Package ‘pls’

September 3, 2021

Title Partial Least Squares and Principal Component Regression

Version 2.8-0

Date 2021-09-02

Author Kristian Hovde Liland [aut, cre],
Bjørn-Helge Mevik [aut],
Ron Wehrens [aut],
Paul Hiemstra [ctb]

Maintainer Kristian Hovde Liland <kristian.liland@nmbu.no>

Encoding UTF-8

LazyData yes

Description Multivariate regression methods
Partial Least Squares Regression (PLSR), Principal Component
Regression (PCR) and Canonical Powered Partial Least Squares (CPPLS).

Depends R (>= 2.10)

Imports grDevices, graphics, methods, stats

Suggests MASS, parallel, Rmpi, testthat, RUnit

License GPL-2

URL https://github.com/khliland/pls

BugReports https://github.com/khliland/pls/issues

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-09-03 04:50:06 UTC

R topics documented:

---

biplot.mvr *Biplots of PLSR and PCR Models.*

Description

Biplot method for mvr objects.

Usage

```
## S3 method for class 'mvr'
biplot(
  x,
  comps = 1:2,
  which = c("x", "y", "scores", "loadings"),
  var.axes = FALSE,
  xlabs,
  ylabs,
  main,
  ...
)
```

Arguments

`x`	an `mvr` object.
`comps`	integer vector of length two. The components to plot.
`which`	character. Which matrices to plot. One of `"x"` (X scores and loadings), `"y"` (Y scores and loadings), `"scores"` (X and Y scores) and `"loadings"` (X and Y loadings).
`var.axes`	logical. If `TRUE`, the second set of points have arrows representing them.
`xlabs`	either a character vector of labels for the first set of points, or `FALSE` for no labels. If missing, the row names of the first matrix is used as labels.
`ylabs`	either a character vector of labels for the second set of points, or `FALSE` for no labels. If missing, the row names of the second matrix is used as labels.
`main`	character. Title of plot. If missing, a title is constructed by `biplot.mvr`.
`...`	Further arguments passed on to `biplot.default`.

Details

`biplot.mvr` can also be called through the `mvr` plot method by specifying `plottype = "biplot"`.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

`mvr`, `plot.mvr`, `biplot.default`

Examples

```
data(oliveoil)
mod <- plsr(sensory ~ chemical, data = oliveoil)
## Not run:
## These are equivalent
biplot(mod)
plot(mod, plottype = "biplot")

## The four combinations of x and y points:
par(mfrow = c(2,2))
biplot(mod, which = "x") # Default
biplot(mod, which = "y")
biplot(mod, which = "scores")
biplot(mod, which = "loadings")
```

```
## End(Not run)
```

coef.mvr *Extract Information From a Fitted PLSR or PCR Model*

Description

Functions to extract information from `mvr` objects: Regression coefficients, fitted values, residuals, the model frame, the model matrix, names of the variables and components, and the X variance explained by the components.

Usage

```
## S3 method for class 'mvr'
coef(object, ncomp = object$ncomp, comps, intercept = FALSE, ...)

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

## S3 method for class 'mvr'
residuals(object, ...)

## S3 method for class 'mvr'
model.frame(formula, ...)

## S3 method for class 'mvr'
model.matrix(object, ...)

respnames(object)

prednames(object, intercept = FALSE)

compnames(object, comps, explvar = FALSE, ...)

explvar(object)
```

Arguments

`object, formula`
: an `mvr` object. The fitted model.

`ncomp, comps`
: vector of positive integers. The components to include in the coefficients or to extract the names of. See below.

`intercept`
: logical. Whether coefficients for the intercept should be included. Ignored if `comps` is specified. Defaults to `FALSE`.

`...`
: other arguments sent to underlying functions. Currently only used for `model.frame.mvr` and `model.matrix.mvr`.

explvar logical. Whether the explained X variance should be appended to the component names.

Details

These functions are mostly used inside other functions. (Functions coef.mvr, fitted.mvr and residuals.mvr are usually called through their generic functions coef, fitted and residuals, respectively.)

coef.mvr is used to extract the regression coefficients of a model, i.e. the B in $y = XB$ (for the Q in $y = TQ$ where T is the scores, see Yloadings). An array of dimension c(nxvar,nyvar,length(ncomp)) or c(nxvar,nyvar,length(comps)) is returned.

If comps is missing (or is NULL), coef()[,,ncomp[i]] are the coefficients for models with ncomp[i] components, for $i = 1, \ldots, length(ncomp)$. Also, if intercept = TRUE, the first dimension is $nxvar + 1$, with the intercept coefficients as the first row.

If comps is given, however, coef()[,,comps[i]] are the coefficients for a model with only the component comps[i], i.e. the contribution of the component comps[i] on the regression coefficients.

fitted.mvr and residuals.mvr return the fitted values and residuals, respectively. If the model was fitted with na.action = na.exclude (or after setting the default na.action to "na.exclude" with options), the fitted values (or residuals) corresponding to excluded observations are returned as NA; otherwise, they are omitted.

model.frame.mvr returns the model frame; i.e. a data frame with all variables neccessary to generate the model matrix. See model.frame for details.

model.matrix.mvr returns the (possibly coded) matrix used as X in the fitting. See model.matrix for details.

prednames, respnames and compnames extract the names of the X variables, responses and components, respectively. With intercept = TRUE in prednames, the name of the intercept variable (i.e. "(Intercept)") is returned as well. compnames can also extract component names from score and loading matrices. If explvar = TRUE in compnames, the explained variance for each component (if available) is appended to the component names. For optimal formatting of the explained variances when not all components are to be used, one should specify the desired components with the argument comps.

explvar extracts the amount of X variance (in per cent) explained by each component in the model. It can also handle score and loading matrices returned by scores and loadings.

Value

coef.mvr returns an array of regression coefficients.

fitted.mvr returns an array with fitted values.

residuals.mvr returns an array with residuals.

model.frame.mvr returns a data frame.

model.matrix.mvr returns the X matrix.

prednames, respnames and compnames return a character vector with the corresponding names.

explvar returns a numeric vector with the explained variances, or NULL if not available.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

mvr, coef, fitted, residuals, model.frame, model.matrix, na.omit

Examples

```
data(yarn)
mod <- pcr(density ~ NIR, data = yarn[yarn$train,], ncomp = 5)
B <- coef(mod, ncomp = 3, intercept = TRUE)
## A manual predict method:
stopifnot(drop(B[1,,] + yarn$NIR[!yarn$train,] %*% B[-1,,]) ==
          drop(predict(mod, ncomp = 3, newdata = yarn[!yarn$train,])))

## Note the difference in formatting:
mod2 <- pcr(density ~ NIR, data = yarn[yarn$train,])
compnames(mod2, explvar = TRUE)[1:3]
compnames(mod2, comps = 1:3, explvar = TRUE)
```

coefplot *Plot Regression Coefficients of PLSR and PCR models*

Description

Function to plot the regression coefficients of an mvr object.

Usage

```
coefplot(
  object,
  ncomp = object$ncomp,
  comps,
  intercept = FALSE,
  separate = FALSE,
  se.whiskers = FALSE,
  nCols,
  nRows,
  labels,
  type = "l",
  lty,
  lwd = NULL,
  pch,
  cex = NULL,
  col,
  legendpos,
  xlab = "variable",
  ylab = "regression coefficient",
  main,
  pretty.xlabels = TRUE,
  xlim,
  ylim,
  ask = nRows * nCols < nPlots && dev.interactive(),
  ...
)
```

Arguments

`object` an `mvr` object. The fitted model.

`ncomp, comps` vector of positive integers. The components to plot. See `coef.mvr` for details.

`intercept` logical. Whether coefficients for the intercept should be plotted. Ignored if `comps` is specified. Defaults to `FALSE`. See `coef.mvr` for details.

`separate` logical. If `TRUE`, coefficients for different model sizes are blotted in separate plots.

`se.whiskers` logical. If `TRUE`, whiskers at plus/minus 1 estimated standard error are added to the plot. This is only available if the model was cross-validated with `jackknife = TRUE`. Also, in the current implementation, `intercept` must be `FALSE`, and `separate` must be `TRUE` if `length(ncomp) > 1`.

`nCols, nRows` integer. The number of coloumns and rows the plots will be laid out in. If not specified, `coefplot` tries to be intelligent.

`labels` optional. Alternative x axis labels. See Details.

`type` character. What type of plot to make. Defaults to `"l"` (lines). Alternative types include `"p"` (points) and `"b"` (both). See `plot` for a complete list of types.

`lty` vector of line types (recycled as neccessary). Line types can be specified as integers or character strings (see `par` for the details).

`lwd` vector of positive numbers (recycled as neccessary), giving the width of the lines.

`pch` plot character. A character string or a vector of single characters or integers (recycled as neccessary). See `points` for all alternatives.

`cex` numeric vector of character expansion sizes (recycled as neccessary) for the plotted symbols.

`col` character or integer vector of colors for plotted lines and symbols (recycled as neccessary). See `par` for the details.

`legendpos` Legend position. Optional. Ignored if `separate` is `TRUE`. If present, a legend is drawn at the given position. The position can be specified symbolically (e.g., `legendpos = "topright"`). This requires >= 2.1.0. Alternatively, the position can be specified explicitly (`legendpos = t(c(x,y))`) or interactively (`legendpos = locator()`). This only works well for plots of single-response models.

`xlab, ylab`	titles for x and y axes. Typically character strings, but can be expressions (e.g., `expression(R^2)` or lists. See `title` for details.
`main`	optional main title for the plot. See Details.
`pretty.xlabels`	logical. If `TRUE`, `coefplot` tries to plot the x labels more nicely. See Details.
`xlim, ylim`	optional vector of length two, with the x or y limits of the plot.
`ask`	logical. Whether to ask the user before each page of a plot.
`...`	Further arguments sent to the underlying plot functions.

Details

`coefplot` handles multiple responses by making one plot for each response. If `separate` is `TRUE`, separate plots are made for each combination of model size and response. The plots are laid out in a rectangular fashion.

If `legendpos` is given, a legend is drawn at the given position (unless `separate` is `TRUE`).

The argument `labels` can be a vector of labels or one of `"names"` and `"numbers"`. The labels are used as x axis labels. If `labels` is `"names"` or `"numbers"`, the variable names are used as labels, the difference being that with `"numbers"`, the variable names are converted to numbers, if possible. Variable names of the forms ‘`"number"`’ or ‘`"number text"`’ (where the space is optional), are handled.

The argument `main` can be used to specify the main title of the plot. It is handled in a non-standard way. If there is only on (sub) plot, `main` will be used as the main title of the plot. If there is *more* than one (sub) plot, however, the presence of `main` will produce a corresponding ‘global’ title on the page. Any graphical parametres, e.g., `cex.main`, supplied to `coefplot` will only affect the ‘ordinary’ plot titles, not the ‘global’ one. Its appearance can be changed by setting the parameters with `par`, which will affect *both* titles. (To have different settings for the two titles, one can override the `par` settings with arguments to `coefplot`.)

The argument `pretty.xlabels` is only used when `labels` is specified. If `TRUE` (default), the code tries to use a ‘pretty’ selection of labels. If `labels` is `"numbers"`, it also uses the numerical values of the labels for horisontal spacing. If one has excluded parts of the spectral region, one might therefore want to use `pretty.xlabels = FALSE`.

When `separate` is `TRUE`, the arguments `lty`, `col`, and `pch` default to their `par()` setting. Otherwise, the default for all of them is `1:nLines`, where `nLines` is the number of model sizes specified, i.e., the length of `ncomp` or `comps`.

The function can also be called through the `mvr` plot method by specifying `plottype = "coefficients"`.

Note

`legend` has many options. If you want greater control over the appearance of the legend, omit the `legendpos` argument and call `legend` manually.

The handling of `labels` and `pretty.xlabels` is experimental.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

mvr, plot.mvr, coef.mvr, plot, legend

Examples

```
data(yarn)
mod.nir <- plsr(density ~ NIR, ncomp = 8, data = yarn)
## Not run:
coefplot(mod.nir, ncomp = 1:6)
plot(mod.nir, plottype = "coefficients", ncomp = 1:6) # Equivalent to the previous
## Plot with legend:
coefplot(mod.nir, ncom = 1:6, legendpos = "bottomright")

## End(Not run)

data(oliveoil)
mod.sens <- plsr(sensory ~ chemical, ncomp = 4, data = oliveoil)
## Not run: coefplot(mod.sens, ncomp = 2:4, separate = TRUE)
```

cppls.fit *CPPLS (Indahl et al.)*

Description

Fits a PLS model using the CPPLS algorithm.

Usage

```
cppls.fit(
  X,
  Y,
  ncomp,
  Y.add = NULL,
  center = TRUE,
  stripped = FALSE,
  lower = 0.5,
  upper = 0.5,
  trunc.pow = FALSE,
  weights = NULL,
  ...
)
```

Arguments

`X` a matrix of observations. `NA`s and `Inf`s are not allowed.

`Y` a vector or matrix of responses. `NA`s and `Inf`s are not allowed.

`ncomp` the number of components to be used in the modelling.

`Y.add` a vector or matrix of additional responses containing relevant information about the observations.

`center` logical, determines if the X and Y matrices are mean centered or not. Default is to perform mean centering.

`stripped` logical. If `TRUE` the calculations are stripped as much as possible for speed; this is meant for use with cross-validation or simulations when only the coefficients are needed. Defaults to `FALSE`.

`lower` a vector of lower limits for power optimisation. Defaults to `0.5`.

`upper` a vector of upper limits for power optimisation. Defaults to `0.5`.

`trunc.pow` logical. If `TRUE` an experimental alternative power algorithm is used. (Optional)

`weights` a vector of individual weights for the observations. (Optional)

`...` other arguments. Currently ignored.

Details

This function should not be called directly, but through the generic functions `cppls` or `mvr` with the argument `method="cppls"`. Canonical Powered PLS (CPPLS) is a generalisation of PLS incorporating discrete and continuous responses (also simultaneously), additional responses, individual weighting of observations and power methodology for sharpening focus on groups of variables. Depending on the input to `cppls` it can produce the following special cases:

- PLS: uni-response continuous `Y`
- PPLS: uni-response continuous `Y`, `(lower || upper) != 0.5`
- PLS-DA (using correlation maximisation - B/W): dummy-coded descrete response `Y`
- PPLS-DA: dummy-coded descrete response `Y`, `(lower || upper) != 0.5`
- CPLS: multi-response `Y` (continuous, discrete or combination)
- CPPLS: multi-response `Y` (continuous, discrete or combination), `(lower || upper) != 0.5`

The name "canonical" comes from canonical correlation analysis which is used when calculating vectors of loading weights, while "powered" refers to a reparameterisation of the vectors of loading weights which can be optimised over a given interval.

Value

A list containing the following components is returned:

`coefficients` an array of regression coefficients for 1, ..., `ncomp` components. The dimensions of `coefficients` are `c(nvar,npred,ncomp)` with `nvar` the number of `X` variables and `npred` the number of variables to be predicted in `Y`.

`scores` a matrix of scores.

`loadings` a matrix of loadings.

`loading.weights`
a matrix of loading weights.

`Yscores` a matrix of Y-scores.

`Yloadings` a matrix of Y-loadings.

`projection` the projection matrix used to convert X to scores.

`Xmeans` a vector of means of the X variables.

`Ymeans` a vector of means of the Y variables.

`fitted.values` an array of fitted values. The dimensions of `fitted.values` are `c(nobj,npred,ncomp)` with `nobj` the number samples and `npred` the number of Y variables.

`residuals` an array of regression residuals. It has the same dimensions as `fitted.values`.

`Xvar` a vector with the amount of X-variance explained by each component.

`Xtotvar` total variance in `X`.

`gammas` gamma-values obtained in power optimisation.

`canonical.correlations`
Canonical correlation values from the calculations of loading weights.

`A` matrix containing vectors of weights `a` from canonical correlation (`cor(Za,Yb)`).

`smallNorms` vector of indices of explanatory variables of length close to or equal to 0.

If `stripped` is `TRUE`, only the components `coefficients`, `Xmeans`, `Ymeans` and `gammas` are returned.

Author(s)

Kristian Hovde Liland

References

Indahl, U. (2005) A twist to partial least squares regression. *Journal of Chemometrics*, **19**, 32–44.

Liland, K.H and Indahl, U.G (2009) Powered partial least squares discriminant analysis, *Journal of Chemometrics*, **23**, 7–18.

Indahl, U.G., Liland, K.H. and Næs, T. (2009) Canonical partial least squares - a unified PLS approach to classification and regression problems. *Journal of Chemometrics*, **23**, 495–504.

See Also

`mvr` `plsr` `pcr` `widekernelpls.fit` `simpls.fit` `oscorespls.fit`

Examples

```
data(mayonnaise)
# Create dummy response
mayonnaise$dummy <-
    I(model.matrix(~y-1, data.frame(y = factor(mayonnaise$oil.type))))

# Predict CPLS scores for test data
may.cpls <- cppls(dummy ~ NIR, 10, data = mayonnaise, subset = train)
may.test <- predict(may.cpls, newdata = mayonnaise[!mayonnaise$train,], type = "score")

# Predict CPLS scores for test data (experimental used design as additional Y information)
```

```
may.cpls.yadd <- cppls(dummy ~ NIR, 10, data = mayonnaise, subset = train, Y.add=design)
may.test.yadd <- predict(may.cpls.yadd, newdata = mayonnaise[!mayonnaise$train,], type = ”score”)

# Classification by linear discriminant analysis (LDA)
library(MASS)
error <- matrix(ncol = 10, nrow = 2)
dimnames(error) <- list(Model = c('CPLS', 'CPLS (Y.add)'), ncomp = 1:10)
for (i in 1:10) {
    fitdata1 <- data.frame(oil.type = mayonnaise$oil.type[mayonnaise$train],
                           NIR.score = I(may.cpls$scores[,1:i,drop=FALSE]))
    testdata1 <- data.frame(oil.type = mayonnaise$oil.type[!mayonnaise$train],
                            NIR.score = I(may.test[,1:i,drop=FALSE]))
    error[1,i] <-
        (42 - sum(predict(lda(oil.type ~ NIR.score, data = fitdata1),
                  newdata = testdata1)$class == testdata1$oil.type)) / 42
    fitdata2 <- data.frame(oil.type = mayonnaise$oil.type[mayonnaise$train],
                           NIR.score = I(may.cpls.yadd$scores[,1:i,drop=FALSE]))
    testdata2 <- data.frame(oil.type = mayonnaise$oil.type[!mayonnaise$train],
                            NIR.score = I(may.test.yadd[,1:i,drop=FALSE]))
    error[2,i] <-
        (42 - sum(predict(lda(oil.type ~ NIR.score, data = fitdata2),
                  newdata = testdata2)$class == testdata2$oil.type)) / 42
}
round(error,2)
```

crossval *Cross-validation of PLSR and PCR models*

Description

A “stand alone” cross-validation function for mvr objects.

Usage

```
crossval(
  object,
  segments = 10,
  segment.type = c(”random”, ”consecutive”, ”interleaved”),
  length.seg,
  jackknife = FALSE,
  trace = 15,
  ...
)
```

Arguments

object an mvr object; the regression to cross-validate.

segments the number of segments to use, or a list with segments (see below).

`segment.type` the type of segments to use. Ignored if `segments` is a list.

`length.seg` Positive integer. The length of the segments to use. If specified, it overrides `segments` unless `segments` is a list.

`jackknife` logical. Whether jackknifing of regression coefficients should be performed.

`trace` if `TRUE`, tracing is turned on. If numeric, it denotes a time limit (in seconds). If the estimated total time of the cross-validation exceeds this limit, tracing is turned on.

`...` additional arguments, sent to the underlying fit function.

Details

This function performs cross-validation on a model fit by `mvr`. It can handle models such as `plsr(y ~ msc(X),...{})` or other models where the predictor variables need to be recalculated for each segment. When recalculation is not needed, the result of `crossval(mvr(...{}))` is identical to `mvr(...{},validation = "CV")`, but slower.

Note that to use `crossval`, the data *must* be specified with a `data` argument when fitting `object`.

If `segments` is a list, the arguments `segment.type` and `length.seg` are ignored. The elements of the list should be integer vectors specifying the indices of the segments. See `cvsegments` for details.

Otherwise, segments of type `segment.type` are generated. How many segments to generate is selected by specifying the number of segments in `segments`, or giving the segment length in `length.seg`. If both are specified, `segments` is ignored.

If `jackknife` is `TRUE`, jackknifed regression coefficients are returned, which can be used for for variance estimation (`var.jack`) or hypothesis testing (`jack.test`).

When tracing is turned on, the segment number is printed for each segment.

By default, the cross-validation will be performed serially. However, it can be done in parallel using functionality in the `parallel` package by setting the option `parallel` in `pls.options`. See `pls.options` for the different ways to specify the parallelism. See also Examples below.

Value

The supplied `object` is returned, with an additional component `validation`, which is a list with components

`method` euqals `"CV"` for cross-validation.

`pred` an array with the cross-validated predictions.

`coefficients` (only if `jackknife` is `TRUE`) an array with the jackknifed regression coefficients. The dimensions correspond to the predictors, responses, number of components, and segments, respectively.

`PRESS0` a vector of PRESS values (one for each response variable) for a model with zero components, i.e., only the intercept.

`PRESS` a matrix of PRESS values for models with 1, …, `ncomp` components. Each row corresponds to one response variable.

`adj` a matrix of adjustment values for calculating bias corrected MSEP. `MSEP` uses this.

`segments` the list of segments used in the cross-validation.

`ncomp` the number of components.

`gammas` if method `cppls` is used, gamma values for the powers of each CV segment are returned.

Note

The `PRESS0` is always cross-validated using leave-one-out cross-validation. This usually makes little difference in practice, but should be fixed for correctness.

The current implementation of the jackknife stores all jackknife-replicates of the regression coefficients, which can be very costly for large matrices. This might change in a future version.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

Mevik, B.-H., Cederkvist, H. R. (2004) Mean Squared Error of Prediction (MSEP) Estimates for Principal Component Regression (PCR) and Partial Least Squares Regression (PLSR). *Journal of Chemometrics*, **18**(9), 422–429.

See Also

`mvr` `mvrCv` `cvsegments` `MSEP` `var.jack` `jack.test`

Examples

```
data(yarn)
yarn.pcr <- pcr(density ~ msc(NIR), 6, data = yarn)
yarn.cv <- crossval(yarn.pcr, segments = 10)
## Not run: plot(MSEP(yarn.cv))

## Not run:
## Parallelised cross-validation, using transient cluster:
pls.options(parallel = 4) # use mclapply (not available on Windows)
pls.options(parallel = quote(parallel::makeCluster(4, type = "PSOCK"))) # parLapply
## A new cluster is created and stopped for each cross-validation:
yarn.cv <- crossval(yarn.pcr)
yarn.loocv <- crossval(yarn.pcr, length.seg = 1)

## Parallelised cross-validation, using persistent cluster:
library(parallel)
## This creates the cluster:
pls.options(parallel = makeCluster(4, type = "FORK")) # not available on Windows
pls.options(parallel = makeCluster(4, type = "PSOCK"))
## The cluster can be used several times:
yarn.cv <- crossval(yarn.pcr)
yarn.loocv <- crossval(yarn.pcr, length.seg = 1)
## The cluster should be stopped manually afterwards:
```

```
stopCluster(pls.options()$parallel)

## Parallelised cross-validation, using persistent MPI cluster:
## This requires the packages snow and Rmpi to be installed
library(parallel)
## This creates the cluster:
pls.options(parallel = makeCluster(4, type = "MPI"))
## The cluster can be used several times:
yarn.cv <- crossval(yarn.pcr)
yarn.loocv <- crossval(yarn.pcr, length.seg = 1)
## The cluster should be stopped manually afterwards:
stopCluster(pls.options()$parallel)
## It is good practice to call mpi.exit() or mpi.quit() afterwards:
mpi.exit()

## End(Not run)
```

cvsegments *Generate segments for cross-validation*

Description

The function generates a list of segments for cross-validation. It can generate random, consecutive and interleaved segments, and supports keeping replicates in the same segment.

Usage

```
cvsegments(
  N,
  k,
  length.seg = ceiling(N/k),
  nrep = 1,
  type = c("random", "consecutive", "interleaved"),
  stratify = NULL
)
```

Arguments

N	Integer. The number of rows in the data set.
k	Integer. The number of segments to return.
length.seg	Integer. The length of the segments. If given, it overrides k.
nrep	Integer. The number of (consecutive) rows that are replicates of the same object. Replicates will always be kept in the same segment.
type	One of "random", "consecutive" and "interleaved". The type of segments to generate. Default is "random".
stratify	Either a list of indices or an integer vector indicating which stratum each sample (or set of replicates) belongs to (see Details).

Details

If length.seg is specified, it is used to calculate the number of segments to generate. Otherwise k must be specified. If $k * length.seg \neq N$, the $k * length.seg - N$ last segments will contain only $length.seg - 1$ indices.

If type is "random", the indices are allocated to segments in random order. If it is "consecutive", the first segment will contain the first $length.seg$ indices, and so on. If type is "interleaved", the first segment will contain the indices $1, length.seg + 1, 2 * lenght.seg + 1, \ldots, (k - 1) * length.seg + 1$, and so on.

If $nrep >$, it is assumed that each nrep consecutive rows are replicates (repeated measurements) of the same object, and care is taken that replicates are never put in different segments.

Warning: If k does not divide N, a specified length.seg does not divide N, or nrep does not divide length.seg, the number of segments and/or the segment length will be adjusted as needed. Warnings are printed for some of these cases, and one should always inspect the resulting segments to make sure they are as expected.

Stratification of samples is enabled by the stratify argument. This is useful if there are subgroups in the data set that should have a proportional representation in the cross-validation segments or if the response is categorical (classifiation). If stratify is combined with nrep, stratify corresponds to the sets of replicates (see example).

Value

A list of vectors. Each vector contains the indices for one segment. The attribute "incomplete" contains the number of incomplete segments, and the attribute "type" contains the type of segments.

Author(s)

Bjørn-Helge Mevik, Ron Wehrens and Kristian Hovde Liland

Examples

```
## Segments for 10-fold randomised cross-validation:
cvsegments(100, 10)

## Segments with four objects, taken consecutive:
cvsegments(60, length.seg = 4, type = "cons")

## Incomplete segments
segs <- cvsegments(50, length.seg = 3)
attr(segs, "incomplete")

## Leave-one-out cross-validation:
cvsegments(100, 100)
## Leave-one-out with variable/unknown data set size n:
n <- 50
cvsegments(n, length.seg = 1)

## Data set with replicates
```

```
cvsegments(100, 25, nrep = 2)
## Note that rows 1 and 2 are in the same segment, rows 3 and 4 in the
## same segment, and so on.

## Stratification
cvsegments(10, 3, type = "consecutive", stratify = c(rep(1,7), rep(2,3)))
## Note that the last three samples are spread across the segments
## according to the stratification vector.
cvsegments(20, 3, type = "consecutive", nrep = 2, stratify = c(rep(1,7), rep(2,3)))
## Note the length of stratify matching number of replicate sets, not samples.
```

`gasoline` *Octane numbers and NIR spectra of gasoline*

Description

A data set with NIR spectra and octane numbers of 60 gasoline samples. The NIR spectra were measured using diffuse reflectance as log(1/R) from 900 nm to 1700 nm in 2 nm intervals, giving 401 wavelengths. Many thanks to John H. Kalivas.

Format

A data frame with 60 observations on the following 2 variables.

octane a numeric vector. The octane number.

NIR a matrix with 401 columns. The NIR spectrum.

Source

Kalivas, John H. (1997) Two Data Sets of Near Infrared Spectra *Chemometrics and Intelligent Laboratory Systems*, **37**, 255–259.

`jack.test` *Jackknife approximate t tests of regression coefficients*

Description

Performes approximate t tests of regression coefficients based on jackknife variance estimates.

Usage

```
jack.test(object, ncomp = object$ncomp, use.mean = TRUE)

## S3 method for class 'jacktest'
print(x, P.values = TRUE, ...)
```

Arguments

`object` an `mvr` object. A cross-validated model fitted with `jackknife = TRUE`.

`ncomp` the number of components to use for estimating the variances

`use.mean` logical. If `TRUE` (default), the mean coefficients are used when estimating the (co)variances; otherwise the coefficients from a model fitted to the entire data set. See `var.jack` for details.

`x` an `jacktest` object, the result of `jack.test`.

`P.values` logical. Whether to print p values (default).

`...` Further arguments sent to the underlying print function `printCoefmat`.

Details

`jack.test` uses the variance estimates from `var.jack` to perform t tests of the regression coefficients. The resulting object has a print method, `print.jacktest`, which uses `printCoefmat` for the actual printing.

Value

`jack.test` returns an object of class `"jacktest"`, with components

`coefficients` The estimated regression coefficients

`sd` The square root of the jackknife variance estimates

`tvalues` The t statistics

`df` The 'degrees of freedom' used for calculating p values

`pvalues` The calculated p values

`print.jacktest` returns the `"jacktest"` object (invisibly).

Warning

The jackknife variance estimates are known to be biased (see `var.jack`). Also, the distribution of the regression coefficient estimates and the jackknife variance estimates are unknown (at least in PLSR/PCR). Consequently, the distribution (and in particular, the degrees of freedom) of the resulting t statistics is unknown. The present code simply assumes a t distribution with $m - 1$ degrees of freedom, where m is the number of cross-validation segments.

Therefore, the resulting p values should not be used uncritically, and should perhaps be regarded as mere indicator of (non-)significance.

Finally, also keep in mind that as the number of predictor variables increase, the problem of multiple tests increases correspondingly.

Author(s)

Bjørn-Helge Mevik

References

Martens H. and Martens M. (2000) Modified Jack-knife Estimation of Parameter Uncertainty in Bilinear Modelling by Partial Least Squares Regression (PLSR). *Food Quality and Preference*, **11**, 5–16.

See Also

`var.jack`, `mvrCv`

Examples

```
data(oliveoil)
mod <- pcr(sensory ~ chemical, data = oliveoil, validation = "LOO", jackknife = TRUE)
jack.test(mod, ncomp = 2)
```

`kernelpls.fit` *Kernel PLS (Dayal and MacGregor)*

Description

Fits a PLSR model with the kernel algorithm.

Usage

```
kernelpls.fit(X, Y, ncomp, center = TRUE, stripped = FALSE, ...)
```

Arguments

`X` a matrix of observations. `NA`s and `Inf`s are not allowed.

`Y` a vector or matrix of responses. `NA`s and `Inf`s are not allowed.

`ncomp` the number of components to be used in the modelling.

`center` logical, determines if the X and Y matrices are mean centered or not. Default is to perform mean centering.

`stripped` logical. If `TRUE` the calculations are stripped as much as possible for speed; this is meant for use with cross-validation or simulations when only the coefficients are needed. Defaults to `FALSE`.

`...` other arguments. Currently ignored.

Details

This function should not be called directly, but through the generic functions `plsr` or `mvr` with the argument `method="kernelpls"` (default). Kernel PLS is particularly efficient when the number of objects is (much) larger than the number of variables. The results are equal to the NIPALS algorithm. Several different forms of kernel PLS have been described in literature, e.g. by De Jong and Ter Braak, and two algorithms by Dayal and MacGregor. This function implements the fastest of the latter, not calculating the crossproduct matrix of X. In the Dyal & MacGregor paper, this is "algorithm 1".

Value

A list containing the following components is returned:

`coefficients` an array of regression coefficients for 1, ..., `ncomp` components. The dimensions of `coefficients` are `c(nvar,npred,ncomp)` with `nvar` the number of `X` variables and `npred` the number of variables to be predicted in `Y`.

`scores` a matrix of scores.

`loadings` a matrix of loadings.

`loading.weights` a matrix of loading weights.

`Yscores` a matrix of Y-scores.

`Yloadings` a matrix of Y-loadings.

`projection` the projection matrix used to convert X to scores.

`Xmeans` a vector of means of the X variables.

`Ymeans` a vector of means of the Y variables.

`fitted.values` an array of fitted values. The dimensions of `fitted.values` are `c(nobj,npred,ncomp)` with `nobj` the number samples and `npred` the number of Y variables.

`residuals` an array of regression residuals. It has the same dimensions as `fitted.values`.

`Xvar` a vector with the amount of X-variance explained by each component.

`Xtotvar` Total variance in `X`.

If `stripped` is `TRUE`, only the components `coefficients`, `Xmeans` and `Ymeans` are returned.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

de Jong, S. and ter Braak, C. J. F. (1994) Comments on the PLS kernel algorithm. *Journal of Chemometrics*, **8**, 169–174.

Dayal, B. S. and MacGregor, J. F. (1997) Improved PLS algorithms. *Journal of Chemometrics*, **11**, 73–85.

See Also

`mvr` `plsr` `cppls` `pcr` `widekernelpls.fit` `simpls.fit` `oscorespls.fit`

mayonnaise *NIR measurements and oil types of mayonnaise*

Description

Raw NIR measurements (351 wavelengths, 1100-2500 nm in steps of 4 nm) taken on 54 samples of mayonnaise based on six different oil types (soybean, sunflower, canola, olive, corn, and grapeseed). The resulting 54 samples were measured in triplicates, resulting in 54 x 3 = 162 different spectra (120/42 training/test).

Format

A data frame with 162 observations on the following 4 variables.

NIR a matrix with 351 columns

oil.type a numeric vector

design a matrix with 5 columns

train a logical vector

Source

Indahl U, Sahni NS, Kirkhus B, Næs T. Multivariate strategies for classification based on NIR-spectra-with application to mayonnaise. Chemometr. Intell. Lab. Sys. 1999; 49: 19-31.

msc *Multiplicative Scatter Correction*

Description

Performs multiplicative scatter/signal correction on a data matrix.

Usage

```
msc(X, reference = NULL)

## S3 method for class 'msc'
predict(object, newdata, ...)

## S3 method for class 'msc'
makepredictcall(var, call)
```

Arguments

`X, newdata` numeric matrices. The data to scatter correct.

`reference` numeric vector. Spectre to use as reference. If `NULL`, the column means of `X` are used.

`object` an object inheriting from class `"msc"`, normally the result of a call to `msc` with a single matrix argument.

`...` other arguments. Currently ignored.

`var` A variable.

`call` The term in the formula, as a call.

Details

`makepredictcall.msc` is an internal utility function; it is not meant for interactive use. See `makepredictcall` for details.

Value

Both `msc` and `predict.msc` return a multiplicative scatter corrected matrix, with attribute `"reference"` the vector used as reference spectre. The matrix is given class `c("msc","matrix")`. For `predict.msc`, the `"reference"` attribute of `object` is used as reference spectre.

Author(s)

Bjørn-Helge Mevik and Ron Wehrens

References

Martens, H., Næs, T. (1989) *Multivariate calibration.* Chichester: Wiley.

See Also

`mvr`, `pcr`, `plsr`, `stdize`

Examples

```
data(yarn)
## Direct correction:
Ztrain <- msc(yarn$NIR[yarn$train,])
Ztest <- predict(Ztrain, yarn$NIR[!yarn$train,])

## Used in formula:
mod <- plsr(density ~ msc(NIR), ncomp = 6, data = yarn[yarn$train,])
pred <- predict(mod, newdata = yarn[!yarn$train,]) # Automatically scatter corrected
```

mvr *Partial Least Squares and Principal Component Regression*

Description

Functions to perform partial least squares regression (PLSR), canonical powered partial least squares (CPPLS) or principal component regression (PCR), with a formula interface. Cross-validation can be used. Prediction, model extraction, plot, print and summary methods exist.

Usage

```
mvr(
  formula,
  ncomp,
  Y.add,
  data,
  subset,
  na.action,
  method = pls.options()$mvralg,
  scale = FALSE,
  center = TRUE,
  validation = c("none", "CV", "LOO"),
  model = TRUE,
  x = FALSE,
  y = FALSE,
  ...
)

plsr(..., method = pls.options()$plsralg)

pcr(..., method = pls.options()$pcralg)

cppls(..., Y.add, weights, method = pls.options()$cpplsalg)
```

Arguments

`formula` a model formula. Most of the `lm` formula constructs are supported. See below.

`ncomp` the number of components to include in the model (see below).

`Y.add` a vector or matrix of additional responses containing relevant information about the observations. Only used for `cppls`.

`data` an optional data frame with the data to fit the model from.

`subset` an optional vector specifying a subset of observations to be used in the fitting process.

`na.action` a function which indicates what should happen when the data contain missing values. The default is set by the `na.action` setting of `options`, and is `na.fail` if that is unset. The ‘factory-fresh’ default is `na.omit`. Another possible value is `NULL`, no action. Value `na.exclude` can be useful. See `na.omit` for other alternatives.

`method` the multivariate regression method to be used. If `"model.frame"`, the model frame is returned.

`scale` numeric vector, or logical. If numeric vector, X is scaled by dividing each variable with the corresponding element of `scale`. If `scale` is `TRUE`, X is scaled by dividing each variable by its sample standard deviation. If cross-validation is selected, scaling by the standard deviation is done for every segment.

`center` logical, determines if the X and Y matrices are mean centered or not. Default is to perform mean centering.

`validation` character. What kind of (internal) validation to use. See below.

`model` a logical. If `TRUE`, the model frame is returned.

`x` a logical. If `TRUE`, the model matrix is returned.

`y` a logical. If `TRUE`, the response is returned.

`...` additional optional arguments, passed to the underlying fit functions, and `mvrCv`. Currently, the fit functions `oscorespls.fit` and `widekernelpls.fit` implement these extra arguments:

tol: numeric. Tolerance used for determining convergence.

maxit: positive integer. The maximal number of iterations used.

and `cppls.fit` implements:

lower: a vector of lower limits for power optimisation.

upper: a vector of upper limits for power optimisation.

trunc.pow: logical. Whether to use an experimental alternative power algorithm.

`mvrCv` implements several arguments; the following are probably the most useful of them:

segments: the number of segments to use, or a list with segments.

segment.type: the type of segments to use.

length.seg: Positive integer. The length of the segments to use.

jackknife: logical. Whether to perform jackknifing of regression coefficients.

See the functions’ documentation for details.

`weights` a vector of individual weights for the observations. Only used for `cppls`. (Optional)

Details

The functions fit PLSR, CPPLS or PCR models with $1, \ldots,$ `ncomp` number of components. Multi-response models are fully supported.

The type of model to fit is specified with the `method` argument. Four PLSR algorithms are available: the kernel algorithm (`"kernelpls"`), the wide kernel algorithm (`"widekernelpls"`), SIMPLS (`"simpls"`) and the classical orthogonal scores algorithm (`"oscorespls"`). One CPPLS algorithm is available ("cppls") providing several extensions to PLS. One PCR algorithm is available: using the singular value decomposition ("svdpc"). If method is "model.frame", the model frame is returned. The functions pcr, plsr and cppls are wrappers for mvr, with different values for method.

The formula argument should be a symbolic formula of the form response ~ terms, where response is the name of the response vector or matrix (for multi-response models) and terms is the name of one or more predictor matrices, usually separated by +, e.g., water ~ FTIR or y ~ X + Z. See lm for a detailed description. The named variables should exist in the supplied data data frame or in the global environment. Note: Do not use mvr(mydata$y ~ mydata$X, ...{}), instead use mvr(y ~ X, data = mydata, ...{}). Otherwise, predict.mvr will not work properly. The chapter 'Statistical models in R' of the manual 'An Introduction to R' distributed with is a good reference on formulas in .

The number of components to fit is specified with the argument ncomp. It this is not supplied, the maximal number of components is used (taking account of any cross-validation).

All implemented algorithms mean-center both predictor and response matrices. This can be turned off by specifying center = FALSE. See Seasholtz and Kowalski for a discussion about centering in PLS regression.

If validation = "CV", cross-validation is performed. The number and type of cross-validation segments are specified with the arguments segments and segment.type. See mvrCv for details. If validation = "LOO", leave-one-out cross-validation is performed. It is an error to specify the segments when validation = "LOO" is specified.

By default, the cross-validation will be performed serially. However, it can be done in parallel using functionality in the parallel package by setting the option parallel in pls.options. See pls.options for the differnt ways to specify the parallelism. See also Examples below.

Note that the cross-validation is optimised for speed, and some generality has been sacrificed. Especially, the model matrix is calculated only once for the complete cross-validation, so models like y ~ msc(X) will not be properly cross-validated. However, scaling requested by scale = TRUE is properly cross-validated. For proper cross-validation of models where the model matrix must be updated/regenerated for each segment, use the separate function crossval.

Value

If method = "model.frame", the model frame is returned. Otherwise, an object of class mvr is returned. The object contains all components returned by the underlying fit function. In addition, it contains the following components:

validation if validation was requested, the results of the cross-validation. See mvrCv for details.

fit.time the elapsed time for the fit. This is used by crossval to decide whether to turn on tracing.

na.action if observations with missing values were removed, na.action contains a vector with their indices. The class of this vector is used by functions like fitted to decide how to treat the observations.

ncomp the number of components of the model.

method the method used to fit the model. See the argument method for possible values.

scale if scaling was requested (with scale), the scaling used.

`call` the function call.

`terms` the model terms.

`model` if `model = TRUE`, the model frame.

`x` if `x = TRUE`, the model matrix.

`y` if `y = TRUE`, the model response.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

Martens, H., Næs, T. (1989) *Multivariate calibration.* Chichester: Wiley.

Seasholtz, M. B. and Kowalski, B. R. (1992) The effect of mean centering on prediction in multivariate calibration. *Journal of Chemometrics*, **6**(2), 103–111.

See Also

`kernelpls.fit`, `widekernelpls.fit`, `simpls.fit`, `oscorespls.fit`, `cppls.fit`, `svdpc.fit`, `mvrCv`, `crossval`, `loadings`, `scores`, `loading.weights`, `coef.mvr`, `predict.mvr`, `R2`, `MSEP`, `RMSEP`, `plot.mvr`

Examples

```
data(yarn)
## Default methods:
yarn.pcr <- pcr(density ~ NIR, 6, data = yarn, validation = "CV")
yarn.pls <- plsr(density ~ NIR, 6, data = yarn, validation = "CV")
yarn.cppls <- cppls(density ~ NIR, 6, data = yarn, validation = "CV")

## Alternative methods:
yarn.oscorespls <- mvr(density ~ NIR, 6, data = yarn, validation = "CV",
                      method = "oscorespls")
yarn.simpls <- mvr(density ~ NIR, 6, data = yarn, validation = "CV",
                  method = "simpls")

## Not run:
## Parallelised cross-validation, using transient cluster:
pls.options(parallel = 4) # use mclapply
pls.options(parallel = quote(makeCluster(4, type = "PSOCK"))) # use parLapply
## A new cluster is created and stopped for each cross-validation:
yarn.pls <- plsr(density ~ NIR, 6, data = yarn, validation = "CV")
yarn.pcr <- pcr(density ~ NIR, 6, data = yarn, validation = "CV")

## Parallelised cross-validation, using persistent cluster:
library(parallel)
## This creates the cluster:
pls.options(parallel = makeCluster(4, type = "PSOCK"))
## The cluster can be used several times:
```

```
yarn.pls <- plsr(density ~ NIR, 6, data = yarn, validation = "CV")
yarn.pcr <- pcr(density ~ NIR, 6, data = yarn, validation = "CV")
## The cluster should be stopped manually afterwards:
stopCluster(pls.options()$parallel)

## Parallelised cross-validation, using persistent MPI cluster:
## This requires the packages snow and Rmpi to be installed
library(parallel)
## This creates the cluster:
pls.options(parallel = makeCluster(4, type = "MPI"))
## The cluster can be used several times:
yarn.pls <- plsr(density ~ NIR, 6, data = yarn, validation = "CV")
yarn.pcr <- pcr(density ~ NIR, 6, data = yarn, validation = "CV")
## The cluster should be stopped manually afterwards:
stopCluster(pls.options()$parallel)
## It is good practice to call mpi.exit() or mpi.quit() afterwards:
mpi.exit()

## End(Not run)

## Multi-response models:
data(oliveoil)
sens.pcr <- pcr(sensory ~ chemical, ncomp = 4, scale = TRUE, data = oliveoil)
sens.pls <- plsr(sensory ~ chemical, ncomp = 4, scale = TRUE, data = oliveoil)

## Classification
# A classification example utilizing additional response information
# (Y.add) is found in the cppls.fit manual ('See also' above).
```

mvrCv *Cross-validation*

Description

Performs the cross-validation calculations for mvr.

This function is not meant to be called directly, but through the generic functions pcr, plsr, cppls or mvr with the argument validation set to "CV" or "LOO". All arguments to mvrCv can be specified in the generic function call.

If segments is a list, the arguments segment.type and length.seg are ignored. The elements of the list should be integer vectors specifying the indices of the segments. See cvsegments for details.

Otherwise, segments of type segment.type are generated. How many segments to generate is selected by specifying the number of segments in segments, or giving the segment length in length.seg. If both are specified, segments is ignored.

If jackknife is TRUE, jackknifed regression coefficients are returned, which can be used for for variance estimation (var.jack) or hypothesis testing (jack.test).

X and Y do not need to be centered.

Note that this function cannot be used in situations where X needs to be recalculated for each segment (except for scaling by the standard deviation), for instance with msc or other preprocessing. For such models, use the more general (but slower) function crossval.

Also note that if needed, the function will silently(!) reduce ncomp to the maximal number of components that can be cross-validated, which is $n - l - 1$, where n is the number of observations and l is the length of the longest segment. The (possibly reduced) number of components is returned as the component ncomp.

By default, the cross-validation will be performed serially. However, it can be done in parallel using functionality in the parallel package by setting the option parallel in pls.options. See pls.options for the different ways to specify the parallelism.

Usage

```
mvrCv(
  X,
  Y,
  ncomp,
  Y.add = NULL,
  weights = NULL,
  method = pls.options()$mvralg,
  scale = FALSE,
  segments = 10,
  segment.type = c("random", "consecutive", "interleaved"),
  length.seg,
  jackknife = FALSE,
  trace = FALSE,
  ...
)
```

Arguments

X	a matrix of observations. NAs and Infs are not allowed.
Y	a vector or matrix of responses. NAs and Infs are not allowed.
ncomp	the number of components to be used in the modelling.
Y.add	a vector or matrix of additional responses containing relevant information about the observations. Only used for cppls.
weights	a vector of individual weights for the observations. Only used for cppls. (Optional)
method	the multivariate regression method to be used.
scale	logical. If TRUE, the learning X data for each segment is scaled by dividing each variable by its sample standard deviation. The prediction data is scaled by the same amount.
segments	the number of segments to use, or a list with segments (see below).
segment.type	the type of segments to use. Ignored if segments is a list.
length.seg	Positive integer. The length of the segments to use. If specified, it overrides segments unless segments is a list.

`jackknife` logical. Whether jackknifing of regression coefficients should be performed.

`trace` logical; if `TRUE`, the segment number is printed for each segment.

`...` additional arguments, sent to the underlying fit function.

Value

A list with the following components:

`method` equals `"CV"` for cross-validation.

`pred` an array with the cross-validated predictions.

`coefficients` (only if `jackknife` is `TRUE`) an array with the jackknifed regression coefficients. The dimensions correspond to the predictors, responses, number of components, and segments, respectively.

`PRESS0` a vector of PRESS values (one for each response variable) for a model with zero components, i.e., only the intercept.

`PRESS` a matrix of PRESS values for models with 1, . . . , `ncomp` components. Each row corresponds to one response variable.

`adj` a matrix of adjustment values for calculating bias corrected MSEP. `MSEP` uses this.

`segments` the list of segments used in the cross-validation.

`ncomp` the actual number of components used.

`gamma` if method `cppls` is used, gamma values for the powers of each CV segment are returned.

Note

The `PRESS0` is always cross-validated using leave-one-out cross-validation. This usually makes little difference in practice, but should be fixed for correctness.

The current implementation of the jackknife stores all jackknife-replicates of the regression coefficients, which can be very costly for large matrices. This might change in a future version.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

Mevik, B.-H., Cederkvist, H. R. (2004) Mean Squared Error of Prediction (MSEP) Estimates for Principal Component Regression (PCR) and Partial Least Squares Regression (PLSR). *Journal of Chemometrics*, **18**(9), 422–429.

See Also

`mvr` `crossval` `cvsegments` `MSEP` `var.jack` `jack.test`

Examples

```
data(yarn)
yarn.pcr <- pcr(density ~ NIR, 6, data = yarn, validation = "CV", segments = 10)
## Not run: plot(MSEP(yarn.pcr))
```

---

mvrVal *MSEP, RMSEP and R2 of PLSR and PCR models*

Description

Functions to estimate the mean squared error of prediction (MSEP), root mean squared error of prediction (RMSEP) and R^2 (A.K.A. coefficient of multiple determination) for fitted PCR and PLSR models. Test-set, cross-validation and calibration-set estimates are implemented.

Usage

```
mvrValstats(
  object,
  estimate,
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  intercept = cumulative,
  se = FALSE,
  ...
)

R2(object, ...)

## S3 method for class 'mvr'
R2(
  object,
  estimate,
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  intercept = cumulative,
  se = FALSE,
  ...
)

MSEP(object, ...)

## S3 method for class 'mvr'
MSEP(
  object,
  estimate,
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  intercept = cumulative,
  se = FALSE,
  ...
)

RMSEP(object, ...)

## S3 method for class 'mvr'
RMSEP(object, ...)
```

Arguments

`object` an `mvr` object

`estimate` a character vector. Which estimators to use. Should be a subset of `c("all","train","CV","adjCV","te` `"adjCV"` is only available for (R)MSEP. See below for how the estimators are chosen.

`newdata` a data frame with test set data.

`ncomp, comps` a vector of positive integers. The components or number of components to use. See below.

`intercept` logical. Whether estimates for a model with zero components should be returned as well.

`se` logical. Whether estimated standard errors of the estimates should be calculated. Not implemented yet.

`...` further arguments sent to underlying functions or (for `RMSEP`) to `MSEP`

Details

`RMSEP` simply calls `MSEP` and takes the square root of the estimates. It therefore accepts the same arguments as `MSEP`.

Several estimators can be used. `"train"` is the training or calibration data estimate, also called (R)MSEC. For `R2`, this is the unadjusted R^2. It is overoptimistic and should not be used for assessing models. `"CV"` is the cross-validation estimate, and `"adjCV"` (for `RMSEP` and `MSEP`) is the bias-corrected cross-validation estimate. They can only be calculated if the model has been cross-validated. Finally, `"test"` is the test set estimate, using `newdata` as test set.

Which estimators to use is decided as follows (see below for `mvrValstats`). If `estimate` is not specified, the test set estimate is returned if `newdata` is specified, otherwise the CV and adjusted CV (for `RMSEP` and `MSEP`) estimates if the model has been cross-validated, otherwise the training data estimate. If `estimate` is `"all"`, all possible estimates are calculated. Otherwise, the specified estimates are calculated.

Several model sizes can also be specified. If `comps` is missing (or is `NULL`), `length(ncomp)` models are used, with `ncomp[1]` components, ..., `ncomp[length(ncomp)]` components. Otherwise, a single model with the components comps[1], ..., comps[length(comps)] is used. If intercept is TRUE, a model with zero components is also used (in addition to the above).

The R^2 values returned by "R2" are calculated as $1 - SSE/SST$, where SST is the (corrected) total sum of squares of the response, and SSE is the sum of squared errors for either the fitted values (i.e., the residual sum of squares), test set predictions or cross-validated predictions (i.e., the $PRESS$). For estimate = "train", this is equivalent to the squared correlation between the fitted values and the response. For estimate = "train", the estimate is often called the prediction R^2.

mvrValstats is a utility function that calculates the statistics needed by MSEP and R2. It is not intended to be used interactively. It accepts the same arguments as MSEP and R2. However, the estimate argument must be specified explicitly: no partial matching and no automatic choice is made. The function simply calculates the types of estimates it knows, and leaves the other untouched.

Value

mvrValstats returns a list with components

SSE three-dimensional array of SSE values. The first dimension is the different estimators, the second is the response variables and the third is the models.

SST matrix of SST values. The first dimension is the different estimators and the second is the response variables.

nobj a numeric vector giving the number of objects used for each estimator.

comps the components specified, with 0 prepended if intercept is TRUE.

cumulative TRUE if comps was NULL or not specified.

The other functions return an object of class "mvrVal", with components

val three-dimensional array of estimates. The first dimension is the different estimators, the second is the response variables and the third is the models.

type "MSEP", "RMSEP" or "R2".

comps the components specified, with 0 prepended if intercept is TRUE.

cumulative TRUE if comps was NULL or not specified.

call the function call

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

Mevik, B.-H., Cederkvist, H. R. (2004) Mean Squared Error of Prediction (MSEP) Estimates for Principal Component Regression (PCR) and Partial Least Squares Regression (PLSR). *Journal of Chemometrics*, **18**(9), 422–429.

See Also

mvr, crossval, mvrCv, validationplot, plot.mvrVal

Examples

```
data(oliveoil)
mod <- plsr(sensory ~ chemical, ncomp = 4, data = oliveoil, validation = "LOO")
RMSEP(mod)
## Not run: plot(R2(mod))
```

---

oliveoil *Sensory and physico-chemical data of olive oils*

Description

A data set with scores on 6 attributes from a sensory panel and measurements of 5 physico-chemical quality parameters on 16 olive oil samples. The first five oils are Greek, the next five are Italian and the last six are Spanish.

Format

A data frame with 16 observations on the following 2 variables.

sensory a matrix with 6 columns. Scores for attributes ‘yellow’, ‘green’, ‘brown’, ‘glossy’, ‘transp’, and ‘syrup’.

chemical a matrix with 5 columns. Measurements of acidity, peroxide, K232, K270, and DK.

Source

Massart, D. L., Vandeginste, B. G. M., Buydens, L. M. C., de Jong, S., Lewi, P. J., Smeyers-Verbeke, J. (1998) *Handbook of Chemometrics and Qualimetrics: Part B*. Elsevier. Tables 35.1 and 35.4.

---

oscorespls.fit *Orthogonal scores PLSR*

Description

Fits a PLSR model with the orthogonal scores algorithm (aka the NIPALS algorithm).

Usage

```
oscorespls.fit(
  X,
  Y,
  ncomp,
  center = TRUE,
  stripped = FALSE,
  tol = .Machine$double.eps^0.5,
  maxit = 100,
  ...
)
```

Arguments

`X` a matrix of observations. `NA`s and `Inf`s are not allowed.

`Y` a vector or matrix of responses. `NA`s and `Inf`s are not allowed.

`ncomp` the number of components to be used in the modelling.

`center` logical, determines if the X and Y matrices are mean centered or not. Default is to perform mean centering.

`stripped` logical. If `TRUE` the calculations are stripped as much as possible for speed; this is meant for use with cross-validation or simulations when only the coefficients are needed. Defaults to `FALSE`.

`tol` numeric. The tolerance used for determining convergence in multi-response models.

`maxit` positive integer. The maximal number of iterations used in the internal Eigenvector calculation.

`...` other arguments. Currently ignored.

Details

This function should not be called directly, but through the generic functions `plsr` or `mvr` with the argument `method="oscorespls"`. It implements the orthogonal scores algorithm, as described in *Martens and Næs (1989)*. This is one of the two “classical” PLSR algorithms, the other being the orthogonal loadings algorithm.

Value

A list containing the following components is returned:

`coefficients` an array of regression coefficients for 1, ..., `ncomp` components. The dimensions of `coefficients` are `c(nvar,npred,ncomp)` with `nvar` the number of `X` variables and `npred` the number of variables to be predicted in `Y`.

`scores` a matrix of scores.

`loadings` a matrix of loadings.

`loading.weights` a matrix of loading weights.

`Yscores` a matrix of Y-scores.

`Yloadings` a matrix of Y-loadings.

`projection` the projection matrix used to convert X to scores.

`Xmeans` a vector of means of the X variables.

`Ymeans` a vector of means of the Y variables.

`fitted.values` an array of fitted values. The dimensions of `fitted.values` are `c(nobj,npred,ncomp)` with `nobj` the number samples and `npred` the number of Y variables.

`residuals` an array of regression residuals. It has the same dimensions as `fitted.values`.

`Xvar` a vector with the amount of X-variance explained by each component.

`Xtotvar` Total variance in `X`.

If `stripped` is `TRUE`, only the components `coefficients`, `Xmeans` and `Ymeans` are returned.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

Martens, H., Næs, T. (1989) *Multivariate calibration.* Chichester: Wiley.

See Also

mvr plsr pcr kernelpls.fit widekernelpls.fit simpls.fit

---

plot.mvr *Plot Method for MVR objects*

Description

plot.mvr plots predictions, coefficients, scores, loadings, biplots, correlation loadings or validation plots (RMSEP curves, etc.).

Usage

```
## S3 method for class 'mvr'
plot(
  x,
  plottype = c("prediction", "validation", "coefficients", "scores", "loadings",
    "biplot", "correlation"),
  ...
)
```

Arguments

x	an object of class mvr. The fitted model to plot.
plottype	character. What kind of plot to plot.
...	further arguments, sent to the underlying plot functions.

Details

The function is simply a wrapper for the underlying plot functions used to make the selected plots. See predplot.mvr, validationplot, coefplot, scoreplot, loadingplot, biplot.mvr or corrplot for details. Note that all arguments except x and plottype must be named.

Value

plot.mvr returns whatever the underlying plot function returns.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

mvr, predplot.mvr, validationplot, coefplot, scoreplot, loadingplot, biplot.mvr, corrplot

Examples

```
data(yarn)
nir.pcr <- pcr(density ~ NIR, ncomp = 9, data = yarn, validation = "CV")
## Not run:
plot(nir.pcr, ncomp = 5) # Plot of cross-validated predictions
plot(nir.pcr, "scores") # Score plot
plot(nir.pcr, "loadings", comps = 1:3) # The three first loadings
plot(nir.pcr, "coef", ncomp = 5) # Coefficients
plot(nir.pcr, "val") # RMSEP curves
plot(nir.pcr, "val", val.type = "MSEP", estimate = "CV") # CV MSEP

## End(Not run)
```

pls *Partial Least Squares and Principal Component Regression*

Description

Multivariate regression methods Partial Least Squares Regression (PLSR), Principal Component Regression (PCR) and Canonical Powered Partial Least Squares (CPPLS).

pls.options *Set or return options for the pls package*

Description

A function to set options for the **pls** package, or to return the current options.

Usage

```
pls.options(...)
```

Arguments

`...` a single list, a single character vector, or any number of named arguments (*name* = *value*).

Details

If called with no arguments, or with an empty list as the single argument, `pls.options` returns the current options.

If called with a character vector as the single argument, a list with the arguments named in the vector are returned.

If called with a non-empty list as the single argument, the list elements should be named, and are treated as named arguments to the function.

Otherwise, `pls.options` should be called with one or more named arguments *name = value*. For each argument, the option named *name* will be given the value *value*.

The recognised options are:

mvralg The fit method to use in `mvr` and `mvrCv`. The value should be one of the allowed methods. Defaults to `"kernelpls"`. Can be overridden with the argument `method` in `mvr` and `mvrCv`.

pcralg The fit method to use in `pcr`. The value should be one of the allowed methods. Defaults to `"svdpc"`. Can be overridden with the argument `method` in `pcr`.

plsralg The fit method to use in `plsr`. The value should be one of the allowed methods. Defaults to `"kernelpls"`. Can be overridden with the argument `method` in `plsr`.

cpplsalg The fit method to use in `cppls`. The value should be one of the allowed methods. Defaults to `"cppls"`. Can be overridden with the argument `method` in `cppls`.

parallel Specification of how the cross-validation (CV) in `mvr` should be performed. If the specification is `NULL` (default) or `1`, the CV is done serially, otherwise it is done in parallel using functionality from the `parallel` package.

If it is an integer greater than 1, the CV is done in parallel with the specified number of processes, using `mclapply`.

If it is a cluster object created by `makeCluster`, the CV is done in parallel on that cluster, using `parLapply`. The user should stop the cluster herself when it is no longer needed, using `stopCluster`.

Finally, if the specification is an unevaluated call to `makeCluster`, the call is evaluated, and the CV is done in parallel on the resulting cluster, using `parLapply`. In this case, the cluster will be stopped (with `stopCluster`) after the CV. Thus, in the final case, the cluster is created and destroyed for each CV, just like when using `mclapply`.

w.tol The tolerance used for removing values close to 0 in the vectors of loading weights in `cppls`. Defaults to .Machine$double.eps.

X.tol The tolerance used for removing predictor variables with L1 norms close to 0 in `cppls`. Defaults to 10^-12.

Value

A list with the (possibly changed) options. If any named argument (or list element) was provided, the list is returned invisibly.

Note

The function is a slight modification of the function `sm.options` from the package **sm**.

Author(s)

Bjørn-Helge Mevik and Ron Wehrens

Examples

```
## Return current options:
pls.options()
pls.options("plsralg")
pls.options(c("plsralg", "pcralg"))

## Set options:
pls.options(plsralg = "simpls", mvralg = "simpls")
pls.options(list(plsralg = "simpls", mvralg = "simpls")) # Equivalent
pls.options()

## Restore `factory settings':
pls.options(list(mvralg = "kernelpls", plsralg = "kernelpls", cpplsalg = "cppls",
                 pcralg = "svdpc", parallel = NULL,
                 w.tol = .Machine$double.eps, X.tol = 10^-12))
pls.options()
```

predict.mvr *Predict Method for PLSR and PCR*

Description

Prediction for mvr (PCR, PLSR) models. New responses or scores are predicted using a fitted model and a new matrix of observations.

Usage

```
## S3 method for class 'mvr'
predict(
  object,
  newdata,
  ncomp = 1:object$ncomp,
  comps,
  type = c("response", "scores"),
  na.action = na.pass,
  ...
)
```

Arguments

`object`	an `mvr` object. The fitted model
`newdata`	a data frame. The new data. If missing, the training data is used.

`ncomp, comps` vector of positive integers. The components to use in the prediction. See below.

`type` character. Whether to predict scores or response values

`na.action` function determining what should be done with missing values in `newdata`. The default is to predict `NA`. See `na.omit` for alternatives.

`...` further arguments. Currently not used

Details

When `type` is `"response"` (default), predicted response values are returned. If `comps` is missing (or is `NULL`), predictions for `length(ncomp)` models with `ncomp[1]` components, `ncomp[2]` components, etc., are returned. Otherwise, predictions for a single model with the exact components in `comps` are returned. (Note that in both cases, the intercept is always included in the predictions. It can be removed by subtracting the `Ymeans` component of the fitted model.)

When `type` is `"scores"`, predicted score values are returned for the components given in `comps`. If `comps` is missing or `NULL`, `ncomps` is used instead.

It is also possible to supply a matrix instead of a data frame as `newdata`, which is then assumed to be the X data matrix. Note that the usual checks for the type of the data are then omitted. Also note that this is *only* possible with `predict`; it will not work in functions like `predplot`, `RMSEP` or `R2`, because they also need the response variable of the new data.

Value

When `type` is `"response"`, a three dimensional array of predicted response values is returned. The dimensions correspond to the observations, the response variables and the model sizes, respectively.

When `type` is `"scores"`, a score matrix is returned.

Note

A warning message like ‘`'newdata' had 10 rows but variable(s) found have 106 rows`’ means that not all variables were found in the `newdata` data frame. This (usually) happens if the formula contains terms like `yarn$NIR`. Do not use such terms; use the `data` argument instead. See `mvr` for details.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

`mvr`, `summary.mvr`, `coef.mvr`, `plot.mvr`

Examples

```
data(yarn)
nir.mvr <- mvr(density ~ NIR, ncomp = 5, data = yarn[yarn$train,])

## Predicted responses for models with 1, 2, 3 and 4 components
```

```
pred.resp <- predict(nir.mvr, ncomp = 1:4, newdata = yarn[!yarn$train,])

## Predicted responses for a single model with components 1, 2, 3, 4
predict(nir.mvr, comps = 1:4, newdata = yarn[!yarn$train,])

## Predicted scores
predict(nir.mvr, comps = 1:3, type = "scores", newdata = yarn[!yarn$train,])
```

predplot *Prediction Plots*

Description

Functions to plot predicted values against measured values for a fitted model.

Usage

```
predplot(object, ...)

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

## S3 method for class 'mvr'
predplot(
  object,
  ncomp = object$ncomp,
  which,
  newdata,
  nCols,
  nRows,
  xlab = "measured",
  ylab = "predicted",
  main,
  ask = nRows * nCols < nPlots && dev.interactive(),
  ...,
  font.main,
  cex.main
)

predplotXy(
  x,
  y,
  line = FALSE,
  labels,
  type = "p",
  main = "Prediction plot",
  xlab = "measured response",
  ylab = "predicted response",
  line.col = par("col"),
  line.lty = NULL,
  line.lwd = NULL,
  ...
)
```

Arguments

`object` a fitted model.

`...` further arguments sent to underlying plot functions.

`ncomp` integer vector. The model sizes (numbers of components) to use for prediction.

`which` character vector. Which types of predictions to plot. Should be a subset of `c("train","validation","test")`. If not specified, `plot.mvr` selects test set predictions if `newdata` is supplied, otherwise cross-validated predictions if the model has been cross-validated, otherwise fitted values from the calibration data.

`newdata` data frame. New data to predict.

`nCols, nRows` integer. The number of coloumns and rows the plots will be laid out in. If not specified, `plot.mvr` tries to be intelligent.

`xlab, ylab` titles for x and y axes. Typically character strings, but can be expressions or lists. See `title` for details.

`main` optional main title for the plot. See Details.

`ask` logical. Whether to ask the user before each page of a plot.

`font.main` font to use for main titles. See `par` for details. Also see Details below.

`cex.main` numeric. The magnification to be used for main titles relative to the current size. Also see Details below.

`x` numeric vector. The observed response values.

`y` numeric vector. The predicted response values.

`line` logical. Whether a target line should be drawn.

`labels` optional. Alternative plot labels to use. Either a vector of labels, or `"names"` or `"numbers"` to use the row names or row numbers of the data as labels.

`type` character. What type of plot to make. Defaults to `"p"` (points). See `plot` for a complete list of types. The argument is ignored if `labels` is specified.

`line.col, line.lty, line.lwd`
character or numeric. The `col`, `lty` and `lwd` parametres for the target line. See `par` for details.

Details

`predplot` is a generic function for plotting predicted versus measured response values, with default and `mvr` methods currently implemented. The default method is very simple, and doesn't handle multiple responses or new data.

The `mvr` method, handles multiple responses, model sizes and types of predictions by making one plot for each combination. It can also be called through the plot method for `mvr`, by specifying `plottype = "prediction"` (the default).

The argument main can be used to specify the main title of the plot. It is handled in a non-standard way. If there is only on (sub) plot, main will be used as the main title of the plot. If there is *more* than one (sub) plot, however, the presence of main will produce a corresponding 'global' title on the page. Any graphical parametres, e.g., cex.main, supplied to coefplot will only affect the 'ordinary' plot titles, not the 'global' one. Its appearance can be changed by setting the parameters with par, which will affect *both* titles (with the exception of font.main and cex.main, which will only affect the 'global' title when there is more than one plot). (To have different settings for the two titles, one can override the par settings with arguments to predplot.)

predplotXy is an internal function and is not meant for interactive use. It is called by the predplot methods, and its arguments, e.g, line, can be given in the predplot call.

Value

The functions invisibly return a matrix with the (last) plotted data.

Note

The font.main and cex.main must be (completely) named. This is to avoid that any argument cex or font matches them.

Tip: If the labels specified with labels are too long, they get clipped at the border of the plot region. This can be avoided by supplying the graphical parameter xpd = TRUE in the plot call.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

mvr, plot.mvr

Examples

```
data(yarn)
mod <- plsr(density ~ NIR, ncomp = 10, data = yarn[yarn$train,], validation = "CV")
## Not run:
predplot(mod, ncomp = 1:6)
plot(mod, ncomp = 1:6) # Equivalent to the previous
## Both cross-validated and test set predictions:
predplot(mod, ncomp = 4:6, which = c("validation", "test"),
         newdata = yarn[!yarn$train,])

## End(Not run)

data(oliveoil)
mod.sens <- plsr(sensory ~ chemical, ncomp = 4, data = oliveoil)
## Not run: plot(mod.sens, ncomp = 2:4) # Several responses gives several plots
```

print.mvr *Summary and Print Methods for PLSR and PCR objects*

Description

Summary and print methods for mvr and mvrVal objects.

Usage

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

## S3 method for class 'mvr'
summary(
  object,
  what = c("all", "validation", "training"),
  digits = 4,
  print.gap = 2,
  ...
)

## S3 method for class 'mvrVal'
print(x, digits = 4, print.gap = 2, ...)
```

Arguments

x, object	an mvr object
...	Other arguments sent to underlying methods.
what	one of "all", "validation" or "training"
digits	integer. Minimum number of significant digits in the output. Default is 4.
print.gap	Integer. Gap between coloumns of the printed tables.

Details

If what is "training", the explained variances are given; if it is "validation", the cross-validated RMSEPs (if available) are given; if it is "all", both are given.

Value

print.mvr and print.mvrVal return the object invisibly.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

mvr, pcr, plsr, RMSEP, MSEP

Examples

```
data(yarn)
nir.mvr <- mvr(density ~ NIR, ncomp = 8, validation = "LOO", data = yarn)
nir.mvr
summary(nir.mvr)
RMSEP(nir.mvr)
```

scoreplot *Plots of Scores, Loadings and Correlation Loadings*

Description

Functions to make scatter plots of scores or correlation loadings, and scatter or line plots of loadings.

Usage

```
scoreplot(object, ...)

## Default S3 method:
scoreplot(
  object,
  comps = 1:2,
  labels,
  identify = FALSE,
  type = "p",
  xlab,
  ylab,
  ...
)

## S3 method for class 'scores'
plot(x, ...)

loadingplot(object, ...)

## Default S3 method:
loadingplot(
  object,
  comps = 1:2,
  scatter = FALSE,
  labels,
  identify = FALSE,
  type,
  lty,
  lwd = NULL,
  pch,
  cex = NULL,
  col,
  legendpos,
  xlab,
  ylab,
  pretty.xlabels = TRUE,
  xlim,
  ...
)

## S3 method for class 'loadings'
plot(x, ...)

corrplot(
  object,
  comps = 1:2,
  labels,
  plotx = TRUE,
  ploty = FALSE,
  radii = c(sqrt(1/2), 1),
  identify = FALSE,
  type = "p",
  xlab,
  ylab,
  col,
  ...
)
```

Arguments

`object` an object. The fitted model.

`...` further arguments sent to the underlying plot function(s).

`comps` integer vector. The components to plot.

`labels` optional. Alternative plot labels or x axis labels. See Details.

`identify` logical. Whether to use `identify` to interactively identify points. See below.

`type` character. What type of plot to make. Defaults to `"p"` (points) for scatter plots and `"l"` (lines) for line plots. See `plot` for a complete list of types (not all types are possible/meaningful for all plots).

`xlab, ylab` titles for x and y axes. Typically character strings, but can be expressions or lists. See `title` for details.

`x` a `scores` or `loadings` object. The scores or loadings to plot.

scatter logical. Whether the loadings should be plotted as a scatter instead of as lines.

lty vector of line types (recycled as neccessary). Line types can be specified as integers or character strings (see par for the details).

lwd vector of positive numbers (recycled as neccessary), giving the width of the lines.

pch plot character. A character string or a vector of single characters or integers (recycled as neccessary). See points for all alternatives.

cex numeric vector of character expansion sizes (recycled as neccessary) for the plotted symbols.

col character or integer vector of colors for plotted lines and symbols (recycled as neccessary). See par for the details.

legendpos Legend position. Optional. Ignored if scatter is TRUE. If present, a legend is drawn at the given position. The position can be specified symbolically (e.g., legendpos = "topright"). This requires >= 2.1.0. Alternatively, the position can be specified explicitly (legendpos = t(c(x,y))) or interactively (legendpos = locator()).

pretty.xlabels logical. If TRUE, loadingplot tries to plot the x labels more nicely. See Details.

xlim optional vector of length two, with the x limits of the plot.

plotx locical. Whether to plot the X correlation loadings. Defaults to TRUE.

ploty locical. Whether to plot the Y correlation loadings. Defaults to FALSE.

radii numeric vector, giving the radii of the circles drawn in corrplot. The default radii represent 50% and 100% explained variance of the X variables by the chosen components.

Details

plot.scores is simply a wrapper calling scoreplot, passing all arguments. Similarly for plot.loadings.

scoreplot is generic, currently with a default method that works for matrices and any object for which scores returns a matrix. The default scoreplot method makes one or more scatter plots of the scores, depending on how many components are selected. If one or two components are selected, and identify is TRUE, the function identify is used to interactively identify points.

Also loadingplot is generic, with a default method that works for matrices and any object where loadings returns a matrix. If scatter is TRUE, the default method works exactly like the default scoreplot method. Otherwise, it makes a lineplot of the selected loading vectors, and if identify is TRUE, uses identify to interactively identify points. Also, if legendpos is given, a legend is drawn at the position indicated.

corrplot works exactly like the default scoreplot method, except that at least two components must be selected. The “correlation loadings”, i.e. the correlations between each variable and the selected components (see References), are plotted as pairwise scatter plots, with concentric circles of radii given by radii. Each point corresponds to a variable. The squared distance between the point and origin equals the fraction of the variance of the variable explained by the components in the panel. The default radii corresponds to 50% and 100% explained variance. By default, only the correlation loadings of the X variables are plotted, but if ploty is TRUE, also the Y correlation loadings are plotted.

scoreplot, loadingplot and corrplot can also be called through the plot method for mvr objects, by specifying plottype as "scores", "loadings" or "correlation", respectively. See plot.mvr.

The argument labels can be a vector of labels or one of "names" and "numbers".

If a scatter plot is produced (i.e., scoreplot, corrplot, or loadingplot with scatter = TRUE), the labels are used instead of plot symbols for the points plotted. If labels is "names" or "numbers", the row names or row numbers of the matrix (scores, loadings or correlation loadings) are used.

If a line plot is produced (i.e., loadingplot), the labels are used as x axis labels. If labels is "names" or "numbers", the variable names are used as labels, the difference being that with "numbers", the variable names are converted to numbers, if possible. Variable names of the forms '"number"' or '"number text"' (where the space is optional), are handled.

The argument pretty.xlabels is only used when labels is specified for a line plot. If TRUE (default), the code tries to use a 'pretty' selection of labels. If labels is "numbers", it also uses the numerical values of the labels for horisontal spacing. If one has excluded parts of the spectral region, one might therefore want to use pretty.xlabels = FALSE.

Value

The functions return whatever the underlying plot function (or identify) returns.

Note

legend has many options. If you want greater control over the appearance of the legend, omit the legendpos argument and call legend manually.

Graphical parametres (such as pch and cex) can also be used with scoreplot and corrplot. They are not listed in the argument list simply because they are not handled specifically in the function (unlike in loadingplot), but passed directly to the underlying plot functions by ...{}.

Tip: If the labels specified with labels are too long, they get clipped at the border of the plot region. This can be avoided by supplying the graphical parameter xpd = TRUE in the plot call.

The handling of labels and pretty.xlabels in coefplot is experimental.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

Martens, H., Martens, M. (2000) Modified Jack-knife Estimation of Parameter Uncertainty in Bilinear Modelling by Partial Least Squares Regression (PLSR). *Food Quality and Preference*, **11**(1–2), 5–16.

See Also

mvr, plot.mvr, scores, loadings, identify, legend

Examples

```
data(yarn)
mod <- plsr(density ~ NIR, ncomp = 10, data = yarn)
## These three are equivalent:
## Not run:
scoreplot(mod, comps = 1:5)
plot(scores(mod), comps = 1:5)
plot(mod, plottype = "scores", comps = 1:5)

loadingplot(mod, comps = 1:5)
loadingplot(mod, comps = 1:5, legendpos = "topright") # With legend
loadingplot(mod, comps = 1:5, scatter = TRUE) # Plot as scatterplots

corrplot(mod, comps = 1:2)
corrplot(mod, comps = 1:3)

## End(Not run)
```

---

scores *Extract Scores and Loadings from PLSR and PCR Models*

Description

These functions extract score and loading matrices from fitted `mvr` models.

Usage

```
loadings(object, ...)

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

scores(object, ...)

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

Yscores(object)

loading.weights(object)

Yloadings(object)
```

Arguments

`object`	a fitted model to extract from.
`...`	extra arguments, currently not used.

Details

All functions extract the indicated matrix from the fitted model, and will work with any object having a suitably named component.

The default scores and loadings methods also handle prcomp objects (their scores and loadings components are called x and rotation, resp.), and add an attribute "explvar" with the variance explained by each component, if this is available. (See explvar for details.)

Value

A matrix with scores or loadings.

Note

There is a loadings function in package **stats**. It simply returns any element named "loadings". See loadings for details. The function can be accessed as stats::loadings(...).

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

mvr, coef.mvr

Examples

```
data(yarn)
plsmod <- plsr(density ~ NIR, 6, data = yarn)
scores(plsmod)
loadings(plsmod)[,1:4]
```

selectNcomp *Suggestions for the optimal number of components in PCR and PLSR models*

Description

Choosing the best number of components in PCR and PLSR models is difficult and usually done on the basis of visual inspection of the validation plots. In cases where large numbers of models are built this choice needs to be automated. This function implements two proposals, one based on randomization (permutation) testing, and an approach based on the standard error of the cross-validation residuals.

Usage

```
selectNcomp(
  object,
  method = c("randomization", "onesigma"),
  nperm = 999,
  alpha = 0.01,
  ncomp = object$ncomp,
  plot = FALSE,
  ...
)
```

Arguments

`object` an `mvr` object. The fitted model. It should contain a `validation` element.

`method` character string, indicating the heuristic to use.

`nperm` number of permutations in the `"randomization"` approach - not used in the `"onesigma"` approach.

`alpha` cutoff for p values in the `"randomization"` approach - not used in the `"onesigma"` approach.

`ncomp` maximum number of components to consider when determining the global minimum in the cross-validation curve.

`plot` whether or not to show a cross-validation plot. The plot for the `"randomization"` approach shows models that do not differ significantly from the global RMSEP minimum with open circles; the `"onesigma"` approach shows the one-sigma bands around the RMSEP values. In both cases, the selection is indicated with a blue dashed line.

`...` Further plotting arguments, e.g., to add a title to the plot, or to limit the plotting range.

Details

In both approaches the results of cross-validation are used, so the model should have been calculated with some form of cross-validation. First, the absolute minimum in the CV curve is determined (considering only the first ncomp components), leading to the reference model. The randomization test approach (Van der Voet, 1994) checks whether the squared prediction errors of models with fewer components are significantly larger than in the reference model. This leads for each model considered to a p value; the smallest model not significantly worse than the reference model is returned as the selected one.

The approach `"onesigma"` simply returns the first model where the optimal CV is within one standard error of the absolute optimum (Hastie, Tibshirani and Friedman, 2009). Note that here we simply use the standard deviation of the cross-validation residuals, in line with the procedure used to calculate the error measure itself. Some other packages implementing similar procedures (such as `glmnet`) calculate an error measure for each validation segment separately and use the average as the final estimate. In such cases the standard error across segments is the relevant measure of spread. For LOO, the two procedures are identical. In other forms of validation, small differences will occur.

Value

A number indicating the suggested number of components in the model.

Author(s)

Ron Wehrens, Hilko van der Voet and Gerie van der Heijden

References

Van der Voet, H. (1994) Comparing the predictive accuracy of models using a simple randomization test. Chemom. Intell. Lab. Syst. 25 (2), 313-323

Hastie, T., Friedman, J. and Tibshirani, R. The Elements of Statistical Learning: data mining, inference, and prediction, Springer (2013), 10th printing with corrections, paragraph 7.10.

See Also

`mvr`, `summary.mvr`

Examples

```
data(yarn)
yarn.pls <- plsr(density ~ NIR, data = yarn, scale = TRUE,
                 ncomp = 20, validation = "LOO")
selectNcomp(yarn.pls, "onesigma", plot = TRUE, ylim = c(0, 3))
selectNcomp(yarn.pls, "randomization", plot = TRUE)
selectNcomp(yarn.pls, "randomization", plot = TRUE,
            ncomp = 10, ylim = c(0, 3))
```

`simpls.fit` *Sijmen de Jong's SIMPLS*

Description

Fits a PLSR model with the SIMPLS algorithm.

Usage

```
simpls.fit(X, Y, ncomp, center = TRUE, stripped = FALSE, ...)
```

Arguments

`X`	a matrix of observations. `NA`s and `Inf`s are not allowed.
`Y`	a vector or matrix of responses. `NA`s and `Inf`s are not allowed.
`ncomp`	the number of components to be used in the modelling.

`center` logical, determines if the X and Y matrices are mean centered or not. Default is to perform mean centering.

`stripped` logical. If `TRUE` the calculations are stripped as much as possible for speed; this is meant for use with cross-validation or simulations when only the coefficients are needed. Defaults to `FALSE`.

`...` other arguments. Currently ignored.

Details

This function should not be called directly, but through the generic functions `plsr` or `mvr` with the argument `method="simpls"`. SIMPLS is much faster than the NIPALS algorithm, especially when the number of X variables increases, but gives slightly different results in the case of multivariate Y. SIMPLS truly maximises the covariance criterion. According to de Jong, the standard PLS2 algorithms lie closer to ordinary least-squares regression where a precise fit is sought; SIMPLS lies closer to PCR with stable predictions.

Value

A list containing the following components is returned:

`coefficients` an array of regression coefficients for 1, ..., `ncomp` components. The dimensions of `coefficients` are `c(nvar,npred,ncomp)` with `nvar` the number of `X` variables and `npred` the number of variables to be predicted in `Y`.

`scores` a matrix of scores.

`loadings` a matrix of loadings.

`Yscores` a matrix of Y-scores.

`Yloadings` a matrix of Y-loadings.

`projection` the projection matrix used to convert X to scores.

`Xmeans` a vector of means of the X variables.

`Ymeans` a vector of means of the Y variables.

`fitted.values` an array of fitted values. The dimensions of `fitted.values` are `c(nobj,npred,ncomp)` with `nobj` the number samples and `npred` the number of Y variables.

`residuals` an array of regression residuals. It has the same dimensions as `fitted.values`.

`Xvar` a vector with the amount of X-variance explained by each component.

`Xtotvar` Total variance in `X`.

If `stripped` is `TRUE`, only the components `coefficients`, `Xmeans` and `Ymeans` are returned.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

de Jong, S. (1993) SIMPLS: an alternative approach to partial least squares regression. *Chemometrics and Intelligent Laboratory Systems*, **18**, 251–263.

See Also

mvr plsr pcr kernelpls.fit widekernelpls.fit oscorespls.fit

---

stdize *Standardization of Data Matrices*

Description

Performs standardization (centering and scaling) of a data matrix.

Usage

```
stdize(x, center = TRUE, scale = TRUE)

## S3 method for class 'stdized'
predict(object, newdata, ...)

## S3 method for class 'stdized'
makepredictcall(var, call)
```

Arguments

`x, newdata`	numeric matrices. The data to standardize.
`center`	logical value or numeric vector of length equal to the number of coloumns of `x`.
`scale`	logical value or numeric vector of length equal to the number of coloumns of `x`.
`object`	an object inheriting from class `"stdized"`, normally the result of a call to `stdize`.
`...`	other arguments. Currently ignored.
`var`	A variable.
`call`	The term in the formula, as a call.

Details

`makepredictcall.stdized` is an internal utility function; it is not meant for interactive use. See `makepredictcall` for details.

If `center` is `TRUE`, `x` is centered by subtracting the coloumn mean from each coloumn. If `center` is a numeric vector, it is used in place of the coloumn means.

If `scale` is `TRUE`, `x` is scaled by dividing each coloumn by its sample standard deviation. If `scale` is a numeric vector, it is used in place of the standard deviations.

Value

Both `stdize` and `predict.stdized` return a scaled and/or centered matrix, with attributes `"stdized:center"` and/or `"stdized:scale"` the vector used for centering and/or scaling. The matrix is given class `c("stdized","matrix")`.

Note

stdize is very similar to scale. The difference is that when scale = TRUE, stdize divides the coloumns by their standard deviation, while scale uses the root-mean-square of the coloumns. If center is TRUE, this is equivalent, but in general it is not.

Author(s)

Bjørn-Helge Mevik and Ron Wehrens

See Also

mvr, pcr, plsr, msc, scale

Examples

```
data(yarn)
## Direct standardization:
Ztrain <- stdize(yarn$NIR[yarn$train,])
Ztest <- predict(Ztrain, yarn$NIR[!yarn$train,])

## Used in formula:
mod <- plsr(density ~ stdize(NIR), ncomp = 6, data = yarn[yarn$train,])
pred <- predict(mod, newdata = yarn[!yarn$train,]) # Automatically standardized
```

---

svdpc.fit *Principal Component Regression*

Description

Fits a PCR model using the singular value decomposition.

Usage

```
svdpc.fit(X, Y, ncomp, center = TRUE, stripped = FALSE, ...)
```

Arguments

X	a matrix of observations. NAs and Infs are not allowed.
Y	a vector or matrix of responses. NAs and Infs are not allowed.
ncomp	the number of components to be used in the modelling.
center	logical, determines if the X and Y matrices are mean centered or not. Default is to perform mean centering.
stripped	logical. If TRUE the calculations are stripped as much as possible for speed; this is meant for use with cross-validation or simulations when only the coefficients are needed. Defaults to FALSE.
...	other arguments. Currently ignored.

Details

This function should not be called directly, but through the generic functions `pcr` or `mvr` with the argument `method="svdpc"`. The singular value decomposition is used to calculate the principal components.

Value

A list containing the following components is returned:

`coefficients` an array of regression coefficients for 1, `...`, `ncomp` components. The dimensions of `coefficients` are `c(nvar,npred,ncomp)` with `nvar` the number of `X` variables and `npred` the number of variables to be predicted in `Y`.

`scores` a matrix of scores.

`loadings` a matrix of loadings.

`Yloadings` a matrix of Y-loadings.

`projection` the projection matrix used to convert X to scores.

`Xmeans` a vector of means of the X variables.

`Ymeans` a vector of means of the Y variables.

`fitted.values` an array of fitted values. The dimensions of `fitted.values` are `c(nobj,npred,ncomp)` with `nobj` the number samples and `npred` the number of Y variables.

`residuals` an array of regression residuals. It has the same dimensions as `fitted.values`.

`Xvar` a vector with the amount of X-variance explained by each component.

`Xtotvar` Total variance in `X`.

If `stripped` is `TRUE`, only the components `coefficients`, `Xmeans` and `Ymeans` are returned.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

References

Martens, H., Næs, T. (1989) *Multivariate calibration.* Chichester: Wiley.

See Also

`mvr` `plsr` `pcr` `cppls`

validationplot *Validation Plots*

Description

Functions to plot validation statistics, such as RMSEP or R^2, as a function of the number of components.

Usage

```
validationplot(
  object,
  val.type = c("RMSEP", "MSEP", "R2"),
  estimate,
  newdata,
  ncomp,
  comps,
  intercept,
  ...
)

## S3 method for class 'mvrVal'
plot(
  x,
  nCols,
  nRows,
  type = "l",
  lty = 1:nEst,
  lwd = par("lwd"),
  pch = 1:nEst,
  cex = 1,
  col = 1:nEst,
  legendpos,
  xlab = "number of components",
  ylab = x$type,
  main,
  ask = nRows * nCols < nResp && dev.interactive(),
  ...
)
```

Arguments

`object`	an `mvr` object.
`val.type`	character. What type of validation statistic to plot.
`estimate`	character. Which estimates of the statistic to calculate. See `RMSEP`.
`newdata`	data frame. Optional new data used to calculate statistic.

`ncomp, comps` integer vector. The model sizes to compute the statistic for. See `RMSEP`.

`intercept` logical. Whether estimates for a model with zero components should be calculated as well.

`...` Further arguments sent to underlying plot functions.

`x` an `mvrVal` object. Usually the result of a `RMSEP`, `MSEP` or `R2` call.

`nCols, nRows` integers. The number of coloumns and rows the plots will be laid out in. If not specified, `plot.mvrVal` tries to be intelligent.

`type` character. What type of plots to create. Defaults to `"l"` (lines). Alternative types include `"p"` (points) and `"b"` (both). See `plot` for a complete list of types.

`lty` vector of line types (recycled as neccessary). Line types can be specified as integers or character strings (see `par` for the details).

`lwd` vector of positive numbers (recycled as neccessary), giving the width of the lines.

`pch` plot character. A character string or a vector of single characters or integers (recycled as neccessary). See `points` for all alternatives.

`cex` numeric vector of character expansion sizes (recycled as neccessary) for the plotted symbols.

`col` character or integer vector of colors for plotted lines and symbols (recycled as neccessary). See `par` for the details.

`legendpos` Legend position. Optional. If present, a legend is drawn at the given position. The position can be specified symbolically (e.g., `legendpos = "topright"`). This requires >= 2.1.0. Alternatively, the position can be specified explicitly (`legendpos = t(c(x,y))`) or interactively (`legendpos = locator()`). This only works well for plots of single-response models.

`xlab, ylab` titles for x and y axes. Typically character strings, but can be expressions (e.g., `expression(R^2)` or lists. See `title` for details.

`main` optional main title for the plot. See Details.

`ask` logical. Whether to ask the user before each page of a plot.

Details

`validationplot` calls the proper validation function (currently `MSEP`, `RMSEP` or `R2`) and plots the results with `plot.mvrVal`. `validationplot` can be called through the `mvr` plot method, by specifying `plottype = "validation"`.

`plot.mvrVal` creates one plot for each response variable in the model, laid out in a rectangle. It uses `matplot` for performing the actual plotting. If `legendpos` is given, a legend is drawn at the given position.

The argument `main` can be used to specify the main title of the plot. It is handled in a non-standard way. If there is only on (sub) plot, `main` will be used as the main title of the plot. If there is *more* than one (sub) plot, however, the presence of `main` will produce a corresponding ‘global’ title on the page. Any graphical parametres, e.g., `cex.main`, supplied to `coefplot` will only affect the ‘ordinary’ plot titles, not the ‘global’ one. Its appearance can be changed by setting the parameters with `par`, which will affect *both* titles. (To have different settings for the two titles, one can override the `par` settings with arguments to the plot function.)

Note

legend has many options. If you want greater control over the appearance of the legend, omit the `legendpos` argument and call `legend` manually.

Author(s)

Ron Wehrens and Bjørn-Helge Mevik

See Also

mvr, plot.mvr, RMSEP, MSEP, R2, matplot, legend

Examples

```
data(oliveoil)
mod <- plsr(sensory ~ chemical, data = oliveoil, validation = "LOO")
## Not run:
## These three are equivalent:
validationplot(mod, estimate = "all")
plot(mod, "validation", estimate = "all")
plot(RMSEP(mod, estimate = "all"))
## Plot R2:
plot(mod, "validation", val.type = "R2")
## Plot R2, with a legend:
plot(mod, "validation", val.type = "MSEP", legendpos = "top") # R >= 2.1.0

## End(Not run)
```

var.jack *Jackknife Variance Estimates of Regression Coefficients*

Description

Calculates jackknife variance or covariance estimates of regression coefficients.

The original (Tukey) jackknife variance estimator is defined as $(g-1)/g\sum_{i=1}^{g}(\tilde{\beta}_{-i}-\bar{\beta})^2$, where g is the number of segments, $\tilde{\beta}_{-i}$ is the estimated coefficient when segment i is left out (called the jackknife replicates), and $\bar{\beta}$ is the mean of the $\tilde{\beta}_{-i}$. The most common case is delete-one jackknife, with $g=n$, the number of observations.

This is the definition `var.jack` uses by default.

However, Martens and Martens (2000) defined the estimator as $(g-1)/g\sum_{i=1}^{g}(\tilde{\beta}_{-i}-\hat{\beta})^2$, where $\hat{\beta}$ is the coefficient estimate using the entire data set. I.e., they use the original fitted coefficients instead of the mean of the jackknife replicates. Most (all?) other jackknife implementations for PLSR use this estimator. `var.jack` can be made to use this definition with `use.mean = FALSE`. In practice, the difference should be small if the number of observations is sufficiently large. Note, however, that all theoretical results about the jackknife refer to the 'proper' definition. (Also note that this option might disappear in a future version.)

Usage

```
var.jack(object, ncomp = object$ncomp, covariance = FALSE, use.mean = TRUE)
```

Arguments

`object`	an `mvr` object. A cross-validated model fitted with `jackknife = TRUE`.
`ncomp`	the number of components to use for estimating the (co)variances
`covariance`	logical. If `TRUE`, covariances are calculated; otherwise only variances. The default is `FALSE`.
`use.mean`	logical. If `TRUE` (default), the mean coefficients are used when estimating the (co)variances; otherwise the coefficients from a model fitted to the entire data set. See Details.

Value

If `covariance` is `FALSE`, an $p \times q \times c$ array of variance estimates, where p is the number of predictors, q is the number of responses, and c is the number of components.

If `covariance` id `TRUE`, an $pq \times pq \times c$ array of variance-covariance estimates.

Warning

Note that the Tukey jackknife variance estimator is not unbiased for the variance of regression coefficients (Hinkley 1977). The bias depends on the X matrix. For ordinary least squares regression (OLSR), the bias can be calculated, and depends on the number of observations n and the number of parameters k in the mode. For the common case of an orthogonal design matrix with ± 1 levels, the delete-one jackknife estimate equals $(n-1)/(n-k)$ times the classical variance estimate for the regression coefficients in OLSR. Similar expressions hold for delete-d estimates. Modifications have been proposed to reduce or eliminate the bias for the OLSR case, however, they depend on the number of parameters used in the model. See e.g. Hinkley (1977) or Wu (1986).

Thus, the results of `var.jack` should be used with caution.

Author(s)

Bjørn-Helge Mevik

References

Tukey J.W. (1958) Bias and Confidence in Not-quite Large Samples. (Abstract of Preliminary Report). *Annals of Mathematical Statistics*, **29**(2), 614.

Martens H. and Martens M. (2000) Modified Jack-knife Estimation of Parameter Uncertainty in Bilinear Modelling by Partial Least Squares Regression (PLSR). *Food Quality and Preference*, **11**, 5–16.

Hinkley D.V. (1977), Jackknifing in Unbalanced Situations. *Technometrics*, **19**(3), 285–292.

Wu C.F.J. (1986) Jackknife, Bootstrap and Other Resampling Methods in Regression Analysis. *Te Annals of Statistics*, **14**(4), 1261–1295.

See Also

mvrCv, jack.test

Examples

```
data(oliveoil)
mod <- pcr(sensory ~ chemical, data = oliveoil, validation = "LOO",
           jackknife = TRUE)
var.jack(mod, ncomp = 2)
```

widekernelpls.fit *Wide Kernel PLS (Rännar et al.)*

Description

Fits a PLSR model with the wide kernel algorithm.

Usage

```
widekernelpls.fit(
  X,
  Y,
  ncomp,
  center = TRUE,
  stripped = FALSE,
  tol = .Machine$double.eps^0.5,
  maxit = 100,
  ...
)
```

Arguments

X	a matrix of observations. NAs and Infs are not allowed.
Y	a vector or matrix of responses. NAs and Infs are not allowed.
ncomp	the number of components to be used in the modelling.
center	logical, determines if the X and Y matrices are mean centered or not. Default is to perform mean centering.
stripped	logical. If TRUE the calculations are stripped as much as possible for speed; this is meant for use with cross-validation or simulations when only the coefficients are needed. Defaults to FALSE.
tol	numeric. The tolerance used for determining convergence in the algorithm.
maxit	positive integer. The maximal number of iterations used in the internal Eigenvector calculation.
...	other arguments. Currently ignored.

Details

This function should not be called directly, but through the generic functions `plsr` or `mvr` with the argument `method="widekernelpls"`. The wide kernel PLS algorithm is efficient when the number of variables is (much) larger than the number of observations. For very wide `X`, for instance 12x18000, it can be twice as fast as `kernelpls.fit` and `simpls.fit`. For other matrices, however, it can be much slower. The results are equal to the results of the NIPALS algorithm.

Value

A list containing the following components is returned:

`coefficients` an array of regression coefficients for 1, ..., `ncomp` components. The dimensions of `coefficients` are `c(nvar,npred,ncomp)` with `nvar` the number of `X` variables and `npred` the number of variables to be predicted in `Y`.

`scores` a matrix of scores.

`loadings` a matrix of loadings.

`loading.weights` a matrix of loading weights.

`Yscores` a matrix of Y-scores.

`Yloadings` a matrix of Y-loadings.

`projection` the projection matrix used to convert X to scores.

`Xmeans` a vector of means of the X variables.

`Ymeans` a vector of means of the Y variables.

`fitted.values` an array of fitted values. The dimensions of `fitted.values` are `c(nobj,npred,ncomp)` with `nobj` the number samples and `npred` the number of Y variables.

`residuals` an array of regression residuals. It has the same dimensions as `fitted.values`.

`Xvar` a vector with the amount of X-variance explained by each component.

`Xtotvar` Total variance in `X`.

If `stripped` is `TRUE`, only the components `coefficients`, `Xmeans` and `Ymeans` are returned.

Note

The current implementation has not undergone extensive testing yet, and should perhaps be regarded as experimental. Specifically, the internal Eigenvector calculation does not always converge in extreme cases where the Eigenvalue is close to zero. However, when it does converge, it always converges to the same results as `kernelpls.fit`, up to numerical inacurracies.

The algorithm also has a bit of overhead, so when the number of observations is moderately high, `kernelpls.fit` can be faster even if the number of predictors is much higher. The relative speed of the algorithms can also depend greatly on which BLAS and/or LAPACK library is linked against.

Author(s)

Bjørn-Helge Mevik

References

Rännar, S., Lindgren, F., Geladi, P. and Wold, S. (1994) A PLS Kernel Algorithm for Data Sets with Many Variables and Fewer Objects. Part 1: Theory and Algorithm. *Journal of Chemometrics*, **8**, 111–125.

See Also

`mvr` `plsr` `cppls` `pcr` `kernelpls.fit` `simpls.fit` `oscorespls.fit`

`yarn` *NIR spectra and density measurements of PET yarns*

Description

A training set consisting of 21 NIR spectra of PET yarns, measured at 268 wavelengths, and 21 corresponding densities. A test set of 7 samples is also provided. Many thanks to Erik Swierenga.

Format

A data frame with components

NIR Numeric matrix of NIR measurements

density Numeric vector of densities

train Logical vector with `TRUE` for the training samples and `FALSE` for the test samples

Source

Swierenga H., de Weijer A. P., van Wijk R. J., Buydens L. M. C. (1999) Strategy for constructing robust multivariate calibration models *Chemometrics and Intelligent Laboratoryy Systems*, **49**(1), 1–17.

Index