'vc' terms in the logistic
## regression model for 'Poor': a first that accounts for the direct
## effect of 'CivStat' and a second that accounts for the moderation of
## 'CivStat' on the relation between 'EduHigh' and 'Poor'. We use here
## the 2-stage procedure with a partitioning- and a pruning stage as
## described in Burgin and Ritschard (2017).
## --------------------------------------------------------------------- #

data(poverty)
poverty$EduHigh <- 1 * (poverty$Edu == "high")
```

```
## fit the model
model.Pov <-
  tvcglm(Poor ~ -1 +  vc(CivStat) + vc(CivStat, by = EduHigh) + NChild,
         family = binomial(), data = poverty, subset = 1:200,
         control = tvcm_control(verbose = TRUE, papply = lapply,
           folds = folds_control(K = 1, type = "subsampling", seed = 7)))

## diagnosis
plot(model.Pov, "cv")
plot(model.Pov, "coef")
summary(model.Pov)
splitpath(model.Pov, steps = 1:3)
prunepath(model.Pov, steps = 1)
```

---

tvcm *Tree-based varying coefficient regression models*

Description

tvcm is the general implementation for tree-based varying coefficient regression. It may be used to combine the two different algorithms tvcolmm and tvcglm.

Usage

```
tvcm(formula, data, fit, family,
     weights, subset, offset, na.action = na.omit,
     control = tvcm_control(), fitargs, ...)
```

Arguments

formula	a symbolic description of the model to fit, e.g., y ~ vc(z1, z2) + vc(z1, z2, by = x) where vc specifies the varying coefficients. See vcrpart-formula.
fit	a character string or a function that specifies the fitting function, e.g., olmm or glm.
family	the model family, e.g., an object of class family.olmm or family.
data	a data frame containing the variables in the model.
weights	an optional numeric vector of weights to be used in the fitting process.
subset	an optional logical or integer vector specifying a subset of 'data' to be used in the fitting process.
offset	this can be used to specify an a priori known component to be included in the linear predictor during fitting.
na.action	a function that indicates what should happen if data contain NAs. The default na.action = na.omit is listwise deletion, i.e., observations with missings on any variable are dropped. See na.action.

control — a list with control parameters as returned by tvcm_control.

fitargs — additional arguments passed to the fitting function fit.

... — additional arguments passed to the fitting function fit. Note that using the fitargs argument is the preferred way to for this.

Details

TVCM partitioning works as follows: In each iteration we fit the current model and select a binary split for one of the current terminal nodes. The selection requires 4 decisions: the vc term, the node, the variable and the cutpoint in the selected variable. The algorithm starts with $M_k = 1$ node for each of the K vc terms and iterates until the criteria defined by control are reached, see tvcm_control. For the specific criteria for the split selection, see tvcolmm and tvcglm.

Alternative tree-based algorithm to tvcm are the MOB (Zeileis et al., 2008) and the PartReg (Wang and Hastie, 2014) algorithms. The MOB algorithm is implemented by the mob function in the packages **party** and **partykit**. For smoothing splines and kernel regression approaches to varying coefficients, see the packages **mgcv**, **svcm**,**mboost** or **np**.

The tvcm function builds on the software infrastructure of the **partykit** package. The authors are grateful for these codes.

Value

An object of class tvcm. The tvcm class itself is based on the party class of the **partykit** package. The most important slots are:

node — an object of class partynode.

data — a data.frame. The model frame with all variables for partitioning.

fitted — an optional data.frame containing at least the fitted terminal node identifiers as element (fitted). In addition, weights may be contained as element (weights) and responses as (response).

info — additional information including control, model and data (all untransformed data, without missings).

Author(s)

Reto Burgin

References

Zeileis, A., T. Hothorn, and K. Hornik (2008). Model-Based Recursive Partitioning. *Journal of Computational and Graphical Statistics*, **17**(2), 492–514.

Wang, J. C. and T. Hastie (2014), Boosted Varying-Coefficient Regression Models for Product Demand Prediction, *Journal of Computational and Graphical Statistics*, **23**(2), 361–382.

Hothorn, T. and A. Zeileis (2014). partykit: A Modular Toolkit for Recursive Partytioning in R. In *Working Papers in Economics and Statistics, Research Platform Empirical and Experimental Economics*, Number 2014-10. Universitaet Innsbruck.

Burgin R. and Ritschard G. (2015), Tree-Based Varying Coefficient Regression for Longitudinal Ordinal Responses. *Computational Statistics & Data Analysis*, **86**, 65–80.

Burgin, R. A. (2015b). Tree-based methods for moderated regression with application to longitudinal data. PhD thesis. University of Geneva.

Burgin, R. and G. Ritschard (2017), Coefficient-Wise Tree-Based Varying Coefficient Regression with vcrpart. *Journal of Statistical Software*, **80**(6), 1–33.

See Also

tvcolmm, tvcglm, tvcm_control, tvcm-methods, tvcm-plot, tvcm-assessment

Examples

```
## --------------------------------------------------------------------- #
## Example 1: Moderated effect of education on poverty
##
## See the help of 'tvcglm'.
## --------------------------------------------------------------------- #

data(poverty)
poverty$EduHigh <- 1 * (poverty$Edu == "high")

## fit the model
model.Pov <-
  tvcm(Poor ~ -1 +  vc(CivStat) + vc(CivStat, by = EduHigh) + NChild,
         family = binomial(), data = poverty, subset = 1:200,
         control = tvcm_control(verbose = TRUE, papply = "lapply",
           folds = folds_control(K = 1, type = "subsampling", seed = 7)))

## diagnosis
plot(model.Pov, "cv")
plot(model.Pov, "coef")
summary(model.Pov)
splitpath(model.Pov, steps = 1:3)
prunepath(model.Pov, steps = 1)


## --------------------------------------------------------------------- #
## Example 2: Moderated effect effect of unemployment
##
## See the help of 'tvcolmm'.
## --------------------------------------------------------------------- #

data(unemp)

## fit the model
model.UE <-
  tvcm(GHQL ~ -1 +
          vc(AGE, FISIT, GENDER, UEREGION, by = UNEMP, intercept = TRUE) +
          re(1|PID),
       data = unemp, control = tvcm_control(sctest = TRUE),
       family = cumulative())
```

```
## diagnosis (no cross-validation was performed since 'sctest = TRUE')
plot(model.UE, "coef")
summary(model.UE)
splitpath(model.UE, steps = 1, details = TRUE)
```

---

tvcm-assessment *Model selection utility functions for* tvcm *objects.*

Description

Pruning, cross-validation to find the optimal pruning parameter and computing validation set errors for tvcm objects.

Usage

```
## S3 method for class 'tvcm'
prune(tree, cp = NULL, alpha = NULL, maxstep = NULL,
      terminal = NULL, original = FALSE, ...)

## S3 method for class 'tvcm'
prunepath(tree, steps = 1L, ...)

## S3 method for class 'tvcm'
cvloss(object, folds = folds_control(), ...)

folds_control(type = c("kfold", "subsampling", "bootstrap"),
      K = ifelse(type == "kfold", 5, 100),
      prob = 0.5, weights = c("case", "freq"),
      seed = NULL)

## S3 method for class 'cvloss.tvcm'
plot(x, legend = TRUE, details = TRUE, ...)

## S3 method for class 'tvcm'
oobloss(object, newdata = NULL, weights = NULL,
        fun = NULL, ...)
```

Arguments

object, tree	an object of class tvcm.
cp	numeric scalar. The complexity parameter to be cross-validated resp. the penalty with which the model should be pruned.
alpha	numeric significance level. Represents the stopping parameter for tvcm objects grown with sctest = TRUE, see tvcm_control. A node is splitted when the p value for any coefficient stability test in that node falls below alpha.

`maxstep`	integer. The maximum number of steps of the algorithm.
`terminal`	a list of integer vectors with the ids of the nodes the inner nodes to be set to terminal nodes. The length of the list must be equal the number of partitions.
`original`	logical scalar. Whether pruning should be based on the trees from partitioning rather than on the current trees.
`steps`	integer vector. The iteration steps from which information should be extracted.
`folds`	a list with control arguments as produced by `folds_control`.
`type`	character string. The type of sampling scheme to be used to divide the data of the input model in a learning and a validation set.
`K`	integer scalar. The number of folds.
`weights`	for `folds_control`, a character that defines whether the weights of `object` are case weights or frequencies of cases; for `oobloss`, a numeric vector of weights corresponding to the rows of `newdata`.
`prob`	numeric between 0 and 1. The probability for the `"subsampling"` cross-validation scheme.
`seed`	an numeric scalar that defines the seed.
`x`	an object of class `cvloss.tvcm` as produced by `cvloss`.
`legend`	logical scalar. Whether a legend should be added.
`details`	logical scalar. Whether the foldwise validation errors should be shown.
`newdata`	a data.frame of out-of-bag data (including the response variable). See also `predict.tvcm`.
`fun`	the loss function for the validation sets. By default, the (possibly weighted) mean of the deviance residuals as defined by the `family` of the fitted `object` is applied.
`...`	other arguments to be passed.

Details

`tvcglm` and `tvcm` processe tree-size selection by default. The functions could be interesting for advanced users.

The `prune` function is used to collapse inner nodes of the tree structures by the tuning parameter `cp`. The aim of pruning by `cp` is to collapse inner nodes to minimize the cost-complexity criterion

$$error(cp) = error(tree) + cp * complexity(tree)$$

where the training error $error(tree)$ is defined by `lossfun` and $complexity(tree)$ is defined as the total number of coefficients times `dfpar` plus the total number of splits times `dfsplit`. The function `lossfun` and the parameters `dfpar` and `dfsplit` are defined by the `control` argument of `tvcm`, see also `tvcm_control`. By default, $error(tree)$ is minus two times the total likelihood of the model and $complexity(tree)$ the number of splits. The minimization of $error(cp)$ is implemented by the following iterative backward-stepwise algorithm

1. fit all `subtree` models that collapse one inner node of the current `tree` model.

2. compute the per-complexity increase in the training error

$$dev = (error(subtree) - error(tree))/(complexity(tree) - complexity(subtree))$$

for all fitted subtree models

3. if any dev < cp then set as the tree model the subtree that minimizes dev and repeated 1 to 3, otherwise stop.

The penalty cp is generally unknown and is estimated adaptively from the data. The cvloss function implements the cross-validation method to do this. cvloss repeats for each fold the following steps

1. fit a new model with tvcm based on the training data of the fold.
2. prune the new model for increasing cp. Compute for each cp the average validation error.

Doing so yields for each fold a sequence of values for cp and a sequence of average validation errors. These sequences are then combined to a finer grid and the average validation error is averaged correspondingly. From these two sequences we choose the cp value that minimizes the validation error. Notice that the average validation error is computed as the total prediction error of the validation set divided by the sum of validation set weights. See also the argument ooblossfun in tvcm_control and the function oobloss.

The prunepath function can be used to backtrack the pruning algorithm. By default, it shows the results from collapsing inner nodes in the first iteration. The interesting iteration(s) can be selected by the steps argument. The output shows several information on the performances when collapsing inner nodes. The node labels shown in the output refer to the initial tree.

The function folds_control is used to specify the cross-validation scheme, where a random 5-fold cross-validation scheme is used by default. Alternatives are type = "subsampling" (random draws without replacement) and type = "bootstrap" (random draws with replacement). For 2-stage models (with random-effects) fitted by olmm, the subsets are based on subject-wise i.e. first stage sampling. For models where weights represent frequencies of observation units (e.g., data from contingency tables), the option weights = "freq" should be considered. cvloss returns an object for which a print and a plot generic is provided.

oobloss can be used to estimate the total prediction error for validation data (the newdata argument). By default, the loss is defined as the sum of deviance residuals, see the return value dev.resids of family resp. family.olmm. Otherwise, the loss function can be defined manually by the argument fun, see the examples below. In general the sum of deviance residual is equal the sum of the -2 log-likelihood errors. A special case is the gaussian family, where the deviance residuals are computed as $\sum_{i=1}^{N} w_i(y_i - \mu)^2$, that is, the deviance residuals ignore the term $log 2\pi\sigma^2$. Therefore, the sum of deviance residuals for the gaussian model (and possibly others) is not exactly the sum of -2 log-likelihood prediction errors (but shifted by a constant). Another special case are models with random effects. For models based on olmm, the deviance residuals are retrieved from marginal predictions (where random effects are integrated out).

Value

prune returns a tvcm object, folds_control returns a list of parameters for building a cross-validation scheme. cvloss returns an cvloss.tvcm object with at least the following components:

grid a list with values for cp.

oobloss	a matrix recording the validated loss for each value in grid for each fold.
cp.hat	numeric scalar. The tuning parameter which minimizes the cross-validated error.
folds	the used folds to extract the learning and the validation sets.

oobloss returns a scalar representing the total prediction error for newdata.

Author(s)

Reto Burgin

References

Breiman, L., J. H. Friedman, R. A. Olshen and C.J. Stone (1984). *Classification and Regression Trees*. New York, USA: Wadsworth.

Hastie, T., R. Tibshirani and J. Friedman (2001). *The Elements of Statistical Learning* (2 ed.). New York, USA: Springer-Verlag.

Burgin, R. and G. Ritschard (2017), Coefficient-Wise Tree-Based Varying Coefficient Regression with vcrpart. *Journal of Statistical Software*, **80**(6), 1–33.

See Also

tvcm

Examples

```
## --------------------------------------------------------- #
## Dummy Example:
##
## Model selection for the 'vcrpart_2' data. The example is
## merely a syntax template.
## --------------------------------------------------------- #

## load the data
data(vcrpart_2)

## fit the model
control <- tvcm_control(maxstep = 2L, minsize = 5L, cv = FALSE)
model <- tvcglm(y ~ vc(z1, z2, by = x1) + vc(z1, by = x2),
                data = vcrpart_2, family = gaussian(),
                control = control, subset = 1:75)

## cross-validate 'dfsplit'
cv <- cvloss(model, folds = folds_control(type = "kfold", K = 2, seed = 1))
cv
plot(cv)

## prune model with estimated 'cp'
model.p <- prune(model, cp = cv$cp.hat)

## backtrack pruning
```

```
prunepath(model.p, steps = 1:3)

## out-of-bag error
oobloss(model, newdata = vcrpart_2[76:100,])

## use an alternative loss function
rfun <- function(y, mu, wt) sum(abs(y - mu))
oobloss(model, newdata = vcrpart_2[76:100,], fun = rfun)
```

---

tvcm-control *Control parameters for* tvcm*.*

Description

Various parameters that control aspects for tvcm.

Usage

```
tvcm_control(minsize = 30, mindev = ifelse(sctest, 0.0, 2.0),
             sctest = FALSE, alpha = 0.05, bonferroni = TRUE,
             trim = 0.1, estfun.args = list(), nimpute = 5,
             maxnomsplit = 5, maxordsplit = 9, maxnumsplit = 9,
             maxstep = 1e3, maxwidth = Inf, maxdepth = Inf,
             lossfun = neglogLik2, ooblossfun = NULL, fast = TRUE,
             cp = 0.0, dfpar = 0.0, dfsplit = 1.0,
             cv = !sctest, folds = folds_control("kfold", 5),
             prune = cv, papply = mclapply, papply.args = list(),
             center = fast, seed = NULL, verbose = FALSE, ...)
```

Arguments

`alpha`, `bonferroni`, `trim`, `estfun.args`, `nimpute`
: See tvcolmm_control

`mindev`, `cv`, `folds`, `prune`, `center`
: See tvcglm_control

`minsize`
: numeric (vector). The minimum sum of weights in terminal nodes.

`sctest`
: logical scalar. Defines whether coefficient constancy tests should be used for the variable and node selection in each iteration.

`maxnomsplit`
: integer. For nominal partitioning variables with more the `maxnomsplit` the categories are ordered an treated as ordinal.

`maxordsplit`
: integer. The maximum number of splits of ordered partitioning variables to be evaluated.

`maxnumsplit`
: integer. The maximum number of splits of numeric partitioning variables to be evaluated.

`maxstep`
: integer. The maximum number of iterations i.e. number of splits to be processed.

`maxwidth`
: integer (vector). The maximum width of the partition(s).

`maxdepth`	integer (vector). The maximum depth of the partition(s).
`lossfun`	a function to extract the training error, typically minus two times the negative log likelihood of the fitted model (see `neglogLik2`). Is currently ignored if a `glm` model is fitted and `fast = TRUE`.
`ooblossfun`	a loss function that defines how to compute the validation error during cross-validation. The function will be assigned to the `fun` argument of `oobloss`.
`fast`	logical scalar. Whether the approximative model should be used to search for the next split. The approximative search model uses only the observations of the node to split and incorporates the fitted values of the current model as offsets. Therewith the estimation is reduces to the coefficients of the added split. If `FALSE`, the accurate search model is used.
`cp`	numeric scalar. The penalty to be multiplied with the complexity of the model during partitioning. The complexity of the model is defined as the number of coefficients times `dfpar` plus the number of splits times `dfsplit`. By default, `cp = 0` (no penalization during partitioning) and `dfpar = 0` and `dfsplit = 1` (the complexity is measured as the total number of splits). `cp` also presents the minimum evaluated value at cross-validation.
`dfpar`	numeric scalar. The degree of freedom per model coefficient. Is used to compute the complexity of the model, see `cp`.
`dfsplit`	a numeric scalar. The degree of freedom per split. Is used to compute the complexity of the model, see `cp`.
`papply`	(parallel) apply function, defaults to `mclapply`. The function will parallelize the partition stage and the evaluation of the cross-validation folds as well as the final pruning stage.
`papply.args`	a list of arguments to be passed to `papply`.
`seed`	an integer specifying which seed should be set at the beginning.
`verbose`	logical. Should information about the fitting process be printed to the screen?
`...`	further, undocumented arguments to be passed.

Value

A list of class `tvcm_control` containing the control parameters for `tvcm`.

Author(s)

Reto Burgin

See Also

`tvcolmm_control`, `tvcglm_control`, `tvcm`, `fvcm`

Examples

```
tvcm_control(minsize = 100)
```

tvcm-methods *Methods for tvcm objects*

Description

Standard methods for computing on tvcm objects.

Usage

```
## S3 method for class 'tvcm'
coef(object, ...)

## S3 method for class 'tvcm'
depth(x, root = FALSE, ...)

## S3 method for class 'tvcm'
extract(object, what = c(
              "control", "model",
              "nodes", "sctest", "p.value",
              "devgrid", "cv", "selected",
              "coef", "sd", "var"),
        steps = NULL, ...)

## S3 method for class 'tvcm'
neglogLik2(object, ...)

## S3 method for class 'tvcm'
predict(object, newdata = NULL,
        type = c("link", "response", "prob", "class",
          "node", "coef", "ranef"),
        ranef = FALSE, na.action = na.pass, ...)

## S3 method for class 'tvcm'
splitpath(tree, steps = 1L,
         details = FALSE, ...)

## S3 method for class 'tvcm'
summary(object, ...)

## S3 method for class 'tvcm'
width(x, ...)
```

Arguments

`object, tree, x`	an object of class tvcm.
`root`	logical scalar. Should the root count be counted in `depth`?

`steps`	integer vector. The iteration steps from which information should be extracted.
`newdata`	an optional data frame in which to look for variables with which to predict, if omitted, the fitted values are used.
`type`	character string. Denotes for `predict` the type of predicted value. See `predict.glm` or `predict.olmm`. `"response"` and `"prob"` are identical.
`na.action`	function determining what should be done with missing values for fixed effects in `newdata`. The default is to predict `NA`: see `na.pass`.
`ranef`	logical scalar or matrix indicating whether prediction should be based on random effects. See `predict.olmm`.
`what`	a character specifying the quantities to `extract`.
`details`	logical scalar. Whether detail results like coefficient constancy tests or loss minimizing grid search should be shown.
`...`	Additional arguments passed to the calls.

Details

The `predict` function has two additional options for the `type` argument. The option `"node"` calls the node id and `"coef"` predicts the coefficients corresponding to an observation. In cases of multiple `vc` terms for the same predictor, the coefficients are summed up.

The `splitpath` function allows to backtrack the partitioning procedure. By default, it shows which split was chosen in the first iteration. The interesting iteration(s) can be selected by the `steps` argument. With `details = TRUE` it is also possible to backtrack the coefficient constancy tests and/or the loss reduction statistics.

`summary` computes summary statistics of the fitted model, including the estimated coefficients. The varying coefficient are printed by means of a printed decision tree. Notice that in cases there is no split for the varying coefficient, the average coefficient will be among the fixed effects.

Further undocumented, available methods are: `fitted`, `formula`, `getCall`, `logLik`, `model.frame`, `nobs`, `print`, `ranef`, `resid`, and `weights`. All these methods have the same arguments as the corresponding default methods.

Value

The `coef.tvcm` and `coefficients.tvcm` methods return a `list` with model coefficients. Slot `vc` stores varying coefficients, `fe` fixed coefficients and `re` coefficients on random effects.

The `depth.tvcm` method returns a integer vector with the depth of the trees of every varying coefficient. `width.tvcm` returns a integer vector with the width of the trees.

The `extract` and `extract.tvcm` methods allow to extract further information of `tvcm` objects, such as the underlying regression model. The type of the return value depends on the input for argument `what`.

The `formula.tvcm` method extracts the model formula, which is an object of class `formula`. See also `formula`.

The methods `fitted.tvcm` and `predict.fvcm` return an object of class `numeric` or `matrix`, depending on the used model or the specification of the argument `type`.

The `getCall.tvcm` method extracts the call for fitting the model, which is an object of class `call`. See also `call`.

The `logLik.tvcm` method returns an object of class `logLik`, see also `logLik`.

The `model.frame.tvcm` method returns a `data.frame`. See also `model.frame`.

The `neglogLik2.tvcm` method returns a single numeric, see also `neglogLik2`.

The `nobs.tvcm` method extracts the number of observations used to fit the model. See also `nobs.tvcm`.

The `print.tvcm` and `summary.tvcm` methods return `NULL`.

The `ranef.tvcm` method returns an object of class `matrix` with values for the random effects. See also `ranef.olmm` and `ranef`.

The `resid.tvcm` and `residuals.tvcm` methods return a `numeric` or a `matrix`, depending on the used model or the type of residuals. See the help of the `resid` method of the used model.

The methods `splitpath` and `splitpath.tvcm` return an object of class `splitpath.tvcm` that contains information on splitting when building the tree.

The `weights.tvcm` method extracts a `numeric` vector with the model weights. See also `weights`.

Author(s)

Reto Burgin

See Also

`tvcm`, `tvcm-assessment`, `tvcm-plot`

Examples

```
## --------------------------------------------------------------------- #
## Dummy example:
##
## Apply various methods on a 'tvcm' object fitted on the 'vcrpart_2'
## data. Cross-validation is omitted to accelerate the computations.
## --------------------------------------------------------------------- #

data(vcrpart_2)

model <- tvcm(y ~ -1 + vc(z1, z2) + vc(z1, z2, by = x1) + x2,
              data = vcrpart_2, family = gaussian(), subset = 1:90,
              control = tvcm_control(cv = FALSE))

coef(model)
extract(model, "selected")
extract(model, "model")
predict(model, newdata = vcrpart_2[91:100,], type = "node")
predict(model, newdata = vcrpart_2[91:100,], type = "response")
splitpath(model, steps = 1)
summary(model, digits = 2)
```

tvcm-plot — plot *method for* tvcm *objects.*

Description

plot method and panel functions for tvcm objects.

Usage

```
## S3 method for class 'tvcm'
plot(x, type = c("default", "coef",
        "simple", "partdep", "cv"),
     main, part = NULL, drop_terminal = TRUE,
     tnex, newpage = TRUE, ask = NULL,
     pop = TRUE, gp = gpar(), ...)

panel_partdep(object, parm = NULL,
              var = NULL, ask = NULL,
              prob = NULL, neval = 50, add = FALSE,
              etalab = c("int", "char", "eta"), ...)

panel_coef(object, parm = NULL,
           id = TRUE, nobs = TRUE,
           exp = FALSE,
           plot_gp = list(),
           margins, yadj = 0.1,
           mean = FALSE, mean_gp = list(),
           conf.int = FALSE, conf.int_gp = list(),
           abbreviate = TRUE, etalab = c("int", "char", "eta"), ...)
```

Arguments

x, object	An object of class tvcm.
type	the type of the plot. Available types are "default", "simple", "coef", "partdep" and "cv".
main	character. A main title for the plot.
drop_terminal	a logical indicating whether all terminal nodes should be plotted at the bottom. See also plot.party.
tnex	a numeric value giving the terminal node extension in relation to the inner nodes. By default the value is computed adaptively to the tree size.
newpage	a logical indicating whether grid.newpage() should be called.
pop	a logical whether the viewport tree should be popped before return.
gp	graphical parameters. See gpar.
part	integer or letter. The partition i.e. varying coefficient component to be plotted.

parm	character vector (panel_partdep and panel_coef) or list of character vectors (panel_coef) with names of model coefficients corresponding to the chosen component. Indicates which coefficients should be visualized. If parm is a list, a separate panel is allocated for each list component.
var	character vector. Indicates the partitioning variables to be visualized.
ask	logical. Whether an input should be asked before printing the next panel.
prob	a probability between 0 and 1. Gives the size of the random subsample over which the coefficients are averaged. May be smaller than 1 if the sample is large.
neval	the maximal number of distinct values of the variable to be evaluated.
add	logical. Whether the panel is to be added into an active plot.
id	logical. Whether the node id should be displayed.
nobs	logical. Whether the number of observations in each node should be displayed.
exp	logical. Whether the labels in the y-axes should be the exponential of coefficients.
plot_gp	a list of graphical parameters for the panels. Includes components xlim, ylim, pch, ylab, type (the type of symbols, e.g. "b"), label (characters for ticks at the x axis), height, width, gp (a list produced by gpar). If parm is a list, plot_gp may be a nested list specifying the graphical parameters for each list component of parm. See examples.
margins	a numeric vector c(bottom, left, top, right) specifying the space on the margins for each panel. By default the values are computed adaptively to the tree size.
yadj	a numeric scalar larger than zero that increases the margin above the panel. May be useful if the edge labels are covered by the coefficient panels.
mean	logical. Whether the average coefficients over the population should be visualized.
mean_gp	list with graphical parameters for plotting the mean coefficients. Includes a component gp = gpar(...) and a component pch. See examples.
conf.int	logical. Whether confidence intervals should be visualized. These are indicative values only. They do not account for the uncertainty of model selection procedure.
conf.int_gp	a list of graphical parameters for the confidence intervals applied to arrow. Includes angle, length, ends and type. See examples.
abbreviate	logical scalar. Whether labels of coefficients should be abbreviated.
etalab	character. Whether category-specific effects should be labeled by integers of categories (default), the labels of the categories ("char") or the index of the predictor ("eta").
...	additional arguments passed to panel_partdep or panel_coef or other methods.

Details

The plot functions allow the diagnosis of fitted `tvcm` objects. `type = "default"`, `type = "coef"` and `type = "simple"` show the tree structure and coefficients in each node. `type = "partdep"` plots partial dependency plots, see Hastie et al. (2001), section 10.13.2. Finally, `type = "cv"` shows, if available, the results from cross-validation.

The functions `panel_partdep` and `panel_coef` are exported to show the additional arguments that can be passed to `...` of a `plot` call.

Notice that user-defined plots can be generated by the use of the `plot.party` function, see **partykit**.

Value

The `plot.fvcm` method returns `NULL`.

Author(s)

Reto Burgin

References

Hastie, T., R. Tibshirani and J. Friedman (2001). *The Elements of Statistical Learning* (2 ed.). New York, USA: Springer-Verlag.

See Also

`tvcm`, `tvcm-methods`

Examples

```
## --------------------------------------------------------------------- #
## Dummy example:
##
## Plotting the types "coef" and "partdep" for a 'tvcm' object fitted
## on the artificial data 'vcrpart_2'.
## --------------------------------------------------------------------- #

data(vcrpart_2)

## fit the model
model <- tvcglm(y ~ vc(z1, z2, by = x1, intercept = TRUE) + x2,
                data = vcrpart_2, family = gaussian(),
                control = tvcm_control(maxwidth = 3, minbucket = 5L))

## plot type "coef"
plot(model, "coef")

## add various (stupid) plot parameters
plot(model, "coef",
     plot_gp = list(type = "p", pch = 2, ylim = c(-4, 4),
       label = c("par1", "par2"), gp = gpar(col = "blue")),
     conf.int_gp = list(angle = 45, length = unit(2, "mm"),
```

```
      ends = "last", type = "closed"),
    mean_gp = list(pch = 16,
      gp = gpar(fontsize = 16, cex = 2, col = "red")))

## separate plots with separate plot parameters
plot(model, "coef", parm = list("(Intercept)", "x1"), tnex = 2,
     plot_gp = list(list(gp = gpar(col = "red")),
                    list(gp = gpar(col = "blue"))),
     mean_gp = list(list(gp = gpar(col = "green")),
                    list(gp = gpar(col = "yellow"))))

## plot type "partdep"
par(mfrow = c(1, 2))
plot(model, "partdep", var = "z1", ask = FALSE)
```

---

tvcolmm — *Tree-based varying coefficient regression based on ordinal and nominal two-stage linear mixed models.*

Description

The tvcolmm function implements the tree-based longitudinal varying coefficient regression algorithm proposed in Burgin and Ritschard (2015). The algorithm approximates varying fixed coefficients in the cumulative logit mixed model by a (multivariate) piecewise constant function using recursive partitioning, i.e., it estimates the fixed effect component of the model separately for strata of the value space of partitioning variables.

Usage

```
tvcolmm(formula, data, family = cumulative(),
        weights, subset, offset, na.action = na.omit,
        control = tvcolmm_control(), ...)

tvcolmm_control(sctest = TRUE, alpha = 0.05, bonferroni = TRUE,
                minsize = 50, maxnomsplit = 5, maxordsplit = 9,
                maxnumsplit = 9, fast = TRUE,
                trim = 0.1, estfun.args = list(), nimpute = 5,
                seed = NULL, maxstep = 1e3, verbose = FALSE, ...)
```

Arguments

`formula` a symbolic description of the model to fit, e.g.,

`y ~ -1 + vc(z1, ..., zL, by = x1 + ... + xP, intercept = TRUE) + re(1|id)`

where `vc` term specifies the varying fixed coefficients. Only one such `vc` term is allowed with tvcolmm (in contrast to tvcglm where multiple `vc` terms can be specified). The above example formula removes the global intercepts and adds locally varying intercepts, by adding a `-1` term and specfiying `intercept = TRUE` in the `vc` term. If varying intercepts are desired, we recommend to always

	remove the global intercepts. For more details on the formula specification, see olmm and vcrpart-formula.
family	the model family. An object of class family.olmm.
data	a data frame containing the variables in the model.
weights	an optional numeric vector of weights to be used in the fitting process.
subset	an optional logical or integer vector specifying a subset of 'data' to be used in the fitting process.
offset	this can be used to specify an a priori known component to be included in the linear predictor during fitting.
na.action	a function that indicates what should happen if data contain NAs. The default na.action = na.omit is listwise deletion, i.e., observations with missings on any variable are dropped. See na.action.
control	a list with control parameters as returned by tvcolmm_control, or by tvcm_control for advanced users.
sctest	logical scalar. Defines whether coefficient constancy tests should be used for the variable and node selection in each iteration.
alpha	numeric significance threshold between 0 and 1. A node is splitted when the smallest (possibly Bonferroni-corrected) p value for any coefficient constancy test in the current step falls below alpha.
bonferroni	logical. Indicates if and how p-values of coefficient constancy tests must be Bonferroni corrected. See details.
minsize	numeric scalar. The minimum sum of weights in terminal nodes.
maxnomsplit, maxordsplit, maxnumsplit	integer scalars for split candidate reduction. See tvcm_control.
fast	logical scalar. Whether the approximative model should be used to search for the next split. See tvcm_control.
trim	numeric between 0 and 1. Specifies the trimming parameter in coefficient constancy tests for continuous partitioning variables. See also the argument from of function supLM in package **strucchange**.
estfun.args	list of arguments to be passed to olmm_gefp. See details.
nimpute	a positive integer scalar. The number of times coefficient constancy tests should be repeated in each iteration. See details.
seed	an integer specifying which seed should be set at the beginning.
maxstep	integer. The maximum number of iterations i.e. number of splits to be processed.
verbose	logical. Should information about the fitting process be printed to the screen?
...	additional arguments passed to the fitting function fit or to tvcm_control.

Details

The tvcolmm function iterates the following steps:

1. Fit the current mixed model
   `y ~ Node:x1 + ... + Node:xP + re(1 + w1 + ... |id)`
   with olmm, where Node is a categorical variable with terminal node labels 1, ..., M.

2. Test the constancy of the fixed effects `Node:x1, ...`, separately for each moderator `z1, ...`, `zL` in each node `1, ..., M`. This yields `L` times `M` (possibly Bonferroni corrected) p-values for rejecting coefficient constancy.
3. If the minimum p-value is smaller than `alpha`, then select the node and the variable corresponding to the minimum p-value. Search and incorporate the optimal among the candidate splits in the selected node and variable by exhaustive likelihood search.
4. Else if minimum p-value is larger than `alpha`, stop the algorithm and return the current model.

The implemented coefficient constancy tests used for node and variable selection (step 2) are based on the M-fluctuation tests of Zeileis and Hornik (2007), using the observation scores of the fitted mixed model. The observation scores can be extracted by `olmm_estfun` for models fitted with `olmm`. To deal with intra-individual correlations between such observation scores, the `olmm_estfun` function decorrelates the observation scores. In cases of unbalanced data, the pre-decorrelation method requires imputation. `nimpute` gives the number of times the coefficient constancy tests are repeated in each iteration. The final p-values are then the averages of the repetations.

The algorithm combines the splitting technique of Zeileis (2008) with the technique of Hajjem et. al (2011) and Sela and Simonoff (2012) to incorporate regression trees into mixed models.

For the exhaustive search, the algorithm implements a number of split point reduction methods to decrease the computational complexity. See the arguments `maxnomsplit`, `maxordsplit` and `maxnumsplit`. By default, the algorithm also uses the approximative search model approach proposed in Burgin and Ritschard (2017). To disable this option to use the original algorithm, set `fast = FALSE` in `tvcolmm_control`.

Special attention is given to varying intercepts, i.e. the terms that account for the direct effects of the moderators. A common specification is

`y ~ -1 + vc(z1, ..., zL, by = x1 + ... + xP, intercept = TRUE) + re(1 + w1 + ... |id)`

Doing so replaces the globale intercept by local intercepts. As mentioned, if a varying intercepts are desired, we recommend to always remove the global intercept.

Value

An object of class `tvcm`

Author(s)

Reto Burgin

References

Zeileis, A., T. Hothorn, and K. Hornik (2008). Model-Based Recursive Partitioning. *Journal of Computational and Graphical Statistics*, **17**(2), 492–514.

Zeileis A., Hornik K. (2007), Generalized M-Fluctuation Tests for Parameter Instability, *Statistica Neerlandica*, **61**(4), 488–508.

Burgin R. and Ritschard G. (2015), Tree-Based Varying Coefficient Regression for Longitudinal Ordinal Responses. *Computational Statistics & Data Analysis*, **86**, 65–80.

Burgin, R. and G. Ritschard (2017), Coefficient-Wise Tree-Based Varying Coefficient Regression with vcrpart. *Journal of Statistical Software*, **80**(6), 1–33.

Sela R. and J. S. Simonoff (2012). RE-EM trees: A Data Mining Approach for Longitudinal and Clustered data, *Machine Learning* **86**(2), 169–207.

A. Hajjem, F. Bellavance and D. Larocque (2011), Mixed Effects Regression Trees for Clustered Data, *Statistics & Probability Letters* **81**(4), 451–459.

See Also

tvcm_control, tvcm-methods, tvcm-plot, fvcolmm, olmm

Examples

```
## ------------------------------------------------------------------- #
## Example: Moderated effect effect of unemployment
##
## Here we fit a varying coefficient ordinal linear mixed on the
## synthetic ordinal longitudinal data 'unemp'. The interest is whether
## the effect of unemployment 'UNEMP' on happiness 'GHQL' is moderated
## by 'AGE', 'FISIT', 'GENDER' and 'UEREGION'. 'FISIT' is the only true
## moderator. For the the partitioning we coefficient constancy tests,
## as described in Burgin and Ritschard (2015)
## ------------------------------------------------------------------- #

data(unemp)

## fit the model
model.UE <-
  tvcolmm(GHQL ~ -1 +
          vc(AGE, FISIT, GENDER, UEREGION, by = UNEMP, intercept = TRUE) +
          re(1|PID), data = unemp)

## diagnosis
plot(model.UE, "coef")
summary(model.UE)
splitpath(model.UE, steps = 1, details = TRUE)
```

---

vcrpart-demo *Synthetic data sets*

Description

Synthetic data for illustrations.

Usage

```
data(vcrpart_1)
data(vcrpart_2)
data(vcrpart_3)
data(unemp)
```

Format

`y` ordered factor. The response variable

`id`, `PID` factor. The subject identification vector.

`wave` numeric. The wave identification vector.

`treat` a dummy variable. The treatment effect.

`x1`, `x2` numeric predictor variables.

`z1`, `z2`, `z3`, `z2` moderator (partitioning) variables.

`GHQL` self rated general happiness.

`YEAR` survey year.

`UNEMP` unemployed or not.

`AGE` age.

`FISIT` self-reported financial situation.

`GENDER` gender.

`UEREGION` regional unemployment.

See Also

`olmm`, `otsplot`, `tvcm`

Examples

```
## --------------------------------------------------------- #
## generating 'vcrpart_1'
## --------------------------------------------------------- #

## create skeletton
set.seed(1)
vcrpart_1 <- data.frame(id = factor(rep(1:50, each = 4)),
                        wave = rep(1:4, 50),
                        treat = sample(0:1, 200, TRUE))

## add partitioning variables
vcrpart_1$z1 <- rnorm(50)[vcrpart_1$id]
vcrpart_1$z2 <- rnorm(200)
vcrpart_1$z3 <- factor(sample(1:2, 50, TRUE)[vcrpart_1$id])
vcrpart_1$z4 <- factor(sample(1:2, 200, TRUE))

## simulate response
eta <- 2 * vcrpart_1$treat * (vcrpart_1$z4 == "1")
eta <- eta + rnorm(50)[vcrpart_1$id] + rlogis(200)
vcrpart_1$y <- cut(-eta, c(-Inf, -1, 1, Inf), 1:3,
                   ordered_result = TRUE)


## --------------------------------------------------------- #
## generating 'vcrpart_2'
## --------------------------------------------------------- #
```

```
set.seed(1)
vcrpart_2 <- data.frame(x1 = rnorm(100),
                        x2 = rnorm(100),
                        z1 = factor(sample(1:3, 100, TRUE)),
                        z2 = factor(sample(1:3, 100, TRUE)))
vcrpart_2$y <- vcrpart_2$x1 * (vcrpart_2$z1 == "2") +
  2 * vcrpart_2$x1 * (vcrpart_2$z1 == "3")
vcrpart_2$y <- vcrpart_2$y + rnorm(100)

## --------------------------------------------------------- #
## generating 'vcrpart_3'
## --------------------------------------------------------- #

set.seed(1)
vcrpart_3 <- data.frame(x1 = rnorm(100),
                        z1 = runif(100, -pi/2, pi/2))
vcrpart_3$y <- vcrpart_3$x1 * sin(vcrpart_3$z1) + rnorm(100)

## --------------------------------------------------------- #
## generating 'unemp'
## --------------------------------------------------------- #

## create skeletton
set.seed(1)
unemp <- data.frame(PID = factor(rep(1:50, each = 4)),
                    UNEMP = rep(c(0, 0, 1, 1), 50),
                    YEAR = rep(2001:2004, 50))

## add partitioning variables
unemp$AGE <- runif(50, 25, 60)[unemp$PID] + unemp$YEAR - 2000
unemp$FISIT <- ordered(sample(1:5, 200, replace = TRUE))
unemp$GENDER <- factor(sample(c("female", "male"), 50, replace = TRUE)[unemp$PID])
unemp$UEREGION <- runif(50, 0.02, 0.1)[unemp$PID]

## simulate response
eta <- 2 * unemp$UNEMP * (unemp$FISIT == "1" | unemp$FISIT == "2")
eta <- eta + rnorm(50)[unemp$PID] + rlogis(200)
unemp$GHQL <- cut(-eta, c(-Inf, -1, 0, 1, Inf), 1:4,
                  ordered_result = TRUE)
```

`vcrpart-formula` *Special terms for formulas.*

Description

Special terms for formulas assigned to `tvcm`, `fvcm` and `olmm`.

Usage

```
fe(formula, intercept = TRUE)
re(formula, intercept = TRUE)
vc(..., by, intercept = missing(by), nuisance = character())
ce(formula)
ge(formula)
```

Arguments

`formula`	a symbolic description for the corresponding component of the formula component. See examples.
`intercept`	logical or character vector. `intercept = TRUE` (default) indicates that an intercept is incorporated. `intercept = FALSE` removes the random intercept from the formula. Note that the sometimes allowed `-1` term is ignored. The character strings `"ce"` (category-specific random intercepts) and `"ge"` (category-global random intercepts) may be used in connection with `olmm`. Intercepts have specific interpretations for `fe`, `re` and `vc`, see the details.
`...`	the names of variables that moderate (i.e. modify) the effects of the variables specified in `by`, separated by commas. It is also possibly to assign a vector that contains the variable names as characters. Note that operators like `factor(x)` are not allowed.
`by`	a formula of predictors the effects of which are moderated by the variables in `....` See `tvcm` and the examples below. Note that the `by` variable must be numeric and factor variables must be recoded to dummy variables by hand.
`nuisance`	character vector of variables in `by` which have to be estimated separately for each partition but the algorithm should not focus on this variable when searching for splits.

Details

Special formula terms to define fixed effects `fe`, varying coefficients `vc` and random effects `re`. The use of these formula terms ensures that the functions `fvcm`, `tvcm` and `olmm` fit the intended model. Some examples are given below and on the documentation pages of the fitting functions.

For all of `fvcm`, `tvcm` and `olmm`, variables which are not defined with one of `fe`, `vc` and `re` are treated as fixed effects. Intercepts can be dropped from the model by the `intercept` argument. The terms `ce` (category-specific effects) and `ge` (global effect or proportional odds effect) are designed for the function `olmm`. Notice that `tvcm` may changes, for internal reasons, the order of the terms in the specified formula. Note that you can put multiple terms within `fe`, `ge` and `ce` terms (e.g., `fe(ce(x1 + x2 + ge(x3 + x4)))`).

At present, the term `"."`, which is often use to extract all variables of the data, is ignored. As an alternative, `vc` interprets character vectors, assigned as unnamed arguments, as lists of variables of moderators to be extracted from `data`. See the examples below.

Default for intercepts in `fe` terms is `intercept = TRUE`, or `intercept = "ce"` for models fitted with `olmm`. This means that an intercept is automatically attached. Alternatives are `intercept = FALSE`, which is equal to `intercept = "none"`, and `intercept = "ge"`, which yields a global-effect intercept for models fitted with `olmm`.

Default for intercepts in `vc` is to introduce an intercept if the by argument is ignored, otherwise no intercept is introduced. Specifically, if input is specified for the by argument, then `intercept = TRUE`, or `intercept = "ce"` for models fitted by `olmm`. Alternatives are `intercept = FALSE`, which is equal to `intercept = "none"`, and `intercept = "ge"`, which yields a global-effect varying intercept.

Default for intercepts in `re` is `intercept = TRUE`, which is equal to `intercept = "ge"`. `intercept = FALSE` is equal to `intercept = "none"`. For category-specific random intercepts, use `intercept = "ge"`. See `olmm`.

Value

a list used by `tvcm`, `fvcm` and `olmm` for constructing the model matrices.

Author(s)

Reto Burgin

See Also

`tvcm`, `fvcm`, `olmm`

Examples

```
## Formula for a model with 2 fixed effects (x1 and x2) and a random
## intercept. The 're' terms indicates that an intercept is fitted for
## each level of 'id'.

formula <- y ~ fe(x1 + x2) + re(1|id)

## Formula for a model with one fixed effect and one varying coefficient
## term with 2 moderators and 2 varying coefficient predictors. 'tvcm'
## will fit one partition to model the effects of 'x2' and 'x3' as
## functions of 'z1' and 'z2'.

formula <- y ~ x1 + vc(z1, z2, by = x2 + x3, intercept = TRUE)

## Similar formula as above, but the predictors 'x2' and 'x3' have
## separate 'vc' terms. 'tvcm' will fit a separate partition for each of
## 'x2' and 'x3' to model their effects as functions of 'z1' and 'z2'.

formula <- y ~ x1 + vc(z1, z2, by = x2) + vc(z1, z2, by = x3)

## As an alternative to '.' you can define variables in a vector
vars <- c("x1", "x2", "x3")
formula <- y ~ x1 + vc(vars, by = x2) + vc(vars, by = x3)
```

Index