Package ‘mdatools’

May 24, 2019

Title Multivariate Data Analysis for Chemometrics
Version 0.9.4
Date 2019-05-24
Author Sergey Kucheryavskiy
Maintainer Sergey Kucheryavskiy <svkucheryavski@gmail.com>
Description Package implements projection based methods for preprocessing, 
exploring and analysis of multivariate data used in chemometrics.
Encoding UTF-8
License MIT + file LICENSE
Imports methods, graphics, grDevices, stats
RoxygenNote 6.1.1
Suggests testthat
NeedsCompilation no
Depends R (>= 2.10)
Repository CRAN
Date/Publication 2019-05-24 13:20:03 UTC

R topics documented:

as.matrix.classres ........................................... 7
as.matrix.ldecomp .............................................. 8
as.matrix.plsdares ............................................ 8
as.matrix.plsres ............................................... 9
as.matrix.regcoeffs .......................................... 9
as.matrix.regres ............................................. 10
bars ............................................................ 10
classify.plsda ................................................. 11
classres ........................................................ 11
crossval ....................................................... 12
crossval.str .................................................. 13
erfinv .......................................................... 13
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>errorbars</td>
</tr>
<tr>
<td>getB</td>
</tr>
<tr>
<td>getCalibrationData</td>
</tr>
<tr>
<td>getCalibrationData.pca</td>
</tr>
<tr>
<td>getCalibrationData.simcam</td>
</tr>
<tr>
<td>getClassificationPerformance</td>
</tr>
<tr>
<td>getConfusionMatrix</td>
</tr>
<tr>
<td>getConfusionMatrix.classres</td>
</tr>
<tr>
<td>getMainTitle</td>
</tr>
<tr>
<td>getProbabilities</td>
</tr>
<tr>
<td>getProbabilities.simca</td>
</tr>
<tr>
<td>getRegcoeffs</td>
</tr>
<tr>
<td>getRegcoeffs.pls</td>
</tr>
<tr>
<td>getSelectedComponents</td>
</tr>
<tr>
<td>getSelectedComponents.classres</td>
</tr>
<tr>
<td>getSelectivityRatio</td>
</tr>
<tr>
<td>getSelectivityRatio.pls</td>
</tr>
<tr>
<td>getVIPScores</td>
</tr>
<tr>
<td>getVIPScores.pls</td>
</tr>
<tr>
<td>imshow</td>
</tr>
<tr>
<td>ipls</td>
</tr>
<tr>
<td>ipls.forward</td>
</tr>
<tr>
<td>ldecomp</td>
</tr>
<tr>
<td>ldecomp.getDistances</td>
</tr>
<tr>
<td>ldecomp.getVariances</td>
</tr>
<tr>
<td>ldecomp.plotLimits</td>
</tr>
<tr>
<td>mda.cbind</td>
</tr>
<tr>
<td>mda.data2im</td>
</tr>
<tr>
<td>mda.df2mat</td>
</tr>
<tr>
<td>mda.exclcols</td>
</tr>
<tr>
<td>mda.exclrows</td>
</tr>
<tr>
<td>mda.getattr</td>
</tr>
<tr>
<td>mda.getexclind</td>
</tr>
<tr>
<td>mda.im2data</td>
</tr>
<tr>
<td>mda.inclcols</td>
</tr>
<tr>
<td>mda.inclrows</td>
</tr>
<tr>
<td>mda.rbind</td>
</tr>
<tr>
<td>mda.setattr</td>
</tr>
<tr>
<td>mda.setimbg</td>
</tr>
<tr>
<td>mda.show</td>
</tr>
<tr>
<td>mda.subset</td>
</tr>
<tr>
<td>mda.t</td>
</tr>
<tr>
<td>mdaplot</td>
</tr>
<tr>
<td>mdaplot.areColors</td>
</tr>
<tr>
<td>mdaplot.formatValues</td>
</tr>
<tr>
<td>mdaplot.getAxesLim</td>
</tr>
<tr>
<td>mdaplot.getColors</td>
</tr>
</tbody>
</table>
R topics documented:

mdaplot.plotAxes ................................................. 44
mdaplot.showColorbar ........................................... 44
mdaplot.showGrid ................................................ 45
mdaplot.showLabels .............................................. 45
mdaplot.showLegend ............................................. 46
mdaplot.showLines .............................................. 46
mdaplot.showRegressionLine ................................... 47
mdaplotg .......................................................... 47
mdatools .......................................................... 49
pca ............................................................... 50
pca.cal ........................................................... 54
pca.crossval ...................................................... 55
pca.mvreplace .................................................... 55
pca.nipals ........................................................ 56
pca.run ........................................................... 57
pca.svd ........................................................... 58
pcares ............................................................. 58
pellets ............................................................ 60
people ............................................................. 61
pinv ............................................................... 62
plot.classres ...................................................... 62
plot.ipls .......................................................... 63
plot.pca ........................................................... 63
plot.pcares ....................................................... 64
plot.pls ........................................................... 64
plot.plsda ........................................................ 65
plot.plsdares ..................................................... 65
plot.plsres ........................................................ 66
plot.randtest ...................................................... 67
plot.regcoeffs ..................................................... 67
plot.regres ......................................................... 68
plot.simca ......................................................... 69
plot.simcam ....................................................... 69
plot.simcamres ................................................... 70
plotBiplot ........................................................ 70
plotBiplot.pca .................................................... 71
plotCooman ....................................................... 71
plotCooman.simcam ............................................. 72
plotCooman.simcamres ......................................... 72
plotCorr .......................................................... 73
plotCorr.randtest ................................................ 74
plotCumVariance .................................................. 74
plotCumVariance.ldecomp ...................................... 75
plotCumVariance.pca ............................................. 75
plotDiscriminationPower ....................................... 76
plotDiscriminationPower.simcam ............................... 77
plotExtreme ....................................................... 77
plotExtreme.simca ............................................... 78
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotHist</td>
</tr>
<tr>
<td>plotHist.randtest</td>
</tr>
<tr>
<td>plotLoadings</td>
</tr>
<tr>
<td>plotLoadings.pca</td>
</tr>
<tr>
<td>plotMisclassified</td>
</tr>
<tr>
<td>plotMisclassified.classmodel</td>
</tr>
<tr>
<td>plotMisclassified.classesres</td>
</tr>
<tr>
<td>plotModelDistance</td>
</tr>
<tr>
<td>plotModelDistance.simcam</td>
</tr>
<tr>
<td>plotModellingPower</td>
</tr>
<tr>
<td>plotModellingPower.simca</td>
</tr>
<tr>
<td>plotModellingPower.simcam</td>
</tr>
<tr>
<td>plotPerformance</td>
</tr>
<tr>
<td>plotPerformance.classmodel</td>
</tr>
<tr>
<td>plotPerformance.classesres</td>
</tr>
<tr>
<td>plotPredictions</td>
</tr>
<tr>
<td>plotPredictions.classmodel</td>
</tr>
<tr>
<td>plotPredictions.classesres</td>
</tr>
<tr>
<td>plotPredictions.pls</td>
</tr>
<tr>
<td>plotPredictions.plsres</td>
</tr>
<tr>
<td>plotPredictions.regres</td>
</tr>
<tr>
<td>plotProbabilities</td>
</tr>
<tr>
<td>plotProbabilities.classesres</td>
</tr>
<tr>
<td>plotRegcoeffs</td>
</tr>
<tr>
<td>plotRegcoeffs.pls</td>
</tr>
<tr>
<td>plotResiduals</td>
</tr>
<tr>
<td>plotResiduals.ldecomp</td>
</tr>
<tr>
<td>plotResiduals.pca</td>
</tr>
<tr>
<td>plotResiduals.pcares</td>
</tr>
<tr>
<td>plotResiduals.simcam</td>
</tr>
<tr>
<td>plotResiduals.simcamres</td>
</tr>
<tr>
<td>plotResiduals.simcares</td>
</tr>
<tr>
<td>plotRMSE</td>
</tr>
<tr>
<td>plotRMSE.ipls</td>
</tr>
<tr>
<td>plotRMSE.pls</td>
</tr>
<tr>
<td>plotRMSE.regres</td>
</tr>
<tr>
<td>plotScores</td>
</tr>
<tr>
<td>plotScores.ldecomp</td>
</tr>
<tr>
<td>plotScores.pca</td>
</tr>
<tr>
<td>plotSelection</td>
</tr>
<tr>
<td>plotSelection.ipls</td>
</tr>
<tr>
<td>plotSelectivityRatio</td>
</tr>
<tr>
<td>plotSelectivityRatio.pls</td>
</tr>
<tr>
<td>plotSensitivity</td>
</tr>
<tr>
<td>plotSensitivity.classmodel</td>
</tr>
<tr>
<td>plotSensitivity.classesres</td>
</tr>
<tr>
<td>plotSpecificity</td>
</tr>
<tr>
<td>plotSpecificity.classmodel</td>
</tr>
</tbody>
</table>
R topics documented:

plotSpecificity.classres ........................................ 108
plotVariance ......................................................... 108
plotVariance.ldaCOMP ............................................. 109
plotVariance.pca .................................................... 109
plotVIPScores ....................................................... 110
plotVIPScores.pls .................................................. 111
plotXCumVariance ................................................... 112
plotXCumVariance.pls ............................................. 112
plotXCumVariance.plsRES ......................................... 113
plotXLoadings ....................................................... 113
plotXLoadings.pls .................................................. 114
plotXResiduals ...................................................... 114
plotXResiduals.pls .................................................. 115
plotXResiduals.plsRES ............................................. 115
plotXScores .......................................................... 116
plotXScores.pls ..................................................... 116
plotXScores.plsRES .................................................. 117
plotXVariance ........................................................ 117
plotXVariance.pls ................................................... 118
plotXVariance.plsRES .............................................. 118
plotXYLoadings ...................................................... 119
plotXYLoadings.pls .................................................. 119
plotXYScores ........................................................ 120
plotXYScores.pls .................................................... 120
plotXYScores.plsRES .............................................. 121
plotYCumVariance ................................................... 121
plotYCumVariance.pls ............................................. 122
plotYCumVariance.plsRES ......................................... 122
plotYResiduals ...................................................... 123
plotYResiduals.pls .................................................. 123
plotYResiduals.regres ............................................. 124
plotYVariance ........................................................ 125
plotYVariance.pls ................................................... 125
plotYVariance.plsRES .............................................. 126
pls ................................................................. 126
pls.cal .............................................................. 131
pls.calculateSelectivityRatio .................................... 132
pls.calculateVIPScores .......................................... 133
pls.croSSLVAL ...................................................... 133
pls.run .............................................................. 134
pls.simps ........................................................... 134
plSDA ............................................................... 135
plSDA.cal ............................................................ 138
plSDA.croSSLVAL ................................................... 139
plSDARes .............................................................. 139
plsres ............................................................... 142
predict.pca ......................................................... 145
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict.pls</td>
</tr>
<tr>
<td>predict.plsda</td>
</tr>
<tr>
<td>predict.simca</td>
</tr>
<tr>
<td>predict.simcam</td>
</tr>
<tr>
<td>prep.autoscale</td>
</tr>
<tr>
<td>prep.msc</td>
</tr>
<tr>
<td>prep.norm</td>
</tr>
<tr>
<td>prep.savgol</td>
</tr>
<tr>
<td>prep.snv</td>
</tr>
<tr>
<td>print.classres</td>
</tr>
<tr>
<td>print.ipls</td>
</tr>
<tr>
<td>print.ldecomp</td>
</tr>
<tr>
<td>print.pca</td>
</tr>
<tr>
<td>print.pcares</td>
</tr>
<tr>
<td>print.pls</td>
</tr>
<tr>
<td>print.plsda</td>
</tr>
<tr>
<td>print.plsdares</td>
</tr>
<tr>
<td>print.plsres</td>
</tr>
<tr>
<td>print.randtest</td>
</tr>
<tr>
<td>print.regcoeffs</td>
</tr>
<tr>
<td>print.regres</td>
</tr>
<tr>
<td>print.simca</td>
</tr>
<tr>
<td>print.simcam</td>
</tr>
<tr>
<td>print.simcamres</td>
</tr>
<tr>
<td>print.simcares</td>
</tr>
<tr>
<td>randtest</td>
</tr>
<tr>
<td>regcoeffs</td>
</tr>
<tr>
<td>regcoeffs.getStat</td>
</tr>
<tr>
<td>regres</td>
</tr>
<tr>
<td>regres.bias</td>
</tr>
<tr>
<td>regres.r2</td>
</tr>
<tr>
<td>regres.rmse</td>
</tr>
<tr>
<td>regres.slope</td>
</tr>
<tr>
<td>reslim.chisq</td>
</tr>
<tr>
<td>reslim.dd</td>
</tr>
<tr>
<td>reslim.hotelling</td>
</tr>
<tr>
<td>reslim jm</td>
</tr>
<tr>
<td>selectCompNum</td>
</tr>
<tr>
<td>selectCompNum.pca</td>
</tr>
<tr>
<td>selectCompNum.pls</td>
</tr>
<tr>
<td>setResLimits</td>
</tr>
<tr>
<td>setResLimits.pca</td>
</tr>
<tr>
<td>showPredictions</td>
</tr>
<tr>
<td>showPredictions.classres</td>
</tr>
<tr>
<td>simca</td>
</tr>
<tr>
<td>simca.classify</td>
</tr>
<tr>
<td>simca.crossval</td>
</tr>
<tr>
<td>simcam</td>
</tr>
</tbody>
</table>
as.matrix.classres

as.matrix.classres

Description

Generic as.matrix function for classification results. Returns matrix with performance values for specific class.

Usage

## S3 method for class 'classres'
as.matrix(x, ncomp = NULL, nc = 1, ...)

Arguments

- **x**: classification results (object of class plsdares, simcamres, etc.).
- **ncomp**: model complexity (number of components) to show the parameters for.
- **nc**: if there are several classes, which class to show the parameters for.
- **...**: other arguments
### as.matrix.ldecomp

**as.matrix method for ldecomp object**

**Description**

Generic `as.matrix` function for linear decomposition. Returns a matrix with information about the decomposition.

**Usage**

```r
## S3 method for class 'ldecomp'
as.matrix(x, ...)
```

**Arguments**

- `x`: object of class `ldecomp`
- `...`: other arguments

### as.matrix.plsdares

**as.matrix method for PLS-DA results**

**Description**

Returns a matrix with model performance statistics for PLS-DA results.

**Usage**

```r
## S3 method for class 'plsdares'
as.matrix(x, ncomp = NULL, nc = NULL, ...)
```

**Arguments**

- `x`: PLS-DA results (object of class `plsdares`)
- `ncomp`: number of components to calculate the statistics for
- `nc`: for which class to calculate the statistics for
- `...`: other arguments
as.matrix.plsres  

as.matrix method for PLS results

Description

Returns a matrix with model performance statistics for PLS results

Usage

```r
## S3 method for class 'plsres'
as.matrix(x, ncomp = NULL, ny = 1, ...)
```

Arguments

- **x**: PLS results (object of class `plsres`)
- **ncomp**: number of components to calculate the statistics for
- **ny**: for which response variable calculate the statistics for
- **...**: other arguments

as.matrix.regcoeffs  

as.matrix method for regression coefficients class

Description

returns matrix with regression coefficients for given response number and amount of components

Usage

```r
## S3 method for class 'regcoeffs'
as.matrix(x, ncomp = 1, ny = 1, ...)
```

Arguments

- **x**: regression coefficients object (class `regcoeffs`)
- **ncomp**: number of components to return the coefficients for
- **ny**: number of response variable to return the coefficients for
- **...**: other arguments
as.matrix.regres \( \text{as.matrix method for regression results} \)

Description

Returns a matrix with model performance statistics for regression results

Usage

\[
\text{### S3 method for class 'regres'}
\text{as.matrix(x, ncomp = NULL, ny = 1, ...)}
\]

Arguments

- \( x \) regression results (object of class regres)
- \( \text{ncomp} \) model complexity (number of components) to calculate the statistics for
- \( \text{ny} \) for which response variable calculate the statistics for
- \( \ldots \) other arguments

bars \( \text{Show bars on axes} \)

Description

Shows bars (bar plot) on predefined axes

Usage

\[
\text{bars(x, y, col = NULL, bwd = 0.8, border = NA)}
\]

Arguments

- \( x \) vector with x values (centers of bars)
- \( y \) vector with y values (height of bars)
- \( \text{col} \) colors of the bars
- \( \text{bwd} \) width of the bars (as a ratio for max width)
- \( \text{border} \) color of bar edges
classify.plsda

**PLS-DA classification**

**Description**

Converts PLS predictions of y values to predictions of classes

**Usage**

```r
classify.plsda(model, y)
```

**Arguments**

- `model`: a PLS-DA model (object of class `plsda`)
- `y`: a matrix with predicted y values

**Details**

This is a service function for PLS-DA class, do not use it manually.

**Value**

Classification results (an object of class `classres`)

---

classres

**Results of classification**

**Description**

classres is used to store results classification for one or multiple classes.

**Usage**

```r
classres(c.pred, c.ref = NULL, p.pred = NULL, ncomp.selected = NULL)
```

**Arguments**

- `c.pred`: matrix with predicted values (+1 or -1) for each class.
- `c.ref`: matrix with reference values for each class.
- `p.pred`: matrix with probability values for each class.
- `ncomp.selected`: vector with selected number of components for each class.
Details

There is no need to create a classres object manually, it is created automatically when build a classification model (e.g. using simca or plsda) or apply the model to new data. For any classification method frommdatools, a class using to represent results of classification (e.g. simcares) inherits fields and methods of classres.

Value

- `c.pred`: predicted class values (+1 or -1).
- `p.pred`: predicted class probabilities.
- `c.ref`: reference (true) class values if provided.

The following fields are available only if reference values were provided.

- `tp`: number of true positives.
- `fp`: number of false positives.
- `fn`: number of false negatives.
- `specificity`: specificity of predictions.
- `sensitivity`: sensitivity of predictions.

See Also

Methods classres class:

- `showPredictions.classres`: shows table with predicted values.
- `plotPredictions.classres`: makes plot with predicted values.
- `plotSensitivity.classres`: makes plot with sensitivity vs. components values.
- `plotSpecificity.classres`: makes plot with specificity vs. components values.
- `plotMisclassified.classres`: makes plot with misclassified ratio values.
- `plotPerformance.classres`: makes plot with misclassified ration, specificity and sensitivity values.

---

crossval

Generate sequence of indices for cross-validation

Description

Generates and returns sequence of object indices for each segment in random segmented cross-validation.

Usage

`crossval(nobj, cv = NULL)`
**Arguments**

- **nobj**
  number of objects in a dataset

- **cv**
  cross-validation settings, can be a number or a list. If cv is a number, it will be used as a number of segments for random cross-validation (if cv = 1, full cross-validation will be preformed), if it is a list, the following syntax can be used: cv = list('rand', nseg, nrep) for random repeated cross-validation with nseg segments and nrep repetitions or cv = list('ven', nseg) for systematic splits to nseg segments ('venetian blinds').

**Value**

- matrix with object indices for each segment

---

**crossval.str**

*String with description of cross-validation method*

**Description**

String with description of cross-validation method

**Usage**

crossval.str(cv)

**Arguments**

- **cv**
  a list with cross-validation settings

**Value**

- a string with the description text

---

**erfinv**

*Inverse error function*

**Description**

Inverse error function

**Usage**

erfinv(x)

**Arguments**

- **x**
  a matrix or vector with data values
errorbars

Show error bars on a plot

Description

Shows error bars (errorbar plot) on predefined axes

Usage

```r
errorbars(x, lower, upper, y = NULL, col = NULL, pch = 16)
```

Arguments

- `x`: vector with x values
- `lower`: vector with lower limits for the bars
- `upper`: vector with upper limits for the bars
- `y`: vector with y values (bid points)
- `col`: color for the error bars
- `pch`: marker symbol for the plot

getB

Low-dimensional approximation of data matrix X

Description

Low-dimensional approximation of data matrix X

Usage

```r
getB(X, k = NULL, rand = c(1, 5), dist = "unif")
```

Arguments

- `X`: data matrix
- `k`: rank of X (number of components)
- `rand`: a vector with two values - number of iterations (q) and oversmapling parameter (p)
- `dist`: distribution for generating random numbers, 'unif' or 'norm'
getCalibrationData

getCalibrationData  Calibration data

Description

Calibration data

Usage

getCalibrationData(obj, ...)

Arguments

  obj  a model object
...  other arguments

Details

Generic function getting calibration data from a linear decomposition model (e.g. PCA)

getCalibrationData.pca

Get calibration data

Description

Get data, used for calibration of the PCA model

Usage

## S3 method for class 'pca'
getCalibrationData(obj, ...)

Arguments

  obj  PCA model (object of class pca)
...  other parameters
getCalibrationData

Get calibration data

Description
Get data, used for calibration of the SIMCAM model.

Usage

## S3 method for class 'simcam'
getCalibrationData(obj, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>SIMCAM model (object of class simcam)</td>
</tr>
<tr>
<td>...</td>
<td>other arguments</td>
</tr>
</tbody>
</table>

Details
See examples in help for simcam function.

getClassificationPerformance

Calculation of classification performance parameters

Description
Calculates and returns performance parameters for classification result (e.g. number of false negatives, false positives, sensitivity, specificity, etc.).

Usage

getClassificationPerformance(c.ref, c.pred)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c.ref</td>
<td>reference class values for objects (vector with numeric or text values)</td>
</tr>
<tr>
<td>c.pred</td>
<td>predicted class values for objects (array nobj x ncomponents x nclasses)</td>
</tr>
</tbody>
</table>

Details
The function is called automatically when a classification result with reference values is created, for example when applying a plsda or simca models.
Value

Returns a list with following fields:
getConfusionMatrix

Confusion matrix for classification results

Description

Confusion matrix for classification results

Usage

getConfusionMatrix(obj, ...)

Arguments

obj classification results (object of class simcares, simcamres, etc)

... other parameters.

Details

Returns confusion matrix for classification results represented by the object.

getConfusionMatrix

Confusion matrix for classification results

Description

The columns of the matrix correspond to classification results, rows - to the real classes. In case of soft classification with multiple classes (e.g. SIMCAM) sum of values for every row will not correspond to the total number of class members as the same object can be classified as a member of several classes or non of them.

Usage

## S3 method for class 'classes'
getConfusionMatrix(obj, ncomp = NULL, ...)
getMainTitle

Arguments

- **obj**: classification results (object of class `simcares`, `simcamres`, etc)
- **ncomp**: number of components to make the matrix for (NULL - use selected for a model).
- **...**: other arguments

Details

Returns confusion matrix for classification results represented by the object.

---

getMainTitle  
*Get main title*

Description

returns main title for a plot depending on a user choice

Usage

getMainTitle(main, ncomp, default)

Arguments

- **main**: main title of a plot, provided by user
- **ncomp**: number of components to select, provided by user
- **default**: default title for the plot

Details

Depedning on a user choice it returns main title for a plot

---

getProbabilities  
*Get class belonging probability*

Description

Compute class belonging probabilities for classification results.

Usage

getProbabilities(obj, ...)

Arguments

- **obj**: an object with classification results (e.g. SIMCA)
- **...**: other parameters
getProbabilities.simca

*Probability of class belonging for PCA/SIMCA results*

**Description**
Computes probability of class belonging for each object based on Q and T2 residuals

**Usage**
```r
## S3 method for class 'simca'
getProbabilities(obj, ncomp, Q, T2, ...)
```

**Arguments**
- `obj`: object with SIMCA model
- `ncomp`: number of components to compute the probabilities for.
- `Q`: vector with Q values for selected component
- `T2`: vector with T2 values for selected component
- `...`: other arguments

getRegcoeffs

*Get regression coefficients*

**Description**
Generic function for getting regression coefficients from PLS model

**Usage**
```r
getRegcoeffs(obj, ...)
```

**Arguments**
- `obj`: a PLS model
- `...`: other parameters
Description

Returns a matrix with regression coefficients for the PLS model which can be applied to a data directly.

Usage

## S3 method for class 'pls'
getRegcoeffs(obj, ncomp = NULL, ny = NULL,  
               full = FALSE, alpha = obj$coeffs.alpha, ...)

Arguments

- obj: a PLS model (object of class pls)
- ncomp: number of components to return the coefficients for
- ny: if y is multivariate which variables you want to see the coefficients for
- full: if TRUE the method also shows p-values and t-values as well as confidence intervals for the coefficients (if available)
- alpha: significance level for confidence intervals (a number between 0 and 1, e.g. for 95% alpha = 0.05)
- ...: other parameters

Details

The method recalculates the regression coefficients found by the PLS algorithm taking into account centering and scaling of predictors and responses, so the matrix with coefficients can be applied directly to original data (yp = Xb).

If number of components is not specified, the optimal number, selected by user or identified by a model will be used.

If Jack-knifing method was used to get statistics for the coefficient the method returns all statistics as well (p-value, t-value, confidence interval). In this case user has to specified a number of y-variable (if there are many) to get the statistics and the coefficients for. The confidence interval is computed for unstandardized coefficients.

Value

A matrix with regression coefficients and (optionally) statistics.
**getSelectedComponents.classres**

Get selected components

### Description

returns number of components depending on a user choice

### Usage

```r
getSelectedComponents(obj, ncomp = NULL)
```

### Arguments

- **obj**: an MDA model or result object (e.g. pca, pls, simca, etc)
- **ncomp**: number of components to select, provided by user

### Details

Depending on a user choice it returns optimal number of component for the model (if user did not provide any value) or check the user choice for correctness and returns it back

---

**getSelectedComponents.classres**

Get selected components

### Description

Returns number of components depending on user selection and object properties

### Usage

```r
getSelectedComponents.classres(obj, ncomp = NULL)
```

### Arguments

- **obj**: object with classification results (e.g. plsdares or simcamres).
- **ncomp**: number of components specified by user.

### Details

This is a technical function used for selection proper value for number of components in plotting functions.
getSelectivityRatio  Selectivity ratio

Description

Generic function for returning selectivity ratio values for regression model (PCR, PLS, etc)

Usage

getSelectivityRatio(obj, ...)

Arguments

obj a regression model
...
other parameters

getSelectivityRatio.pls

Selectivity ratio for PLS model

Description

Returns vector with selectivity ratio values for given number of components and response variable

Usage

## S3 method for class 'pls'
getSelectivityRatio(obj, ncomp = NULL, ny = 1, ...)

Arguments

obj a PLS model (object of class pls)
ncomp number of components to get the values for (if NULL user selected as optimal will be used)
ny which response to get the values for (if y is multivariate)
...
other parameters

Value

vector with selectivity ratio values

References

getVIPScores

**VIP scores**

**Description**

Generic function for returning VIP scores values for regression model (PCR, PLS, etc)

**Usage**

getVIPScores(obj, ...)

**Arguments**

- obj: a regression model
- ...: other parameters

---

getVIPScores.pls

**VIP scores for PLS model**

**Description**

Returns vector with VIP scores values for given number of components and response variable

**Usage**

```r
## S3 method for class 'pls'
getVIPScores(obj, ny = 1, ...)
```

**Arguments**

- obj: a PLS model (object of class pls)
- ny: which response to get the values for (if y is multivariate)
- ...: other parameters

**Value**

vector with VIP scores values

**References**

**imshow**

**Description**

show image data as an image

**Usage**

```r
imshow(data, channels = 1, show.excluded = FALSE, main = NULL, 
       colmap = "jet")
```

**Arguments**

- **data**: data with image
- **channels**: indices for one or three columns to show as image channels
- **show.excluded**: logical, if TRUE the method also shows the excluded (hidden) pixels
- **main**: main title for the image
- **colmap**: colormap using to show the intensity levels

---

**ipls**

**Variable selection with interval PLS**

**Description**

Applies iPLS algorithm to find variable intervals most important for prediction

**Usage**

```r
ipls(x, y, glob.ncomp = 10, center = T, scale = F, cv = 10, 
     exclcols = NULL, exclrows = NULL, int.ncomp = glob.ncomp, 
     int.num = NULL, int.width = NULL, int.limits = NULL, 
     int.riter = NULL, ncomp.selcrit = "min", method = "forward", 
     silent = F)
```

**Arguments**

- **x**: a matrix with predictor values
- **y**: a vector with response values
- **glob.ncomp**: maximum number of components for a global PLS model
- **center**: logical, center or not the data values
- **scale**: logical, standardize or not the data values
- **cv**: number of segments for cross-validation (1 - full CV)
exclcols columns of x to be excluded from calculations (numbers, names or vector with logical values)
exclrows rows to be excluded from calculations (numbers, names or vector with logical values)
int.ncomp maximum number of components for interval PLS models
int.num number of intervals
int.width width of intervals
int.limits a two column matrix with manual intervals specification
int.niter maximum number of iterations (if NULL it will be the same as number of intervals)
ncomp.selcrit criterion for selecting optimal number of components (‘min’ for minimum of RMSECV)
method iPLS method (‘forward’ or ‘backward’)
silent logical, show or not information about selection process

Details
The algorithm splits the predictors into several intervals and tries to find a combination of the intervals, which gives best prediction performance. There are two selection methods: "forward" when the intervals are successively included, and "backward" when the intervals are successively excluded from a model. On the first step the algorithm finds the best (forward) or the worst (backward) individual interval. Then it tests the others to find the one which gives the best model in a combination with the already selected/excluded one. The procedure continues until the maximum number of iteration is reached.

There are several ways to specify the intervals. First of all either number of intervals (int.num) or width of the intervals (int.width) can be provided. Alternatively one can specify the limits (first and last variable number) of the intervals manually with int.limits.

Value
object of 'ipls' class with several fields, including:

var.selected a vector with indices of selected variables
int.selected a vector with indices of selected intervals
int.num total number of intervals
int.width width of the intervals
int.limits a matrix with limits for each interval
int.stat a data frame with statistics for the selection algorithm
glob.stat a data frame with statistics for the first step (individual intervals)
gm global PLS model with all variables included
om optimized PLS model with selected variables
ipls.backward  

**References**


**Examples**

```R
library(mdatools)

## forward selection for simdata

data(simdata)
Xc = simdata$spectra[, 3, drop = FALSE]
yc = simdata$conc

# run iPLS and show results
im = ipls(Xc, yc, int.ncomp = 5, int.num = 10, cv = 4, method = "forward")
summary(im)
plot(im)

# show "developing" of RMSECV during the algorithm execution
plotRMSE(im)

# plot predictions before and after selection
par(mfrow = c(1, 2))
plotPredictions(im$gm)
plotPredictions(im$om)

# show selected intervals on spectral plot
ind = im$var.selected
mspectrum = apply(Xc, 2, mean)
plot(simdata$wavelength, mspectrum, type = 'l', col = 'lightblue')
points(simdata$wavelength[ind], mspectrum[ind], pch = 16, col = 'blue')
```

---

**ipls.backward**

*Runs the backward iPLS algorithm*

**Description**

Runs the backward iPLS algorithm

**Usage**

`ipls.backward(x, y, obj)`

**Arguments**

- **x**: a matrix with predictor values
- **y**: a vector with response values
- **obj**: object with initial settings for iPLS algorithm
ipls.forward

**Runs the forward iPLS algorithm**

**Description**

Runs the forward iPLS algorithm

**Usage**

ipls.forward(x, y, obj)

**Arguments**

- **x**: a matrix with predictor values
- **y**: a vector with response values
- **obj**: object with initial settings for iPLS algorithm

ldecomp

**Linear decomposition of data**

**Description**

Creates an object of ldecomp class.

**Usage**

ldecomp(scores = NULL, residuals = NULL, loadings = NULL,  
ncomp.selected = NULL, attrs = NULL, tnorm = NULL, dist = NULL,  
var = NULL, cal = FALSE, totvar = NULL)

**Arguments**

- **scores**: matrix with score values (nobj x ncomp).
- **residuals**: matrix with data residuals
- **loadings**: matrix with loading values (nvar x ncomp).
- **ncomp.selected**: number of selected components
- **attrs**: list with attributes of original dataset
- **tnorm**: singular values for score normalization
- **dist**: list with calculated T2 and Q values (e.g. for CV)
- **var**: list with explained and cumulative explained variance (e.g. for CV)
- **cal**: logical, true if data is for calibration of a LDECOMP based model
- **totvar**: full variance of original data, preprocessed and centered
Details

`ldecomp` is a general class for decomposition `X = TP' + E`. Here, `X` is a data matrix, `T` - matrix with scores, `P` - matrix with loadings and `E` - matrix with residuals. It is used, for example, for PCA results (`pcares`), in PLS and other methods. The class also includes methods for calculation and plotting residuals, variances, and so on.

There is no need to use the `ldecomp` manually. For example, when build PCA model with `pca` or apply it to a new data, the results will automatically inherit all methods of `ldecomp`.

Value

Returns an object (list) of `ldecomp` class with following fields:

- `scores` matrix with score values (nobj x ncomp).
- `residuals` matrix with data residuals (nobj x nvar).
- `T2` matrix with T2 distances (nobj x ncomp).
- `Q` matrix with Q statistic (nobj x ncomp).
- `tnorm` vector with singular values used for scores normalization.
- `ncomp_selected` selected number of components.
- `expvar` explained variance for each component.
- `cumexpvar` cumulative explained variance.
- `modpower` modelling power of variables.

Description

Computes residual distances (Q and T2) and modelling power for a data decomposition `X = TP' + E`.

Usage

```r
ldecomp.getDistances(scores, loadings, residuals, tnorm = NULL, cal = FALSE)
```

Arguments

- `scores` matrix with scores (T).
- `loadings` matrix with loadings (P).
- `residuals` matrix with residuals (E).
- `tnorm` vector with singular values for scores normalisation
- `cal` if TRUE method will realize that these distances are calculated for calibration set
Details
The distances are calculated for every 1:n components, where n goes from 1 to ncomp (number of columns in scores and loadings).

Value
Returns a list with Q, Qvar, T2 and modelling power values for each component.

ldecomp.getVariances Explained variance for linear decomposition

Description
Computes explained variance and cumulative explained variance for a data decomposition X = TP' + E.

Usage
ldecomp.getVariances(Q, totvar)

Arguments
Q Q values (squared residuals distance from object to component space).
totvar Total variance of the original data (after preprocessing).

Value
Returns a list with two vectors.

ldecomp.plotLimits Shows lines with critical limits on residuals plot

Description
Shows lines with critical limits on residuals plot

Usage
ldecomp.plotLimits(lim, lim.type, lim.col, lim.lwd, lim.lty)

Arguments
lim matrix with residual limits (2x2)
lim.type type of limits
lim.col vector with two values - line color for extreme and outlier borders
lim.lwd vector with two values - line width for extreme and outlier borders
lim.lty vector with two values - line type for extreme and outlier borders
**mda.cbind**

A wrapper for `cbind()` method with proper set of attributes

**Description**

A wrapper for `cbind()` method with proper set of attributes

**Usage**

`mda.cbind(...)`

**Arguments**

`...` datasets (data frames or matrices) to bind

**Value**

the merged datasets

---

**mda.data2im**

Convert data matrix to an image

**Description**

Convert data matrix to an image

**Usage**

`mda.data2im(data)`

**Arguments**

`data` data matrix
mda.df2mat

Convert data frame to a matrix

Description

The function converts data frame to a numeric matrix.

Usage

mda.df2mat(x, full = FALSE)

Arguments

x a data frame
full logical, if TRUE number of dummy variables for a factor will be the same as number of levels, otherwise by one smaller

Details

If one or several columns of the data frame are factors they will be converted to a set of dummy variables. If any columns/rows were hidden in the data frame they will remain hidden in the matrix. If there are factors among the hidden columns, the corresponding dummy variables will be hidden as well.

All other attributes (names, axis names, etc.) will be inherited.

Value

a numeric matrix

mda.exclcols

Exclude/hide columns in a dataset

Description

Exclude/hide columns in a dataset

Usage

mda.exclcols(x, ind)

Arguments

x dataset (data frame or matrix).
ind indices of columns to exclude (numbers, names or logical values)
Details

The method assign attribute 'exclcols', which contains number of columns, which should be excluded/hidden from calculations and plots (without removing them physically). The argument `ind` should contain column numbers (excluding already hidden), names or logical values.

Value
dataset with excluded columns

`mda.exclcols` Exclude/hide rows in a dataset

Description

Exclude/hide rows in a dataset

Usage

`mda.exclrows(x, ind)`

Arguments

x dataset (data frame or matrix).
ind indices of rows to exclude (numbers, names or logical values)

Details

The method assign attribute 'exclrows', which contains number of rows, which should be excluded/hidden from calculations and plots (without removing them physically). The argument `ind` should contain rows numbers (excluding already hidden), names or logical values.

Value
dataset with excluded rows

`mda.exclrows` Exclude/hide rows in a dataset

Description

Returns a list with important data attributes (name, xvalues, excluded rows and columns, etc.)

Usage

`mda.getattr(x)`

Arguments

x a dataset
mda.getexclind  
*Get indices of excluded rows or columns*

**Description**

Get indices of excluded rows or columns

**Usage**

`mda.getexclind(excl, names, n)`

**Arguments**

- `excl` vector with excluded values (logical, text or numbers)
- `names` vector with names for rows or columns
- `n` number of rows or columns

mda.im2data  
*Convert image to data matrix*

**Description**

Convert image to data matrix

**Usage**

`mda.im2data(img)`

**Arguments**

- `img` an image (3-way array)
**mda.inclcols**

*Include/unhide the excluded columns*

**Description**

include columns specified by user (earlier excluded using mda.exclcols)

**Usage**

```r
mda.inclcols(x, ind)
```

**Arguments**

- `x` dataset (data frame or matrix).
- `ind` number of excluded columns to include

**Value**

dataset with included columns.

---

**mda.inclrows**

*Include/unhide the excluded rows*

**Description**

include rows specified by user (earlier excluded using mda.exclrows)

**Usage**

```r
mda.inclrows(x, ind)
```

**Arguments**

- `x` dataset (data frame or matrix).
- `ind` number of excluded rows to include

**Value**

dataset with included rows
**mda.rbind**

*A wrapper for rbind() method with proper set of attributes*

**Description**

A wrapper for rbind() method with proper set of attributes

**Usage**

```r
dna.rbind(...)```

**Arguments**

- `...` datasets (data frames or matrices) to bind

**Value**

the merged datasets

---

**mda.setattr**

*Set data attributes*

**Description**

Set most important data attributes (name, xvalues, excluded rows and columns, etc.) to a dataset

**Usage**

```r
dna.setattr(x, attrs, type = "all")```

**Arguments**

- `x` a dataset
- `attrs` list with attributes
- `type` a text variable telling which attributes to set (`"all"`, "row", "col")
**mda.setimbg**

Remove background pixels from image data

**Description**

Remove background pixels from image data

**Usage**

```r
mda.setimbg(data, bgpixels)
```

**Arguments**

- `data`: a matrix with image data
- `bgpixels`: vector with indices or logical values corresponding to background pixels

---

**mda.show**

Wrapper for `show()` method

**Description**

Wrapper for `show()` method

**Usage**

```r
mda.show(x, n = 50)
```

**Arguments**

- `x`: data set
- `n`: number of rows to show
**mda.subset**

* A wrapper for subset() method with proper set of attributes

### Description

A wrapper for subset() method with proper set of attributes

### Usage

```r
mda.subset(x, subset = NULL, select = NULL)
```

### Arguments

- **x**: dataset (data frame or matrix)
- **subset**: which rows to keep (indices, names or logical values)
- **select**: which columns to select (indices, names or logical values)

### Details

The method works similar to the standard subset() method, with minor differences. First of all it keeps (and correct, if necessary) all important attributes. If only columns are selected, it keeps all excluded rows as excluded. If only rows are selected, it keeps all excluded columns. If both rows and columns are selected it removed all excluded elements first and then makes the subset.

The parameters subset and select may each be a vector with numbers or names without excluded elements, or a logical expression.

### Value

- a data with the subset

---

**mda.t**

* A wrapper for t() method with proper set of attributes

### Description

A wrapper for t() method with proper set of attributes

### Usage

```r
mda.t(x)
```

### Arguments

- **x**: dataset (data frames or matrices) to transpose
Value

the transposed dataset

mdaplot

Plotting function for a single set of objects

Description

mdaplot is used to make different kinds of plot for one set of data objects.

Usage

mdaplot(data = NULL, plot.data = NULL, type = "p", pch = 16, col = NULL, lty = 1, lwd = 1, bwd = 0.8, cgroup = NULL, xlim = NULL, ylim = NULL, colmap = "default", labels = NULL, main = NULL, xlab = NULL, ylab = NULL, show.labels = F, show.colorbar = T, show.lines = F, show.grid = T, show.axes = T, xticks = NULL, yticks = NULL, xticklabels = NULL, yticklabels = NULL, xlas = 0, ylas = 0, lab.col = "darkgray", lab.cex = 0.65, show.excluded = FALSE, col.excluded = "#E0E0E0", nbins = 256, colramp = mdaplot.getColors, force.x.values = NA, opacity = 1, ...)

Arguments

data a vector, matrix or a data.frame with data values.
plot.data a list of parameters and values obtained after preprocessing of original data provided by a user (if NULL it will be created automatically)
type type of the plot ("p", "l", "b", "h", "e", "i").
pch a character for markers (same as plot parameter).
col a color for markers or lines (same as plot parameter).
lty the line type (same as plot parameter).
lwd the line width (thickness) (same as plot parameter).
bwd a width of a bar as a percent of a maximum space available for each bar.
cgroup a vector with values to use for make color groups.
xlim limits for the x axis (if NULL, will be calculated automatically).
ylim limits for the y axis (if NULL, will be calculated automatically).
colmap a colormap to use for coloring the plot items.
labels a vector with text labels for data points or one of the following: 'names', 'indices', 'values'.
main an overall title for the plot (same as plot parameter).
xlab a title for the x axis (same as plot parameter).
**mdaplot**

- **ylab**: a title for the y axis (same as plot parameter).
- **show.labels**: logical, show or not labels for the data objects.
- **show.colorbar**: logical, show or not colorbar legend if color grouping is on.
- **show.lines**: vector with two coordinates (x, y) to show horizontal and vertical line across the point.
- **show.grid**: logical, show or not a grid for the plot.
- **show.axes**: logical, make a normal plot or show only elements (markers, lines, bars) without axes.
- **xticks**: values for x ticks
- **yticks**: values for y ticks
- **xticklabels**: labels for x ticks.
- **yticklabels**: labels for y ticks.
- **xlas**: orientation of xticklabels
- **ylas**: orientation of yticklabels
- **lab.col**: color for data point labels.
- **lab.cex**: size for data point labels.
- **show.excluded**: logical, show or hide rows marked as excluded (attribute ‘exclrows’)
- **col.excluded**: color for the excluded objects (rows)
- **nbins**: if scatter density plot is shown, number of segments to split the plot area into (see also ?smoothScatter)
- **colramp**: Colramp function for density scatter plot
- **force.x.values**: vector with corrected x-values for a bar plot (do not specify this manually)
- **opacity**: opacity for plot colors (value between 0 and 1)
- **...**: other plotting arguments.

**Details**

Most of the parameters are similar to what are used with standard plot function. The differences are described below.

The function makes a plot of one set of objects. It can be a set of points (scatter plot), bars, lines, scatter-lines, errorbars or an image. The data is organized as a data frame, matrix or vector. For scatter and only first two columns will be used, for bar plot only values from the first row. It is recommended to use `mda.subset` method if plot should be made only for a subset of the data, especially if you have any excluded rows or columns or other special attributed, described in the Bookdown tutorial.

If data is a data frame and contains one or more factors, they will be converted to a dummy variables (using function `mda.df2mat`) and appears at the end (last columns) if line or bar plot is selected.

The function allows to colorize lines and points according to values of a parameter `cgroup`. The parameter must be a vector with the same elements as number of objects (rows) in the data. The values are divided into up to eight intervals and for each interval a particular color from a selected color scheme is assigned. Parameter `show.colorbar` allows to turn off and on a color bar legend for this option.
The used color scheme is defined by the `colmap` parameter. The default scheme is based on color brewer (colorbrewer2.org) diverging scheme with eight colors. There is also a gray scheme (`colmap = 'gray'`) and user can define its own just by specifying the needed sequence of colors (e.g. `colmap = c('red', 'yellow', 'green'), two colors is minimum). The scheme will then be generated automatically as a gradient among the colors.

Besides that the function allows to change tick values and corresponding tick labels for x and y axis, see Bookdown tutorial for more details.

Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

See Also

`mdaplotg` - to make plots for several sets of data objects (groups of objects).

Examples

```r
# See all examples in the tutorial.
```

---

### `mdaplot.areColors` Check color values

**Description**

Checks if elements of argument are valid color values

**Usage**

```r
mdaplot.areColors(palette)
```

**Arguments**

- `palette` vector with possibly color values (names, RGB, etc.)
mdaplot.formatValues  *Format vector with numeric values*

**Description**

Format vector with values, so only significant decimal numbers are left.

**Usage**

```r
mdaplot.formatValues(data, round.only = F, digits = 3)
```

**Arguments**

- `data` vector or matrix with values
- `round.only` logical, do formatting or only round the values
- `digits` how many significant digits take into account

**Details**

Function takes into account difference between values and the values themselves.

**Value**

matrix with formatted values

mdaplot.getAxesLim  *Calculate axes limits*

**Description**

Calculates axes limits depending on data values that have to be plotted, extra plot elements that have to be shown and margins.

**Usage**

```r
mdaplot.getAxesLim(x.values, y.values, lower = NULL, upper = NULL, show.colorbar = F, show.lines = F, legend = NULL, show.legend = F, legend.position = "topright", show.labels = F)
```
**mdaplot.getColors**

*Arguments*

- `x.values` a vector with x values.
- `y.values` a vector or a matrix with y values.
- `lower` a lower margin for y limits.
- `upper` an upper margin for y limits.
- `show.colorbar` logical, show or not the colorbar on the plot.
- `show.lines` logical or numeric with line coordinates to be shown on the plot.
- `legend` vector with legend items.
- `show.legend` logical, show or not legend on the plot.
- `legend.position` position of the legend (see `mdaplotg` for details).
- `show.labels` logical, show or not labels for the data objects.

*Details*

Data can be a list with several matrices or just one matrix. The matrices can have single.x configuration, where first column is x values and the others are y values or normal configuration, where every odd column is x values and every even is corresponding y values.

*Value*

Returns a list with four limits for the x and y axes.

---

**mdaplot.getColors**  
*Color values for plot elements*

**Description**

Generate vector with color values for plot objects (lines, points, bars), depending on number of groups for the objects.

**Usage**

```
mdaplot.getColors(ngroups = 1, cgroup = NULL, colmap = "default", opacity = 1)
```

**Arguments**

- `ngroups` number of groups.
- `cgroup` vector of values, used for color grouping of plot points or lines.
- `colmap` which colormap to use (`'default'`, `'gray'`, or user defined in form `c('color1', 'color2', ...)`).
- `opacity` opacity for colors (between 0 and 1)
mdaplot.plotAxes  

Description

Creates an empty axes plane for given parameters

Usage

mdaplot.plotAxes(xticklabels = NULL, yticklabels = NULL, xticks = NULL, yticks = NULL, lim = NULL, main = NULL, xlab = NULL, ylab = NULL, xlas = 0, ylas = 0)

Arguments

- xticklabels: labels for x ticks
- yticklabels: labels for y ticks
- xticks: values for x ticks
- yticks: values for y ticks
- lim: vector with limits for x and y axis
- main: main title for the plot
- xlab: label for x axis
- ylab: label for y axis
- xlas: orientation of xticklabels
- ylas: orientation of yticklabels

mdaplot.showColorbar  

Description

Shows a colorbar if plot has color grouping of elements (points or lines).

Usage

mdaplot.showColorbar(cgroup, colmap = "default", lab.col = "darkgray", lab.cex = 0.65)
mdaplot.showGrid

**Arguments**
- `cgroup` a vector with values used to make color grouping of the elements
- `colmap` a colormap to be used for color generation
- `lab.col` color for legend labels
- `lab.cex` size for legend labels

**Description**
Shows grid for a plot

**Usage**
```r
mdaplot.showGrid(lwd = 0.5)
```

**Arguments**
- `lwd` line width for the grid

mdaplot.showLabels

**Plot labels** Shows labels for data elements (points, bars) on a plot.

**Description**
Plot labels Shows labels for data elements (points, bars) on a plot.

**Usage**
```r
mdaplot.showLabels(x.values, y.values, labels, pos = 3, cex = 0.65,
                    col = "darkgray", type = NULL)
```

**Arguments**
- `x.values` a vector with x-values
- `y.values` a vector with y-values
- `labels` a vector with labels
- `pos` position of the labels relative to the points
- `cex` size of the labels text
- `col` color of the labels text
- `type` type of the plot
Details

Rownames of matrix data are used as labels. If matrix has no rownames, row numbers will be used instead.

---

$mdbplot.showLegend$  \hspace{1cm} \textit{Plot legend}

Description

Shows a legend for plot elements or their groups.

Usage

$mdbplot.showLegend(legend, col, pch = NULL, lty = NULL, lwd = NULL,$
\hspace{1cm} \text{bty = "o", position = "topright", plot = T)}

Arguments

- \textit{legend} \hspace{1cm} \text{vector with text elements for the legend items}
- \textit{col} \hspace{1cm} \text{vector with color values for the legend items}
- \textit{pch} \hspace{1cm} \text{vector with marker symbols for the legend items}
- \textit{lty} \hspace{1cm} \text{vector with line types for the legend items}
- \textit{lwd} \hspace{1cm} \text{vector with line width values for the legend items}
- \textit{bty} \hspace{1cm} \text{border type for the legend}
- \textit{position} \hspace{1cm} \text{legend position ('topright', 'topleft', 'bottomright', 'bottomleft', 'top', 'bottom')}
- \textit{plot} \hspace{1cm} \text{logical, show legend or just calculate and return its size}

---

$mdbplot.showLines$  \hspace{1cm} \textit{Plot lines}

Description

Shows horizontal and vertical lines on a plot.

Usage

$mdbplot.showLines(point, lty = 2, lwd = 0.75, col = rgb(0.2, 0.2,$
\hspace{1cm} \text{0.2))$
**Arguments**

- **point**: vector with two values: x coordinate for vertical point y for horizontal
- **lty**: line type
- **lwd**: line width
- **col**: color of lines

**Details**

If it is needed to show only one line, the other coordinate shall be set to NA.

---

**Description**

Shows linear fit line for data points.

**Usage**

```r
mdaplot.showRegressionLine(data, lty = 1, lwd = 1, 
                          colmap = "default", col = NULL)
```

**Arguments**

- **data**: data values
- **lty**: line type
- **lwd**: line width
- **colmap**: color map
- **col**: color of lines

---

**Description**

`mdaplotg` is used to make different kinds of plots or their combination for several sets of objects.
Usage

mdaplotg(data, groupby = NULL, type = "p", pch = 16, lty = 1, lwd = 1, bwd = 0.8, legend = NULL, xlab = NULL, ylab = NULL, main = NULL, labels = NULL, ylim = NULL, xlim = NULL, colmap = "default", legend.position = "topright", show.legend = T, show.labels = F, show.lines = F, show.grid = T, xticks = NULL, xticklabels = NULL, yticks = NULL, yticklabels = NULL, show.excluded = FALSE, lab.col = "darkgray", lab.cex = 0.65, xlas = 1, ylas = 1, ...)

Arguments

data a matrix, data frame or a list with data values (see details below).
groupby one or several factors used to create groups of data matrix rows (works if data is a matrix)
type type of the plot ("p", "l", "b", "h", "e").
pch a character for markers (same as plot parameter).
lty the line type (same as plot parameter).
lwd the line width (thickness) (same as plot parameter).
bwd a width of a bar as a percent of a maximum space available for each bar.
legend a vector with legend elements (if NULL, no legend will be shown).
xlab a title for the x axis (same as plot parameter).
ylab a title for the y axis (same as plot parameter).
main an overall title for the plot (same as plot parameter).
labels what to use as labels ("names" - row names, "indices" - row indices, "values" - values).
ylim limits for the y axis (if NULL, will be calculated automatically).
xlim limits for the x axis (if NULL, will be calculated automatically).
colmap a colormap to use for coloring the plot items.
legend.position position of the legend ("topleft", "topright", "top", "bottomleft", "bottomright", "bottom").
show.legend logical, show or not legend for the data objects.
show.labels logical, show or not labels for the data objects.
show.lines vector with two coordinates (x, y) to show horizontal and vertical line cross the point.
show.grid logical, show or not a grid for the plot.
xticks tick values for x axis.
xticklabels labels for x ticks.
yticks tick values for y axis.
yticklabels labels for y ticks.
show.excluded logical, show or hide rows marked as excluded (attribute 'exclrows')
lab.col color for data point labels.
lab.cex size for data point labels.
xlas orientation of xticklabels
ylas orientation of yticklabels
... other plotting arguments.

Details

The mdaplotg function is used to make a plot with several sets of objects. Simply speaking, use it when you need a plot with legend. For example to show line plot with spectra from calibration and test set, scatter plot for height and weight values for women and men, and so on.

Most of the parameters are similar to mdaplot, the difference is described below.

The data should be organized as a list, every item is a matrix with data for one set of objects. Alternatively you can provide data as a matrix and use parameter groupby to create groups. See tutorial for more details.

There is no color grouping option, because color is used to separate the sets. Marker symbol, line style and type, etc. can be defined as a single value (one for all sets) and as a vector with one value for each set.

Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

Package for Multivariate Data Analysis (Chemometrics)

Description

This package contains classes and functions for most common methods used in Chemometrics. For a complete list of functions, use library(help = 'mdatools').

Details

The project is hosted on GitHub, there you can also find a Bookdown user tutorial (https://svkucheryavski.github.io/mdatools/) explaining most important features of the package.

Every method is represented by two classes: a model class for keeping all parameters and information about the model, and a class for keeping and visualising results of applying the model to particular data values.

Every model class, e.g. pls, has all needed functionality implemented as class methods, including model calibration, validation (test set and cross-validation), visualisation of the calibration and validation results with various plots and summary statistics.

So far the following modelling methods are implemented:
Methods for data preprocessing:

- prep.autoscale: data mean centering and/or standardization.
- prep.savog: Savitzky-Golay transformation.
- prep.snv: Standard normal variate.
- prep.msc: Multiplicative scatter correction.
- prep.norm: Spectra normalization.

All plotting methods are based on two functions, `mdaplot` and `mdaplotg`. The functions extend the basic functionality of R plots and allow to make automatic legend and color grouping of data points or lines with colorbar legend, automatically adjust axes limits when several data groups are plotted and so on.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)
scale logical, do standardization of data or not.
cv number of segments for random cross-validation (1 for full cross-validation).
exclrows rows to be excluded from calculations (numbers, names or vector with logical values)
exclcols columns to be excluded from calculations (numbers, names or vector with logical values)
x.test a numerical matrix with test data.
method method to compute principal components ('svd', 'nipals').
r and vector with parameters for randomized PCA methods (if NULL, conventional PCA is used instead)
lim.type which method to use for calculation of critical limits for residuals (see details)
alpha significance level for calculating critical limits for T2 and Q residuals.
gamma significance level for calculating outlier limits for T2 and Q residuals.
info a short text line with model description.

Details

By default pca uses number of components (ncomp) as a minimum of number of objects - 1, number of variables and default or provided value. Besides that, there is also a parameter for selecting an optimal number of components (ncomp.selected). The optimal number of components is used to build a residuals plot (with Q residuals vs. Hotelling T2 values), calculate confidence limits for Q residuals, as well as for SIMCA classification.

You can provide number, names or logical values to exclude rows or columns from calibration and validation of PCA model. In this case the outcome, e.g. scores and loadings will correspond to the original size of the data, but:

1. Loadings (and all performance statistics) will be computed without excluded objects and variables
2. Matrix with loadings will have zero values for the excluded variables and the corresponding columns will be hidden.
3. Matrix with scores will have score values calculated for the hidden objects but the rows will be hidden.

You can see scores and loadings for hidden rows and columns by using parameter 'show.excluded = T' in plots. If you see other packages to make plots (e.g. ggplot2) you will not be able to distinguish between hidden and normal objects.

By default loadings are computed for the original dataset using either SVD or NIPALS algorithm. However, for datasets with large number of rows (e.g. hyperspectral images), there is a possibility to run algorithms based on random permutations [1, 2]. In this case you have to define parameter rand as a vector with two values: p - oversampling parameter and k - number of iterations. Usually rand = c(15, 0) or rand = c(5, 1) are good options, which give quite precise solution using several times less computational time. It must be noted that statistical limits for residuals will not be computed in this case.
There are several ways to calculate critical limits for Q and T2 residuals. In mdatools you can specify one of the following methods via parameter lim.type: 'jm' - method based on Jackson-Mudholkar approach [3], 'chisq' - method based on chi-square distribution [4] and 'ddrobust' and 'ddmoments' - both related to data driven method proposed by Pomerantsev and Rodionova [5]. The 'ddmoments' is based on method of moments for estimation of distribution parameters while 'ddrobust' is based on robust estimation.

It must be noted that the first two methods calculate limits for Q-residuals only, assuming, that limits for T2 residuals must be computed using Hotelling’s T-squared distribution. The methods based on the data driven approach calculate limits for both Q and T2 residuals based on chi-square distribution and parameters estimated from the calibration data.

The critical limits are calculated for a significance level defined by parameter 'alpha'. You can also specify another parameter, 'gamma', which is used to calculate acceptance limit for outliers (shown as dashed line on residuals plot).

You can also recalculate the limits for existent model by using different values for alpha and gamme, without recomputing the model itself. In this case use the following code (it is assumed that you current PCA/SIMCA model is stored in variable m): m = setResLimits(m, alpha, gamma).

In case of PCA the critical limits are just shown on residual plot as lines and can be used for detection of extreme objects (solid line) and outliers (dashed line). When PCA model is used for classification in SIMCA (see simca) the limits are utilized for classification of objects.

Value

Returns an object of pca class with following fields:

- ncomp: number of components included to the model.
- ncomp.selected: selected (optimal) number of components.
- loadings: matrix with loading values (nvar x ncomp).
- eigenvals: vector with eigenvalues for all existent components.
- expvar: vector with explained variance for each component (in percent).
- cumexpvar: vector with cumulative explained variance for each component (in percent).
- T2lim: statistical limit for T2 distance.
- Qlim: statistical limit for Q residuals.
- info: information about the model, provided by user when build the model.
- calres: an object of class pcarest with PCA results for a calibration data.
- testres: an object of class pcarest with PCA results for a test data, if it was provided.
- cvres: an object of class pcarest with PCA results for cross-validation, if this option was chosen.

More details and examples can be found in the Bookdown tutorial.

Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)
References


See Also

Methods for pca objects:

- `plot.pca` makes an overview of PCA model with four plots.
- `summary.pca` shows some statistics for the model.
- `selectCompNum.pca` set number of optimal components in the model
- `setResLimits.pca` set critical limits for residuals
- `predict.pca` applies PCA model to a new data.
- `plotScores.pca` shows scores plot.
- `plotLoadings.pca` shows loadings plot.
- `plotVariance.pca` shows explained variance plot.
- `plotCumVariance.pca` shows cumulative explained variance plot.
- `plotResiduals.pca` shows Q vs. T2 residuals plot.

Most of the methods for plotting data are also available for PCA results (`pcares`) objects. Also check `pca.mvreplace`, which replaces missing values in a data matrix with approximated using iterative PCA decomposition.

Examples

```r
library(mdatools)
### Examples for PCA class

### 1. Make PCA model for People data with autoscaling
### and full cross-validation

data(people)
model = pca(people, scale = TRUE, cv = 1, info = 'Simple PCA model')
model = selectCompNum(model, 4)
summary(model)
plot(model, show.labels = TRUE)

### 3. Show scores and loadings plots for the model
par(mfrow = c(2, 2))
plotScores(model, comp = c(1, 3), show.labels = TRUE)
plotScores(model, comp = 2, type = 'h', show.labels = TRUE)
plotLoadings(model, comp = c(1, 3), show.labels = TRUE)
plotLoadings(model, comp = c(1, 2), type = 'h', show.labels = TRUE)
```
par(mfrow = c(1, 1))

## 4. Show residuals and variance plots for the model
par(mfrow = c(2, 2))
plotVariance(model, type = 'h')
plotCumVariance(model, show.labels = TRUE, legend.position = 'bottomright')
plotResiduals(model, show.labels = TRUE)
plotResiduals(model, ncomp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

describe PCA model calibration

Description
Calibrates (builds) a PCA model for given data and parameters

Usage
pca.cal(x, ncomp, center, scale, method, exclcols = NULL,
exclrows = NULL, cv, rand, lim.type, alpha, gamma, info)

Arguments
x matrix with data values
ncomp number of principal components to calculate
center logical, do mean centering or not
scale logical, do standardization or not
method algorithm for computing PC space (only 'svd' and 'nipals' are supported so far)
exclcols columns to be excluded from calculations (numbers, names or vector with logical values)
exclrows rows to be excluded from calculations (numbers, names or vector with logical values)
cv number of segments for random cross-validation (1 for full cross-validation).
rand vector with parameters for randomized PCA methods (if NULL, conventional PCA is used instead)
lim.type which method to use for calculation of critical limits for residuals (see details for pca)
alpha significance level for calculating critical limits for T2 and Q residuals.
gamma significance level for calculating outlier limits for T2 and Q residuals.
info a short text line with model description.

Value
an object with calibrated PCA model
**pca.crossval**

*Cross-validation of a PCA model*

**Description**

Does the cross-validation of a PCA model

**Usage**

```r
generic_function(pca.crossval, model, x, cv, center = T, scale = F)
```

**Arguments**

- `model`: a PCA model (object of class `pca`)
- `x`: a matrix with data values (calibration set)
- `cv`: number of segments (if cv = 1, full cross-validation will be used)
- `center`: logical, do mean centering or not
- `scale`: logical, do standardization or not

**Value**

object of class `pcares` with results of cross-validation

---

**pca.mvreplace**

*Replace missing values in data*

**Description**

`pca.mvreplace` is used to replace missing values in a data matrix with approximated by iterative PCA decomposition.

**Usage**

```r
generic_function(pca.mvreplace, x, center = T, scale = F, maxncomp = 7, expvarlim = 0.95, covlim = 10^-6, maxiter = 100)
```

**Arguments**

- `x`: a matrix with data, containing missing values.
- `center`: logical, do centering of data values or not.
- `scale`: logical, do standardization of data values or not.
- `maxncomp`: maximum number of components in PCA model.
- `expvarlim`: minimum amount of variance, explained by chosen components (used for selection of optimal number of components in PCA models).
- `covlim`: convergence criterion.
- `maxiter`: maximum number of iterations if convergence criterion is not met.
Details

The function uses iterative PCA modeling of the data to approximate and impute missing values. The result is most optimal for data sets with low or moderate level of noise and with number of missing values less than 10% for small dataset and up to 20% for large data.

Value

Returns the same matrix $x$ where missing values are replaced with approximated.

Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

References


Examples

```r
library(mdatools)

## A very simple example of imputing missing values in a data with no noise

# generate a matrix with values
s = 1:6
odata = cbind(s, 2*s, 4*s)

# make a matrix with missing values
mdata = odata
mdata[5, 2] = mdata[2, 3] = NA

# replace missing values with approximated
rdata = pca.mvreplace(mdata, scale = TRUE)

# show all matrices together
show(cbind(odata, mdata, round(rdata, 2)))
```

---

**pca.nipals**

*NIPALS based PCA algorithm*

Description

Calculates principal component space using non-linear iterative partial least squares algorithm (NIPALS)
Usage

\texttt{pca.nipals(x, ncomp)}

Arguments

- \texttt{x} a matrix with data values (preprocessed)
- \texttt{ncomp} number of components to calculate

Value

a list with scores, loadings and eigenvalues for the components

References


---

\texttt{pca.run} \hspace{1cm} \textit{Runs one of the selected PCA methods}

Description

Runs one of the selected PCA methods

Usage

\texttt{pca.run(x, ncomp, method, rand = NULL)}

Arguments

- \texttt{x} data matrix
- \texttt{ncomp} number of components
- \texttt{method} name of PCA methods (’svd’, ’nipals’)
- \texttt{rand} parameters for randomized algorithm (if not NULL)
**pca.svd**

Singular Values Decomposition based PCA algorithm

**Description**
Computes principal component space using Singular Values Decomposition

**Usage**

\[
\text{pca.svd}(x, \ ncomp = \text{NULL})
\]

**Arguments**

\[
\begin{align*}
  x & \quad \text{a matrix with data values (preprocessed)} \\
  n\text{comp} & \quad \text{number of components to calculate}
\end{align*}
\]

**Value**
a list with scores, loadings and eigencalues for the components

---

**pcares**

Results of PCA decomposition

**Description**

pcares is used to store results for PCA decomposition of data.

**Usage**

\[
\text{pcares}(...)\]

**Arguments**

\[
\ldots \quad \text{other arguments supported by ldecomp.}
\]

**Details**

In fact pcares is a wrapper for ldecomp - general class for storing results for linear decomposition \(X = TP' + E\). So, most of the methods, arguments and returned values are inherited from ldecomp.

There is no need to create a pcares object manually, it is created automatically when build a PCA model (see pca) or apply the model to a new data (see predict.pca). The object can be used to show summary and plots for the results.
**Value**

Returns an object (list) of class `pcares` and `ldecomp` with following fields:

- **scores**
  matrix with score values (nobj x ncomp).
- **residuals**
  matrix with data residuals (nobj x nvar).
- **T2**
  matrix with T2 distances (nobj x ncomp).
- **Q**
  matrix with Q residuals (nobj x ncomp).
- **tnorm**
  vector with singular values used for scores normalization.
- **ncomp.selected**
  selected number of components.
- **expvar**
  explained variance for each component.
- **cumexpvar**
  cumulative explained variance.

**See Also**

Methods for `pcares` objects:

- `print.pcares` shows information about the object.
- `summary.pcares` shows statistics for the PCA results.

Methods, inherited from `ldecomp` class:

- `plotScores.ldecomp` makes scores plot.
- `plotVariance.ldecomp` makes explained variance plot.
- `plotCumVariance.ldecomp` makes cumulative explained variance plot.
- `plotResiduals.ldecomp` makes Q vs. T2 residuals plot.

Check also `pca` and `ldecomp`.

**Examples**

```r
### Examples for PCA results class

library(mdatools)

## 1. Make a model for every odd row of People data
## and apply it to the objects from every even row

data(people)
x = people[seq(1, 32, 2), ]
x.new = people[seq(1, 32, 2), ]

model = pca(people, scale = TRUE, cv = 1, info = 'Simple PCA model')
model = selectCompNum(model, 4)
```
res = predict(model, x.new)
summary(res)
plot(res)

## 1. Make PCA model for People data with autoscaling
## and full cross-validation and get calibration results

data(people)
model = pca(people, scale = TRUE, cv = 1, info = 'Simple PCA model')
model = selectCompNum(model, 4)

res = model$calres
summary(res)
plot(res)

## 2. Show scores plots for the results
par(mfrow = c(2, 2))
plotScores(res)
plotScores(res, cgroup = people[, 'Beer'], show.labels = TRUE)
plotScores(res, comp = c(1, 3), show.labels = TRUE)
plotScores(res, comp = 2, type = 'h', show.labels = TRUE)
par(mfrow = c(1, 1))

## 3. Show residuals and variance plots for the results
par(mfrow = c(2, 2))
plotResiduals(res)
plotCumVariance(res, legend.position = 'bottomright')
plotResiduals(res, cgroup = people[, 'Sex'])
plotResiduals(res, ncomp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

pellets

Image data

Description

Dataset for showing how mdatools works with images. It is an RGB image represented as 3-way array.

Usage

data(people)

Format

a 3-way array (height x width x channels).
Details

This is an image with pellets of four different colours mixed in a glas volume.

---

**People data**

Description

Dataset for exploratory analysis with 32 objects (male and female persons) and 12 variables.

Usage

data(people)

Format

a matrix with 32 observations (persons) and 12 variables.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Height in cm.</td>
</tr>
<tr>
<td>2</td>
<td>Weight in kg.</td>
</tr>
<tr>
<td>3</td>
<td>Hair length (-1 for short, +1 for long).</td>
</tr>
<tr>
<td>4</td>
<td>Shoe size (EU standard).</td>
</tr>
<tr>
<td>5</td>
<td>Age, years.</td>
</tr>
<tr>
<td>6</td>
<td>Income, euro per year.</td>
</tr>
<tr>
<td>7</td>
<td>Beer consumption, liters per year.</td>
</tr>
<tr>
<td>8</td>
<td>Wine consumption, liters per year.</td>
</tr>
<tr>
<td>9</td>
<td>Sex (-1 for male, +1 for female).</td>
</tr>
<tr>
<td>10</td>
<td>Swimming ability (index, based on 500 m swimming time).</td>
</tr>
<tr>
<td>11</td>
<td>Region (-1 for Scandinavia, +1 for Mediterranean.</td>
</tr>
<tr>
<td>12</td>
<td>IQ (European standardized test).</td>
</tr>
</tbody>
</table>

Details

The data was taken from the book [1] and is in fact a small subset of a pan-European demographic survey. It includes information about 32 persons, 16 represent northern Europe (Scandinavians) and 16 are from the Mediterranean regions. In both groups there are 8 male and 8 female persons. The data includes both quantitative and qualitative variables and is particularly useful for benchmarking exploratory data analysis methods.

Source

**pinv**

*Pseudo-inverse matrix*

**Description**

Computes pseudo-inverse matrix using SVD

**Usage**

```
pinv(data)
```

**Arguments**

- `data`: a matrix with data values to compute inverse for

**plot.classres**

*Plot function for classification results*

**Description**

Generic plot function for classification results. Shows predicted class values.

**Usage**

```
## S3 method for class 'classres'
plot(x, nc = NULL, ...)
```

**Arguments**

- `x`: classification results (object of class `plsdares`, `simcamres`, etc.).
- `nc`: if there are several classes, which class to make the plot for (NULL - summary for all classes).
- `...`: other arguments
plot.ipls  

*Overview plot for iPLS results*

**Description**

Shows a plot for iPLS results.

**Usage**

```r
## S3 method for class 'ipls'
plot(x, ...)
```

**Arguments**

- `x` a (object of class `pca`)
- `...` other arguments

**Details**

See details for `plotSelection.ipls`.

---

plot.pca  

*Model overview plot for PCA*

**Description**

Shows a set of plots (scores, loadings, residuals and explained variance) for PCA model.

**Usage**

```r
## S3 method for class 'pca'
plot(x, comp = c(1, 2), show.labels = FALSE, show.legend = TRUE, ...)
```

**Arguments**

- `x` a PCA model (object of class `pca`)
- `comp` vector with two values - number of components to show the scores and loadings plots for
- `show.labels` logical, show or not labels for the plot objects
- `show.legend` logical, show or not a legend on the plot
- `...` other arguments

**Details**

See examples in help for `pca` function.
**plot.pcares**  
*Plot method for PCA results object*

**Description**
Show several plots to give an overview about the PCA results

**Usage**
```r
## S3 method for class 'pcares'
plot(x, comp = c(1, 2), show.labels = T, ...)
```

**Arguments**
- `x` PCA results (object of class `pcares`)
- `comp` which components to show the scores plot for (can be one value or vector with two values).
- `show.labels` logical, show or not labels for the plot objects
- `...` other arguments

**plot.pls**  
*Model overview plot for PLS*

**Description**
Shows a set of plots (x residuals, regression coefficients, RMSE and predictions) for PLS model.

**Usage**
```r
## S3 method for class 'pls'
plot(x, ncomp = NULL, ny = 1, show.legend = T,
     show.labels = F, ...)
```

**Arguments**
- `x` a PLS model (object of class `pls`)
- `ncomp` how many components to use (if NULL - user selected optimal value will be used)
- `ny` which y variable to show the summary for (if NULL, will be shown for all)
- `show.legend` logical, show or not a legend on the plot
- `show.labels` logical, show or not labels for the plot objects
- `...` other arguments

**Details**
See examples in help for `pls` function.
plot.plsda

Model overview plot for PLS-DA

Description

Shows a set of plots (x residuals, regression coefficients, misclassification ratio and predictions) for PLS-DA model.

Usage

```r
## S3 method for class 'plsda'
plot(x, ncomp = NULL, nc = 1, show.legend = T,
     show.labels = F, ...)
```

Arguments

- `x`: a PLS-DA model (object of class `plsda`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `nc`: which class to show the plots
- `show.legend`: logical, show or not a legend on the plot
- `show.labels`: logical, show or not labels for the plot objects
- `...`: other arguments

Details

See examples in help for `plsda` function.

plot.plsdares

Overview plot for PLS-DA results

Description

Shows a set of plots (x residuals, y variance, classification performance and predictions) for PLS-DA results.

Usage

```r
## S3 method for class 'plsdares'
plot(x, nc = NULL, ncomp = NULL, show.labels = F,
     show.line = T, ...)
```
plot.plsres

Overview plot for PLS results

Description

Shows a set of plots for PLS results.

Usage

```r
## S3 method for class 'plsres'
plot(x, ncomp = NULL, ny = 1, show.labels = F, ...)
```

Arguments

- `x`: PLS results (object of class `plsres`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `ny`: which y variable to show the summary for (if NULL, will be shown for all)
- `show.labels`: logical, show or not labels for the plot objects
- `...`: other arguments

Details

See examples in help for `plsres` function.
plot.randtest  

Plot for randomization test results

Description

Makes a bar plot with alpha values for each component.

Usage

```r
## S3 method for class 'randtest'
plot(x, main = "Alpha", xlab = "Components", ylab = "", ...)  
```

Arguments

- `x`: results of randomization test (object of class 'randtest')
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `...`: other optional arguments

Details

See examples in help for `randtest` function.

plot.regcoeffs  

Regression coefficients plot

Description

Shows plot with regression coefficient values for every predictor variable (x)

Usage

```r
## S3 method for class 'regcoeffs'
plot(x, ncomp = 1, ny = 1, type = NULL, col = NULL, main = NULL, ylab = NULL, show.line = T, show.ci = T, alpha = 0.05, ...)  
```
plot.regres

**Arguments**

- `x` regression coefficients object (class `regcoeffs`)
- `ncomp` number of components to return the coefficients for
- `ny` number of response variable to return the coefficients for
- `type` type of the plot
- `col` vector with colors for the plot (vector or one value)
- `main` main plot title
- `ylab` label for y axis
- `show.line` logical, show or not line for 0 value
- `show.ci` logical, show or not confidence intervals if they are available
- `alpha` significance level for confidence intervals (a number between 0 and 1, e.g. for 95% alpha = 0.05)
- `...` other arguments

**Description**

plot method for regression results

**Usage**

```r
## S3 method for class 'regres'
plot(x, ny = 1, ...)
```

**Arguments**

- `x` regression results (object of class `regres`)
- `ny` which response to show the plot for (if y is multivariate)
- `...` other plot parameters (see `mdaplot` for details)

**Details**

Shows prediction plot for the results (the same as `plotPredictions.regres`)
**plot.simca**

*Model overview plot for SIMCA*

---

**Description**

Shows a set of plots for SIMCA model.

**Usage**

```r
## S3 method for class 'simca'
plot(x, ncomp = NULL, ...)
```

**Arguments**

- `x`: a SIMCA model (object of class `simca`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `...`: other arguments

**Details**

See examples in help for `simcam` function.

---

**plot.simcam**

*Model overview plot for SIMCAM*

---

**Description**

Shows a set of plots for SIMCAM model.

**Usage**

```r
## S3 method for class 'simcam'
plot(x, nc = c(1, 2), ...)
```

**Arguments**

- `x`: a SIMCAM model (object of class `simcam`)
- `nc`: vector with two values - classes (SIMCA models) to show the plot for
- `...`: other arguments

**Details**

See examples in help for `simcam` function.
plotBiplot

Description

Biplot

Usage

plotBiplot(obj, ...)

Arguments

obj a model or result object

... other arguments

Details

Generic function for biplot
plotBiplot.pca

PCA biplot

Description

Shows a biplot for selected components.

Usage

```r
## S3 method for class 'pca'
plotBiplot(obj, comp = c(1, 2), pch = c(16, NA),
            col = mdaplot.getColors(2), main = "Biplot", lty = 1, lwd = 1,
            show.labels = FALSE, show.axes = TRUE, show.excluded = FALSE,
            lab.col = c("#90A0D0", "#D09000"), ...)
```

Arguments

- `obj` a PCA model (object of class `pca`)
- `comp` a value or vector with several values - number of components to show the plot for
- `pch` a vector with two values - markers for scores and loadings
- `col` a vector with two colors for scores and loadings
- `main` main title for the plot
- `lty` line type for loadings
- `lwd` line width for loadings
- `show.labels` logical, show or not labels for the plot objects
- `show.axes` logical, show or not a axes lines crossing origin (0,0)
- `show.excluded` logical, show or hide rows marked as excluded (attribute 'exclrows')
- `lab.col` a vector with two colors for scores and loadings labels
- `...` other plot parameters (see `mdaplotg` for details)

plotCooman

Cooman's plot

Description

Cooman's plot

Usage

```r
plotCooman(obj, ...)
```
Arguments

- **obj**: classification model or result object
- **...**: other arguments

Details

Generic function for Cooman’s plot

```
plotCooman.simcam
```

Description

Shows a Cooman’s plot for a pair of SIMCA models

Usage

```r
## S3 method for class 'simcam'
plotCooman(obj, nc = c(1, 2), ...)
```

Arguments

- **obj**: a SIMCAM model (object of class `simcam`)
- **nc**: vector with two values - classes (SIMCA models) to show the plot for
- **...**: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `simcam` function.

```
plotCooman.simcamres
```

Description

Shows a Cooman’s plot for a pair of SIMCA models

Usage

```r
## S3 method for class 'simcamres'
plotCooman(obj, nc = c(1, 2), type = "p",
    main = "Cooman's plot", xlab = NULL, ylab = NULL,
    show.limits = T, legend = NULL, ...)
```
plotCorr

Arguments

- `obj`: SIMCAM results (object of class `simcamres`)
- `nc`: vector with two values - classes (SIMCA models) to show the plot for
- `type`: type of the plot
- `main`: main plot title
- `xlab`: label for x axis
- `ylab`: label for y axis
- `show.limits`: logical, show or not lines with statistical limits for the residuals
- `legend`: vector with legend items
- `...`: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `simcamres` function.

---

**plotCorr**  
*Correlation plot*

**Description**

Correlation plot

**Usage**

`plotCorr(obj, ...)`

**Arguments**

- `obj`: a model or result object
- `...`: other arguments

**Details**

Generic function for correlation plot
plotCorr.randtest  
*Correlation plot for randomization test results*

**Description**

Makes a plot with statistic values vs. coefficient of determination between permuted and reference y-values.

**Usage**

```r
## S3 method for class 'randtest'
plotCorr(obj, comp = NULL, main = NULL,
         xlab = expression(r^2), ylab = "Test statistic", ylim = NULL, ...)
```

**Arguments**

- `obj`: results of randomization test (object of class 'randtest')
- `comp`: number of component to make the plot for
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `ylim`: limits for y axis
- `...`: other optional arguments

**Details**

See examples in help for `randtest` function.

---

plotCumVariance  
*Variance plot*

**Description**

Variance plot

**Usage**

```r
plotCumVariance(obj, ...)
```

**Arguments**

- `obj`: a model or result object
- `...`: other arguments
plotCumVariance.ldecomp

Cumulative explained variance plot for linear decomposition

Description
Shows a plot with cumulative explained variance values vs. number of components.

Usage
```
## S3 method for class 'ldecomp'
plotCumVariance(obj, type = "b",
    main = "Cumulative variance", xlab = "Components",
    ylab = "Explained variance, %", show.labels = F,
    labels = "values", ...)
```

Arguments
- `obj`: object of ldecomp class.
- `type`: type of the plot
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `show.labels`: logical, show or not labels for the plot objects
- `labels`: what to show as labels for plot objects
- `...`: most of graphical parameters from mdaplot function can be used.

plotCumVariance.pca

Cumulative explained variance plot for PCA

Description
Shows a plot with cumulative explained variance for components.

Usage
```
## S3 method for class 'pca'
plotCumVariance(obj, xlab = "Components",
    ylab = "Explained variance, %", main = "Cumulative variance", ...)
```
plotDiscriminationPower

Arguments

- **obj**: a PCA model (object of class `pca`)
- **xlab**: label for x axis
- **ylab**: label for y axis
- **main**: main title for the plot
- **...**: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `pca` function.

Description

Discrimination power plot

Usage

```r
plotDiscriminationPower(obj, ...)
```

Arguments

- **obj**: a model object
- **...**: other arguments

Details

Generic function for plotting discrimination power values for classification model
plotDiscriminationPower.simcam

Discrimination power plot for SIMCAM model

Description

Shows a plot with discrimination power of predictors for a pair of SIMCA models

Usage

## S3 method for class 'simcam'
plotDiscriminationPower(obj, nc = c(1, 2), type = "h",
    main = NULL, xlab = "Variables", ylab = "", ...)

Arguments

- `obj`: a SIMCAM model (object of class `simcam`)
- `nc`: vector with two values - classes (SIMCA models) to show the plot for
- `type`: type of the plot
- `main`: main plot title
- `xlab`: label for x axis
- `ylab`: label for y axis
- ... other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `simcam` function.

plotExtreme

Shows extreme plot for SIMCA model

Description

Generic function for creating extreme plot for SIMCA model

Usage

plotExtreme(obj, ...)

Arguments

- `obj`: a SIMCA model
- ... other parameters
plotExtreme.simca  
*Shows extreme plot for SIMCA model*

**Description**

The plot shows the number of extreme objects rejected by the model vs. the expected number, which depends on significance level and total number of objects. The light blue area shows 95 tolerance limits. See more details in [1].

**Usage**

```r
## S3 method for class 'simca'
plotExtreme(obj, ncomp = NULL, main = NULL,
             xlab = "Expected", ylab = "Observed", ...)
```

**Arguments**

- `obj`: SIMCA model
- `ncomp`: Number of components to show the plot for
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `...`: other arguments

**References**


plotHist  
*Statistic histogram*

**Description**

Statistic histogram

**Usage**

```r
plotHist(obj, ...)
```

**Arguments**

- `obj`: a model or result object
- `...`: other arguments
plotHist.randtest

Details

Generic function for plotting statistic histogram plot

plotHist.randtest  Histogram plot for randomization test results

Description

Makes a histogram for statistic values distribution for particular component, also show critical value as a vertical line.

Usage

## S3 method for class 'randtest'
plotHist(obj, comp = NULL, main = NULL,
         xlab = "Test statistic", ylab = "Frequency", ...)

Arguments

- **obj**: results of randomization test (object of class 'randtest')
- **comp**: number of component to make the plot for
- **main**: main title for the plot
- **xlab**: label for x axis
- **ylab**: label for y axis
- **...**: other optional arguments

Details

See examples in help for randtest function.

plotLoadings

Description

Loadings plot

Usage

plotLoadings(obj, ...)

Arguments

- **obj**: a model or result object
- **...**: other arguments
plotLoadings.pca

Details

Generic function for plotting loadings values for data decomposition

---

plotLoadings.pca  Loadings plot for PCA

Description

Shows a loadings plot for selected components.

Usage

```r
## S3 method for class 'pca'
plotLoadings(obj, comp = c(1, 2), type = NULL,
              main = "Loadings", xlab = NULL, ylab = NULL, show.labels = NULL,
              show.legend = TRUE, show.axes = TRUE, ...)
```

Arguments

- `obj`: a PCA model (object of class `pca`)
- `comp`: a value or vector with several values - number of components to show the plot for
- `type`: type of the plot ("b", "l", "h")
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `show.labels`: logical, show or not labels for the plot objects
- `show.legend`: logical, show or not a legend on the plot
- `show.axes`: logical, show or not a axes lines crossing origin (0,0)
- `...`: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `pca` function.
plotMisclassified

Description

Misclassification ratio plot

Usage

plotMisclassified(obj, ...)

Arguments

- obj: a model or a result object
- ...: other arguments

Details

Generic function for plotting missclassification values for classification model or results

plotMisclassified.classmodel

Description

Makes a plot with misclassified ratio values vs. model complexity (e.g. number of components)

Usage

## S3 method for class 'classmodel'
plotMisclassified(obj, nc = NULL, ...)

Arguments

- obj: classification model (object of class plsda, simca, etc.).
- nc: if there are several classes, which class to make the plot for (NULL - summary for all classes).
- ...: most of the graphical parameters from mdaplotg function can be used.

Details

See examples in description of plsda, simca or simcam.
**plotMisclassified.classes**

*Misclassified ratio plot for classification results*

**Description**

Makes a plot with misclassified ratio values vs. model complexity (e.g. number of components) for classification results.

**Usage**

```r
## S3 method for class 'classres'
plotMisclassified(obj, nc = NULL, ...)
```

**Arguments**

- `obj` classification results (object of class `plsdares`, `simcamres`, etc.).
- `nc` if there are several classes, which class to make the plot for (NULL - summary for all classes).
- `...` most of the graphical parameters from `mdaplot` function can be used.

**Details**

See examples in description of `plsdares`, `simcamres`, etc.

---

**plotModelDistance**

*Model distance plot*

**Description**

Model distance plot

**Usage**

```r
plotModelDistance(obj, ...)```

**Arguments**

- `obj` a model object
- `...` other arguments

**Details**

Generic function for plotting distance from object to a multivariate model
plotModelDistance.simcam

Modelling distance plot for SIMCAM model

Description

Shows a plot with distance from data objects to a SIMCA model

Usage

```r
## S3 method for class 'simcam'
plotModelDistance(obj, nc = 1, type = "h",
                  main = NULL, xlab = "Models", ylab = ",", ...)
```

Arguments

- `obj`: a SIMCAM model (object of class `simcam`)
- `nc`: for which class (SIMCA model) to show the plot for
- `type`: type of the plot
- `main`: main plot title
- `xlab`: label for x axis
- `ylab`: label for y axis
- `...`: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `simcam` function.

plotModellingPower

Modelling power plot

Description

Modelling power plot

Usage

```r
plotModellingPower(obj, ...)
```

Arguments

- `obj`: a model object
- `...`: other arguments
Details

Generic function for plotting modelling power values for classification model

plotModellingPower.simca

Modelling power plot for SIMCA model

Description

Shows a plot with modelling power values for each predictor

Usage

## S3 method for class 'simca'
plotModellingPower(obj, ncomp = NULL, type = "h", main = NULL, ylab = "", ...)

Arguments

- obj: a SIMCA model (object of class simca)
- ncomp: number of components to show the values for
- type: type of the plot
- main: main plot title
- ylab: label for y axis
- ...: other plot parameters (see mdaplotg for details)

plotModellingPower.simcam

Modelling power plot for SIMCAM model

Description

Shows a plot with modelling power values for each predictor of selected SIMCA model

Usage

## S3 method for class 'simcam'
plotModellingPower(obj, nc = 1, main = NULL, ...)

Arguments

- obj: a SIMCAM model (object of class simcam)
- nc: which classe (SIMCA model) to show the plot for
- main: main plot title
- ...: other plot parameters (see mdaplotg for details)
plotPerformance

Details
See examples in help for simcam function.

plotPerformance

Classification performance plot

Description
Classification performance plot

Usage
plotPerformance(obj, ...)

Arguments
obj a model or result object
... other arguments

Details
Generic function for plotting classification performance for model or results

plotPerformance.classmodel

Performance plot for classification model

Description
Makes a plot with sensitivity values vs. model complexity (e.g. number of components)

Usage
## S3 method for class 'classmodel'
plotPerformance(obj, nc = NULL,
    param = "specificity", type = "h", main = NULL,
    xlab = "Components", ylab = "", ylim = c(0, 1.15), ...)


Arguments

- **obj**: classification model (object of class plsda, simca, etc.).
- **nc**: if there are several classes, which class to make the plot for (NULL - summary for all classes).
- **param**: which parameter to make the plot for ("specificity", "sensitivity", or "misclassified")
- **type**: type of the plot
- **main**: main title for the plot
- **xlab**: label for x axis
- **ylab**: label for y axis
- **ylim**: vector with two values - limits for y axis
- **...**: most of the graphical parameters from mdaplot function can be used.

**plotPerformance.cclassres**

*Performance plot for classification results*

**Description**

Makes a plot with classification performance parameters vs. model complexity (e.g. number of components) for classification results.

**Usage**

```r
## S3 method for class 'classres'
plotPerformance(obj, nc = NULL, param = "all",
    type = "h", legend = NULL, main = NULL, xlab = "Components",
    ylab = "", ylim = c(0, 1.1), ...)"'
```

**Arguments**

- **obj**: classification results (object of class plsdares, simcamres, etc.).
- **nc**: if there are several classes, which class to make the plot for (NULL - summary for all classes).
- **param**: which performance parameter to make the plot for ("sensitivity", "specificity", "misclassified", "all").
- **type**: type of the plot
- **legend**: vector with legend items
- **main**: main title for the plot
- **xlab**: label for x axis
- **ylab**: label for y axis
- **ylim**: vector with two values - limits for y axis
- **...**: most of the graphical parameters from mdaplot function can be used.
plotPredictions

Details

See examples in description of `plsdares, simcamres`, etc.

---

plotPredictions  Predictions plot

---

Description

Predictions plot

Usage

plotPredictions(obj, ...)

Arguments

- `obj`: a model or result object
- `...`: other arguments

Details

Generic function for plotting predicted values for classification or regression model or results

---

plotPredictions.classmodel

Predictions plot for classification model

---

Description

Makes a plot with class predictions for a classification model.

Usage

```r
## S3 method for class 'classmodel'
plotPredictions(obj, res = NULL, nc = NULL,
                ncomp = NULL, main = NULL, ...)
```
plotPredictions.classres

Arguments

- **obj**: a classification model (object of class simca, plsda, etc.). If NULL value is specified, the result will be selected automatically by checking the nearest available from test, cv and calibration results.
- **res**: which result to make the plot for ('calres', 'cvres' or 'testres').
- **nc**: if there are several classes, which class to make the plot for (NULL - for all).
- **ncomp**: what number of components to make the plot for (NULL - for selected in the model).
- **main**: main title for the plot
- **...**: most of the graphical parameters from mdaPlot function can be used.

Details

See examples in description of plsda, simca or simcam.

plotPredictions.classres

Prediction plot for classification results

Description

Makes a plot with predicted class values for classification results.

Usage

```r
## S3 method for class 'classres'
plotPredictions(obj, nc = NULL, ncomp = NULL,
    type = "p", main = NULL, ylab = "", ...)```

Arguments

- **obj**: classification results (object of class plsdares, simcamres, etc.).
- **nc**: if there are several classes, which class to make the plot for (NULL - summary for all classes).
- **ncomp**: which number of components to make the plot for (one value, if NULL - model selected number will be used). This parameter shall not be used for multiclass models or results as predictions in this case are only for optimal number of components
- **type**: type of the plot
- **main**: main title for the plot
- **ylab**: label for y axis
- **...**: most of the graphical parameters from mdaPlot function can be used.
plotPredictions.pls

Details

See examples in description of `plsdare`, `simcamres`, etc.

---

plotPredictions.pls  Predictions plot for PLS

Description

Shows plot with predicted vs. reference (measured) y values for selected components.

Usage

```r
## S3 method for class 'pls'
plotPredictions(obj, ncomp = NULL, ny = 1, main = NULL,
    legend.position = "topleft", show.line = T, colmap = "default",
    col = NULL, ...)
```

Arguments

- **obj**: a PLS model (object of class `pls`)
- **ncomp**: how many components to use (if NULL - user selected optimal value will be used)
- **ny**: number of response variable to make the plot for (if y is multivariate)
- **main**: main plot title
- **legend.position**: position of legend on the plot (if shown)
- **show.line**: logical, show or not line fit for the plot points
- **colmap**: a colormap to use for coloring the plot items
- **col**: a vector with color values for target lines fitted the points
- **...**: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `pls` function.
plotPredictions.plsres

Predictions plot for PLS results

Description

Shows plot with predicted vs. reference (measured) y values for selected components.

Usage

```r
## S3 method for class 'plsres'
plotPredictions(obj, ny = 1, ncomp = NULL,
    main = NULL, ...)
```

Arguments

- `obj`: PLS results (object of class `plsres`)
- `ny`: number of response variable to make the plot for (if y is multivariate)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `main`: main plot title
- `...`: other plot parameters (see `mdaplot` for details)

Details

See examples in help for `plsres` function.

See Also

- `plotPredictions.regres` - prediction plot for regression results.

plotPredictions.regres

Predictions plot for regression results

Description

Shows plot with predicted y values.

Usage

```r
## S3 method for class 'regres'
plotPredictions(obj, ny = 1, ncomp = NULL,
    show.line = T, show.stat = F, stat.col = "#606060",
    stat.cex = 0.85, axes.equal = T, col = mdaplot.getColors(1), ...)
```
plotProbabilities

Arguments

- **obj**: regression results (object of class *regres*)
- **ny**: number of predictor to show the plot for (if `y` is multivariate)
- **ncomp**: complexity of model (e.g. number of components) to show the plot for
- **show.line**: logical, show or not line fit for the plot points
- **show.stat**: logical, show or not legend with statistics on the plot
- **stat.col**: color of text in legend with statistics
- **stat.cex**: size of text in legend with statistics
- **axes.equal**: logical, make limits for x and y axes equal or not
- **col**: color for the plot objects.
- **...**: other plot parameters (see `mdatplot` for details)

Details

If reference values are available, the function shows a scatter plot with predicted vs. reference values, otherwise predicted values are shown vs. object numbers.

---

**plotProbabilities**  
*Plot for class belonging probability*

---

Description

Makes a plot with class belonging probabilities for each object of the classification results. Works only with classification methods, which compute this probability (e.g. SIMCA).

Usage

`plotProbabilities(obj, ...)`

Arguments

- **obj**: an object with classification results (e.g. SIMCA)
- **...**: other parameters
plotProbabilities.classres

Plot for class belonging probability

Description

Makes a plot with class belonging probabilities for each object of the classification results. Works only with classification methods, which compute this probability (e.g. SIMCA).

Usage

```r
## S3 method for class 'classres'
plotProbabilities(obj, ncomp = obj$ncomp.selected,
    nc = 1, type = "h", xlab = "Objects", ylab = "Probability",
    main = NULL, ylim = c(0, 1.1), show.lines = c(NA, 0.5), ...)
```

Arguments

- `obj`: classification results (e.g. object of class `simcamres`).
- `ncomp`: number of components to use the probabilities for.
- `nc`: if there are several classes, which class to make the plot for.
- `type`: type of the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `main`: main plot title
- `ylim`: vector with limits for y-axis
- `show.lines`: shows a horizontal line at p = 0.5
- `...`: most of the graphical parameters from `mdaplot` function can be used.

plotRegcoeffs

Regression coefficients plot

Description

Regression coefficients plot

Usage

```r
plotRegcoeffs(obj, ...)
```

Arguments

- `obj`: a model or result object
- `...`: other arguments
Details

Generic function for plotting regression coefficients values for a regression model

plotRegcoeffs.pls  Regression coefficient plot for PLS

Description

Shows plot with regression coefficient values for selected components.

Usage

```r
## S3 method for class 'pls'
plotRegcoeffs(obj, ncomp = NULL, ...)
```

Arguments

- `obj` a PLS model (object of class `pls`)
- `ncomp` how many components to use (if NULL - user selected optimal value will be used)
- `...` other plot parameters (see `mdaplotg` and `plotRegcoeffs` for details)

Details

See examples in help for `pls` function.

plotResiduals  Residuals plot

Description

Residuals plot

Usage

```r
plotResiduals(obj, ...)
```

Arguments

- `obj` a model or result object
- `...` other arguments

Details

Generic function for plotting residual values for data decomposition
plotResiduals.ldcomp  Residuals plot for linear decomposition

Description

Shows a plot with T2 vs Q values for data objects.

Usage

```r
## S3 method for class 'ldcomp'
plotResiduals(obj, ncomp = NULL, main = NULL,
  xlab = NULL, ylab = NULL, show.labels = F, ...)
```

Arguments

- **obj**: object of ldcomp class.
- **ncomp**: what number of components to show the plot for (if NULL, model selected value will be used).
- **main**: main title for the plot
- **xlab**: label for x axis
- **ylab**: label for y axis
- **show.labels**: logical, show or not labels for the plot objects
- **...**: most of graphical parameters from mdaplot function can be used.

plotResiduals.pca  Residuals plot for PCA

Description

Shows a plot with Q residuals vs. Hotelling T2 values for selected number of components.

Usage

```r
## S3 method for class 'pca'
plotResiduals(obj, ncomp = NULL, norm = F, main = NULL,
  xlab = NULL, ylab = NULL, show.labels = F, show.legend = T,
  show.limits = T, xlim = NULL, ylim = NULL, lim.col = c("#333333",
  
  
  "#333333"), lim.lwd = c(1, 1), lim.lty = c(2, 3), ...)
```
**plotResiduals.pcares**

### Arguments

- `obj`: a PCA model (object of class `pca`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `norm`: logical, show normalized Q vs T2 (norm = T) values or original ones (norm = F)
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `show.labels`: logical, show or not labels for the plot objects
- `show.legend`: logical, show or not a legend on the plot
- `show.limits`: logical, show or not lines with statistical limits for the residuals
- `xlim`: limits for x-axis
- `ylim`: limits for y-axis
- `lim.col`: vector with two values - line color for extreme and outlier borders
- `lim.lwd`: vector with two values - line width for extreme and outlier borders
- `lim.lty`: vector with two values - line type for extreme and outlier borders
- `...`: other plot parameters (see `mdaplotg` for details)

### Details

See examples in help for `pca` function.

### Description

Shows a plot with T2 vs Q values for data objects.

### Usage

```r
## S3 method for class 'pcares'
plotResiduals(obj, ncomp = NULL, main = NULL,
    xlab = NULL, ylab = NULL, show.labels = F, show.limits = T,
    norm = F, xlim = NULL, ylim = NULL, lim.col = c("#333333",
    "#333333"), lim.lwd = c(1, 1), lim.lty = c(2, 3), ...)
```
Arguments

- `obj`: object of `ldecomp` class.
- `ncomp`: what number of components to show the plot for (if NULL, model selected value will be used).
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `show.labels`: logical, show or not labels for the plot objects
- `show.limits`: logical, show or not lines for statistical limits of the residuals
- `norm`: logical, show normalized Q vs T2 (norm = T) values or original ones (norm = F)
- `xlim`: limits for x-axis
- `ylim`: limits for y-axis
- `lim.col`: vector with two values - line color for extreme and outlier borders
- `lim.lwd`: vector with two values - line width for extreme and outlier borders
- `lim.lty`: vector with two values - line type for extreme and outlier borders
- `...`: most of graphical parameters from `mdaplot` function can be used.

---

**plotResiduals.simcam**  
*Residuals plot for SIMCAM model*

Description

Shows a plot with residuals for SIMCAM model

Usage

```r
## S3 method for class 'simcam'
plotResiduals(obj, ...)
```

Arguments

- `obj`: a SIMCAM model (object of class `simcam`)
- `...`: other plot parameters (see `mdaplot` for details)

Details

See examples in help for `simcam` function.
plotResiduals.simcamres

Residuals plot for SIMCAM results

Description

Shows a plot with Q vs. T2 residuals for SIMCAM results

Usage

```r
## S3 method for class 'simcamres'
plotResiduals(obj, nc = 1, main = NULL, ...)
```

Arguments

- `obj`: SIMCAM results (object of class `simcamres`)
- `nc`: which class (SIMCA model) to show the plot for
- `main`: main plot title
- `...`: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `simcamres` function.

plotResiduals.simcares

Residuals plot for SIMCA results

Description

Shows a plot with Q vs. T2 residuals for SIMCA results

Usage

```r
## S3 method for class 'simcares'
plotResiduals(obj, ncomp = NULL, main = NULL,
              xlab = NULL, ylab = NULL, norm = F, show.limits = T,
              legend = NULL, lim.col = c("#c00a0a", "#906060"), lim.lwd = c(1, 1),
              lim.lty = c(2, 3), ...)
```
Arguments

- **obj**: SIMCA results (object of class `simcares`)
- **ncomp**: which principal components to show the plot for
- **main**: main plot title
- **xlab**: label for x axis
- **ylab**: label for y axis
- **norm**: logical, show normalized Q vs T2 (norm = T) values or original ones (norm = F)
- **show.limits**: logical, show or not lines with statistical limits for the residuals
- **legend**: vector with legend items
- **lim.col**: vector with two values - line color for extreme and outlier borders
- **lim.lwd**: vector with two values - line width for extreme and outlier borders
- **lim.lty**: vector with two values - line type for extreme and outlier borders
- **...**: other plot parameters (see `mdaplot` for details)

Details

See examples in help for `simcares` function.

```
plotRMSE
```

RMSE plot

Description

RMSE plot

Usage

```
plotRMSE(obj, ...)
```

Arguments

- **obj**: a model or result object
- **...**: other arguments

Details

Generic function for plotting RMSE values vs. complexity of a regression model
 RMSE development plot

### Description

Shows how RMSE develops for each iteration of iPLS selection algorithm

### Usage

```r
## S3 method for class 'ipls'
plotRMSE(obj, glob.ncomp = NULL,
         main = "RMSE development", xlab = "Iterations", ylab = "RMSEcv",
         xlim = NULL, ylim = NULL, ...
)
```

### Arguments

- `obj`: iPLS results (object of class ipls)
- `glob.ncomp`: number of components for global PLS model with all intervals
- `main`: main title for the plot
- `xlab`: label for x-axis
- `ylab`: label for y-axis
- `xlim`: limits for x-axis
- `ylim`: limits for y-axis
- `...`: other arguments

### Details

The plot shows RMSE values obtained at each iteration of the iPLS algorithm as bars. The first bar correspond to the global model with all variables included, second - to the model obtained at the first iteration and so on. Number at the bottom of each bar corresponds to the interval included or excluded at the particular iteration.

### See Also

`summary.ipls`, `plotSelection.ipls`
### plotRMSE.pls

**RMSE plot for PLS**

**Description**

Shows plot with root mean squared error values vs. number of components for PLS model.

**Usage**

```r
## S3 method for class 'pls'
plotRMSE(obj, ny = 1, type = "b", main = "RMSE",
          xlab = "Components", ylab = NULL, labels = "values", ...)
```

**Arguments**

- `obj`: a PLS model (object of class `pls`)
- `ny`: number of response variable to make the plot for (if `y` is multivariate)
- `type`: type of the plot ("b", "l" or "h")
- `main`: main plot title
- `xlab`: label for x axis
- `ylab`: label for y axis
- `labels`: what to show as labels (if this option is on)
- `...`: other plot parameters (see `mdaplotg` for details)

**Details**

See examples in help for `pls` function.

### plotRMSE.regres

**RMSE plot for regression results**

**Description**

Shows plot with RMSE values vs. model complexity (e.g. number of components).

**Usage**

```r
## S3 method for class 'regres'
plotRMSE(obj, ny = 1, type = "b", labels = "values",
          ...)
```

**Arguments**

- `obj`: an object of class `regres`
- `ny`: number of response variable to make the plot for (if `y` is multivariate)
- `type`: type of the plot ("b", "l" or "h")
- `labels`: what to show as labels (if this option is on)
- `...`: other plot parameters (see `mdaplotg` for details)
**Arguments**

- `obj`: regression results (object of class `regres`)
- `ny`: number of predictor to show the plot for (if `y` is multivariate)
- `type`: type of the plot
- `labels`: what to show as labels for plot objects.
- `...`: other plot parameters (see `mdaplot` for details)

---

**Description**

Scores plot

**Usage**

```r
plotScores(obj, ...)  
```

**Arguments**

- `obj`: a model or result object
- `...`: other arguments

**Details**

Generic function for scores values for data decomposition

---

**plotScores.ldecomp**  
**Scores plot for linear decomposition**

**Description**

Shows a plot with scores values for data objects.

**Usage**

```r
# S3 method for class 'ldecomp'
plotScores(obj, comp = c(1, 2), main = "Scores",  
  type = "p", xlab = NULL, ylab = NULL, show.labels = FALSE,  
  show.legend = TRUE, show.axes = TRUE, ...)
```
### Arguments

- **obj**  
  object of `ldecomp` class.

- **comp**  
  which components to show the plot for (can be one value or vector with two values).

- **main**  
  main title for the plot

- **type**  
  type of the plot

- **xlab**  
  label for x-axis.

- **ylab**  
  label for y-axis.

- **show.labels**  
  logical, show or not labels for the plot objects

- **show.legend**  
  logical, show or not a legend on the plot.

- **show.axes**  
  logical, show or not a axes lines crossing origin (0,0)

- **...**  
  most of graphical parameters from `mdaplot` function can be used.

---

### Description

Shows a scores plot for selected components.

### Usage

```r
## S3 method for class 'pca'
plotScores(obj, comp = c(1, 2), type = "p",  
            main = "Scores", xlab = NULL, ylab = NULL, show.labels = F,  
            show.legend = NULL, cgroup = NULL, show.axes = TRUE, ...)
```

### Arguments

- **obj**  
  a PCA model (object of class `pca`)

- **comp**  
  a value or vector with several values - number of components to show the plot for

- **type**  
  type of the plot ("b", "l", "h")

- **main**  
  main title for the plot

- **xlab**  
  label for x axis

- **ylab**  
  label for y axis

- **show.labels**  
  logical, show or not labels for the plot objects

- **show.legend**  
  logical, show or not a legend on the plot

- **cgroup**  
  a vector with numeric values or a factor used for color grouping of plot points.

- **show.axes**  
  logical, show or not a axes lines crossing origin (0,0)

- **...**  
  other plot parameters (see `mdaplot` for details)
Details

See examples in help for `pca` function.

---

**plotSelection**

*Selected intervals plot*

---

**Description**

Selected intervals plot

**Usage**

```r
plotSelection(obj, ...)
```

**Arguments**

- `obj` a model or result object
- `...` other arguments

---

**Details**

Generic function for plotting selected intervals or variables

---

**plotSelection.ipls**

*iPLS performance plot*

---

**Description**

Shows PLS performance for each selected or excluded intervals at the first iteration

**Usage**

```r
## S3 method for class 'ipls'
plotSelection(obj, glob.ncomp = NULL,
              main = "iPLS results", xlab = obj$xaxis.name, ylab = "RMSECV",
              xlim = NULL, ylim = NULL, ...)
```

**Arguments**

- `obj` iPLS results (object of class ipls)
- `glob.ncomp` number of components for global PLS model with all intervals
- `main` main title for the plot
- `xlab` label for x-axis
- `ylab` label for y-axis
- `xlim` limits for x-axis
- `ylim` limits for y-axis
- `...` other arguments
Details

The plot shows intervals as bars, which height corresponds to RMSECV obtained when particular interval was selected (forward) or excluded (backward) from a model at the first iteration. The intervals found optimal after backward/forward iPLS selection are shown with green color while the other intervals are gray.

See examples in help for \texttt{ipls} function.

@seealso\texttt{summary.ipls, plotRMSE.ipls}

---

\texttt{plotSelectivityRatio} \quad Selectivity ratio plot

Description

Generic function for plotting selectivity ratio values for regression model (PCR, PLS, etc)

Usage

\texttt{plotSelectivityRatio(obj, \ldots)}

Arguments

- \texttt{obj} \quad a regression model
- \texttt{\ldots} \quad other parameters

---

\texttt{plotSelectivityRatio.pls} \quad Selectivity ratio plot for PLS model

Description

Shows a plot with selectivity ratio values for given number of components and response variable

Usage

\texttt{## S3 method for class \'pls\'
plotSelectivityRatio(obj, ncomp = NULL, ny = 1,
        type = "1", main = NULL, ylab = "", \ldots)}
Arguments

- **obj**: a PLS model (object of class `pls`)
- **ncomp**: number of components to get the values for (if NULL user selected as optimal will be used)
- **ny**: which response to get the values for (if y is multivariate)
- **type**: type of the plot
- **main**: main title for the plot
- **ylab**: label for y axis
- **...**: other plot parameters (see `mdaplot` for details)

References

plotSensitivity.classmodel

*Sensitivity plot for classification model*

Description

Makes a plot with sensitivity values vs. model complexity (e.g. number of components)

Usage

```r
## S3 method for class 'classmodel'
plotSensitivity(obj, nc = NULL, ...)
```

Arguments

- `obj` classification model (object of class `plsda`, `simca`, etc.).
- `nc` if there are several classes, which class to make the plot for (NULL - summary for all classes).
- `...` most of the graphical parameters from `mdaplot` function can be used.

Details

See examples in description of `plsda`, `simca` or `simcam`.

plotSensitivity.classres

*Sensitivity plot for classification results*

Description

Makes a plot with sensitivity values vs. model complexity (e.g. number of components) for classification results.

Usage

```r
## S3 method for class 'classres'
plotSensitivity(obj, nc = NULL, ...)
```

Arguments

- `obj` classification results (object of class `plsdares`, `simcamres`, etc.).
- `nc` if there are several classes, which class to make the plot for (NULL - summary for all classes).
- `...` most of the graphical parameters from `mdaplot` function can be used.
plotSpecificity

Details
See examples in description of `plsdares, simcamres, etc.

plotSpecificity Specificity plot

Description
Specificity plot

Usage
plotSpecificity(obj, ...)

Arguments
obj a model or result object
... other arguments

Details
Generic function for plotting specificity values for classification model or results

plotSpecificity.classmodel Specificity plot for classification model

Description
Makes a plot with specificity values vs. model complexity (e.g. number of components)

Usage
## S3 method for class 'classmodel'
plotSpecificity(obj, nc = NULL, ...)

Arguments
obj classification model (object of class `plsda, `simca, etc.).
nc if there are several classes, which class to make the plot for (NULL - summary for all classes).
... most of the graphical parameters from `mdaplotg function can be used.

Details
See examples in description of `plsda, `simca or `simcam.
plotSpecificity.classres

*Specificity plot for classification results*

Description

Makes a plot with specificity values vs. model complexity (e.g. number of components) for classification results.

Usage

```r
## S3 method for class 'classres'
plotSpecificity(obj, nc = NULL, ...)
```

Arguments

- `obj`: classification results (object of class `plsdares`, `simcamres`, etc.).
- `nc`: if there are several classes, which class to make the plot for (NULL - summary for all classes).
- `...`: most of the graphical parameters from `mdaplot` function can be used.

Details

See examples in description of `plsdares`, `simcamres`, etc.

plotVariance

*Variance plot*

Description

Variance plot

Usage

```r
plotVariance(obj, ...)
```

Arguments

- `obj`: a model or result object
- `...`: other arguments

Details

Generic function for plotting explained variance for data decomposition
plotVariance.ldecomp  
**Explained variance plot for linear decomposition**

**Description**

Shows a plot with explained variance values vs. number of components.

**Usage**

```r
## S3 method for class 'ldecomp'
plotVariance(obj, type = "b", main = "Variance",
             xlab = "Components", ylab = "Explained variance, %",
             show.labels = F, labels = "values", ...)
```

**Arguments**

- `obj`: object of `ldecomp` class.
- `type`: type of the plot.
- `main`: main title for the plot.
- `xlab`: label for x axis.
- `ylab`: label for y axis.
- `show.labels`: logical, show or not labels for plot objects.
- `labels`: what to show as labels for plot objects.
- `...`: most of graphical parameters from `mdaplot` function can be used.

plotVariance.pca  
**Explained variance plot for PCA**

**Description**

Shows a plot with explained variance or cumulative explained variance for components.

**Usage**

```r
## S3 method for class 'pca'
plotVariance(obj, type = "b", variance = "expvar",
             main = "Variance", xlab = "Components",
             ylab = "Explained variance, %", show.legend = T, ...)
```
Arguments

- `obj`: a PCA model (object of class `pca`)
- `type`: type of the plot ('b', 'l', 'h')
- `variance`: which variance to use ('expvar', 'cumexpvar')
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `show.legend`: logical, show or not a legend on the plot
- `...`: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `pca` function.

---

plotVariance.pls

Variance plot for PLS

Description

Shows plot with variance values vs. number of components.

Usage

```r
## S3 method for class 'pls'
plotVariance(obj, decomp = "xdecomp",
             variance = "expvar", type = "b", main = "X variance",
             xlab = "Components", ylab = "Explained variance, \%",
             labels = "values", ...)
```

Arguments

- `obj`: a PLS model (object of class `pls`)
- `decomp`: which decomposition to use ('xdecomp' for x or 'ydecomp' for y)
- `variance`: which variance to use ('expvar', 'cumexpvar')
- `type`: type of the plot('b', 'l' or 'h')
- `main`: main plot title
- `xlab`: label for x axis
- `ylab`: label for y axis
- `labels`: what to show as labels for plot objects.
- `...`: other plot parameters (see `mdaplotg` for details)

Details

See examples in help for `pls` function.
plotVIPScores

Description
Generic function for plotting VIP scores values for regression model (PCR, PLS, etc)

Usage
plotVIPScores(obj, ...)

Arguments
obj a regression model
... other parameters

plotVIPScores.pls

Description
Shows a plot with VIP scores values for given number of components and response variable

Usage
## S3 method for class 'pls'
plotVIPScores(obj, ny = 1, type = "l", main = NULL,
ylab = "", ...)

Arguments
obj a PLS model (object of class pls)
ny which response to get the values for (if y is multivariate)
type type of the plot
main main title for the plot
ylab label for y axis
... other plot parameters (see mdaplot for details)

References
plotXCumVariance  
*X cumulative variance plot*

**Description**

X cumulative variance plot

**Usage**

plotXCumVariance(obj, ...)

**Arguments**

- **obj**: a model or result object
- **...**: other arguments

**Details**

Generic function for plotting cumulative explained variance for decomposition of x data

---

**plotXCumVariance.pls**  
*Cumulative explained X variance plot for PLS*

**Description**

Shows plot with cumulative explained X variance vs. number of components.

**Usage**

```r
## S3 method for class 'pls'
plotXCumVariance(obj, type = "b",
                  main = "X cumulative variance", xlab = "Components",
                  ylab = "Explained variance, %", ...
)
```

**Arguments**

- **obj**: a PLS model (object of class pls)
- **type**: type of the plot('b', 'l' or 'h')
- **main**: main plot title
- **xlab**: label for x axis
- **ylab**: label for y axis
- **...**: other plot parameters (see mdaplotg for details)

**Details**

See examples in help for pls function.
plotXCumVariance.plsres

Explained cumulative X variance plot for PLS results

Description

Shows plot with cumulative explained X variance vs. number of components.

Usage

## S3 method for class 'plsres'
plotXCumVariance(obj, main = "X cumulative variance",
                 ...)

Arguments

- obj: PLS results (object of class plsres)
- main: main plot title
- ...: other plot parameters (see mdaplot for details)

Details

See examples in help for plsres function.

plotXLoadings

X loadings plot

Description

X loadings plot

Usage

plotXLoadings(obj, ...)

Arguments

- obj: a model or result object
- ...: other arguments

Details

Generic function for plotting loadings values for decomposition of x data
plotXLoadings

### Description

Shows plot with X loading values for selected components.

### Usage

```r
## S3 method for class 'pls'
plotXLoadings(obj, comp = c(1, 2), type = "p",
               main = "X loadings", show.axes = T, ...)
```

### Arguments

- **obj**: a PLS model (object of class `pls`)
- **comp**: which components to show the plot for (one or vector with several values)
- **type**: type of the plot
- **main**: main plot title
- **show.axes**: logical, show or not a axes lines crossing origin (0,0)
- **...**: other plot parameters (see `mdaplotg` for details)

### Details

See examples in help for `pls` function.

---

plotXResiduals

### Description

X residuals plot

### Usage

```r
plotXResiduals(obj, ...)
```

### Arguments

- **obj**: a model or result object
- **...**: other arguments

### Details

Generic function for plotting x residuals for classification or regression model or results
**plotXResiduals.pls**  

*X residuals plot for PLS*

**Description**

Shows a plot with Q residuals vs. Hotelling T2 values for PLS decomposition of x data.

**Usage**

```r
## S3 method for class 'pls'
plotXResiduals(obj, ncomp = NULL, main = NULL,
               xlab = "T2", ylab = "Squared residual distance (Q)", ...)
```

**Arguments**

- `obj`: a PLS model (object of class `pls`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
- `...`: other plot parameters (see `mdaplotg` for details)

**Details**

See examples in help for `pls` function.

---

**plotXResiduals.plsres**  

*X residuals plot for PLS results*

**Description**

Shows a plot with Q residuals vs. Hotelling T2 values for PLS decomposition of x data.

**Usage**

```r
## S3 method for class 'plsres'
plotXResiduals(obj, ncomp = NULL, main = NULL, ...
```

**Arguments**

- `obj`: PLS results (object of class `plsres`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `main`: main title for the plot
- `...`: other plot parameters (see `mdaplotg` for details)
plotXScores.pls

Details
See examples in help for plsres function.

plotXScores X scores plot

Description
X scores plot

Usage
plotXScores(obj, ...)

Arguments
obj a model or result object
... other arguments

Details
Generic function for plotting scores values for decomposition of x data

plotXScores.pls X scores plot for PLS

Description
Shows plot with X scores values for selected components.

Usage
## S3 method for class 'pls'
plotXScores(obj, comp = c(1, 2), main = "X scores",
show.axes = T, ...)

Arguments
obj a PLS model (object of class pls)
comp which components to show the plot for (one or vector with several values)
main main plot title
show.axes logical, show or not a axes lines crossing origin (0,0)
... other plot parameters (see mdaplotg for details)

Details
See examples in help for pls function.
plotXScores.plsres  

X scores plot for PLS results

Description

Shows plot with scores values for PLS decomposition of x data.

Usage

```r
## S3 method for class 'plsres'
plotXScores(obj, comp = c(1, 2), main = "X scores",
            ...)  
```

Arguments

- `obj` : PLS results (object of class `plsres`)
- `comp` : which components to show the plot for (one or vector with several values)
- `main` : main plot title
- `...` : other plot parameters (see `mdaplot` for details)

Details

See examples in help for `plsres` function.

plotXVariance  

X variance plot

Description

X variance plot

Usage

`plotXVariance(obj, ...)`

Arguments

- `obj` : a model or result object
- `...` : other arguments

Details

Generic function for plotting explained variance for decomposition of x data
plotXVariance.pls  
*Explained X variance plot for PLS*

**Description**

Shows plot with explained X variance vs. number of components.

**Usage**

```
## S3 method for class 'pls'
plotXVariance(obj, type = "b", main = "X variance",
               xlab = "Components", ylab = "Explained variance, %", ...)
```

**Arguments**

- `obj` a PLS model (object of class `pls`)
- `type` type of the plot ("b", "l" or "h")
- `main` main plot title
- `xlab` label for x axis
- `ylab` label for y axis
- `...` other plot parameters (see `mdaplotg` for details)

**Details**

See examples in help for `pls` function.

---

plotXVariance.plsres  
*Explained X variance plot for PLS results*

**Description**

Shows plot with explained X variance vs. number of components.

**Usage**

```
## S3 method for class 'plsres'
plotXVariance(obj, main = "X variance", ...)
```

**Arguments**

- `obj` PLS results (object of class `plsres`)
- `main` main plot title
- `...` other plot parameters (see `mdaplot` for details)
plotXYLoadings

Details
See examples in help for \texttt{plsres} function.

\begin{verbatim}
plotXYLoadings X loadings plot
\end{verbatim}

Description
X loadings plot

Usage
\begin{verbatim}
plotXYLoadings(obj, ...)
\end{verbatim}

Arguments
\begin{itemize}
\item obj \hspace{1cm} a model or result object
\item \ldots \hspace{1cm} other arguments
\end{itemize}

Details
Generic function for plotting loadings values for decomposition of x and y data

\begin{verbatim}
plotXYLoadings.pls XY loadings plot for PLS
\end{verbatim}

Description
Shows plot with X and Y loading values for selected components.

Usage
\begin{verbatim}
## S3 method for class 'pls'
plotXYLoadings(obj, comp = c(1, 2), main = "XY loadings",
 show.axes = F, ...)
\end{verbatim}

Arguments
\begin{itemize}
\item obj \hspace{1cm} a PLS model (object of class pls)
\item comp \hspace{1cm} which components to show the plot for (one or vector with several values)
\item main \hspace{1cm} main plot title
\item show.axes \hspace{1cm} logical, show or not a axes lines crossing origin (0,0)
\item \ldots \hspace{1cm} other plot parameters (see \texttt{mdaplotg} for details)
\end{itemize}

Details
See examples in help for \texttt{pls} function.
**Description**

XY scores plot

**Usage**

```r
plotXYScores(obj, ...)```

**Arguments**

- `obj`: a model or result object
- `...`: other arguments

**Details**

Generic function for plotting scores values for decomposition of x and y data.

**Description**

Shows plot with X vs. Y scores values for selected component.

**Usage**

```r
## S3 method for class 'pls'
plotXYScores(obj, comp = 1, main = "XY scores",
             show.axes = T, ...)```

**Arguments**

- `obj`: a PLS model (object of class `pls`)
- `comp`: which component to show the plot for
- `main`: main plot title
- `show.axes`: logical, show or not a axes lines crossing origin (0,0)
- `...`: other plot parameters (see `mdaplotg` for details)

**Details**

See examples in help for `pls` function.
plotXYScores.plsres  

**XY scores plot for PLS results**

### Description

Shows plot with X vs. Y scores values for PLS results.

### Usage

```
## S3 method for class 'plsres'
plotXYScores(obj, ncomp = 1, ...)
```

### Arguments

- `obj`: PLS results (object of class `plsres`)
- `ncomp`: which component to show the plot for
- `...`: other plot parameters (see `mdaplot` for details)

### Details

See examples in help for `plsres` function.

---

plotYCumVariance

**Y cumulative variance plot**

### Description

Y cumulative variance plot

### Usage

```
plotYCumVariance(obj, ...)
```

### Arguments

- `obj`: a model or result object
- `...`: other arguments

### Details

Generic function for plotting cumulative explained variance for decomposition of y data
plotYCumVariance.pls

**Cumulative explained Y variance plot for PLS**

---

**Description**

Shows plot with cumulative explained Y variance vs. number of components.

**Usage**

```r
## S3 method for class 'pls'
plotYCumVariance(obj, type = "b",
                 main = "Y cumulative variance", xlab = "Components",
                 ylab = "Explained variance, %", ...)
```

**Arguments**

- `obj`: a PLS model (object of class `pls`)
- `type`: type of the plot('b', 'l' or 'h')
- `main`: main plot title
- `xlab`: label for x axis
- `ylab`: label for y axis
- `...`: other plot parameters (see `mdaplotg` for details)

**Details**

See examples in help for `pls` function.

---

plotYCumVariance.plsres

**Explained cumulative Y variance plot for PLS results**

---

**Description**

Shows plot with cumulative explained Y variance vs. number of components.

**Usage**

```r
## S3 method for class 'plsres'
plotYCumVariance(obj, main = "Y cumulative variance",
                 ...)
```

---
**plotYResiduals**

**Arguments**

- `obj` PLS results (object of class `plsres`)
- `main` main plot title
- `...` other plot parameters (see `mdaplot` for details)

**Details**

See examples in help for `plsres` function.

---

**plotYResiduals**

_Y residuals plot_

---

**Description**

_Y residuals plot_

**Usage**

`plotYResiduals(obj, ...)`

**Arguments**

- `obj` a model or result object
- `...` other arguments

**Details**

Generic function for plotting y residuals for classification or regression model or results

---

**plotYResiduals.pls**

_Y residuals plot for PLS_

---

**Description**

Shows plot with y residuals for selected components.

**Usage**

```
# S3 method for class 'pls'
plotYResiduals(obj, ncomp = NULL, ny = 1, main = NULL,
    show.line = T, ...)
```
Arguments

obj      a PLS model (object of class pls)
ncomp    how many components to use (if NULL - user selected optimal value will be used)
ny       number of response variable to make the plot for (if y is multivariate)
main     main plot title
show.line logical, show or not line for 0 value
...      other plot parameters (see mdaplotg for details)

Details

See examples in help for pls function.

plotYResiduals.regres  Residuals plot for regression results

Description

Shows plot with Y residuals (difference between predicted and reference values) for selected response variable and complexity (number of components).

Usage

## S3 method for class 'regres'
plotYResiduals(obj, ny = 1, ncomp = NULL,
               show.line = T, ...)

Arguments

obj       regression results (object of class regres)
ny        number of predictor to show the plot for (if y is multivariate)
ncomp     complexity of model (e.g. number of components) to show the plot for
show.line logical, show or not zero line on the plot
...       other plot parameters (see mdaplot for details)
plotYVariance

Description

Y variance plot

Usage

plotYVariance(obj, ...)

Arguments

obj a model or result object

... other arguments

Details

Generic function for plotting explained variance for decomposition of y data

plotYVariance.pls

Explanation Y variance plot for PLS

Description

Shows plot with explained Y variance vs. number of components.

Usage

## S3 method for class 'pls'
plotYVariance(obj, type = "b", main = "Y variance",
              xlab = "Components", ylab = "Explained variance, %", ...)

Arguments

obj a PLS model (object of class pls)
type type of the plot ('b', 'l' or 'h')
main main plot title
xlab label for x axis
ylab label for y axis

... other plot parameters (see mdaplotg for details)

Details

See examples in help for pls function.
plotYVariance.plsres  
*Explained Y variance plot for PLS results*

**Description**

Shows plot with explained Y variance vs. number of components.

**Usage**

```r
## S3 method for class 'plsres'
plotYVariance(obj, main = "Y variance", ...)
```

**Arguments**

- `obj`  
  PLS results (object of class `plsres`)  
- `main`  
  main plot title  
- `...`  
  other plot parameters (see `mdaplot` for details)

**Details**

See examples in help for `plsres` function.

---

**pls**  
*Partial Least Squares regression*

**Description**

`pls` is used to calibrate, validate and use of partial least squares (PLS) regression model.

**Usage**

```r
pls(x, y, ncomp = 15, center = T, scale = F, cv = NULL,
    exclcols = NULL, exclrows = NULL, x.test = NULL, y.test = NULL,
    method = "simpls", alpha = 0.05, coefs.ci = NULL,
    coefs.alpha = 0.05, info = "", light = F, ncomp.selcrit = "min")
```

**Arguments**

- `x`  
  matrix with predictors.  
- `y`  
  matrix with responses.  
- `ncomp`  
  maximum number of components to calculate.  
- `center`  
  logical, center or not predictors and response values.  
- `scale`  
  logical, scale (standardize) or not predictors and response values.
cv

number of segments for cross-validation (if cv = 1, full cross-validation will be
used).

exclcols

columns of x to be excluded from calculations (numbers, names or vector with
logical values)

exclrows

rows to be excluded from calculations (numbers, names or vector with logical
values)

x.test

matrix with predictors for test set.

y.test

matrix with responses for test set.

method

algorithm for computing PLS model (only 'simpls' is supported so far)

alpha

significance level for calculating statistical limits for residuals.

coeffs.ci

method to calculate p-values and confidence intervals for regression coefficients
(so far only jack-knifing is available: = 'jk').

coeffs.alpha

significance level for calculating confidence intervals for regression coefficients.

info

short text with information about the model.

light

run normal or light (faster) version of PLS without calculating some perfor-
mance statistics.

ncomp.selcrit

criterion for selecting optimal number of components ('min' for first local mini-
um of RMSECV and 'wold' for Wold's rule.)

Details

So far only SIMPLS method [1] is available, more coming soon. Implementation works both with
one and multiple response variables.

Like in pca, pls uses number of components (ncomp) as a minimum of number of objects - 1,
number of x variables and the default or provided value. Regression coefficients, predictions and
other results are calculated for each set of components from 1 to ncomp: 1, 1:2, 1:3, etc. The optimal
number of components, (ncomp.selected), is found using Wold's R criterion, but can be adjusted
by user using function (selectCompNum.pls). The selected optimal number of components is used
for all default operations - predictions, plots, etc.

Selectivity ratio [2] and VIP scores [3] are calculated for any PLS model automatically, however
while selectivity ratio values are calculated for all computed components, the VIP scores are com-
puted only for selected components (to save calculation time) and recalculated every time when
selectCompNum() is called for the model.

Calculation of confidence intervals and p-values for regression coefficients are available only by
jack-knifing so far. See help for regcoeffs objects for details.

Value

Returns an object of pls class with following fields:

ncomp

number of components included to the model.

ncomp.selected

selected (optimal) number of components.

xloadings

matrix with loading values for x decomposition.
yloadings matrix with loading values for y decomposition.
weights matrix with PLS weights.
selratio array with selectivity ratio values.
vipscores matrix with VIP scores values.
coeffs object of class regcoeffs with regression coefficients calculated for each component.
info information about the model, provided by user when build the model.
calres an object of class plsres with PLS results for a calibration data.
testres an object of class plsres with PLS results for a test data, if it was provided.
cvres an object of class plsres with PLS results for cross-validation, if this option was chosen.

Author(s)
Sergey Kucheryavskiy (svkucheryavski@gmail.com)

References

See Also
Methods for pls objects:

print prints information about a pls object.
summary.pls shows performance statistics for the model.
plot.pls shows plot overview of the model.
pls.simpls implementation of SIMPLS algorithm.
predict.pls applies PLS model to a new data.
selectCompNum.pls set number of optimal components in the model.
plotPredictions.pls shows predicted vs. measured plot.
plotRegecoeffs.pls shows regression coefficients plot.
plotXScores.pls shows scores plot for x decomposition.
plotXYScores.pls shows scores plot for x and y decomposition.
plotXLoadings.pls shows loadings plot for x decomposition.
plotXYLoadings.pls shows loadings plot for x and y decomposition.
plotRMSE.pls shows RMSE plot.
plotXVariance.pls shows explained variance plot for x decomposition.
plotYVariance.pls shows explained variance plot for y decomposition.
plotXCumVariance.pls shows cumulative explained variance plot for x decomposition.
plotYCumVariance.pls shows cumulative explained variance plot for y decomposition.
plotXResiduals.pls shows T2 vs. Q plot for x decomposition.
plotYResiduals.pls shows residuals plot for y values.
plotSelectivityRatio.pls shows plot with selectivity ratio values.
plotVIPScores.pls shows plot with VIP scores values.
getSelectivityRatio.pls returns vector with selectivity ratio values.
getVIPScores.pls returns vector with VIP scores values.
getRegcoeffs.pls returns matrix with regression coefficients.

Most of the methods for plotting data (except loadings and regression coefficients) are also available for PLS results (plsres) objects. There is also a randomization test for PLS-regression (randtest).

Examples

```r
### Examples of using PLS model class
library(mdatools)

## 1. Make a PLS model for concentration of first component
## using full-cross validation and automatic detection of
## optimal number of components and show an overview

data(simdata)
x = simdata$spectra[,1]
y = simdata$conc[,1]

model = pls(x, y, ncomp = 8, cv = 1)
summary(model)
plot(model)

## 2. Make a PLS model for concentration of first component
## using test set and 10 segment cross-validation and show overview

data(simdata)
x = simdata$spectra[,1]
y = simdata$conc[,1]
x.t = simdata$spectra.t
y.t = simdata$conc.t[,1]

model = pls(x, y, ncomp = 8, cv = 10, x.test = x.t, y.test = y.t)
model = selectCompNum(model, 2)
summary(model)
plot(model)

## 3. Make a PLS model for concentration of first component
## using only test set validation and show overview

data(simdata)
x = simdata$spectra[,1]
y = simdata$conc[,1]
x.t = simdata$spectra.t
y.t = simdata$conc.t[,1]

model = pls(x, y, ncomp = 6, x.test = x.t, y.test = y.t)
model = selectCompNum(model, 2)
summary(model)
plot(model)
```
## 4. Show variance and error plots for a PLS model
par(mfrow = c(2, 2))
plotXCumVariance(model, type = 'h')
plotYCumVariance(model, type = 'b', show.labels = TRUE, legend.position = 'bottomright')
plotRMSE(model)
plotRMSE(model, type = 'h', show.labels = TRUE)
par(mfrow = c(1, 1))

## 5. Show scores plots for a PLS model
par(mfrow = c(2, 2))
plotXScores(model)
plotXScores(model, comp = c(1, 3), show.labels = TRUE)
plotXYScores(model)
plotXYScores(model, comp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

## 6. Show loadings and coefficients plots for a PLS model
par(mfrow = c(2, 2))
plotXLoadings(model)
plotXLoadings(model, comp = c(1, 2), type = 'l')
plotXYLoadings(model, comp = c(1, 2), legend.position = 'topleft')
plotRegCoeffs(model)
par(mfrow = c(1, 1))

## 7. Show predictions and residuals plots for a PLS model
par(mfrow = c(2, 2))
plotXResiduals(model, show.label = TRUE)
plotYResiduals(model, show.label = TRUE)
plotPredictions(model)
plotPredictions(model, ncomp = 4, xlab = 'C', reference', ylab = 'C, predictions')
par(mfrow = c(1, 1))

## 8. Selectivity ratio and VIP scores plots
par(mfrow = c(2, 2))
plotSelectivityRatio(model)
plotSelectivityRatio(model, ncomp = 1)
par(mfrow = c(1, 1))

## 9. Variable selection with selectivity ratio
selratio = getSelectivityRatio(model)
selvar = !(selratio < 8)
xsel = x[, selvar]
modelsel = pls(xsel, y, ncomp = 6, cv = 1)
modelsel = selectCompNum(modelsel, 3)
summary(model)
summary(modelsel)

## 10. Calculate average spectrum and show the selected variables
i = 1:ncol(x)
ms = apply(x, 2, mean)
pls.cal

\begin{verbatim}
par(mfrow = c(2, 2))

plot(i, ms, type = 'p', pch = 16, col = 'red', main = 'Original variables')
plotPredictions(model)

plot(i, ms, type = 'p', pch = 16, col = 'lightgray', main = 'Selected variables')
points(i[selvar], ms[selvar], col = 'red', pch = 16)
plotPredictions(modelsel)

par(mfrow = c(1, 1))
\end{verbatim}

\begin{center}
\textbf{pls.cal} \hspace{1cm} \textit{PLS model calibration}
\end{center}

\section*{Description}
Calibrates (builds) a PLS model for given data and parameters.

\section*{Usage}
\begin{verbatim}
pls.cal(x, y, ncomp, center, scale, method, cv, alpha, coeffs.ci, coeffs.alpha, info, light, exclcols = NULL, exclrows = NULL, ncomp.selcrit)
\end{verbatim}

\section*{Arguments}
\begin{itemize}
\item \textbf{x} \hspace{1cm} a matrix with x values (predictors)
\item \textbf{y} \hspace{1cm} a matrix with y values (responses)
\item \textbf{ncomp} \hspace{1cm} number of components to calculate
\item \textbf{center} \hspace{1cm} logical, do mean centering or not
\item \textbf{scale} \hspace{1cm} logical, do standardization or not
\item \textbf{method} \hspace{1cm} algorithm for computing PLS model (only 'simpls' is supported so far)
\item \textbf{cv} \hspace{1cm} logical, does calibration for cross-validation or not
\item \textbf{alpha} \hspace{1cm} significance level for calculating statistical limits for residuals.
\item \textbf{coeffs.ci} \hspace{1cm} method to calculate p-values and confidence intervals for regression coefficients (so far only jack-knifing is available: = 'jk').
\item \textbf{coeffs.alpha} \hspace{1cm} significance level for calculating confidence intervals for regression coefficients.
\item \textbf{info} \hspace{1cm} short text with information about the model.
\item \textbf{light} \hspace{1cm} run normal or light (faster) version of PLS without calculating some performance statistics.
\item \textbf{exclcols} \hspace{1cm} columns of x to be excluded from calculations (numbers, names or vector with logical values)
\end{itemize}
pls.calculateSelectivityRatio

Selectivity ratio calculation

Description

Calculates selectivity ratio for each component and response variable in the PLS model

Usage

pls.calculateSelectivityRatio(model, x)

Arguments

model  a PLS model (object of class pls)
x  predictor values from calibration set, preprocessed, centered and scaled

Value

array nvar x ncomp x ny with selectivity ratio values

References

pls.calculateVIPScores

VIP scores calculation for PLS model

Description

Calculates VIP (Variable Importance in Projection) scores for each component and response variable in the PLS model

Usage

pls.calculateVIPScores(object)

Arguments

object a PLS model (object of class pls)

Value

matrix nvar x ny with VIP score values for selected number of components

pls.crossval Cross-validation of a PLS model

Description

Does the cross-validation of a PLS model

Usage

pls.crossval(model, x, y, cv, center, scale, method, jack.knife = T)

Arguments

model a PLS model (object of class pls)
x a matrix with x values (predictors from calibration set)
y a matrix with y values (responses from calibration set)
cv number of segments (if cv = 1, full cross-validation will be used)
center logical, do mean centering or not
scale logical, do standardization or not
method algorithm for computing PLS model
jack.knife logical, do jack-knifing or not

Value

object of class plsres with results of cross-validation
pls.run  

*Runs selected PLS algorithm*

**Description**

Runs selected PLS algorithm

**Usage**

```r
pls.run(x, y, ncomp, method, cv)
```

**Arguments**

- `x`: a matrix with x values (predictors from calibration set)
- `y`: a matrix with y values (responses from calibration set)
- `ncomp`: how many components to compute
- `method`: algorithm for computing PLS model
- `cv`: logical, is this for CV or not

pls.simpls  

*SIMPLS algorithm*

**Description**

SIMPLS algorithm for calibration of PLS model

**Usage**

```r
pls.simpls(x, y, ncomp, cv = FALSE)
```

**Arguments**

- `x`: a matrix with x values (predictors)
- `y`: a matrix with y values (responses)
- `ncomp`: number of components to calculate
- `cv`: logical, is model calibrated during cross-validation or not

**Value**

a list with computed regression coefficients, loadings and scores for x and y matrices, and weights.

**References**

Partial Least Squares Discriminant Analysis

Description

`plsda` is used to calibrate, validate and use a partial least squares discrimination analysis (PLS-DA) model.

Usage

```r
plsda(x, c, ncomp = 15, center = T, scale = F, cv = NULL,
     exclcols = NULL, exclrows = NULL, x.test = NULL, c.test = NULL,
     method = "simpls", alpha = 0.05, coefs.ci = NULL,
     coefs.alpha = 0.1, info = "", light = F, ncomp.selcrit = "min",
     classname = NULL)
```

Arguments

- **x**: matrix with predictors.
- **c**: vector with class membership (should be either a factor with class names/numbers in case of multiple classes or a vector with logical values in case of one class model).
- **ncomp**: maximum number of components to calculate.
- **center**: logical, center or not predictors and response values.
- **scale**: logical, scale (standardize) or not predictors and response values.
- **cv**: number of segments for cross-validation (if `cv = 1`, full cross-validation will be used).
- **exclcols**: columns of x to be excluded from calculations (numbers, names or vector with logical values).
- **exclrows**: rows to be excluded from calculations (numbers, names or vector with logical values).
- **x.test**: matrix with predictors for test set.
- **c.test**: vector with reference class values for test set (same format as calibration values).
- **method**: method for calculating PLS model.
- **alpha**: significance level for calculating statistical limits for residuals.
- **coefs.ci**: method to calculate p-values and confidence intervals for regression coefficients (so far only jack-knifing is available: = 'jk').
- **coefs.alpha**: significance level for calculating confidence intervals for regression coefficients.
- **info**: short text with information about the model.
- **light**: run normal or light (faster) version of PLS without calculation of some performance statistics.
The `plsda` class is based on `pls` with extra functions and plots covering classification functionality. All plots for `pls` can be used. E.g. of you want to see the real predicted values (y in PLS) instead of classes use `plotPredictions.pls(model)` instead of `plotPredictions(model)`.

Calculation of confidence intervals and p-values for regression coefficients are available only by jack-knifing so far. See help for `regcoeffs` objects for details.

**Value**

Returns an object of `plsda` class with following fields (most inherited from `class pls`):

- `ncomp` number of components included to the model.
- `ncomp.selected` selected (optimal) number of components.
- `xloadings` matrix with loading values for x decomposition.
- `yloadings` matrix with loading values for y (c) decomposition.
- `weights` matrix with PLS weights.
- `coeffs` matrix with regression coefficients calculated for each component.
- `info` information about the model, provided by user when build the model.
- `calres` an object of class `plsdares` with PLS-DA results for a calibration data.
- `testres` an object of class `plsdares` with PLS-DA results for a test data, if it was provided.
- `cvres` an object of class `plsdares` with PLS-DA results for cross-validation, if this option was chosen.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

**See Also**

Specific methods for `plsda` class:

- `print.plsda` prints information about a `pls` object.
- `summary.plsda` shows performance statistics for the model.
- `plot.plsda` shows plot overview of the model.
- `predict.plsda` applies PLS-DA model to a new data.
Methods, inherited from class model class:

- `plotPredictions.classmodel` shows plot with predicted values.
- `plotSensitivity.classmodel` shows sensitivity plot.
- `plotSpecificity.classmodel` shows specificity plot.
- `plotMisclassified.classmodel` shows misclassified ratio plot.

See also methods for class `pls`.

**Examples**

```r
### Examples for PLS-DA model class

library(mdatools)

## 1. Make a PLS-DA model with full cross-validation and show model overview

# make a calibration set from iris data (3 classes)
# use names of classes as class vector
x.cal = iris[seq(1, nrow(iris), 2), 1:4]
c.cal = iris[seq(1, nrow(iris), 2), 5]

model = plsda(x.cal, c.cal, ncomp = 3, cv = 1, info = 'IRIS data example')
model = selectCompNum(model, 1)

# show summary and basic model plots
# misclassification will be shown only for first class
summary(model)
plot(model)

# summary and model plots for second class
summary(model, nc = 2)
plot(model, nc = 2)

# summary and model plot for specific class and number of components
summary(model, nc = 3, ncomp = 3)
plot(model, nc = 3, ncomp = 3)

## 2. Show performance plots for a model
par(mfrow = c(2, 2))
plotSpecificity(model)
plotSensitivity(model)
plotMisclassified(model)
plotMisclassified(model, nc = 2)
par(mfrow = c(1, 1))

## 3. Show both class and y values predictions
par(mfrow = c(2, 2))
plotPredictions(model)
plotPredictions(model, res = 'calres', ncomp = 2, nc = 2)
plotPredictions(structure(model, class = "pls"))
```
plotPredictions(structure(model, class = "pls"), ncomp = 2, ny = 2)
par(mfrow = c(1, 1))

### 4. All plots from ordinary PLS can be used, e.g.:
par(mfrow = c(2, 2))
plotXYScores(model)
plotYVariance(model)
plotXResiduals(model)
plotRegcoeffs(model, ny = 2)
par(mfrow = c(1, 1))

---

**Calibrate PLS-DA model**

**Description**

Calibrate PLS-DA model

**Usage**

```r
plsd.cal(x, c, ncomp, center, scale, cv, method, light, alpha, coefs.ci,
coefs.alpha, info, exclcols = NULL, exclrows = NULL, ncomp.selcrit,
classname = NULL)
```

**Arguments**

- `x`: matrix with predictors.
- `c`: vector with reference class values.
- `ncomp`: maximum number of components to calculate.
- `center`: logical, center or not predictors and response values.
- `scale`: logical, scale (standardize) or not predictors and response values.
- `cv`: number of segments for cross-validation (if `cv` = 1, full cross-validation will be used).
- `method`: method for calculating PLS model.
- `light`: run normal or light (faster) version of PLS without calculating some performance statistics.
- `alpha`: significance level for calculating statistical limits for residuals.
- `coefs.ci`: method to calculate p-values and confidence intervals for regression coefficients (so far only jack-knifing is available: = 'jk').
- `coefs.alpha`: significance level for calculating confidence intervals for regression coefficients.
- `info`: short text with information about the model.
- `exclcols`: columns of `x` to be excluded from calculations (numbers, names or vector with logical values).
**Description**

Does the cross-validation of a PLS-DA model

**Usage**

```r
plsda.crossval(model, x, c, y, center, scale, method)
```

**Arguments**

- `model`: a PLS-DA model (object of class `plsda`)
- `x`: a matrix with x values (predictors from calibration set)
- `c`: a vector with c values (classes from calibration set)
- `y`: a matrix with dummy y-values for PLS regression
- `center`: logical, do mean centering or not
- `scale`: logical, do standardization or not
- `method`: method for calculating PLS model.

**Value**

Object of class `plsdares` with results of cross-validation

---

**Description**

`plsdares` is used to store and visualize results of applying a PLS-DA model to a new data.

**Usage**

```r
plsdares(plsres, cres)
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plsres</td>
<td>PLS results for the data.</td>
</tr>
<tr>
<td>cres</td>
<td>Classification results for the data.</td>
</tr>
</tbody>
</table>

Details

Do not use plsdares manually, the object is created automatically when one applies a PLS-DA model to a new data set, e.g. when calibrate and validate a PLS-DA model (all calibration and validation results in PLS-DA model are stored as objects of plsdares class) or use function predict.plsda.

The object gives access to all PLS-DA results as well as to the plotting methods for visualisation of the results. The plsdares class also inherits all properties and methods of classres and plsres classes.

If no reference values provided, classification statistics will not be calculated and performance plots will not be available.

Value

Returns an object of plsdares class with fields, inherited from classres and plsres.

See Also

Methods for plsda objects:

- `print.plsda` shows information about the object.
- `summary.plsda` shows statistics for results of classification.
- `plot.plsda` shows plots for overview of the results.

Methods, inherited from classres class:

- `showPredictions.classres` show table with predicted values.
- `plotPredictions.classres` makes plot with predicted values.
- `plotSensitivity.classres` makes plot with sensitivity vs. components values.
- `plotSpecificity.classres` makes plot with specificity vs. components values.
- `plotPerformance.classres` makes plot with both specificity and sensitivity values.

Methods, inherited from plsres class:

- `plotPredictions.plsres` shows predicted vs. measured plot.
- `plotXScores.plsres` shows scores plot for x decomposition.
- `plotXYScores.plsres` shows scores plot for x and y decomposition.
- `plotRMSE.regres` shows RMSE plot.
- `plotXVariance.plsres` shows explained variance plot for x decomposition.
- `plotYVariance.plsres` shows explained variance plot for y decomposition.
- `plotXCumVariance.plsres` shows cumulative explained variance plot for y decomposition.
plotYCumVariance.plsres shows cumulative explained variance plot for y decomposition.
plotXResiduals.plsres shows T2 vs. Q plot for x decomposition.
plotYResiduals.regres shows residuals plot for y values.

See also plsda - a class for PLS-DA models, predict.plsda applying PLS-DA model for a new dataset.

Examples

```r
### Examples for PLS-DA results class

library(mdatools)

## 1. Make a PLS-DA model with full cross-validation, get
## calibration results and show overview

# make a calibration set from iris data (3 classes)
# use names of classes as class vector
x.cal = iris[seq(1, nrow(iris), 2), 1:4]
c.cal = iris[seq(1, nrow(iris), 2), 5]

model = plsda(x.cal, c.cal, ncomp = 3, cv = 1, info = 'IRIS data example')
model = selectCompNum(model, 1)

res = model$calres

# show summary and basic plots for calibration results
summary(res)
plot(res)

## 2. Apply the calibrated PLS-DA model to a new dataset

# make a new data
x.new = iris[seq(2, nrow(iris), 2), 1:4]
c.new = iris[seq(2, nrow(iris), 2), 5]

res = predict(model, x.new, c.new)
summary(res)
plot(res)

## 3. Show performance plots for the results
par(mfrow = c(2, 2))
plotSpecificity(res)
plotSensitivity(res)
plotMisclassified(res)
plotMisclassified(res, nc = 2)
par(mfrow = c(1, 1))

## 3. Show both class and y values predictions
par(mfrow = c(2, 2))
plotPredictions(res)
```
plotPredictions(res, ncomp = 2, nc = 2)
plotPredictions(structure(res, class = "plsres"))
plotPredictions(structure(res, class = "plsres"), ncomp = 2, ny = 2)
par(mfrow = c(1, 1))

## 4. All plots from ordinary PLS results can be used, e.g.:
par(mfrow = c(2, 2))
plotXYScores(res)
plotYVariance(res, type = 'h')
plotXVariance(res, type = 'h')
plotXResiduals(res)
par(mfrow = c(1, 1))

---

### plsres

**PLS results**

**Description**

`plsres` is used to store and visualize results of applying a PLS model to a new data.

**Usage**

```r
plsres(y.pred, y.ref = NULL, ncomp.selected = NULL, xdecomp = NULL,
ydecomp = NULL, info = "")
```

**Arguments**

- `y.pred`  
  predicted y values.
- `y.ref`  
  reference (measured) y values.
- `ncomp.selected`  
  selected (optimal) number of components.
- `xdecomp`  
  PLS decomposition of X data (object of class `ldecomp`).
- `ydecomp`  
  PLS decomposition of Y data (object of class `ldecomp`).
- `info`  
  information about the object.

**Details**

Do not use `plsres` manually, the object is created automatically when one applies a PLS model to a new data set, e.g. when calibrate and validate a PLS model (all calibration and validation results in PLS model are stored as objects of `plsres` class) or use function `predict.pls`.

The object gives access to all PLS results as well as to the plotting methods for visualisation of the results. The `plsres` class also inherits all properties and methods of `regres` - general class for regression results.

If no reference values provided, regression statistics will not be calculated and most of the plots not available. The class is also used for cross-validation results, in this case some of the values and methods are not available (e.g. scores and scores plot, etc.).
All plots are based on `mdaplot` function, so most of its options can be used (e.g. color grouping, etc.).

RPD is ratio of standard deviation of response values to standard error of prediction (SDy/SEP).

**Value**

Returns an object of `plsres` class with following fields:

- `ncomp` number of components included to the model.
- `ncompselected` selected (optimal) number of components.
- `y.ref` a matrix with reference values for responses.
- `y.pred` a matrix with predicted values for responses.
- `rmse` a matrix with root mean squared error values for each response and component.
- `slope` a matrix with slope values for each response and component.
- `r2` a matrix with determination coefficients for each response and component.
- `bias` a matrix with bias values for each response and component.
- `sep` a matrix with standard error values for each response and component.
- `rpd` a matrix with RPD values for each response and component.
- `xdecomp` decomposition of predictors (object of class `ldecomp`).
- `ydecomp` decomposition of responses (object of class `ldecomp`).
- `info` information about the object.

**See Also**

Methods for `plsres` objects:

- `print` prints information about a `plsres` object.
- `summary.plsres` shows performance statistics for the results.
- `plot.plsres` shows plot overview of the results.
- `plotPredictions.plsres` shows predicted vs. measured plot.
- `plotXScores.plsres` shows scores plot for x decomposition.
- `plotXYScores.plsres` shows scores plot for x and y decomposition.
- `plotRMSE.regres` shows RMSE plot.
- `plotXVariance.plsres` shows explained variance plot for x decomposition.
- `plotYVariance.plsres` shows explained variance plot for y decomposition.
- `plotXCumVariance.plsres` shows cumulative explained variance plot for y decomposition.
- `plotYCumVariance.plsres` shows cumulative explained variance plot for y decomposition.
- `plotXResiduals.plsres` shows T2 vs. Q plot for x decomposition.
- `plotYResiduals.regres` shows residuals plot for y values.

See also `pls` - a class for PLS models.
Examples

### Examples of using PLS result class

```r
library(mdatools)

# 1. Make a PLS model for concentration of first component
# using full-cross validation and get calibration results

data(simdata)
x = simdata$spectra.c
y = simdata$conc.c[, 1]

model = pls(x, y, ncomp = 8, cv = 1)
model = selectCompNum(model, 2)
res = model$calres

summary(res)
plot(res)

# 2. Make a PLS model for concentration of first component
# and apply model to a new dataset

data(simdata)
x = simdata$spectra.c
y = simdata$conc.c[, 1]

model = pls(x, y, ncomp = 6, cv = 1)
model = selectCompNum(model, 2)

x.new = simdata$spectra.t
y.new = simdata$conc.t[, 1]
res = predict(model, x.new, y.new)

summary(res)
plot(res)

# 3. Show variance and error plots for PLS results
par(mfrow = c(2, 2))
plotXCumVariance(res, type = 'h')
plotYCumVariance(res, type = 'b', show.labels = TRUE, legend.position = 'bottomright')
plotRMSE(res)
plotRMSE(res, type = 'h', show.labels = TRUE)
par(mfrow = c(1, 1))

# 4. Show scores plots for PLS results
# (for results plot we can use color grouping)
par(mfrow = c(2, 2))
plotXScores(res)
plotXScores(res, show.labels = TRUE, cgroup = y.new)
plotXYScores(res)
plotXYScores(res, comp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

# 5. Show predictions and residuals plots for PLS results
```
predict.pca

par(mfrow = c(2, 2))
plotXResiduals(res, show.label = TRUE, cgroup = y.new)
plotYResiduals(res, show.label = TRUE)
plotPredictions(res)
plotPredictions(res, ncomp = 4, xlab = 'C, reference', ylab = 'C, predictions')
par(mfrow = c(1, 1))

---

**predict.pca**  
*PCA predictions*

**Description**  
Applies PCA model to a new data

**Usage**  
```r
## S3 method for class 'pca'
predict(object, x, cal = FALSE, ...)
```

**Arguments**

- **object**: a PCA model (object of class `pca`)
- **x**: a matrix with data values
- **cal**: logical, if TRUE the predictions are made for calibration set
- **...**: other arguments

**Value**

PCA results (an object of class `pcares`)

---

**predict.pls**  
*PLS predictions*

**Description**  
Applies PLS model to a new data set

**Usage**  
```r
## S3 method for class 'pls'
predict(object, x, y = NULL, ...)
```
**Arguments**

- **object**: a PLS model (object of class `pls`)
- **x**: a matrix with x values (predictors)
- **y**: a matrix with reference y values (responses)
- **...**: other arguments

**Details**

See examples in help for `pls` function.

**Value**

PLS results (an object of class `plsres`)

---

**predict.plsda**

---

**Description**

Applies PLS-DA model to a new data set

**Usage**

```r
## S3 method for class 'plsda'
predict(object, x, c.ref = NULL, ...)
```

**Arguments**

- **object**: a PLS-DA model (object of class `plsda`)
- **x**: a matrix with x values (predictors)
- **c.ref**: a vector with reference class values (should be a factor)
- **...**: other arguments

**Details**

See examples in help for `plsda` function.

**Value**

PLS-DA results (an object of class `plsdares`)

---
predict.simca

SIMCA predictions

Description
Applies SIMCA model to a new data set

Usage
## S3 method for class 'simca'
predict(object, x, c.ref = NULL, cal = FALSE, ...)

Arguments
- `object`: a SIMCA model (object of class `simca`)
- `x`: a matrix with x values (predictors)
- `c.ref`: a vector with reference class names (same as class names for models)
- `cal`: logical, are predictions for calibration set or not
- `...`: other arguments

Details
See examples in help for `simca` function.

Value
SIMCA results (an object of class `simcares`)

predict.simcam

SIMCA multiple classes predictions

Description
Applies SIMCAM model (SIMCA for multiple classes) to a new data set

Usage
## S3 method for class 'simcam'
predict(object, x, c.ref = NULL, cv = F, ...)

Arguments
- `object`: a SIMCA model (object of class `simca`)
- `x`: a matrix with x values (predictors)
- `c.ref`: a vector with reference class names (same as class names for models)
- `cv`: logical, are predictions for calibration set or not
- `...`: other arguments
prep.autoscale

Arguments

object    a SIMCAM model (object of class simcam)
x         a matrix with x values (predictors)
c.ref     a vector with reference class names (same as class names in models)
cv        logical, are predictions for cross-validation or not
...       other arguments

Details

See examples in help for simcam function.

Value

SIMCAM results (an object of class simcamres)

Description

Autoscale (mean center and standardize) values in columns of data matrix.
The use of ‘max.cov’ allows to avoid overestimation of inert variables, which vary very little. Note, that the ‘max.cov’ value is already in percent, e.g. if ‘max.cov = 0.1’ it will compare the coefficient of variation of every variable with 0.1 want to use this option simply keep ‘max.cov = 0’.

Usage

prep.autoscale(data, center = T, scale = F, max.cov = 0)

Arguments

data          a matrix with data values
center        a logical value or vector with numbers for centering
scale         a logical value or vector with numbers for weighting
max.cov       columns that have coefficient of variation (in percent) below or equal to ‘max.cov’ will not be scaled

Value

data matrix with processed values
**prep.msc**

*Multiplicative Scatter Correction transformation*

**Description**

Applies Multiplicative Scatter Correction (MSC) transformation to data matrix (spectra)

**Usage**

```r
prep.msc(spectra, mspectrum = NULL)
```

**Arguments**

- `spectra`: a matrix with spectra values
- `mspectrum`: mean spectrum (if NULL will be calculated from `spectra`)

**Details**

MSC is used to remove scatter effects (baseline offset and slope) from spectral data, e.g. NIR spectra.

**@examples**

### Apply MSC to spectra from simdata

```r
library(mdatools) data(simdata)
spectra = simdata$spectra[, simdata$wavelength]
res = prep.msc(spectra) cspectra = res$cspectra
par(mfrow = c(2, 1)) mdaplot(cbind(wavelength, t(spectra)), type = 'l', main = 'Before MSC')
mdaplot(cbind(wavelength, t(cspectra)), type = 'l', main = 'After MSC')
```

**Value**

list with two fields - preprocessed spectra and calculated mean spectrum

**prep.norm**

*Normalization*

**Description**

Normalizes signals (rows of data matrix) to unit area or unit length

**Usage**

```r
prep.norm(data, type = "area")
```
Arguments

data a matrix with data values
type type of normalization 'area' or 'length'

Value

data matrix with normalized values

---

prep.savgol  *Savitzky-Golay filter*

**Description**

Applies Savitzky-Golay filter to the rows of data matrix

**Usage**

```
prep.savgol(data, width = 3, porder = 1, dorder = 0)
```

**Arguments**

data a matrix with data values
width width of the filter window
porder order of polynomial used for smoothing
dorder order of derivative to take (0 - no derivative)

---

prep.snv  *Standard Normal Variate transformation*

**Description**

Applies Standard Normal Variate (SNV) transformation to the rows of data matrix

**Usage**

```
prep.snv(data)
```

**Arguments**

data a matrix with data values
Details

SNV is a simple preprocessing to remove scatter effects (baseline offset and slope) from spectral data, e.g. NIR spectra.

@examples

```r
### Apply SNV to spectra from simdata
library(mdatools) data(simdata)
spectra = simdata$spectra c wavelength = simdata$wavelength
cspectra = prep.snv(spectra)
par(mfrow = c(2, 1)) mdaplot(cbind(wavelength, t(spectra)), type = 'l', main = 'Before SNV')
mdaplot(cbind(wavelength, t(cspectra)), type = 'l', main = 'After SNV')
```

Value

data matrix with processed values

---

`print.classres` | *Print information about classification result object*

Description

Generic `print` function for classification results. Prints information about major fields of the object.

Usage

```r
## S3 method for class 'classres'
print(x, str = NULL, ...)
```

Arguments

- `x` classification results (object of class `plsdare`, `simcamres`, etc.).
- `str` User specified text (e.g. to be used for particular method, like PLS-DA, etc).
- `...` other arguments
print.ipls  
*Print method for iPLS*

**Description**

Prints information about the iPLS object structure

**Usage**

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

**Arguments**

- `x`  
  a iPLS (object of class ipls)
- `...`  
  other arguments

---

print.ldecomp  
*Print method for linear decomposition*

**Description**

Generic print function for linear decomposition. Prints information about the ldecomp object.

**Usage**

```r
## S3 method for class 'ldecomp'
print(x, str = NULL, ...)
```

**Arguments**

- `x`  
  object of class ldecomp
- `str`  
  user specified text to show as a description of the object
- `...`  
  other arguments
**print.pca**  
*Print method for PCA model object*

**Description**  
Prints information about the object structure

**Usage**  
```r
## S3 method for class 'pca'
print(x, ...)
```

**Arguments**  
- `x`  
a PCA model (object of class `pca`)
- `...`  
other arguments

---

**print.pcares**  
*Print method for PCA results object*

**Description**  
Prints information about the object structure

**Usage**  
```r
## S3 method for class 'pcares'
print(x, ...)
```

**Arguments**  
- `x`  
PCA results (object of class `pcares`)
- `...`  
other arguments
print.pls

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x`: a PLS model (object of class `pls`)
- `...`: other arguments

print.plsda

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x`: a PLS-DA model (object of class `plsda`)
- `...`: other arguments
**print.plsdares**

*Print method for PLS-DA results object*

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x` PLS-DA results (object of class `plsdares`)
- `...` other arguments

---

**print.plsres**

*print method for PLS results object*

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x` PLS results (object of class `plsres`)
- `...` other arguments
print.randtest  
*Print method for randtest object*

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x`: a randomization test results (object of class `randtest`)
- `...`: other arguments

---

print.regcoeffs  
*Print method for regression coefficients class*

**Description**

prints regression coefficient values for given response number and amount of components

**Usage**

```r
## S3 method for class 'regcoeffs'
print(x, ncomp = 1, ny = 1, digits = 3, ...)
```

**Arguments**

- `x`: regression coefficients object (class `regcoeffs`)
- `ncomp`: number of components to return the coefficients for
- `ny`: number of response variable to return the coefficients for
- `digits`: decimal digits round the coefficients to
- `...`: other arguments
print.regres

Description
Prints information about the object structure

Usage
```r
## S3 method for class 'regres'
print(x, ...)
```

Arguments
- `x` regression results (object of class `regres`)
- `...` other arguments

print.simca

Description
Prints information about the object structure

Usage
```r
## S3 method for class 'simca'
print(x, ...)
```

Arguments
- `x` a SIMCA model (object of class `simca`)
- `...` other arguments
print.simcam  \hspace{1cm} Print method for SIMCAM model object

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x`  
  a SIMCAM model (object of class simcam)
- `...`  
  other arguments

print.simcamres  \hspace{1cm} Print method for SIMCAM results object

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x`  
  SIMCAM results (object of class simcamres)
- `...`  
  other arguments
print.simcares

Print method for SIMCA results object

Description
Prints information about the object structure

Usage
### S3 method for class 'simcares'
print(x, ...)

Arguments
x SIMCA results (object of class simcares)

... other arguments

randtest
Randomization test for PLS regression

Description
randtest is used to carry out randomization/permutation test for a PLS regression model

Usage
randtest(x, y, ncomp = 15, center = T, scale = F, nperm = 1000,
sig.level = 0.05, silent = TRUE, exclcols = NULL,
exclrows = NULL)

Arguments
x matrix with predictors.
y vector or one-column matrix with response.
ncomp maximum number of components to test.
center logical, center or not predictors and response values.
scale logical, scale (standardize) or not predictors and response values.
nperm number of permutations.
sig.level significance level.
silent logical, show or not test progress.
exclcols columns of x to be excluded from calculations (numbers, names or vector with logical values)
exclrows rows to be excluded from calculations (numbers, names or vector with logical values)
Details

The class implements a method for selection of optimal number of components in PLS1 regression based on the randomization test [1]. The basic idea is that for each component from 1 to \( n_{\text{comp}} \) a statistic \( T \), which is a covariance between t-score (X score, derived from a PLS model) and the reference Y values, is calculated. By repeating this for randomly permuted Y-values a distribution of the statistic is obtained. A parameter \( \alpha \) is computed to show how often the statistic \( T \), calculated for permuted Y-values, is the same or higher than the same statistic, calculated for original data without permutations.

If a component is important, then the covariance for unpermuted data should be larger than the covariance for permuted data and therefore the value for \( \alpha \) will be quite small (there is still a small chance to get similar covariance). This makes \( \alpha \) very similar to p-value in a statistical test.

The randtest procedure calculates alpha for each component, the values can be observed using summary or plot functions. There are also several function, allowing e.g. to show distribution of statistics and the critical value for each component.

Value

Returns an object of randtest class with following fields:

- \( n_{\text{perm}} \) number of permutations used for the test.
- \( \text{stat} \) statistic values calculated for each component.
- \( \alpha \) alpha values calculated for each component.
- \( \text{statperm} \) matrix with statistic values for each permutation.
- \( \text{corrperm} \) matrix with correlation between predicted and reference y-values for each permutation.
- \( n_{\text{comp}.\text{selected}} \) suggested number of components.

References


See Also

Methods for randtest objects:

- \text{print.randtest} \hspace{1cm} \text{prints information about a randtest object.}
- \text{summary.randtest} \hspace{1cm} \text{shows summary statistics for the test.}
- \text{plot.randtest} \hspace{1cm} \text{shows bar plot for alpha values.}
- \text{plotHist.randtest} \hspace{1cm} \text{shows distribution of statistic plot.}
- \text{plotCorr.randtest} \hspace{1cm} \text{shows determination coefficient plot.}

Examples

### Examples of using the test
## Get the spectral data from Simdata set and apply SNV transformation

```r
data(simdata)
y = simdata$conc.c[, 3]
x = simdata$spectra.c
x = prep.snv(x)
```

## Run the test and show summary

```r
# (normally use higher nperm values > 1000)
R = randtest(x, y, ncomp = 4, nperm = 200, silent = FALSE)
summary(R)
```

## Show plots

```r
par(mfrow = c(3, 2))
plot(R)
plotHist(R, comp = 3)
plotHist(R, comp = 4)
plotCorr(R, 3)
plotCorr(R, 4)
par(mfrow = c(1, 1))
```

---

### regcoeffs

**Regression coefficients**

#### Description

class for storing and visualisation of regression coefficients for regression models

#### Usage

```r
regcoeffs(coeffs, ci.coeffs = NULL)
```

#### Arguments

- **coeffs**: vector or matrix with regression coefficients
- **ci.coeffs**: array (nobj x ncomp x ny x cv) with regression coefficients for computing confidence intervals (e.g. from jack-knifing)

#### Value

a list (object of regcoeffs class) with fields, including:

- **values**: an array (nvar x ncomp x ny) with regression coefficients
- **ci**: an array (nvar x ncomp x ny) with confidence intervals for coefficients
- **p.values**: an array (nvar x ncomp x ny) with p-values for coefficients
last two fields are available if proper values for calculation of the statistics were provided.

---

**regcoeffs.getStat**  
*Confidence intervals and p-values for regression coefficients*

---

**Description**
- calculates confidence intervals and t-test based p-values for regression coefficients based on jack-knifing procedure

**Usage**
- `regcoeffs.getStat(coeffs.values, ci.coeffs)`

**Arguments**
- `coeffs.values` regression coefficients array for a model
- `ci.coeffs` array with regression coefficients for calculation of confidence intervals

**Value**
- a list with statistics (`$ci` - array with confidence intervals, `$p.values` - array with p-values, `$t.values` - array with t-values)

---

**regres**  
*Regression results*

---

**Description**
- Class for storing and visualisation of regression predictions

**Usage**
- `regres(y.pred, y.ref = NULL, ncomp.selected = 1)`

**Arguments**
- `y.pred` vector or matrix with y predicted values
- `y.ref` vector with reference (measured) y values
- `ncomp.selected` if y.pred calculated for different components, which to use as default
Value

a list (object of `regres` class) with fields, including:

- `y.pred`: a matrix with predicted values
- `y.ref`: a vector with reference (measured) values
- `ncomp.selected`: selected column/number of components for predictions
- `rmse`: root mean squared error for predicted vs measured values
- `slope`: slope for predicted vs measured values
- `r2`: coefficient of determination for predicted vs measured values
- `bias`: bias for predicted vs measured values
- `rpd`: RPD values

---

**regres.bias**

**Prediction bias**

Description

Calculates matrix with bias (average prediction error) for every response and components

Usage

```r
regres.bias(y.ref, y.pred)
```

Arguments

- `y.ref`: vector with reference values
- `y.pred`: matrix with predicted values

---

**regres.r2**

**Determination coefficient**

Description

Calculates matrix with coefficient of determination for every response and components

Usage

```r
regres.r2(y.ref, y.pred)
```

Arguments

- `y.ref`: vector with reference values
- `y.pred`: matrix with predicted values
**regres.rmse**  \[ RMSE \]

**Description**

Calculates matrix with root mean squared error of prediction for every response and components.

**Usage**

```r
regres.rmse(y.ref, y.pred)
```

**Arguments**

- `y.ref`: vector with reference values
- `y.pred`: matrix with predicted values

**regres.slope**  \[ Slope \]

**Description**

Calculates matrix with slope of predicted and measured values for every response and components.

**Usage**

```r
regres.slope(y.ref, y.pred)
```

**Arguments**

- `y.ref`: vector with reference values
- `y.pred`: matrix with predicted values
reslim.chisq  
Calculates critical limits or statistic values for Q-residuals using Chi-squared distribution

Description
The method is based on Chi-squared distribution with DF = 2 * (m(Q)/s(Q)^2

Usage
reslim.chisq(Q, alpha = 0.05, gamma = 0.01, Qlim = NULL, return = "limits")

Arguments
- Q: vector with Q-residuals for selected component
- alpha: significance level for extreme objects
- gamma: significance level for outliers
- Qlim: vector with Q limits for selected number of components (from model)
- return: what to return: 'limits' or 'probability'

reslim.dd  
Statistical limits for Q and T2 residuals using Data Driven approach

Description
Method is based on paper by Pomerantsev, Rodionova (JChem, 2014)

Usage
reslim.dd(Q, T2, type = "ddmoments", alpha = 0.05, gamma = 0.01, Qlim = NULL, T2lim = NULL, return = "limits")

Arguments
- Q: vector with Q-residuals for selected component
- T2: vector with T2-residuals for selected component
- type: which estimator to use: 'moments' or 'robust'
- alpha: significance level for extreme objects
- gamma: significance level for outliers
- Qlim: vector with Q limits for selected number of components (from model)
- T2lim: vector with T2 limits for selected number of components (from model)
- return: what to return: 'limits' or 'probability'
### reslim.hotelling

*Calculates critical limits for T2-residuals using Hotelling T2 distribution*

**Description**

The method is based on \( n \)

**Usage**

```r
reslim.hotelling(ncomp, T2 = NULL, alpha = 0.05, gamma = 0.01, T2lim = NULL, return = "limits")
```

**Arguments**

- `ncomp`: number of components
- `T2`: vector with T2-residuals for selected component
- `alpha`: significance level for extreme objects
- `gamma`: significance level for outliers
- `T2lim`: T2 limits from a PCA model (needed for probabilities)
- `return`: what to return: 'limits' or 'probability'

### reslim.jm

*Calculates critical limits for Q-residuals using classic JM approach*

**Description**

The method is based on

**Usage**

```r
reslim.jm(eigenvals, Q, ncomp, alpha = 0.05, gamma = 0.01, return = "limits")
```

**Arguments**

- `eigenvals`: vector with eigenvalues for all variables
- `Q`: vector with Q-residuals for selected component
- `ncomp`: number of components
- `alpha`: significance level for extreme objects
- `gamma`: significance level for outliers
- `return`: what to return: 'limits' or 'probability'
selectCompNum

Select optimal number of components for a model

Description

Generic function for selecting number of components for multivariate models (e.g. PCA, PLS, ...)

Usage

selectCompNum(model, ncomp = NULL, ...)

Arguments

model a model object
ncomp number of components to select
... other arguments

selectCompNum.pca

Select optimal number of components for PCA model

Description

Allows user to select optimal number of components for PCA model

Usage

## S3 method for class 'pca'
selectCompNum(model, ncomp, ...)

Arguments

model PCA model (object of class pca)
ncomp number of components to select
... other parameters if any

Value

the same model with selected number of components
select CompNum.pls       Select optimal number of components for PLS model

Description

Allows user to select optimal number of components for PLS model

Usage

## S3 method for class 'pls'
selectCompNum(model, ncomp = NULL,
              selcrit = model$ncomp.selcrit, ...)

Arguments

  model       PLS model (object of class pls)
  ncomp       number of components to select
  selcrit     criterion for selecting optimal number of components ('min' for first local minimum of RMSECV and 'wold' for Wold's rule.)
              ... other parameters if any

Details

If number of components is not specified, the cross-validation statistics will be used. It can be either first local minimum of RMSECV ('selcrit='min'') or Wold's rule ('selcrit='wold'') based on ratio of PRESS values and threshold of 0.95.

See examples in help for pls function.

Value

the same model with selected number of components

set ResLimits       Set residual limits for PCA model

Description

Calculates and set critical limits for residuals of PCA model

Usage

setResLimits(obj, ...)

Arguments

  obj       a SIMCA model
  ...      other parameters
setResLimits.pca

**Set statistical limits for Q and T2 residuals for PCA model**

**Description**

Computes statistical limits for Q and T2 residuals for a PCA model and assigns the calculated values as corresponding model properties.

**Usage**

```r
## S3 method for class 'pca'
setResLimits(obj, alpha = obj$alpha, gamma = obj$gamma, 
              ...)  
```

**Arguments**

- `obj` object with PCA model
- `alpha` significance level for detection of extreme objects
- `gamma` significance level for detection of outliers (for data driven approach)
- `...` other arguments

**Details**

If data driven method is used, first two rows of Qlim and T2lim will contain slope and intercept of line defined by the method, otherwise they contain the critical values (first row for extreme values and second for outliers) for each of the residuals.

Third row contains average values and fourth row contains degrees of freedom.

See help for `pca` for more details.

**Value**

Returns a list with two matrices: T2lim and Qlim. Each matrix contains limits for extreme objects and outliers (first two rows), mean residual and degrees of freedom, calculated for each number of components included to the model.
Description

Predictions

Usage

showPredictions(obj, ...)

Arguments

obj a model or result object
...

other arguments

Details

Generic function for showing predicted values for classification or regression model or results

Description

Show predicted class values

Usage

## S3 method for class 'classres'
showPredictions(obj, ncomp = NULL, ...)

Arguments

obj object with classification results (e.g. plsdares or simcamres).
ncomp number of components to show the predictions for (NULL - use selected for a model).
...

other parameters

Details

The function prints a matrix where every column is a class and every row is an data object. The matrix has either -1 (does not belong to the class) or +1 (belongs to the class) values.
**Description**

`simca` is used to make SIMCA (Soft Independent Modelling of Class Analogies) model for one-class classification.

**Usage**

```r
simca(x, classname, ncomp = 15, center = T, scale = F, cv = NULL,
  exclcols = NULL, exclrows = NULL, x.test = NULL, c.test = NULL,
  method = "svd", rand = NULL, lim.type = "jm", alpha = 0.05,
  gamma = 0.01, info = "")
```

**Arguments**

- `x` a numerical matrix with data values.
- `classname` short text (up to 20 symbols) with class name.
- `ncomp` maximum number of components to calculate.
- `center` logical, do mean centering of data or not.
- `scale` logical, do standardization of data or not.
- `cv` number of segments for random cross-validation (1 for full cross-validation).
- `exclcols` columns to be excluded from calculations (numbers, names or vector with logical values).
- `exclrows` rows to be excluded from calculations (numbers, names or vector with logical values).
- `x.test` a numerical matrix with test data.
- `c.test` a vector with classes of test data objects (can be text with names of classes or logical).
- `method` method to compute principal components.
- `rand` vector with parameters for randomized PCA methods (if NULL, conventional PCA is used instead).
- `lim.type` which method to use for calculation of critical limits for residuals (see details).
- `alpha` significance level for calculating critical limits for T2 and Q residuals.
- `gamma` significance level for calculating outlier limits for T2 and Q residuals.
- `info` text with information about the model.

**Details**

SIMCA is in fact PCA model with additional functionality, so `simca` class inherits most of the functionality of `pca` class. It uses critical limits calculated for Q and T2 residuals calculated for PCA model for making classification decision.
Value

Returns an object of simca class with following fields:

- `classname`: a short text with class name.
- `modpower`: a matrix with modelling power of variables.
- `calres`: an object of class `simcares` with classification results for a calibration data.
- `testres`: an object of class `simcares` with classification results for a test data, if it was provided.
- `cvres`: an object of class `simcares` with classification results for cross-validation, if this option was chosen.

Fields, inherited from `pca` class:

- `ncomp`: number of components included to the model.
- `ncomp.selected`: selected (optimal) number of components.
- `loadings`: matrix with loading values (nvar x ncomp).
- `eigenvals`: vector with eigenvalues for all existent components.
- `expvar`: vector with explained variance for each component (in percent).
- `cumexpvar`: vector with cumulative explained variance for each component (in percent).
- `T2lim`: statistical limit for T2 distance.
- `Qlim`: statistical limit for Q residuals.
- `info`: information about the model, provided by user when build the model.

References


See Also

Methods for simca objects:

- `print.simca`: shows information about the object.
- `summary.simca`: shows summary statistics for the model.
- `plot.simca`: makes an overview of SIMCA model with four plots.
- `predict.simca`: applies SIMCA model to a new data.
- `plotModellingPower.simca`: shows plot with modelling power of variables.

Methods, inherited from classmodel class:

- `plotPredictions.classmodel`: shows plot with predicted values.
- `plotSensitivity.classmodel`: shows sensitivity plot.
- `plotSpecificity.classmodel`: shows specificity plot.
plotMisclassified.classmodel shows misclassified ratio plot.

Methods, inherited from pca class:

- selectCompNum.pca: set number of optimal components in the model
- plotScores.pca: shows scores plot.
- plotLoadings.pca: shows loadings plot.
- plotVariance.pca: shows explained variance plot.
- plotCumVariance.pca: shows cumulative explained variance plot.
- plotResiduals.pca: shows Q vs. T2 residuals plot.

Examples

```r
# make a SIMCA model for Iris setosa class with full cross-validation
library(mdatools)
data(iris)
class = iris[, 5]

# take first 20 objects of setosa as calibration set
se = data[1:20, ]

# make SIMCA model and apply to test set
model = simca(se, 'setosa', cv = 1)
model = selectCompNum(model, 1)

# show information, summary and plot overview
print(model)
summary(model)
plot(model)

# show predictions
par(mfrow = c(2, 1))
plotPredictions(model, show.labels = TRUE)
plotPredictions(model, res = 'calres', ncomp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

# show performance, modelling power and residuals for ncomp = 2
par(mfrow = c(2, 2))
plotSensitivity(model)
plotMisclassified(model)
plotModellingPower(model, ncomp = 2, show.labels = TRUE)
plotResiduals(model, ncomp = 2)
par(mfrow = c(1, 1))
```
simca.classify  

SIMCA classification

Description
Make classification based on calculated T2 and Q values and corresponding limits

Usage
simca.classify(model, res)

Arguments
- model: a SIMCA model (object of class simca)
- res: results of projection data to PCA space

Details
This is a service function for SIMCA class, do not use it manually.

Value
vector with predicted class values (c.pred)

simca.crossval  

Cross-validation of a SIMCA model

Description
Does the cross-validation of a SIMCA model

Usage
simca.crossval(model, x, cv, center = T, scale = F)

Arguments
- model: a SIMCA model (object of class simca)
- x: a matrix with x values (predictors from calibration set)
- cv: number of segments (if cv = 1, full cross-validation will be used)
- center: logical, do mean centering or not
- scale: logical, do standardization or not

Value
object of class simcares with results of cross-validation
**simcam**

**SIMCA multiclass classification**

**Description**

simcam is used to combine several one-class SIMCA models for multiclass classification.

**Usage**

```r
simcam(models, info = "")
```

**Arguments**

- `models` list with SIMCA models (simca objects).
- `info` text with information about the object.

**Details**

Besides the possibility for multiclass classification, SIMCAM also provides tools for investigation of relationship among individual models (classes), such as discrimination power of variables, Cooman's plot, model distance, etc.

When create simcam object, the calibration data from all individual SIMCA models is extracted and combined for making predictions and calculate performance of the multi-class model. The results are stored in `calres` field of the model object.

**Value**

Returns an object of simcam class with following fields:

- `models` a list with provided SIMCA models.
- `dispower` an array with discrimination power of variables for each pair of individual models.
- `moddist` a matrix with distance between each each pair of individual models.
- `classnames` vector with names of individual classes.
- `nclasses` number of classes in the object.
- `info` information provided by user when create the object.
- `calres` an object of class `simcamres` with classification results for a calibration data.

**See Also**

Methods for simca objects:

- `print.simcam` shows information about the object.
- `summary.simcam` shows summary statistics for the models.
- `plot.simcam` makes an overview of SIMCAM model with two plots.
- `predict.simcam` applies SIMCAM model to a new data.
plotModelDistance.simcam shows plot with distance between individual models.
plotDiscriminationPower.simcam shows plot with discrimination power.
plotModellingPower.simcam shows plot with modelling power for individual model.
plotCooman.simcam shows Cooman’s plot for calibration data.
plotResiduals.simcam shows plot with Q vs. T2 residuals for calibration data.

Methods, inherited from classmodel class:

plotPredictions.classmodel shows plot with predicted values.
plotSensitivity.classmodel shows sensitivity plot.
plotSpecificity.classmodel shows specificity plot.
plotMisclassified.classmodel shows misclassified ratio plot.

Since SIMCAM objects and results are calculated only for optimal number of components, there is no sense to show such plots like sensitivity or specificity vs. number of components. However they are available as for any other classification model.

Examples

```r
## make a multiclass SIMCA model for Iris data
library(mdatools)

# split data
caldata = iris[seq(1, nrow(iris), 2), 1:4]
se = caldata[1:25,]
ve = caldata[26:50,]
vi = caldata[51:75,]

testdata = iris[seq(2, nrow(iris), 2), 1:4]
testdata.cref = iris[seq(2, nrow(iris), 2), 6]

# create individual models
semodel = simca(se, classname = 'setosa')
semodel = selectCompNum(semodel, 1)

vimodel = simca(vi, classname = 'virginica')
vimodel = selectCompNum(vimodel, 1)

vemodel = simca(ve, classname = 'versicolor')
vemodel = selectCompNum(vemodel, 1)

# combine models into SIMCAM objects, show statistics and plots
model = simcam(list(semodel, vimodel, vemodel), info = 'Iris data')
summary(model)
plot(model)

# show predictions and residuals for calibration data
par(mfrow = c(2, 2))
plotPredictions(model)
```
simcam.getPerformanceStatistics

Performance statistics for SIMCAM model

Description
Calculates discrimination power and distance between models for SIMCAM model.

Usage
simcam.getPerformanceStatistics(model)

Arguments
model SIMCAM model (object of class simcam)

simcamres Results of SIMCA multiclass classification

Description
simcamres is used to store results for SIMCA multiclass classification.

Usage
simcamres(cres, pred.res)
Arguments

cres results of classification (class classres).
pred.res prediction results from each model (as pcares)

Details

Class simcamres inherits all properties and methods of class classres, plus store values necessary to visualise prediction decisions (e.g. Cooman’s plot or Residuals plot).

In contrast to simcares here only values for optimal (selected) number of components in each individual SIMCA models are presented.

There is no need to create a simcamres object manually, it is created automatically when make a SIMCAM model (see simcam) or apply the model to a new data (see predict.simcam). The object can be used to show summary and plots for the results.

Value

Returns an object (list) of class simcamres with the same fields as classres plus extra fields for Q and T2 values and limits:

c.pred predicted class values.
c.ref reference (true) class values if provided.
T2 matrix with T2 values for each object and class.
Q matrix with Q values for each object and class.
T2lim vector with T2 statistical limits for each class.
Qlim vector with Q statistical limits for each class.

The following fields are available only if reference values were provided.

tp number of true positives.
fp number of false positives.
fn number of false negatives.
specificity specificity of predictions.
sensitivity sensitivity of predictions.

See Also

Methods for simcamres objects:

print.simcamres shows information about the object.
summary.simcamres shows statistics for results of classification.
plotResiduals.simcamres makes Q vs. T2 residuals plot.
plotCooman.simcamres makes Cooman’s plot.

Methods, inherited from classres class:
showPredictions.classes show table with predicted values.
plotPredictions.classes makes plot with predicted values.

Check also simcam.

Examples

```r
## make a multiclass SIMCA model for Iris data and apply to test set
library(mdatools)

# split data
caldata = iris[seq(1, nrow(iris), 2), 1:4]
se = caldata[1:25,]
ve = caldata[26:50,]
vi = caldata[51:75,]

testdata = iris[seq(2, nrow(iris), 2), 1:4]
testdata.cref = iris[seq(2, nrow(iris), 2), 5]

# create individual models
semodel = simca(se, classname = 'setosa')
semodel = selectCompNum(semodel, 1)

vimodel = simca(vi, classname = 'virginica')
vimodel = selectCompNum(vimodel, 1)

vemodel = simca(ve, classname = 'versicolor')
vemodel = selectCompNum(vemodel, 1)

# combine models into SIMCAM object, show statistics
model = simcam(list(semodel, vimodel, vemodel), info = 'Iris data')
res = predict(model, testdata, testdata.cref)
summary(res)

# show predicted values
showPredictions(res)

# plot predictions
par(mfrow = c(2, 2))
plotPredictions(res)
plotPredictions(res, nc = 1)
plotPredictions(res, nc = c(1, 2))
plotPredictions(res, show.labels = TRUE)
par(mfrow = c(1, 1))

# show residuals and Cooman's plot
par(mfrow = c(2, 2))
plotCooman(res)
plotCooman(res, nc = c(1, 3))
plotResiduals(res)
```
plotResiduals(res, nc = 3)
par(mfrow = c(1, 1))

Results of SIMCA one-class classification
@description simcares is used to store results for SIMCA one-class classification.

simcares

Description

Results of SIMCA one-class classification
@description simcares is used to store results for SIMCA one-class classification.

Usage

simcares(pres, cres)

Arguments

pres results of PCA decomposition of data (class pcares).
cres results of classification (class classres).

Details

Class simcares inherits all properties and methods of class pcares, and has additional properties
and functions for representing of classification results, inherited from class classres.
There is no need to create a simcares object manually, it is created automatically when build a
SIMCA model (see simca) or apply the model to a new data (see predict.simca). The object can
be used to show summary and plots for the results.

Value

Returns an object (list) of class simcares with the same fields as pcares plus extra fields, inherited
from classres:
c.pred predicted class values (+1 or -1).
c.ref reference (true) class values if provided.
The following fields are available only if reference values were provided.
tp number of true positives.
fp number of false positives.
fn number of false negatives.
specificity specificity of predictions.
sensitivity sensitivity of predictions.

See Also

Methods for simcares objects:
print.simcares shows information about the object.
summary.simcares shows statistics for results of classification.

Methods, inherited from `classres` class:

- `showPredictions.classres` show table with predicted values.
- `plotPredictions.classres` makes plot with predicted values.
- `plotSensitivity.classres` makes plot with sensitivity vs. components values.
- `plotSpecificity.classres` makes plot with specificity vs. components values.
- `plotPerformance.classres` makes plot with both specificity and sensitivity values.

Methods, inherited from `ldecomp` class:

- `plotResiduals.ldecomp` makes Q2 vs. T2 residuals plot.
- `plotScores.ldecomp` makes scores plot.
- `plotVariance.ldecomp` makes explained variance plot.
- `plotCumVariance.ldecomp` makes cumulative explained variance plot.

Check also `simca` and `pcares`.

Examples

```r
## make a SIMCA model for Iris setosa class and show results for calibration set
library(mdatools)

data = iris[, 1:4]
class = iris[, 5]

# take first 30 objects of setosa as calibration set
se = data[1:30, ]

# make SIMCA model and apply to test set
model = simca(se, 'Se')
model = selectCompNum(model, 1)

# show information and summary
print(model$calres)
summary(model$calres)

# show plots
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
plotPredictions(model$calres, show.labels = TRUE)
plotResiduals(model$calres, show.labels = TRUE)
plotPerformance(model$calres, show.labels = TRUE, legend.position = 'bottomright')
layout(1, 1, 1)

# show predictions table
```
Simdata contains training and test set with spectra and concentration values of polyaromatic hydrocarbons mixings.

Usage

data(simdata)

Format

The data is a list with following fields:

- `$spectra.c` - a matrix (100x150) with spectral values for the training set.
- `$spectra.t` - a matrix (100x150) with spectral values for the test set.
- `$conc.c` - a matrix (100x3) with concentration of components for the training set.
- `$conc.t` - a matrix (100x3) with concentration of components for the test set.
- `$wavelength` - a vector with spectra wavelength in nm.

Details

This is a simulated data containing UV/Vis spectra of three component (polyaromatic hydrocarbons) mixings - C1, C2 and C3. The spectral range is between 210 and 360 nm. The spectra were simulated as a linear combination of pure component spectra plus 5% of random noise. The concentration range is (in moles): C1 [0, 1], C2 [0, 0.5], C3 [0, 0.1].

There are 100 mixings in a training set and 50 mixings in a test set. The data can be used for multivariate regression examples.

Summary statistics about classification result object

Usage

## S3 method for class 'classres'
summary(object, ncomp = NULL, nc = NULL, ...)

Summary statistics about classification result object

Usage

## S3 method for class 'classres'
summary(object, ncomp = NULL, nc = NULL, ...)

Show predictions for the model:

showPredictions(model$calres)
**summary.ipls**  

**Arguments**

- **object**
  - classification results (object of class `plsdares`, `simcamres`, etc.).

- **ncomp**
  - which number of components to make the plot for (can be one value for all classes or vector with separate values for each, if NULL - model selected number will be used).

- **nc**
  - if there are several classes, which class to make the plot for (NULL - summary for all classes).

- **...**
  - other arguments

---

**Summary for iPLS results**

**Description**

Shows statistics and algorithm parameters for iPLS results.

**Usage**

```r
## S3 method for class 'ipls'
summary(object, glob.ncomp = NULL, ...)
```

**Arguments**

- **object**
  - a iPLS (object of class `ipls`)

- **glob.ncomp**
  - number of components for global PLS model with all intervals

- **...**
  - other arguments

**Details**

The method shows information on the algorithm parameters as well as a table with selected or excluded interval. The table has the following columns: `step` showing on which iteration an interval was selected or excluded, `start` and `end` show variable indices for the interval, `nComp` is a number of components used in a model, `RMSE` is RMSECV for the model and `R2` is coefficient of determination for the same model.
### summary.ldecomp

**Summary statistics for linear decomposition**

**Description**

Generic summary function for linear decomposition. Prints statistic about the decomposition.

**Usage**

```r
## S3 method for class 'ldecomp'
summary(object, str = NULL, ...)
```

**Arguments**

- `object`: object of class `ldecomp`
- `str`: user specified text to show as a description of the object
- `...`: other arguments

### summary.pca

**Summary method for PCA model object**

**Description**

Shows some statistics (explained variance, eigenvalues) for the model.

**Usage**

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

**Arguments**

- `object`: a PCA model (object of class `pca`)
- `...`: other arguments
summary.pcares  

Summary method for PCA results object

Description

Shows some statistics (explained variance, eigenvalues) about the results.

Usage

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

Arguments

- `object`: PCA results (object of class `pcares`)
- `...`: other arguments

summary.pls  

Summary method for PLS model object

Description

Shows performance statistics for the model.

Usage

```r
## S3 method for class 'pls'
summary(object, ncomp = NULL, ny = NULL, ...)
```

Arguments

- `object`: a PLS model (object of class `pls`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `ny`: which y variable to show the summary for (if NULL, will be shown for all)
- `...`: other arguments
summary.plsda  

Summary method for PLS-DA model object

Description

Shows some statistics for the model.

Usage

```r
## S3 method for class 'plsda'
summary(object, ncomp = NULL, nc = NULL, ...)
```

Arguments

- **object**: a PLS-DA model (object of class plsda)
- **ncomp**: how many components to use (if NULL - user selected optimal value will be used)
- **nc**: which class to show the summary for (if NULL, will be shown for all)
- **...**: other arguments

summary.plsdares  

Summary method for PLS-DA results object

Description

Shows performance statistics for the results.

Usage

```r
## S3 method for class 'plsdares'
summary(object, nc = NULL, ...)
```

Arguments

- **object**: PLS-DA results (object of class plsdares)
- **nc**: which class to show the summary for (if NULL, will be shown for all)
- **...**: other arguments
**summary.plsres**  
*summary method for PLS results object*

### Description

Shows performance statistics for the results.

### Usage

```r
## S3 method for class 'plsres'
summary(object, ny = NULL, ncomp = NULL, ...)
```

### Arguments

- **object**  
  PLS results (object of class `plsres`)
- **ny**  
  for which response variable show the summary for
- **ncomp**  
  how many components to use (if NULL - user selected optimal value will be used)
- **...**  
  other arguments

---

**summary.randtest**  
*Summary method for randtest object*

### Description

Shows summary for randomization test results.

### Usage

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

### Arguments

- **object**  
  randomization test results (object of class `randtest`)
- **...**  
  other arguments
### summary.regcoefs

**Summary method for regcoefs object**

**Description**

Shows estimated coefficients and statistics (if available).

**Usage**

```r
## S3 method for class 'regcoefs'
summary(object, ncomp = 1, ny = 1, alpha = 0.05, ...)
```

**Arguments**

- `object`: object of class `regcoefs`
- `ncomp`: how many components to use
- `ny`: which y variable to show the summary for
- `alpha`: significance level for confidence interval (if statistics available)
- `...`: other arguments

**Details**

Statistics are shown if Jack-Knifing was used when model is calibrated.

### summary.regres

**summary method for regression results object**

**Description**

Shows performance statistics for the regression results.

**Usage**

```r
## S3 method for class 'regres'
summary(object, ncomp = NULL, ny = NULL, ...)
```

**Arguments**

- `object`: regression results (object of class `regres`)
- `ncomp`: model complexity to show the summary for
- `ny`: for which response variable show the summary for
- `...`: other arguments
**summary.simca**  
*Summary method for SIMCA model object*

**Description**

Shows performance statistics for the model.

**Usage**

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

**Arguments**

- `object`  
  a SIMCA model (object of class simca)
- `...`  
  other arguments

---

**summary.simcam**  
*Summary method for SIMCAM model object*

**Description**

Shows performance statistics for the model.

**Usage**

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

**Arguments**

- `object`  
  a SIMCAM model (object of class simcam)
- `...`  
  other arguments
Summary method for SIMCAM results object

Description
Shows performance statistics for the results.

Usage
```r
summary(object, ...)  
```

Arguments
- `object`: SIMCAM results (object of class `simcamres`)
- `...`: other arguments

Summary method for SIMCA results object

Description
Shows performance statistics for the results.

Usage
```r
summary(object, ...)  
```

Arguments
- `object`: SIMCA results (object of class `simcares`)
- `...`: other arguments
Index

*Topic datasets
pellets, 60
gsdata, 182

as.matrix.classes, 7
as.matrix.ldecomp, 8
as.matrix.plsdares, 8
as.matrix.plsres, 9
as.matrix.regcoeffs, 9
as.matrix.regres, 10

bars, 10

classify.plsda, 11
classes, 11, 140, 178, 180, 181
crossval, 12
crossval.str, 13

erfinv, 13
erorbars, 14

getB, 14
getCalibrationData, 15
getCalibrationData.pca, 15
getCalibrationData.simcam, 16
getClassificationPerformance, 16
getConfusionMatrix, 18
getConfusionMatrix.classes, 18
getMainTitle, 19
getProbabilities, 19
getProbabilities.simca, 20
getRegcoeffs, 20
getRegcoeffs.pls, 21, 129
getSelectedComponents, 22
getSelectedComponents.classes, 22
getSelectivityRatio, 23
getSelectivityRatio.pls, 23, 129
getVIPScores, 24
getVIPScores.pls, 24, 129

imshow, 25
gls, 25, 50, 104
ipls.backward, 27
ipls.forward, 28

ldecomp, 28, 58, 59, 181
ldecomp.getDistances, 29
ldecomp.getVariances, 30
ldecomp.plotLimits, 30

mda.cbind, 31
mda.data2im, 31
mda.df2mat, 32, 40
mda.exclcols, 32
mda exclrows, 33
mda.getattr, 33
mda.getexclind, 34
mda.im2data, 34
mda.inclcols, 35
mda.inclrows, 35
mda.rbind, 36
mda.setattr, 36
mda.setimbg, 37
mda.show, 37
mda.subset, 38, 40
mda.t, 38

mdaplot, 39, 49, 50, 75, 82, 86, 88, 92, 94, 96, 102, 106, 108, 109, 143

mdaplot.areColors, 41
mdaplot.formatValues, 42
mdaplot.axesLim, 42
mdaplot.getColors, 43
mdaplot.plotAxes, 44
mdaplot.showColorbar, 44
mdaplot.showGrid, 45
mdaplot.showLabels, 45
mdaplot.showLegend, 46
mdaplot.showLines, 46
mdaplot.showRegressionLine, 47
mdaplotg, 41, 43, 47, 50, 81, 86, 88, 106, 107
mdatools, 49
pca, 29, 50, 50, 58, 59, 63, 76, 80, 95, 103, 110, 127, 169, 171–173
pca.cal, 54
pca.crossval, 55
pca.mvreplace, 53, 55
pca.nipals, 56
pca.run, 57
pca.svd, 58
pcares, 29, 50, 52, 53, 58, 180, 181
pellets, 60
people, 61
pinv, 62
plot.classres, 62
plot.ipls, 63
plot.pca, 63
plot.pcareres, 64
plot.pls, 64, 128
plot.plsda, 65
plot.plsdares, 65
plot.plsres, 66, 143
plot.randtest, 67, 160
plot.regcoeffs, 67
plot.regres, 68
plot.simca, 69
plot.simcam, 69
plot.simcamres, 70
plotBiplot, 70
plotBiplot.pca, 71
plotCooman, 71
plotCooman.simcam, 72, 176
plotCooman.simcamres, 72, 178
plotCorr, 73
plotCorr.randtest, 74, 160
plotCumVariance, 74
plotCumVariance.ldecomp, 59, 75, 181
plotCumVariance.pca, 53, 75, 173
plotDiscriminationPower, 76
plotDiscriminationPower.simcam, 77, 176
plotExtreme, 77
plotExtreme.simca, 78
plotHist, 78
plotHist.randtest, 79, 160
plotLoadings, 79
plotLoadings.pca, 53, 80, 173
plotMisclassified, 81
plotMisclassified.classmodel, 81, 137, 173, 176
plotMisclassified.classres, 12, 82
plotModelDistance, 82
plotModelDistance.simcam, 83, 176
plotModellingPower, 83
plotModellingPower.simca, 84, 172
plotModellingPower.simcam, 84, 176
plotPerformance, 85
plotPerformance.classmodel, 85
plotPerformance.classres, 12, 86, 140, 181
plotPredictions, 87
plotPredictions.classmodel, 87, 137, 172, 176
plotPredictions.classres, 12, 88, 140, 179, 181
plotPredictions.pls, 89, 128
plotPredictions.plsres, 90, 140, 143
plotPredictions.regres, 90, 90
plotProbabilities, 91
plotProbabilities.classres, 92
plotRegcoeffs, 92
plotRegcoeffs.pls, 93, 128
plotResiduals, 93
plotResiduals.ldecomp, 59, 94, 181
plotResiduals.pca, 53, 94, 173
plotResiduals.pcareres, 95
plotResiduals.simcam, 96, 176
plotResiduals.simcamres, 97, 178
plotResiduals.simcares, 97
plotRMSE, 98
plotRMSE.ipls, 99, 104
plotRMSE.pls, 100, 128
plotRMSE.regres, 100, 140, 143
plotScores, 101
plotScores.ldecomp, 59, 101, 181
plotScores.pca, 53, 102, 173
plotSelection, 103
plotSelection.ipls, 63, 99, 103
plotSelectivityRatio, 104
plotSelectivityRatio.pls, 104, 128
plotSensitivity, 105
plotSensitivity.classmodel, 106, 137, 172, 176
plotSensitivity.classres, 12, 106, 140, 181
plotSpecificity, 107
plotSpecificity.classmodel, 107, 137, 172, 176
showPredictions, 170
showPredictions.classes, \[12, 140, 170, 179, 181\]
simca, \[12, 50, 52, 81, 88, 106, 107, 147, 171, 180, 181\]
simca.classify, 174
simca.crossval, 174
simcam, \[16, 50, 69, 72, 77, 81, 83, 85, 88, 96, 106, 107, 148, 175, 178, 179\]
simcam.getPerformanceStatistics, 177
simcamres, \[50, 70, 73, 82, 87, 89, 97, 107, 108, 175, 177\]
simcares, \[12, 50, 98, 172, 180\]
simdata, 182
summary.classres, 182
summary.ipls, \[99, 104, 183\]
summary.idecomp, 184
summary.pca, 184
summary.pcares, 185
summary.pls, \[128, 185\]
summary.plsda, 186
summary.plsdares, 186
summary.plsres, \[143, 187\]
summary.randtest, \[160, 187\]
summary.regcoeffs, 188
summary.regres, 188
summary.simca, 189
summary.simcam, 189
summary.simcamres, \[190\]
summary.simcares, \[190\]