Package: rchemo (via r-universe)

June 30, 2026

Type Package

Title Dimension Reduction, Regression and Discrimination for Chemometrics

Version 0.1-4

Description Data exploration and prediction with focus on high dimensional data and chemometrics. The package was initially designed about partial least squares regression and discrimination models and variants, in particular locally weighted PLS models (LWPLS). Then, it has been expanded to many other methods for analyzing high dimensional data. The name 'rchemo' comes from the fact that the package is orientated to chemometrics, but most of the provided methods are fully generic to other domains. Functions such as transform(), predict(), coef() and summary() are available. Tuning the predictive models is facilitated by generic functions gridscore() (validation dataset) and gridcv() (cross-validation). Faster versions are also available for models based on latent variables (LVs) (gridscorelv() and gridcvlv()) and ridge regularization (gridscorelb() and gridcvlb()).

Imports stats, graphics, grDevices, data.table, FNN, signal, e1071, utils

Depends R (>= 4.0)

VignetteBuilder knitr

Suggests knitr, rmarkdown

URL https://github.com/ChemHouse-group/rchemo/

License GPL-3

LazyData yes

NeedsCompilation no

Author Marion Brandolini-Bunlon [aut, cre], Benoit Jaillais [aut], Jean-Michel Roger [aut], Matthieu Lesnoff [aut]

Maintainer Marion Brandolini-Bunlon
<marion.brandolini-bunlon@inrae.fr>

Repository https://cran.r-universe.dev

Date/Publication 2026-06-30 16:00:07 UTC

RemoteUrl https://github.com/cran/rchemo

RemoteRef HEAD

RemoteSha 635857048e27ac266306df9a716a274b3cf237da

Contents

`aggmean` *Centers of classes*

Description

Calculation of the centers (means) of classes of row observations of a data set.

Usage

```
aggmean(X, y = NULL)
```

Arguments

`X`	Data (n, p) for which are calculated the centers (column-wise means).
`y`	Class membership $(n, 1)$ of the row of `X`. Default to `NULL` (all the rows of are considered).

Value

`ct`	centers (column-wise means)
`lev`	classes
`ni`	number of observations in each per class

Examples

```
n <- 8 ; p <- 6
X <- matrix(rnorm(n * p, mean = 10), ncol = p, byrow = TRUE)
y <- sample(1:2, size = n, replace = TRUE)
aggmean(X, y)

data(forages)
Xtrain <- forages$Xtrain
ytrain <- forages$ytrain
table(ytrain)
u <- aggmean(Xtrain, ytrain)$ct
headm(u)
plotsp(u, col = 1:4, main = "Means")
x <- Xtrain[1:20, ]
plotsp(x, ylab = "Absorbance", col = "grey")
u <- aggmean(x)$ct
```

```
plotsp(u, col = "red", add = TRUE, lwd = 2)
```

---

aicplsr *AIC and Cp for Univariate PLSR Models*

Description

Computation of the AIC and Mallows's Cp criteria for univariate PLSR models (Lesnoff et al. 2021). This function may receive modifications in the future (work in progress).

Usage

```
aicplsr(
    X, y, nlv, algo = NULL,
    meth = c("cg", "div", "cov"),
    correct = TRUE, B = 50,
    print = FALSE, ...)
```

Arguments

X	A nxp matrix or data frame of training observations.
y	A vector of length n of training responses.
nlv	The maximal number of latent variables (LVs) to consider in the model.
algo	a PLS algorithm. Default to NULL (plskern is used).
meth	Method used for estimating df. Possible values are "cg" (dfplsr_cg), "cov" (dfplsr_cov)or "div" (dfplsr_div).
correct	Logical. If TRUE (default), the AICc corection is applied to the criteria.
B	For meth = "div": the number of observations in the data receiving perturbation (maximum is n; see dfplsr_cov). For meth = "cov": the number of bootstrap replications (see dfplsr_cov).
print	Logical. If TRUE, fitting information are printed.
...	Optionnal arguments to pass in algo.

Details

For a model with a latent variables (LVs), function aicplsr calculates AIC and Cp by:

$AIC(a) = n * log(SSR(a)) + 2 * (df(a) + 1)$

$Cp(a) = SSR(a)/n + 2 * df(a) * s2/n$

where SSR is the sum of squared residuals for the current evaluated model, $df(a)$ the estimated PLSR model complexity (i.e. nb. model's degrees of freedom), $s2$ an estimate of the irreductible error variance (computed from a low biased model) and n the number of training observations.

By default (argument `correct`), the small sample size correction (so-called AICc) is applied to AIC and Cp for deucing the bias.

The functions returns two estimates of Cp ($cp1$ and $cp2$), each corresponding to a different estimate of $s2$.

The model complexity df can be computed from three methods (argument `meth`).

Value

`crit`	dataframe with n, and the etimated criteria (df, ct, ssr, aic, $cp1, cp2$) for 0 to nlv latent variables in the model.
`delta`	dataframe with the differences between the estimated values of aic, $cp1$ and $cp2$, and those of the model with the lowest estimated values of aic, $cp1$ and $cp2$, for models with 0 to nlv latent variables
`opt`	vector with the optimal number of latent variables in the model (i.e. minimizing aic, cp1 and cp2 values)

References

Burnham, K.P., Anderson, D.R., 2002. Model selection and multimodel inference: a practical informationtheoretic approach, 2nd ed. Springer, New York, NY, USA.

Burnham, K.P., Anderson, D.R., 2004. Multimodel Inference: Understanding AIC and BIC in Model Selection. Sociological Methods & Research 33, 261-304. https://doi.org/10.1177/0049124104268644

Efron, B., 2004. The Estimation of Prediction Error. Journal of the American Statistical Association 99, 619-632. https://doi.org/10.1198/016214504000000692

Eubank, R.L., 1999. Nonparametric Regression and Spline Smoothing, 2nd ed, Statistics: Textbooks and Monographs. Marcel Dekker, Inc., New York, USA.

Hastie, T., Tibshirani, R.J., 1990. Generalized Additive Models, Monographs on statistics and applied probablity. Chapman and Hall/CRC, New York, USA.

Hastie, T., Tibshirani, R., Friedman, J., 2009. The elements of statistical learning: data mining, inference, and prediction, 2nd ed. Springer, NewYork.

Hastie, T., Tibshirani, R., Wainwright, M., 2015. Statistical Learning with Sparsity: The Lasso and Generalizations. CRC Press

Hurvich, C.M., Tsai, C.-L., 1989. Regression and Time Series Model Selection in Small Samples. Biometrika 76, 297. https://doi.org/10.2307/2336663

Lesnoff, M., Roger, J.M., Rutledge, D.N., Submitted. Monte Carlo methods for estimating Mallows's Cp and AIC criteria for PLSR models. Illustration on agronomic spectroscopic NIR data. Journal of Chemometrics.

Mallows, C.L., 1973. Some Comments on Cp. Technometrics 15, 661-675. https://doi.org/10.1080/00401706.1973.1048910

Ye, J., 1998. On Measuring and Correcting the Effects of Data Mining and Model Selection. Journal of the American Statistical Association 93, 120-131. https://doi.org/10.1080/01621459.1998.10474094

Zuccaro, C., 1992. Mallows'Cp Statistic and Model Selection in Multiple Linear Regression. International Journal of Market Research. 34, 1-10. https://doi.org/10.1177/147078539203400204

Examples

```
data(cassav)

Xtrain <- cassav$Xtrain
ytrain <- cassav$ytrain

nlv <- 25
res <- aicplsr(Xtrain, ytrain, nlv = nlv)
names(res)
headm(res$crit)

z <- res$crit
oldpar <- par(mfrow = c(1, 1))
par(mfrow = c(1, 4))
plot(z$df[-1])
plot(z$aic[-1], type = "b", main = "AIC")
plot(z$cp1[-1], type = "b", main = "Cp1")
plot(z$cp2[-1], type = "b", main = "Cp2")
par(oldpar)
```

---

asdgap *asdgap*

Description

ASD NIRS dataset, with gaps in the spectra at wawelengths = 1000 and 1800 nm.

Usage

```
data(asdgap)
```

Format

A list with 1 element: the data frame X with 5 spectra and 2151 variables.

References

Thanks to J.-F. Roger (Inrae, France) and M. Ecarnot (Inrae, France) for the method.

Examples

```
data(asdgap)
names(asdgap)
X <- asdgap$X

numcol <- which(colnames(X) == "1000" | colnames(X) == "1800")
numcol
plotsp(X, lwd = 1.5)
```

```
abline(v = as.numeric(colnames(X)[1]) + numcol - 1, col = "grey", lty = 3)
```

`blockscal` *Block autoscaling*

Description

Functions managing blocks of data.

- `blockscal`: Autoscales a list of blocks (i.e. sets of columns) of a training X-data, and eventually the blocks of new X-data. The scaling factor (computed on the training) is the "norm" of the block, i.e. the square root of the sum of the variances of each column of the block.

- `mblocks`: Makes a list of blocks from X-data.

- `hconcat`: Concatenates horizontally the blocks of a list.

Usage

```
blockscal(Xtrain, X = NULL, weights = NULL)

mblocks(X, blocks)

hconcat(X)
```

Arguments

`Xtrain`	A list of blocks of training X-data
`X`	For `blockscal`: A list of blocks of new X-data. For `mblocks`: X-data. For `hconcat`: a list of blocks of X-data.
`blocks`	A list (of same length as the number of blocks) giving the column numbers in `X`.
`weights`	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to `NULL` (weights are set to $1/n$).

Value

For `mblocks`: a list of blocks of X-data.

For `hconcat`: a matrix concatenating a list of data blocks.

For `blockscal`:

`Xtrain`	A list of blocks of training X-data, after block autoscaling.
`X`	A list of blocks of new X-data, after block autoscaling.
`disp`	The scaling factor (computed on the training).

Note

The second example is equivalent to MB-PLSR

Examples

```
n <- 10 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
m <- 2

Xtest <- matrix(rnorm(m * p), ncol = p)
colnames(Xtest) <- paste("v", 1:p, sep = "")
Xtrain
Xtest

blocks <- list(1:2, 4, 6:8)
zXtrain <- mblocks(Xtrain, blocks = blocks)
zXtest <- mblocks(Xtest, blocks = blocks)

zXtrain
blockscal(zXtrain, zXtest)

res <- blockscal(zXtrain, zXtest)
hconcat(res$Xtrain)
hconcat(res$X)

## example of equivalence with MB-PLSR

n <- 10 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
m <- 2

Xtest <- matrix(rnorm(m * p), ncol = p)
colnames(Xtest) <- paste("v", 1:p, sep = "")
Xtrain
Xtest

blocks <- list(1:2, 4, 6:8)
X1 <- mblocks(Xtrain, blocks = blocks)
X1 <- lapply(1:length(X1), function(x) scale(X1[[x]]))
res <- blockscal(X1)
zXtrain <- hconcat(res$Xtrain)

nlv <- 3
fm <- plskern(zXtrain, ytrain, nlv = nlv)
```

---

cassav *cassav*

Description

A NIRS dataset (absorbance) describing the concentration of a natural pigment in samples of tropical shrubs. Spectra were recorded from 400 to 2498 nm at 2 nm intervals.

Usage

```
data(cassav)
```

Format

A list with the following components:

For the reference (calibration) data:

`Xtrain` A matrix whose rows are the NIR absorbance spectra (= log10(1 / Reflectance)).

`ytrain` A vector of the response variable (pigment concentration).

`year` A vector of the year of data collection (2009 to 2012; the test set correponds to year 2013).

For the test data:

`Xtest` A matrix whose rows are the NIR absorbance spectra (= log10(1 / Reflectance)).

`ytest` A vector of the response variable (pigment concentration).

References

Davrieux, F., Dufour, D., Dardenne, P., Belalcazar, J., Pizarro, M., Luna, J., Londono, L., Jaramillo, A., Sanchez, T., Morante, N., Calle, F., Becerra Lopez-Lavalle, L., Ceballos, H., 2016. LOCAL regression algorithm improves near infrared spectroscopy predictions when the target constituent evolves in breeding populations. Journal of Near Infrared Spectroscopy 24, 109. https://doi.org/10.1255/jnirs.1213

CIAT Cassava Project (Colombia), CIRAD Qualisud Research Unit, and funded mainly by the CGIAR Research Program on Roots, Tubers and Bananas (RTB) with support from CGIAR Trust Fund contributors (https://www.cgiar.org/funders/).

Examples

```
data(cassav)
str(cassav)
```

---

cglsr *CG Least Squares Models*

Description

Conjugate gradient algorithm (CG) for the normal equations (CGLS algorithm 7.4.1, Bjorck 1996, p.289)

Usage

```
cglsr(X, y, nlv, reorth = TRUE, filt = FALSE)

## S3 method for class 'Cglsr'
coef(object, ..., nlv = NULL)

## S3 method for class 'Cglsr'
predict(object, X, ..., nlv = NULL)
```

Arguments

X	For the main function: Training X-data (n, p). — For auxiliary functions: New X-data (m, p) to consider.
y	Univariate training Y-data $(n, 1)$.
nlv	The number(s) of CG iterations.
reorth	Logical. If TRUE, a Gram-Schmidt reorthogonalization of the normal equation residual vectors is done.
filt	Logical. If TRUE, the filter factors are computed (output F).
object	For auxiliary functions: A fitted model, output of a call to the main functions.
...	For auxiliary functions: Optional arguments. Not used.

Details

The code for re-orthogonalization (Hansen 1998) and filter factors (Vogel 1987, Hansen 1998) computations is a transcription (with few adaptations) of the matlab function 'cgls' (Saunders et al. https://web.stanford.edu/group/SOL/software/cgls/; Hansen 2008).

The filter factors can be used to compute the model complexity of CGLSR and PLSR models (see dfplsr_cg).

Data X and y are internally centered.

Missing values are not allowed.

Value

For cglsr:

B	matrix with the model coefficients for the fix nlv.
gnew	squared norm of the s vector
xmeans	variable means for the training X-data
ymeans	variable means for the training Y-data
F	If filt = TRUE, the filter factors

For coef.Cglsr :

int	intercept value.

`B` matrix with the model coefficients.

For `predict.Cglsr` :

`pred` list of matrices, with the predicted values for each number `nlv` of CG iterations

References

Bjorck, A., 1996. Numerical Methods for Least Squares Problems, Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9781611971484

Hansen, P.C., 1998. Rank-Deficient and Discrete Ill-Posed Problems, Mathematical Modeling and Computation. Society for Industrial and Applied Mathematics. https://doi.org/10.1137/1.9780898719697

Hansen, P.C., 2008. Regularization Tools version 4.0 for Matlab 7.3. Numer Algor 46, 189-194. https://doi.org/10.1007/s11075-007-9136-9

Manne R. Analysis of two partial-least-squares algorithms for multivariate calibration. Chemometrics Intell. Lab. Syst. 1987; 2: 187-197.

Phatak A, De Hoog F. Exploiting the connection between PLS, Lanczos methods and conjugate gradients: alternative proofs of some properties of PLS. J. Chemometrics 2002; 16: 361-367.

Vogel, C. R., "Solving ill-conditioned linear systems using the conjugate gradient method", Report, Dept. of Mathematical Sciences, Montana State University, 1987.

Examples

```
z <- ozone$X
u <- which(!is.na(rowSums(z)))
X <- z[u, -4]
y <- z[u, 4]
dim(X)
headm(X)
Xtest <- X[1:2, ]
ytest <- y[1:2]

nlv <- 10
fm <- cglsr(X, y, nlv = nlv)

coef(fm)
coef(fm, nlv = 1)

predict(fm, Xtest)
predict(fm, Xtest, nlv = 1:3)

pred <- predict(fm, Xtest)$pred
msep(pred, ytest)

cglsr(X, y, nlv = 5, filt = TRUE)$F
```

checkdupl *Duplicated rows in datasets*

Description

Finding and removing duplicated row observations in datasets.

Usage

```
checkdupl(X, Y = NULL, digits = NULL)
```

Arguments

X	A dataset.
Y	A dataset compared to X.
digits	The number of digits when rounding the data before the duplication test. Default to NULL (no rounding.

Value

a dataframe with the row numbers in the first and second datasets that are identical, and the values of the variables.

Examples

```
X1 <- matrix(c(1:5, 1:5, c(1, 2, 7, 4, 8)), nrow = 3, byrow = TRUE)
dimnames(X1) <- list(1:3, c("v1", "v2", "v3", "v4", "v5"))

X2 <- matrix(c(6:10, 1:5, c(1, 2, 7, 6, 12)), nrow = 3, byrow = TRUE)
dimnames(X2) <- list(1:3, c("v1", "v2", "v3", "v4", "v5"))

X1
X2

checkdupl(X1, X2)

checkdupl(X1)

checkdupl(matrix(rnorm(20), nrow = 5))

res <- checkdupl(X1)
s <- unique(res$rownum2)
zX1 <- X1[-s, ]
zX1
```

checkna *Find and count NA values in a dataset*

Description

Find and count NA values in each row observation of a dataset.

Usage

```
checkna(X)
```

Arguments

X	A dataset.

Value

A data frame summarizing the numbers of NA by rows.

Examples

```
X <- data.frame(
  v1 = c(NA, rnorm(9)),
  v2 = c(NA, rnorm(8), NA),
  v3 = c(NA, NA, NA, rnorm(7))
)
X

checkna(X)
```

consensuspca *consensus PCA*

Description

Algorithms fitting a consensus PCA of a list of matrices $Xlist$. A chosen PCA algorithm is applied on the X-matrix obtained after variable scaling, blockscaling, block concatenation.

Auxiliary functions

`transform` Calculates the principal components for any new matrix X from the model.

`summary` returns summary information for the model.

Usage

```
consensuspca(Xlist, blockscaling = TRUE, weights = NULL, nlv,
      Xscaling = c("none", "pareto", "sd")[1],
      algo = c("svd","eigen","eigenk","nipals","nipalsna","sph")[1],
      gs = TRUE, tol = .Machine$double.eps^0.5, maxit = 200)

## S3 method for class 'Consensuspca'
transform(object, X, ..., nlv = NULL)

## S3 method for class 'Consensuspca'
summary(object, X, ...)
```

Arguments

Xlist	For the main function: list of training X-data ($n$rows).
X	For the auxiliary functions: list of new X-data, with the same variables than the training X-data.
blockscaling	logical. If TRUE, the scaling factor (computed on the training) is the "norm" of the block, i.e. the square root of the sum of the variances of each column of the block.
weights	excepted for "nipalsna". Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	For the main functions: The number(s) of PCs to calculate. — For the auxiliary functions: The number(s) of PCs to consider.
Xscaling	vector (of length Xlist) of variable scaling for each datablock, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
algo	For consensuspca, the algorithm for the PCA among : "svd" (SVD factorization of $D^(1/2) * X$, using function svd), "eigen" (Eigen factorization of $X' * D * X$, using function eigen), "eigenk" (Eigen factorization of $D^(1/2) * X * X' D^(1/2)$, using function eigen), "nipals" (Eigen factorization of $X' * D * X$ using NIPALS), "nipalsna" (Eigen factorization of $X' * D * X$ using NIPALS allowing missing data in X), "sph" (Robust spherical PCA)
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Specific for the NIPALS algorithm

gs	Logical indicating if a Gram-Schmidt orthogonalization is implemented or not (default to TRUE).
tol	Tolerance for testing convergence of the NIPALS iterations for each PC.
maxit	Maximum number of NIPALS iterations for each PC.

Value

For `consensuspca`:

`T`	The X-score matrix (n, nlv).
`P`	The X-loadings matrix (p, nlv).
`sv`	The singular values $(min(n,p), 1)$ except for NIPALS = $(nlv, 1)$.
`eig`	The eigenvalues (= `sv^2`) $(min(n,p), 1)$ except for NIPALS = $(nlv, 1)$.
`xmeans`	The list of centering vectors of $Xlist$.
`xscales`	The list of $Xlist$ variable standard deviations.
`weights`	Weights applied to the training observations.
`blockscaling`	block scaling.
`Xnorms`	"norm" of each block, i.e. the square root of the sum of the variances of each column of each block, computed on the training, and used as scaling factor

.

`niter`	Numbers of iterations of the NIPALS.
`conv`	Logical indicating if the NIPALS converged before reaching the maximal number of iterations.

For `transform.Consensuspca`: X-scores matrix for new Xlist-data.

For `summary.Consensuspca`:

`explvarx`	matrix of explained variances.
`contr_ind`	observation contributions.
`contr_var`	variable contributions.
`coord_var`	variable coordinates.
`cor_circle`	variable coordinates on the correlation circle.

References

Mangamana, E.T., Cariou, V., Vigneau, E., Glele Kakai, R.L., Qannari, E.M., 2019. Unsupervised multiblock data analysis: A unified approach and extensions. Chemometrics and Intelligent Laboratory Systems 194, 103856. https://doi.org/10.1016/j.chemolab.2019.103856

Westerhuis, J.A., Kourti, T., MacGregor, J.F., 1998. Analysis of multiblock and hierarchical PCA and PLS models. Journal of Chemometrics 12, 301–321. https://doi.org/10.1002/(SICI)1099-128X(199809/10)12:5<301::AID-CEM515>3.0.CO;2-S

Examples

```
n <- 10 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)

m <- 2
Xtest <- matrix(rnorm(m * p), ncol = p)

colnames(Xtrain) <- colnames(Xtest) <- paste("v", 1:p, sep = "")

Xtrain
Xtest

blocks <- list(1:2, 4, 6:8)
X1 <- mblocks(Xtrain, blocks = blocks)
X2 <- mblocks(Xtest, blocks = blocks)

nlv <- 3
fm <- consensuspca(Xlist = X1, Xscaling = c("sd","none","none"),
blockscaling = TRUE, weights = NULL, nlv = nlv)

summary(fm, X1)
transform(fm, X2)
```

---

covsel *CovSel*

Description

Variable selection for high-dimensionnal data with the COVSEL method (Roger et al. 2011).

Usage

```
covsel(X, Y, nvar = NULL, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL)
```

Arguments

X	X-data (n, p).
Y	Y-data (n, q).
nvar	Number of variables to select in X.
Xscaling	X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.

Yscaling	Y variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).

Value

sel	A dataframe where variable sel shows the column numbers of the variables selected in X.
weights	The weights used for the row observations.

References

Roger, J.M., Palagos, B., Bertrand, D., Fernandez-Ahumada, E., 2011. CovSel: Variable selection for highly multivariate and multi-response calibration: Application to IR spectroscopy. Chem. Lab. Int. Syst. 106, 216-223.

Examples

```
n <- 6 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
Y <- matrix(rnorm(n * 2), ncol = 2)

covsel(X, Y, nvar = 3)
```

---

covsellmr *CovSel-linear regression model*

Description

Variable selection for high-dimensionnal data with the COVSEL method (Roger et al. 2011), followed by a linear regression model.

Auxiliary functions

predict Calculates the predictions from the regression model for any new set of variables contained in the selection.

Usage

```
covsellmr(X, Y, nvar = NULL, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL)

## S3 method for class 'Covsellmr'
predict(object, X, ..., nvar = NULL)
```

Arguments

X	X-data (n, p).
Y	Y-data (n, q).
nvar	Number of variables to select in X.
Xscaling	X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	Y variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

sel	A dataframe where variable sel shows the column indexes of the variables selected in X.
fm	List of linear regression models, involving 1 to nvar selected explicative variables.
weights	The weights used for the row observations.

References

Roger, J.M., Palagos, B., Bertrand, D., Fernandez-Ahumada, E., 2011. CovSel: Variable selection for highly multivariate and multi-response calibration: Application to IR spectroscopy. Chem. Lab. Int. Syst. 106, 216-223.

Examples

```
n <- 6 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
Y <- matrix(rnorm(n * 2), ncol = 2)

sel <- covsellmr(X, Y, nvar = 3)

predict(sel, X, nvar = c(2,3))
```

covselrda *CovSel-discriminant analysis*

Description

Variable selection for high-dimensionnal data with the COVSEL method (Roger et al. 2011), followed by a linear regression model.

The training variable y (univariate class membership) is firstly transformed to a dummy table containing $nclas$ columns, where $nclas$ is the number of classes present in y. Each column is a dummy variable (0/1). Then, a variable selection, based on the COVSEL method, is implemented on the $X-$data and the dummy table, returning a set of X-variables that are used as dependent variables in a DA model.

- covselrda: A linear regression model predicts the Y-dummy table from the selected X-variables. For a given observation, the final prediction is the class corresponding to the dummy variable for which the prediction is the highest.

- covsellda and covselqda: Probabilistic LDA and QDA are run over the selected X-variables, respectively.

Auxiliary functions

predict Calculates the predictions for any new set of variables contained in the selection.

Usage

```
covselrda(X, y, nvar = NULL, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL)

covsellda(X, y, nvar = NULL, prior = c("unif", "prop"),
Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL)

covselqda(X, y, nvar = NULL, prior = c("unif", "prop"),
Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL)

## S3 method for class 'Covselrda'
predict(object, X, ..., nvar = NULL)

## S3 method for class 'Covselprobda'
predict(object, X, ..., nvar = NULL)
```

Arguments

X	X-data (n, p).
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.

nvar	Number of variables to select in X. Can be a vector for the auxiliary functions
prior	The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
Xscaling	X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	Y variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For covselrda, covselda, covselqda:

sel	A dataframe where variable sel shows the column indexes of the variables selected in X.
fm	List of linear regression or discriminant models, involving 1 to nvar selected explicative variables.
lev	classes
ni	number of observations in each class
weights	The weights used for the row observations.

For predict.Covselrda, predict.Covselprobda:

pred	predicted class for each observation
posterior	calculated probability of belonging to a class for each observation

References

Roger, J.M., Palagos, B., Bertrand, D., Fernandez-Ahumada, E., 2011. CovSel: Variable selection for highly multivariate and multi-response calibration: Application to IR spectroscopy. Chem. Lab. Int. Syst. 106, 216-223.

Examples

```
n <- 6 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
y <- c("A","A","B","B","C","C")

sel <- covselrda(X, y, nvar = 3)

predict(sel, X, nvar = c(2,3))
```

dderiv *Derivation by finite difference*

Description

Calculation of the first derivatives, by finite differences, of the row observations (e.g. spectra) of a dataset.

Usage

```
dderiv(X, n = 5, ts = 1)
```

Arguments

X	X-data (n, p).
n	The number of points (i.e. columns of X) defining the window over wich is calculate each finite difference. The derivation is calculated for the point at the center of the window. Therefore, n must be an odd integer, and be higher or equal to 3.
ts	A scaling factor for the finite differences (by default, ts = 1.)

Value

A matrix of the transformed data.

Examples

```
data(cassav)

X <- cassav$Xtest

n <- 15
Xp_derivate1 <- dderiv(X, n = n)
Xp_derivate2 <- dderiv(dderiv(X, n), n)

oldpar <- par(mfrow = c(1, 1))
par(mfrow = c(1, 2))
plotsp(X, main = "Signal")
plotsp(Xp_derivate1, main = "Corrected signal")
abline(h = 0, lty = 2, col = "grey")
par(oldpar)
```

`detrend` *Polynomial de-trend transformation*

Description

Polynomial de-trend transformation of row observations (e.g. spectra) of a dataset. The function fits an orthogonal polynom of a given degree to each observation and returns the residuals.

Usage

```
detrend(X, degree = 1)
```

Arguments

`X`	X-data (n, p).
`degree`	Degree of the polynom.

Details

`detrend` uses function `poly` of package `stats`.

Value

A matrix of the transformed data.

Examples

```
data(cassav)

X <- cassav$Xtest

degree <- 1
Xp <- detrend(X, degree = degree)

oldpar <- par(mfrow = c(1, 1))
par(mfrow = c(1, 2))
plotsp(X, main = "Signal")
plotsp(Xp, main = "Corrected signal")
abline(h = 0, lty = 2, col = "grey")
par(oldpar)
```

dfplsr_cg *Degrees of freedom of Univariate PLSR Models*

Description

Computation of the model complexity df (number of degrees of freedom) of univariate PLSR models (with intercept). See Lesnoff et al. 2021 for an illustration.

(1) Estimation from the CGLSR algorithm (Hansen, 1998).

- `dfplsr_cov`

(2) Monte Carlo estimation (Ye, 1998 and Efron, 2004). Details in relation with the functions are given in Lesnoff et al. 2021.

- `dfplsr_cov`: The covariances are computed by parametric bootstrap (Efron, 2004, Eq. 2.16). The residual variance $sigma^2$ is estimated from a low-biased model.

- `dfplsr_div`: The divergencies dy_fit/dy are computed by perturbation analysis(Ye, 1998 and Efron, 2004). This is a Stein unbiased risk estimation (SURE) of df.

Usage

```
dfplsr_cg(X, y, nlv, reorth = TRUE)

dfplsr_cov(
    X, y, nlv, algo = NULL,
    maxlv = 50, B = 30, print = FALSE, ...)

dfplsr_div(
    X, y, nlv, algo = NULL,
    eps = 1e-2, B = 30, print = FALSE, ...)
```

Arguments

`X`	A nxp matrix or data frame of training observations.
`y`	A vector of length n of training responses.
`nlv`	The maximal number of latent variables (LVs) to consider in the model.
`reorth`	For `dfplsr_cg`: Logical. If `TRUE`, a Gram-Schmidt reorthogonalization of the normal equation residual vectors is done.
`algo`	a PLS algorithm. Default to `NULL` (`plskern` is used).
`maxlv`	For `dfplsr_cov`: dDmension of the PLSR model (nb. LVs) used for parametric bootstrap.
`eps`	For `dfplsr_div`: The $epsilon$ quantity used for scaling the perturbation analysis.
`B`	For `dfplsr_cov`: Number of bootstrap replications. For `dfplsr_div`: number of observations in the data receiving perturbation (the maximum is n).
`print`	Logical. If `TRUE`, fitting information are printed.
`...`	Optionnal arguments to pass in the function defined in `algo`.

Details

Missing values are not allowed.

The example below reproduces the numerical illustration given by Kramer & Sugiyama 2011 on the Ozone data (Fig. 1, center). The $pls.model$ function from the R package "plsdof" v0.2-9 (Kramer & Braun 2019) is used for df calculations ($df.kramer$), and automatically scales the X matrix before PLS. The example scales also X for consistency when using the other functions.

For the Monte Carlo estimations, `B` Should be increased for more stability

Value

A list of outputs :

`df`	vector with the model complexity for the models with $a = 0, 1, ..., nlv$ components.
`cov`	For `dfplsr_cov`: vector with covariances, computed by parametric bootstrap.

References

Efron, B., 2004. The Estimation of Prediction Error. Journal of the American Statistical Association 99, 619-632. https://doi.org/10.1198/016214504000000692

Hastie, T., Tibshirani, R.J., 1990. Generalized Additive Models, Monographs on statistics and applied probablity. Chapman and Hall/CRC, New York, USA.

Hastie, T., Tibshirani, R., Friedman, J., 2009. The elements of statistical learning: data mining, inference, and prediction, 2nd ed. Springer, NewYork.

Hastie, T., Tibshirani, R., Wainwright, M., 2015. Statistical Learning with Sparsity: The Lasso and Generalizations. CRC Press

Kramer, N., Braun, M.L., 2007. Kernelizing PLS, degrees of freedom, and efficient model selection, in: Proceedings of the 24th International Conference on Machine Learning, ICML 07. Association for Computing Machinery, New York, NY, USA, pp. 441-448. https://doi.org/10.1145/1273496.1273552

Kramer, N., Sugiyama, M., 2011. The Degrees of Freedom of Partial Least Squares Regression. Journal of the American Statistical Association 106, 697-705. https://doi.org/10.1198/jasa.2011.tm10107

Kramer, N., Braun, M. L. 2019. plsdof: Degrees of Freedom and Statistical Inference for Partial Least Squares Regression. R package version 0.2-9. https://cran.r-project.org

Lesnoff, M., Roger, J.M., Rutledge, D.N., 2021. Monte Carlo methods for estimating Mallow's Cp and AIC criteria for PLSR models. Illustration on agronomic spectroscopic NIR data. Journal of Chemometrics, 35(10), e3369. https://doi.org/10.1002/cem.3369

Stein, C.M., 1981. Estimation of the Mean of a Multivariate Normal Distribution. The Annals of Statistics 9, 1135-1151.

Ye, J., 1998. On Measuring and Correcting the Effects of Data Mining and Model Selection. Journal of the American Statistical Association 93, 120-131. https://doi.org/10.1080/01621459.1998.10474094

Zou, H., Hastie, T., Tibshirani, R., 2007. On the degrees of freedom of the lasso. The Annals of Statistics 35, 2173-2192. https://doi.org/10.1214/009053607000000127

Examples

```
## EXAMPLE 1

data(ozone)

z <- ozone$X
u <- which(!is.na(rowSums(z)))
X <- z[u, -4]
y <- z[u, 4]
dim(X)

Xs <- scale(X)

nlv <- 12
res <- dfplsr_cg(Xs, y, nlv = nlv)

df.kramer <- c(1.000000, 3.712373, 6.456417, 11.633565, 12.156760, 11.715101, 12.349716,
  12.192682, 13.000000, 13.000000, 13.000000, 13.000000, 13.000000)

znlv <- 0:nlv
plot(znlv, res$df, type = "l", col = "red",
     ylim = c(0, 15),
     xlab = "Nb components", ylab = "df")
lines(znlv, znlv + 1, col = "grey40")
points(znlv, df.kramer, pch = 16)
abline(h = 1, lty = 2, col = "grey")
legend("bottomright", legend=c("dfplsr_cg","Naive df","df.kramer"), col=c("red","grey40","black"),
lty=c(1,1,0), pch=c(NA,NA,16), bty="n")

## EXAMPLE 2

data(ozone)

z <- ozone$X
u <- which(!is.na(rowSums(z)))
X <- z[u, -4]
y <- z[u, 4]
dim(X)

Xs <- scale(X)

nlv <- 12
B <- 50
u <- dfplsr_cov(Xs, y, nlv = nlv, B = B)
v <- dfplsr_div(Xs, y, nlv = nlv, B = B)

df.kramer <- c(1.000000, 3.712373, 6.456417, 11.633565, 12.156760, 11.715101, 12.349716,
  12.192682, 13.000000, 13.000000, 13.000000, 13.000000, 13.000000)

znlv <- 0:nlv
plot(znlv, u$df, type = "l", col = "red",
     ylim = c(0, 15),
```

```
     xlab = "Nb components", ylab = "df")
lines(znlv, v$df, col = "blue")
lines(znlv, znlv + 1, col = "grey40")
points(znlv, df.kramer, pch = 16)
abline(h = 1, lty = 2, col = "grey")
legend("bottomright", legend=c("dfplsr_cov","dfplsr_div","Naive df","df.kramer"),
col=c("blue","red","grey40","black"),
lty=c(1,1,1,0), pch=c(NA,NA,NA,16), bty="n")
```

---

dkplsr *Direct KPLSR Models*

Description

Direct kernel PLSR (DKPLSR) (Bennett & Embrechts 2003). The method builds kernel Gram matrices and then runs a usual PLSR algorithm on them. This is faster (but not equivalent) to the "true" NIPALS KPLSR algorithm such as described in Rosipal & Trejo (2001).

Usage

```
dkplsr(X, Y, weights = NULL, nlv, kern = "krbf", ...)

## S3 method for class 'Dkpls'
transform(object, X, ..., nlv = NULL)

## S3 method for class 'Dkpls'
coef(object, ..., nlv = NULL)

## S3 method for class 'Dkplsr'
predict(object, X, ..., nlv = NULL)
```

Arguments

X	For the main function: Matrix with the training X-data (n, p). — For auxiliary functions: A matrix with new X-data (m, p) to consider.
Y	Matrix with the training Y-data (n, q).
weights	vector of weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	For the main function: The number(s) of LVs to calculate. — For auxiliary functions: The number(s) of LVs to consider.
kern	Name of the function defining the considered kernel for building the Gram matrix. See krbf for syntax, and other available kernel functions (krbf, kpol, ktanh).
...	Optional arguments to pass in the kernel function defined in kern (e.g. gamma for krbf, gamma and coef0 for ktanh, gamma and coef0 and degree for kpol).
object	For auxiliary functions: A fitted model, output of a call to the main function.

Value

For `dkplsr`:

`X`	Matrix with the training X-data (n, p).
`fm`	List with the outputs of the PLSR ((`T`): the X-score matrix (n,nlv); (`P`): the X-loadings matrix (p,nlv); (`R`): The PLS projection matrix (p,nlv); (`W`): The X-loading weights matrix (p,nlv); (`C`): The Y-loading weights matrix; (`TT`): the X-score normalization factor; (`xmeans`): the centering vector of X (p,1); (`ymeans`): the centering vector of Y (q,1); (`weights`): the weights vector of X-variables (p,1); (`U`): intermediate output.
`K`	kernel Gram matrix
`kern`	kernel function
`dots`	Optional arguments passed in the kernel function

For `transform.Dkplsr` : A matrix (m, nlv) with the projection of the new X-data on the X-scores

For `predict.Dkplsr`:

`pred`	A list of matrices (m, q) with the Y predicted values for the new X-data
`K`	kernel Gram matrix (m, nlv), with values for the new X-data

For `coef.Dkplsr`:

`int`	matrix (1,nlv) with the intercepts
`B`	matrix (n,nlv) with the coefficients

Note

The second example concerns the fitting of the function sinc(x) described in Rosipal & Trejo 2001 p. 105-106

References

Bennett, K.P., Embrechts, M.J., 2003. An optimization perspective on kernel partial least squares regression, in: Advances in Learning Theory: Methods, Models and Applications, NATO Science Series III: Computer & Systems Sciences. IOS Press Amsterdam, pp. 227-250.

Rosipal, R., Trejo, L.J., 2001. Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Space. Journal of Machine Learning Research 2, 97-123.

Examples

```
## EXAMPLE 1

n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
```

```
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

nlv <- 2
fm <- dkplsr(Xtrain, Ytrain, nlv = nlv, kern = "krbf", gamma = .8)
transform(fm, Xtest)
transform(fm, Xtest, nlv = 1)
coef(fm)
coef(fm, nlv = 1)

predict(fm, Xtest)
predict(fm, Xtest, nlv = 0:nlv)$pred

pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)

nlv <- 2
fm <- dkplsr(Xtrain, Ytrain, nlv = nlv, kern = "kpol", degree = 2, coef0 = 10)
predict(fm, Xtest, nlv = nlv)

## EXAMPLE 2

x <- seq(-10, 10, by = .2)
x[x == 0] <- 1e-5
n <- length(x)
zy <- sin(abs(x)) / abs(x)
y <- zy + rnorm(n, 0, .2)
plot(x, y, type = "p")
lines(x, zy, lty = 2)
X <- matrix(x, ncol = 1)

nlv <- 3
fm <- dkplsr(X, y, nlv = nlv)
pred <- predict(fm, X)$pred
plot(X, y, type = "p")
lines(X, zy, lty = 2)
lines(X, pred, col = "red")
```

---

dkrr *Direct KRR Models*

Description

Direct kernel ridge regression (DKRR), following the same approcah as for DKPLSR (Bennett & Embrechts 2003). The method builds kernel Gram matrices and then runs a RR algorithm on them. This is not equivalent to the "true" KRR (= LS-SVM) algorithm.

Usage

```
dkrr(X, Y, weights = NULL, lb = 1e-2, kern = "krbf", ...)
```

```
## S3 method for class 'Dkrr'
coef(object, ..., lb = NULL)

## S3 method for class 'Dkrr'
predict(object, X, ..., lb = NULL)
```

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
Y	Training Y-data (n, q).
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
lb	A value of regularization parameter $lambda$.
kern	Name of the function defining the considered kernel for building the Gram matrix. See krbf for syntax, and other available kernel functions.
...	Optional arguments to pass in the kernel function defined in kern (e.g. gamma for krbf).
object	For the auxiliary functions: A fitted model, output of a call to the main function.

Value

For dkrr:

X	Matrix with the training X-data (n, p).
fm	List with the outputs of the RR ((V): eigenvector matrix of the correlation matrix (n,n); (TtDY): intermediate output; (sv): singular values of the matrix (1,n); (lb): value of regularization parameter $lambda$; (xmeans): the centering vector of X (p,1); (ymeans): the centering vector of Y (q,1); (weights): the weights vector of X-variables (p,1)
K	kernel Gram matrix
kern	kernel function
dots	Optional arguments passed in the kernel function

For predict.Dkrr:

pred	A list of matrices (m, q) with the Y predicted values for the new X-data
K	kernel Gram matrix (m, nlv), with values for the new X-data

For coef.Dkrr:

int	matrix (1,nlv) with the intercepts
B	matrix (n,nlv) with the coefficients
df	model complexity (number of degrees of freedom)

Note

The second example concerns the fitting of the function sinc(x) described in Rosipal & Trejo 2001 p. 105-106

References

Bennett, K.P., Embrechts, M.J., 2003. An optimization perspective on kernel partial least squares regression, in: Advances in Learning Theory: Methods, Models and Applications, NATO Science Series III: Computer & Systems Sciences. IOS Press Amsterdam, pp. 227-250.

Rosipal, R., Trejo, L.J., 2001. Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Space. Journal of Machine Learning Research 2, 97-123.

Examples

```
## EXAMPLE 1

n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

lb <- 2
fm <- dkrr(Xtrain, Ytrain, lb = lb, kern = "krbf", gamma = .8)
coef(fm)
coef(fm, lb = .6)
predict(fm, Xtest)
predict(fm, Xtest, lb = c(0.1, .8))

pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)

lb <- 2
fm <- dkrr(Xtrain, Ytrain, lb = lb, kern = "kpol", degree = 2, coef0 = 10)
predict(fm, Xtest)

## EXAMPLE 1

x <- seq(-10, 10, by = .2)
x[x == 0] <- 1e-5
n <- length(x)
zy <- sin(abs(x)) / abs(x)
y <- zy + rnorm(n, 0, .2)
plot(x, y, type = "p")
lines(x, zy, lty = 2)
X <- matrix(x, ncol = 1)

fm <- dkrr(X, y, lb = .01, gamma = .5)
pred <- predict(fm, X)$pred
```

```
plot(X, y, type = "p")
lines(X, zy, lty = 2)
lines(X, pred, col = "red")
```

---

dmnorm *Multivariate normal probability density*

Description

Prediction of the normal probability density of multivariate observations.

Usage

```
dmnorm(X = NULL, mu = NULL, sigma = NULL)

## S3 method for class 'Dmnorm'
predict(object, X, ...)
```

Arguments

X	For the main function: Training data (n, p) used for estimating the mean and the covariance matrix population (if mu or/and sigma are not provided). — For the auxiliary functions: New data (m, p) for which the density has to be predicted.
mu	The mean $(p, 1)$ of the normal distribution. If NULL (default), mu is estimated by the column-wise mean of the training data.
sigma	The covariance matrix (pxp) of the normal distribution. If NULL (default), $sigma$ is estimated by the empirical covariance matrix (denominator $n-1$) of the training data.
object	For the auxiliary functions: A result of a call to dmnorm.
...	For the auxiliary functions: Optional arguments.

Value

For dmnorm:

mu	means of the X variables
Uinv	inverse of the Cholesky decomposition of the covariance matrix
det	squared determinant of the Cholesky decomposition of the covariance matrix

For predict:

pred	Prediction of the normal probability density of new multivariate observations

Examples

```
data(iris)

X <- iris[, 1:2]

Xtrain <- X[1:40, ]
Xtest <- X[40:50, ]

fm <- dmnorm(Xtrain)
fm

k <- 50
x1 <- seq(min(Xtrain[, 1]), max(Xtrain[, 1]), length.out = k)
x2 <- seq(min(Xtrain[, 2]), max(Xtrain[, 2]), length.out = k)
zX <- expand.grid(x1, x2)
pred <- predict(fm, zX)$pred
contour(x1, x2, matrix(pred, nrow = 50))

points(Xtest, col = "red", pch = 16)
```

---

dtagg *Summary statistics of data subsets*

Description

Faster alternative to `aggregate` to calculate a summary statistic over data subsets. `dtagg` uses function `data.table::data.table` of package `data.table`.

Usage

```
dtagg(formula, data, FUN = mean, ...)
```

Arguments

`formula`	A left and right-hand-sides formula defing the variable and the aggregation levels on which is calculated the statistic.
`data`	A dataframe.
`FUN`	Function defining the statistic to compute (default to `mean`).
`...`	Eventual additional arguments to pass through `FUN`.

Value

A dataframe, with the values of the agregation level(s) and the corresponding computed statistic value.

Examples

```
dat <- data.frame(matrix(rnorm(2 * 100), ncol = 2))
names(dat) <- c("y1", "y2")
dat$typ1 <- sample(1:2, size = nrow(dat), TRUE)
dat$typ2 <- sample(1:3, size = nrow(dat), TRUE)

headm(dat)

dtagg(y1 ~ 1, data = dat)

dtagg(y1 ~ typ1 + typ2, data = dat)

dtagg(y1 ~ typ1 + typ2, data = dat, trim = .2)
```

---

dummy *Table of dummy variables*

Description

The function builds a table of dummy variables from a qualitative variable. A binary (i.e. 0/1) variable is created for each level of the qualitative variable.

Usage

```
dummy(y)
```

Arguments

y	A qualitative variable.

Value

Y	A matrix of dummy variables (i.e. binary variables), each representing a given level of the qualitative variable.
lev	levels of the qualitative variable.
ni	number of observations per level of the qualitative variable.

Examples

```
y <- c(1, 1, 3, 2, 3)
dummy(y)

y <- c("B", "a", "B")
dummy(y)
dummy(as.factor(y))
```

eposvd *External parameter orthogonalization (EPO)*

Description

Pre-processing a X-dataset by external parameter orthogonalization (EPO; Roger et al 2003). The objective is to remove from a dataset X (n, p) some "detrimental" information (e.g. humidity effect) represented by a dataset $D(m, p)$.

EPO consists in orthogonalizing the row observations of X to the detrimental sub-space defined by the first nlv non-centered PCA loadings vectors of D.

Function `eposvd` uses a SVD factorization of D and returns $M(p, p)$ the orthogonalization matrix, and P the considered loading vectors of D.

Usage

```
eposvd(D, nlv)
```

Arguments

`D`	A dataset (m, p) containing detrimental information.
`nlv`	The number of first loadings vectors of D considered for the orthogonalization.

Details

The data corrected from the detrimental information D can be computed by $Xcorrected = X * M$. Rows of the corrected matrix Xcorr are orthogonal to the loadings vectors (columns of P): $Xcorr * P$.

Value

`M`	orthogonalization matrix.
`P`	detrimental directions matrix (p, nlv) (loadings of D = columns of P).

References

Roger, J.-M., Chauchard, F., Bellon-Maurel, V., 2003. EPO-PLS external parameter orthogonalisation of PLS application to temperature-independent measurement of sugar content of intact fruits. Chemometrics and Intelligent Laboratory Systems 66, 191-204. https://doi.org/10.1016/S0169-7439(03)00051-0

Roger, J.-M., Boulet, J.-C., 2018. A review of orthogonal projections for calibration. Journal of Chemometrics 32, e3045. https://doi.org/10.1002/cem.3045

Examples

```
n <- 4 ; p <- 8
X <- matrix(rnorm(n * p), ncol = p)
m <- 3
D <- matrix(rnorm(m * p), ncol = p)

nlv <- 2
res <- eposvd(D, nlv = nlv)
M <- res$M
P <- res$P
M
P
```

`euclsq` *Matrix of distances*

Description

—— Matrix (n, m) of distances between row observations of two datasets X (n, p) and Y (m, p)

- `euclsq`: Squared Euclidean distance

- `mahsq`: Squared Mahalanobis distance

—— Matrix $(n, 1)$ of distances between row observations of a dataset X (n, p) and a vector p (n)

- `euclsq_mu`: Squared Euclidean distance

- `mahsq_mu`: Squared Euclidean distance

Usage

```
euclsq(X, Y = NULL)

euclsq_mu(X, mu)

mahsq(X, Y = NULL, Uinv = NULL)

mahsq_mu(X, mu, Uinv = NULL)
```

Arguments

`X`	X-data (n, p).
`Y`	Data (m, p) compared to `X`. If `NULL` (default), `Y` is set equal to `X`.
`mu`	Vector (p) compared to `X`.
`Uinv`	For Mahalanobis distance. The inverse of a Choleski factorization matrix of the covariance matrix of `X`. If `NULL` (default), `Uinv` is calculated internally.

Value

A distance matrix.

Examples

```
n <- 5 ; p <- 3
X <- matrix(rnorm(n * p), ncol = p)

euclsq(X)
as.matrix(stats::dist(X)^2)
euclsq(X, X)

Y <- X[c(1, 3), ]
euclsq(X, Y)
euclsq_mu(X, Y[2, ])

i <- 3
euclsq(X, X[i, , drop = FALSE])
euclsq_mu(X, X[i, ])

S <- cov(X) * (n - 1) / n
i <- 3
mahsq(X)[i, , drop = FALSE]
stats::mahalanobis(X, X[i, ], S)

mahsq(X)
Y <- X[c(1, 3), ]
mahsq(X, Y)
```

---

fda *Factorial discriminant analysis*

Description

Factorial discriminant analysis (FDA). The functions maximize the compromise $p'Bp/p'Wp$, i.e. $maxp'Bp$ with constraint $p'Wp = 1$. Vectors p are the linear discrimant coefficients "LD".

- fda: Eigen factorization of $W^(-1)B$

- fdasvd: Weighted SVD factorization of the matrix of the class centers.

If W is singular, W^(-1) is replaced by a MP pseudo-inverse.

Usage

```
fda(X, y, nlv = NULL)

fdasvd(X, y, nlv = NULL)

## S3 method for class 'Fda'
```

```
transform(object, X, ..., nlv = NULL)

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

Arguments

X	For the main functions: Training X-data (n, p).— For the auxiliary functions: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
nlv	For the main functions: The number(s) of LVs to calculate. — For the auxiliary functions: The number(s) of LVs to consider.
object	For the auxiliary functions: A fitted model, output of a call to the main function.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For fda and fdasvd:

T	X-scores matrix (n,nlv).
P	X-loadings matrix (p,nlv) = coefficients of the linear discriminant function = "LD" of function lda of package MASS.
Tcenters	projection of the class centers in the score space.
eig	vector of eigen values
sstot	total variance
W	unbiased within covariance matrix
xmeans	means of the X variables
lev	y levels
ni	number of observations per level of the y variable

For transform.Fda: scores of the new X-data in the model.

For summary.Fda:

explvar	Explained variance by PCA of the class centers in transformed scale.

References

Saporta G., 2011. Probabilités analyse des données et statistique. Editions Technip, Paris, France.

Examples

```
data(iris)

X <- iris[, 1:4]
y <- iris[, 5]
table(y)

fm <- fda(X, y)
headm(fm$T)

transform(fm, X[1:3, ])

summary(fm)
plotxy(fm$T, group = y, ellipse = TRUE,
    zeroes = TRUE, pch = 16, cex = 1.5, ncol = 2)
points(fm$Tcenters, pch = 8, col = "blue", cex = 1.5)
```

`forages` *forages*

Description

A NIRS dataset (pre-processed absorbance) describing the class membership of forages. Spectra were recorded from 1100 to 2498 nm at 2 nm intervals.

Usage

```
data(forages)
```

Format

A list with 4 components: `Xtrain`, `ytrain`, `Xtest`, `ytest`.

For the reference (calibration) data:

`Xtrain` A matrix whose rows are the pre-processed NIR absorbance spectra (= log10(1 / Reflectance)).

`ytrain` A vector of the response variable (class membership).

For the test data:

`Xtest` A matrix whose rows are the pre-processed NIR absorbance spectra (= log10(1 / Reflectance)).

`ytest` A vector of the response variable (class membership).

Examples

```
data(forages)
str(forages)
```

getknn *KNN selection*

Description

Function getknn selects the k nearest neighbours of each row observation of a new data set (= query) within a training data set, based on a dissimilarity measure.

getknn uses function get.knnx of package FNN (Beygelzimer et al.) available on CRAN.

Usage

```
getknn(Xtrain, X, k = NULL, diss = c("eucl", "mahal"),
  algorithm = "brute", list = TRUE)
```

Arguments

Xtrain	Training X-data (n, p).
X	New X-data (m, p) to consider.
k	The number of nearest neighbors to select in Xtrain for each observation of X.
diss	The type of dissimilarity used. Possible values are "eucl" (default; Euclidean distance) or "mahal" (Mahalanobis distance).
algorithm	Search algorithm used for Euclidean and Mahalanobis distances. Default to "brute". See get.knnx.
list	If TRUE (default), a list format is also returned for the outputs.

Value

A list of outputs, such as:

nn	A dataframe (mxk) with the indexes of the neighbors.
d	A dataframe (mxk) with the dissimilarities between the neighbors and the new observations.
listnn	Same as $nn but in a list format.
listd	Same as $d but in a list format.

Examples

```
n <- 10
p <- 4
X <- matrix(rnorm(n * p), ncol = p)
Xtrain <- X
Xtest <- X[c(1, 3), ]
m <- nrow(Xtest)
```

```
k <- 3
getknn(Xtrain, Xtest, k = k)

fm <- pcasvd(Xtrain, nlv = 2)
Ttrain <- fm$T
Ttest <- transform(fm, Xtest)
getknn(Ttrain, Ttest, k = k, diss = "mahal")
```

`gridcv` *Cross-validation*

Description

Functions for cross-validating predictive models.

The functions return "scores" (average error rates) of predictions for a given model and a grid of parameter values, calculated from a cross-validation process.

- `gridcv`: Can be used for any model.

- `gridcvlv`: Specific to models using regularization by latent variables (LVs) (e.g. PLSR). Much faster than `gridcv`.

- `gridcvlb`: Specific to models using ridge regularization (e.g. RR). Much faster than `gridcv`.

Usage

```
gridcv(X, Y, segm, score, fun, pars, verb = TRUE)

gridcvlv(X, Y, segm, score, fun, nlv, pars = NULL, verb = TRUE)

gridcvlb(X, Y, segm, score, fun, lb, pars = NULL, verb = TRUE)
```

Arguments

`X`	Training X-data (n, p), or list of training X-data.
`Y`	Training Y-data (n, q).
`segm`	CV segments, typically output of `segmkf` or `segmts`.
`score`	A function calculating a prediction score (e.g. `msep`).
`fun`	A function corresponding to the predictive model.
`nlv`	For `gridcvlv`. A vector of numbers of LVs.
`lb`	For `gridcvlb`. A vector of ridge regulariation parameters.
`pars`	A list of named vectors. Each vector must correspond to an argument of the model function and gives the parameter values to consider for this argument. (see details)
`verb`	Logical. If `TRUE`, fitting information are printed.

Details

Argument `pars` (the grid) must be a list of named vectors, each vector corresponding to an argument of the model function and giving the parameter values to consider for this argument. This list can eventually be built with function `mpars`, which returns all the combinations of the input parameters, see the examples.

For `gridcvlv`, `pars` must not contain `nlv` (nb. LVs), and for `gridcvlb`, `lb` (regularization parameter $lambda$).

Value

Dataframes with the prediction scores for the grid.

Note

Examples are given: - with PLSR, using gridcv and gridcvlv (much faster) - with PLSLDA, using gridcv and gridcvlv (much faster) - with RR, using gridcv and gridcvlb (much faster) - with KRR, using gridcv and gridcvlb (much faster) - with LWPLSR, using gridcvlv

Examples

```
## EXAMPLE WITH PLSR

n <- 50 ; p <- 8
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)
Y <- cbind(y, 10 * rnorm(n))

K = 3
segm <- segmkf(n = n, K = K, nrep = 1)
segm

nlv <- 5
pars <- mpars(nlv = 1:nlv)
pars
gridcv(
    X, Y, segm,
    score = msep,
    fun = plskern,
    pars = pars, verb = TRUE)

gridcvlv(
    X, Y, segm,
    score = msep,
    fun = plskern,
    nlv = 0:nlv, verb = TRUE)

## EXAMPLE WITH PLSLDA

n <- 50 ; p <- 8
X <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
y <- sample(c(1, 4, 10), size = n, replace = TRUE)
```

```
K = 3
segm <- segmkf(n = n, K = K, nrep = 1)
segm

nlv <- 5
pars <- mpars(nlv = 1:nlv, prior = c("unif", "prop"))
pars
gridcv(
    X, y, segm,
    score = err,
    fun = plslda,
    pars = pars, verb = TRUE)

pars <- mpars(prior = c("unif", "prop"))
pars
gridcvlv(
    X, y, segm,
    score = err,
    fun = plslda,
    nlv = 1:nlv, pars = pars, verb = TRUE)

## EXAMPLE WITH RR

n <- 50 ; p <- 8
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)
Y <- cbind(y, 10 * rnorm(n))

K = 3
segm <- segmkf(n = n, K = K, nrep = 1)
segm

lb <- c(.1, 1)
pars <- mpars(lb = lb)
pars
gridcv(
    X, Y, segm,
    score = msep,
    fun = rr,
    pars = pars, verb = TRUE)

gridcvlb(
    X, Y, segm,
    score = msep,
    fun = rr,
    lb = lb, verb = TRUE)

## EXAMPLE WITH KRR

n <- 50 ; p <- 8
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)
```

```
Y <- cbind(y, 10 * rnorm(n))

K = 3
segm <- segmkf(n = n, K = K, nrep = 1)
segm

lb <- c(.1, 1)
gamma <- 10^(-1:1)
pars <- mpars(lb = lb, gamma = gamma)
pars
gridcv(
    X, Y, segm,
    score = msep,
    fun = krr,
    pars = pars, verb = TRUE)

pars <- mpars(gamma = gamma)
gridcvlb(
    X, Y, segm,
    score = msep,
    fun = krr,
    lb = lb, pars = pars, verb = TRUE)

## EXAMPLE WITH LWPLSR

n <- 50 ; p <- 8
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)
Y <- cbind(y, 10 * rnorm(n))

K = 3
segm <- segmkf(n = n, K = K, nrep = 1)
segm

nlvdis <- 5
h <- c(1, Inf)
k <- c(10, 20)
nlv <- 5
pars <- mpars(nlvdis = nlvdis, diss = "mahal",
              h = h, k = k)
pars
res <- gridcvlv(
    X, Y, segm,
    score = msep,
    fun = lwplsr,
    nlv = 0:nlv, pars = pars, verb = TRUE)
res
```

---

gridscore *Tuning of predictive models on a validation dataset*

Description

Functions for tuning predictive models on a validation set.

The functions return "scores" (average error rates) of predictions for a given model and a grid of parameter values, calculated on a validation dataset.

- `gridscore`: Can be used for any model.

- `gridscorelv`: Specific to models using regularization by latent variables (LVs) (e.g. PLSR). Much faster than `gridscore`.

- `gridscorelb`: Specific to models using ridge regularization (e.g. RR). Much faster than `gridscore`.

Usage

```
gridscore(Xtrain, Ytrain, X, Y, score, fun, pars, verb = FALSE)

gridscorelv(Xtrain, Ytrain, X, Y, score, fun, nlv, pars = NULL, verb = FALSE)

gridscorelb(Xtrain, Ytrain, X, Y, score, fun, lb, pars = NULL, verb = FALSE)
```

Arguments

`Xtrain`	Training X-data (n, p).
`Ytrain`	Training Y-data (n, q).
`X`	Validation X-data (n, p).
`Y`	Validation Y-data (n, q).
`score`	A function calculating a prediction score (e.g. `msep`).
`fun`	A function corresponding to the predictive model.
`nlv`	For `gridscorelv`. A vector of numbers of LVs.
`lb`	For `gridscorelb`. A vector of ridge regulariation parameters.
`pars`	A list of named vectors. Each vector must correspond to an argument of the model function and gives the parameter values to consider for this argument. (see details)
`verb`	Logical. If `TRUE`, fitting information are printed.

Details

Argument `pars` (the grid) must be a list of named vectors, each vector corresponding to an argument of the model function and giving the parameter values to consider for this argument. This list can eventually be built with function `mpars`, which returns all the combinations of the input parameters, see the examples.

For `gridscorelv`, `pars` must not contain `nlv` (nb. LVs), and for `gridscorelb`, `lb` (regularization parameter $lambda$).

Value

A dataframe with the prediction scores for the grid.

Note

Examples are given: - with PLSR, using gridscore and gridscorelv (much faster) - with PLSLDA, using gridscore and gridscorelv (much faster) - with RR, using gridscore and gridscorelb (much faster) - with KRR, using gridscore and gridscorelb (much faster) - with LWPLSR, using gridscorelv

Examples

```
## EXAMPLE WITH PLSR

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 10 * rnorm(n))
m <- 3
Xtest <- Xtrain[1:m, ]
Ytest <- Ytrain[1:m, ] ; ytest <- Ytest[, 1]

nlv <- 5
pars <- mpars(nlv = 1:nlv)
pars
gridscore(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = plskern,
    pars = pars, verb = TRUE
    )

gridscorelv(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = plskern,
    nlv = 0:nlv, verb = TRUE
    )

fm <- plskern(Xtrain, Ytrain, nlv = nlv)
pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)

## EXAMPLE WITH PLSLDA

n <- 50 ; p <- 8
X <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
y <- sample(c(1, 4, 10), size = n, replace = TRUE)
Xtrain <- X ; ytrain <- y
m <- 5
Xtest <- X[1:m, ] ; ytest <- y[1:m]

nlv <- 5
pars <- mpars(nlv = 1:nlv, prior = c("unif", "prop"))
pars
gridscore(
    Xtrain, ytrain, Xtest, ytest,
```

```
    score = err,
    fun = plslda,
    pars = pars, verb = TRUE
    )

fm <- plslda(Xtrain, ytrain, nlv = nlv)
pred <- predict(fm, Xtest)$pred
err(pred, ytest)

pars <- mpars(prior = c("unif", "prop"))
pars
gridscorelv(
    Xtrain, ytrain, Xtest, ytest,
    score = err,
    fun = plslda,
    nlv = 1:nlv, pars = pars, verb = TRUE
    )

## EXAMPLE WITH RR

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 10 * rnorm(n))
m <- 3
Xtest <- Xtrain[1:m, ]
Ytest <- Ytrain[1:m, ] ; ytest <- Ytest[, 1]

lb <- c(.1, 1)
pars <- mpars(lb = lb)
pars
gridscore(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = rr,
    pars = pars, verb = TRUE
    )

gridscorelb(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = rr,
    lb = lb, verb = TRUE
    )

## EXAMPLE WITH KRR

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 10 * rnorm(n))
m <- 3
Xtest <- Xtrain[1:m, ]
```

```
Ytest <- Ytrain[1:m, ] ; ytest <- Ytest[, 1]

lb <- c(.1, 1)
gamma <- 10^(-1:1)
pars <- mpars(lb = lb, gamma = gamma)
pars
gridscore(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = krr,
    pars = pars, verb = TRUE
    )

pars <- mpars(gamma = gamma)
gridscorelb(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = krr,
    lb = lb, pars = pars, verb = TRUE
    )

## EXAMPLE WITH LWPLSR

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 10 * rnorm(n))
m <- 3
Xtest <- Xtrain[1:m, ]
Ytest <- Ytrain[1:m, ] ; ytest <- Ytest[, 1]

nlvdis <- 5
h <- c(1, Inf)
k <- c(10, 20)
nlv <- 5
pars <- mpars(nlvdis = nlvdis, diss = "mahal",
    h = h, k = k)
pars
res <- gridscorelv(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = lwplsr,
    nlv = 0:nlv, pars = pars, verb = TRUE
    )
res
```

---

headm *Display of the first part of a data set*

Description

Function headm displays the first part and the dimension of a data set.

Usage

```
headm(X)
```

Arguments

X	A matrix or dataframe.

Value

first 6 rows and columns of a dataset, number of rows, number of columns, dataset class.

Examples

```
n <- 1000
p <- 200
X <- matrix(rnorm(n * p), nrow = n)

headm(X)
```

---

interpl *Resampling of spectra by interpolation methods*

Description

Resampling of signals by interpolation methods, including linear, spline, and cubic interpolation.

The function uses function interp1 of package signal available on the CRAN.

Usage

```
interpl(X, w, meth = "cubic", ...)
```

Arguments

X	X-data (nxp). For the interpolation, the column names of X are taken as numeric values, $w0$. If they are not numeric or missing, they are automatically set to w0 = 1:p.
w	A vector of the values where to interpolate (typically within the range of $w0$).
meth	The method of interpolation. See interp1.
...	Optional arguments to pass in function splinefun if meth = "spline".

Value

A matrix of the interpolated signals.

Examples

```
data(cassav)

X <- cassav$Xtest
headm(X)

w <- seq(500, 2400, length = 10)
zX <- interpl(X, w, meth = "spline")
headm(zX)
plotsp(zX)
```

---

knnda *KNN-DA*

Description

KNN weighted discrimination. For each new observation to predict, a number of k nearest neighbors is selected and the prediction is calculated by the most frequent class in y in this neighborhood.

Usage

```
knnda(X, y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k)

## S3 method for class 'Knnda'
predict(object, X, ...)
```

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
nlvdis	The number of LVs to consider in the global PLS used for the dimension reduction before calculating the dissimilarities. If `nlvdis = 0`, there is no dimension reduction. (see details)
diss	The type of dissimilarity used for defining the neighbors. Possible values are "eucl" (default; Euclidean distance), "mahal" (Mahalanobis distance), or "correlation". Correlation dissimilarities are calculated by sqrt(.5 * (1 - rho)).

h	A scale scalar defining the shape of the weight function. Lower is h, sharper is the function. See wdist.
k	The number of nearest neighbors to select for each observation to predict.
object	For the auxiliary functions: A fitted model, output of a call to the main function.
...	For the auxiliary functions: Optional arguments. Not used.

Details

In function knnda, the dissimilarities used for computing the neighborhood and the weights can be calculated from the original X-data or after a dimension reduction (argument nlvdis). In the last case, global PLS scores are computed from (X, Y) and the dissimilarities are calculated on these scores. For high dimension X-data, the dimension reduction is in general required for using the Mahalanobis distance.

Value

For knnda:list with input arguments.

For predict.Knnda:

pred	prediction calculated for each observation by the most frequent class in y in its neighborhood.
listnn	list with the neighbors used for each observation to be predicted
listd	list with the distances to the neighbors used for each observation to be predicted
listw	list with the weights attributed to the neighbors used for each observation to be predicted

References

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]

nlvdis <- 5 ; diss <- "mahal"
h <- 2 ; k <- 10
fm <- knnda(
    Xtrain, ytrain,
    nlvdis = nlvdis, diss = diss,
    h = h, k = k
    )
res <- predict(fm, Xtest)
names(res)
res$pred
```

```
err(res$pred, ytest)
```

---

knnr *KNN-R*

Description

KNN weighted regression. For each new observation to predict, a number of k nearest neighbors is selected and the prediction is calculated by the average (eventually weighted) of the response Y over this neighborhood.

Usage

```
knnr(X, Y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k)

## S3 method for class 'Knnr'
predict(object, X, ...)
```

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
Y	Training Y-data (n, q).
nlvdis	The number of LVs to consider in the global PLS used for the dimension reduction before calculating the dissimilarities. If `nlvdis = 0`, there is no dimension reduction. (see details)
diss	The type of dissimilarity used for defining the neighbors. Possible values are "eucl" (default; Euclidean distance), "mahal" (Mahalanobis distance), or "correlation". Correlation dissimilarities are calculated by sqrt(.5 * (1 - rho)).
h	A scale scalar defining the shape of the weight function. Lower is h, sharper is the function. See `wdist`.
k	The number of nearest neighbors to select for each observation to predict.
object	— For the auxiliary functions: A fitted model, output of a call to the main function.
...	— For the auxiliary functions: Optional arguments. Not used.

Details

In function `knnr`, the dissimilarities used for computing the neighborhood and the weights can be calculated from the original X-data or after a dimension reduction (argument `nlvdis`). In the last case, global PLS scores are computed from (X, Y) and the dissimilarities are calculated on these scores. For high dimension X-data, the dimension reduction is in general required for using the Mahalanobis distance.

Value

For knnr:list with input arguments.

For predict.Knnr:

pred	prediction calculated for each observation by the average (eventually weighted) of the response Y over its neighborhood.
listnn	list with the neighbors used for each observation to be predicted
listd	list with the distances to the neighbors used for each observation to be predicted
listw	list with the weights attributed to the neighbors used for each observation to be predicted

References

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

Examples

```
n <- 30 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 100 * ytrain)
m <- 4
Xtest <- matrix(rnorm(m * p), ncol = p)
ytest <- rnorm(m)
Ytest <- cbind(ytest, 10 * ytest)

nlvdis <- 5 ; diss <- "mahal"
h <- 2 ; k <- 10
fm <- knnr(
    Xtrain, Ytrain,
    nlvdis = nlvdis, diss = diss,
    h = h, k = k)
res <- predict(fm, Xtest)
names(res)
res$pred
msep(res$pred, Ytest)
```

---

kpca *KPCA*

Description

Kernel PCA (Scholkopf et al. 1997, Scholkopf & Smola 2002, Tipping 2001) by SVD factorization of the weighted Gram matrix $D^(1/2) * Phi(X) * Phi(X)' * D^(1/2)$. D is a (n, n) diagonal matrix of weights for the observations (rows of X).

Usage

```
kpca(X, weights = NULL, nlv, kern = "krbf", ...)

## S3 method for class 'Kpca'
transform(object, X, ..., nlv = NULL)

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

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	The number of PCs to calculate.
kern	Name of the function defining the considered kernel for building the Gram matrix. See krbf for syntax, and other available kernel functions.
...	Optional arguments to pass in the kernel function defined in kern (e.g. gamma for krbf).
object	— For the auxiliary functions: A fitted model, output of a call to the main functions.

Value

For kpca:

X	Training X-data (n, p).
Kt	Gram matrix
T	X-scores matrix.
P	X-loadings matrix.
sv	vector of singular values
eig	vector of eigenvalues.
weights	vector of observation weights.
kern	kern function.
dots	Optional arguments.

For transform.Kpca: X-scores matrix for new X-data.

For summary.Kpca:

explvar	explained variance matrix.

References

Scholkopf, B., Smola, A., Muller, K.-R., 1997. Kernel principal component analysis, in: Gerstner, W., Germond, A., Hasler, M., Nicoud, J.-D. (Eds.), Artificial Neural Networks - ICANN 97, Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 583-588. https://doi.org/10.1007/BFb0020217

Scholkopf, B., Smola, A.J., 2002. Learning with kernels: support vector machines, regularization, optimization, and beyond, Adaptive computation and machine learning. MIT Press, Cambridge, Mass.

Tipping, M.E., 2001. Sparse kernel principal component analysis. Advances in neural information processing systems, MIT Press. http://papers.nips.cc/paper/1791-sparse-kernel-principal-component-analysis.pdf

Examples

```
## EXAMPLE 1

n <- 5 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)

nlv <- 3
kpca(X, nlv = nlv, kern = "krbf")

fm <- kpca(X, nlv = nlv, kern = "krbf", gamma = .6)
fm$T
transform(fm, X[1:2, ])
transform(fm, X[1:2, ], nlv = 1)
summary(fm)

## EXAMPLE 2

n <- 5 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
nlv <- 3
pcasvd(X, nlv = nlv)$T
kpca(X, nlv = nlv, kern = "kpol")$T
```

---

kplsr *KPLSR Models*

Description

NIPALS Kernel PLSR algorithm described in Rosipal & Trejo (2001).

The algorithm is slow for $n >= 500$.

Usage

```
kplsr(X, Y, weights = NULL, nlv, kern = "krbf",
     tol = .Machine$double.eps^0.5, maxit = 100, ...)

## S3 method for class 'Kplsr'
transform(object, X, ..., nlv = NULL)

## S3 method for class 'Kplsr'
coef(object, ..., nlv = NULL)

## S3 method for class 'Kplsr'
predict(object, X, ..., nlv = NULL)
```

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
Y	Training Y-data (n, q).
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	The number(s) of LVs to calculate. — For the auxiliary functions: The number(s) of LVs to consider.
kern	Name of the function defining the considered kernel for building the Gram matrix. See krbf for syntax, and other available kernel functions.
tol	Tolerance level for stopping the NIPALS iterations.
maxit	Maximum number of NIPALS iterations.
...	Optional arguments to pass in the kernel function defined in kern (e.g. gamma for krbf).
object	For the auxiliary functions: A fitted model, output of a call to the main function.

Value

For kplsr:

X	Training X-data (n, p).
Kt	Gram matrix
T	X-scores matrix.
C	The Y-loading weights matrix.
U	intermediate output.
R	The PLS projection matrix (p,nlv).
ymeans	the centering vector of Y (q,1).
weights	vector of observation weights.
kern	kern function.

dots Optional arguments.

For transform.Kplsr: X-scores matrix for new X-data.

For coef.Kplsr:

int intercept values matrix.

beta beta coefficient matrix.

For predict.Kplsr:

pred predicted values matrix for new X-data.

Note

The second example concerns the fitting of the function sinc(x) described in Rosipal & Trejo 2001 p. 105-106

References

Rosipal, R., Trejo, L.J., 2001. Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Space. Journal of Machine Learning Research 2, 97-123.

Examples

```
## EXAMPLE 1

n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

nlv <- 2
fm <- kplsr(Xtrain, Ytrain, nlv = nlv, kern = "krbf", gamma = .8)
transform(fm, Xtest)
transform(fm, Xtest, nlv = 1)
coef(fm)
coef(fm, nlv = 1)

predict(fm, Xtest)
predict(fm, Xtest, nlv = 0:nlv)$pred

pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)

nlv <- 2
fm <- kplsr(Xtrain, Ytrain, nlv = nlv, kern = "kpol", degree = 2, coef0 = 10)
predict(fm, Xtest, nlv = nlv)

## EXAMPLE 2
```

```
x <- seq(-10, 10, by = .2)
x[x == 0] <- 1e-5
n <- length(x)
zy <- sin(abs(x)) / abs(x)
y <- zy + rnorm(n, 0, .2)
plot(x, y, type = "p")
lines(x, zy, lty = 2)
X <- matrix(x, ncol = 1)

nlv <- 2
fm <- kplsr(X, y, nlv = nlv)
pred <- predict(fm, X)$pred
plot(X, y, type = "p")
lines(X, zy, lty = 2)
lines(X, pred, col = "red")
```

---

kplsrda *KPLSR-DA models*

Description

Discrimination (DA) based on kernel PLSR (KPLSR)

Usage

```
kplsrda(X, y, weights = NULL, nlv, kern = "krbf", ...)

## S3 method for class 'Kplsrda'
predict(object, X, ..., nlv = NULL)
```

Arguments

X	For main function: Training X-data (n, p). — For auxiliary function: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
weights	Weights (n) to apply to the training observations for the PLS2. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	For main function: The number(s) of LVs to calculate. — For auxiliary function: The number(s) of LVs to consider.
kern	Name of the function defining the considered kernel for building the Gram matrix. See krbf for syntax, and other available kernel functions.
...	Optional arguments to pass in the kernel function defined in kern (e.g. gamma for krbf).
object	For auxiliary function: A fitted model, output of a call to the main functions.

Details

The training variable y (univariate class membership) is transformed to a dummy table containing $nclas$ columns, where $nclas$ is the number of classes present in y. Each column is a dummy variable (0/1). Then, a kernel PLSR (KPLSR) is run on the $X-$data and the dummy table, returning predictions of the dummy variables. For a given observation, the final prediction is the class corresponding to the dummy variable for which the prediction is the highest.

Value

For `kplsrda`:

`fm`	list with the kplsrda model: (`X`): the training X-data (n, p); (`Kt`): the Gram matrix; (`T`): X-scores matrix; (`C`): The Y-loading weights matrix; (`U`): intermediate output; (`R`): The PLS projection matrix (p,nlv); (`ymeans`): the centering vector of Y (q,1); (`weights`): vector of observation weights; (`kern`): kern function; (`dots`): Optional arguments.
`lev`	y levels
`ni`	number of observations by level of y

For `predict.Kplsrda`:

`pred`	predicted class for each observation
`posterior`	calculated probability of belonging to a class for each observation

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)
m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]

nlv <- 2
fm <- kplsrda(Xtrain, ytrain, nlv = nlv)
names(fm)
predict(fm, Xtest)

pred <- predict(fm, Xtest)$pred
err(pred, ytest)

predict(fm, Xtest, nlv = 0:nlv)$posterior
predict(fm, Xtest, nlv = 0)$posterior

predict(fm, Xtest, nlv = 0:nlv)$pred
predict(fm, Xtest, nlv = 0)$pred
```

krbf *Kernel functions*

Description

Building Gram matrices for different kernels (e.g. Scholkopf & Smola 2002).

- radial basis: exp(-gamma * |x - y|^2)

- polynomial: (gamma * x' * y + coef0)^degree

- sigmoid: tanh(gamma * x' * y + coef0)

Usage

```
krbf(X, Y = NULL, gamma = 1)

kpol(X, Y = NULL, degree = 1, gamma = 1, coef0 = 0)

ktanh(X, Y = NULL, gamma = 1, coef0 = 0)
```

Arguments

X	Dataset (n, p).
Y	Dataset (m, p). The resulting Gram matrix $K(X, Y)$ has dimensionnality (n, m). If NULL (default), Y is set equal to X.
gamma	value of the gamma parameter in the kernel calculation.
degree	For kpol: value of the degree parameter in the polynomial kernel calculation.
coef0	For kpol and ktanh: value of the coef0 parameter in the polynomial or sigmoid kernel calculation.

Value

Gram matrix

References

Scholkopf, B., Smola, A.J., 2002. Learning with kernels: support vector machines, regularization, optimization, and beyond, Adaptive computation and machine learning. MIT Press, Cambridge, Mass.

Examples

```
n <- 5 ; p <- 3
Xtrain <- matrix(rnorm(n * p), ncol = p)
Xtest <- Xtrain[1:2, , drop = FALSE]

gamma <- .8
```

```
krbf(Xtrain, gamma = gamma)

krbf(Xtest, Xtrain, gamma = gamma)
exp(-.5 * euclsq(Xtest, Xtrain) / gamma^2)

kpol(Xtrain, degree = 2, gamma = .5, coef0 = 1)
```

---

krr *KRR (LS-SVMR)*

Description

Kernel ridge regression models (KRR = LS-SVMR) (Suykens et al. 2000, Bennett & Embrechts 2003, Krell 2018).

Usage

```
krr(X, Y, weights = NULL, lb = 1e-2, kern = "krbf", ...)

## S3 method for class 'Krr'
coef(object, ..., lb = NULL)

## S3 method for class 'Krr'
predict(object, X, ..., lb = NULL)
```

Arguments

`X`	For main function: Training X-data (n, p). — For auxiliary function: New X-data (m, p) to consider.
`Y`	Training Y-data (n, q).
`weights`	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to `NULL` (weights are set to $1/n$).
`lb`	A value of regularization parameter $lambda$. If `lb = 0`, a pseudo-inverse is used in the RR.
`kern`	Name of the function defining the considered kernel for building the Gram matrix. See `krbf` for syntax, and other available kernel functions.
`...`	Optional arguments to pass in the kernel function defined in `kern` (e.g. `gamma` for `krbf`).
`object`	— For auxiliary function: A fitted model, output of a call to the main function.

Value

For `krr`:

`X`	Training X-data (n, p).
`K`	Gram matrix
`Kt`	Gram matrix
`U`	intermediate output.
`UtDY`	intermediate output.
`sv`	singular values of the matrix (1,n)
`lb`	value of regularization parameter $lambda$
`ymeans`	the centering vector of Y (q,1)
`weights`	the weights vector of X-variables (p,1)
`kern`	kern function.
`dots`	Optional arguments.

For `coef.Krr`:

`int`	matrix (1,nlv) with the intercepts
`alpha`	matrix (n,nlv) with the coefficients
`df`	model complexity (number of degrees of freedom)

For `predict.Krr`:

`pred`	A list of matrices (m, q) with the Y predicted values for the new X-data

Note

KRR is close to the particular SVMR setting the $epsilon$ coefficient to zero (no marges excluding observations). The difference is that a L2-norm optimization is done, instead L1 in SVM.

The second example concerns the fitting of the function sinc(x) described in Rosipal & Trejo 2001 p. 105-106

References

Bennett, K.P., Embrechts, M.J., 2003. An optimization perspective on kernel partial least squares regression, in: Advances in Learning Theory: Methods, Models and Applications, NATO Science Series III: Computer & Systems Sciences. IOS Press Amsterdam, pp. 227-250.

Cawley, G.C., Talbot, N.L.C., 2002. Reduced Rank Kernel Ridge Regression. Neural Processing Letters 16, 293-302. https://doi.org/10.1023/A:1021798002258

Krell, M.M., 2018. Generalizing, Decoding, and Optimizing Support Vector Machine Classification. arXiv:1801.04929.

Saunders, C., Gammerman, A., Vovk, V., 1998. Ridge Regression Learning Algorithm in Dual Variables, in: In Proceedings of the 15th International Conference on Machine Learning. Morgan Kaufmann, pp. 515-521.

Suykens, J.A.K., Lukas, L., Vandewalle, J., 2000. Sparse approximation using least squares support vector machines. 2000 IEEE International Symposium on Circuits and Systems. Emerging Technologies for the 21st Century. Proceedings (IEEE Cat No.00CH36353). https://doi.org/10.1109/ISCAS.2000.856439

Welling, M., n.d. Kernel ridge regression. Department of Computer Science, University of Toronto, Toronto, Canada. https://www.ics.uci.edu/~welling/classnotes/papers_class/Kernel-Ridge.pdf

Examples

```
## EXAMPLE 1

n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

lb <- 2
fm <- krr(Xtrain, Ytrain, lb = lb, kern = "krbf", gamma = .8)
coef(fm)
coef(fm, lb = .6)
predict(fm, Xtest)
predict(fm, Xtest, lb = c(0.1, .6))

pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)

lb <- 2
fm <- krr(Xtrain, Ytrain, lb = lb, kern = "kpol", degree = 2, coef0 = 10)
predict(fm, Xtest)

## EXAMPLE 2

x <- seq(-10, 10, by = .2)
x[x == 0] <- 1e-5
n <- length(x)
zy <- sin(abs(x)) / abs(x)
y <- zy + rnorm(n, 0, .2)
plot(x, y, type = "p")
lines(x, zy, lty = 2)
X <- matrix(x, ncol = 1)

fm <- krr(X, y, lb = .1, gamma = .5)
pred <- predict(fm, X)$pred
plot(X, y, type = "p")
lines(X, zy, lty = 2)
lines(X, pred, col = "red")
```

krrda *KRR-DA models*

Description

Discrimination (DA) based on kernel ridge regression (KRR).

Usage

```
krrda(X, y, weights = NULL, lb = 1e-5, kern = "krbf", ...)

## S3 method for class 'Krrda'
predict(object, X, ..., lb = NULL)
```

Arguments

X	For main function: Training X-data (n, p). — For auxiliary function: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
weights	Weights (n) to apply to the training observations for the PLS2. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
lb	A value of regularization parameter $lambda$. If lb = 0, a pseudo-inverse is used in the RR.
kern	Name of the function defining the considered kernel for building the Gram matrix. See krbf for syntax, and other available kernel functions.
...	Optional arguments to pass in the kernel function defined in kern (e.g. gamma for krbf).
object	— For auxiliary function: A fitted model, output of a call to the main functions.

Details

The training variable y (univariate class membership) is transformed to a dummy table containing $nclas$ columns, where $nclas$ is the number of classes present in y. Each column is a dummy variable (0/1). Then, a kernel ridge regression (KRR) is run on the $X-$data and the dummy table, returning predictions of the dummy variables. For a given observation, the final prediction is the class corresponding to the dummy variable for which the prediction is the highest.

Value

For krrda:

fm	List with the outputs of the RR ((X): Training X-data (n, p); (K): Gram matrix; (Kt): Gram matrix; (U): intermediate output; (UtDY): intermediate output;

	(sv): singular values of the matrix (1,n); (lb): value of regularization parameter $lambda$; (ymeans): the centering vector of Y (q,1); (weights): the weights vector of X-variables (p,1); (kern): kern function; (dots): Optional arguments.
lev	y levels
ni	number of observations by level of y

For predict.Krrda:

pred	matrix or list of matrices (if lb is a vector), with predicted class for each observation
posterior	matrix or list of matrices (if lb is a vector), calculated probability of belonging to a class for each observation

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]

lb <- 1
fm <- krrda(Xtrain, ytrain, lb = lb)
names(fm)
predict(fm, Xtest)

pred <- predict(fm, Xtest)$pred
err(pred, ytest)

predict(fm, Xtest, lb = 0:2)
predict(fm, Xtest, lb = 0)
```

---

lda *LDA and QDA*

Description

Probabilistic (parametric) linear and quadratic discriminant analysis.

Usage

```
lda(X, y, prior = c("unif", "prop"))

qda(X, y, prior = c("unif", "prop"))

## S3 method for class 'Lda'
```

```
predict(object, X, ...)
## S3 method for class 'Qda'
predict(object, X, ...)
```

Arguments

X	For the main functions: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
prior	The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Details

For each observation to predict, the posterior probability to belong to a given class is estimated using the Bayes' formula, assuming priors (proportional or uniform) and a multivariate Normal distribution for the dependent variables X. The prediction is the class with the highest posterior probability.

LDA assumes homogeneous $X-$covariance matrices for the classes while QDA assumes different covariance matrices. The functions use dmnorm for estimating the multivariate Normal densities.

Value

For lda and qda:

ct	centers (column-wise means) for classes of observations.
W	unbiased within covariance matrices for classes of observations.
wprior	prior probabilities of the classes.
lev	y levels.
ni	number of observations by level of y.

For predict.Lda and predict.Qda:

pred	predicted classes of observations.
ds	Prediction of the normal probability density.
posterior	posterior probabilities of the classes.

References

Saporta, G., 2011. Probabilités analyse des données et statistique. Editions Technip, Paris, France.

Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

Examples

```
## EXAMPLE 1

data(iris)

X <- iris[, 1:4]
y <- iris[, 5]
N <- nrow(X)

nTest <- round(.25 * N)
nTraining <- N - nTest
s <- sample(1:N, nTest)
Xtrain <- X[-s, ]
ytrain <- y[-s]
Xtest <- X[s, ]
ytest <- y[s]

prior <- "unif"

fm <- lda(Xtrain, ytrain, prior = prior)
res <- predict(fm, Xtest)
names(res)

headm(res$pred)
headm(res$ds)
headm(res$posterior)

err(res$pred, ytest)

## EXAMPLE 2

data(iris)

X <- iris[, 1:4]
y <- iris[, 5]
N <- nrow(X)

nTest <- round(.25 * N)
nTraining <- N - nTest
s <- sample(1:N, nTest)
Xtrain <- X[-s, ]
ytrain <- y[-s]
Xtest <- X[s, ]
ytest <- y[s]

prior <- "prop"

fm <- lda(Xtrain, ytrain, prior = prior)
res <- predict(fm, Xtest)
names(res)

headm(res$pred)
```

```
headm(res$ds)
headm(res$posterior)

err(res$pred, ytest)
```

`lmr` *Linear regression models*

Description

Linear regression models (uses function `lm`).

Usage

```
lmr(X, Y, weights = NULL)

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

## S3 method for class 'Lmr'
predict(object, X, ...)
```

Arguments

`X`	For the main function: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
`Y`	Training Y-data (n, q).
`weights`	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to `NULL` (weights are set to $1/n$).
`object`	For the auxiliary functions:A fitted model, output of a call to the main functions.
`...`	For the auxiliary functions: Optional arguments. Not used.

Value

For `lmr`:

`coefficients`	coefficient matrix.
`residuals`	residual matrix.
`effects`	component relating to the linear fit, for use by extractor functions.
`rank`	the numeric rank of the fitted linear model.
`fitted.values`	the fitted mean values.
`assign`	component relating to the linear fit, for use by extractor functions.
`qr`	component relating to the linear fit, for use by extractor functions.

df.residual	the residual degrees of freedom.
xlevels	(only where relevant) a record of the levels of the factors used in fitting.
call	the matched call.
terms	the terms object used.
model	the model frame used.

For coef.Lmr:

int	matrix (1,nlv) with the intercepts
B	matrix (n,nlv) with the coefficients

For predict.Lmr:

pred	A list of matrices (m, q) with the Y predicted values for the new X-data

Examples

```
n <- 8 ; p <- 3
X <- matrix(rnorm(n * p, mean = 10), ncol = p, byrow = TRUE)
y <- rnorm(n)
Y <- cbind(y, rnorm(n))
Xtrain <- X[1:6, ] ; Ytrain <- Y[1:6, ]
Xtest <- X[7:8, ] ; Ytest <- Y[7:8, ]

fm <- lmr(Xtrain, Ytrain)
coef(fm)

predict(fm, Xtest)

pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)
```

---

lmrda *LMR-DA models*

Description

Discrimination (DA) based on linear regression (LMR).

Usage

```
lmrda(X, y, weights = NULL)

## S3 method for class 'Lmrda'
predict(object, X, ...)
```

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary function: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
weights	Weights (n) to apply to the training observations for the PLS2. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
object	For the auxiliary function: A fitted model, output of a call to the main functions.
...	For the auxiliary function: Optional arguments. Not used.

Details

The training variable y (univariate class membership) is transformed to a dummy table containing $nclas$ columns, where $nclas$ is the number of classes present in y. Each column is a dummy variable (0/1). Then, a linear regression model (LMR) is run on the $X-$data and the dummy table, returning predictions of the dummy variables. For a given observation, the final prediction is the class corresponding to the dummy variable for which the prediction is the highest.

Value

For lrmda:

fm	List with the outputs((coefficients): coefficient matrix; (residuals): residual matrix; (fitted.values): the fitted mean values; (effects): component relating to the linear fit, for use by extractor functions; (weights): Weights (n) applied to the training observations for the PLS2; (rank): the numeric rank of the fitted linear model; (assign): component relating to the linear fit, for use by extractor functions; (qr): component relating to the linear fit, for use by extractor functions; (df.residual): the residual degrees of freedom; (xlevels): (only where relevant) a record of the levels of the factors used in fitting; (call): the matched call; (terms): the terms object used; (model): the model frame used).
lev	y levels.
ni	number of observations by level of y.

For predict.Lrmda:

pred	predicted classes of observations.
posterior	posterior probability of belonging to a class for each observation.

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)
m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]
```

```
fm <- lmrda(Xtrain, ytrain)
names(fm)
predict(fm, Xtest)

coef(fm$fm)

pred <- predict(fm, Xtest)$pred
err(pred, ytest)
```

`locw` *Locally weighted models*

Description

`locw` and `locwlv` are generic working functions returning predictions of KNN locally weighted (LW) models. One specific (= local) model is fitted for each observation to predict, and a prediction is returned. See the wrapper `lwplsr` (KNN-LWPLSR) for an example of use.

In KNN-LW models, the prediction is built from two sequential steps, therafter referred to as $weighting"1"$ and $weighting"2"$, respectively. For each new observation to predict, the two steps are as follow:

- $Weighting"1"$. The k nearest neighbors (in the training data set) are selected and the prediction model is fitted (in the next step) only on this neighborhood. It is equivalent to give a weight = 1 to the neighbors, and a weight = 0 to the other training observations, which corresponds to a binary weighting.

- $Weighting"2"$. Each of the k nearest neighbors eventually receives a weight (different from the usual $1/k$) before fitting the model. The weight depend from the dissimilarity (preliminary calculated) between the observation and the neighbor. This corresponds to a within-neighborhood weighting.

The prediction model used in step "2" has to be defined in a function specified in argument `fun`. If there are m new observations to predict, a list of m vectors defining the m neighborhoods has to be provided (argument `listnn`). Each of the m vectors contains the indexes of the nearest neighbors in the training set. The m vectors are not necessary of same length, i.e. the neighborhood size can vary between observations to predict. If there is a weighting in step "2", a list of m vectors of weights have to be provided (argument `listw`). Then `locw` fits the model successively for each of the m neighborhoods, and returns the corresponding m predictions.

Function `locwlv` is dedicated to prediction models based on latent variables (LVs) calculations, such as PLSR. It is much faster and recommended.

Usage

```
locw(Xtrain, Ytrain, X, listnn, listw = NULL, fun, verb = FALSE, ...)

locwlv(Xtrain, Ytrain, X, listnn, listw = NULL, fun, nlv, verb = FALSE, ...)
```

Arguments

Xtrain	Training X-data (n, p).
Ytrain	Training Y-data (n, q).
X	New X-data (m, p) to predict.
listnn	A list of m vectors defining weighting "1". Component i of this list is a vector (of length between 1 and n) of indexes. These indexes define the training observations that are the nearest neighbors of new observation i. Typically, listnn can be built from getknn, but any other list of length m can be provided. The m vectors can have equal length (i.e. the m neighborhoods are of equal size) or not (the number of neighbors varies between the observations to predict).
listw	A list of m vectors defining weighting "2". Component i of this list is a vector (that must have the same length as component i of listnn) of the weights given to the nearest neighbors when the prediction model is fitted. Internally, weights are "normalized" to sum to 1 in each component. Default to NULL (weights are set to $1/k$ where $k$is the size of the neihborhodd).
fun	A function corresponding to the prediction model to fit on the m neighborhoods.
nlv	For locwlv : The number of LVs to calculate.
verb	Logical. If TRUE, fitting information are printed.
...	Optional arguments to pass in function fun.

Value

pred	matrix or list of matrices (if nlv is a vector), with predictions

References

Lesnoff M, Metz M, Roger J-M. Comparison of locally weighted PLS strategies for regression and discrimination on agronomic NIR data. Journal of Chemometrics. 2020;n/a(n/a):e3209. doi:10.1002/cem.3209.

Examples

```
n <- 50 ; p <- 30
Xtrain <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 100 * ytrain)
m <- 4
Xtest <- matrix(rnorm(m * p), ncol = p, byrow = TRUE)
ytest <- rnorm(m)
Ytest <- cbind(ytest, 10 * ytest)

k <- 5
z <- getknn(Xtrain, Xtest, k = k)
listnn <- z$listnn
listd <- z$listd
listnn
listd

listw <- lapply(listd, wdist, h = 2)
```

```
listw

nlv <- 2
locw(Xtrain, Ytrain, Xtest,
     listnn = listnn, fun = plskern, nlv = nlv)
locw(Xtrain, Ytrain, Xtest,
     listnn = listnn, listw = listw, fun = plskern, nlv = nlv)

locwlv(Xtrain, Ytrain, Xtest,
     listnn = listnn, listw = listw, fun = plskern, nlv = nlv)
locwlv(Xtrain, Ytrain, Xtest,
     listnn = listnn, listw = listw, fun = plskern, nlv = 0:nlv)
```

`lwplsr` *KNN-LWPLSR*

Description

Function `lwplsr` fits KNN-LWPLSR models described in Lesnoff et al. (2020). The function uses functions `getknn`, `locw` and PLSR functions. See the code for details. Many variants of such pipelines can be build using `locw`.

Usage

```
lwplsr(X, Y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
    cri = 4,
    verb = FALSE)

## S3 method for class 'Lwplsr'
predict(object, X, ..., nlv = NULL)
```

Arguments

`X`	— For the main function: Training X-data (n, p). — For the auxiliary function: New X-data (m, p) to consider.
`Y`	Training Y-data (n, q).
`nlvdis`	The number of LVs to consider in the global PLS used for the dimension reduction before calculating the dissimilarities (see details). If `nlvdis = 0`, there is no dimension reduction.
`diss`	The type of dissimilarity used for defining the neighbors. Possible values are "eucl" (default; Euclidean distance), "mahal" (Mahalanobis distance), or "correlation". Correlation dissimilarities are calculated by sqrt(.5 * (1 - rho)).

h	A scale scalar defining the shape of the weight function. Lower is h, sharper is the function. See wdist.
k	The number of nearest neighbors to select for each observation to predict.
nlv	The number(s) of LVs to calculate in the local PLSR models.
cri	Argument cri in function wdist.
verb	Logical. If TRUE, fitting information are printed.
object	— For the auxiliary function: A fitted model, output of a call to the main function.
...	— For the auxiliary function: Optional arguments.

Details

- LWPLSR: This is a particular case of "weighted PLSR" (WPLSR) (e.g. Schaal et al. 2002). In WPLSR, a priori weights, different from the usual $1/n$ (standard PLSR), are given to the n training observations. These weights are used for calculating (i) the PLS scores and loadings and (ii) the regression model of the response(s) over the scores (by weighted least squares). LWPLSR is a particular case of WPLSR. "L" comes from "localized": the weights are defined from dissimilarities (e.g. distances) between the new observation to predict and the training observations. By definition of LWPLSR, the weights, and therefore the fitted WPLSR model, change for each new observation to predict.

- KNN-LWPLSR: Basic versions of LWPLSR (e.g. Sicard & Sabatier 2006, Kim et al 2011) use, for each observation to predict, all the n training observation. This can be very time consuming, in particular for large n. A faster and often more efficient strategy is to preliminary select, in the training set, a number of k nearest neighbors to the observation to predict (this is referred to as "$weighting1$" in function locw) and then to apply LWPLSR only to this pre-selected neighborhood (this is referred to as$weighting$"2" in locw). This strategy corresponds to KNN-LWPLSR.

In function lwplsr, the dissimilarities used for computing the weights can be calculated from the original X-data or after a dimension reduction (argument nlvdis). In the last case, global PLS scores are computed from (X, Y) and the dissimilarities are calculated on these scores. For high dimension X-data, the dimension reduction is in general required for using the Mahalanobis distance.

Value

For lwplsr: object of class Lwplsr

For predict.Lwplsr:

pred	prediction calculated for each observation
listnn	list with the neighbors used for each observation to be predicted
listd	list with the distances to the neighbors used for each observation to be predicted
listw	list with the weights attributed to the neighbors used for each observation to be predicted

References

Kim, S., Kano, M., Nakagawa, H., Hasebe, S., 2011. Estimation of active pharmaceutical ingredients content using locally weighted partial least squares and statistical wavelength selection. Int. J. Pharm., 421, 269-274.

Lesnoff, M., Metz, M., Roger, J.-M., 2020. Comparison of locally weighted PLS strategies for regression and discrimination on agronomic NIR data. Journal of Chemometrics, e3209. https://doi.org/10.1002/cem.3209

Schaal, S., Atkeson, C., Vijayamakumar, S. 2002. Scalable techniques from nonparametric statistics for the real time robot learning. Applied Intell., 17, 49-60.

Sicard, E. Sabatier, R., 2006. Theoretical framework for local PLS1 regression and application to a rainfall data set. Comput. Stat. Data Anal., 51, 1393-1410.

Examples

```
n <- 30 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 100 * ytrain)
m <- 4
Xtest <- matrix(rnorm(m * p), ncol = p)
ytest <- rnorm(m)
Ytest <- cbind(ytest, 10 * ytest)

nlvdis <- 5 ; diss <- "mahal"
h <- 2 ; k <- 10
nlv <- 2
fm <- lwplsr(
    Xtrain, Ytrain,
    nlvdis = nlvdis, diss = diss,
    h = h, k = k,
    nlv = nlv)
res <- predict(fm, Xtest)
names(res)
res$pred
msep(res$pred, Ytest)

res <- predict(fm, Xtest, nlv = 0:2)
res$pred
```

---

lwplsr_agg *Aggregation of KNN-LWPLSR models with different numbers of LVs*

Description

Ensemblist method where the predictions are calculated by averaging the predictions of KNN-LWPLSR models (lwplsr) built with different numbers of latent variables (LVs).

For instance, if argument `nlv` is set to `nlv = "5:10"`, the prediction for a new observation is the simple average of the predictions returned by the models with 5 LVS, 6 LVs, ... 10 LVs, respectively.

Usage

```
lwplsr_agg(
    X, Y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
    cri = 4,
    verb = FALSE
    )

## S3 method for class 'Lwplsr_agg'
predict(object, X, ...)
```

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary function: New X-data (m, p) to consider.
Y	Training Y-data (n, q).
nlvdis	The number of LVs to consider in the global PLS used for the dimension reduction before calculating the dissimilarities. If nlvdis = 0, there is no dimension reduction.
diss	The type of dissimilarity used for defining the neighbors. Possible values are "eucl" (default; Euclidean distance), "mahal" (Mahalanobis distance), or "correlation". Correlation dissimilarities are calculated by sqrt(.5 * (1 - rho)).
h	A scale scalar defining the shape of the weight function. Lower is h, sharper is the function. See wdist.
k	The number of nearest neighbors to select for each observation to predict.
nlv	A character string such as "5:20" defining the range of the numbers of LVs to consider (here: the models with nb LVS = 5, 6, ..., 20 are averaged). Syntax such as "10" is also allowed (here: correponds to the single model with 10 LVs).
cri	Argument cri in function wdist.
verb	Logical. If TRUE, fitting information are printed.
object	For the auxiliary function: A fitted model, output of a call to the main function.
...	For the auxiliary function: Optional arguments. Not used.

Value

For lwplsr_agg: object of class Lwplsr_agg

For predict.Lwplsr_agg:

pred	prediction calculated for each observation, which is the most occurent level (vote) over the predictions returned by the models with different numbers of LVS respectively
listnn	list with the neighbors used for each observation to be predicted

listd	list with the distances to the neighbors used for each observation to be predicted
listw	list with the weights attributed to the neighbors used for each observation to be predicted

Note

In the examples, gridscore and gricv have been used as there is no sense to use gridscorelv and gricvlv.

Examples

```
## EXAMPLE 1

n <- 30 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 100 * ytrain)
m <- 4
Xtest <- matrix(rnorm(m * p), ncol = p)
ytest <- rnorm(m)
Ytest <- cbind(ytest, 10 * ytest)

nlvdis <- 5 ; diss <- "mahal"
h <- 2 ; k <- 10
nlv <- "2:6"
fm <- lwplsr_agg(
    Xtrain, Ytrain,
    nlvdis = nlvdis, diss = diss,
    h = h, k = k,
    nlv = nlv)
names(fm)
res <- predict(fm, Xtest)
names(res)
res$pred
msep(res$pred, Ytest)

## EXAMPLE 2

n <- 30 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 100 * ytrain)
m <- 4
Xtest <- matrix(rnorm(m * p), ncol = p)
ytest <- rnorm(m)
Ytest <- cbind(ytest, 10 * ytest)

nlvdis <- 5 ; diss <- "mahal"
h <- c(2, Inf)
k <- c(10, 20)
nlv <- c("1:3", "2:5")
pars <- mpars(nlvdis = nlvdis, diss = diss,
```

```
    h = h, k = k, nlv = nlv)
pars
res <- gridscore(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = lwplsr_agg,
    pars = pars)
res

## EXAMPLE 3

n <- 30 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 100 * ytrain)
m <- 4
Xtest <- matrix(rnorm(m * p), ncol = p)
ytest <- rnorm(m)
Ytest <- cbind(ytest, 10 * ytest)

K = 3
segm <- segmkf(n = n, K = K, nrep = 1)
segm
res <- gridcv(
    Xtrain, Ytrain,
    segm, score = msep,
    fun = lwplsr_agg,
    pars = pars,
    verb = TRUE)
res
```

`lwplsrda` *KNN-LWPLS-DA Models*

Description

- `lwplsrda`: KNN-LWPLSRDA models. This is the same methodology as for `lwplsr` except that PLSR is replaced by PLSRDA (`plsrda`). See the help page of `lwplsr` for details.

- `lwplslda` and `lwplsqda`: Same as above, but PLSRDA is replaced by either PLSLDA (`plslda`) or PLSQDA ((`plsqda`), respecively.

Usage

```
lwplsrda(
    X, y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
```

```
    cri = 4,
    verb = FALSE
    )

lwplslda(
    X, y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
    prior = c("unif", "prop"),
    cri = 4,
    verb = FALSE
    )

lwplsqda(
    X, y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
    prior = c("unif", "prop"),
    cri = 4,
    verb = FALSE
    )

## S3 method for class 'Lwplsrda'
predict(object, X, ..., nlv = NULL)

## S3 method for class 'Lwplsprobda'
predict(object, X, ..., nlv = NULL)
```

Arguments

X	For the main functions: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
nlvdis	The number of LVs to consider in the global PLS used for the dimension reduction before calculating the dissimilarities. If `nlvdis = 0`, there is no dimension reduction.
diss	The type of dissimilarity used for defining the neighbors. Possible values are "eucl" (default; Euclidean distance), "mahal" (Mahalanobis distance), or "correlation". Correlation dissimilarities are calculated by sqrt(.5 * (1 - rho)).
h	A scale scalar defining the shape of the weight function. Lower is h, sharper is the function. See wdist.
k	The number of nearest neighbors to select for each observation to predict.
nlv	The number(s) of LVs to calculate in the local PLSDA models.

prior	The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
cri	Argument cri in function wdist.
verb	Logical. If TRUE, fitting information are printed.
object	For the auxiliary functions: A fitted model, output of a call to the main function.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For lwplsrda, lwplslda, lwplsqda: object of class Lwplsrda or Lwplsprobda,

For predict.Lwplsrda, predict.Lwplsprobda :

pred	class predicted for each observation
listnn	list with the neighbors used for each observation to be predicted
listd	list with the distances to the neighbors used for each observation to be predicted
listw	list with the weights attributed to the neighbors used for each observation to be predicted

Examples

```
n <- 50 ; p <- 7
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)
m <- 4
Xtest <- matrix(rnorm(m * p), ncol = p)
ytest <- sample(c(1, 4, 10), size = m, replace = TRUE)

nlvdis <- 5 ; diss <- "mahal"
h <- 2 ; k <- 10
nlv <- 2
fm <- lwplsrda(
    Xtrain, ytrain,
    nlvdis = nlvdis, diss = diss,
    h = h, k = k,
    nlv = nlv
    )
res <- predict(fm, Xtest)
res$pred
res$listnn
err(res$pred, ytest)

res <- predict(fm, Xtest, nlv = 0:2)
res$pred
```

lwplsrda_agg *Aggregation of KNN-LWPLSDA models with different numbers of LVs*

Description

Ensemblist method where the predictions are calculated by "averaging" the predictions of KNN-LWPLSDA models built with different numbers of latent variables (LVs).

For instance, if argument `nlv` is set to `nlv = "5:10"`, the prediction for a new observation is the most occurent level (vote) over the predictions returned by the models with 5 LVS, 6 LVs, ... 10 LVs, respectively.

- `lwplsrda_agg`: use `plsrda`.

- `lwplslda_agg`: use `plslda`.

- `lwplsqda_agg`: use `plsqda`.

Usage

```
lwplsrda_agg(
    X, y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
    cri = 4,
    verb = FALSE
    )

lwplslda_agg(
    X, y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
    prior = c("unif", "prop"),
    cri = 4,
    verb = FALSE
    )

lwplsqda_agg(
    X, y,
    nlvdis, diss = c("eucl", "mahal"),
    h, k,
    nlv,
    prior = c("unif", "prop"),
    cri = 4,
    verb = FALSE
    )
```

```
## S3 method for class 'Lwplsrda_agg'
predict(object, X, ...)

## S3 method for class 'Lwplsprobda_agg'
predict(object, X, ...)
```

Arguments

X	For the main functions: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
nlvdis	The number of LVs to consider in the global PLS used for the dimension reduction before calculating the dissimilarities. If nlvdis = 0, there is no dimension reduction.
diss	The type of dissimilarity used for defining the neighbors. Possible values are "eucl" (default; Euclidean distance), "mahal" (Mahalanobis distance), or "correlation". Correlation dissimilarities are calculated by sqrt(.5 * (1 - rho)).
h	A scale scalar defining the shape of the weight function. Lower is h, sharper is the function. See wdist.
k	The number of nearest neighbors to select for each observation to predict.
nlv	A character string such as "5:20" defining the range of the numbers of LVs to consider (here: the models with nb LVS = 5, 6, ..., 20 are averaged). Syntax such as "10" is also allowed (here: correponds to the single model with 10 LVs).
prior	For lwplslda_agg and lwplsqda_agg: The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
cri	Argument cri in function wdist.
verb	Logical. If TRUE, fitting information are printed.
object	For the auxiliary functions: A fitted model, output of a call to the main function.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For lwplsrda_agg, lwplslda_agg and lwplsqda_agg: object of class lwplsrda_agg, lwplslda_agg or lwplsqda_agg

For predict.Lwplsrda_agg and predict.Lwplsprobda_agg:

pred	prediction calculated for each observation, which is the most occurent level (vote) over the predictions returned by the models with different numbers of LVS respectively
listnn	list with the neighbors used for each observation to be predicted
listd	list with the distances to the neighbors used for each observation to be predicted
listw	list with the weights attributed to the neighbors used for each observation to be predicted

Note

The first example concerns KNN-LWPLSRDA-AGG. The second example concerns KNN-LWPLSLDA-AGG.

Examples

```
## KNN-LWPLSRDA-AGG

n <- 40 ; p <- 7
X <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
y <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtrain <- X ; ytrain <- y
m <- 5
Xtest <- X[1:m, ] ; ytest <- y[1:m]

nlvdis <- 5 ; diss <- "mahal"
h <- 2 ; k <- 10
nlv <- "2:4"
fm <- lwplsrda_agg(
    Xtrain, ytrain,
    nlvdis = nlvdis, diss = diss,
    h = h, k = k,
    nlv = nlv)
res <- predict(fm, Xtest)
res$pred
res$listnn


nlvdis <- 5 ; diss <- "mahal"
h <- c(2, Inf)
k <- c(10, 15)
nlv <- c("1:3", "2:4")
pars <- mpars(nlvdis = nlvdis, diss = diss,
              h = h, k = k, nlv = nlv)
pars

res <- gridscore(
    Xtrain, ytrain, Xtest, ytest,
    score = err,
    fun = lwplsrda_agg,
    pars = pars)
res

segm <- segmkf(n = n, K = 3, nrep = 1)
res <- gridcv(
    Xtrain, ytrain,
    segm, score = err,
    fun = lwplsrda_agg,
    pars = pars,
    verb = TRUE)
names(res)
```

```
res$val


## KNN-LWPLSLDA-AGG

n <- 40 ; p <- 7
X <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
y <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtrain <- X ; ytrain <- y
m <- 5
Xtest <- X[1:m, ] ; ytest <- y[1:m]

nlvdis <- 5 ; diss <- "mahal"
h <- 2 ; k <- 10
nlv <- "2:4"
fm <- lwplslda_agg(
    Xtrain, ytrain,
    nlvdis = nlvdis, diss = diss,
    h = h, k = k,
    nlv = nlv, prior = "prop")
res <- predict(fm, Xtest)
res$pred
res$listnn

nlvdis <- 5 ; diss <- "mahal"
h <- c(2, Inf)
k <- c(10, 15)
nlv <- c("1:3", "2:4")
pars <- mpars(nlvdis = nlvdis, diss = diss,
              h = h, k = k, nlv = nlv,
              prior = c("unif", "prop"))
pars

res <- gridscore(
    Xtrain, ytrain, Xtest, ytest,
    score = err,
    fun = lwplslda_agg,
    pars = pars)
res

segm <- segmkf(n = n, K = 3, nrep = 1)
res <- gridcv(
    Xtrain, ytrain,
    segm, score = err,
    fun = lwplslda_agg,
    pars = pars,
    verb = TRUE)
names(res)
res$val
```

matW *Between and within covariance matrices*

Description

Calculation of within (matW) and between (matB) covariance matrices for classes of observations.

Usage

```
matW(X, y)

matB(X, y)
```

Arguments

X	Data (n, p) on whch are calculated the covariances.
y	Class membership $(n, 1)$.

Details

The denominator in the variance calculations is n.

Value

For (matW):

W	within covariance matrix.
Wi	list of covariance matrices for each class.
lev	classes
ni	number of observations in each per class

For (matB):

B	between covariance matrix.
ct	matrix of class centers.
lev	classes
ni	number of observations in each per class

Examples

```
n <- 8 ; p <- 3
X <- matrix(rnorm(n * p), ncol = p)
y <- sample(1:2, size = n, replace = TRUE)
X
y

matW(X, y)

matB(X, y)

matW(X, y)$W + matB(X, y)$B
(n - 1) / n * cov(X)
```

---

mavg *Smoothing by moving average*

Description

Smoothing, by moving average, of the row observations (e.g. spectra) of a dataset.

Usage

```
mavg(X, n = 5)
```

Arguments

X	X-data (n, p).
n	The number of points (i.e. columns of X) defining the window over wich is calculate each average. The smoothing is calculated for the point at the center of the window. Therefore, n must be an odd integer, and be higher or equal to 3.

Value

A matrix of the transformed data.

Examples

```
data(cassav)

X <- cassav$Xtest
headm(X)

Xp <- mavg(X, n = 11)
headm(Xp)

oldpar <- par(mfrow = c(1, 1))
par(mfrow = c(1, 2))
```

```
plotsp(X, main = "Signal")
plotsp(Xp, main = "Corrected signal")
abline(h = 0, lty = 2, col = "grey")
par(oldpar)
```

`mbplsr` *multi-block PLSR algorithms*

Description

Algorithm fitting a multi-block PLS1 or PLS2 model between dependent variables $Xlist$ and responses Y, based on the "Improved kernel algorithm #1" proposed by Dayal and MacGregor (1997).

For weighted versions, see for instance Schaal et al. 2002, Siccard & Sabatier 2006, Kim et al. 2011 and Lesnoff et al. 2020.

Auxiliary functions

`transform` Calculates the LVs for any new matrix X from the model.

`summary` returns summary information for the model.

`coef` Calculates b-coefficients from the model, adjuted for raw data.

`predict` Calculates the predictions for any new matrix X from the model.

Usage

```
mbplsr(Xlist, Y, blockscaling = TRUE, weights = NULL, nlv,
Xscaling = c("none", "pareto", "sd")[1], Yscaling = c("none", "pareto", "sd")[1])

## S3 method for class 'Mbplsr'
transform(object, X, ..., nlv = NULL)

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

## S3 method for class 'Mbplsr'
coef(object, ..., nlv = NULL)

## S3 method for class 'Mbplsr'
predict(object, X, ..., nlv = NULL)
```

Arguments

`Xlist`	For the main function: list of training X-data ($n$rows).
`X`	For the auxiliary functions: list of new X-data, with the same variables than the training X-data.
`Y`	Training Y-data (n, q).

blockscaling	logical. If TRUE, the scaling factor (computed on the training) is the "norm" of the block, i.e. the square root of the sum of the variances of each column of the block.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	For the main functions: The number(s) of LVs to calculate. — For the auxiliary functions: The number(s) of LVs to consider.
Xscaling	vector (of length Xlist) of variable scaling for each datablock, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	character. variable scaling for the Y-block, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

A list of outputs, such as

T	The X-score matrix (n, nlv).
P	The X-loadings matrix (p, nlv).
W	The X-loading weights matrix (p, nlv).
C	The Y-loading weights matrix (C = t(Beta), where Beta is the scores regression coefficients matrix).
R	The PLS projection matrix (p, nlv).
xmeans	The list of centering vectors of $Xlist$.
ymeans	The centering vector of Y $(q, 1)$.
xscales	The list of $Xlist$ variable standard deviations.
yscales	The vector of Y variable standard deviations $(q, 1)$.
weights	Weights applied to the training observations.
TT	the X-score normalization factor.
blockscaling	block scaling.
Xnorms	"norm" of each block, i.e. the square root of the sum of the variances of each column of each block, computed on the training, and used as scaling factor

.

U	intermediate output.

For transform.Mbplsr: X-scores matrix for new Xlist-data.

For summary.Mbplsr:

`explvarx` matrix of explained variances.

For `coef.Mbplsr`:

`int` matrix (1,nlv) with the intercepts

`B` matrix (n,nlv) with the coefficients

For `predict.Mbplsr`:

`pred` A list of matrices (m, q) with the Y predicted values for the new Xlist-data

References

Andersson, M., 2009. A comparison of nine PLS1 algorithms. Journal of Chemometrics 23, 518-529.

Dayal, B.S., MacGregor, J.F., 1997. Improved PLS algorithms. Journal of Chemometrics 11, 73-85.

Hoskuldsson, A., 1988. PLS regression methods. Journal of Chemometrics 2, 211-228. https://doi.org/10.1002/cem.1180020

Kim, S., Kano, M., Nakagawa, H., Hasebe, S., 2011. Estimation of active pharmaceutical ingredients content using locally weighted partial least squares and statistical wavelength selection. Int. J. Pharm., 421, 269-274.

Lesnoff, M., Metz, M., Roger, J.M., 2020. Comparison of locally weighted PLS strategies for regression and discrimination on agronomic NIR Data. Journal of Chemometrics. e3209. https://onlinelibrary.wiley.com/doi/ab

Rannar, S., Lindgren, F., Geladi, P., 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. https://doi.org/10.1002/cem.1180080204

Schaal, S., Atkeson, C., Vijayamakumar, S. 2002. Scalable techniques from nonparametric statistics for the real time robot learning. Applied Intell., 17, 49-60.

Sicard, E. Sabatier, R., 2006. Theoretical framework for local PLS1 regression and application to a rainfall data set. Comput. Stat. Data Anal., 51, 1393-1410.

Tenenhaus, M., 1998. La régression PLS: théorie et pratique. Editions Technip, Paris, France.

Wold, S., Sjostrom, M., Eriksson, l., 2001. PLS-regression: a basic tool for chemometrics. Chem. Int. Lab. Syst., 58, 109-130.

See Also

`mbplsr_mbplsda_allsteps` function to help determine the optimal number of latent variables, perform a permutation test, calculate model parameters and predict new observations.

Examples

```
n <- 10 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)

m <- 2
Xtest <- matrix(rnorm(m * p), ncol = p)
```

```
colnames(Xtrain) <- colnames(Xtest) <- paste("v", 1:p, sep = "")

Xtrain
Xtest

blocks <- list(1:2, 4, 6:8)
X1 <- mblocks(Xtrain, blocks = blocks)
X2 <- mblocks(Xtest, blocks = blocks)

nlv <- 3
fm <- mbplsr(Xlist = X1, Y = ytrain, Xscaling = c("sd","none","none"),
blockscaling = TRUE, weights = NULL, nlv = nlv)

summary(fm, X1)
coef(fm)
transform(fm, X2)
predict(fm, X2)
```

---

mbplsr_mbplsda_allsteps

MBPLSR or MBPLSDA analysis steps

Description

Help determine the optimal number of latent variables by cross-validation, perform a permutation test, calculate model parameters and predict new observations, for mbplsr (mbplsr), mbplsrda (mbplsrda), mbplslda (mbplslda) or mbplsqda (mbplsqda) models.

Usage

```
mbplsr_mbplsda_allsteps(Xlist, Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y, Yscaling = c("none","pareto","sd")[1], weights = NULL,
                    newXlist = NULL, newXnames = NULL,

                    method = c("mbplsr", "mbplsrda","mbplslda","mbplsqda")[1],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[1],
                    nlv,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[1],
                    nbrep = 30,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 30,
```

```
criterion = c("err","rmse")[1],
selection = c("localmin","globalmin","1std")[1],

import = c("R","ChemFlow","W4M")[1],
outputfilename = NULL)
```

Arguments

Xlist	list of training X-data (n, p).
Xnames	names of the X-matrices
Xscaling	vector of Xlist length. X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Y	Training Y-data (n, q) for plsr models, and $(n, 1)$ for plsrda, plslda or plsqda models.
Yscaling	Y variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
newXlist	list of new X-data (m, p) to consider.
newXnames	names of the newX-matrices
method	method to apply among "mbplsr", "mbplsrda","mbplslda","mbplsqda"
prior	for mbplslda or mbplsqda models : The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
step	step of the analysis among "nlvtest" (cross-validation to help determine the optimal number of latent variables), "permutation" (permutation test),"model" (model calculation),"prediction" (prediction of newX-data or X-data if any))
nlv	number of latent variables to test if step is "nlvtest"; number of latent variables of the model if step is not "nlvtest".
modeloutput	if step is "model": outputs among "scores", "loadings", "coef" (regression coefficients), "vip" (Variable Importance in Projection; the VIP calculation being based on the proportion of Y-variance explained by the components, as proposed by Mehmood et al (2012, 2020).)
cvmethod	if step is "nlvtest" or "permutation": "kfolds" for k-folds cross-validation, or "loo" for leave-one-out.
nbrep	if step is "nlvtest" and cvmethod is "kfolds": An integer, setting the number of CV repetitions. Default value is 30. Must me set to 1 if cvmethod is "loo"
seed	if step is "nlvtest" and cvmethod is "kfolds", or if step is "permutation: a numeric. Seed used for the repeated resampling

samplingk	A vector of length n. The elements are the values of a qualitative variable used for stratified partition creation. If NULL, the first observation is set in the first fold, the second observation in the second fold, etc...
nfolds	if cvmethod is "kfolds". An integer, setting the number of partitions to create. Default value is 10.
npermut	if step is "permutation": An integer, setting the number of Y-Block with permutated responses to create. Default value is 30.
criterion	if step is "nlvtest" or "permutation" and method is "mbplsrda", "mbplslda" or "mbplsqda": optimisation criterion among "rmse" and "err" (for classification error rate)))
selection	if step is "nlvtest": a character indicating the selection method to use to choose the optimal combination of components, among "localmin","globalmin","1std". If "localmin": the optimal combination corresponds to the first local minimum of the mean CV rmse or error rate. If "globalmin" : the optimal combination corresponds to the minimum mean CV rmse or error rate. If "1std" (one standard error rule) : it corresponds to the first combination after which the mean cross-validated rmse or error rate does not decrease significantly.
import	If "R", X and Y are in the global environment, and the observation names are in rownames. If "ChemFlow", X and Y are tabulated tables (.txt), and the observation names are in the first column. If "W4M", X and Y are tabulated tables (.txt), and the observation names are in the headers of X, and in the first column of Y.
outputfilename	character: If not NULL, name of the tabular file, in which the function outputs have to be written.)

Value

If step is "nlvtest": table with rmsecv or cross-validated classification error rates. The suggested optimal number of latent variables is indicated by the binary "optimum" variable.

If step is "permutation": table with the dissimilarity between the original and the permutated Y-block, and the rmsecv or cross-validated classification error rates obtained with the permutated Y-block by the model and the given number of latent variables.

If step is "model": tables of scores, loadings, regression coefficients, and vip values, depending of the "modeloutput" parameter.

If step is "prediction": table of predicted scores and predicted classes or values.

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
colnames(Xtrain) <- paste0("V",1:p)

ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]

Xtrainlist <- list(Xtrain[,1:3], Xtrain[,4:8])
```

```
Xtestlist <- list(Xtest[,1:3], Xtest[,4:8])

nlv <- 5

resnlvtestmbplsrda <- mbplsr_mbplsda_allsteps(Xlist = Xtrainlist,
                    Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newXlist = Xtestlist, newXnames = NULL,

                    method = c("mbplsr", "mbplsrda","mbplslda","mbplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[1],
                    nlv = 5,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)

respermutationmbplsrda <- mbplsr_mbplsda_allsteps(Xlist = Xtrainlist,
                    Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newXlist = Xtestlist, newXnames = NULL,

                    method = c("mbplsr", "mbplsrda","mbplslda","mbplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[2],
                    nlv = 1,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)
```

```
plotxy(respermutationmbplsrda, pch=16)
abline (h = respermutationmbplsrda[respermutationmbplsrda[,"permut_dyssimilarity"]==0,"res_permut"])

resmodelmbplsrda <- mbplsr_mbplsda_allsteps(Xlist = Xtrainlist,
                    Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newXlist = Xtestlist, newXnames = NULL,

                    method = c("mbplsr", "mbplsrda","mbplslda","mbplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[3],
                    nlv = 1,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)


resmodelmbplsrda$scores
resmodelmbplsrda$loadings
resmodelmbplsrda$coef
resmodelmbplsrda$vip

respredictionmbplsrda <- mbplsr_mbplsda_allsteps(Xlist = Xtrainlist,
                    Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newXlist = Xtestlist, newXnames = NULL,

                    method = c("mbplsr", "mbplsrda","mbplslda","mbplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[4],
                    nlv = 1,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
```

```
selection = c("localmin","globalmin","1std")[1],

outputfilename = NULL)
```

---

mbplsrda *multi-block PLSDA models*

Description

Multi-block discrimination (DA) based on PLS.

The training variable y (univariate class membership) is firstly transformed to a dummy table containing $nclas$ columns, where $nclas$ is the number of classes present in y. Each column is a dummy variable (0/1). Then, a PLS2 is implemented on the $X-$data and the dummy table, returning latent variables (LVs) that are used as dependent variables in a DA model.

- `mbplsrda`: Usual "PLSDA". A linear regression model predicts the Y-dummy table from the PLS2 LVs. This corresponds to the PLSR2 of the X-data and the Y-dummy table. For a given observation, the final prediction is the class corresponding to the dummy variable for which the prediction is the highest.

- `mbplslda` and `mbplsqda`: Probabilistic LDA and QDA are run over the PLS2 LVs, respectively.

Usage

```
mbplsrda(Xlist, y, blockscaling = TRUE, weights = NULL, nlv,
Xscaling = c("none", "pareto", "sd")[1], Yscaling = c("none", "pareto", "sd")[1])

mbplslda(Xlist, y, blockscaling = TRUE, weights = NULL, nlv, prior = c("unif", "prop"),
Xscaling = c("none", "pareto", "sd")[1], Yscaling = c("none", "pareto", "sd")[1])

mbplsqda(Xlist, y, blockscaling = TRUE, weights = NULL, nlv, prior = c("unif", "prop"),
Xscaling = c("none", "pareto", "sd")[1], Yscaling = c("none", "pareto", "sd")[1])

## S3 method for class 'Mbplsrda'
predict(object, X, ..., nlv = NULL)

## S3 method for class 'Mbplsprobda'
predict(object, X, ..., nlv = NULL)
```

Arguments

`Xlist`	For the main functions: list of training X-data ($n$rows).
`X`	For the auxiliary functions: list of new X-data (n rows), with the same variables than the training X-data.
`y`	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.

blockscaling	logical. If TRUE, the scaling factor (computed on the training) is the "norm" of the block, i.e. the square root of the sum of the variances of each column of the block.
weights	Weights (n) to apply to the training observations for the PLS2. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	The number(s) of LVs to calculate.
prior	The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
Xscaling	vector (of length Xlist) of variable scaling for each datablock, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	character. variable scaling for the Y-block after binary transformation, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For mbplsrda:

fm	list with the MB-PLS model: (T): X-scores matrix; (P): X-loading matrix;(R): The PLS projection matrix (p,nlv); (W): X-loading weights matrix ;(C): The Y-loading weights matrix; (TT): the X-score normalization factor; (xmeans): the centering vector of X (p,1); (ymeans): the centering vector of Y (q,1); (weights): vector of observation weights; (blockscaling): block scaling; (Xnorms): "norm" of each block; (U): intermediate output.
lev	classes
ni	number of observations in each class

For mbplslda, mbplsqda:

fm	list with [[1]] the MB-PLS model: (T): X-scores matrix; (P): X-loading matrix;(R): The PLS projection matrix (p,nlv); (W): X-loading weights matrix ;(C): The Y-loading weights matrix; (TT): the X-score normalization factor; (xmeans): the centering vectors of X; (ymeans): the centering vector of Y (q,1); (xscales): the scaling vector of X (p,1); (yscales): the scaling vector of Y (q,1); (weights): vector of observation weights; (blockscaling): block scaling; (Xnorms): "norm" of each block; (U): intermediate output. [[2]] lda or qda models.
lev	classes
ni	number of observations in each class

For predict.Mbplsrda, predict.Mbplsprobda:

pred	predicted class for each observation
posterior	calculated probability of belonging to a class for each observation

Note

The first example concerns MB-PLSDA, and the second one concerns MB-PLS LDA. `fm` are PLS1 models, and `zfm` are PLS2 models.

See Also

`mbplsr_mbplsda_allsteps` function to help determine the optimal number of latent variables, perform a permutation test, calculate model parameters and predict new observations.

Examples

```
## EXAMPLE OF MB-PLSDA

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
Xtrainlist <- list(Xtrain[,1:3], Xtrain[,4:8])

ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]
Xtestlist <- list(Xtest[,1:3], Xtest[,4:8])

nlv <- 5
fm <- mbplsrda(Xtrainlist, ytrain, Xscaling = "sd", nlv = nlv)
names(fm)

predict(fm, Xtestlist)
predict(fm, Xtestlist, nlv = 0:2)$pred

pred <- predict(fm, Xtestlist)$pred
err(pred, ytest)

zfm <- fm$fm
transform(zfm, Xtestlist)
transform(zfm, Xtestlist, nlv = 1)
summary(zfm, Xtrainlist)
coef(zfm)
coef(zfm, nlv = 0)
coef(zfm, nlv = 2)

## EXAMPLE OF MB-PLS LDA

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
Xtrainlist <- list(Xtrain[,1:3], Xtrain[,4:8])

ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]
Xtestlist <- list(Xtest[,1:3], Xtest[,4:8])

nlv <- 5
```

```
fm <- mbplslda(Xtrainlist, ytrain, Xscaling = "none", nlv = nlv)
predict(fm, Xtestlist)
predict(fm, Xtestlist, nlv = 1:2)$pred

zfm <- fm[[1]][[1]]
class(zfm)
names(zfm)
summary(zfm, Xtrainlist)
transform(zfm, Xtestlist)
coef(zfm)

## EXAMPLE OF MB-PLS QDA

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
Xtrainlist <- list(Xtrain[,1:3], Xtrain[,4:8])

ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]
Xtestlist <- list(Xtest[,1:3], Xtest[,4:8])

nlv <- 5
fm <- mbplsqda(Xtrainlist, ytrain, Xscaling = "none", nlv = nlv)
predict(fm, Xtestlist)
predict(fm, Xtestlist, nlv = 1:2)$pred

zfm <- fm[[1]][[1]]
class(zfm)
names(zfm)
summary(zfm, Xtrainlist)
transform(zfm, Xtestlist)
coef(zfm)
```

---

mse *Residuals and prediction error rates*

Description

Residuals and prediction error rates (MSEP, SEP, etc. or classification error rate) for models with quantitative or qualitative responses.

Usage

```
residreg(pred, Y)
residcla(pred, y)

msep(pred, Y)
rmsep(pred, Y)
```

```
sep(pred, Y)
bias(pred, Y)
cor2(pred, Y)
r2(pred, Y)
rpd(pred, Y)
rpdr(pred, Y)
mse(pred, Y, digits = 3)

err(pred, y)
```

Arguments

`pred`	Prediction (m, q); output of a function `predict`.
`Y`	Observed response (m, q).
`y`	Observed response $(m, 1)$.
`digits`	Number of digits for the numerical outputs.

Details

The rate $R2$ is calculated by $R2 = 1 - MSEP(currentmodel)/MSEP(nullmodel)$, where $MSEP = Sum((y_i - pred_i)^2)/n$ and "null model" is the overall mean of y. For predictions over CV or Test sets, and/or for non linear models, it can be different from the square of the correlation coefficient ($cor2$) between the observed values and the predictions.

Function `sep` computes the SEP, referred to as "corrected SEP" (SEP_c) in Bellon et al. 2010. SEP is the standard deviation of the residuals. There is the relation: $MSEP = BIAS^2 + SEP^2$.

Function `rpd` computes the ratio of the "deviation" (sqrt of the mean of the squared residuals for the null model when it is defined by the simple average) to the "performance" (sqrt of the mean of the squared residuals for the current model, i.e. RMSEP), i.e. $RPD = SD/RMSEP = RMSEP(nullmodel)/RMSEP$ (see eg. Bellon et al. 2010).

Function `rpdr` computes a robust RPD.

Value

Residuals or prediction error rates.

References

Bellon-Maurel, V., Fernandez-Ahumada, E., Palagos, B., Roger, J.-M., McBratney, A., 2010. Critical review of chemometric indicators commonly used for assessing the quality of the prediction of soil attributes by NIR spectroscopy. TrAC Trends in Analytical Chemistry 29, 1073-1081. https://doi.org/10.1016/j.trac.2010.05.006

Examples

```
## EXAMPLE 1

n <- 6 ; p <- 4
```

```
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE]
ytest <- Ytest[1:m, 1]
nlv <- 3
fm <- plskern(Xtrain, Ytrain, nlv = nlv)
pred <- predict(fm, Xtest)$pred

residreg(pred, Ytest)
msep(pred, Ytest)
rmsep(pred, Ytest)
sep(pred, Ytest)
bias(pred, Ytest)
cor2(pred, Ytest)
r2(pred, Ytest)
rpd(pred, Ytest)
rpdr(pred, Ytest)
mse(pred, Ytest, digits = 3)

## EXAMPLE 2

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)
Xtest <- Xtrain[1:5, ]
ytest <- ytrain[1:5]
nlv <- 5
fm <- plsrda(Xtrain, ytrain, nlv = nlv)
pred <- predict(fm, Xtest)$pred

residcla(pred, ytest)
err(pred, ytest)
```

`nipals` *nipals*

Description

Nipals to compute the first score and loading vectors of a matrix.

Usage

```
nipals(X, tol = 1e-6, maxit = 100)
```

Arguments

X	Datamatrix (n, p).
tol	Tolerance for testing convergence of the NIPALS iterations for each PC.
maxit	Maximum number of NIPALS iterations for each PC.

Value

u	left singular vector (u * sv = scores).
v	right singular vector (loadings).
sv	singular value.

References

K.R. Gabriel, S. Zamir, Lower rank approximation of matrices by least squares with any choice of weights, Technometrics 21 (1979) 489–498.

Examples

```
n <- 8 ; p <- 6
X <- matrix(rnorm(n * p, mean = 10), ncol = p, byrow = TRUE)
nipals(X)
```

---

octane *octane*

Description

Octane dataset.

Near infrared (NIR) spectra (absorbance) of n = 39 gasoline samples over p = 226 wavelengths (1102 nm to 1552 nm, step = 2 nm).

Samples 25, 26, and 36-39 contain added alcohol (outliers).

Usage

```
data(octane)
```

Format

A list with 1 component: the matrix X with 39 samples and 226 variables.

Source

K.H. Esbensen, S. Schoenkopf and T. Midtgaard Multivariate Analysis in Practice, Trondheim, Norway: Camo, 1994.

Todorov, V. 2020. rrcov: Robust Location and Scatter Estimation and Robust Multivariate Analysis with High Breakdown. R Package version 1.5-5. https://cran.r-project.org/.

References

M. Hubert, P. J. Rousseeuw, K. Vanden Branden (2005), ROBPCA: a new approach to robust principal components analysis, Technometrics, 47, 64-79.

P. J. Rousseeuw, M. Debruyne, S. Engelen and M. Hubert (2006), Robustness and Outlier Detection in Chemometrics, Critical Reviews in Analytical Chemistry, 36(3-4), 221-242.

Examples

```
data(octane)

X <- octane$X
headm(X)

plotsp(X, xlab = "Wawelength", ylab = "Absorbance")
plotsp(X[c(25:26, 36:39), ], add = TRUE, col = "red")
```

---

odis *Orthogonal distances from a PCA or PLS score space*

Description

`odis` calculates the orthogonal distances (OD = "X-residuals") for a PCA or PLS model. OD is the Euclidean distance of a row observation to its projection to the score plan (see e.g. Hubert et al. 2005, Van Branden & Hubert 2005, p. 66; Varmuza & Filzmoser, 2009, p. 79).

A distance cutoff is computed using a moment estimation of the parameters of a Chi-squared distribution for OD^2 (see Nomikos & MacGregor 1995, and Pomerantzev 2008). In the function output, column `dstand` is a standardized distance defined as $OD/cutoff$. A value `dstand` > 1 can be considered as extreme.

The cutoff for detecting extreme OD values is computed using a moment estimation of a Chi-squared distrbution for the squared distance.

Usage

```
odis(
    object, Xtrain, X = NULL,
    nlv = NULL,
    rob = TRUE, alpha = .01
    )
```

Arguments

`object`	A fitted model, output of a call to a fitting function.
`Xtrain`	Training X-data that was used to fit the model.
`X`	New X-data.
`nlv`	Number of components (PCs or LVs) to consider.

rob	Logical. If TRUE, the moment estimation of the distance cutoff is robustified. This can be recommended after robust PCA or PLS on small data sets containing extreme values.
alpha	Risk-I level for defining the cutoff detecting extreme values.

Value

res.train	matrix with distance and a standardized distance calculated for Xtrain.
res	matrix with distance and a standardized distance calculated for X.
cutoff	distance cutoff computed using a moment estimation of the parameters of a Chi-squared distribution for OD^2.

References

M. Hubert, P. J. Rousseeuw, K. Vanden Branden (2005). ROBPCA: a new approach to robust principal components analysis. Technometrics, 47, 64-79.

Nomikos, P., MacGregor, J.F., 1995. Multivariate SPC Charts for Monitoring Batch Processes. null 37, 41-59. https://doi.org/10.1080/00401706.1995.10485888

Pomerantsev, A.L., 2008. Acceptance areas for multivariate classification derived by projection methods. Journal of Chemometrics 22, 601-609. https://doi.org/10.1002/cem.1147

K. Vanden Branden, M. Hubert (2005). Robuts classification in high dimension based on the SIMCA method. Chem. Lab. Int. Syst, 79, 10-21.

K. Varmuza, P. Filzmoser (2009). Introduction to multivariate statistical analysis in chemometrics. CRC Press, Boca Raton.

Examples

```
n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Xtest <- Xtrain[1:3, , drop = FALSE]

nlv <- 3
fm <- pcasvd(Xtrain, nlv = nlv)
odis(fm, Xtrain)
odis(fm, Xtrain, nlv = 2)
odis(fm, Xtrain, X = Xtest, nlv = 2)
```

---

orthog *Orthogonalization of a matrix to another matrix*

Description

Function orthog orthogonalizes a matrix Y to a matrix X. The row observations can be weighted.

The function uses function lm.

Usage

```
orthog(X, Y, weights = NULL)
```

Arguments

X	A nxp matrix or data frame.
Y	A nxq matrix or data frame to orthogonalize to X.
weights	A vector of length n defining a priori weights to apply to the observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).

Value

Y	The Y matrix orthogonalized to X.
b	The regression coefficients used for orthogonalization.

Examples

```
n <- 8 ; p <- 3
set.seed(1)
X <- matrix(rnorm(n * p, mean = 10), ncol = p, byrow = TRUE)
Y <- matrix(rnorm(n * 2, mean = 10), ncol = 2, byrow = TRUE)
colnames(Y) <- c("y1", "y2")
set.seed(NULL)
X
Y

res <- orthog(X, Y)
res$Y
crossprod(res$Y, X)
res$b

# Same as:
fm <- lm(Y ~ X)
Y - fm$fitted.values
fm$coef

#### WITH WEIGHTS

w <- 1:n
fm <- lm(Y ~ X, weights = w)
Y - fm$fitted.values
fm$coef

res <- orthog(X, Y, weights = w)
res$Y
t(res$Y)
res$b
```

ozone *ozone*

Description

Los Angeles ozone pollution data in 1976 (sources: Breiman & Friedman 1985, Leisch & Dimitriadou 2020).

Usage

```
data(ozone)
```

Format

A list with 1 component: the matrix `X` with 366 observations, 13 variables. The variable to predict is V4.

`V1` Month: 1 = January, ..., 12 = December

`V2` Day of month

`V3` Day of week: 1 = Monday, ..., 7 = Sunday

`V4` Daily maximum one-hour-average ozone reading

`V5` 500 millibar pressure height (m) measured at Vandenberg AFB

`V6` Wind speed (mph) at Los Angeles International Airport (LAX)

`V7` Humidity (%) at LAX

`V8` Temperature (degrees F) measured at Sandburg, CA

`V9` Temperature (degrees F) measured at El Monte, CA

`V10` Inversion base height (feet) at LAX

`V11` Pressure gradient (mm Hg) from LAX to Daggett, CA

`V12` Inversion base temperature (degrees F) at LAX

`V13` Visibility (miles) measured at LAX

Source

Breiman L., Friedman J.H. 1985. Estimating optimal transformations for multiple regression and correlation, JASA, 80, pp. 580-598.

Leisch, F. and Dimitriadou, E. (2010). mlbench: Machine Learning Benchmark Problems. R package version 1.1-6. https://cran.r-project.org/.

Examples

```
data(ozone)

z <- ozone$X
head(z)

plotxna(z)
```

pcasvd *PCA algorithms*

Description

Algorithms fitting a centered weighted PCA of a matrix X.

Noting D a (n, n) diagonal matrix of weights for the observations (rows of X), the functions consist in:

- pcasvd: SVD factorization of $D^(1/2) * X$, using function svd.

- pcaeigen: Eigen factorization of $X' * D * X$, using function eigen.

- pcaeigenk: Eigen factorization of $D^(1/2) * X * X'D^(1/2)$, using function eigen. This is the "kernel cross-product trick" version of the PCA algorithm (Wu et al. 1997). For wide matrices ($n << p$) and n not too large, this algorithm can be much faster than the others.

- pcanipals: Eigen factorization of $X' * D * X$ using NIPALS.

- pcanipalsna: Eigen factorization of $X' * D * X$ using NIPALS allowing missing data in X.

- pcasph: Robust spherical PCA (Locantore et al. 1990, Maronna 2005, Daszykowski et al. 2007).

Function pcanipalsna accepts missing data (NAs) in X, unlike the other functions. The part of pcanipalsna accounting specifically for missing missing data is based on the efficient code of K. Wright in the R package nipals (https://cran.r-project.org/web/packages/nipals/index.html).

Gram-Schmidt orthogonalization in the NIPALS algorithm

The PCA NIPALS is known to generate a loss of orthogonality of the PCs (due to the accumulation of rounding errors in the successive iterations), particularly for large matrices or with high degrees of column collinearity.

With missing data, orthogonality of loadings is not satisfied neither.

An approach for coming back to orthogonality (PCs and loadings) is the iterative classical Gram-Schmidt orthogonalization (Lingen 2000, Andrecut 2009, and vignette of R package nipals), referred to as the iterative CGS. It consists in adding a CGS orthorgonalization step in each iteration of the PCs and loadings calculations.

For the case with missing data, the iterative CGS does not insure that the orthogonalized PCs are centered.

Auxiliary function

transform Calculates the PCs for any new matrix X from the model.

summary returns summary information for the model.

Usage

```
pcasvd(X, weights = NULL, nlv)

pcaeigen(X, weights = NULL, nlv)

pcaeigenk(X,weights = NULL,  nlv)
```

```
pcanipals(X, weights = NULL, nlv,
    gs = TRUE,
    tol = .Machine$double.eps^0.5, maxit = 200)

pcanipalsna(X, nlv,
    gs = TRUE,
    tol = .Machine$double.eps^0.5, maxit = 200)

pcasph(X, weights = NULL, nlv)

## S3 method for class 'Pca'
transform(object, X, ..., nlv = NULL)

## S3 method for class 'Pca'
summary(object, X, ...)
```

Arguments

X	For the main functions and auxiliary function summary: Training X-data (n, p). — For the other auxiliary functions: New X-data (m, p) to consider.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	The number of PCs to calculate.
object	A fitted model, output of a call to the main functions.
...	Optional arguments.

Specific for the NIPALS algorithm

gs	Logical indicating if a Gram-Schmidt orthogonalization is implemented or not (default to TRUE).
tol	Tolerance for testing convergence of the NIPALS iterations for each PC.
maxit	Maximum number of NIPALS iterations for each PC.

Value

A list of outputs, such as:

T	The score matrix (n, nlv).
P	The loadings matrix (p, nlv).
R	The projection matrix (= P) (p, nlv).
sv	The singular values $(min(n, p), 1)$ except for NIPALS = $(nlv, 1)$.
eig	The eigenvalues (= sv^2) $(min(n, p), 1)$ except for NIPALS = $(nlv, 1)$.
xmeans	The centering vector of X $(p, 1)$.
niter	Numbers of iterations of the NIPALS.

conv	Logical indicating if the NIPALS converged before reaching the maximal number of iterations.

For transform.Pca: X-scores matrix for new Xlist-data.

For summary.Pca:

explvarx	matrix of explained variances.
contr.ind	observation contributions.
contr.var	variable contributions.
coord.var	variable coordinates.
cor.circle	variable coordinates on the correlation circle.

References

Andrecut, M., 2009. Parallel GPU Implementation of Iterative PCA Algorithms. Journal of Computational Biology 16, 1593-1599. https://doi.org/10.1089/cmb.2008.0221

Gabriel, R. K., 2002. Le biplot - Outil d\'exploration de données multidimensionnelles. Journal de la Société Française de la Statistique, 143, 5-55.

Lingen, F.J., 2000. Efficient Gram-Schmidt orthonormalisation on parallel computers. Communications in Numerical Methods in Engineering 16, 57-66. https://doi.org/10.1002/(SICI)1099-0887(200001)16:1<57::AID-CNM320>3.0.CO;2-I

Tenenhaus, M., 1998. La régression PLS: théorie et pratique. Editions Technip, Paris, France.

Wright, K., 2018. Package nipals: Principal Components Analysis using NIPALS with Gram-Schmidt Orthogonalization. https://cran.r-project.org/

Wu, W., Massart, D.L., de Jong, S., 1997. The kernel PCA algorithms for wide data. Part I: Theory and algorithms. Chemometrics and Intelligent Laboratory Systems 36, 165-172. https://doi.org/10.1016/S0169-7439(97)00010-5

For Spherical PCA:

Daszykowski, M., Kaczmarek, K., Vander Heyden, Y., Walczak, B., 2007. Robust statistics in data analysis - A review. Chemometrics and Intelligent Laboratory Systems 85, 203-219. https://doi.org/10.1016/j.chemolab.2006

Locantore N., Marron J.S., Simpson D.G., Tripoli N., Zhang J.T., Cohen K.L. Robust principal component analysis for functional data, Test 8 (1999) 1-7

Maronna, R., 2005. Principal components and orthogonal regression based on robust scales, Technometrics, 47:3, 264-273, DOI: 10.1198/004017005000000166

Examples

```
n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), nrow = n)
s <- c(3, 4, 7, 10, 11, 15, 21:24)
zX <- replace(Xtrain, s, NA)
Xtrain
zX
m <- 2
Xtest <- matrix(rnorm(m * p), nrow = m)
```

```
pcasvd(Xtrain, nlv = 3)
pcaeigen(Xtrain, nlv = 3)
pcaeigenk(Xtrain, nlv = 3)
pcanipals(Xtrain, nlv = 3)
pcanipalsna(Xtrain, nlv = 3)
pcanipalsna(zX, nlv = 3)

fm <- pcaeigen(Xtrain, nlv = 3)
fm$T
transform(fm, Xtest)
transform(fm, Xtest, nlv = 2)

pcaeigen(Xtrain, nlv = 3)$T
pcaeigen(Xtrain, nlv = 3, weights = 1:n)$T


Ttrain <- fm$T
Ttest <- transform(fm, Xtest)
T <- rbind(Ttrain, Ttest)
group <- c(rep("Training", nrow(Ttrain)), rep("Test", nrow(Ttest)))
i <- 1
plotxy(T[, i:(i+1)], group = group, pch = 16, zeroes = TRUE, cex = 1.3, main = "scores")

plotxy(fm$P, zeroes = TRUE, label = TRUE, cex = 2, col = "red3", main ="loadings")

summary(fm, Xtrain)
res <- summary(fm, Xtrain)
plotxy(res$cor.circle, zeroes = TRUE, label = TRUE, cex = 2, col = "red3",
    circle = TRUE, ylim = c(-1, 1))
```

`pinv` *Moore-Penrose pseudo-inverse of a matrix*

Description

Calculation of the Moore-Penrose (MP) pseudo-inverse of a matrix X.

Usage

```
pinv(X, tol = sqrt(.Machine$double.eps))
```

Arguments

`X`	X-data (n, p).
`tol`	A relative tolerance to detect zero singular values.

Value

Xplus	The MP pseudo-inverse.
sv	singular values.

Examples

```
n <- 7 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)

pinv(X)

tcrossprod(pinv(X)$Xplus, t(y))
lm(y ~ X - 1)
```

---

plotjit *Jittered plot*

Description

Plot comparing classes with jittered points (random noise is added to the x-axis values for avoiding overplotting).

Usage

```
plotjit(x, y, group = NULL,
  jit = 1, col = NULL, alpha.f = .8,
  legend = TRUE, legend.title = NULL, ncol = 1, med = TRUE,
  ...)
```

Arguments

x	A vector of length n defining the class membership of the observations (x-axis).
y	A vector of length n defining the variable to plot (y-axis).
group	A vector of length n defining groups of observations to be plotted with different colors (default to NULL).
jit	Scalar defining the jittering magnitude. Default to 1.
alpha.f	Scalar modifying the opacity of the points in the graphics; typically in [0,1]. See adjustcolor.
col	A color, or a vector of colors (of length equal to the number of classes or groups), defining the color(s) of the points.
legend	Only if there are groups. Logical indicationg is a legend is drawn for groups (Default to FALSE).
legend.title	Character string indicationg a title for the legend.

ncol	Number of columns drawn in the legend box.
med	Logical. If TRUE (default), the median of each class is plotted.
...	Other arguments to pass in plot.

Value

Jittered plot.

Examples

```
n <- 500
x <- c(rep("A", n), rep("B", n))
y <- c(rnorm(n), rnorm(n, mean = 5, sd = 3))
group <- sample(1:2, size = 2 * n, replace = TRUE)

plotjit(x, y, pch = 16, jit = .5, alpha.f = .5)

plotjit(x, y, pch = 16, jit = .5, alpha.f = .5,
  group = group)
```

---

plotscore *Plotting errors rates*

Description

Plotting scores of prediction errors (error rates).

Usage

```
plotscore(x, y, group = NULL,
    col = NULL, steplab = 2, legend = TRUE, legend.title = NULL, ncol = 1, ...)
```

Arguments

x	Horizontal axis vector (n).
y	Vertical axis vector (n)
group	Groups of data (n) to be plotted with different colors.
col	A color, or a vector of colors (of length equal to the number of groups), defining the color(s) of the groups.
steplab	A step for the horizontal axis. Can be NULL (automatic step).
legend	Only if there are groups. Logical indicationg is a legend is drawn for groups (Default to FALSE).
legend.title	Character string indicationg a title for the legend.
ncol	Number of columns drawn in the legend box.
...	Other arguments to pass in function plot.

Value

A plot.

Examples

```
n <- 50 ; p <- 20
Xtrain <- matrix(rnorm(n * p), ncol = p, byrow = TRUE)
ytrain <- rnorm(n)
Ytrain <- cbind(ytrain, 10 * rnorm(n))
m <- 3
Xtest <- Xtrain[1:m, ]
Ytest <- Ytrain[1:m, ] ; ytest <- Ytest[, 1]

nlv <- 15
res <- gridscorelv(
    Xtrain, ytrain, Xtest, ytest,
    score = msep,
    fun = plskern,
    nlv = 0:nlv, verb = TRUE
    )
plotscore(res$nlv, res$y1,
          main = "MSEP", xlab = "Nb. LVs", ylab = "Value")

nlvdis <- 5
h <- c(1, Inf)
k <- c(10, 20)
nlv <- 15
pars <- mpars(nlvdis = nlvdis, diss = "mahal",
              h = h, k = k)
res <- gridscorelv(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = lwplsr,
    nlv = 0:nlv, pars = pars, verb = TRUE)
headm(res)
group <- paste("h=", res$h, " k=", res$k, sep = "")
plotscore(res$nlv, res$y1, group = group,
          main = "MSEP", xlab = "Nb. LVs", ylab = "Value")
```

---

plotsp *Plotting spectra*

Description

`plotsp` plots lines corresponding to the row observations (e.g. spectra) of a data set.

`plotsp1` plots only one observation per plot (e.g. spectrum by spectrum) by scrolling the rows. After running a `plotsp1` command, the plots are printed successively by pushing the R console "entry button", and stopped by entering any character in the R console.

Usage

```
plotsp(X,
  type = "l", col = NULL, zeroes = FALSE, labels = FALSE,
  add = FALSE,
  ...)

plotsp1(X, col = NULL, zeroes = FALSE, ...)
```

Arguments

X	Data (n, p) to plot.
type	1-character string giving the type of plot desired. Default value to "l" (lines). See plot.default for other options.
col	A color, or a vector of colors (of length n), defining the color(s) of the lines representing the rows.
zeroes	Logical indicationg if an horizontal line is drawn at coordonates (0, 0) (Default to FALSE).
labels	Logical indicating if the row names of X are plotted (default to FALSE).
add	Logical defining if the frame of the plot is plotted (add = FALSE; default) or not (add = TRUE). This allows to add new observations to a plot without red-building the frame.
...	Other arguments to pass in functions plot or lines

.

Value

A plot (see examples).

Note

For the first example, see ?hcl.colors and ?hcl.pals, and try with col <- hcl.colors(n = n, alpha = 1, rev = FALSE, palette = "Green-Orange") col <- terrain.colors(n, rev = FALSE) col <- rainbow(n, rev = FALSE, alpha = .2)

The second example is with plotsp1 (Scrolling plot of PCA loadings). After running the code, type Enter in the R console for starting the scrolling, and type any character in the R console

Examples

```
## EXAMPLE 1

data(cassav)

X <- cassav$Xtest
n <- nrow(X)

plotsp(X)
```

```
plotsp(X, col = "grey")
plotsp(X, col = "lightblue",
  xlim = c(500, 1500),
  xlab = "Wavelength (nm)", ylab = "Absorbance")

col <- hcl.colors(n = n, alpha = 1, rev = FALSE, palette = "Grays")
plotsp(X, col = col)

plotsp(X, col = "grey")
plotsp(X[23, , drop = FALSE], lwd = 2, add = TRUE)
plotsp(X[c(23, 16), ], lwd = 2, add = TRUE)

plotsp(X[5, , drop = FALSE], labels = TRUE)

plotsp(X[c(5, 61), ], labels = TRUE)

col <- hcl.colors(n = n, alpha = 1, rev = FALSE, palette = "Grays")
plotsp(X, col = col)
plotsp(X[5, , drop = FALSE], col = "red", lwd = 2, add = TRUE, labels = TRUE)

## EXAMPLE 2 (Scrolling plot of PCA loadings)

data(cassav)
X <- cassav$Xtest
fm <- pcaeigenk(X, nlv = 20)
P <- fm$P

plotsp1(t(P), ylab = "Value")
```

`plotxna` *Plotting Missing Data in a Matrix*

Description

Plot the location of missing data in a matrix.

Usage

```
plotxna(X, pch = 16, col = "red", grid = FALSE, asp = 0, ...)
```

Arguments

`X`	A data set (nxp).
`pch`	Type of point. See `points`.
`col`	A color defining the color of the points.
`grid`	Logical. If `TRUE`, a grid is plotted for representing the matrix rows an columns. Default to `FALSE`.

asp	Scalar. Giving the aspect ratio y/x. The value asp = 0 is the default in plot.default (no constraints on the ratio). See plot.default.
...	Other arguments to pass in functions plot.

Value

A plot.

Examples

```
data(octane)
X <- octane$X
n <- nrow(X)
p <- ncol(X)
N <- n * p

s <- sample(1:N, size = 50)
zX <- replace(X, s, NA)
plotxna(zX)
plotxna(zX, grid = TRUE, asp = 0)
```

---

plotxy *2-d scatter plot*

Description

2-dimension scatter plot.

Usage

```
plotxy(X, group = NULL,
    asp = 0, col = NULL, alpha.f = .8,
    zeroes = FALSE, circle = FALSE, ellipse = FALSE,
    labels = FALSE,
    legend = TRUE, legend.title = NULL, ncol = 1,
    ...)
```

Arguments

X	Data (n, p) to plot. If $p > 2$, only the first two columns are considered.
group	Groups of observations (n) to be plotted with different colors (default to NULL).
asp	Scalar. Giving the aspect ratio y/x. The value asp = 0 is the default in plot.default (no constraints on the ratio). See plot.default.
col	A color, or a vector of colors (of length equal to the number of groups), defining the color(s) of the groups.

alpha.f	Scalar modifying the opacity of the points in the graphics; typically in [0,1]. See adjustcolor.
zeroes	Logical indicationg if an horizontal and vertical lines are drawn at coordonates (0, 0) (Default to FALSE).
circle	Not still working. Logical indicating if a correlation circle is plotted (default to FALSE).
ellipse	Logical indicating if a Gaussian ellipse is plotted (default to FALSE). If there are groups, an ellipse is drawn for each group.
labels	Logical indicating if the row names of X (instead of points) are plotted (default to FALSE).
legend	Only if there are groups. Logical indicationg is a legend is drawn for groups (Default to FALSE).
legend.title	Character string indicationg a title for the legend.
ncol	Number of columns drawn in the legend box.
...	Other arguments to pass in functions plot, points, axis and text.

Value

A plot.

Examples

```
n <- 50 ; p <- 10
Xtrain <- matrix(rnorm(n * p), ncol = p)
Xtest <- Xtrain[1:5, ] + .4

fm <- pcaeigen(Xtrain, nlv = 5)
Ttrain <- fm$T
Ttest <- transform(fm, Xtest)
T <- rbind(Ttrain, Ttest)
group <- c(rep("Training", nrow(Ttrain)), rep("Test", nrow(Ttest)))
i <- 1
plotxy(T[, i:(i+1)], group = group,
       pch = 16, zeroes = TRUE,
       main = "PCA")

plotxy(T[, i:(i+1)], group = group,
       pch = 16, zeroes = TRUE, asp = 1,
       main = "PCA")
```

plskern *PLSR algorithms*

Description

Algorithms fitting a PLS1 or PLS2 model between dependent variables X and responses Y.

- plskern: "Improved kernel algorithm #1" proposed by Dayal and MacGregor (1997). This algorithm is stable and fast (Andersson 2009), and returns the same results as the NIPALS.

- plsnipals: NIPALS algorithm (e.g. Tenenhaus 1998, Wold 2002). In the function, the usual PLS2 NIPALS iterative is replaced by a direct calculation of the weights vector w by SVD decomposition of matrix $X'Y$ (Hoskuldsson 1988 p.213).

- plsrannar: Kernel algorithm proposed by Rannar et al. (1994) for "wide" matrices, i.e. with low number of rows and very large number of columns (p » n; e.g. p = 20000). In such a situation, this algorithm is faster than the others (but it becomes much slower in other situations). If the algorithm converges, it returns the same results as the NIPALS (Note: discrepancies can be observed if too many PLS components are requested compared to the low number of observations).

For weighted versions, see for instance Schaal et al. 2002, Siccard & Sabatier 2006, Kim et al. 2011 and Lesnoff et al. 2020.

Auxiliary functions

transform Calculates the LVs for any new matrix X from the model.

summary returns summary information for the model.

coef Calculates b-coefficients from the model.

predict Calculates the predictions for any new matrix X from the model.

Usage

```
plskern(X, Y, weights = NULL, nlv,
Xscaling = c("none", "pareto", "sd")[1], Yscaling = c("none", "pareto", "sd")[1])

plsnipals(X, Y, weights = NULL, nlv,
Xscaling = c("none", "pareto", "sd")[1], Yscaling = c("none", "pareto", "sd")[1])

plsrannar(X, Y, weights = NULL, nlv,
Xscaling = c("none", "pareto", "sd")[1], Yscaling = c("none", "pareto", "sd")[1])

## S3 method for class 'Plsr'
transform(object, X, ..., nlv = NULL)

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

## S3 method for class 'Plsr'
coef(object, ..., nlv = NULL)
```

```
## S3 method for class 'Plsr'
predict(object, X, ..., nlv = NULL)
```

Arguments

X	For the main functions: Training X-data (n, p). — For the auxiliary functions: Training X-data (n, p).
Y	Training Y-data (n, q).
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	For the main functions: The number(s) of LVs to calculate. — For the auxiliary functions: The number(s) of LVs to consider.
Xscaling	X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	Y variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For plskern, plsnipals, plsrannar: A list of outputs, such as

T	The X-score matrix (n, nlv).
P	The X-loadings matrix (p, nlv).
W	The X-loading weights matrix (p, nlv).
C	The Y-loading weights matrix (C = t(Beta), where Beta is the scores regression coefficients matrix).
R	The PLS projection matrix (p, nlv).
xmeans	The centering vector of X $(p, 1)$.
ymeans	The centering vector of Y $(q, 1)$.
xscales	The vector of X variable standard deviations $(p, 1)$.
yscales	The vector of Y variable standard deviations $(q, 1)$.
weights	Weights applied to the training observations.
TT	the X-score normalization factor.
U	intermediate output.

For transform.Plsr: X-scores matrix for new X-data.

For summary.Plsr:

explvarx	matrix of explained variances.

For `coef.Plsr`:

`int`	matrix (1,nlv) with the intercepts
`B`	matrix (n,nlv) with the coefficients

For `predict.Plsr`:

`pred`	A list of matrices (m, q) with the Y predicted values for the new X-data

References

Andersson, M., 2009. A comparison of nine PLS1 algorithms. Journal of Chemometrics 23, 518-529.

Dayal, B.S., MacGregor, J.F., 1997. Improved PLS algorithms. Journal of Chemometrics 11, 73-85.

Hoskuldsson, A., 1988. PLS regression methods. Journal of Chemometrics 2, 211-228. https://doi.org/10.1002/cem.1180020

Kim, S., Kano, M., Nakagawa, H., Hasebe, S., 2011. Estimation of active pharmaceutical ingredients content using locally weighted partial least squares and statistical wavelength selection. Int. J. Pharm., 421, 269-274.

Lesnoff, M., Metz, M., Roger, J.M., 2020. Comparison of locally weighted PLS strategies for regression and discrimination on agronomic NIR Data. Journal of Chemometrics. e3209. https://onlinelibrary.wiley.com/doi/ab

Rannar, S., Lindgren, F., Geladi, P., 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. https://doi.org/10.1002/cem.1180080204

Schaal, S., Atkeson, C., Vijayamakumar, S. 2002. Scalable techniques from nonparametric statistics for the real time robot learning. Applied Intell., 17, 49-60.

Sicard, E. Sabatier, R., 2006. Theoretical framework for local PLS1 regression and application to a rainfall data set. Comput. Stat. Data Anal., 51, 1393-1410.

Tenenhaus, M., 1998. La régression PLS: théorie et pratique. Editions Technip, Paris, France.

Wold, S., Sjostrom, M., Eriksson, l., 2001. PLS-regression: a basic tool for chemometrics. Chem. Int. Lab. Syst., 58, 109-130.

See Also

`plsr_plsda_allsteps` function to help determine the optimal number of latent variables, perform a permutation test, calculate model parameters and predict new observations.

Examples

```
n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

nlv <- 3
```

```
plskern(Xtrain, Ytrain, Xscaling = "sd", nlv = nlv)
plsnipals(Xtrain, Ytrain, Xscaling = "sd", nlv = nlv)
plsrannar(Xtrain, Ytrain, Xscaling = "sd", nlv = nlv)

plskern(Xtrain, Ytrain, Xscaling = "none", nlv = nlv)
plskern(Xtrain, Ytrain, nlv = nlv)$T
plskern(Xtrain, Ytrain, nlv = nlv, weights = 1:n)$T

fm <- plskern(Xtrain, Ytrain, nlv = nlv)
coef(fm)
coef(fm, nlv = 0)
coef(fm, nlv = 1)

fm$T
transform(fm, Xtest)
transform(fm, Xtest, nlv = 1)

summary(fm, Xtrain)

predict(fm, Xtest)
predict(fm, Xtest, nlv = 0:3)

pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)
```

---

plsr_agg *PLSR with aggregation of latent variables*

Description

Ensemblist approach where the predictions are calculated by averaging the predictions of PLSR models (plskern) built with different numbers of latent variables (LVs).

For instance, if argument nlv is set to nlv = "5:10", the prediction for a new observation is the average (without weighting) of the predictions returned by the models with 5 LVS, 6 LVs, ... 10 LVs.

Usage

```
plsr_agg(X, Y, weights = NULL, nlv)

## S3 method for class 'Plsr_agg'
predict(object, X, ...)
```

Arguments

For plsr_agg:

X	For the main function: Training X-data (n, p). — For the auxiliary function: New X-data (m, p) to consider.
Y	Training Y-data (n, q).
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	A character string such as "5:20" defining the range of the numbers of LVs to consider (here: the models with nb LVS = 5, 6, ..., 20 are averaged). Syntax such as "10" is also allowed (here: correponds to the single model with 10 LVs).
object	For the auxiliary function: A fitted model, output of a call to the main functions.
...	For the auxiliary function: Optional arguments. Not used.

Value

For plsr_agg:

fm	list contaning the model: (fm)=(T): X-scores matrix; (P): X-loading matrix;(R): The PLS projection matrix (p,nlv); (W): X-loading weights matrix ;(C): The Y-loading weights matrix; (TT): the X-score normalization factor; (xmeans): the centering vector of X (p,1); (ymeans): the centering vector of Y (q,1); (weights): vector of observation weights; (U): intermediate output.
nlv	range of the numbers of LVs considered

For predict.Plsr_agg:

pred	Final predictions (after aggregation)
predlv	Intermediate predictions (Per nb. LVs)

Note

In the example, zfm is the maximal PLSR model, and there is no sense to use gridscorelv or gridcvlv instead of gridscore or gridcv.

Examples

```
n <- 20 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

nlv <- "1:3"

fm <- plsr_agg(Xtrain, ytrain, nlv = nlv)
names(fm)

zfm <- fm$fm
class(zfm)
```

```
names(zfm)
summary(zfm, Xtrain)


res <- predict(fm, Xtest)
names(res)

res$pred
msep(res$pred, ytest)

res$predlv

pars <- mpars(nlv = c("1:3", "2:5"))
pars
res <- gridscore(
    Xtrain, Ytrain, Xtest, Ytest,
    score = msep,
    fun = plsr_agg,
    pars = pars)
res

K = 3
segm <- segmkf(n = n, K = K, nrep = 1)
segm
res <- gridcv(
    Xtrain, Ytrain,
    segm, score = msep,
    fun = plsr_agg,
    pars = pars,
    verb = TRUE)
res
```

`plsr_plsda_allsteps` *PLSR or PLSDA analysis steps*

Description

Help determine the optimal number of latent variables by cross-validation, perform a permutation test, calculate model parameters and predict new observations, for plsr (`plskern`), plsrda (`plsrda`), plslda (`plslda`) or plsqda (`plsqda`) models.

Usage

```
plsr_plsda_allsteps(X, Xname = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y, Yscaling = c("none","pareto","sd")[1], weights = NULL,
                    newX = NULL, newXname = NULL,

                    method = c("plsr", "plsrda","plslda","plsqda")[1],
                    prior = c("unif", "prop")[1],
```

```
step = c("nlvtest","permutation","model","prediction")[1],
nlv,
modeloutput = c("scores","loadings","coef","vip"),

cvmethod = c("kfolds","loo")[1],
nbrep = 30,
seed = 123,
samplingk = NULL,
nfolds = 10,
npermut = 30,

criterion = c("err","rmse")[1],
selection = c("localmin","globalmin","1std")[1],

import = c("R","ChemFlow","W4M")[1],
outputfilename = NULL)
```

Arguments

X	Training X-data (n, p).
Xname	name of the X-matrix
Xscaling	X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Y	Training Y-data (n, q) for plsr models, and $(n, 1)$ for plsrda, plslda or plsqda models.
Yscaling	Y variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
newX	New X-data (m, p) to consider.
newXname	name of the newX-matrix
method	method to apply among "plsr", "plsrda","plslda","plsqda"
prior	for plslda or plsqda models : The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
step	step of the analysis among "nlvtest" (cross-validation to help determine the optimal number of latent variables), "permutation" (permutation test),"model" (model calculation),"prediction" (prediction of newX-data or X-data if any))
nlv	number of latent variables to test if step is "nlvtest"; number of latent variables of the model if step is not "nlvtest".

modeloutput	if step is "model": outputs among "scores", "loadings", "coef" (regression coefficients), "vip" (Variable Importance in Projection; the VIP calculation being based on the proportion of Y-variance explained by the components, as proposed by Mehmood et al (2012, 2020).)
cvmethod	if step is "nlvtest" or "permutation": "kfolds" for k-folds cross-validation, or "loo" for leave-one-out.
nbrep	if step is "nlvtest" and cvmethod is "kfolds": An integer, setting the number of CV repetitions. Default value is 30. Must me set to 1 if cvmethod is "loo"
seed	if step is "nlvtest" and cvmethod is "kfolds", or if step is "permutation: a numeric. Seed used for the repeated resampling
samplingk	A vector of length n. The elements are the values of a qualitative variable used for stratified partition creation. If NULL, the first observation is set in the first fold, the second observation in the second fold, etc...
nfolds	if cvmethod is "kfolds". An integer, setting the number of partitions to create. Default value is 10.
npermut	if step is "permutation": An integer, setting the number of Y-Block with permutated responses to create. Default value is 30.
criterion	if step is "nlvtest" or "permutation" and method is "plsrda", "plslda" or "plsqda": optimisation criterion among "rmse" and "err" (for classification error rate)))
selection	if step is "nlvtest": a character indicating the selection method to use to choose the optimal combination of components, among "localmin","globalmin","1std". If "localmin": the optimal combination corresponds to the first local minimum of the mean CV rmse or error rate. If "globalmin" : the optimal combination corresponds to the minimum mean CV rmse or error rate. If "1std" (one standard error rule) : it corresponds to the first combination after which the mean cross-validated rmse or error rate does not decrease significantly.
import	If "R", X and Y are in the global environment, and the observation names are in rownames. If "ChemFlow", X and Y are tabulated tables (.txt), and the observation names are in the first column. If "W4M", X and Y are tabulated tables (.txt), and the observation names are in the headers of X, and in the first column of Y.
outputfilename	character: If not NULL, name of the tabular file, in which the function outputs have to be written.)

Value

If step is "nlvtest": table with rmsecv or cross-validated classification error rates. The suggested optimal number of latent variables is indicated by the binary "optimum" variable.

If step is "permutation": table with the dissimilarity between the original and the permutated Y-block, and the rmsecv or cross-validated classification error rates obtained with the permutated Y-block by the model and the given number of latent variables.

If step is "model": tables of scores, loadings, regression coefficients, and vip values, depending of the "modeloutput" parameter.

If step is "prediction": table of predicted scores and predicted classes or values.

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
colnames(Xtrain) <- paste0("V",1:p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]

resnlvtestplsrda <- plsr_plsda_allsteps(X = Xtrain, Xname = NULL,
                    Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newX = Xtest, newXname = NULL,

                    method = c("plsr", "plsrda","plslda","plsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[1],
                    nlv = 5,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)

respermutationplsrda <- plsr_plsda_allsteps(X = Xtrain, Xname = NULL,
                    Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newX = Xtest, newXname = NULL,

                    method = c("plsr", "plsrda","plslda","plsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[2],
                    nlv = 2,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
```

```
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)

plotxy(respermutationplsrda, pch=16)
abline (h = respermutationplsrda[respermutationplsrda[,"permut_dyssimilarity"]==0,"res_permut"])

resmodelplsrda <- plsr_plsda_allsteps(X = Xtrain, Xname = NULL,
                    Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newX = Xtest, newXname = NULL,

                    method = c("plsr", "plsrda","plslda","plsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[3],
                    nlv = 2,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)


resmodelplsrda$scores
resmodelplsrda$loadings
resmodelplsrda$coef
resmodelplsrda$vip

respredictionplsrda <- plsr_plsda_allsteps(X = Xtrain, Xname = NULL,
                    Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newX = Xtest, newXname = NULL,

                    method = c("plsr", "plsrda","plslda","plsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[4],
                    nlv = 2,
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
```

```
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)
```

---

plsrda *PLSDA models*

Description

Discrimination (DA) based on PLS.

The training variable y (univariate class membership) is firstly transformed to a dummy table containing $nclas$ columns, where $nclas$ is the number of classes present in y. Each column is a dummy variable (0/1). Then, a PLS2 is implemented on the $X-$data and the dummy table, returning latent variables (LVs) that are used as dependent variables in a DA model.

- `plsrda`: Usual "PLSDA". A linear regression model predicts the Y-dummy table from the PLS2 LVs. This corresponds to the PLSR2 of the X-data and the Y-dummy table. For a given observation, the final prediction is the class corresponding to the dummy variable for which the prediction is the highest.

- `plslda` and `plsqda`: Probabilistic LDA and QDA are run over the PLS2 LVs, respectively.

Usage

```
plsrda(X, y, weights = NULL, nlv,
Xscaling = c("none","pareto","sd")[1], Yscaling = c("none","pareto","sd")[1])

plslda(X, y, weights = NULL, nlv, prior = c("unif", "prop"),
Xscaling = c("none","pareto","sd")[1], Yscaling = c("none","pareto","sd")[1])

plsqda(X, y, weights = NULL, nlv, prior = c("unif", "prop"),
Xscaling = c("none","pareto","sd")[1], Yscaling = c("none","pareto","sd")[1])

## S3 method for class 'Plsrda'
predict(object, X, ..., nlv = NULL)

## S3 method for class 'Plsprobda'
predict(object, X, ..., nlv = NULL)
```

Arguments

`X`	For the main functions: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.

y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
weights	Weights (n) to apply to the training observations for the PLS2. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
nlv	The number(s) of LVs to calculate.
prior	The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
Xscaling	X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	Y variable scaling, once converted to binary variables, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For plsrda, plslda, plsqda:

fm	list with the model: (T): X-scores matrix; (P): X-loading matrix;(R): The PLS projection matrix (p,nlv); (W): X-loading weights matrix ;(C): The Y-loading weights matrix; (TT): the X-score normalization factor; (xmeans): the centering vector of X (p,1); (ymeans): the centering vector of Y (q,1); (xscales): the scaling vector of X (p,1); (yscales): the scaling vector of Y (q,1); (weights): vector of observation weights; (U): intermediate output.
lev	classes
ni	number of observations in each class

For predict.Plsrda, predict.Plsprobda:

pred	predicted class for each observation
posterior	calculated probability of belonging to a class for each observation

Note

The first example concerns PLSDA, and the second one concerns PLS LDA. fm are PLS1 models, and zfm are PLS2 models to predict the disjunctive matrix.

See Also

plsr_plsda_allsteps function to help determine the optimal number of latent variables, perform a permutation test, calculate model parameters and predict new observations.

Examples

```
## EXAMPLE OF PLSDA

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]

nlv <- 5
fm <- plsrda(Xtrain, ytrain, Xscaling = "sd", nlv = nlv)
names(fm)

predict(fm, Xtest)
predict(fm, Xtest, nlv = 0:2)$pred

pred <- predict(fm, Xtest)$pred
err(pred, ytest)

zfm <- fm$fm
transform(zfm, Xtest)
transform(zfm, Xtest, nlv = 1)
summary(zfm, Xtrain)
coef(zfm)
coef(zfm, nlv = 0)
coef(zfm, nlv = 2)

## EXAMPLE OF PLS LDA

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)
Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]

nlv <- 5
fm <- plslda(Xtrain, ytrain, Xscaling = "sd", nlv = nlv)
predict(fm, Xtest)
predict(fm, Xtest, nlv = 1:2)$pred

zfm <- fm$fm[[1]]
class(zfm)
names(zfm)
summary(zfm, Xtrain)
transform(zfm, Xtest[1:2, ])
coef(zfm)
```

---

plsrda_agg *PLSDA with aggregation of latent variables*

Description

Ensemblist approach where the predictions are calculated by "averaging" the predictions of PLSDA models built with different numbers of latent variables (LVs).

For instance, if argument `nlv` is set to `nlv = "5:10"`, the prediction for a new observation is the most occurent level (vote) over the predictions returned by the models with 5 LVS, 6 LVs, ... 10 LVs.

- `plsrda_agg`: use `plsrda`.

- `plslda_agg`: use `plslda`.

- `plsqda_agg`: use `plsqda`.

Usage

```
plsrda_agg(X, y, weights = NULL, nlv)

plslda_agg(X, y, weights = NULL, nlv, prior = c("unif", "prop"))

plsqda_agg(X, y, weights = NULL, nlv, prior = c("unif", "prop"))

## S3 method for class 'Plsda_agg'
predict(object, X, ...)
```

Arguments

`X`	For the main functions: Training X-data (n, p). — For the auxiliary function: New X-data (m, p) to consider.
`y`	Training class membership (n). **Note:** If `y` is a factor, it is replaced by a character vector.
`weights`	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to `NULL` (weights are set to $1/n$).
`nlv`	A character string such as "5:20" defining the range of the numbers of LVs to consider (here: the models with nb LVS = 5, 6, ..., 20 are averaged). Syntax such as "10" is also allowed (here: correponds to the single model with 10 LVs).
`prior`	The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in `y`).
`object`	For the auxiliary function: A fitted model, output of a call to the main functions.
`...`	For the auxiliary function: Optional arguments. Not used.

Value

For `plsrda_agg`, `plslda_agg` and `plsqda_agg`:

`fm`	list contaning: the model((`fm`)=(`T`): X-scores matrix; (`P`): X-loading matrix;(`R`): The PLS projection matrix (p,nlv); (`W`): X-loading weights matrix ;(`C`): The Y-loading weights matrix; (`TT`): the X-score normalization factor; (`xmeans`):

	the centering vector of X (p,1); (ymeans): the centering vector of Y (q,1); (weights): vector of observation weights; (U): intermediate output), (lev):classes, (ni):number of observations in each class
nlv	range of the numbers of LVs considered

For predict.Plsda_agg:

pred	Final predictions (after aggregation)
predlv	Intermediate predictions (Per nb. LVs)

Note

the first example concerns PLSRDA-AGG, and the second one concerns PLSLDA-AGG.

Examples

```
## EXAMPLE OF PLSRDA-AGG

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10, 2), size = n, replace = TRUE)

m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]

nlv <- "2:5"
fm <- plsrda_agg(Xtrain, ytrain, nlv = nlv)
names(fm)
res <- predict(fm, Xtest)
names(res)
res$pred
err(res$pred, ytest)
res$predlv

pars <- mpars(nlv = c("1:3", "2:5"))
pars

res <- gridscore(
    Xtrain, ytrain, Xtest, ytest,
    score = err,
    fun = plsrda_agg,
    pars = pars)
res

segm <- segmkf(n = n, K = 3, nrep = 1)
res <- gridcv(
    Xtrain, ytrain,
    segm, score = err,
    fun = plslda_agg,
    pars = pars,
    verb = TRUE)
res
```

```
## EXAMPLE OF PLSLDA-AGG

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10, 2), size = n, replace = TRUE)
#ytrain <- sample(c("a", "10", "d"), size = n, replace = TRUE)
m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]

nlv <- "2:5"
fm <- plslda_agg(Xtrain, ytrain, nlv = nlv, prior = "unif")
names(fm)
res <- predict(fm, Xtest)
names(res)
res$pred
err(res$pred, ytest)
res$predlv

pars <- mpars(nlv = c("1:3", "2:5"), prior = c("unif", "prop"))
pars
res <- gridscore(
    Xtrain, ytrain, Xtest, ytest,
    score = err,
    fun = plslda_agg,
    pars = pars)
res

segm <- segmkf(n = n, K = 3, nrep = 1)
res <- gridcv(
    Xtrain, ytrain,
    segm, score = err,
    fun = plslda_agg,
    pars = pars,
    verb = TRUE)
res
```

`rmgap` *Removing vertical gaps in spectra*

Description

Remove the vertical gaps in spectra (rows of matrix X), e.g. for ASD. This is done by extrapolation from simple linear regressions computed on the left side of the gaps.

Usage

```
rmgap(X, indexcol, k = 5)
```

Arguments

X	A dataset.
indexcol	The column indexes corresponding to the gaps. For instance, if two gaps are observed between indexes 651-652 and between indexes 1451-1452, respectively, then indexcol = c(651, 1451).
k	The number of columns used on the left side of the gaps for fitting the linear regressions.

Value

The corrected data X.

Note

In the example, two gaps are at wavelengths 1000-1001 nm and 1800-1801 nm.

Examples

```
data(asdgap)
X <- asdgap$X

indexcol <- which(colnames(X) == "1000" | colnames(X) == "1800")
indexcol
plotsp(X, lwd = 1.5)
abline(v = as.numeric(colnames(X)[1]) + indexcol - 1, col = "lightgrey", lty = 3)

zX <- rmgap(X, indexcol = indexcol)
plotsp(zX, lwd = 1.5)
abline(v = as.numeric(colnames(zX)[1]) + indexcol - 1, col = "lightgrey", lty = 3)
```

---

rr *Linear Ridge Regression*

Description

Fitting linear ridge regression models (RR) (Hoerl & Kennard 1970, Hastie & Tibshirani 2004, Hastie et al 2009, Cule & De Iorio 2012) by SVD factorization.

Usage

```
rr(X, Y, weights = NULL, lb = 1e-2)

## S3 method for class 'Rr'
coef(object, ..., lb = NULL)

## S3 method for class 'Rr'
predict(object, X, ..., lb = NULL)
```

Arguments

X	For the main function: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
Y	Training Y-data (n, q).
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
lb	A value of regularization parameter $lambda$. If lb = 0, a pseudo-inverse is used.
object	For the auxiliary functions: A fitted model, output of a call to the main function.
...	— For the auxiliary functions: Optional arguments. Not used.

Value

For rr:

V	eigenvector matrix of the correlation matrix (n,n).
TtDY	intermediate output.
sv	singular values of the matrix $(1, n)$.
lb	value of regularization parameter $lambda$.
xmeans	the centering vector of X $(p, 1)$.
ymeans	the centering vector of Y $(q, 1)$.
weights	the weights vector of X-variables $(p, 1)$.

For coef.Rr:

int	matrix (1,nlv) with the intercepts
B	matrix (n,nlv) with the coefficients
df	model complexity (number of degrees of freedom)

For predict.Rr:

pred	A list of matrices (m, q) with the Y predicted values for the new X-data

References

Cule, E., De Iorio, M., 2012. A semi-automatic method to guide the choice of ridge parameter in ridge regression. arXiv:1205.0686.

Hastie, T., Tibshirani, R., 2004. Efficient quadratic regularization for expression arrays. Biostatistics 5, 329-340. https://doi.org/10.1093/biostatistics/kxh010

Hastie, T., Tibshirani, R., Friedman, J., 2009. The elements of statistical learning: data mining, inference, and prediction, 2nd ed. Springer, New York.

Hoerl, A.E., Kennard, R.W., 1970. Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics 12, 55-67. https://doi.org/10.1080/00401706.1970.10488634

Wu, W., Massart, D.L., de Jong, S., 1997. The kernel PCA algorithms for wide data. Part I: Theory and algorithms. Chemometrics and Intelligent Laboratory Systems 36, 165-172. https://doi.org/10.1016/S0169-7439(97)00010-5

Examples

```
n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

lb <- .1
fm <- rr(Xtrain, Ytrain, lb = lb)
coef(fm)
coef(fm, lb = .8)
predict(fm, Xtest)
predict(fm, Xtest, lb = c(0.1, .8))

pred <- predict(fm, Xtest)$pred
msep(pred, Ytest)
```

`rrda` *RR-DA models*

Description

Discrimination (DA) based on ridge regression (RR).

Usage

```
rrda(X, y, weights = NULL, lb = 1e-5)

## S3 method for class 'Rrda'
predict(object, X, ..., lb = NULL)
```

Arguments

For `rrda`:

`X`	For the main function: Training X-data (n, p). — For the auxiliary function: New X-data (m, p) to consider.
`y`	Training class membership (n). **Note:** If `y` is a factor, it is replaced by a character vector.
`weights`	Weights (n) to apply to the training observations for the PLS2. Internally, weights are "normalized" to sum to 1. Default to `NULL` (weights are set to $1/n$).
`lb`	A value of regularization parameter $lambda$. If `lb = 0`, a pseudo-inverse is used in the RR.
`object`	For the auxiliary function: A fitted model, output of a call to the main functions.
`...`	For the auxiliary function: Optional arguments. Not used.

Details

The training variable y (univariate class membership) is transformed to a dummy table containing $nclas$ columns, where $nclas$ is the number of classes present in y. Each column is a dummy variable (0/1). Then, a ridge regression (RR) is run on the $X-$data and the dummy table, returning predictions of the dummy variables. For a given observation, the final prediction is the class corresponding to the dummy variable for which the prediction is the highest.

Value

For `rrda`:

`fm`	List with the outputs of the RR ((`V`): eigenvector matrix of the correlation matrix (n,n); (`TtDY`): intermediate output; (`sv`): singular values of the matrix $(1, n)$; (`lb`): value of regularization parameter $lambda$; (`xmeans`): the centering vector of X $(p, 1)$; (`ymeans`): the centering vector of Y $(q, 1)$; (`weights`): the weights vector of X-variables $(p, 1)$.
`lev`	classes
`ni`	number of observations in each class

For `predict.Rrda`:

`pred`	matrix or list of matrices (if lb is a vector), with predicted class for each observation
`posterior`	matrix or list of matrices (if lb is a vector), calculated probability of belonging to a class for each observation

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]

lb <- 1
fm <- rrda(Xtrain, ytrain, lb = lb)
predict(fm, Xtest)

pred <- predict(fm, Xtest)$pred
err(pred, ytest)

predict(fm, Xtest, lb = 0:2)
predict(fm, Xtest, lb = 0)
```

`sampcla` *Within-class sampling*

Description

The function divides a datset in two sets, "train" vs "test", using a stratified sampling on defined classes.

If argument `y = NULL` (default), the sampling is random within each class. If not, the sampling is systematic (regular grid) within each class over the quantitative variable y.

Usage

```
sampcla(x, y = NULL, m)
```

Arguments

`x`	A vector (length m) defining the class membership of the observations.
`y`	A vector (length m) defining the quantitative variable for the systematic sampling. If `NULL` (default), the sampling is random within each class.
`m`	Either an integer defining the equal number of test observation(s) to select per class, or a vector of integers defining the numbers to select for each class. In the last case, vector `m` must have a length equal to the number of classes present in `x`, and be ordered in the same way as the ordered class membership.

Value

`train`	Indexes (i.e. position in x) of the selected observations, for the training set.
`test`	Indexes (i.e. position in x) of the selected observations, for the test set.
`lev`	classes
`ni`	number of observations in each class

Note

The second example is a representative stratified sampling from an unsupervised clustering.

References

Naes, T., 1987. The design of calibration in near infra-red reflectance analysis by clustering. Journal of Chemometrics 1, 121-134.

Examples

```
## EXAMPLE 1

x <- sample(c(1, 3, 4), size = 20, replace = TRUE)
table(x)

sampcla(x, m = 2)
s <- sampcla(x, m = 2)$test
x[s]

sampcla(x, m = c(1, 2, 1))
s <- sampcla(x, m = c(1, 2, 1))$test
x[s]

y <- rnorm(length(x))
sampcla(x, y, m = 2)
s <- sampcla(x, y, m = 2)$test
x[s]

## EXAMPLE 2

data(cassav)
X <- cassav$Xtrain
y <- cassav$ytrain
N <- nrow(X)

fm <- pcaeigenk(X, nlv = 10)
z <- stats::kmeans(x = fm$T, centers = 3, nstart = 25, iter.max = 50)
x <- z$cluster
z <- table(x)
z
p <- c(z) / N
p

psamp <- .20
m <- round(psamp * N * p)
m

random_sampling <- sampcla(x, m = m)
s <- random_sampling$test
table(x[s])

Systematic_sampling_for_y <- sampcla(x, y, m = m)
s <- Systematic_sampling_for_y$test
table(x[s])
```

---

sampdp *Duplex sampling*

Description

The function divides the data X in two sets, "train" vs "test", using the Duplex algorithm (Snee, 1977). The two sets are of equal size. If needed, the user can add $aposteriori$ the eventual remaining observations (not in "train" nor "test") to "train".

Usage

```
sampdp(X, k, diss = c("eucl", "mahal"))
```

Arguments

X	X-data (n, p) to be sampled.
k	An integer defining the number of training observations to select. Must be <= $n/2$.
diss	The type of dissimilarity used for selecting the observations in the algorithm. Possible values are "eucl" (default; Euclidean distance) or "mahal" (Mahalanobis distance).

Value

train	Indexes (i.e. row numbers in X) of the selected observations, for the training set.
test	Indexes (i.e. row numbers in X) of the selected observations, for the test set.
remain	Indexes (i.e., row numbers in X) of the remaining observations.

References

Kennard, R.W., Stone, L.A., 1969. Computer aided design of experiments. Technometrics, 11(1), 137-148.

Snee, R.D., 1977. Validation of Regression Models: Methods and Examples. Technometrics 19, 415-428. https://doi.org/10.1080/00401706.1977.10489581

Examples

```
n <- 10 ; p <- 3
X <- matrix(rnorm(n * p), ncol = p)

k <- 4
sampdp(X, k = k)
sampdp(X, k = k, diss = "mahal")
```

sampks *Kennard-Stone sampling*

Description

The function divides the data X in two sets, "train" vs "test", using the Kennard-Stone (KS) algorithm (Kennard & Stone, 1969). The two sets correspond to two different underlying probability distributions: set "train" has higher dispersion than set "test".

Usage

```
sampks(X, k, diss = c("eucl", "mahal"))
```

Arguments

X	X-data (n, p) to be sampled.
k	An integer defining the number of training observations to select.
diss	The type of dissimilarity used for selecting the observations in the algorithm. Possible values are "eucl" (default; Euclidean distance) or "mahal" (Mahalanobis distance).

Value

train	Indexes (i.e. row numbers in X) of the selected observations, for the training set.
test	Indexes (i.e. row numbers in X) of the selected observations, for the test set.

References

Kennard, R.W., Stone, L.A., 1969. Computer aided design of experiments. Technometrics, 11(1), 137-148.

Examples

```
n <- 10 ; p <- 3
X <- matrix(rnorm(n * p), ncol = p)

k <- 7
sampks(X, k = k)

n <- 10 ; k <- 25
X <- expand.grid(1:n, 1:n)
X <- X + rnorm(nrow(X) * ncol(X), 0, .1)
s <- sampks(X, k)$train
plot(X)
points(X[s, ], pch = 19, col = 2, cex = 1.5)
```

savgol *Savitzky-Golay smoothing*

Description

Smoothing by derivation, with a Savitzky-Golay filter, of the row observations (e.g. spectra) of a data set.

The function uses function sgolayfilt of package signal available on the CRAN.

Usage

```
savgol(X, m, n, p, ts = 1)
```

Arguments

X	X-data).
m	Derivation order.
n	Filter length (must be odd), i.e. the number of colums in X defining the filter window.
p	Polynomial order.
ts	Scaling factor (e.g. the absolute step between two columns in matrix X), see argument ts in function sgolayfilt. This has not impact on the form of the transformed output.

Value

A matrix of the transformed data.

Examples

```
X <- cassav$Xtest

m <- 1 ; n <- 11 ; p <- 2
Xp <- savgol(X, m, n, p)

oldpar <- par(mfrow = c(1, 1))
par(mfrow = c(1, 2))
plotsp(X, main = "Signal")
plotsp(Xp, main = "Corrected signal")
abline(h = 0, lty = 2, col = "grey")
par(oldpar)
```

scordis *Score distances (SD) in a PCA or PLS score space*

Description

scordis calculates the score distances (SD) for a PCA or PLS model. SD is the Mahalanobis distance of the projection of a row observation on the score plan to the center of the score space.

A distance cutoff is computed using a moment estimation of the parameters of a Chi-squared distrbution for SD^2 (see e.g. Pomerantzev 2008). In the function output, column dstand is a standardized distance defined as $SD/cutoff$. A value dstand > 1 can be considered as extreme.

The Winisi "GH" is also provided (usually considered as extreme if GH > 3).

Usage

```
scordis(
    object, X = NULL,
    nlv = NULL,
    rob = TRUE, alpha = .01
    )
```

Arguments

object	A fitted model, output of a call to a fitting function (for example from pcasvd, plskern,...).
X	New X-data.
nlv	Number of components (PCs or LVs) to consider.
rob	Logical. If TRUE, the moment estimation of the distance cutoff is robustified. This can be recommended after robust PCA or PLS on small data sets containing extreme values.
alpha	Risk-I level for defining the cutoff detecting extreme values.

Value

res.train	matrix with distances, standardized distances and Winisi "GH", for the training set.
res	matrix with distances, standardized distances and Winisi "GH", for new X-data if any.
cutoff	cutoff value

References

M. Hubert, P. J. Rousseeuw, K. Vanden Branden (2005). ROBPCA: a new approach to robust principal components analysis. Technometrics, 47, 64-79.

Pomerantsev, A.L., 2008. Acceptance areas for multivariate classification derived by projection methods. Journal of Chemometrics 22, 601-609. https://doi.org/10.1002/cem.1147

Examples

```
n <- 6 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Xtest <- Xtrain[1:3, , drop = FALSE]

nlv <- 3
fm <- pcasvd(Xtrain, nlv = nlv)
scordis(fm)
scordis(fm, nlv = 2)
scordis(fm, Xtest, nlv = 2)
```

---

segmkf *Segments for cross-validation*

Description

Build segments of observations for K-Fold or "test-set" cross-validation (CV).

The CV can eventually be randomly repeated. For each repetition:

- **K-fold CV** - Function `segmkf` returns the K segments.

- **Test-set CV** - Function `segmts` returns a segment (of a given length) randomly sampled in the dataset.

CV of blocks

Argument `y` allows sampling **blocks of observations** instead of observations. This can be required when there are repetitions in the data. In such a situation, CV should account for the repetition level (if not, the error rates are in general highly underestimated). For implementing such a CV, object `y` must be a a vector (n) defining the blocks, in the same order as in the data.

In any cases (`y = NULL` or not), the functions return a list of vector(s). Each vector contains the indexes of the observations defining the segment.

Usage

```
segmkf(n, y = NULL, K = 5,
    type = c("random", "consecutive", "interleaved"), nrep = 1)

segmts(n, y = NULL, m, nrep)
```

Arguments

`n`	The total number of row observations in the dataset. If `y = NULL`, the CV is implemented on `1:n`. If `y != NULL`, blocks of observations (defined in `y`) are sampled instead of observations (but indexes of observations are returned).
`y`	A vector (n) defining the blocks. Default to `NULL`.

K	For segmkf.The number of folds (i.e. segments) in the K-fold CV.
type	For segmkf.The type K-fold CV. Possible values are "random" (default), "consecutive" and "interleaved".
m	For segmts. If y = NULL, the number of observations in the segment. If not, the number of blocks in the segment.
nrep	The number of replications of the repeated CV. Default to nrep = 1.

Value

The segments (lists of indexes).

Examples

```
Kfold <- segmkf(n = 10, K = 3)

interleavedKfold <- segmkf(n = 10, K = 3, type = "interleaved")

LeaveOneOut <- segmkf(n = 10, K = 10)

RepeatedKfold <- segmkf(n = 10, K = 3, nrep = 2)

repeatedTestSet <- segmts(n = 10, m = 3, nrep = 5)

n <- 10
y <- rep(LETTERS[1:5], 2)
y

Kfold_withBlocks <- segmkf(n = n, y = y, K = 3, nrep = 1)
z <- Kfold_withBlocks
z
y[z$rep1$segm1]
y[z$rep1$segm2]
y[z$rep1$segm3]

TestSet_withBlocks <- segmts(n = n, y = y, m = 3, nrep = 1)
z <- TestSet_withBlocks
z
y[z$rep1$segm1]
```

---

selwold *Heuristic selection of the dimension of a latent variable model with the Wold's criterion*

Description

The function helps selecting the dimensionnality of latent variable (LV) models (e.g. PLSR) using the "Wold criterion".

The criterion is the "precision gain ratio" $R = 1 - r(a+1)/r(a)$ where r is an observed error rate quantifying the model performance (msep, classification error rate, etc.) and a the model dimensionnality (= nb. LVs). It can also represent other indicators such as the eigenvalues of a PCA.

R is the relative gain in efficiency after a new LV is added to the model. The iterations continue until R becomes lower than a threshold value $alpha$. By default and only as an indication, the default $alpha = .05$ is set in the function, but the user should set any other value depending on his data and parcimony objective.

In the original article, Wold (1978; see also Bro et al. 2008) used the ratio of **cross-validated** over **training** residual sums of squares, i.e. PRESS over SSR. Instead, `selwold` compares values of consistent nature (the successive values in the input vector r), e.g. PRESS only . For instance, r was set to PRESS values in Li et al. (2002) and Andries et al. (2011), which is equivalent to the "punish factor" described in Westad & Martens (2000).

The ratio R is often erratic, making difficult the dimensionnaly selection. Function `selwold` proposes to calculate a smoothing of R (argument $smooth$).

Usage

```
selwold(
    r, indx = seq(length(r)),
    smooth = TRUE, f = 1/3,
    alpha = .05, digits = 3,
    plot = TRUE,
    xlab = "Index", ylab = "Value", main = "r",
    ...
    )
```

Arguments

`r`	Vector of a given error rate (n) or any other indicator.
`indx`	Vector of indexes (n), typically the nb. of Lvs.
`smooth`	Logical. If `TRUE` (default), the selection is done on the smoothed R.
`f`	Window for smoothing R with function `lowess`.
`alpha`	Proportion $alpha$ used as threshold for R.
`digits`	Number of digits for R.
`plot`	Logical. If `TRUE` (default), results are plotted.
`xlab`	x-axis label of the plot of r (left-side in the graphic window).
`ylab`	y-axis label of the plot of r (left-side in the graphic window).
`main`	Title of the plot of r (left-side in the graphic window).
`...`	Other arguments to pass in function `lowess`.

Value

`res`	matrix with for each number of Lvs: r, the observed error rate quantifying the model performance; $diff$, the difference between $r(a+1)$ and $r(a)$; R, the relative gain in efficiency after a new LV is added to the model; Rs, smoothing of R.
`opt`	The index of the minimum for r.
`sel`	The index of the selection from the R (or smoothed R) threshold.

References

Andries, J.P.M., Vander Heyden, Y., Buydens, L.M.C., 2011. Improved variable reduction in partial least squares modelling based on Predictive-Property-Ranked Variables and adaptation of partial least squares complexity. Analytica Chimica Acta 705, 292-305. https://doi.org/10.1016/j.aca.2011.06.037

Bro, R., Kjeldahl, K., Smilde, A.K., Kiers, H.A.L., 2008. Cross-validation of component models: A critical look at current methods. Anal Bioanal Chem 390, 1241-1251. https://doi.org/10.1007/s00216-007-1790-1

Li, B., Morris, J., Martin, E.B., 2002. Model selection for partial least squares regression. Chemometrics and Intelligent Laboratory Systems 64, 79-89. https://doi.org/10.1016/S0169-7439(02)00051-5

Westad, F., Martens, H., 2000. Variable Selection in near Infrared Spectroscopy Based on Significance Testing in Partial Least Squares Regression. J. Near Infrared Spectrosc., JNIRS 8, 117-124.

Wold S. Cross-Validatory Estimation of the Number of Components in Factor and Principal Components Models. Technometrics. 1978;20(4):397-405

Examples

```
data(cassav)

Xtrain <- cassav$Xtrain
ytrain <- cassav$ytrain
X <- cassav$Xtest
y <- cassav$ytest

nlv <- 20
res <- gridscorelv(
    Xtrain, ytrain, X, y,
    score = msep, fun = plskern,
    nlv = 0:nlv
    )
selwold(res$y1, res$nlv, f = 2/3)
```

snv *Standard normal variate transformation (SNV)*

Description

SNV transformation of the row observations (e.g. spectra) of a dataset. By default, each observation is centered on its mean and divided by its standard deviation.

Usage

```
snv(X, center = TRUE, scale = TRUE)
```

Arguments

X	X-data (n, p).
center	Logical. If TRUE (default), the centering in the SNV is done.
scale	Logical. If TRUE (default), the scaling in the SNV is done.

Value

A matrix of the transformed data.

Examples

```
data(cassav)

X <- cassav$Xtest

Xp <- snv(X)

oldpar <- par(mfrow = c(1, 1))
par(mfrow = c(1, 2))
plotsp(X, main = "Signal")
plotsp(Xp, main = "Corrected signal")
abline(h = 0, lty = 2, col = "grey")
par(oldpar)
```

sopls *Block dimension reduction by SO-PLS*

Description

Function soplsr implements dimension reductions of pre-selected blocks of variables (= set of columns) of a reference (= training) matrix, by sequential orthogonalization-PLS (said "SO-PLS").

Function soplsrcv perfoms repeteated cross-validation of an SO-PLS model in order to choose the optimal lv combination from the different blocks.

SO-PLS is described for instance in Menichelli et al. (2014), Biancolillo et al. (2015) and Biancolillo (2016).

The block reduction consists in calculating latent variables (= scores) for each block, each block being sequentially orthogonalized to the information computed from the previous blocks.

The function allows giving a priori weights to the rows of the reference matrix in the calculations.

Auxiliary functions

transform Calculates the LVs for any new matrices list $Xlist$ from the model.

predict Calculates the predictions for any new matrices list $Xlist$ from the model.

Usage

```
soplsr(Xlist, Y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlv)

soplsrcv(Xlist, Y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlvlist = list(),
nbrep = 30, cvmethod = "kfolds", seed = 123, samplingk = NULL, nfolds = 7,
optimisation = c("global","sequential")[1],
selection = c("localmin","globalmin","1std")[1], majorityvote = FALSE)


## S3 method for class 'Soplsr'
transform(object, X, ...)

## S3 method for class 'Soplsr'
predict(object, X, ...)
```

Arguments

Xlist	A list of matrices or data frames of reference (= training) observations.
X	For the auxiliary functions: list of new X-data, with the same variables than the training X-data.
Y	A nxq matrix or data frame, or a vector of length n, of reference (= training) responses.

Xscaling	vector (of length Xlist) of variable scaling for each datablock, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	variable scaling for the Y-block, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	a priori weights to the rows of the reference matrix in the calculations.
nlv	A vector of same length as the number of blocks defining the number of scores to calculate for each block, or a single number. In this last case, the same number of scores is used for all the blocks.
nlvlist	A list of same length as the number of X-blocks. Each component of the list gives the number of PLS components of the corresponding X-block to test.
nbrep	An integer, setting the number of CV repetitions. Default value is 30.
cvmethod	"kfolds" for k-folds cross-validation, or "loo" for leave-one-out.
seed	a numeric. Seed used for the repeated resampling, and if cvmethod is "kfolds" and samplingk is not NULL.
samplingk	A vector of length n. The elements are the values of a qualitative variable used for stratified partition creation. If NULL, the first observation is set in the first fold, the second observation in the second fold, etc...
nfolds	An integer, setting the number of partitions to create. Default value is 7.
optimisation	"global" or "sequential" optimisation of the number of components. If "sequential", the optimal lv number is found for the first X-block, then for the 2nd one, etc...
selection	a character indicating the selection method to use to choose the optimal combination of components, among "localmin","globalmin","1std". If "localmin": the optimal combination corresponds to the first local minimum of the mean CV rmse. If "globalmin" : the optimal combination corresponds to the minimum mean CV rmse. If "1std" (one standard errror rule): it corresponds to the first combination after which the mean cross-validated rmse does not decrease significantly.
majorityvote	only if optimisation is "global" or one X-block. If majorityvote is TRUE, the optimal combination is chosen for each Y variable, with the chosen selection, before a majority vote. If majorityvote is "FALSE, the optimal combination is simply chosen with the chosen selection.
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For soplsr:

fm	A list of the plsr models.
T	A matrix with the concatenated scores calculated from the X-blocks.

pred	A matrice nxq with the calculated fitted values.
xmeans	list of vectors of X-mean values.
ymeans	vector of Y-mean values.
xscales	list of vectors of X-scaling values.
yscales	vector of Y-scaling values.
b	A list of X-loading weights, used in the orthogonalization step.
weights	Weights applied to the training observations.
nlv	vector of numbers of latent variables from each X-block.

For transform.Soplsr: the LVs calculated for the new matrices list $Xlist$ from the model.

For predict.Soplsr: predicted values for each observation

For soplsrcv:

lvcombi	matrix or list of matrices, of tested component combinations.
optimcombi	the number of PLS components of each X-block allowing the optimisation of the mean rmseCV.
rmseCV_byY	matrix or list of matrices of mean and sd of cross-validated RMSE in the model for each combination and each response variable.
ExplVarCV_byY	matrix or list of matrices of mean and sd of cross-validated explained variances in the model for each combination and each response variable.
rmseCV	matrix or list of matrices of mean and sd of cross-validated RMSE in the model for each combination and response variables.
ExplVarCV	matrix or list of matrices of mean and sd of cross-validated explained variances in the model for each combination and response variables.

References

- Biancolillo et al. , 2015. Combining SO-PLS and linear discriminant analysis for multi-block classification. Chemometrics and Intelligent Laboratory Systems, 141, 58-67.

- Biancolillo, A. 2016. Method development in the area of multi-block analysis focused on food analysis. PhD. University of copenhagen.

- Menichelli et al., 2014. SO-PLS as an exploratory tool for path modelling. Food Quality and Preference, 36, 122-134.

- Tenenhaus, M., 1998. La régression PLS: théorie et pratique. Editions Technip, Paris, France.

See Also

soplsr_soplsda_allsteps function to help determine the optimal number of latent variables, perform a permutation test, calculate model parameters and predict new observations.

Examples

```
N <- 10 ; p <- 12
set.seed(1)
X <- matrix(rnorm(N * p, mean = 10), ncol = p, byrow = TRUE)
Y <- matrix(rnorm(N * 2, mean = 10), ncol = 2, byrow = TRUE)
colnames(X) <- paste("varx", 1:ncol(X), sep = "")
colnames(Y) <- paste("vary", 1:ncol(Y), sep = "")
rownames(X) <- rownames(Y) <- paste("obs", 1:nrow(X), sep = "")
set.seed(NULL)
X
Y

n <- nrow(X)

X_list <- list(X[,1:4], X[,5:7], X[,9:ncol(X)])
X_list_2 <- list(X[1:2,1:4], X[1:2,5:7], X[1:2,9:ncol(X)])

soplsrcv(X_list, Y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL,
nlvlist=list(0:1, 1:2, 0:1), nbrep=1, cvmethod="loo", seed = 123, samplingk=NULL,
optimisation="global", selection="localmin", majorityvote=FALSE)


ncomp <- 2
fm <- soplsr(X_list, Y, nlv = ncomp)
transform(fm, X_list_2)
predict(fm, X_list_2)

mse(predict(fm, X_list), Y)

# VIP calculation based on the proportion of Y-variance explained by the components
vip(fm$fm[[1]], X_list[[1]], Y = NULL, nlv = ncomp)
vip(fm$fm[[2]], X_list[[2]], Y = NULL, nlv = ncomp)
vip(fm$fm[[3]], X_list[[3]], Y = NULL, nlv = ncomp)

ncomp <- c(2, 0, 3)
fm <- soplsr(X_list, Y, nlv = ncomp)
transform(fm, X_list_2)
predict(fm, X_list_2)
mse(predict(fm, X_list), Y)

ncomp <- 0
fm <- soplsr(X_list, Y, nlv = ncomp)
transform(fm, X_list_2)
predict(fm, X_list_2)

ncomp <- 2
weights <- rep(1 / n, n)
#w <- 1:n
fm <- soplsr(X_list, Y, Xscaling = c("sd","pareto","none"), nlv = ncomp, weights = weights)
transform(fm, X_list_2)
predict(fm, X_list_2)
```

soplsr_soplsda_allsteps *SOPLSR or SOPLSDA analysis steps*

Description

Help determine the optimal number of latent variables by cross-validation, perform a permutation test, calculate model parameters and predict new observations, for soplsr (soplsr), soplsrda (soplsrda), soplslda (soplslda) or soplsqda (soplsqda) models.

Usage

```
soplsr_soplsda_allsteps(Xlist, Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y, Yscaling = c("none","pareto","sd")[1], weights = NULL,
                    newXlist = NULL, newXnames = NULL,

                    method = c("soplsr", "soplsrda","soplslda","soplsqda")[1],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[1],
                    nlv = c(),
                    nlvlist = list(),
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[1],
                    nbrep = 30,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 30,

                    optimisation = c("global","sequential")[1],
                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    import = c("R","ChemFlow","W4M")[1],
                    outputfilename = NULL)
```

Arguments

Xlist	list of training X-data (n, p).
Xnames	name of the X-matrices
Xscaling	vector of Xlist length. X variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.

Y	Training Y-data (n, q) for plsr models, and $(n, 1)$ for plsrda, plslda or plsqda models.
Yscaling	Y variable scaling among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	Weights $(n, 1)$ to apply to the training observations. Internally, weights are "normalized" to sum to 1. Default to NULL (weights are set to $1/n$).
newXlist	list of new X-data (m, p) to consider.
newXnames	names of the newX-matrices
method	method to apply among "soplsr", "soplsrda","soplslda","soplsqda"
prior	for soplslda or soplsqda models : The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
step	step of the analysis among "nlvtest" (cross-validation to help determine the optimal number of latent variables), "permutation" (permutation test),"model" (model calculation),"prediction" (prediction of newX-data or X-data if any))
nlv	if step is not "nlvtest". A vector of same length as the number of blocks defining the number of scores to calculate for each block, or a single number. In this last case, the same number of scores is used for all the blocks.
nlvlist	if step is not "nlvtest". A list of same length as the number of X-blocks. Each component of the list gives the number of PLS components of the corresponding X-block to test.
modeloutput	if step is "model": outputs among "scores", "loadings", "coef" (regression coefficients), "vip" (Variable Importance in Projection; the VIP calculation being based on the proportion of Y-variance explained by the components, as proposed by Mehmood et al (2012, 2020).)
cvmethod	if step is "nlvtest" or "permutation": "kfolds" for k-folds cross-validation, or "loo" for leave-one-out.
nbrep	if step is "nlvtest" and cvmethod is "kfolds": An integer, setting the number of CV repetitions. Default value is 30. Must me set to 1 if cvmethod is "loo"
seed	if step is "nlvtest" and cvmethod is "kfolds", or if step is "permutation: a numeric. Seed used for the repeated resampling
samplingk	A vector of length n. The elements are the values of a qualitative variable used for stratified partition creation. If NULL, the first observation is set in the first fold, the second observation in the second fold, etc...
nfolds	if cvmethod is "kfolds". An integer, setting the number of partitions to create. Default value is 10.
npermut	if step is "permutation": An integer, setting the number of Y-Block with permutated responses to create. Default value is 30.
optimisation	if step is "nlvtest", method for the error optimisation among "global" (the optimal number of latent variables is determined after all the ordred block combination have been computed), or "sequential" (the optimal number of latent variables is determined after each addition of X-block)

criterion	if step is "nlvtest" or "permutation" and method is "soplsrda", "soplslda" or "soplsqda": optimisation criterion among "rmse" and "err" (for classification error rate)))
selection	if step is "nlvtest": a character indicating the selection method to use to choose the optimal combination of components, among "localmin","globalmin","1std". If "localmin": the optimal combination corresponds to the first local minimum of the mean CV rmse or error rate. If "globalmin" : the optimal combination corresponds to the minimum mean CV rmse or error rate. If "1std" (one standard error rule) : it corresponds to the first combination after which the mean cross-validated rmse or error rate does not decrease significantly.
import	If "R", X and Y are in the global environment, and the observation names are in rownames. If "ChemFlow", X and Y are tabulated tables (.txt), and the observation names are in the first column. If "W4M", X and Y are tabulated tables (.txt), and the observation names are in the headers of X, and in the first column of Y.
outputfilename	character: If not NULL, name of the tabular file, in which the function outputs have to be written.)

Value

If step is "nlvtest": table with rmsecv or cross-validated classification error rates. The suggested optimal number of latent variable combination is indicated by the binary "optimum" variable.

If step is "permutation": table with the dissimilarity between the original and the permutated Y-block, and the rmsecv or cross-validated classification error rates obtained with the permutated Y-block by the model and the given number of latent variables.

If step is "model": list of tables of scores, loadings, regression coefficients, and vip values by X-Block, depending of the "modeloutput" parameter.

If step is "prediction": table of predicted scores and predicted classes or values.

Examples

```
n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
colnames(Xtrain) <- paste0("V",1:p)

ytrain <- sample(c(1, 4, 10), size = n, replace = TRUE)

Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]

Xtrainlist <- list(Xtrain[,1:3], Xtrain[,4:8])

Xtestlist <- list(Xtest[,1:3], Xtest[,4:8])

nlv <- 5

resnlvtestsoplsrda <- soplsr_soplsda_allsteps(Xlist = Xtrainlist,
                  Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                  Y = ytrain, Yscaling = "none", weights = NULL,
                  newXlist = Xtestlist, newXnames = NULL,
```

```
                    method = c("soplsr", "soplsrda","soplslda","soplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[1],
                    nlvlist = list(1:2, 1:2),
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    optimisation = "global",
                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)

respermutationsoplsrda <- soplsr_soplsda_allsteps(Xlist = Xtrainlist,
                    Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newXlist = Xtestlist, newXnames = NULL,

                    method = c("soplsr", "soplsrda","soplslda","soplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[2],
                    nlv = c(2,1),
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)

plotxy(respermutationsoplsrda, pch=16)
abline (h = respermutationsoplsrda[respermutationsoplsrda[,"permut_dyssimilarity"]==0,"res_permut"])

resmodelsoplsrda <- soplsr_soplsda_allsteps(Xlist = Xtrainlist,
                    Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newXlist = Xtestlist, newXnames = NULL,
```

```
                    method = c("soplsr", "soplsrda","soplslda","soplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[3],
                    nlv = c(2,1),
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)


resmodelsoplsrda$scores
resmodelsoplsrda$loadings
resmodelsoplsrda$coef
resmodelsoplsrda$vip

respredictionsoplsrda <- soplsr_soplsda_allsteps(Xlist = Xtrainlist,
                    Xnames = NULL, Xscaling = c("none","pareto","sd")[1],
                    Y = ytrain, Yscaling = "none", weights = NULL,
                    newXlist = Xtestlist, newXnames = NULL,

                    method = c("soplsr", "soplsrda","soplslda","soplsqda")[2],
                    prior = c("unif", "prop")[1],

                    step = c("nlvtest","permutation","model","prediction")[4],
                    nlv = c(2,1),
                    modeloutput = c("scores","loadings","coef","vip"),

                    cvmethod = c("kfolds","loo")[2],
                    nbrep = 1,
                    seed = 123,
                    samplingk = NULL,
                    nfolds = 10,
                    npermut = 5,

                    criterion = c("err","rmse")[1],
                    selection = c("localmin","globalmin","1std")[1],

                    outputfilename = NULL)
```

---

soplsrda *Block dimension reduction by SO-PLS-DA*

Description

Function soplsrda implements dimension reductions of pre-selected blocks of variables (= set of columns) of a reference (= training) matrix, by sequential orthogonalization-PLS (said "SO-PLS") in a context of discrimination.

Function soplsrdacv perfoms repeteated cross-validation of an SO-PLS-RDA model in order to choose the optimal lv combination from the different blocks.

The block reduction consists in calculating latent variables (= scores) for each block, each block being sequentially orthogonalized to the information computed from the previous blocks.

The function allows giving a priori weights to the rows of the reference matrix in the calculations.

Insoplslda and soplsqda, probabilistic LDA and QDA are run over the PLS2 LVs, respectively.

Usage

```
soplsrda(Xlist, y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlv)

soplslda(Xlist, y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlv,
prior = c("unif", "prop"))

soplsqda(Xlist, y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlv,
prior = c("unif", "prop"))

soplsrdacv(Xlist, y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlvlist=list(),
nbrep=30, cvmethod="kfolds", seed = 123, samplingk = NULL, nfolds = 7,
optimisation = c("global","sequential")[1],
criterion = c("err","rmse")[1], selection = c("localmin","globalmin","1std")[1])

soplsldacv(Xlist, y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlvlist=list(),
prior = c("unif", "prop"), nbrep = 30, cvmethod = "kfolds", seed = 123, samplingk = NULL,
nfolds = 7, optimisation = c("global","sequential")[1],
criterion = c("err","rmse")[1], selection = c("localmin","globalmin","1std")[1])

soplsqdacv(Xlist, y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL, nlvlist = list(),
prior = c("unif", "prop"), nbrep = 30, cvmethod = "kfolds", seed = 123, samplingk = NULL,
nfolds = 7, optimisation = c("global","sequential")[1],
criterion = c("err","rmse")[1], selection = c("localmin","globalmin","1std")[1])

## S3 method for class 'Soplsrda'
transform(object, X, ...)

## S3 method for class 'Soplsprobda'
transform(object, X, ...)
```

```
## S3 method for class 'Soplsrda'
predict(object, X, ...)

## S3 method for class 'Soplsprobda'
predict(object, X, ...)
```

Arguments

Xlist	For the main functions: A list of matrices or data frames of reference (= training) observations.
X	For the auxiliary functions: list of new X-data, with the same variables than the training X-data.
y	Training class membership (n). **Note:** If y is a factor, it is replaced by a character vector.
Xscaling	vector (of length Xlist) of variable scaling for each datablock, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
Yscaling	variable scaling for the Y-block, among "none" (mean-centering only), "pareto" (mean-centering and pareto scaling), "sd" (mean-centering and unit variance scaling). If "pareto" or "sd", uncorrected standard deviation is used.
weights	a priori weights to the rows of the reference matrix in the calculations.
nlv	A vector of same length as the number of blocks defining the number of scores to calculate for each block, or a single number. In this last case, the same number of scores is used for all the blocks.
nlvlist	A list of same length as the number of X-blocks. Each component of the list gives the number of PLS components of the corresponding X-block to test.
nbrep	An integer, setting the number of CV repetitions. Default value is 30.
cvmethod	"kfolds" for k-folds cross-validation, or "loo" for leave-one-out.
seed	a numeric. Seed used for the repeated resampling, and if cvmethod is "kfolds" and samplingk is not NULL.
samplingk	Optional. A vector of length n. The elements are the values of a qualitative variable used for stratified partition creation.
nfolds	An integer, setting the number of partitions to create. Default value is 7.
optimisation	"global" or "sequential" optimisation of the number of components. If "sequential", the optimal lv number is found for the first X-block, then for the 2nd one, etc...
criterion	optimisation criterion among "rmse" and "err" (for classification error rate)
selection	a character indicating the selection method to use to choose the optimal combination of components, among "localmin","globalmin","1std". If "localmin": the optimal combination corresponds to the first local minimum of the mean CV rmse or error rate. If "globalmin" : the optimal combination corresponds to the

	minimum mean CV rmse or error rate. If "1std" (one standard error rule) : it corresponds to the first combination after which the mean cross-validated rmse or error rate does not decrease significantly.
prior	The prior probabilities of the classes. Possible values are "unif" (default; probabilities are set equal for all the classes) or "prop" (probabilities are set equal to the observed proportions of the classes in y).
object	For the auxiliary functions: A fitted model, output of a call to the main functions.
...	For the auxiliary functions: Optional arguments. Not used.

Value

For soplsrda, soplslda, soplsqda:

fm	list with the PLS models: (T): X-scores matrix; (P): X-loading matrix;(R): The PLS projection matrix (p,nlv); (W): X-loading weights matrix ;(C): The Y-loading weights matrix; (TT): the X-score normalization factor; (xmeans): the centering vector of X (p,1); (ymeans): the centering vector of Y (q,1); (weights): vector of observation weights; (Xscales): X scaling values; (Yscales): Y scaling values; (U): intermediate output.
lev	classes
ni	number of observations in each class

For transform.Soplsrda, transform.Soplsprobda: the LVs Calculated for the new matrices list $Xlist$ from the model.

For predict.Soplsrda, predict.Soplsprobda:

pred	predicted class for each observation
posterior	calculated probability of belonging to a class for each observation

For soplsrdacv, soplsldacv, soplsqdacv:

lvcombi	matrix or list of matrices, of tested component combinations.
optimCombiLine	number of the combination line corresponding to the optimal one. In the case of a sequential optimisation, it is the number of the combination line in the model with all the X-blocks.
optimcombi	the number of PLS components of each X-block allowing the optimisation of the mean rmseCV.
optimExplVarCV	cross-validated explained variance for the optimal soplsda model.
rmseCV	matrix or list of matrices of mean and sd of cross-validated rmse in the model for each combination and response variables.
ExplVarCV	matrix or list of matrices of mean and sd of cross-validated explained variances in the model for each combination and response variables.
errCV	matrix or list of matrices of mean and sd of cross-validated classification error rates in the model for each combination and response variables.

References

- Biancolillo et al. , 2015. Combining SO-PLS and linear discriminant analysis for multi-block classification. Chemometrics and Intelligent Laboratory Systems, 141, 58-67.

- Biancolillo, A. 2016. Method development in the area of multi-block analysis focused on food analysis. PhD. University of copenhagen.

- Menichelli et al., 2014. SO-PLS as an exploratory tool for path modelling. Food Quality and Preference, 36, 122-134.

- Tenenhaus, M., 1998. La régression PLS: théorie et pratique. Editions Technip, Paris, France.

See Also

soplsr_soplsda_allsteps function to help determine the optimal number of latent variables, perform a permutation test, calculate model parameters and predict new observations.

Examples

```
N <- 10 ; p <- 12
set.seed(1)
X <- matrix(rnorm(N * p, mean = 10), ncol = p, byrow = TRUE)
y <- matrix(sample(c("1", "4", "10"), size = N, replace = TRUE), ncol=1)
colnames(X) <- paste("x", 1:ncol(X), sep = "")
set.seed(NULL)

n <- nrow(X)

X_list <- list(X[,1:4], X[,5:7], X[,9:ncol(X)])
X_list_2 <- list(X[1:2,1:4], X[1:2,5:7], X[1:2,9:ncol(X)])

# EXEMPLE WITH SO-PLS-RDA
soplsrdacv(X_list, y, Xscaling = c("none", "pareto", "sd")[1],
Yscaling = c("none", "pareto", "sd")[1], weights = NULL,
nlvlist=list(0:1, 1:2, 0:1), nbrep=1, cvmethod="loo", seed = 123,
samplingk = NULL, nfolds = 3, optimisation = "global",
criterion = c("err","rmse")[1], selection = "localmin")

ncomp <- 2
fm <- soplsrda(X_list, y, nlv = ncomp)
predict(fm,X_list_2)
transform(fm,X_list_2)

ncomp <- c(2, 0, 3)
fm <- soplsrda(X_list, y, nlv = ncomp)
predict(fm,X_list_2)
transform(fm,X_list_2)

ncomp <- 0
fm <- soplsrda(X_list, y, nlv = ncomp)
predict(fm,X_list_2)
transform(fm,X_list_2)
```

```
# EXEMPLE WITH SO-PLS-LDA
ncomp <- 2
weights <- rep(1 / n, n)
#w <- 1:n
soplslda(X_list, y, Xscaling = "none", nlv = ncomp, weights = weights)
soplslda(X_list, y, Xscaling = "pareto", nlv = ncomp, weights = weights)
soplslda(X_list, y, Xscaling = "sd", nlv = ncomp, weights = weights)

fm <- soplslda(X_list, y, Xscaling = c("none","pareto","sd"), nlv = ncomp, weights = weights)
predict(fm,X_list_2)
transform(fm,X_list_2)
```

`sourcedir` *Source R functions in a directory*

Description

Source all the R functions contained in a directory.

Usage

```
sourcedir(path, trace = TRUE, ...)
```

Arguments

`path`	A character vector of full path names; the default corresponds to the working directory, `getwd()`.
`trace`	Logical. Default to `TRUE`. See the code.
`...`	Additional arguments to pass in the function `list.files`.

Value

Sourcing.

Examples

```
path <- "D:/Users/Fun"
sourcedir(path, FALSE)
```

summ *Description of the quantitative variables of a data set*

Description

Displays summary statistics for each quantitative column of the data set.

Usage

```
summ(X, nam = NULL, digits = 3)
```

Arguments

X	A matrix or data frame containing the variables to summarize.
nam	Names of the variables to summarize (vector of character strings). Default to NULL (all the columns are considered).
digits	Number of digits for the numerical outputs.

Value

tab	A dataframe of summary statistics : $NbVal$, $Mean$, $Min.$, $Max.$, $Stdev$, $Median$, $X1st.Qu.$, $X3rd.Qu.$, $NbNA$
ntot	number of observations

Examples

```
dat <- data.frame(
  v1 = rnorm(10),
  v2 = c(NA, rnorm(8), NA),
  v3 = c(NA, NA, NA, rnorm(7))
  )
dat

summ(dat)
summ(dat, nam = c("v1", "v3"))
```

svmr *SVM Regression and Discrimination*

Description

SVM models with Gaussian (RBF) kernel.

svmr: SVM regression (SVMR).

svmda: SVM discrimination (SVMC).

The SVM models are fitted with parameterization $'C'$, not the $'nu'$ parameterization.

The RBF kernel is defined by: exp(-gamma * |x - y|^2).

For tuning the model, usual preliminary ranges are for instance:

- cost = 10^(-5:15)

- epsilon = seq(.1, .3, by = .1)

- gamma = 10^(-6:3)

The functions uses function svm of package e1071 (Meyer et al. 2021) available on CRAN (e1071 uses the tool box LIVSIM; Chang & Lin, http://www.csie.ntu.edu.tw/~cjlin/libsvm).

Usage

```
svmr(X, y, cost = 1, epsilon = .1, gamma = 1, scale = FALSE)

svmda(X, y, cost = 1, epsilon = .1, gamma = 1, scale = FALSE)

## S3 method for class 'Svm'
predict(object, X, ...)

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

Arguments

X	For the main functions: Training X-data (n, p). — For the auxiliary functions: New X-data (m, p) to consider.
y	Training Y-data (n).
cost	The cost of constraints violation $cost$ parameter. See svm.
epsilon	The $epsilon$ parameter in the insensitive-loss function. See svm.
gamma	The $gamma$ parameter in the RBF kernel.
scale	Logical. If TRUE, X and Y are scaled internally.
object	For the auxiliary functions: A fitted model, output of a call to the main function.
...	For the auxiliary functions: Optional arguments.

Value

For svmr and svmda:

fm — list of outputs such as: call; type; kernel; cost; degree; gamma; coef0; nu; epsilon; sparse; scaled; x.scale; y.scale; nclasses; levels; tot.nSV; nSV; labels; SV: The resulting support vectors (possibly scaled); index: The index of the resulting support vectors in the data matrix. Note that this index refers to the preprocessed data (after the possible effect of na.omit and subset); rho: The negative intercept; compprob; probA, probB: numeric vectors of length k(k-1)/2, k number of classes, containing the parameters of the logistic distributions fitted to the decision values of the binary classifiers (1 / (1 + exp(a x + b))); sigma: In case of a probabilistic regression model, the scale parameter of the hypothesized (zero-mean) laplace distribution estimated by maximum likelihood; coefs: The corresponding coefficients times the training labels; na.action; fitted; decision.values; residuals; isnum.

For predict.Svm:

pred — predictions for each observation.

For summary.Svm:display of call, parameters, and number of support vectors.

Note

The first example illustrates SVMR. The second one is the example of fitting the function sinc(x) described in Rosipal & Trejo 2001 p. 105-106. The third one illustrates SVMC.

References

Meyer, M. 2021 Support Vector Machines - The Interface to libsvm in package e1071. FH Technikum Wien, Austria, David.Meyer@R-Project.org. https://cran.r-project.org/web/packages/e1071/vignettes/svmdoc.pdf

Chang, cost.-cost. & Lin, cost.-J. (2001). LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm. Detailed documentation (algorithms, formulae, . . .) can be found in http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.ps.gz

Examples

```
## EXAMPLE 1 (SVMR)

n <- 50 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
ytest <- ytrain[1:m]

fm <- svmr(Xtrain, ytrain)
predict(fm, Xtest)

pred <- predict(fm, Xtest)$pred
```

```
msep(pred, ytest)

summary(fm)

## EXAMPLE 2

x <- seq(-10, 10, by = .2)
x[x == 0] <- 1e-5
n <- length(x)
zy <- sin(abs(x)) / abs(x)
y <- zy + rnorm(n, 0, .2)
plot(x, y, type = "p")
lines(x, zy, lty = 2)
X <- matrix(x, ncol = 1)

fm <- svmr(X, y, gamma = .5)
pred <- predict(fm, X)$pred
plot(X, y, type = "p")
lines(X, zy, lty = 2)
lines(X, pred, col = "red")

## EXAMPLE 3 (SVMC)

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c("a", "10", "d"), size = n, replace = TRUE)
m <- 5
Xtest <- Xtrain[1:m, ] ; ytest <- ytrain[1:m]

cost <- 100 ; epsilon <- .1 ; gamma <- 1
fm <- svmda(Xtrain, ytrain,
    cost = cost, epsilon = epsilon, gamma = gamma)
predict(fm, Xtest)

pred <- predict(fm, Xtest)$pred
err(pred, ytest)

summary(fm)
```

`transform` *Generic transform function*

Description

Transformation of the X-data by a fitted model.

Usage

```
transform(object, X, ...)
```

Arguments

object	A fitted model, output of a call to a fitting function.
X	New X-data to consider.
...	Optional arguments.

Value

the transformed X-data

Examples

```
## EXAMPLE 1

n <- 6 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)

fm <- pcaeigen(X, nlv = 3)

fm$T
transform(fm, X[1:2, ], nlv = 2)

## EXAMPLE 2

n <- 6 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)

fm <- plskern(X, y, nlv = 3)
fm$T
transform(fm, X[1:2, ], nlv = 2)
```

---

vip *Variable Importance in Projection (VIP)*

Description

vip calculates the Variable Importance in Projection (VIP) for a PLS model.

Usage

```
vip(object, X, Y = NULL, nlv = NULL)
```

Arguments

object	A fitted model, output of a call to a fitting function among plskern, plsnipals, plsrannar, plsrda, plslda), plsqda).
X	X-data involved in the fitted model
Y	Y-data involved in the fitted model. If Y is NULL (default value), the VIP calculation is based on the proportion of Y-variance explained by the components, as proposed by Mehmood et al (2012, 2020). If Y is not NULL, the VIP calculation is based on the redundancy, as proposed by Tenenhaus (1998).
nlv	Number of components (LVs) to consider.

Value

matrix ((q, nlv)) with VIP values, for models with 1 to nlv latent variables.

References

Mehmood, T.,Liland, K.H.,Snipen, L.,Sæbø, S., 2012. A review of variable selection methods in Partial Least Squares Regression. Chemometrics and Intelligent Laboratory Systems, 118, 62-69.

Mehmood, T., Sæbø, S.,Liland, K.H., 2020. Comparison of variable selection methods in partial least squares regression. Journal of Chemometrics, 34, e3226.

Tenenhaus, M., 1998. La régression PLS: théorie et pratique. Editions Technip, Paris, France.

Examples

```
## EXAMPLE OF PLS

n <- 50 ; p <- 4
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- rnorm(n)
Ytrain <- cbind(y1 = ytrain, y2 = 100 * ytrain)
m <- 3
Xtest <- Xtrain[1:m, , drop = FALSE]
Ytest <- Ytrain[1:m, , drop = FALSE] ; ytest <- Ytest[1:m, 1]

nlv <- 3
fm <- plskern(Xtrain, Ytrain, nlv = nlv)
vip(fm, Xtrain, Ytrain, nlv = nlv)
vip(fm, Xtrain, nlv = nlv)

fm <- plskern(Xtrain, ytrain, nlv = nlv)
vip(fm, Xtrain, ytrain, nlv = nlv)
vip(fm, Xtrain, nlv = nlv)

## EXAMPLE OF PLSDA

n <- 50 ; p <- 8
Xtrain <- matrix(rnorm(n * p), ncol = p)
ytrain <- sample(c("1", "4", "10"), size = n, replace = TRUE)
```

```
Xtest <- Xtrain[1:5, ] ; ytest <- ytrain[1:5]

nlv <- 5
fm <- plsrda(Xtrain, ytrain, nlv = nlv)
vip(fm, Xtrain, ytrain, nlv = nlv)
```

---

wdist *Distance-based weights*

Description

Calculation of weights from a vector of distances using a decreasing inverse exponential function.

Let d be a vector of distances.

1- Preliminary weights are calculated by $w = exp(-d/(h * mad(d)))$, where h is a scalar > 0 (scale factor).

2- The weights corresponding to distances higher than $median(d) + cri * mad(d)$, where cri is a scalar > 0, are set to zero. This step is used for removing outliers.

3- Finally, the weights are "normalized" between 0 and 1 by $w = w/max(w)$.

Usage

```
wdist(d, h, cri = 4, squared = FALSE)
```

Arguments

d	A vector of distances.
h	A scaling factor (positive scalar). Lower is h, sharper is the decreasing function. See the examples.
cri	A positive scalar used for defining outliers in the distances vector.
squared	Logical. If TRUE, distances d are replaced by the squared distances in the decreasing function, which corresponds to a Gaussian (RBF) kernel function. Default to $FALSE$).

Value

A vector of weights.

Examples

```
x1 <- sqrt(rchisq(n = 100, df = 10))
x2 <- sqrt(rchisq(n = 10, df = 40))
d <- c(x1, x2)
h <- 2 ; cri <- 3
w <- wdist(d, h = h, cri = cri)
```

```
oldpar <- par(mfrow = c(1, 1))
par(mfrow = c(2, 2))
plot(d)
hist(d, n = 50)
plot(w, ylim = c(0, 1)) ; abline(h = 1, lty = 2)
plot(d, w, ylim = c(0, 1)) ; abline(h = 1, lty = 2)
par(oldpar)

d <- seq(0, 15, by = .5)
h <- c(.5, 1, 1.5, 2.5, 5, 10, Inf)
for(i in 1:length(h)) {
  w <- wdist(d, h = h[i])
  z <- data.frame(d = d, w = w, h = rep(h[i], length(d)))
  if(i == 1) res <- z else res <- rbind(res, z)
  }
res$h <- as.factor(res$h)
headm(res)
plotxy(res[, c("d", "w")], asp = 0, group = res$h, pch = 16)
```

`xfit` *Matrix fitting from a PCA or PLS model*

Description

Function `xfit` calculates an approximate of matrix X (X_fit) from a PCA or PLS fitted on X.

Function `xresid` calculates the residual matrix $E = X - X_fit$.

Usage

```
xfit(object, X, ...)

## S3 method for class 'Pca'
xfit(object, X, ..., nlv = NULL)

## S3 method for class 'Plsr'
xfit(object, X, ..., nlv = NULL)

xresid(object, X, ..., nlv = NULL)
```

Arguments

`object`	A fitted model, output of a call to a fitting function.
`X`	The X-data that was used to fit the model `object`.
`nlv`	Number of components (PCs or LVs) to consider.
`...`	Optional arguments.

Value

For xfit:matrix of fitted values.

For xresid:matrix of residuals.

Examples

```
n <- 6 ; p <- 4
X <- matrix(rnorm(n * p), ncol = p)
y <- rnorm(n)

nlv <- 3
fm <- pcasvd(X, nlv = nlv)
xfit(fm, X)
xfit(fm, X, nlv = 1)
xfit(fm, X, nlv = 0)

X - xfit(fm, X)
xresid(fm, X)

X - xfit(fm, X, nlv = 1)
xresid(fm, X, nlv = 1)
```

Zhang2023 *Zhang2023*

Description

A selection of reduced non-targeted metabolomics datasets from the article of Zhang et al.(2023)

Usage

```
data(Zhang2023)
```

Format

A list with 6 components: GCTOF, HILICNEG, HILICPOS, metadata.

CHSNEG A matrix whose rows are 68 mice and columns are 163 metabolomic variables obtained by LC-MS.

CHSPOS A matrix whose rows are 68 mice and columns are 288 metabolomic variables obtained by LC-MS.

GCTOF A matrix whose rows are 68 mice and columns are 108 metabolomic variables obtained by GC-MS.

HILICNEG A matrix whose rows are 68 mice and columns are 44 metabolomic variables obtained by LC-MS.

HILICPOS A matrix whose rows are 68 mice and columns are 133 metabolomic variables obtained by LC-MS.

metadata A matrix whose rows are 68 mice and columns are the genotype group (Mutant/Wild) and the gender (Male/Female).

Details

The 6 samples from 5 mutant groups (Dhfr, Gnpda1, Plk1, Sra1, Ulk3) and the 40 controls were retained, with the exception of 2 animals wM_035 (wild Male) et mM_102 (mutant Male) that had missing values in non-targeted metabolomics. Three HILIC-NEG (Ser_Asn, Théophylline, Val_Asp) and 2 HILIC-POS variables (Ser-His, Thr-Arg) were then removed due to missing or infinite values.

Source

Zhang, Y.; Barupal, D.K.; Fan, S.; Gao, B.; Zhu, C.; Flenniken, A.M.; McKerlie, C.; Nutter, L.M.J.; Lloyd, K.C.K.; Fiehn, O. Sexual Dimorphism of the Mouse Plasma Metabolome Is Associated with Phenotypes of 30 Gene Knockout Lines. Metabolites 2023, 13, 947. https://doi.org/10.3390/metabo13080947

Examples

```
data(Zhang2023)

X <- Zhang2023$GCTOF
head(X)

Y <- Zhang2023$metadata
head(Y)
```

Index