Package ‘mdatools’

September 13, 2021

Title  Multivariate Data Analysis for Chemometrics
Version 0.12.0
Date 2021-09-12
Author Sergey Kucheryavskiy (<https://orcid.org/0000-0002-3145-7244>)
Maintainer Sergey Kucheryavskiy <svkucheryavski@gmail.com>
Description 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, Matrix
RoxygenNote 7.1.1
Suggests testthat
NeedsCompilation no
Depends R (>= 3.5.0)
Repository CRAN
Date/Publication 2021-09-13 07:40:02 UTC

R topics documented:

  as.matrix.classres .................................................. 10
  as.matrix.ldecomp .................................................. 10
  as.matrix.plsdares ............................................... 11
  as.matrix.plsres .................................................. 11
  as.matrix.regcoeffs ............................................... 12
  as.matrix.regres .................................................. 12
  as.matrix.simcamres .............................................. 13
  as.matrix.simcares .............................................. 13
  capitalize .......................................................... 14
  carbs .................................................................. 14
  categorize .......................................................... 15
R topics documented:

categorize.pca .......................................................... 15
categorize.pls ........................................................... 16
chisq.crit ................................................................. 17
chisq.prob ................................................................. 17
classify.plsda ............................................................ 18
classify.simca ............................................................ 18
classmodel.processRefValues ......................................... 19
classes ................................................................. 19
classes.getPerformance ................................................ 20
confint.regcoeffs ......................................................... 21
constraint ............................................................... 22
constraintAngle .......................................................... 22
constraintClosure ........................................................ 23
constraintNonNegativity ................................................ 23
constraintNorm ............................................................ 24
constraints.list .......................................................... 24
constraintUnimod ........................................................ 24
crossval ................................................................. 25
crossval.getParams ....................................................... 25
crossval.regmodel ....................................................... 26
crossval.simca ............................................................ 26
crossval.str .............................................................. 27
dd.crit ................................................................. 27
ddmoments.param ........................................................ 28
ddrobust.param .......................................................... 28
ellipse ................................................................. 29
employ.constraint ....................................................... 29
employ.prep .............................................................. 30
eye ................................................................. 30
fprintf ................................................................. 31
getCalibrationData ...................................................... 31
getCalibrationData.pca ................................................ 32
getCalibrationData.simcam ........................................... 32
getConfidenceEllipse ................................................... 33
getConfusionMatrix ..................................................... 33
getConfusionMatrix.classes ........................................... 34
getConvexHull ............................................................ 34
getDataLabels ............................................................ 35
getImplementedConstraints .......................................... 35
getImplementedPrepMethods ......................................... 35
getLabelsAsIndices ...................................................... 36
getLabelsAsValues ...................................................... 36
getMainTitle ............................................................. 36
getPlotColors ............................................................ 37
getProbabilities ........................................................ 37
getProbabilities.pca ................................................... 38
getProbabilities.simca ................................................ 38
getPureVariables ......................................................... 39
R topics documented:

getR ................................................................. 39
getRegcoeffs ..................................................... 40
getRegcoeffs.remodel ......................................... 40
getRes ............................................................. 41
getSelectedComponents ....................................... 42
getSelectivityRatio ............................................. 42
getSelectivityRatio.pls ....................................... 43
getVariance.mcr ................................................ 43
getVIPScores .................................................... 44
getVIPScores.pls ............................................... 44
hotelling.crit .................................................... 45
hotelling.prob ................................................... 45
imshow ............................................................ 46
ipls ................................................................. 46
ipls.backward ................................................... 49
ipls.forward ..................................................... 49
jm.crit ............................................................. 50
jm.prob ............................................................ 50
ldecomp .......................................................... 51
ldecomp.getDistances ......................................... 52
ldecomp.getLimitsCoordinates ............................... 52
ldecomp.getLimParams ........................................ 53
ldecomp.getQLimits ............................................ 53
ldecomp.getT2Limits .......................................... 54
ldecomp.getVariances ........................................ 54
ldecomp.plotResiduals ........................................ 55
mcr ................................................................. 56
mcrals ............................................................ 57
mcrals.cal ......................................................... 61
mcralsfcnlnls .................................................... 63
mcralsnls ........................................................ 63
mcralsols ........................................................ 64
mcrpure ........................................................... 65
mda.cbind ........................................................ 68
mda.data2im ...................................................... 68
mda.dl2mat ........................................................ 69
mda.exclcols ...................................................... 69
mda.exclrows ..................................................... 70
mda.getattr ......................................................... 70
mda.getexclind .................................................. 71
mda.im2data ....................................................... 71
mda.inclcols ....................................................... 72
mda.inclrows ....................................................... 72
mda.purge ........................................................ 73
mda.purgeCols ................................................... 73
mda.purgeRows ................................................... 73
mda.rbind ........................................................ 74
mda.setattr ......................................................... 74
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<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.getColors</td>
</tr>
<tr>
<td>mdaplot.getXAxisLim</td>
</tr>
<tr>
<td>mdaplot.getYAxisLim</td>
</tr>
<tr>
<td>mdaplot.getXTickLabels</td>
</tr>
<tr>
<td>mdaplot.getXTicks</td>
</tr>
<tr>
<td>mdaplot.getYTickLabels</td>
</tr>
<tr>
<td>mdaplot.getYTicks</td>
</tr>
<tr>
<td>mdaplot.plotAxes</td>
</tr>
<tr>
<td>mdaplot.prepareColors</td>
</tr>
<tr>
<td>mdaplot.showColorbar</td>
</tr>
<tr>
<td>mdaplot.showLines</td>
</tr>
<tr>
<td>mdaplotg</td>
</tr>
<tr>
<td>mdaplotg.getLegend</td>
</tr>
<tr>
<td>mdaplotg.getXLim</td>
</tr>
<tr>
<td>mdaplotg.getYLim</td>
</tr>
<tr>
<td>mdaplotg.prepareData</td>
</tr>
<tr>
<td>mdaplotg.processParam</td>
</tr>
<tr>
<td>mdaplotg.showLegend</td>
</tr>
<tr>
<td>mdatools</td>
</tr>
<tr>
<td>pca</td>
</tr>
<tr>
<td>pca.cal</td>
</tr>
<tr>
<td>pca.getB</td>
</tr>
<tr>
<td>pca.mvreplace</td>
</tr>
<tr>
<td>pca.nipals</td>
</tr>
<tr>
<td>pca.run</td>
</tr>
<tr>
<td>pca.svd</td>
</tr>
<tr>
<td>pcares</td>
</tr>
<tr>
<td>pcv</td>
</tr>
<tr>
<td>pellets</td>
</tr>
<tr>
<td>people</td>
</tr>
<tr>
<td>pinv</td>
</tr>
<tr>
<td>plot.classres</td>
</tr>
<tr>
<td>plot.ipls</td>
</tr>
<tr>
<td>plot.mcr</td>
</tr>
<tr>
<td>plot.pca</td>
</tr>
<tr>
<td>plot.pcares</td>
</tr>
<tr>
<td>plot.pls</td>
</tr>
<tr>
<td>plot.plsda</td>
</tr>
<tr>
<td>plot.plsdares</td>
</tr>
<tr>
<td>plot.plsres</td>
</tr>
<tr>
<td>R topics documented</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>plot.randtest</td>
</tr>
<tr>
<td>plot.regcoeffs</td>
</tr>
<tr>
<td>plot.regres</td>
</tr>
<tr>
<td>plot.simca</td>
</tr>
<tr>
<td>plot.simcam</td>
</tr>
<tr>
<td>plot.simcamres</td>
</tr>
<tr>
<td>plotBars</td>
</tr>
<tr>
<td>plotBiplot</td>
</tr>
<tr>
<td>plotBiplot.pca</td>
</tr>
<tr>
<td>plotConfidenceEllipse</td>
</tr>
<tr>
<td>plotContributions</td>
</tr>
<tr>
<td>plotContributions.mcr</td>
</tr>
<tr>
<td>plotConvexHull</td>
</tr>
<tr>
<td>plotCooman</td>
</tr>
<tr>
<td>plotCooman.simcam</td>
</tr>
<tr>
<td>plotCooman.simcamres</td>
</tr>
<tr>
<td>plotCorr</td>
</tr>
<tr>
<td>plotCorr.randtest</td>
</tr>
<tr>
<td>plotCumVariance</td>
</tr>
<tr>
<td>plotCumVariance.ldecomp</td>
</tr>
<tr>
<td>plotCumVariance.mcr</td>
</tr>
<tr>
<td>plotCumVariance.pca</td>
</tr>
<tr>
<td>plotDensity</td>
</tr>
<tr>
<td>plotDiscriminationPower</td>
</tr>
<tr>
<td>plotDiscriminationPower.simcam</td>
</tr>
<tr>
<td>plotDistDoF</td>
</tr>
<tr>
<td>plotErrorbars</td>
</tr>
<tr>
<td>plotExtreme</td>
</tr>
<tr>
<td>plotExtreme.pca</td>
</tr>
<tr>
<td>plotHist</td>
</tr>
<tr>
<td>plotHist.randtest</td>
</tr>
<tr>
<td>plotHotellingEllipse</td>
</tr>
<tr>
<td>plotLines</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.classres</td>
</tr>
<tr>
<td>plotModelDistance</td>
</tr>
<tr>
<td>plotModelDistance.simcam</td>
</tr>
<tr>
<td>plotModellingPower</td>
</tr>
<tr>
<td>plotPerformance</td>
</tr>
<tr>
<td>plotPerformance.classmodel</td>
</tr>
<tr>
<td>plotPerformance.classres</td>
</tr>
<tr>
<td>plotPointsShape</td>
</tr>
<tr>
<td>plotPredictions</td>
</tr>
<tr>
<td>plotPredictions.classmodel</td>
</tr>
<tr>
<td>plotPredictions.classres</td>
</tr>
</tbody>
</table>
plotPredictions.regmodel ................................................................. 146
plotPredictions.regres ................................................................. 147
plotPredictions.simcam ................................................................. 148
plotPredictions.simcamres ......................................................... 149
plotProbabilities ................................................................. 149
plotProbabilities.classes ................................................................. 150
plotPurity ................................................................. 150
plotPurity.mcrpure ................................................................. 151
plotPuritySpectra ................................................................. 151
plotPuritySpectra.mcrpure ................................................................. 152
plotQDoF ................................................................. 153
plotRegcoeffs ................................................................. 153
plotRegcoeffs.regmodel ................................................................. 154
plotRegressionLine ................................................................. 154
plotResiduals ................................................................. 155
plotResiduals.ldecomp ................................................................. 155
plotResiduals.pca ................................................................. 156
plotResiduals.regres ................................................................. 157
plotRMSE ................................................................. 158
plotRMSE.ipls ................................................................. 159
plotRMSE.regmodel ................................................................. 160
plotRMSE.regres ................................................................. 160
plotScores ................................................................. 162
plotScores.ldecomp ................................................................. 163
plotScores.pca ................................................................. 163
plotSelection ................................................................. 164
plotSelection.ipls ................................................................. 165
plotSelectivityRatio ................................................................. 166
plotSelectivityRatio.pls ................................................................. 166
plotSensitivity ................................................................. 167
plotSensitivity.classmodel ................................................................. 167
plotSensitivity.classes ................................................................. 168
plotSeries ................................................................. 168
plotSpecificity ................................................................. 169
plotSpecificity.classmodel ................................................................. 170
plotSpecificity.classes ................................................................. 170
plotSpectra ................................................................. 171
plotSpectra.mcr ................................................................. 171
plotT2DoF ................................................................. 172
plotVariance ................................................................. 172
plotVariance.ldecomp ................................................................. 173
plotVariance.mcr ................................................................. 174
plotVariance.pca ................................................................. 174
plotVariance.pls ................................................................. 175
plotVariance.plsres ................................................................. 176
plotVIPScores ................................................................. 177
plotVIPScores.pls ................................................................. 177
R topics documented:

plotWeights .......................................................... 178
plotWeights.pls ....................................................... 178
plotXCumVariance ..................................................... 179
plotXCumVariance.pls ............................................... 179
plotXCumVariance.plsres ........................................... 180
plotXLoadings .......................................................... 180
plotXLoadings.pls ................................................... 181
plotXResiduals ........................................................ 181
plotXResiduals.pls .................................................. 182
plotXResiduals.plsres ................................................ 183
plotXScores ............................................................ 184
plotXScores.pls ....................................................... 184
plotXScores.plsres .................................................. 185
plotXVariance .......................................................... 186
plotXVariance.pls ................................................... 186
plotXVariance.plsres ................................................ 187
plotXYLoadings ........................................................ 187
plotXYLoadings.pls .................................................. 188
plotXYResiduals ........................................................ 188
plotXYResiduals.pls .................................................. 189
plotXYResiduals.plsres ............................................. 190
plotXYScores ........................................................... 191
plotXYScores.pls ..................................................... 191
plotXYScores.plsres ................................................ 192
plotYCumVariance ...................................................... 193
plotYCumVariance.pls ............................................... 193
plotYCumVariance.plsres .......................................... 194
plotYResiduals ........................................................ 194
plotYResiduals.pls .................................................. 195
plotYResiduals.plsres .............................................. 195
plotYResiduals.regmodel .......................................... 195
plotYVariance ........................................................... 196
plotYVariance.pls .................................................... 196
plotYVariance.plsres ................................................ 197
pls ................................................................. 198
pls.cal ............................................................... 203
pls.getLimitsCoordinates ......................................... 204
pls.getZLimits ........................................................ 204
pls.run ............................................................... 205
pls.simpls ............................................................ 205
plsda ................................................................. 206
plsdares .............................................................. 209
plsres ................................................................. 212
predict.mcrals ........................................................ 215
predict.mcrpure ........................................................ 215
predict.pca ............................................................ 216
predict.pls ............................................................. 216
predict.plsda ........................................................ 217
predict.simca .......................................................... 218
<table>
<thead>
<tr>
<th>R topics documented:</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>predict.simcam</td>
<td>218</td>
</tr>
<tr>
<td>prep</td>
<td>219</td>
</tr>
<tr>
<td>prep.alsbasecorr</td>
<td>220</td>
</tr>
<tr>
<td>prep.autoscale</td>
<td>221</td>
</tr>
<tr>
<td>prep.generic</td>
<td>221</td>
</tr>
<tr>
<td>prep.list</td>
<td>222</td>
</tr>
<tr>
<td>prep.msc</td>
<td>222</td>
</tr>
<tr>
<td>prep.norm</td>
<td>223</td>
</tr>
<tr>
<td>prep.ref2km</td>
<td>223</td>
</tr>
<tr>
<td>prep.savgol</td>
<td>224</td>
</tr>
<tr>
<td>prep.snv</td>
<td>225</td>
</tr>
<tr>
<td>prep.transform</td>
<td>225</td>
</tr>
<tr>
<td>prep.varsel</td>
<td>226</td>
</tr>
<tr>
<td>preparePlotData</td>
<td>227</td>
</tr>
<tr>
<td>prepCalData</td>
<td>227</td>
</tr>
<tr>
<td>print.classres</td>
<td>228</td>
</tr>
<tr>
<td>print.ipls</td>
<td>228</td>
</tr>
<tr>
<td>print.ldecomp</td>
<td>229</td>
</tr>
<tr>
<td>print.mcrals</td>
<td>229</td>
</tr>
<tr>
<td>print.mcrpure</td>
<td>230</td>
</tr>
<tr>
<td>print.pca</td>
<td>230</td>
</tr>
<tr>
<td>print.pcares</td>
<td>231</td>
</tr>
<tr>
<td>print.pls</td>
<td>231</td>
</tr>
<tr>
<td>print.plsda</td>
<td>232</td>
</tr>
<tr>
<td>print.plsdares</td>
<td>232</td>
</tr>
<tr>
<td>print.plsres</td>
<td>233</td>
</tr>
<tr>
<td>print.randtest</td>
<td>233</td>
</tr>
<tr>
<td>print.regcoeffs</td>
<td>234</td>
</tr>
<tr>
<td>print.regmodel</td>
<td>234</td>
</tr>
<tr>
<td>print.regres</td>
<td>235</td>
</tr>
<tr>
<td>print.simca</td>
<td>235</td>
</tr>
<tr>
<td>print.simcam</td>
<td>236</td>
</tr>
<tr>
<td>print.simcamres</td>
<td>236</td>
</tr>
<tr>
<td>print.simcares</td>
<td>237</td>
</tr>
<tr>
<td>randtest</td>
<td>237</td>
</tr>
<tr>
<td>regcoeffs</td>
<td>239</td>
</tr>
<tr>
<td>regcoeffs.getStats</td>
<td>240</td>
</tr>
<tr>
<td>regres</td>
<td>241</td>
</tr>
<tr>
<td>regres.bias</td>
<td>241</td>
</tr>
<tr>
<td>regres.err</td>
<td>242</td>
</tr>
<tr>
<td>regres.r2</td>
<td>242</td>
</tr>
<tr>
<td>regres.rmse</td>
<td>243</td>
</tr>
<tr>
<td>regres.slope</td>
<td>243</td>
</tr>
<tr>
<td>regress.addattrs</td>
<td>243</td>
</tr>
<tr>
<td>remat</td>
<td>244</td>
</tr>
<tr>
<td>rotationMatrixToX1</td>
<td>244</td>
</tr>
<tr>
<td>selectCompNum</td>
<td>245</td>
</tr>
<tr>
<td>selectCompNum.pca</td>
<td>245</td>
</tr>
</tbody>
</table>
as.matrix.classres    as.matrix method for classification results

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

Arguments
x object of class ldecomp
ncomp number of components to get the result for (if NULL will return for each available)
... other arguments
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 = 1, ...)
```

Arguments

- `x`: PLS-DA results (object of class `plsdares`)
- `ncomp`: number of components to calculate the statistics for (if NULL gets for all components)
- `nc`: for which class to calculate the statistics for
- `...`: other arguments

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  

Description

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

Usage

## 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  

Description

Returns a matrix with model performance statistics for regression results

Usage

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

Arguments

- `x`: regression results (object of class regres)
- `ncomp`: model complexity (number of components) to calculate the statistics for (can be a vector)
- `ny`: for which response variable calculate the statistics for
- `...`: other arguments
as.matrix.simcamres  

as.matrix method for SIMCAM results

Description

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

Usage

## S3 method for class 'simcamres'
as.matrix(x, nc = seq_len(x$nclasses), ...)

Arguments

x classification results (object of class plsdares, simcamres, etc.).
nc vector with classes to use.
... other arguments

as.matrix.simcares  

as.matrix method for SIMCA classification results

Description

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

Usage

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

Arguments

x classification results (object of class plsdares, simcamres, etc.).
ncomp model complexity (number of components) to show the parameters for.
... other arguments
Description

Capitalize text or vector with text values

Usage

capitalize(str)

Arguments

str text of vector with text values

---

carbs Raman spectra of carbonhydrates

---

Description

The dataset consists of Raman spectra of fructose, lactose, and ribose as well as spectra of their mixtures.

Usage

data(simdata)

Format

The data is a list (carbs) with the following fields:

$D$ a matrix (21x1401) with spectral values for the mixtures.

$S$ a matrix (1401x3) with spectral values for the pure components.

$C$ a matrix (21x3) with concentration of the pure components.

Details

The dataset consists of Raman spectra of fructose, lactose, and ribose as well as spectra of their mixtures. The original spectra were downloaded from publicly available SPECARB library [1], created by S.B. Engelsen. The spectra were truncated to the range from 200 to 1600 cm⁻¹.
The spectra of mixtures were created by linear combinations of the original spectra:

$D = CS' + E$

Concentrations of the components, $C$, follow a simplex lattice design with four levels. Some noise calculated as a random number uniformly distributed between 0% and 3% of maximum initial...
intensity (E) was added to each spectrum of the dataset, D, individually.

References

1. Engelsen S.B., Database on Raman spectra of carbohydrates. Available at: http://www.models.life.ku.dk/~specarb/specarb.html [visited 31.05.2020]

categorize

Categorize PCA results

Description

Categorize PCA results

Usage

categorize(obj, ...)

Arguments

obj  object with PCA model
...

other parameters

categorize.pca

Categorize PCA results based on orthogonal and score distances.

Description

The method compares score and orthogonal distances of PCA results from res with critical limits computed for the PCA model and categorizes the corresponding objects as "regular", "extreme" or "outlier".

Usage

## S3 method for class 'pca'
categorize(obj, res = obj$res$cal, ncomp = obj$ncomp.selected, ...)

Arguments

obj  object with PCA model
res  object with PCA results
ncomp  number of components to use for the categorization
...

other parameters
Details
The method does not categorize hidden values if any.

Value
vector (factor) with results of categorization.

categorize.pls  Categorize data rows based on PLS results and critical limits for total distance.

Description
The method uses full distance for decomposition of X-data and squared Y-residuals of PLS results from res with critical limits computed for the PLS model and categorizes the corresponding objects as "regular", "extreme" or "outlier".

Usage
## S3 method for class 'pls'
categorize(obj, res = obj$res$cal, ncomp = obj$ncomp.selected, ...)

Arguments
obj object with PCA model
res object with PCA results
ncomp number of components to use for the categorization
... other parameters

Details
The method does not categorize hidden values if any. It is based on the approach described in [1] and works only if data driven approach is used for computing critical limits.

Value
vector (factor) with results of categorization.

References
chisq.crit

Calculates critical limits for distance values using Chi-square distribution

**Description**

The method is based on Chi-squared distribution with DF = 2 * (m(u)/s(u))^2

**Usage**

chisq.crit(param, alpha = 0.05, gamma = 0.01)

**Arguments**

- **param**: matrix with distribution parameters
- **alpha**: significance level for extreme objects
- **gamma**: significance level for outliers

chisq.prob

Calculate probabilities for distance values using Chi-square distribution

**Description**

Calculate probabilities for distance values using Chi-square distribution

**Usage**

chisq.prob(u, param)

**Arguments**

- **u**: vector with distances
- **param**: vector with distribution parameters
**classify.plsda**  
*PLS-DA classification*

**Description**
Converts PLS predictions of y values to predictions of classes

**Usage**
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*)

---

**classify.simca**  
*SIMCA classification*

**Description**
Make classification based on calculated T2 and Q values and corresponding limits

**Usage**
classify.simca(obj, pca.res, c.ref = NULL)

**Arguments**
- **obj**: a SIMCA model (object of class *simca*)
- **pca.res**: results of projection data to PCA space
- **c.ref**: vector with class reference values

**Details**
This is a service function for SIMCA class, do not use it manually.

**Value**
vector with predicted class values (**c.pred**)
classmodel.processRefValues

Check reference class values and convert it to a factor if necessary

Description

Check reference class values and convert it to a factor if necessary

Usage

classmodel.processRefValues(c.ref, classnames = NULL)

Arguments

c.ref class reference values provided by user
classnames text with class name in case of logical reference values

classres

Results of classification

Description

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

Usage

classres(c.pred, c.ref = NULL, p.pred = NULL, ncomp.selected = 1)

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 from mdatools, 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.
tn: number of true negatives.
fp: number of false positives.
fn: number of false negatives.
specificity: specificity of predictions.
sensitivity: sensitivity of predictions.
misclassified: ratio of misclassified objects.

See Also

Methods classres class:

- showPredictions.classres: shows table with predicted values.
- plotPredictions.classres: makes plot with predicted values.
- plotSensitivity.classres: makes sn plot.
- plotSpecificity.classres: makes specificity plot.
- plotMisclassified.classres: makes ms ratio plot.
- plotPerformance.classres: makes plot with misclassified ratio, specificity and sensitivity values.

---

classres.getPerformance

*Calculation of classification performance parameters*

**Description**

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

**Usage**

```r
classres.getPerformance(c.ref, c.pred)
```

**Arguments**

- `c.ref`: reference class values for objects (vector with numeric or text values).
- `c.pred`: predicted class values for objects (array nobj x ncomponents x nclasses).
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:

- $fn$: number of false negatives (nclasses x ncomponents)
- $fp$: number of false positives (nclasses x ncomponents)
- $tp$: number of true positives (nclasses x ncomponents)
- $sensitivity$: sn values (nclasses x ncomponents)
- $specificity$: specificity values (nclasses x ncomponents)
- $specificity$: ms ratio values (nclasses x ncomponents)

Description

returns matrix with confidence intervals for regression coefficients for given response number and number of components.

Usage

```r
## S3 method for class 'regcoeffs'
confint(object, parm = NULL, level = 0.95, ncomp = 1, ny = 1, ...)
```

Arguments

- `object`: regression coefficients object (class `regcoeffs`)
- `parm`: not used, needed for compatibility with general method
- `level`: confidence level
- `ncomp`: number of components (one value)
- `ny`: index of response variable (one value)
- `...`: other arguments
constraint

Class for MCR-ALS constraint

Description

Class for MCR-ALS constraint

Usage

constraint(name, params = NULL, method = NULL)

Arguments

name short text with name for the constraint
params a list with parameters for the constraint method (if NULL - default parameters will be used)
method method to call when applying the constraint, provide it only for user defined constraints

Details

Use this class to create constraints and add them to a list for MCR-ALS curve resolution (see mcrals). Either provide name and parameters to one of the existing constraint implementations or make your own. See the list of implemented constraints by running constraints()

For your own constraint you need to create a method, which takes matrix with values (either spectra or contributions being resolved) as the first argument, does something and then return a matrix with the same dimension as the result. The method can have any number of optional parameters.

See help for mcrals or Bookdown tutorial for details.

constraintAngle

Method for angle constraint

Description

Adds a small portion of mean to contributions or spectra to increase contrast

Usage

constraintAngle(x, d, weight = 0.05)

Arguments

x data matrix (spectra or contributions)
d matrix with the original spectral values
weight how many percent of mean to add (between 0 and 1)
**constraintClosure**  
Method for closure constraint

---

**Description**
Force rows of data sum up to given value

**Usage**
```
constraintClosure(x, d, sum = 1)
```

**Arguments**
- **x**: data matrix (spectra or contributions)
- **d**: matrix with the original spectral values
- **sum**: which value the spectra or contributions should sum up to

---

**constraintNonNegativity**  
Method for non-negativity constraint

---

**Description**
Set all negative values in the matrix to 0

**Usage**
```
constraintNonNegativity(x, d)
```

**Arguments**
- **x**: data matrix (spectra or contributions)
- **d**: matrix with the original spectral values
constraintNorm  Method for normalization constraint

**Description**

Normalize rows of matrix to unit length or area

**Usage**

```
constraintNorm(x, d, type = "length")
```

**Arguments**

- **x**  
  data matrix (spectra or contributions)
- **d**  
  matrix with the original spectral values
- **type**  
  type of normalization ("area", "length" or "sum")

constraints.list  Shows information about all implemented constraints

**Description**

Shows information about all implemented constraints

**Usage**

```
constraints.list()
```

constraintUnimod  Method for unimodality constraint

**Description**

forces column of matrix to have one maximum each

**Usage**

```
constraintUnimod(x, d, tol = 0)
```

**Arguments**

- **x**  
  data matrix (spectra or contributions)
- **d**  
  matrix with the original spectral values
- **tol**  
  tolerance (value between 0 and 1) to take make method stable to small fluctuations
**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**

```r
crossval(cv = 1, nobj = NULL, resp = NULL)
```

**Arguments**

- `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 performed), 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'`).
- `nobj`: number of objects in a dataset
- `resp`: vector with response values to use in case of venetian blinds

**Value**

matrix with object indices for each segment

**crossval.getParams**

*Define parameters based on 'cv' value*

**Description**

Define parameters based on 'cv' value

**Usage**

```r
crossval.getParams(cv, nobj)
```

**Arguments**

- `cv`: settings for cross-validation provided by user
- `nobj`: number of objects in calibration set
crossval.regmodel  Cross-validation of a regression model

Description
Does cross-validation of a regression model

Usage
crossval.regmodel(obj, x, y, cv, cal.fun)

Arguments
- obj  a regression model (object of class regmodel)
- 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)
- cal.fun  reference to function for model calibration

Value
object of class plsres with results of cross-validation

crossval.simca  Cross-validation of a SIMCA model

Description
Does the cross-validation of a SIMCA model

Usage
crossval.simca(obj, x, cv)

Arguments
- obj  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)

Value
object of class simcares with results of cross-validation
### crossval.str

**String with description of cross-validation method**

**Description**

String with description of cross-validation method

**Usage**

```r
crossval.str(cv)
```

**Arguments**

- `cv`: a list with cross-validation settings

**Value**

a string with the description text

---

### dd.crit

*Calculates critical limits for distance values using Data Driven moments approach*

**Description**

Calculates critical limits for distance values using Data Driven moments approach

**Usage**

```r
dd.crit(paramQ, paramT2, alpha = 0.05, gamma = 0.01)
```

**Arguments**

- `paramQ`: matrix with parameters for distribution of Q distances
- `paramT2`: matrix with parameters for distribution of T2 distances
- `alpha`: significance level for extreme objects
- `gamma`: significance level for outliers
ddmoments.param  Calculates critical limits for distance values using Data Driven moments approach

Description
Calculates critical limits for distance values using Data Driven moments approach

Usage
ddmoments.param(U)

Arguments
U  matrix or vector with distance values

ddrobust.param  Calculates critical limits for distance values using Data Driven robust approach

Description
Calculates critical limits for distance values using Data Driven robust approach

Usage
ddrobust.param(U, ncomp, alpha, gamma)

Arguments
U  matrix or vector with distance values
ncomp  number of components
alpha  significance level for extreme objects
gamma  significance level for outliers
**ellipse**  
*Create ellipse on the current plot*

**Description**  
Create ellipse on the current plot

**Usage**  
```r  
ellipse(xc = 0, yc = 0, a, b, col = "black", lty = 1, ...)  
```

**Arguments**
- `xc` coordinate of center (x)
- `yc` coordinate of center (y)
- `a` major axis
- `b` minor axis
- `col` color of the ellipse line
- `lty` type of the ellipse line
- `...` any argument suitable for `lines` function

**employ.constraint**  
*Applies constraint to a dataset*

**Description**  
Applies constraint to a dataset

**Usage**  
```r  
employ.constraint(obj, x, d, ...)  
```

**Arguments**
- `obj` object with constraint
- `x` matrix with pure spectra or contributions
- `d` matrix with original spectral values
- `...` other arguments
**employ.prep**

*Applies a list with preprocessing methods to a dataset*

**Description**

Applies a list with preprocessing methods to a dataset

**Usage**

`employ.prep(obj, x, ...)`

**Arguments**

- `obj` list with preprocessing methods (created using `prep` function).
- `x` matrix with dataset
- `...` other arguments

---

**eye**

*Create the identity matrix*

**Description**

Create the identity matrix

**Usage**

`eye(n)`

**Arguments**

- `n` Size of the matrix

**Value**

The identity matrix (n x n)
**fprintf**

*Imitation of printf() function*

**Description**

Imitation of printf() function

**Usage**

```plaintext
fprintf(…)
```

**Arguments**

```plaintext
… arguments for sprintf function
```

---

**getCalibrationData**

*Calibration data*

**Description**

Calibration data

**Usage**

```plaintext
getCalibrationData(obj, …)
```

**Arguments**

```plaintext
obj a model object
… other arguments
```

**Details**

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

*Returns matrix with original calibration data*

---

**Description**

Returns matrix with original calibration data

**Usage**

```r
## S3 method for class 'pca'
getCalibrationData(obj)
```

**Arguments**

- `obj` object with PCA model

---

getCalibrationData.simcam

*Get calibration data*

---

**Description**

Get data, used for calibration of the SIMCAM individual models and combine to one dataset.

**Usage**

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

**Arguments**

- `obj` SIMCAM model (object of class simcam)
- `...` other arguments

**Details**

See examples in help for `simcam` function.
**getConfidenceEllipse**  
*Compute confidence ellipse for a set of points*

**Description**

Compute confidence ellipse for a set of points

**Usage**

getConfidenceEllipse(points, conf.level = 0.95, n = 100)

**Arguments**

- **points**: matrix of data frame with coordinates of the points
- **conf.level**: confidence level for the ellipse
- **n**: number of points in the ellipse coordinates

**Value**

matrix with coordinates of the ellipse points (x and y)

---

**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 = obj$ncomp.selected, ...)

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.

getConvexHull

Compute coordinates of a closed convex hull for data points

Description

Compute coordinates of a closed convex hull for data points

Usage

getConvexHull(points)

Arguments

- `points` matrix of data frame with coordinates of the points
getDataLabels

Create a vector with labels for plot series

Description
For scatter plots labels correspond to rows of the data (names, values, indices, etc.). For non-scatter plots labels correspond to the columns (names, indices or max value for each column)

Usage
getDataLabels(ps, labels = NULL)

Arguments
- ps: 'plotseries' object
- labels: vector with user defined labels or type of labels to show ("values", "names", "indices")

getImplementedConstraints

Shows a list with implemented constraints

Description
Shows a list with implemented constraints

Usage
getImplementedConstraints()

getImplementedPrepMethods

Shows a list with implemented preprocessing methods

Description
Shows a list with implemented preprocessing methods

Usage
getImplementedPrepMethods()
getLabelsAsIndices  

Create labels as column or row indices

Description

Create labels as column or row indices

Usage

getLabelsAsIndices(ps)

Arguments

ps  
'plotseries' object

getLabelsAsValues  

Create labels from data values

Description

Create labels from data values

Usage

getLabelsAsValues(ps)

Arguments

ps  
'plotseries' object

getMainTitle  

Get main title

Description

returns main title for a plot depending on a user choice

Usage

getMainTitle(main, ncomp, default)
### getPlotColors

**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**

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

---

### getPlotColors

*Define colors for plot series*

**Description**

Define colors for plot series

**Usage**

getPlotColors(ps, col, opacity, cgroup, colmap)

**Arguments**

- **ps** 'plotseries' object
- **col** color specified by user (if any)
- **opacity** opacity for the color
- **cgroup** vector for color grouping (if any)
- **colmap** name or values for colormap

---

### 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.pca  Probabilities for residual distances

Description

Probabilities for residual distances

Usage

## S3 method for class 'pca'
getProbabilities(obj, ncomp, q, h, ...)

Arguments

- obj: object with PCA model
- ncomp: number of components to compute the probability for
- q: vector with squared orthogonal distances for given number of components
- h: vector with score distances for given number of components
- ...: other parameters

Details

Computes p-value for every object being from the same population as calibration set based on its orthogonal and score distances.

getProbabilities.simca

Probabilities of class belonging for PCA/SIMCA results

Description

Probabilities of class belonging for PCA/SIMCA results

Usage

## S3 method for class 'simca'
getProbabilities(obj, ncomp, q, h, ...)

Arguments

- obj: object with PCA model
- ncomp: number of components to compute the probability for
- q: vector with squared orthogonal distances for given number of components
- h: vector with score distances for given number of components
- ...: other parameters
getPureVariables

Details

Computes p-value for every object being from the same population as calibration set based on its orthogonal and score distances.

Description

The method identifies indices of pure variables using the SIMPLISMA algorithm.

Usage

gapureVariables(D, ncomp, purevars, offset)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>matrix with the spectra</td>
</tr>
<tr>
<td>ncomp</td>
<td>number of pure components</td>
</tr>
<tr>
<td>purevars</td>
<td>user provided values for pure variables (no calculation will be run in this case)</td>
</tr>
<tr>
<td>offset</td>
<td>offset (between 0 and 1) for calculation of parameter alpha</td>
</tr>
</tbody>
</table>

Value

The function returns a list with following fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncomp</td>
<td>number of pure components</td>
</tr>
<tr>
<td>purvars</td>
<td>vector with indices for pure variables.</td>
</tr>
<tr>
<td>purityspec</td>
<td>matrix with purity values for each resolved components.</td>
</tr>
<tr>
<td>purity</td>
<td>vector with purity values for resolved components.</td>
</tr>
</tbody>
</table>

getR

Description

Creates rotation matrix to map a set vectors base1 to a set of vectors base2.

Usage

gap(base1, base2)

Description

In both sets vectors should be orthonormal.
getRegcoeffs.regmodel

Arguments

base1 Matrix (JxA) with A orthonormal vectors as columns to be rotated (A <= J)
base2 Matrix (JxA) with A orthonormal vectors as columns, base1 should be aligned with

Value

Rotation matrix (JxJ)

getRegcoeffs

Get regression coefficients

Description

Generic function for getting regression coefficients from PLS model

Usage

getRegcoeffs(obj, ...)

Arguments

obj a PLS model
...
other parameters

getRegcoeffs.regmodel

Regression coefficients for PLS model'

Description

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

Usage

## S3 method for class 'regmodel'
getRegcoeffs(  
  obj,  
  ncomp = obj$ncomp.selected,  
  ny = 1,  
  full = FALSE,  
  alpha = 0.05,  
  ...  
)
getRes

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. 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 ($y_p = Xb$).

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.

---

getRes

Return list with valid results

Description

Return list with valid results

Usage

getRes(res, classname = "ldecomp")

Arguments

- **res**: list with results
- **classname**: name of class (for result object) to look for
getSelectedComponents *Get selected components*

**Description**

returns number of components depending on a user choice

**Usage**

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**

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

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. This function is a proxy for selratio and will be removed in future releases.

Usage
```r
## S3 method for class 'pls'
getSelectivityRatio(obj, ncomp = obj$ncomp.selected, ...)
```

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)
- `...`: other parameters

Value
vector with selectivity ratio values

References

getVariance.mcr

Compute explained variance for MCR case

Description
Compute explained variance for MCR case

Usage
```r
getVariance.mcr(obj, x)
```

Arguments
- `obj`: object of class mcr
- `x`: original spectral data
**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. This function is a proxy for `vipscores` and will be removed in future releases.

**Usage**

```
## S3 method for class 'pls'
getVIPScores(obj, ncomp = obj$ncomp.selected, ...)
```

**Arguments**

- `obj` a PLS model (object of class `pls`)
- `ncomp` number of components to count
- `...` other parameters

**Value**

matrix `nvar x 1` with VIP score values
**hotelling.crit**

*Calculate critical limits for distance values using Hotelling T2 distribution*

**Description**

Calculate critical limits for distance values using Hotelling T2 distribution

**Usage**

```r
hotelling.crit(nobj, ncomp, alpha = 0.05, gamma = 0.01)
```

**Arguments**

- `nobj` number of objects in calibration set
- `ncomp` number of components
- `alpha` significance level for extreme objects
- `gamma` significance level for outliers

**Value**

vector with four values: critical limits for given alpha and gamma, mean distance and DoF.

**hotelling.prob**

*Calculate probabilities for distance values and given parameters using Hotelling T2 distribution*

**Description**

Calculate probabilities for distance values and given parameters using Hotelling T2 distribution

**Usage**

```r
hotelling.prob(u, ncomp, nobj)
```

**Arguments**

- `u` vector with distances
- `ncomp` number of components
- `nobj` number of objects in calibration set
**imshow**  
*show image data as an image*

**Description**

show image data as an image

**Usage**

```r
.imshow(
  data,
  channels = 1,
  show.excluded = FALSE,
  main = paste0(" ", colnames(data)[channels]),
  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 = TRUE,
  scale = FALSE,
  cv = list("ven", 10),
  exclcols = NULL,
  exclrows = NULL,
  int.ncomp = glob.ncomp,
```
int.num = NULL,
int.width = NULL,
int.limits = NULL,
int.niter = NULL,
ncomp.selcrit = "min",
method = "forward",
x.test = NULL,
y.test = NULL,
silent = FALSE
)

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 cross-validation settings (see details)
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')
x.test matrix with predictors for test set (by default is NULL, if specified, is used instead of cv).
y.test matrix with responses for test set.
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.

Cross-validation settings, cv, 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 re-
peated cross-validation with nseg segments and nrep repetitions or cv = list(‘ven’, nseg) for
systematic splits to nseg segments (‘venetian blinds’).

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

References

[1] Lars Noergaard at al. Interval partial least-squares regression (iPLS): a comparative chemometric
study with an example from near-infrared spectroscopy. Appl.Spec. 2000; 54: 413-419

Examples

library(mdatools)

## forward selection for simdata
data(simdata)
Xc = simdata$spectra.c
yc = simdata$conc.c[, 3, drop = FALSE]

# 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, int.stat, glob.stat)

Arguments

x a matrix with predictor values
y a vector with response values
obj object with initial settings for iPLS algorithm
int.stat data frame with initial interval statistics
glob.stat data frame with initial global statistics

ipls.forward

Runs the forward iPLS algorithm

Description

Runs the forward iPLS algorithm

Usage

ipls.forward(x, y, obj, int.stat, glob.stat)
Arguments

- **x**: a matrix with predictor values
- **y**: a vector with response values
- **obj**: object with initial settings for iPLS algorithm
- **int.stat**: data frame with initial interval statistics
- **glob.stat**: data frame with initial global statistics

**jm.crit**

*Calculate critical limits for distance values using Jackson-Mudholkar approach*

**Description**

Calculate critical limits for distance values using Jackson-Mudholkar approach

**Usage**

```
jm.crit(residuals, eigenvals, alpha = 0.05, gamma = 0.01)
```

**Arguments**

- **residuals**: matrix with PCA residuals
- **eigenvals**: vector with eigenvalues for PCA components
- **alpha**: significance level for extreme objects
- **gamma**: significance level for outliers

**Value**

vector with four values: critical limits for given alpha and gamma, mean distance and DoF.

**jm.prob**

*Calculate probabilities for distance values and given parameters using Hotelling T2 distribution*

**Description**

Calculate probabilities for distance values and given parameters using Hotelling T2 distribution

**Usage**

```
jm.prob(u, eigenvals, ncomp)
```
Arguments

- `u`: vector with distances
- `eigenvals`: vector with eigenvalues for PCA components
- `ncomp`: number of components

**ldecomp**

Class for storing and visualising linear decomposition of dataset \( X = TP' + E \)

Description

Creates an object of ldecomp class.

Usage

```
ldecomp(scores, loadings, residuals, eigenvals, ncomp.selected = ncol(scores))
```

Arguments

- `scores`: matrix with score values (I x A).
- `loadings`: matrix with loading values (J x A).
- `residuals`: matrix with data residuals (I x J).
- `eigenvals`: vector with eigenvalues for the loadings
- `ncomp.selected`: number of selected components

Details

ldecomp is a general class for storing results of decomposition of dataset in form \( X = TP' + E \). 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 of residual distances and explained variance.

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 (I x A).
- `residuals`: matrix with data residuals (I x J).
- `T2`: matrix with score distances (I x A).
- `Q`: matrix with orthogonal distances (I x A).
- `ncomp.selected`: selected number of components.
- `expvar`: explained variance for each component.
- `cumexpvar`: cumulative explained variance.
### ldecomp.getDistances  
*Compute score and residual distances*

**Description**
Compute orthogonal Euclidean distance from object to PC space \((Q, q)\) and Mahalanobis squared distance between projection of the object to the space and its origin \((T2, h)\).

**Usage**
```
ldecomp.getDistances(scores, loadings, residuals, eigenvals)
```

**Arguments**
- `scores`: matrix with scores \((T)\).
- `loadings`: matrix with loadings \((P)\).
- `residuals`: matrix with residuals \((E)\).
- `eigenvals`: vector with eigenvalues for the components

**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 \(Q\), \(T2\) and tnorm values for each component.

### ldecomp.getLimitsCoordinates  
*Compute coordinates of lines or curves with critical limits*

**Description**
Compute coordinates of lines or curves with critical limits

**Usage**
```
ldecomp.getLimitsCoordinates(  
    Qlim,  
    T2lim,  
    ncomp,  
    norm,  
    log,  
    show.limits = c(TRUE, TRUE)  
)
```

**Arguments**
- `Qlim`:  
- `T2lim`:  
- `ncomp`:  
- `norm`:  
- `log`:  
- `show.limits`: a logical vector indicating if limits should be displayed.
ldecomp.getLimParams

Arguments

- **Qlim**: matrix with critical limits for orthogonal distances
- **T2lim**: matrix with critical limits for score distances
- **ncomp**: number of components for computing the coordinates
- **norm**: logical, shall distance values be normalized or not
- **log**: logical, shall log transformation be applied or not
- **show.limits**: vector with two logical values defining if limits for extreme and/or outliers must be shown

Value

- list with two matrices (x and y coordinates of corresponding limits)

---

ldecomp.getLimParams  Compute parameters for critical limits based on calibration results

Description

Compute parameters for critical limits based on calibration results

Usage

ldecomp.getLimParams(U)

Arguments

- **U**: matrix with residual distances

---

ldecomp.getQLimits  Compute critical limits for orthogonal distances (Q)

Description

Compute critical limits for orthogonal distances (Q)

Usage

ldecomp.getQLimits(lim.type, alpha, gamma, params, residuals, eigenvals)
Arguments

- **lim.type**: which method to use for calculation of critical limits for residuals
- **alpha**: significance level for extreme limits.
- **gamma**: significance level for outlier limits.
- **params**: distribution parameters returned by ldecomp.getLimParams
- **residuals**: matrix with residuals (E)
- **eigenvals**: eigenvalues for the components used to decompose the data

**ldecomp.getT2Limits**  
*Compute critical limits for score distances (T2)*

**Description**

Compute critical limits for score distances (T2)

**Usage**

```r
ldecomp.getT2Limits(lim.type, alpha, gamma, params)
```

**Arguments**

- **lim.type**: which method to use for calculation ("chisq", "ddmoments", "ddrobust")
- **alpha**: significance level for extreme limits.
- **gamma**: significance level for outlier limits.
- **params**: distribution parameters returned by ldecomp.getLimParams

**ldecomp.getVariances**  
*Compute explained variance*

**Description**

Computes explained variance and cumulative explained variance for data decomposition.

**Usage**

```r
ldecomp.getVariances(scores, loadings, residuals, Q)
```

**Arguments**

- **scores**: matrix with scores (T).
- **loadings**: matrix with loadings (P).
- **residuals**: matrix with residuals (E).
- **Q**: matrix with squared orthogonal distances.
ldecomp.plotResiduals

Value

Returns a list with two vectors.

Description

Residuals distance plot for a set of ldecomp objects

Shows a plot with score (T2, h) vs orthogonal (Q, q) distances and corresponding critical limits for given number of components.

Usage

ldecomp.plotResiduals(
  res, Qlim, T2lim, ncomp, log = FALSE, norm = FALSE, cgroup = NULL, xlim = NULL, ylim = NULL, show.limits = c(TRUE, TRUE), lim.col = c("darkgray", "darkgray"), lim.lwd = c(1, 1), lim.lty = c(2, 3), show.legend = TRUE, legend.position = "topright", show.excluded = FALSE, ...
)

Arguments

res list with result objects to show the plot for
Qlim matrix with critical limits for orthogonal distance
T2lim matrix with critical limits for score distance
ncomp how many components to use (by default optimal value selected for the model will be used)
log logical, apply log tranformation to the distances or not (see details)
norm logical, normalize distance values or not (see details)
cgroup color grouping of plot points (works only if one result object is available)
xlim limits for x-axis (if NULL will be computed automatically)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ylim</td>
<td>limits for y-axis (if NULL will be computed automatically)</td>
</tr>
<tr>
<td>show.limits</td>
<td>vector with two logical values defining if limits for extreme and/or outliers must be shown</td>
</tr>
<tr>
<td>lim.col</td>
<td>vector with two values - line color for extreme and outlier limits</td>
</tr>
<tr>
<td>lim.lwd</td>
<td>vector with two values - line width for extreme and outlier limits</td>
</tr>
<tr>
<td>lim.lty</td>
<td>vector with two values - line type for extreme and outlier limits</td>
</tr>
<tr>
<td>show.legend</td>
<td>logical, show or not legend on the plot (if more than one result object)</td>
</tr>
<tr>
<td>legend.position</td>
<td>if legend must be shown, where it should be</td>
</tr>
<tr>
<td>show.excluded</td>
<td>logical, show or hide rows marked as excluded (attribute ‘exclrows’).</td>
</tr>
<tr>
<td>...</td>
<td>other plot parameters (see mdpplotg for details)</td>
</tr>
</tbody>
</table>

**Details**

The function is a bit more advanced version of `plotResiduals.ldecomp`. It allows to show distance values for several result objects (e.g. calibration and test set or calibration and new prediction set) as well as display the corresponding critical limits in form of lines or curves.

Depending on how many result objects your model has or how many you specified manually, using the `res` parameter, the plot behaves in a bit different way.

If only one result object is provided, then it allows to colorise the points using `cgroup` parameter.

If two or more result objects are provided, then the function show distances in groups, and adds corresponding legend.

The function can show distance values normalised (h/h0 and q/q0) as well as with log transformation (log(1 + h/h0), log(1 + q/q0)). The latter is useful if distribution of the points is skewed and most of them are densely located around bottom left corner.

---

**mcr**

*General class for Multivariate Curve Resolution model*

**Description**

mcr is used to store and visualise general MCR data and results.

**Usage**

```r
mcr(x, ncomp, method, exclrows = NULL, exclcols = NULL, info = "", ...)```

**Arguments**

- `x` spectra of mixtures (as matrix or data frame)
- `ncomp` number of pure components to resolve
- `method` function for computing spectra of pure components
mcrals

**Description**

`mcrals` allows to resolve spectroscopic data to linear combination of individual spectra and contributions using the alternating least squares (ALS) algorithm with constraints.

**Usage**

```r
mcrals(
  x,
  ncomp,
  cont.constraints = list(),
  spec.constraints = list(),
  spec.ini = matrix(runif(ncol(x) * ncomp), ncol(x), ncomp),
  cont.forced = matrix(NA, nrow(x), ncomp),
  spec.forced = matrix(NA, ncol(x), ncomp),
  cont.solver = mcrals.nnls,
  spec.solver = mcrals.nnls,
  exclrows = NULL,
  exclcols = NULL,
  verbose = FALSE,
  max.niter = 100,
  tol = 10^-6,
  info = ""
)
```

**Arguments**

- `x` spectra of mixtures (matrix or data frame).
- `ncomp` number of components to calculate.
- `cont.constraints` a list with constraints to be applied to contributions (see details).
- `spec.constraints` a list with constraints to be applied to spectra (see details).
- `spec.ini` a matrix with initial estimation of the pure components spectra.
- `cont.forced` a matrix which allows to force some of the concentration values (see details).
The method implements the iterative ALS algorithm, where, at each iteration, spectra and contributions of each chemical component are estimated and then a set of constraints is applied to each. The method is well described in [1, 2].

The method assumes that the spectra (D) is a linear combination of pure components spectra (S) and pure component concentrations (C):

\[ D = CS' + E \]

So the task is to get C and S by knowing D. In order to do that you need to provide:

1. Constraints for spectra and contributions. The constraints should be provided as a list with name of the constraint and all necessary parameters. You can see which constraints and parameters are currently supported by running `constraintList()`. See the code examples below or a Bookdown tutorial for more details.

2. Initial estimation of the pure components spectra, S. By default method uses a matrix with random numbers but you can provide a better guess (for example by running `mcrpure`) as a first step.

3. Which solver to use for resolving spectra and concentrations. There are two built in solvers: `mcrals.nnls` (default) and `mcrals.ols`. The first implements non-negative least squares method which gives non-negative (thus physically meaningful) solutions. The second is ordinary least squares and if you want to get non-negative spectra and/or contributions in this case you need to provide a non-negativity constraint.

The algorithm iteratively resolves C and S and checks how well CS’ is to D. The iterations stop either when number exceeds value in `max.niter` or when improvements (difference between explained variance on current and previous steps) is smaller than `tol` value.

Parameters `cont.force` and `spec.force` allows you to force some parts of the contributions or the spectra to be equal to particular pre-defined values. In this case you need to provide the parameters (or just one of them) in form of a matrix. For example `cont.force` should have as many rows as many you have in the original spectral data x and as many columns as many pure components you want to resolve. Feel all values of this matrix with NA and the values you want to force with

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spec.forced</code></td>
<td>a matrix which allows to force some of the spectra values (see details).</td>
</tr>
<tr>
<td><code>cont.solver</code></td>
<td>which function to use as a solver for resolving of pure components contributions (see details).</td>
</tr>
<tr>
<td><code>spec.solver</code></td>
<td>which function to use as a solver for resolving of pure components spectra (see details).</td>
</tr>
<tr>
<td><code>exclrows</code></td>
<td>rows to be excluded from calculations (numbers, names or vector with logical values).</td>
</tr>
<tr>
<td><code>exclcols</code></td>
<td>columns to be excluded from calculations (numbers, names or vector with logical values).</td>
</tr>
<tr>
<td><code>verbose</code></td>
<td>logical, if TRUE information about every iteration will be shown.</td>
</tr>
<tr>
<td><code>max.niter</code></td>
<td>maximum number of iterations.</td>
</tr>
<tr>
<td><code>tol</code></td>
<td>tolerance, when explained variance change is smaller than this value, iterations stop.</td>
</tr>
<tr>
<td><code>info</code></td>
<td>a short text with description of the case (optional).</td>
</tr>
</tbody>
</table>
real numbers. For example if you know that in the first measurement concentration of 2 and 3 components was zero, set the corresponding values of `cont.force` to zero. See also the last case in the examples section.

Value

Returns an object of `mcrpure` class with the following fields:

- `resspec` matrix with resolved spectra.
- `rescont` matrix with resolved contributions.
- `cont.constraints` list with contribution constraints provided by user.
- `spec.constraints` list with spectra constraints provided by user.
- `expvar` vector with explained variance for each component (in percent).
- `cumexpvar` vector with cumulative explained variance for each component (in percent).
- `ncomp` number of resolved components
- `max.niter` maximum number of iterations
- `info` information about the model, provided by user when build the model.

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

Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

References


See Also

Methods for `mcrals` objects:

- `summary.mcrals` shows some statistics for the case.
- `predict.mcrals` computes contributions by projection of new spectra to the resolved ones.

Plotting methods for `mcrals` objects:

- `plotSpectra.mcr` shows plot with resolved spectra.
- `plotContributions.mcr` shows plot with resolved contributions.
- `plotVariance.mcr` shows plot with explained variance.
- `plotCumVariance.mcr` shows plot with cumulative explained variance.
library(mdatools)

# resolve mixture of carbon hydrates Raman spectra
data(carbs)

# define constraints for contributions
cc <- list(constraint("nonneg")
)

# define constraints for spectra
cs <- list(constraint("nonneg"), constraint("norm", params = list(type = "area"))

# because by default initial approximation is made by using random numbers
# we need to seed the generator in order to get reproducible results
set.seed(6)

# run ALS
m <- mcrals(carbs$D, ncomp = 3, cont.constraints = cc, spec.constraints = cs)
summary(m)

# plot cumulative and individual explained variance
par(mfrow = c(1, 2))
plotVariance(m)
plotCumVariance(m)

# plot resolved spectra (all of them or individually)
par(mfrow = c(2, 1))
plotSpectra(m)
plotSpectra(m, comp = 2:3)

# plot resolved contributions (all of them or individually)
par(mfrow = c(2, 1))
plotContributions(m)
plotContributions(m, comp = 2:3)

# of course you can do this manually as well, e.g. show original
# and resolved spectra
par(mfrow = c(1, 1))
mdaplotg(
  list("original" = prep.norm(carbs$D, "area"),
       "resolved" = prep.norm(mda.subset(mda.t(m$resspec), 1), "area")),
  col = c("gray", "red"), type = "l"
in case if you have reference spectra of components you can compare them with
# the resolved ones:
par(mfrow = c(3, 1))
for (i in 1:3) {
  mda.plotg(
    list(
      "pure" = prep.norm(mda.subset(mda.t(carbs$S), 1), "area"),
      "resolved" = prep.norm(mda.subset(mda.t(m$resspec), 1), "area")
    ), col = c("gray", "red"), type = "l", lwd = c(3, 1)
  )
}

This example shows how to force some of the contribution values
First of all we combine the matrix with mixtures and the pure spectra, so the pure
spectra are on top of the combined matrix
Dplus <- mda.rbind(mda.t(carbs$S), carbs$D)

since we know that concentration of C2 and C3 is zero in the first row (it is a pure
spectrum of first component), we can force them to be zero in the optimization procedure.
Similarly we can do this for second and third rows.

cont.forced <- matrix(NA, nrow(Dplus), 3)
cont.forced[1, ] <- c(NA, 0, 0)
cont.forced[2, ] <- c(0, NA, 0)
cont.forced[3, ] <- c(0, 0, NA)

m <- mcrals(Dplus, 3, cont.forced = cont.forced, cont.constraints = cc, spec.constraints = cs)
plot(m)

See bookdown tutorial for more details.

tables

mcrals.cal

Identifies pure variables

Description

The method identifies indices of pure variables using the SIMPLISMA algorithm.

Usage

mcrals.cal(
  D,
  ncomp,
  cont.constraints,
  spec.constraints,
spec.ini,
cont.forced,
spec.forced,
cont.solver,
spec.solver,
max.niter,
tol,
verbose
)

Arguments

D                    matrix with the spectra
ncomp                 number of pure components
cont.constraints      a list with constraints to be applied to contributions (see details).
spec.constraints      a list with constraints to be applied to spectra (see details).
spec.ini              a matrix with initial estimation of the pure components spectra.
cont.forced           a matrix which allows to force some of the concentration values (see details).
spec.forced           a matrix which allows to force some of the spectra values (see details).
cont.solver           which function to use as a solver for resolving of pure components contributions (see details).
spec.solver           which function to use as a solver for resolving of pure components spectra (see details).
max.niter             maximum number of iterations.
tol                   tolerance, when explained variance change is smaller than this value, iterations stop.
verbose               logical, if TRUE information about every iteration will be shown.

Value

The function returns a list with with following fields:

ncomp                 number of pure components.
resspec               matrix with resolved spectra.
rescont               matrix with resolved contributions.
cont.constraints      list with contribution constraints provided by user.
spec.constraints      list with spectra constraints provided by user.
max.niter             maximum number of iterations
mcrals.fcnnls

**Fast combinatorial non-negative least squares**

**Description**

Fast combinatorial non-negative least squares

**Usage**

```r
mcrals.fcnnls(
  D,
  A,
  tol = 10 * .Machine$double.eps * as.numeric(sqrt(crossprod(A[, 1]))) * nrow(A)
)
```

**Arguments**

- `D`: a matrix
- `A`: a matrix
- `tol`: tolerance parameter for algorithm convergence

**Details**

Computes Fast combinatorial NNLS solution for B: D = AB' subject to B >= 0. Implements the method described in [1].

**References**


mcrals.nnls

**Non-negative least squares**

**Description**

Non-negative least squares

**Usage**

```r
mcrals.nnls(
  D,
  A,
  tol = 10 * .Machine$double.eps * as.numeric(sqrt(crossprod(A[, 1]))) * nrow(A)
)
```
Arguments

\[ D \]  
\[ A \]  
\[ \text{tol} \]  

Details

Computes NNLS solution for \( B: D = AB' \) subject to \( B \geq 0 \). Implements the active-set based algorithm proposed by Lawson and Hanson [1].

References


---

\[ \text{mcrals.ols} \quad \text{Ordinary least squares} \]

Description

Ordinary least squares

Usage

\[ \text{mcrals.ols}(D, A) \]

Arguments

\[ D \]  
\[ A \]  

Details

Computes OLS solution for \( D = AB' \) (or \( D' = AB' \)), where \( D, A \) are known
Description

mcrpure allows to resolve spectroscopic data to linear combination of individual spectra and contributions using the pure variables approach.

Usage

mcrpure(
  x,  
  ncomp, 
  purevars = NULL, 
  offset = 0.05, 
  exclrows = NULL, 
  exclcols = NULL, 
  info = ""
)

Arguments

x spectra of mixtures (matrix or data frame).

ncomp maximum number of components to calculate.

purevars vector with indices for pure variables (optional, if you want to provide the variables directly).

offset offset for correcting noise in computing maximum angles (should be value within [0, 1]).

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).

info a short text with description of the case (optional).

Details

The method estimates purity of each variable and then uses the purest ones to decompose the spectral data into spectra (‘resspec’) and contributions (‘rescont’) of individual chemical components by ordinary least squares.

The pure variables are identified using stepwise maximum angle calculations and described in detail in [1]. So the purity of a spectral variable (wavelength, wavenumber) is actually an angle (measured in degrees) between the variable and vector of ones for the first component; and between the variable and space formed by previously found pure variables for the other components.
Value

Returns an object of mcrpure class with the following fields:

- resspec: matrix with resolved spectra.
- rescont: matrix with resolved contributions.
- purevars: indices of the selected pure variables.
- purevals: purity values for the selected pure variables.
- purityspec: purity spectra (matrix with purity values for each variable and component).
- expvar: vector with explained variance for each component (in percent).
- cumexpvar: vector with cumulative explained variance for each component (in percent).
- offset: offset value used to compute the purity
- ncomp: number of resolved components
- info: information about the model, provided by user when build the model.

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

Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

References


See Also

Methods for mcrpure objects:

- summary.mcrpure: shows some statistics for the case.
- unmix.mcrpure: makes unmixing of new set of spectra.
- predict.mcrpure: computes contributions by projection of new spectra to the resolved ones.

Plotting methods for mcrpure objects:

- plotPurity.mcrpure: shows plot with maximum purity of each component.
- plotPuritySpectra.mcrpure: shows plot with purity spectra.
- plotSpectra.mcr: shows plot with resolved spectra.
- plotContributions.mcr: shows plot with resolved contributions.
- plotVariance.mcr: shows plot with explained variance.
- plotCumVariance.mcr: shows plot with cumulative explained variance.
Examples

library(mdatools)

# resolve mixture of carbohydrates Raman spectra

data(carbs)
m = mcrpure(carbs$D, ncomp = 3)

# examples for purity spectra plot (you can select which components to show)
par(mfrow = c(2, 1))
plotPuritySpectra(m)
plotPuritySpectra(m, comp = 2:3)

# you can do it manually and combine e.g. with original spectra
par(mfrow = c(1, 1))
mdaplotg(
  list(
    "spectra" = prep.norm(carbs$D, "area"),
    "purity" = prep.norm(mda.subset(mda.t(m$resspec), 1), "area")
  ), col = c("gray", "red"), type = "l"
)

# show the maximum purity for each component
par(mfrow = c(1, 1))
plotPurity(m)

# plot cumulative and individual explained variance
par(mfrow = c(1, 2))
plotVariance(m)
plotCumVariance(m)

# plot resolved spectra (all of them or individually)
par(mfrow = c(2, 1))
plotSpectra(m)
plotSpectra(m, comp = 2:3)

# plot resolved contributions (all of them or individually)
par(mfrow = c(2, 1))
plotContributions(m)
plotContributions(m, comp = 2:3)

# of course you can do this manually as well, e.g. show original
# and resolved spectra
par(mfrow = c(1, 1))
mdaplotg(
  list(
    "original" = prep.norm(carbs$D, "area"),
    "resolved" = prep.norm(mda.subset(mda.t(m$resspec), 1), "area")
  ), col = c("gray", "red"), type = "l"
)

# in case if you have reference spectra of components you can compare them with
# the resolved ones:
par(mfrow = c(3, 1))
for (i in 1:3) {
    mda.plotg(
        list(
            "pure" = prep.norm(mda.subset(mda.t(carbs$S), 1), "area"),
            "resolved" = prep.norm(mda.subset(mda.t(m$resspec), 1), "area")
        ), col = c("gray", "red"), type = "l", lwd = c(3, 1)
    )
}

# See bookdown tutorial for more details.

---

**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**

```r
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**

```r
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

<table>
<thead>
<tr>
<th>mda.exclrows</th>
<th>Exclude/hide rows in a dataset</th>
</tr>
</thead>
</table>

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

<table>
<thead>
<tr>
<th>mda.getattr</th>
<th>Get data attributes</th>
</tr>
</thead>
</table>

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

```r
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

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

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

**Value**  
dataset with included rows
### mda.purge

Removes excluded (hidden) rows and columns from data

**Description**

Removes excluded (hidden) rows and columns from data

**Usage**

```r
mda.purge(data)
```

**Arguments**

- `data` : data frame or matrix with data

---

### mda.purgeCols

Removes excluded (hidden) columns from data

**Description**

Removes excluded (hidden) columns from data

**Usage**

```r
mda.purgeCols(data)
```

**Arguments**

- `data` : data frame or matrix with data

---

### mda.purgeRows

Removes excluded (hidden) rows from data

**Description**

Removes excluded (hidden) rows from data

**Usage**

```r
mda.purgeRows(data)
```

**Arguments**

- `data` : data frame or matrix with data
**mda.rbind**

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

**Description**

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

**Usage**

`mda.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, `x`values, excluded rows and columns, etc.) to a dataset

**Usage**

`mda.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 attributed

**Description**

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

**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

Description

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

Usage

mdaplot(
    data = NULL,
    ps = NULL,
    type = "p",
    pch = 16,
    col = NULL,
    bg = par("bg"),
    bwd = 0.8,
    border = NA,
    lty = 1,
    lwd = 1,
    cex = 1,
    cgroup = NULL,
    xlim = NULL,
    ylim = NULL,
    colmap = "default",
    labels = NULL,
    main = NULL,
    xlab = NULL,
    ylab = NULL,
    show.labels = FALSE,
    show.colorbar = !is.null(cgroup),
    show.lines = FALSE,
    show.grid = TRUE,
    grid.lwd = 0.5,
    grid.col = "lightgray",
    show.axes = TRUE,
    xticks = NULL,
    yticks = NULL,
    xticklabels = NULL,
    yticklabels = NULL,
    xlas = 0,
    ylas = 0,
    lab.col = "darkgray"
\begin{verbatim}
lab.cex = 0.65,
show.excluded = FALSE,
col.excluded = "#C0C0C0",
nbins = 60,
force.x.values = NA,
opacity = 1,
pch.colinv = FALSE,
\end{verbatim}

Arguments

data: a vector, matrix or a data.frame with data values.
ps: 'plotseries' object, if NULL will be created based on the provided data values

Arguments

data: a vector, matrix or a data.frame with data values.
ps: 'plotseries' object, if NULL will be created based on the provided data values
type: type of the plot ("p", "d", "l", "b", "h", "e").
pch: a character for markers (same as plot parameter).
col: a color for markers or lines (same as plot parameter).
bg: background color for scatter plots with 'pch=21:25'.
bwd: a width of a bar as a percent of a maximum space available for each bar.
border: color for border of bars (if barplot is used)
lty: line type
lwd: line width
cex: scale factor for the marker
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).
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 cross the point.
show.grid: logical, show or not a grid for the plot.
grid.lwd: line thickness (width) for the grid.
grid.col: line color for the grid.
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)
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).
pch.colinv allows to swap values for ‘col’ and ‘bg’ for scatter plots with ‘pch’ values from 21 to 25.
... 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 attributes, described in the Bookdown tutorial.

If data is a data frame and contains one or more factors, they will be converted to 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

# See all examples in the tutorial.

mdaplot.areColors( Check color values

Description

Checks if elements of argument are valid color values

Usage

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

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
**mdaplot.getColors**

**Details**

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

**Value**

matrix with formatted values

---

**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**

```r
mdaplot.getColors(
    ngroups = NULL,
    cgroup = NULL,
    colmap = "default",
    opacity = 1,
    maxsplits = 64
)
```

**Arguments**

- `ngroups`: number of colors to create.
- `cgroup`: vector of values, used for color grouping of plot points or lines.
- `colmap`: which colormap to use ('default', 'gray', 'old', or user defined in form c('col1', 'col2', ...)).
- `opacity`: opacity for colors (between 0 and 1)
- `maxsplits`: if continuous values are used for color grouping - how many groups to create?

**Value**

Returns vector with generated color values
mdaplot.getXAxisLim  *Calculate limits for x-axis.*

**Description**

Calculates limits for x-axis depending on data values that have to be plotted, extra plot elements that have to be shown and margins.

**Usage**

```r
mdaplot.getXAxisLim(
    ps,
    xlim,
    show.labels = FALSE,
    show.lines = FALSE,
    show.excluded = FALSE,
    bwd = 0.8
)
```

**Arguments**

- `ps`: 'plotseries' object.
- `xlim`: limits provided by user
- `show.labels`: logical, will data labels be shown on the plot
- `show.lines`: logical or numeric with line coordinates to be shown on the plot.
- `show.excluded`: logical, will excluded values be shown on the plot
- `bwd`: if limits are computed for bar plot, this is a bar width (otherwise NULL)

**Value**

Returns a vector with two limits.

---

mdaplot.getXTickLabels  *Prepare xticklabels for plot*

**Description**

Prepare xticklabels for plot.

**Usage**

```r
mdaplot.getXTickLabels(xticklabels, xticks, excluded_cols)
```
**mdaplot.getXTicks**

Prepare xticks for plot

**Arguments**

- `xticklabels`: xticklabels provided by user (if any)
- `xticks`: xticks (provided or computed)
- `excluded_cols`: columns excluded from plot data (if any)

**Usage**

`mdaplot.getXTicks(xticks, xlim, x_values = NULL, type = NULL)`

**Arguments**

- `xticks`: xticks provided by user (if any)
- `xlim`: limits for x axis
- `x_values`: x values for the plot data object
- `type`: type of the plot

**mdaplot.getYAxisLim**

Calculate limits for y-axis.

**Description**

Calculates limits for y-axis depending on data values that have to be plotted, extra plot elements that have to be shown and margins.

**Usage**

```r
mdaplot.getYAxisLim(
  ps,
  ylim,
  show.lines = FALSE,
  show.excluded = FALSE,
  show.labels = FALSE,
  show.colorbar = FALSE
)
```
Arguments

- `ps` 'plotseries' object.
- `ylim` limits provided by user
- `show.lines` logical or numeric with line coordinates to be shown on the plot.
- `show.excluded` logical, will excluded values be shown on the plot
- `show.labels` logical, will data labels be shown on the plot
- `show.colorbar` logical, will colorbar be shown on the plot

Value

Returns a vector with two limits.

---

**mdaplot.getYTickLabels**

*Prepare yticklabels for plot*

**Description**

Prepare yticklabels for plot

**Usage**

```r
mdaplot.getYTickLabels(yticklabels, yticks, excluded_rows)
```

**Arguments**

- `yticklabels` yticklabels provided by user (if any)
- `yticks` yticks (provided or computed)
- `excluded_rows` rows excluded from plot data (if any)

---

**mdaplot.getYTicks**

*Prepare yticks for plot*

**Description**

Prepare yticks for plot

**Usage**

```r
mdaplot.getYTicks(yticks, ylim, y_values = NULL, type = NULL)
```
**mdaplot.plotAxes**

### Arguments

- `yticks`: yticks provided by user (if any)
- `ylim`: limits for y axis
- `y_values`: y values for the plot data object
- `type`: type of the plot

### Description

`mdaplot.plotAxes`: Create axes plane

Creates an empty axes plane for given parameters

### Usage

```r
mdaplot.plotAxes(
  xticklabels = NULL,
  yticklabels = NULL,
  xlim = xlim,
  ylim = ylim,
  xticks = NULL,
  yticks = NULL,
  main = NULL,
  xlab = NULL,
  ylab = NULL,
  xlas = 0,
  ylas = 0,
  show.grid = TRUE,
  grid.lwd = 0.5,
  grid.col = "lightgray"
)
```

### Arguments

- `xticklabels`: labels for x ticks
- `yticklabels`: labels for y ticks
- `xlim`: vector with limits for x axis
- `ylim`: vector with limits for y axis
- `xticks`: values for x ticks
- `yticks`: values for y ticks
- `main`: main title for the plot
- `xlab`: label for x axis
- `ylab`: label for y axis
mdaplot.prepareColors Prepare colors based on palette and opacity value

Description
Prepare colors based on palette and opacity value

Usage
mdaplot.prepareColors(palette, ncolors, opacity)

Arguments
- palette vector with main colors for current palette
- ncolors number of colors to generate
- opacity opacity for the colors (one value or individual for each color)

Value
vector with colors

mdaplot.showColorbar Plot colorbar

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.showLines**

**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 horizontal and vertical lines on a plot.

**Usage**
```r
mdaplot.showLines(point, lty = 2, lwd = 0.75, col = rgb(0.2, 0.2, 0.2))
```

**Arguments**
- `point`: vector with two values: x coordinate for vertical, 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.

**mdaplotg**

**Plotting function for several plot series**

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

```r
mdaplotg(
  data,
  groupby = NULL,
  type = "p",
  pch = 16,
  lty = 1,
  lwd = 1,
  cex = 1,
  col = NULL,
  bwd = 0.8,
  legend = NULL,
  xlab = NULL,
  ylab = NULL,
  main = NULL,
  labels = NULL,
  ylim = NULL,
  xlim = NULL,
  colmap = "default",
  legend.position = "topright",
  show.legend = TRUE,
  show.labels = FALSE,
  show.lines = FALSE,
  show.grid = TRUE,
  grid.lwd = 0.5,
  grid.col = "lightgray",
  xticks = NULL,
  xticklabels = NULL,
  yticks = NULL,
  yticklabels = NULL,
  show.excluded = FALSE,
  lab.col = "darkgray",
  lab.cex = 0.65,
  xlas = 1,
  ylas = 1,
  opacity = 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).

cex  
the cex factor for the markers (same as plot parameter).

col  
colors for the plot series

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 generate colors if col is not provided

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.

grid.lwd  
line thickness (width) for the grid

grid.col  
line color for the grid

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

opacity  
opacity for plot colors (value between 0 and 1)

...  
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 with 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 (or data frame) with data for one set of objects. Alternatively you can provide data as a matrix and use parameter `groupby` to create the 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)

---

`mdaplotg.getLegend(ps, data.names, legend = NULL)`

Description

Create and return vector with legend values

Usage

`mdaplotg.getLegend(ps, data.names, legend = NULL)`

Arguments

- `ps` list with plot series
- `data.names` names of the data sets
- `legend` legend values provided by user

Value

vector of text values for the legend
mdaplotg.getXLim

Description

Compute x-axis limits for mdaplotg

Usage

mdaplotg.getXLim(
    ps,
    xlim,
    show.excluded,
    show.legend,
    show.labels,
    legend.position,
    bwd = NULL
)

Arguments

ps  list with plotseries
xlim limits provided by user
show.excluded logical, will excluded values also be shown
show.legend will legend be shown on the plot
show.labels will labels be shown on the plot
legend.position position of legend on the plot (if shown)
bwd  size of bar for bar plot

Value

vector with two values

mdaplotg.getYLim

Description

Compute y-axis limits for mdaplotg
Usage

mdaplotg.getYLim(
    ps,
    ylim,
    show.excluded,
    show.legend,
    legend.position,
    show.labels
)

Arguments

ps list with plotseries
ylim limits provided by user
show.excluded logical, will excluded values also be shown
show.legend will legend be shown on the plot
legend.position position of legend on the plot (if shown)
show.labels logical, will data point labels also be shown

Value

vector with two values

Description

Prepare data for mdaplotg

Usage

mdaplotg.prepareData(data, type, groupby)

Arguments

data datasets (in form of list, matrix or data frame)
type vector with type for dataset
groupby factor or data frame with factors - used to split data matrix into groups
Value

list of datasets
The method should prepare data as a list of datasets (matrices or data frames). One list element will be used to create one plot series.
If 'data' is matrix or data frame and not 'groupby' parameter is provided, then every row will be taken as separate set. This option is available only for line or bar plots.

mdaplotg.processParam  Check mdaplotg parameters and replicate them if necessary

Description
Check mdaplotg parameters and replicate them if necessary

Usage
mdaplotg.processParam(param, name, is.type, ngroups)

Arguments
<table>
<thead>
<tr>
<th>param</th>
<th>A parameter to check</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name of the parameter (needed for error message)</td>
</tr>
<tr>
<td>is.type</td>
<td>function to use for checking parameter type</td>
</tr>
<tr>
<td>ngroups</td>
<td>number of groups (plot series)</td>
</tr>
</tbody>
</table>

mdaplotg.showLegend  Show legend for mdaplotg

Description
Shows a legend for plot elements or their groups.

Usage
mdaplotg.showLegend(
  legend,
  col,
  pt.bg = NA,
  pch = NULL,
  lty = NULL,
  lwd = NULL,
  cex = 1,
  bty = "o",
  position = "topright",
  plot = TRUE,
  ...
)
Arguments

- `legend` vector with text elements for the legend items
- `col` vector with color values for the legend items
- `pt.bg` vector with background colors for the legend items (e.g. for `pch = 21:25`)
- `pch` vector with marker symbols for the legend items
- `lty` vector with line types for the legend items
- `lwd` vector with line width values for the legend items
- `cex` vector with cex factor for the points
- `bty` border type for the legend
- `position` legend position ("topright", "topleft", "bottomright", "bottomleft", "top", "bottom")
- `plot` logical, show legend or just calculate and return its size
- `...` other parameters

mdaplotyy

Create line plot with double y-axis

Description

mdaplotyy create line plot for two plot series and uses separate y-axis for each.

Usage

```R
mdaplotyy(
  data,
  type = "l",
  col = mdaplot.getColors(2),
  lty = c(1, 1),
  lwd = c(1, 1),
  pch = (if (type == "b") c(16, 16) else c(NA, NA)),
  cex = 1,
  xlim = NULL,
  ylim = NULL,
  main = attr(data, "name"),
  xlab = attr(data, "xaxis.name"),
  ylab = rownames(data),
  labels = "values",
  show.labels = FALSE,
  lab.cex = 0.65,
  lab.col = "darkgray",
  show.grid = TRUE,
  grid.lwd = 0.5,
  grid.col = "lightgray",
```

Arguments

data a matrix or a data.frame with two rows of values.
type type of the plot ("l" or "b").
col a color for markers or lines (same as plot parameter) for each series.
lty line type for each series (two values)
lwd line width for each series (two values)
pch a character for markers (same as plot parameter) for each series (two values).
cex scale factor for the markers
xlim limits for the x axis (if NULL, will be calculated automatically).
ylim limits for the y axis, either list with two vectors (one for each series) or NULL.
main an overall title for the plot (same as plot parameter).
xlab a title for the x axis (same as plot parameter).
ylab a title for each of the two y axis (as a vector of two text values).
labels a vector with text labels for data points or one of the following: "names", "indices", "values".
show.labels logical, show or not labels for the data objects.
lab.cex size for data point labels.
lab.col color for data point labels.
show.grid logical, show or not a grid for the plot.
grid.lwd line thickness (width) for the grid.
grid.col line color for the grid.
xticks values for x ticks.
xticklabels labels for x ticks.
xlas orientation of xticklabels.
ylas orientation of yticklabels (will be applied to both y axes).
show.legend logical show legend with name of each plot series or not
legend.position position of legend if it must be shown
legend values for the legend
... other plotting arguments.
Details

This plot has properties both mdaplot and mdaplotg, so when you specify color, line properties etc. you have to do it for both plot series.

Author(s)

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

See Also

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

Examples

# See all examples in the tutorial.

---

mdatools 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 (https://svkucheryavski.github.io/mdatools/), there you can also find a Bookdown user tutorial explaining most important features of the package. There is also a dedicated YouTube channel (https://www.youtube.com/channel/UCox0H4utfMq4Flu2yymuyTA) with introductory Chemometric course with examples based on mdatools functionality.

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 and validation methods are implemented:

- **pca, pcares**: Principal Component Analysis (PCA).
- **pls, plsres**: Partial Least Squares regression (PLS).
- **simca, simcares**: Soft Independent Modelling of Class Anallogues (SIMCA).
- **simcam, simcamres**: SIMCA for multiple classes case (SIMCA).
- **plsda, plsdares**: Partial Least Squares Discriminant Analysis (PLS-DA).
- **randtest**: Randomization test for PLS-regression.
- **ipls**: Interval PLS variable.
Multivariate Curve Resolution with Alternating Least Squares.
Multivariate Curve Resolution with Purity approach.
Procrustes Cross Validation.

Methods for data preprocessing:

```r
prep.autoscale`  data mean centering and/or standardization.
prep.savgol     Savitzky-Golay transformation.
prep.snv        Standard normal variate.
prep.msc        Multiplicative scatter correction.
prep.norm       Spectra normalization.
prep.alsbasecorr Baseline correction with Asymmetric Least Squares.
```

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)

---

**pca**

*Principal Component Analysis*

**Description**

*pca* is used to build and explore a principal component analysis (PCA) model.

**Usage**

```r
pca(
  x,
  ncomp = min(nrow(x) - 1, ncol(x), 20),
  center = TRUE,
  scale = FALSE,
  exclrows = NULL,
  exclcols = NULL,
  x.test = NULL,
  method = "svd",
  rand = NULL,
  lim.type = "ddmoments",
  alpha = 0.05,
  gamma = 0.01,
  info = ""
)
```
**Arguments**

- `x`: calibration data (matrix or data frame).
- `ncomp`: maximum number of components to calculate.
- `center`: logical, do mean centering of data or not.
- `scale`: logical, do standardization of data or not.
- `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`: test data (matrix or data frame).
- `method`: method to compute principal components ("svd", "nipals").
- `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 residual distances (see details).
- `alpha`: significance level for extreme limits for T2 and Q distances.
- `gamma`: significance level for outlier limits for T2 and Q distances.
- `info`: a short text with model description.

**Details**

Note, that from v. 0.10.0 cross-validation is no more supported in PCA.

If number of components is not specified, a minimum of number of objects - 1 and number of variables in calibration set is used. One can also specified an optimal number of component, once model is calibrated (`ncomp.selected`). The optimal number of components is used to build a residuals distance plot, as well as for SIMCA classification.

If some of rows of calibration set should be excluded from calculations (e.g. because they are outliers) you can provide row numbers, names, or logical values as parameter `exclrows`. In this case they will be completely ignored we model is calibrated. However, score and residuals distances will be computed for these rows as well and then hidden. You can show them on corresponding plots by using parameter `show.excluded = TRUE`.

It is also possible to exclude selected columns from calculations by providing parameter `exclcols` in form of column numbers, names or logical values. In this case loading matrix will have zeros for these columns. This allows to compute PCA models for selected variables without removing them physically from a dataset.

Take into account that 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 almost precise solution but much faster.
There are several ways to calculate critical limits for orthogonal (Q, q) and score (T2, h) distances. In `mdatools` you can specify one of the following methods via parameter `lim.type`: "jm" Jackson-Mudholkar approach [3], "chisq" - method based on chi-square distribution [4], "ddmoments" and "ddrobust" - related to data driven method proposed in [5]. The "ddmoments" is based on method of moments for estimation of distribution parameters (also known as "classical" approach) while "ddrobust" is based on robust estimation.

If `lim.type="chisq"` or `lim.type="jm"` is used, only limits for Q-distances are computed based on corresponding approach, limits for T2-distances are computed using Hotelling’s T-squared distribution. The methods utilizing the data driven approach calculate limits for combination of the distances bases 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 residual distance plot).

You can also recalculate the limits for existent model by using different values for alpha and gamma, 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`): 

```r
m = setDistanceLimits(m, lim.type, 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 also employed 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 `pcares` with PCA results for a calibration data.
- `testres` an object of class `pcares` with PCA results for a test data, if it was provided.

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.
- categorize.pca: categorize data rows as "normal", "extreme" or "outliers".
- selectCompNum.pca: set number of optimal components in the model
- setDistanceLimits.pca: set critical limits for residuals
- predict.pca: applies PCA model to a new data.

Plotting methods for pca objects:

- 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 plot for residual distances (Q vs. T2).
- plotBiplot.pca: shows bi-plot.
- plotExtreme.pca: shows extreme plot.
- plotT2DoF: plot with degrees of freedom for score distance.
- plotQDoF: plot with degrees of freedom for orthogonal distance.
- plotDistDoF: plot with degrees of freedom for both distances.

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

```
library(mdatools)

### Examples for PCA class
```
## 1. Make PCA model for People data with autoscaling

```r
data(people)
model = pca(people, scale = TRUE, info = "Simple PCA model")
model = selectCompNum(model, 4)
summary(model)
plot(model, show.labels = TRUE)
```

## 2. Show scores and loadings plots for the model

```r
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))
```

## 3. Show residual distance and variance plots for the model

```r
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))
```

---

### pca.cal  
**PCA model calibration**

**Description**

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

**Usage**

```r
pca.cal(x, ncomp, center, scale, method, rand = NULL)
```

**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)
- **rand**: vector with parameters for randomized PCA methods (if NULL, conventional PCA is used instead)
Value

an object with calibrated PCA model

---

`pca.getB`  
*Low-dimensional approximation of data matrix X*

Description

Low-dimensional approximation of data matrix X

Usage

`pca.getB(X, k = NULL, rand = NULL, 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"`

---

`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
pca.mvreplace(
  x,
  center = TRUE,
  scale = FALSE,
  maxncomp = 10,
  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

```r
pca.nipals(x, ncomp = min(ncol(x), nrow(x) - 1), tol = 10^-10)
```

#### Arguments

- **x**: a matrix with data values (preprocessed)
- **ncomp**: number of components to calculate
- **tol**: tolerance (if difference in eigenvalues is smaller - convergence achieved)

#### Value

a list with scores, loadings and eigenvalues for the components

#### References


### pca.run

Runs one of the selected PCA methods

#### Description

Runs one of the selected PCA methods

#### Usage

```r
pca.run(x, ncomp, method, rand = NULL)
```

#### Arguments

- **x**: data matrix
- **ncomp**: number of components
- **method**: name of PCA methods ('svd', 'nipals')
- **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**

```r
pca.svd(x, ncomp = min(ncol(x), nrow(x) - 1))
```

**Arguments**

- `x`: a matrix with data values (preprocessed)
- `ncomp`: number of components to calculate

**Value**

a list with scores, loadings and eigenvalues for the components

---

**pcares**

* Results of PCA decomposition

**Description**

pcares is used to store and visualise results for PCA decomposition.

**Usage**

```r
pcares(...)```

**Arguments**

... all 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.

It is assumed that data is a matrix or data frame with I rows and J columns.
Value

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

- **scores**: matrix with score values (I x A).
- **residuals**: matrix with data residuals (I x J).
- **T2**: matrix with score distances (I x A).
- **Q**: matrix with orthogonal distances (I x A).
- **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 distance 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, 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, 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))
plotVariance(res, type = "h")
plotCumVariance(res, show.labels = TRUE)
plotResiduals(res, show.labels = TRUE, cgroup = people[, "Sex"])
plotResiduals(res, ncomp = 2, show.labels = TRUE)
par(mfrow = c(1, 1))

---

**pcv**

*Compute matrix with pseudo-validation set*

**Description**

The method computes pseudo-validation matrix \( X_{pv} \), based on PCA decomposition of calibration set \( X \) and systematic (venetian blinds) cross-validation. It is assumed that data rows are ordered correctly, so systematic cross-validation can be applied.

All details can be found in [1]

**Usage**

```r
pcv(
    x,
    ncomp = min(round(nrow(x)/nseg) - 1, col(x), 20),
    nseg = 4,
    scale = FALSE
)
```
Arguments

- **x**: matrix with calibration set (IxJ)
- **ncomp**: number of components for PCA decomposition
- **nseg**: number of segments in cross-validation
- **scale**: logical, standardize columns of X prior to decompositon or not

Value

Pseudo-validation matrix (IxJ)

References


<table>
<thead>
<tr>
<th>pellets</th>
<th>Image data</th>
</tr>
</thead>
</table>

Description

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

Usage

```r
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  

*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.

\[ 1 \]
\[ 2 \]
\[ 3 \]
\[ 4 \]
\[ 5 \]
\[ 6 \]
\[ 7 \]
\[ 8 \]
\[ 9 \]
\[ 10 \]
\[ 11 \]
\[ 12 \]

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

**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**

Description
Computes pseudo-inverse matrix using SVD

Usage
```r
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. Alias for `plotPredictions.classres`.

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

Arguments
- `x`: classification results (object of class `plsdares`, `simcamres`, etc.).
- `...`: other arguments for `plotPredictions()` method.

---

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.mcr  

Plot summary for MCR model

Description
Plot summary for MCR model

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

Arguments
  x        mcr model object
  ...      other parameters

plot.pca  

Model overview plot for PCA

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

Usage
## S3 method for class 'pca'
plot(
  x,
  comp = c(1, 2),
  ncomp = x$ncomp.selected,
  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
  ncomp    number of components to show the residuals plot 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
plot.pcares

Plot method for PCA results object

Description

Show several plots to give an overview about the PCA results

Usage

## S3 method for class 'pcares'
plot(x, comp = c(1, 2), ncomp = x$ncomp.selected, show.labels = TRUE, ...)

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).

ncomp
  how many components to use for showing the residual distance plot

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

## S3 method for class 'pls'
plot(x, ncomp = x$ncomp.selected, ny = 1, show.legend = TRUE, ...)

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

... other arguments

Details

See examples in help for pca function.
plot.plsda

**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 = x$ncomp.selected, nc = 1, show.legend = TRUE, ...)
```

**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
- `...`: 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 = 1, ncomp = x$ncomp.selected, show.labels = FALSE, ...)
```
plot.plsres

Arguments

- **x**  PLS-DA results (object of class `plsdares`)
- **nc**  which class to show the plot for
- **ncomp**  how many components to use
- **show.labels**  logical, show or not labels for the plot objects
- **...**  other arguments

Details

See examples in help for `pls` function.

---

**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 = x$ncomp.selected, ny = 1, show.labels = FALSE, ...)
```

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

## 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

## S3 method for class 'regcoeffs'
plot(  
  x,
  ncomp = 1,
  ny = 1,
  type = (if (x$nvar > 30) "l" else "h"),
  col = c(mdaplot.getColors(1), "lightgray"),
  show.lines = c(NA, 0),
  show.ci = FALSE,
  alpha = 0.05,
  ylab = paste0("Coefficients (", x$respnames[ny], ")")  
  ...
)
**Arguments**

- **x**: regression coefficients object (class `regcoeffs`)
- **ncomp**: number of components to use for creating the plot
- **ny**: index of response variable to make the plot for
- **type**: type of the plot
- **col**: vector with two colors for the plot (one is used to show real coefficient and another one to show confidence intervals)
- **show.lines**: allows to show horizontal line at c(NA, 0)
- **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)
- **ylab**: label for y-axis
- **...**: other arguments for plotting methods (e.g. main, xlab, etc)

**plot.regres**

*Plot method for regression results*

**Description**

Plot method for regression results

**Usage**

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

**Arguments**

- **x**: regression results (object of class `regres`)
- **...**: other arguments

**Details**

This is a shortcut for `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, comp = c(1, 2), ncomp = x$ncomp.selected, ...)
```

**Arguments**

- `x` a SIMCA model (object of class `simca`)
- `comp` which components to show on scores and loadings plot
- `ncomp` how many components to use for residuals plot
- `...` 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.
plot.simcamres  

Model overview plot for SIMCAM results  

Description

Just shows a prediction plot for SIMCAM results.

Usage

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

Arguments

- `x`: SIMCAM results (object of class `simcamres`)
- `...`: other arguments

Details

See examples in help for `simcamres` function.

plotBars  

Show plot series as bars  

Description

First row of the data matrix is taken for creating the bar series. In case of barplot color grouping is made based on columns (not rows as for all other plots).

Usage

```r
plotBars(ps, col = ps$col, bwd = 0.8, border = NA, force.x.values = NA)
```

Arguments

- `ps`: ‘plotseries’ object
- `col`: colors of the bars
- `bwd`: width of the bars (as a ratio for max width)
- `border`: color of bar edges
- `force.x.values`: vector with corrected x-values for a bar plot (needed for group plots, do not change manually).
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

## 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 = adjustcolor(col, alpha.f = 0.5),
  ...
)
plotConfidenceEllipse

**Arguments**

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

**plotConfidenceEllipse**  *Add confidence ellipse for groups of points on scatter plot*

**Description**

The method shows confidence ellipse for groups of points on a scatter plot made using `mdaplot()` function with `cgroup` parameter. It will work only if `cgroup` is a factor.

**Usage**

```r
plotConfidenceEllipse(p, conf.level = 0.95, lwd = 1, lty = 1, opacity = 0)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>plot data returned by function <code>mdaplot()</code>.</td>
</tr>
<tr>
<td>conf.level</td>
<td>confidence level to make the ellipse for (between 0 and 1).</td>
</tr>
<tr>
<td>lwd</td>
<td>thickness of line used to show the hull.</td>
</tr>
<tr>
<td>lty</td>
<td>type of line used to show the hull.</td>
</tr>
<tr>
<td>opacity</td>
<td>of opacity is 0 ellipse is transparent otherwise semi-transparent.</td>
</tr>
</tbody>
</table>
Examples

# adds 90% confidence ellipse with semi-transparent area over two clusters of points

library(mdatools)
data(people)
group <- factor(people[, "Sex"], labels = c("Male", "Female"))

# first make plot and then add confidence ellipse
p <- mdaplot(people, type = "p", cgroup = group)
plotConfidenceEllipse(p, conf.level = 0.90, opacity = 0.2)

plotContributions

Plot resolved contributions

Description

Plot resolved contributions

Usage

plotContributions(obj, ...)

Arguments

obj object with mcr case
...
other parameters

plotContributions.mcr

Show plot with resolved contributions

Description

Show plot with resolved contributions

Usage

## S3 method for class 'mcr'
plotContributions(
  obj,
  comp = seq_len(obj$ncomp),
  type = "l",
  col = mdaplot.getColors(obj$ncomp),
  ...
)
plotConvexHull

Arguments

- **obj**: object of class `mcr`
- **comp**: vector with number of components to make the plot for
- **type**: type of the plot
- **col**: vector with colors for individual components
- **...**: other parameters suitable for `mdaplotg`

Description

The method shows convex hull for groups of points on a scatter plot made using `mdaplot()` function with `cgroup` parameter. It will work only if `cgroup` is a factor.

Usage

```r
plotConvexHull(p, lwd = 1, lty = 1, opacity = 0)
```

Arguments

- **p**: plot data returned by function `mdaplot()`. 
- **lwd**: thickness of line used to show the hull.
- **lty**: type of line used to show the hull.
- **opacity**: of opacity is larger than 0 a semi-transparent polygon is shown over points.

Examples

```r
# adds convex hull with semi-transparent area over two clusters of points

library(mdatools)
data(people)
group <- factor(people[, "Sex"], labels = c("Male", "Female"))

p <- mdaplot(people, type = "p", cgroup = group)
plotConvexHull(p)
```
**plotCooman**  
*Cooman’s plot*

**Description**

Cooman’s plot

**Usage**

```
plotCooman(obj, ...)
```

**Arguments**

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

**Details**

Generic function for Cooman’s plot

---

**plotCooman.simcam**  
*Cooman’s plot for SIMCAM model*

**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),
  res = list(cal = obj$res[["cal"]]),
  groupby = res[[1]]$c.ref,
  main = "Cooman's plot",
  show.limits = TRUE,
  ...
)
```

```
plotCooman.simcamres

Arguments

obj    a SIMCAM model (object of class simcam)
nc     vector with two values - classes (SIMCA models) to show the plot for
res    list with results to show the plot for
groupby factor to use for grouping points on the plot
main   title of the plot
show.limits logical, show or not critical limits
...    other plot parameters (see mdaplotg for details)

Details

Cooman’s plot shows squared orthogonal distance from data points to two selected SIMCA models as well as critical limits for the distance (optional). In case if critical limits must be shown they are computed using chi-square distribution regardless which type of limits is employed for classification.

If only one result object is provided (e.g. results for calibration set or new predictions), then the points can be color grouped using 'groupby' parameter (by default reference class values are used to make the groups). In case of multiple result objects, the points are color grouped according to the objects (e.g. calibration set and test set).

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),
    main = "Cooman's plot",
    cgroup = obj$c.ref,
    show.plot = TRUE,
    ...
)
```
Arguments

- **obj**: SIMCAM results (object of class `simcamres`)
- **nc**: vector with two values - classes (SIMCA models) to show the plot for
- **main**: main plot title
- **cgroup**: vector of values to use for color grouping of plot points
- **show.plot**: logical, show plot or just return plot data
- **...**: other plot parameters (see `mdaplotg` for details)

Details

The plot is similar to `plotCooman.simcam` but shows points only for this result object and does not show critical limits (which are part of a model).
**plotCumVariance**

**Description**

Variance plot

**Usage**

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

**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, 
  ncomp = obj$ncomp.selected, 
  ylim = NULL, 
  xlab = expression(r^2), 
  ylab = "Test statistic", 
  ...
)
```

**Arguments**

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

**Details**

See examples in help for `randtest` function.
plotCumVariance.ldecomp

Arguments

  obj  a model or result object
  ... other arguments

Details

Generic function for plotting explained variance for data decomposition

Description

Show a plot with cumulative explained variance vs. number of components.

Usage

## S3 method for class 'ldecomp'
plotCumVariance(obj, type = "b", labels = "values", show.plot = TRUE, ...)

Arguments

  obj  object of ldecomp class.
  type type of the plot
  labels what to show as labels for plot objects
  show.plot logical, shall plot be created or just plot series object is needed
  ... most of graphical parameters from mdaplot function can be used.

plotCumVariance.mcr  Show plot with cumulative explained variance

Description

Show plot with cumulative explained variance

Usage

## S3 method for class 'mcr'
plotCumVariance(
  obj,
  type = "b",
  labels = "values",
  main = "Cumulative variance",
  xticks = seq_len(obj$ncomp),
  ...
)
Arguments

obj object of class mcr
type type of the plot
labels what to use as data labels
main title of the plot
xticks vector with ticks for x-axis
... other parameters suitable for mdaPlot

plotCumVariance.pca  Cumulative explained variance plot for PCA model

Description

Shows a plot with cumulative explained variance for components.

Usage

## S3 method for class 'pca'
plotCumVariance(obj, legend.position = "bottomright", ...)

Arguments

obj a PCA model (object of class pca)
legend.position position of the legend
... other plot parameters (see mdaPlot for details)

Details

See examples in help for pca function.

plotDensity  Show plot series as density plot (using hex binning)

Description

Show plot series as density plot (using hex binning)

Usage

plotDensity(ps, nbins = 60, colmap = ps$colmap)
plotDiscriminationPower

**Arguments**

- `ps`: 'plotseries' object
- `nbins`: number of bins in one dimension
- `colmap`: colormap name or values used to create color gradient

**Description**

Discrimination power plot

**Usage**

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

**Arguments**

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

**Details**

Generic function for plotting discrimination power values for classification model

plotDiscriminationPower.simcam

**Description**

Discrimination power plot for SIMCAM model

**Usage**

```
## S3 method for class 'simcam'
plotDiscriminationPower(
  obj,
  nc = c(1, 2),
  type = "h",
  main = paste0("Discrimination power: ", obj$classnames[nc[1]], " vs. ", obj$classnames[nc[2]]),
  xlab = attr(obj$dispower, "xaxis.name"),
  ylab = "",
  ...
)
```
plotDistDoF

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

Discrimination power shows an ability of variables to separate classes. The power is computed similar to model distance, using variance of residuals. However in this case instead of sum the variance across all variables, we take the ratio separately for individual variables.

Discrimination power equal or above 3 is considered as high.

---

**plotDistDoF**

*Degrees of freedom plot for both distances*

Description

Shows a plot with degrees of freedom computed for score and orthogonal distances at given number of components using data driven approach ("ddmoments" or "ddrobust").

Usage

```r
plotDistDoF(
  obj,
  type = "b",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  ...
)
```

Arguments

- **obj** a PCA model (object of class `pca`)
- **type** type of the plot ("b", "l", "h")
- **labels** what to show as data points labels
- **xticks** vector with tick values for x-axis
- **...** other plot parameters (see `mdaplotg` for details)

Details

Work only if parameter `lim.type` equal to "ddmoments" or "ddrobust".
plotErrorbars

Description
It is assumed that first row of dataset contains the y-coordinates of points, second rows contains size of lower error bar and third - size for upper error bar. If only two rows are provided it is assumed that error bars are symmetric.

Usage
plotErrorbars(ps, col = ps$col, pch = 16, lwd = 1, cex = 1, ...)

Arguments
- ps: ‘plotseries’ object
- col: color for the error bars
- pch: marker symbol for the plot
- lwd: line width for the error bars
- cex: scale factor for the marker
- ...: other arguments for function ‘points()’.

plotExtreme

Description
Generic function for creating extreme plot for SIMCA model

Usage
plotExtreme(obj, ...)

Arguments
- obj: a SIMCA model
- ...: other parameters
### Description

Shows a plot with number of expected vs. number of observed extreme objects for different significance levels (alpha values).

### Usage

```r
## S3 method for class 'pca'
plotExtreme(
  obj,
  res = obj$res[["cal"]],
  comp = obj$ncomp.selected,
  main = "Extreme plot",
  xlab = "Expected",
  ylab = "Observed",
  pch = rep(21, length(comp)),
  bg = mdaplot.getColors(length(comp)),
  col = rep("white", length(comp)),
  lwd = ifelse(pch %in% 21:25, 0.25, 1),
  cex = rep(1.2, length(comp)),
  ellipse.col = "#cceeef",
  legend.position = "bottomright",
  ...
)
```

### Arguments

- `obj` a PCA model (object of class `pca`)
- `res` object with PCA results to show the plot for (e.g. calibration, test, etc)
- `comp` vector, number of components to show the plot for
- `main` plot title
- `xlab` label for x-axis
- `ylab` label for y-axis
- `pch` vector with values for `pch` parameter for each number of components
- `bg` vector with background color values for series of points (if `pch`=21:25)
- `col` vector with color values for series of points
- `lwd` line width for point symbols
- `cex` scale factor for data points
- `ellipse.col` color for tolerance ellipse
- `legend.position` position of the legend
- `...` other arguments
**plotHist**

Statistic histogram

**Description**

Statistic histogram

**Usage**

plotHist(obj, ...)

**Arguments**

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

**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, ncomp = obj$ncomp.selected, bwd = 0.9, ...)

**Arguments**

- **obj** results of randomization test (object of class 'randtest')
- **ncomp** number of component to make the plot for
- **bwd** width of bars (between 0 and 1)
- **...** other optional arguments

**Details**

See examples in help for randtest function.
### Description

Add Hotelling ellipse to a scatter plot

### Usage

```r
plotHotellingEllipse(p, conf.lim = 0.95, col = "#a0a0a0", lty = 3, ...)
```

### Arguments

- `p`: plot series (e.g. from PCA scores plot)
- `conf.lim`: confidence limit
- `col`: color of the ellipse line
- `lty`: line type (e.g. 1 for solid, 2 for dashed, etc.)
- `...`: any argument suitable for `lines` function

### Details

The method is created to be used with PCA and PLS scores plots, so it shows the statistical limits computed using Hotelling $T^2$ distribution in form of ellipse. The function works similar to `plotConvexHull` and `plotConfidenceEllipse` but does not require grouping of data points. Can be used together with functions `plotScores.pca`, `plotScores.ldecomp`, `plotXScores.pls`, `plotXScores.plsres`.

See examples for more details.

### Examples

```r
# create PCA model for People data
data(people)
m <- pca(people, 4, scale = TRUE)

# make scores plot and show Hotelling ellipse with default settings
p <- plotScores(m, xlim = c(-8, 8), ylim = c(-8, 8))
plotHotellingEllipse(p)

# make scores plot and show Hotelling ellipse with manual settings
p <- plotScores(m, xlim = c(-8, 8), ylim = c(-8, 8))
plotHotellingEllipse(p, conf.lim = 0.99, col = "red")

# in case if you have both calibration and test set, 'plotScores()' returns
# plot series data for both, so you have to subset it and take the first series
# (calibration set) as shown below.
ind <- seq(1, 32, by = 4)
```
xc <- people[-ind, , drop = FALSE]
x <- people[ind, , drop = FALSE]
m <- pca(xc, 4, scale = TRUE, x.test = xt)

p <- plotScores(m, xlim = c(-8, 8), ylim = c(-8, 8))
plotHotellingEllipse(p[[1]])

---

**plotLines**

Show plot series as set of lines

**Description**

Show plot series as set of lines

**Usage**

```r
plotLines(
  ps,
  col = ps$col,
  lty = 1,
  lwd = 1,
  cex = 1,
  col.excluded = "darkgray",
  show.excluded = FALSE,
  ...
)
```

**Arguments**

- `ps` : 'plotseries' object
- `col` : a color for markers or lines (same as `plot` parameter).
- `lty` : line type
- `lwd` : line width
- `cex` : scale factor for the marker
- `col.excluded` : color for the excluded lines.
- `show.excluded` : logical, show or not the excluded data points
- `...` : other arguments for function 'lines()'.

plotLoadings  

*Loadings plot*

**Description**

Loadings plot

**Usage**

```r
plotLoadings(obj, ...)```

**Arguments**

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

**Details**

Generic function for plotting loadings values for data decomposition

---

plotLoadings.pca  

*Loadings plot for PCA model*

**Description**

Shows a loadings plot for selected components.

**Usage**

```r
## S3 method for class 'pca'
plotLoadings(
    obj,
    comp = c(1, 2),
    type = (if (length(comp == 2)) "p" else "l"),
    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')
- **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

**Misclassification ratio plot**

**Description**

Misclassification ratio plot

**Usage**

```r
plotMisclassified(obj, ...)
```

**Arguments**

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

**Details**

Generic function for plotting misclassification values for classification model or results
plotMisclassified.classmodel

*Misclassified ratio plot for classification model*

**Description**

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

**Usage**

```r
## S3 method for class 'classmodel'
plotMisclassified(obj, ...)
```

**Arguments**

- `obj` classification model (object of class `plsda`, `simca`, etc.).
- `...` parameters for `plotPerformance.classmodel` function.

**Details**

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

plotMisclassified.classres

*Misclassified ratio plot for classification results*

**Description**

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

**Usage**

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

**Arguments**

- `obj` classification results (object of class `plsdares`, `simcamres`, etc.).
- `...` other parameters for `plotPerformance.classres`

**Details**

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

*Model distance plot*

**Description**

Model distance plot

**Usage**

`plotModelDistance(obj, ...)`

**Arguments**

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

**Details**

Generic function for plotting distance from object to a multivariate model

---

```r
## S3 method for class 'simcam'
plotModelDistance

plotModelDistance.simcam  

*Model distance plot for SIMCAM model*

**Description**

Shows a plot with distance between one SIMCA model to others.

**Usage**

```r
## S3 method for class 'simcam'
plotModelDistance(
  obj,
  nc = 1,
  type = "h",
  xticks = seq_len(obj$nclasses),
  xticklabels = obj$classnames,
  main = paste0("Model distance (", obj$classnames[nc], ")"),
  xlab = "Models",
  ylab = "",
  ...
)
```
Arguments

- **obj**: a SIMCAM model (object of class `simcam`)
- **nc**: one value - number of class (SIMCA model) to show the plot for
- **type**: type of the plot ("h", "l" or "b")
- **xticks**: vector with tick values for x-axis
- **xticklabels**: vector with tick labels for x-axis
- **main**: main plot title
- **xlab**: label for x axis
- **ylab**: label for y axis
- **...**: other plot parameters (see `mdaplotg` for details)

Details

The plot shows similarity between a selected model and the others as a ratio of residual variance using the following algorithm. Let’s take two SIMCA/PCA models, m1 and m2, which have optimal number of components A1 and A2. The models have been calibrated using calibration sets X1 and X2 with number of rows n1 and n2. Then we do the following:

1. Project X2 to model m1 and compute residuals, E12
2. Compute variance of the residuals as s12 = sum(E1^2) / n1
3. Project X1 to model m2 and compute residuals, E21
4. Compute variance of the residuals as s21 = sum(E2^2) / n2
5. Compute variance of residuals for m1 as s1 = sum(E1^2) / (n1 - A1 - 1)
6. Compute variance of residuals for m2 as s2 = sum(E2^2) / (n2 - A2 - 1)

The model distance then can be computed as: \( d = \sqrt{\frac{\text{s12} + \text{s21}}{\text{s1} + \text{s2}}} \)

As one can see, if the two models and corresponding calibration sets are identical, then the distance will be \( \sqrt{\frac{n - A - 1}{n}} \). For example, if n = 25 and A = 2, then the distance between the model and itself is \( \sqrt{\frac{22}{25}} = \sqrt{0.88} = 0.938 \). This case is demonstrated in the example section.

In general, if distance between models is below one classes are overlapping. If it is above 3 the classes are well separated.

Examples

```r
# create two calibration sets with n = 25 objects in each
data(iris)
x1 <- iris[1:25, 1:4]
x2 <- iris[51:75, 1:4]

# create to SIMCA models with A = 2
m1 <- simca(x1, 'setosa', ncomp = 2)
m2 <- simca(x2, 'versicolor', ncomp = 2)

# combine the models into SIMCAM class
m <- simcam(list(m1, m2))
```
# show the model distance plot with distance values as labels
# note, that distance between setosa and setosa is 0.938
plotModelDistance(m, show.labels = TRUE, labels = "values")

---

### 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

---

### plotPerformance  
*Classification performance plot*

**Description**

Classification performance plot

**Usage**

```r
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

```r
## S3 method for class 'classmodel'
plotPerformance(
  obj,
  nc = 1,
  param = "misclassified",
  type = "b",
  labels = "values",
  ylab = "",
  ylim = c(0, 1.15),
  xticks = seq_len(dim(obj$res$cal$c.pred)[2]),
  res = obj$res,
  ...
)
```

Arguments

- **obj**: classification model (object of class plsda, simca, etc.).
- **nc**: class number to make the plot for.
- **param**: which parameter to make the plot for ("specificity", "sensitivity", or "misclassified")
- **type**: type of the plot
- **labels**: what to show as labels for plot objects.
- **ylab**: label for y axis
- **ylim**: vector with two values - limits for y axis
- **xticks**: vector with tick values for x-axis
- **res**: list with result objects to show the plot for
- **...**: most of the graphical parameters from mdaplotg function can be used.
plotPerformance.classres

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

## S3 method for class 'classres'
plotPerformance(
  obj,
  nc = 1,
  type = "b",
  param = c("sensitivity", "specificity", "misclassified"),
  labels = "values",
  ylab = "",
  ylim = c(0, 1.1),
  xticks = seq_len(obj$ncomp),
  show.plot = TRUE,
  ...
)

Arguments

- **obj** classification results (object of class plsdares, simcamres, etc.).
- **nc** if there are several classes, which class to make the plot for.
- **type** type of the plot
- **param** which performance parameter to make the plot for (can be a vector with several values).
- **labels** what to show as labels for plot objects.
- **ylab** label for y axis
- **ylim** vector with two values - limits for y axis
- **xticks** vector with x-axis tick values
- **show.plot** logical, shall plot be created or just plot series object is needed
- **...** most of the graphical parameters from mdaplot function can be used.

Details

See examples in description of plsdares, simcamres, etc.
**plotPointsShape**  
*Add confidence ellipse or convex hull for group of points*

**Description**
Add confidence ellipse or convex hull for group of points

**Usage**
```r
plotPointsShape(p, lwd, lty, opacity, shape_function, ...)
```

**Arguments**
- `p`: plot data returned by function `mdaplot`
- `lwd`: thickness of line used to show the hull
- `lty`: type of line used to show the hull
- `opacity`: if opacity is larger than 0 a semi-transparent polygon is shown over points
- `shape_function`: function which calculates and return coordinates of the shape
- `...`: extra parameters for `shape_function`

**plotPredictions**  
*Predictions plot*

**Description**
Predictions plot

**Usage**
```r
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
Description

Makes a plot with class predictions for a classification model.

Usage

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

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.name**: name of result object to make the plot for ("test", "cv" or "cal").
- **nc**: vector with class numbers to make the plot for.
- **ncomp**: what number of components to make the plot for.
- **main**: title of the plot (if NULL will be set automatically)
- **...**: most of the graphical parameters from `mdaplotg` function can be used.

Details

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

Description

Makes a plot with predicted class values for classification results.
plotPredictions.regmodel

Usage

## S3 method for class 'regmodel'
plotPredictions(
  obj,
  ncomp = obj$ncomp.selected,
  ny = 1,
  legend.position = "topleft",
  show.line = TRUE,
  res = obj$res,
  ...
)

Arguments

obj classification results (object of class plsdares, simcamres, etc.).
nc vector with classes to show predictions for.
ncomp model complexity (number of components) to make the plot for.
ylab label for y axis
show.plot logical, shall plot be created or just plot series object is needed
...
most of the graphical parameters from mdaplotg or mdaplot function can be used.

Details

See examples in description of plsdares, simcamres, etc.

Description

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

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>a regression model (object of class <code>regmodel</code>)</td>
</tr>
<tr>
<td>ncomp</td>
<td>how many components to use (if NULL - user selected optimal value will be used)</td>
</tr>
<tr>
<td>ny</td>
<td>number of response variable to make the plot for (if y is multivariate)</td>
</tr>
<tr>
<td>legend.position</td>
<td>position of legend on the plot (if shown)</td>
</tr>
<tr>
<td>show.line</td>
<td>logical, show or not line fit for the plot points</td>
</tr>
<tr>
<td>res</td>
<td>list with result objects</td>
</tr>
<tr>
<td>...</td>
<td>other plot parameters (see mdaplotg for details)</td>
</tr>
</tbody>
</table>

---

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 = obj$ncomp.selected,  
  show.line = TRUE,  
  show.stat = FALSE,  
  stat.col = "#606060",  
  stat.cex = 0.85,  
  xlim = NULL,  
  ylim = NULL,  
  axes.equal = TRUE,  
  show.plot = TRUE,  
  ...
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>regression results (object of class <code>regres</code>)</td>
</tr>
<tr>
<td>ny</td>
<td>number of predictor to show the plot for (if y is multivariate)</td>
</tr>
<tr>
<td>ncomp</td>
<td>complexity of model (e.g. number of components) to show the plot for</td>
</tr>
<tr>
<td>show.line</td>
<td>logical, show or not line fit for the plot points</td>
</tr>
<tr>
<td>show.stat</td>
<td>logical, show or not legend with statistics on the plot</td>
</tr>
</tbody>
</table>
plotPredictions.simcam

Predictions plot for SIMCAM model

Description

Makes a plot with class predictions for calibration dataset.

Usage

```r
## S3 method for class 'simcam'
plotPredictions(
  obj,
  nc = seq_len(obj$nclasses),
  main = "SIMCAM Predictions (cal)",
  ...,
)
```

Arguments

- `obj` a SIMCAM model (object of class simcam)
- `nc` vector with class numbers to make the plot for.
- `main` plot title.
- `...` most of the graphical parameters from `mdaplotg` function can be used.

Details

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

stat.col color of text in legend with statistics
stat.cex size of text in legend with statistics
xlim limits for x-axis (if NULL will be computed automatically)
ylim limits for y-axis (if NULL will be computed automatically)
axes.equal logical, make limits for x and y axes equal or not
show.plot logical, show plot or just return plot data
... other plot parameters (see `mdaplotg` 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.
plotPredictions.simcamres

*Prediction plot for SIMCAM results*

**Description**

Makes a plot with predicted class values for classification results.

**Usage**

```r
## S3 method for class 'simcamres'
plotPredictions(obj, nc = seq_len(obj$nclasses), main = "Predictions", ...)
```

**Arguments**

- `obj` classification results (object of class `plsdares`, `simcamres`, etc.).
- `nc` vector with classes to show predictions for.
- `main` title of the plot
- `...` most of the graphical parameters from `mdaplotg` or `mdaplot` function can be used.

**Details**

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

---

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**

```r
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", 
  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
- `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.

---

plotPurity

*Plot purity values*

**Description**

Plot purity values

**Usage**

```r
plotPurity(obj, ...)
```
**Arguments**

- `obj` object with mcr pure case
- `...` other parameters

---

**Description**

Purity values plot

**Usage**

```r
## S3 method for class 'mcrpure'
plotPurity(
  obj,
  xticks = seq_len(obj$ncomp),
  type = "h",
  labels = "values",
  ...
)
```

**Arguments**

- `obj` mcrpure object
- `xticks` ticks for x axis
- `type` type of the plot
- `labels` what to use as data labels
- `...` other parameters suitable for `mdaplot`

The plot shows largest weighted purity value for each component graphically.

---

**plotPuritySpectra**  
*Plot purity spectra*

**Description**

Plot purity spectra

**Usage**

```r
plotPuritySpectra(obj, ...)
```

**Arguments**

- `obj` object with mcr pure case
- `...` other parameters
plotPuritySpectra.mcrpure

Purity spectra plot

Description

Purity spectra plot

Usage

## S3 method for class 'mcrpure'
plotPuritySpectra(
  obj, comp = seq_len(obj$ncomp),
  type = "l",
  col = mdaplot.getColors(obj$ncomp),
  show.lines = TRUE,
  lines.col = adjustcolor(col, alpha.f = 0.75),
  lines.lty = 3,
  lines.lwd = 1,
  ...
)

Arguments

obj mcrpure object
comp vector of components to show the purity spectra for
type type of the plot
col colors for the plot (should be a vector with one value for each component in obj)
show.lines if TRUE show the selected pure variables as vertical lines
lines.col color for the selected pure variable lines (by default same as for plots but semi-transparent)
lines.lty line type for the purity lines
lines.lwd line width for the purity lines
... other parameters suitable for mdaplot

The plot shows weighted purity value of each variable separately for each specified component.
plotQDoF

Degrees of freedom plot for orthogonal distance (Nh)

Description

Shows a plot with degrees of freedom computed for score distances at given number of components using data driven approach ("ddmoments" or "ddrobust").

Usage

plotQDoF(
  obj,
  type = "b",  # type of the plot ("b", "l", "h")
  labels = "values",  # what to show as data points labels
  xticks = seq_len(obj$ncomp),  # vector with tick values for x-axis
  ylab = "Nq",  # label for y-axis
  ...  # other plot parameters (see mdaplotg for details)
)

Arguments

obj a PCA model (object of class pca)
type type of the plot ("b", "l", "h")
labels what to show as data points labels
xticks vector with tick values for x-axis
ylab label for y-axis
... other plot parameters (see mdaplotg for details)

Details

Work only if parameter lim.type equal to "ddmoments" or "ddrobust".

plotRegcoeffs

Regression coefficients plot

Description

Regression coefficients plot

Usage

plotRegcoeffs(obj, ...)

plotRegressionLine

Arguments

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

Details

Generic function for plotting regression coefficients values for a regression model

plotRegcoeffs.regmodel

Regression coefficient plot for regression model

Description

Shows plot with regression coefficient values. Is a proxy for link(plot.regcoeffs) method.

Usage

## S3 method for class 'regmodel'
plotRegcoeffs(obj, ncomp = obj$ncomp.selected, ...)

Arguments

- obj: a regression model (object of class regmodel)
- ncomp: number of components to show the plot for
- ...: other plot parameters (see link(plot.regcoeffs) for details)

plotRegressionLine

Add regression line for data points

Description

Shows linear fit line for data points.

Usage

plotRegressionLine(p, col = p$col, ...)

Arguments

- p: plot data returned by function ‘mdaplot()’
- col: color of line
- ...: other parameters available for ‘abline()’ function
**plotResiduals**

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.ldecomp**

Residual distance plot

**Description**

Shows a plot with orthogonal (Q, q) vs. score (T2, h) distances for data objects.

**Usage**

```r
## S3 method for class 'ldecomp'
plotResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  norm = FALSE,
  log = FALSE,
  show.labels = FALSE,
  labels = "names",
  show.plot = TRUE,
  ...
)
```

---
plotResiduals.pca

Arguments

obj  object of ldecomp class.
ncomp number of components to show the plot for (if NULL, selected by model value will be used).
norm logical, normalize distance values or not (see details)
log logical, apply log transformation to the distances or not (see details)
show.labels logical, show or not labels for the plot objects
labels what to show as labels if necessary
show.plot logical, shall plot be created or just plot series object is needed
...

plotResiduals.pca  Residuals distance plot for PCA model

Description

Shows a plot with score (T2, h) vs orthogonal (Q, q) distances and corresponding critical limits for given number of components.

Usage

## S3 method for class 'pca'
plotResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  log = FALSE,
  norm = TRUE,
  cgroup = NULL,
  xlim = NULL,
  ylim = NULL,
  show.limits = TRUE,
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  lim.lty = c(2, 3),
  res = obj$res,
  show.legend = TRUE,
  ...
)

Arguments

obj  a PCA model (object of class pca)
ncomp  how many components to use (by default optimal value selected for the model will be used)
plotResiduals.regres

Description

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

log
logical, apply log tranformation to the distances or not (see details)
norm
logical, normalize distance values or not (see details)
cgroup
color grouping of plot points (works only if one result object is available)
xlim
limits for x-axis
ylim
limits for y-axis
show.limits
logical, show or not lines/curves with critical limits for the distances
lim.col
vector with two values - line color for extreme and outlier limits
lim.lwd
vector with two values - line width for extreme and outlier limits
lim.lty
vector with two values - line type for extreme and outlier limits
res
list with result objects to show the plot for (by defaul, model results are used)
show.legend
logical, show or not a legend on the plot (needed if several result objects are available)

Details

The function is a bit more advanced version of plotResiduals.ldecomp. It allows to show distance values for several result objects (e.g. calibration and test set or calibration and new prediction set) as well as display the corresponding critical limits in form of lines or curves.

Depending on how many result objects your model has or how many you specified manually, using the res parameter, the plot behaves in a bit different way.

If only one result object is provided, then it allows to colorise the points using cgroup parameter. If you specify cgroup = "categories" then it will show points as three groups: normal, extreme and outliers. If two or more result objects are provided, then the function show distances in groups, and adds corresponding legend.

The function can show distance values normalised (h/h0 and q/q0) as well as with log transformation (log(1 + h/h0), log(1 + q/q0)). The latter is useful if distribution of the points is skewed and most of them are densely located around bottom left corner.

See examples in help for pca function.
plotRMSE

Usage

```r
## S3 method for class 'regres'
plotResiduals(
  obj,
  ny = 1,
  ncomp = obj$ncomp.selected,
  show.lines = c(NA, 0),
  show.plot = TRUE,
  ...
)
```

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.lines`: allows to show the horizontal line at y = 0
- `show.plot`: logical, show plot or just return plot data
- `...`: other plot parameters (see `mdaplot` for details)

Description

RMSE plot

Usage

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

Arguments

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

Details

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

Shows how RMSE develops for each iteration of iPLS selection algorithm

Usage

```r
## S3 method for class 'ipls'
plotRMSE(
  obj,
  glob.ncomp = obj$gm$ncomp.selected,
  main = "RMSE development",
  xlab = "Iterations",
  ylab = if (is.null(obj$cv)) "RMSEP" else "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.regmodel  \hspace{1cm} \textit{RMSE plot for regression model}

\textbf{Description}

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

\textbf{Usage}

```r
## S3 method for class 'regmodel'
plotRMSE(
  obj,
  ny = 1,
  type = "b",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  res = obj$res,
  ylab = paste0("RMSE (", obj$res$cal$respnames[ny], ")"),
  ...
)
```

\textbf{Arguments}

- \textbf{obj}  a regression model (object of class \texttt{regmodel})
- \textbf{ny}  number of response variable to make the plot for (if y is multivariate)
- \textbf{type}  type of the plot("b", "l" or "h")
- \textbf{labels}  what to show as labels (vector or name, e.g. "names", "values", "indices")
- \textbf{xticks}  vector with ticks for x-axis values
- \textbf{res}  list with result objects
- \textbf{ylab}  label for y-axis
- \textbf{...}  other plot parameters (see \texttt{mdaplotg} for details)

plotRMSE.regres  \hspace{1cm} \textit{RMSE plot for regression results}

\textbf{Description}

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

Usage

```r
## S3 method for class 'regres'
plotRMSE(
  obj,
  ny = 1,
  type = "b",
  xticks = seq_len(obj$ncomp),
  labels = "values",
  show.plot = TRUE,
  ylab = paste0("RMSE (", obj$resnames[ny], ")"),
  ...
)
```

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
- `xticks`: vector with ticks for x-axis
- `labels`: what to use as labels ("names", "values" or "indices")
- `show.plot`: logical, show plot or just return plot data
- `ylab`: label for y-axis
- `...`: other plot parameters (see `mdaplot` for details)

---

plotScatter  

Show plot series as set of points

Description

Show plot series as set of points

Usage

```r
plotScatter(
  ps,
  pch = 16,
  col = ps$col,
  bg = "white",
  lwd = 1,
  cex = 1,
  col.excluded = "lightgray",
  pch.colinv = FALSE,
  show.excluded = FALSE,
  ...)
```
plotScores

Scores plot

Description

Scores plot

Usage

plotScores(obj, ...)

Arguments

obj a model or result object
...
other arguments

Details

Generic function for scores values for data decomposition
plotScores.ldecomp

Scores plot

Description

Shows a plot with scores values for data objects.

Usage

## S3 method for class 'ldecomp'
plotScores(
  obj,
  comp = c(1, 2),
  type = "p",
  show.axes = TRUE,
  show.plot = TRUE,
  ...
)

Arguments

obj object of ldecomp class.
comp which components to show the plot for (can be one value or vector with two values).
type type of the plot
show.axes logical, show or not a axes lines crossing origin (0,0)
show.plot logical, shall plot be created or just plot series object is needed
...
most of graphical parameters from mdaplot function can be used.

plotScores.pca

Scores plot for PCA model

Description

Shows a scores plot for selected components.

Usage

## S3 method for class 'pca'
plotScores(
  obj,
  comp = c(1, 2),
  type = "p",
  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
for
type        type of the plot ("p", "l", "b", "h")
show.axes   logical, show or not a axes lines crossing origin (0,0)
show.legend logical, show or not a legend on the plot
res         list with result objects to show the variance for
            other plot parameters (see mdaplotg for details)

Details

If plot is created only for one result object (e.g. calibration set), then the behaviour and all settings for the scores plot are identical to plotScores.ldecomp. In this case you can show scores as a scatter, line or bar plot for any number of components.

Otherwise (e.g. if model contains results for calibration and test set) the plot is a group plot created using mdaplotg method and only scatter plot can be used.

See examples in help for pca function.
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 = obj$gm$ncomp.selected,
  main = "iPLS results",
  xlab = obj$xaxis.name,
  ylab = if (is.null(obj$cv)) "RMSEP" else "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 `ipls` function.

@seealso `summary.ipls`, `plotRMSE.ipls`
plotSelectivityRatio  Selectivity ratio plot

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

Usage
plotSelectivityRatio(obj, ...)

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

plotSelectivityRatio.pls
Selectivity ratio plot for PLS model

Description
Computes and shows a plot for Selectivity ratio values for given number of components and response variable

Usage
## S3 method for class 'pls'
plotSelectivityRatio(obj, ny = 1, ncomp = obj$ncomp.selected, type = "l", ...)

Arguments
- obj: a PLS model (object of class pls)
- ny: which response to plot the values for (if y is multivariate), can be a vector.
- ncomp: number of components to count
- type: type of the plot
- ...: other plot parameters (see mdaplot for details)

Details
See vipscores for more details.
plotSensitivity

**Description**
Sensitivity plot

**Usage**
plotSensitivity(obj, ...)

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

**Details**
Generic function for plotting sensitivity values for classification model or results

---

plotSensitivity.classmodel

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

**Usage**
```r
## S3 method for class 'classmodel'
plotSensitivity(obj, legend.position = "bottomright", ...)
```

**Arguments**
- obj: classification model (object of class plsda, simca, etc.).
- legend.position: position of the legend (as in mdaplotg).
- ...: parameters for plotPerformance.classmodel function.

**Details**
See examples in description of plsda, simca or simcam.
plotSensitivity.classres

*Sensitivity plot for classification results*

**Description**

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

**Usage**

### S3 method for class 'classres'

```r
plotSensitivity(obj, legend.position = "bottomright", ...)
```

**Arguments**

- `obj` classification results (object of class plsdares, simcamres, etc.).
- `legend.position` position of the legend (as in mdaplotg).
- `...` other parameters for `plotPerformance.classres`

**Details**

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

---

plotseries

Create plot series object based on data, plot type and parameters

**Description**

The ‘plotseries’ object contains all necessary paremeters to create main plots from data values, including values for x and y, correct handling of excluded rows and columns, color grouping (if any), limits and labels.

If both ‘col’ and ‘cgroup’ are specified, ‘cgroup’ will be ignored.

Labels can be either provided by user or generated automatically based on values, names or indices of data rows and columns. If series is made for scatter plot 'type="p"’ then labels are required for each row of the original dataset. Otherwise (for line, bar and errobar plot) labels correspond to data columns (variables).

The object has the following plotting methods once created: `plotScatter plotLines plotBars plotDensity plotErrorbars`
Use `plotSpecificity` to create a specificity plot.

### Usage

```r
plotSpecificity(obj, ...)  
```

### Arguments

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

### Details

A 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**

```r
## S3 method for class 'classmodel'
plotSpecificity(obj, legend.position = "bottomright", ...)
```

**Arguments**

- `obj` classification model (object of class `plsda`, `simca`, etc.).
- `legend.position` position of the legend (as in `mdaplot`).
- `...` other parameters for `plotPerformance.classmodel` function.

**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, legend.position = "bottomright", ...)
```

**Arguments**

- `obj` classification results (object of class `plsdares`, `simcamres`, etc.).
- `legend.position` position of the legend (as in `mdaplot`).
- `...` other parameters for `plotPerformance.classres`
**plotSpectra**

*Plot resolved spectra*

---

**Details**

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

---

**Description**

Plot resolved spectra

**Usage**

```r
plotSpectra(obj, ...)
```

**Arguments**

- `obj`: object with mcr case
- `...`: other parameters

---

**plotSpectra.mcr**

*Show plot with resolved spectra*

---

**Description**

Show plot with resolved spectra

**Usage**

```r
## S3 method for class 'mcr'
plotSpectra(
  obj,
  comp = seq_len(obj$ncomp),
  type = "l",
  col = mdaplot.getColors(obj$ncomp),
  ...
)
```

**Arguments**

- `obj`: object of class `mcr`
- `comp`: vector with number of components to make the plot for
- `type`: type of the plot
- `col`: vector with colors for individual components
- `...`: other parameters suitable for `mdaplotg`
plotT2DoF

Degrees of freedom plot for score distance (Nh)

Description

Shows a plot with degrees of freedom computed for score distances at given number of components using data driven approach ("ddmoments" or "ddrobust").

Usage

plotT2DoF(
  obj,  
  type = "b",  
  labels = "values",  
  xticks = seq_len(obj$ncomp),  
  ylab = "Nh",  
  ...  
)

Arguments

- obj: a PCA model (object of class pca)
- type: type of the plot ("b", "l", "h")
- labels: what to show as data points labels
- xticks: vector with tick values for x-axis
- ylab: label for y-axis
- ...: other plot parameters (see mdaPlotg for details)

Details

Work only if parameter lim.type equal to "ddmoments" or "ddrobust".

plotVariance

Variance plot

Description

Variance plot

Usage

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

Description

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

Usage

## S3 method for class 'ldecomp'
plotVariance(
  obj,
  type = "b",
  variance = "expvar",
  labels = "values",
  xticks = seq_len(obj$ncomp),
  show.plot = TRUE,
  ylab = "Explained variance, %",
  ...
)

Arguments

obj object of ldecomp class.

type type of the plot

variance string, which variance to make the plot for ("expvar", "cumexpvar")

labels what to show as labels for plot objects.

xticks vector with ticks for x-axis

show.plot logical, shall plot be created or just plot series object is needed

ylab label for y-axis

... most of graphical parameters from mdaplot function can be used.
Description
Show plot with explained variance

Usage
```r
## S3 method for class 'mcr'
plotVariance(
  obj,
  type = "h",
  labels = "values",
  main = "Variance",
  xticks = seq_len(obj$ncomp),
  ...
)
```

Arguments
- `obj`  object of class `mcr`
- `type`  type of the plot
- `labels` what to use as data labels
- `main`  title of the plot
- `xticks`  vector with ticks for x-axis
- `...`  other parameters suitable for `mdaplot`

Description
Explained variance plot for PCA model

Usage
```r
## S3 method for class 'pca'
plotVariance(
  obj,
  type = "b",
  labels = "values",
  variance = "expvar",
  xticks = seq_len(obj$ncomp),
```
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj</td>
<td>a PCA model (object of class pca)</td>
</tr>
<tr>
<td>type</td>
<td>type of the plot (&quot;b&quot;, &quot;l&quot;, &quot;h&quot;)</td>
</tr>
<tr>
<td>labels</td>
<td>what to use as labels (if show.labels = TRUE)</td>
</tr>
<tr>
<td>variance</td>
<td>which variance to show</td>
</tr>
<tr>
<td>xticks</td>
<td>vector with ticks for x-axis</td>
</tr>
<tr>
<td>res</td>
<td>list with result objects to show the variance for</td>
</tr>
<tr>
<td>ylab</td>
<td>label for y-axis</td>
</tr>
<tr>
<td>...</td>
<td>other plot parameters (see mdatplotg for details)</td>
</tr>
</tbody>
</table>

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",
  labels = "values",
  res = obj$res,
  ylab = "Explained variance, %",
  ...
)
```
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")
labels    what to show as labels for plot objects.
res       list with result objects to show the plot for (by defaul, model results are used)
ylab      label for y-axis
...       other plot parameters (see mdaplot for details)

Details

See examples in help for pls function.

---

plotVariance.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'
plotVariance(obj, decomp = "xdecomp", variance = "expvar", ...)

Arguments

obj       PLS results (object of class plsres)
decomp    which decomposition to use ("xdecomp" or "ydecomp")
variance  which variance to use ("expvar", "cumexpvar")
...       other plot parameters (see mdaplot for details)

Details

See examples in help for plsres function.
plotVIPScores  

VIP scores plot

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  

VIP scores plot for PLS model

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, ncomp = obj$ncomp.selected, type = "l", ...)

Arguments

obj a PLS model (object of class pls)
ny which response to plot the values for (if y is multivariate), can be a vector.
ncomp number of components to count
type type of the plot...
other plot parameters (see mdaplot for details)

Details

See vipscores for more details.
plotWeights  

Plot for PLS weights

Description

Plot for PLS weights

Usage

plotWeights(obj, ...)

Arguments

obj          a model or result object
...          other arguments

Details

Generic function for weight plot

plotWeights.pls  

X loadings plot for PLS

Description

Shows plot with X loading values for selected components.

Usage

## S3 method for class 'pls'
plotWeights(
  obj,
  comp = 1,
  type = (if (nrow(obj$weights) < 20) "h" else "l"),
  show.axes = TRUE,
  show.legend = TRUE,
  ...
)

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
show.axes    logical, show or not a axes lines crossing origin (0,0)
show.legend  logical, show or not a legend
...          other plot parameters (see mdaplotg for details)
plotXCumVariance

Details

See examples in help for pls function.

plotXCumVariance X cumulative variance plot

Description

X cumulative variance plot

Usage

plotXCumVariance(obj, ...)

Arguments

obj a model or result object

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

## S3 method for class 'pls'
plotXCumVariance(obj, type = "b", main = "Cumulative variance (X)", ...)

Arguments

obj a PLS model (object of class pls)
type type of the plot("b", "l" or "h")
main title for the plot

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**

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

**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**

```r
plotXLoadings(obj, ...)
```

**Arguments**

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

**Details**

Generic function for plotting loadings values for decomposition of x data
plotXLoadings.pls          X loadings plot for PLS

Description

Shows plot with X loading values for selected components.

Usage

## S3 method for class 'Var'
plotXLoadings

obj, comp = c(1, 2), type = "p", show.axes = TRUE, show.legend = TRUE,

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
show.axes logical, show or not a axes lines crossing origin (0,0)
show.legend logical, show or not legend on the plot (when it is available)

Details

See examples in help for pls function.

plotXResiduals           X residuals plot

Description

X residuals plot

Usage

plotXResiduals(obj, ...)

# S3 method for class 'pl'
plotXResiduals

obj, comp = c(1, 2), type = "p", show.axes = TRUE, show.legend = TRUE,

Arguments

obj  a PLS model (object of class pl)
comp which components to show the plot for (one or vector with several values)
type  type of the plot
show.axes logical, show or not a axes lines crossing origin (0,0)
show.legend logical, show or not legend on the plot (when it is available)

Details

See examples in help for pl function.
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  Residual distance plot for decomposition of X data

Description

Shows a plot with orthogonal distance vs score distance for PLS decomposition of X data.

Usage

## S3 method for class 'pls'
plotXResiduals(
  obj,  
ncomp = obj$ncomp.selected,  
norm = TRUE,  
log = FALSE,  
main = sprintf("X-differences (ncomp = %d)", ncomp),  
cgroup = NULL,  
xlim = NULL,  
ylim = NULL,  
show.limits = c(TRUE, TRUE),  
lim.col = c("darkgray", "darkgray"),  
lim.lwd = c(1, 1),  
lim.lty = c(2, 3),  
show.legend = TRUE,  
legend.position = "topright",  
res = obj$res,  
...  
)

Arguments

obj   a PLS model (object of class pls)
ncomp how many components to use (by default optimal value selected for the model will be used)
norm logical, normalize distance values or not (see details)
log   logical, apply log tranformation to the distances or not (see details)
main  title for the plot
plotXResiduals.plsres

- `cgroup`: color grouping of plot points (works only if one result object is available)
- `xlim`: limits for x-axis
- `ylim`: limits for y-axis
- `show.limits`: vector with two logical values defining if limits for extreme and/or outliers must be shown
- `lim.col`: vector with two values - line color for extreme and outlier limits
- `lim.lwd`: vector with two values - line width for extreme and outlier limits
- `lim.lty`: vector with two values - line type for extreme and outlier limits
- `show.legend`: logical, show or not a legend on the plot (needed if several result objects are available)
- `legend.position`: position of legend (if shown)
- `res`: list with result objects to show the plot for (by default, model results are used)
- `...`: other plot parameters (see `mdaplotg` for details)

**Details**

The function is almost identical to `plotResiduals.pca`.

---

**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 = obj$ncomp.selected,
  norm = TRUE,
  log = FALSE,
  main = sprintf("X-distances (ncomp = %d)", ncomp),
  ...
)
```

**Arguments**

- `obj`: PLS results (object of class `plsres`)
- `ncomp`: how many components to use (if NULL - user selected optimal value will be used)
- `norm`: logical, normalize distance values or not (see details)
plotXScores

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),
  show.axes = T,
  main = "Scores (X)",
  res = obj$res,
  ...
)

log logical, apply log tranformation to the distances or not (see details)
main main title for the plot
... other plot parameters (see mdaplot for details)

Details

See examples in help for plsres function.
plotXScores.plsres

Arguments

- **obj**: a PLS model (object of class `pls`)
- **comp**: which components to show the plot for (one or vector with several values)
- **show.axes**: logical, show or not a axes lines crossing origin (0,0)
- **main**: main plot title
- **res**: list with result objects to show the plot for (by default, model results are used)
- **...**: other plot parameters (see `mdaplot` 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 = "Scores (X)", ...)
```

**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

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 = "Variance (X)", ...)

Arguments

obj a PLS model (object of class pls)
type type of the plot("b", "l" or "h")
main title for the plot
...
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**

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

**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.

---

plotXYLoadings  
*X loadings plot*

**Description**

X loadings plot

**Usage**

`plotXYLoadings(obj, ...)`

**Arguments**

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

**Details**

Generic function for plotting loadings values for decomposition of x and y data
plotXYLoadings.pls  
*XY loadings plot for PLS*

**Description**

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

**Usage**

```r
## S3 method for class 'pls'
plotXYLoadings(obj, comp = c(1, 2), show.axes = TRUE, ...)
```

**Arguments**

- `obj`: a PLS model (object of class `pls`)
- `comp`: which components to show the plot for (one or vector with several values)
- `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.

---

plotXYResiduals  
*Plot for XY-residuals*

**Description**

Plot for XY-residuals

**Usage**

```r
plotXYResiduals(obj, ...)
```

**Arguments**

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

**Details**

Generic function for XY-residuals plot
**plotXYResiduals.pls**  
*Residual XY-distance plot*

**Description**

Shows a plot with full X-distance (f) vs. orthogonal Y-distance (z) for PLS model results.

**Usage**

```r
## S3 method for class 'pls'
plotXYResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  norm = TRUE,
  log = FALSE,
  main = sprintf("XY-distances (ncomp = %d)", ncomp),
  cgroup = NULL,
  xlim = NULL,
  ylim = NULL,
  show.limits = c(TRUE, TRUE),
  lim.col = c("darkgray", "darkgray"),
  lim.lwd = c(1, 1),
  lim.lty = c(2, 3),
  show.legend = TRUE,
  legend.position = "topright",
  res = obj$res,
  ...
)
```

**Arguments**

- `obj`  
a PLS model (object of class pls)
- `ncomp`  
how many components to use (by default optimal value selected for the model will be used)
- `norm`  
logical, normalize distance values or not (see details)
- `log`  
logical, apply log tranformation to the distances or not (see details)
- `main`  
title for the plot
- `cgroup`  
color grouping of plot points (works only if one result object is available)
- `xlim`  
limits for x-axis
- `ylim`  
limits for y-axis
- `show.limits`  
vector with two logical values defining if limits for extreme and/or outliers must be shown
- `lim.col`  
vector with two values - line color for extreme and outlier limits
- `lim.lwd`  
vector with two values - line width for extreme and outlier limits
lim.lty vector with two values - line type for extreme and outlier limits
show.legend logical, show or not a legend on the plot (needed if several result objects are available)
legend.position position of legend (if shown)
res list with result objects to show the plot for (by default, model results are used)
... other plot parameters (see mdaplotg for details)

Details

The function presents a way to identify extreme objects and outliers based on both full distance for X-decomposition (known as f) and squared residual distance for Y-decomposition (z). The approach has been proposed in [1].

The plot is available only if data driven methods (classic or robust) have been used for computing of critical limits.

References


plotXYResiduals.plsres

Residual distance plot

Description

Shows a plot with orthogonal (Q, q) vs. score (T2, h) distances for data objects.

Usage

```r
## S3 method for class 'plsres'
plotXYResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  norm = TRUE,
  log = FALSE,
  show.labels = FALSE,
  labels = "names",
  show.plot = TRUE,
  ...
)
```
Arguments

- **obj**: object of `ldecomp` class.
- **ncomp**: number of components to show the plot for (if NULL, selected by model value will be used).
- **norm**: logical, normalize distance values or not (see details)
- **log**: logical, apply log transformation to the distances or not (see details)
- **show.labels**: logical, show or not labels for the plot objects
- **labels**: what to show as labels if necessary
- **show.plot**: logical, shall plot be created or just plot series object is needed
- **...**: most of graphical parameters from `mdaplot` function can be used.

---

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

---

plotXYScores.pls **XY scores plot for PLS**

Description

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

Usage

```r
## S3 method for class 'pls'
plotXYScores(obj, ncomp = 1, show.axes = T, res = obj$res, ...)
```
**Arguments**

- `obj` a PLS model (object of class `pls`)
- `ncomp` which component to show the plot for
- `show.axes` logical, show or not axes lines crossing origin (0,0)
- `res` list with result objects to show the plot for (by default, model results are used)
- `...` other plot parameters (see `mdaplot` 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**

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

**Arguments**

- `obj` PLS results (object of class `plsres`)
- `ncomp` which component to show the plot for
- `show.plot` logical, show plot or just return plot data
- `...` 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

## S3 method for class 'pls'
plotYCumVariance(obj, type = "b", main = "Cumulative variance (Y)", ...)

Arguments

obj a PLS model (object of class pls)
type type of the plot("b", "l" or "h")
main title for the plot
... 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 = "Cumulative variance (Y)", ...)
```

**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**

```r
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.plsres

Y residuals plot for PLS results

Description

Shows a plot with Y residuals vs reference Y values for selected component.

Usage

## S3 method for class 'plsres'
plotYResiduals(obj, ncomp = obj$ncomp.selected, ...)

Arguments

obj

PLS results (object of class plsres)

ncomp

how many components to use (if NULL - user selected optimal value will be used)

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

Details

Proxy for plotResiduals.regres function.

plotYResiduals.regmodel

Y residuals plot for regression model

Description

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

Usage

## S3 method for class 'regmodel'
plotYResiduals(
  obj,
  ncomp = obj$ncomp.selected,
  ny = 1,
  show.lines = c(NA, 0),
  res = obj$res,
  ...
)

Arguments

obj  a regression model (object of class `regmodel`)
ncomp how many components to use (if NULL - user selected optimal value will be used)
y number of response variable to make the plot for (if y is multivariate)
show.lines  allows to show the horizontal line at 0 level
res  list with result objects
... other plot parameters (see `mdaplotg` for details)

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

Description

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

Usage

```r
## S3 method for class 'pls'
plotYVariance(obj, type = "b", main = "Variance (Y)", ...)
```
plotYVariance.plsres

Arguments

  obj     a PLS model (object of class pls)
  type    type of the plot("b", "l" or "h")
  main    title for the plot
  ...     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

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

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.
Partial Least Squares regression

Description

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

Usage

```r
pls(
x,                   # matrix with predictors.
y,                   # matrix with responses.
ncomp = min(nrow(x) - 1, ncol(x), 20),
center = TRUE,        # logical, center or not predictors and response values.
scale = FALSE,        # logical, scale (standardize) or not predictors and response values.
cv = NULL,            # cross-validation settings (see details).
exclcols = NULL,      # columns of x to be excluded from calculations (numbers, names or vector with logical values)
exclrows = NULL,      # rows to be excluded from calculations (numbers, names or vector with logical values)
x.test = NULL,        # matrix with predictors for test set.
y.test = NULL,        # matrix with responses for test set.
method = "simpls",    # algorithm for computing PLS model (only 'simpls' is supported so far)
info = "",            # short text with information about the model.
ncomp.selcrit = "min",# maximum number of components to calculate.
lim.type = "ddmoments",# logical, center or not predictors and response values.
alpha = 0.05,         # logical, scale (standardize) or not predictors and response values.
gamma = 0.01)
```

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**: cross-validation settings (see details).
- **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).
- **info**: short text with information about the model.
ncomp.selcrit  criterion for selecting optimal number of components ('min' for first local minimum of RMSECV and 'wold' for Wold's rule.)
lim.type  which method to use for calculation of critical limits for residual distances (see details)
alpha  significance level for extreme limits for T2 and Q distances.
gamma  significance level for outlier limits for T2 and Q distances.

Details

So far only SIMPLS method [1] is available. 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 first local minimum, but can be also forced to user defined value using function (selectCompNum.pls). The selected optimal number of components is used for all default operations - predictions, plots, etc.

Cross-validation settings, cv, 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').

Calculation of confidence intervals and p-values for regression coefficients can by done based on Jack-Knifing resampling. This is done automatically if cross-validation is used. However it is recommended to use at least 10 segments for stable JK result. See help for regcoeffs objects for more 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.
xcenter  vector with values used to center the predictors (x).
ycenter  vector with values used to center the responses (y).
xscale  vector with values used to scale the predictors (x).
yscale  vector with values used to scale the responses (y).
xloadings  matrix with loading values for x decomposition.
yloadings  matrix with loading values for y decomposition.
xeigenvals  vector with eigenvalues of components (variance of x-scores).
yeigenvals  vector with eigenvalues of components (variance of y-scores).
weights  matrix with PLS weights.
coeffs  object of class regcoeffs with regression coefficients calculated for each component.
info  information about the model, provided by user when build the model.
cv  information cross-validation method used (if any).
res  a list with result objects (e.g. calibration, cv, etc.)

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

References

See Also
Main 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.
- setDistanceLimits.pls  allows to change parameters for critical limits.
- categorize.pls  categorize data rows similar to categorize.pca.
- selratio  computes matrix with selectivity ratio values.
- vipscores  computes matrix with VIP scores values.

Plotting methods for pls objects:

- 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.
- 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 y decomposition.
- plotYCumVariance.pls  shows cumulative explained variance plot for y decomposition.
- plotXResiduals.pls  shows distance/residuals plot for x decomposition.
- plotXYResiduals.pls  shows joint distance plot for x and y decomposition.
- plotWeights.pls  shows plot with weights.
- plotSelectivityRatio.pls  shows plot with selectivity ratio values.
- plotVIPScores.pls  shows plot with VIP scores values.

Methods inherited from regmodel object (parent class for pls):
plotPredictions.regmodel shows predicted vs. measured plot.
plotRMSE.regmodel shows RMSE plot.
plotYResiduals.regmodel shows residuals plot for y values.
getRegcoeffs.regmodel 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) and implementation of interval PLS algorithm for variable selection (ipls)

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$spectra.c
y = simdata$conc.c[, 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$spectra.c
y = simdata$conc.c[, 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$spectra.c
y = simdata$conc.c[, 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
```r
i = 1:ncol(x)
ms = apply(x, 2, mean)

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))
```

---

### Description

Calibrates (builds) a PLS model for given data and parameters

### Usage

```r
pls.cal(x, y, ncomp, center, scale, method = "simpls", cv = FALSE)
```

### Arguments

- **x**: a matrix with x values (predictors)
- **y**: a matrix with y values (responses)
- **ncomp**: number of components to calculate
- **center**: logical, do mean centering or not
- **scale**: logical, do standardization or not
- **method**: algorithm for computing PLS model (only 'simpls' is supported so far)
- **cv**: logical, is model calibrated during cross-validation or not (or cv settings for calibration)

### Value

- **model**: an object with calibrated PLS model
pls.getZLimits

*ppl* getZLimits

*Description*

Compute critical limits for orthogonal distances (Q)

*Usage*

```r
pls.getZLimits(lim.type, alpha, gamma, params)
```

*Arguments*

- **lim.type** which method to use for calculation of critical limits for residuals
- **alpha** significance level for extreme limits.
- **gamma** significance level for outlier limits.
- **params** distribution parameters returned by ldecomp.getLimParams

*Value*

list with two matrices (x and y coordinates of corresponding limits)
**pls.run**

*Runs selected PLS algorithm*

**Description**

Runs selected PLS algorithm

**Usage**

```r
pls.run(x, y, ncomp = min(nrow(x) - 1, ncol(x)), method = "simpls", cv = FALSE)
```

**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 of partial least squares discrimination analysis (PLS-DA) model.

Usage

```r
plsda(
  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 = min(nrow(x) - 1, ncol(x), 20),  # maximum number of components to calculate.
  center = TRUE,  # logical, center or not predictors and response values.
  scale = FALSE,  # logical, scale (standardize) or not predictors and response values.
  cv = NULL,  # cross-validation settings (see details).
  exclcols = NULL,  # columns of x to be excluded from calculations (numbers, names or vector with logical values)
  exclrows = NULL,  # rows to be excluded from calculations (numbers, names or vector with logical values)
  x.test = NULL,  # matrix with predictors for test set.
  c.test = NULL,
  method = "simpls",  # "simpls", "modpls", or "modplsm".
  lim.type = "ddmoments",  # logical, scale (standardize) or not predictors and response values.
  alpha = 0.05,  # logical, scale (standardize) or not predictors and response values.
  gamma = 0.01,  # logical, scale (standardize) or not predictors and response values.
  info = "",  # "",  # logical, scale (standardize) or not predictors and response values.
  ncomp.selcrit = "min",  # logical, scale (standardize) or not predictors and response values.
  classname = NULL  # logical, scale (standardize) or not predictors and response values.
)
```

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**: cross-validation settings (see details).
- **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.
lim.type which method to use for calculation of critical limits for residual distances (see details)
alpha significance level for extreme limits for T2 and Q distances.
gamma significance level for outlier limits for T2 and Q distances.
info short text with information about the model.
ncomp.selcrit criterion for selecting optimal number of components ('min' for first local minimum of RMSECV and 'wold' for Wold's rule.)
classname name (label) of class in case if PLS-DA is used for one-class discrimination model. In this case it is expected that parameter 'c' will be a vector with logical values.

Details

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).

Cross-validation settings, cv, 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').

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 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.

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 = "cal", ncomp = 2, nc = 2)
plotPredictions(structure(model, class = "regmodel"))
plotPredictions(structure(model, class = "regmodel"), 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))

---

**plsdares**

**PLS-DA results**

### Description

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

### Usage

`plsdares(plsres, cres)`

### Arguments

- **plsres** PLS results for the data.
- **cres** Classification results for the data.

### 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 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.
- `plotXScores.plsres` shows scores plot for x decomposition.
- `plotXYScores.plsres` shows scores plot for x and y decomposition.
- `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.plsres` shows residuals plot for y values.

Methods inherited from regres class (parent class for plsres):

- `plotPredictions.regres` shows predicted vs. measured plot.
- `plotRMSE.regres` shows RMSE plot.
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$calfres

# 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 = "regres"))
plotPredictions(structure(res, class = "regres"), ncomp = 2, ny = 2)
par(mfrow = c(1, 1))

## 4. All plots from ordinary PLS results can be used, e.g.:
```
### 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 = dim(y.pred)[2],
  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.
- `ncomp.selected` 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.
- `plotXScores.plsres` shows scores plot for x decomposition.
- `plotXYScores.plsres` shows scores plot for x and y decomposition.
- `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.plsres` shows residuals plot for y values.

Methods inherited from `regres` class (parent class for `plsres`):

- `plotPredictions.regres` shows predicted vs. measured plot.
- `plotRMSE.regres` shows RMSE plot.
See also **pls** - a class for PLS models.

**Examples**

```r
### Examples of using PLS result class
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
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 = 'Var', ylab = 'Var')
par(mfrow = c(1, 1))

---

### `predict.mcrals`

**MCR ALS predictions**

**Description**
Applies MCR-ALS model to a new set of spectra and returns matrix with contributions.

**Usage**
```r
## S3 method for class 'mcrals'
predict(object, x, ...)
```

**Arguments**
- `object` an MCR model (object of class `mcr`).
- `x` spectral values (matrix or data frame).
- `...` other arguments.

**Value**
Matrix with contributions

---

### `predict.mcrpure`

**MCR predictions**

**Description**
Applies MCR model to a new set of spectra and returns matrix with contributions.

**Usage**
```r
## S3 method for class 'mcrpure'
predict(object, x, ...)
```
Arguments

object: an MCR model (object of class mcr).

x: spectral values (matrix or data frame).

...: other arguments.

Value

Matrix with contributions

---

**predict.pca**  
*PCA predictions*

Description

Applies PCA model to a new data set.

Usage

```r
## S3 method for class 'pca'
predict(object, x, ...)  
```

Arguments

object: a PCA model (object of class pca).

x: data values (matrix or data frame).

...: 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, cv = FALSE, ...)  
```
predict.plsda

Arguments

object  a PLS model (object of class pls)
x       a matrix with x values (predictors)
y       a matrix with reference y values (responses)
cv      logical, shall predictions be made for cross-validation procedure or not
...     other arguments

Details

See examples in help for pls function.

Value

PLS results (an object of class plsres)

predict.plsda  PLS-DA predictions

Description

Applies PLS-DA model to a new data set

Usage

## 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, ...)

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)
- ...: other arguments
**prep**

**Details**

See examples in help for `simcam` function.

**Value**

SIMCAM results (an object of class `simcamres`)

---

**prep**

*Class for preprocessing object*

---

**Description**

Class for preprocessing object

**Usage**

```r
prep(name, params = NULL, method = NULL)
```

**Arguments**

- **name**: short text with name for the preprocessing method.
- **params**: a list with parameters for the method (if NULL - default parameters will be used).
- **method**: method to call when applying the preprocessing, provide it only for user defined methods.

**Details**

Use this class to create a list with a sequence of preprocessing methods to keep them together in right order and with defined parameters. The list/object can be provided as an extra argument to any modelling function (e.g. `pca`, `pls`, etc), so the optimal model parameters and the optimal preprocessing will be stored together and can be applied to a raw data by using method `predict`.

For your own preprocessing method you need to create a function, which takes matrix with values (dataset) as the first argument, does something and then return a matrix with the same dimension and same attributes as the result. The method can have any number of optional parameters.

See Bookdown tutorial for details.
Description

Baseline correction using asymmetric least squares

Usage

prep.alsbasecorr(data, plambda = 5, p = 0.1, max.niter = 10)

Arguments

data | matrix with spectra (rows correspond to individual spectra)
plambda | power of the penalty parameter (e.g. if plambda = 5, lambda = 10^5)
p | assymetry ratio (should be between 0 and 1)
max.niter | maximum number of iterations

Details

The function implements baseline correction algorithm based on Whittaker smoother. The method was first shown in [1]. The function has two main parameters - power of a penalty parameter (usually varies between 2 and 9) and the ratio of assymetry (usually between 0.1 and 0.001). The choice of the parameters depends on how broad the disturbances of the baseline are and how narrow the original spectral peaks are.

Value

preprocessed spectra.

Examples

# take spectra from carbs dataset
data(carbs)
spectra = mda.t(carbs$S)

# apply the correction
pspectra = prep.alsbasecorr(spectra, plambda = 3, p = 0.01)

# show the original and the corrected spectra individually
par(mfrow = c(3, 1))
for (i in 1:3) {
    mdpplot(list(
        original = mda.subset(spectra, i),
        corrected = mda.subset(pspectra, i)
    ), type = "l", col = c("black", "red"), lwd = c(2, 1), main = rownames(spectra)[i])
}
**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 = TRUE, scale = FALSE, 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

---

**Description**

Generic function for preprocessing

**Usage**

prep.generic(x, f, ...)

**Arguments**

- **x**: data matrix to be preprocessed
- **f**: function for preprocessing
- **...**: arguments for the function f
prep.list

Shows information about all implemented preprocessing methods.

Description

Shows information about all implemented preprocessing methods.

Usage

prep.list()

prep.msc

Multiplicative Scatter Correction transformation

Description

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

Usage

prep.msc(data, mspectrum = NULL)

Arguments

data a matrix with data values (spectra)
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
library(mdatools) data(simdata)
spectra = simdata$spectra.c cspectra = prep.msc(spectra)
par(mfrow = c(2, 1)) mdaplot(spectra, type = "l", main = "Before MSC") mdaplot(cspectra, type = "l", main = "After MSC")

Value

preprocessed spectra (calculated mean spectrum is assigned as attribut `mspectrum`)
prep.norm

**Normalization**

**Description**

Normalizes signals (rows of data matrix).

**Usage**

```r
prep.norm(data, type = "area", col.ind = NULL)
```

**Arguments**

- **data**: a matrix with data values
- **type**: type of normalization "area", "length", "sum", "snv", or "is".
- **col.ind**: indices of columns (can be either integer or logical values) for normalization to internal standard peak.

**Details**

The "area", "length", "sum" types do preprocessing to unit area (sum of absolute values), length or sum of all values in every row of data matrix. Type "snv" does the Standard Normal Variate normalization, similar to `prep.snv`. Type "is" does the normalization to internal standard peak, whose position is defined by parameter 'col.ind'. If the position is a single value, the rows are normalized to the height of this peak. If 'col.ind' points on several adjacent values, the rows are normalized to the area under the peak - sum of the intensities.

**Value**

data matrix with normalized values

---

prep.ref2km

**Kubelka-Munk transformation**

**Description**

Applies Kubelka-Munk (km) transformation to data matrix (spectra)

**Usage**

```r
prep.ref2km(data)
```

**Arguments**

- **data**: a matrix with spectra values (absolute reflectance values)
Kubelka-Munk is useful preprocessing method for diffuse reflection spectra (e.g. taken for powders or rough surface). It transforms the reflectance spectra $R$ to K/M units as follows: $(1 - R)^2 / 2R$

Value

preprocessed spectra.

Description

Applies Savitzky-Golay filter to the rows of data matrix

Usage

```r
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)

Details

The function implements algorithm described in [1] which handles the edge points correctly and does not require to cut the spectra.

References

**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.

```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

**Transformation**

**Description**

Transforms values from using any mathematical function (e.g. log).

**Usage**

```
prep.transform(data, fun, ...)
```
Arguments

- `data`: a matrix with data values
- `fun`: reference to a transformation function, e.g. `log` or `function(x) x^2`.
- `...`: optional parameters for the transformation function

Value

data matrix with transformed values

Examples

```r
# generate a matrix with two columns
y <- cbind(rnorm(100, 10, 1), rnorm(100, 20, 2))

# apply log transformation
py1 = prep.transform(y, log)

# apply power transformation
py2 = prep.transform(y, function(x) x^-1.25)

# show distributions
par(mfrow = c(2, 3))
for (i in 1:2) {
  hist(y[, i], main = paste0("Original values, column ", i))
  hist(py1[, i], main = paste0("Log-transformed, column ", i))
  hist(py2[, i], main = paste0("Power-transformed, column ", i))
}
```

---

**prep.varsel**

**Variable selection**

Description

Returns dataset with selected variables

Usage

```r
prep.varsel(data, var.ind)
```

Arguments

- `data`: a matrix with data values
- `var.ind`: indices of variables (columns) to select, can be either numeric or logical

Value

data matrix with the selected variables (columns)
preparePlotData  
*Take dataset and prepare them for plot*

**Description**

The function checks that ‘data’ contains correct numeric values, check for mandatory attributes (row and column names, x- and y-axis values and names, etc.) and add them if necessary.

Another things is to remove hidden columns and split the rest to visible and hidden values (if excluded rows are present).

**Usage**

```r
preparePlotData(data)
```

**Arguments**

- `data`  
  dataset (vector, matrix or data frame)

---

prepCalData  
*Prepares calibration data*

**Description**

Prepares calibration data

**Usage**

```r
prepCalData(x, exclrows = NULL, exclcols = NULL, min.nrows = 1, min.ncols = 2)
```

**Arguments**

- `x`  
  matrix or data frame with values (calibration set)
- `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)
- `min.nrows`  
  smallest number of rows which must be in the dataset
- `min.ncols`  
  smallest number of columns which must be in the dataset
print.classres  

Print information about classification result object

Description

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

Usage

## S3 method for class 'classres'
print(x, str = "Classification results (class classres)\nMajor fields:", ...)

Arguments

x  classification results (object of class plsdares, 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

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

Arguments

x  a iPLS (object of class ipls)
...  other arguments
**print.ldecomp**

**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.mcrals**

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- **x**: `mcrpure` object
- **...**: other arguments
print.mcrpure  

**Print method for mcrpure object**

**Description**

Prints information about the object structure

**Usage**

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

**Arguments**

- `x`  
  mcrpure 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

Description

Prints information about the object structure

Usage

## 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

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

Arguments

x a PLS model (object of class pls)
...
other arguments
print.plsda  
*Print method for PLS-DA model object*

**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.plsres  

print method for PLS results object

Description

Prints information about the object structure

Usage

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

Arguments

x  

PLS results (object of class plsres)

...  

other arguments

print.randtest

print.randtest  

Print method for randtest object

Description

Prints information about the object structure

Usage

## 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, ...)  
```

#### Arguments

- `x`: regression coefficients object (class `regcoeffs`)
- `...`: other arguments

---

### print.regmodel

**Print method for PLS model object**

#### Description

Prints information about the object structure

#### Usage

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

#### Arguments

- `x`: a regression model (object of class `regmodel`)
- `...`: other arguments
**print.regres**  
*print method for regression results object*

**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**  
*Print method for SIMCA model object*

**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**  
*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**  
*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 ncomp 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:

- **nperm**: number of permutations used for the test.
- **stat**: statistic values calculated for each component.
- **alpha**: alpha values calculated for each component.
- **statperm**: matrix with statistic values for each permutation.
- **corrperm**: matrix with correlation between predicted and reference y-values for each permutation.
- **ncomp.selected**: suggested number of components.
### Examples of using the test

```r
## Get the spectral data from Simdata set and apply SNV transformation

data(simdata)

y = simdata$conc[, 3]
x = simdata$spectra.c
x = prep.snv(x)

## Run the test and show summary
## (normally use higher nperm values > 1000)
r = randtest(x, y, ncomp = 4, nperm = 200, silent = FALSE)
summary(r)

## Show plots

par( mfrow = c(3, 2))
plot(r)
plotHist(r, ncomp = 3)
plotHist(r, ncomp = 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

regcoeffs(coeffs, ci.coeffs = NULL, use.mean = TRUE)

Arguments

coeffs
array (npred x ncomp x nresp) with regression coefficients
ci.coeffs
array (npred x ncomp x nresp x cv) with regression coefficients for computing confidence intervals (e.g. from cross-validation) using Jack-Knifing method
use.mean
logical, tells how to compute standard error for regression coefficients. If TRUE mean values for ci.coeffs is computed first. If FALSE, values (coefficients computed for global model) are used as mean.

Value

a list (object of regcoeffs class) with fields, including:

values
an array (nvar x ncomp x ny) with regression coefficients
se
an array (nvar x ncomp x ny) with standard errors for the coefficients
t.values
an array (nvar x ncomp x ny) with t-values for the coefficients
p.values
an array (nvar x ncomp x ny) with p-values for coefficients

last three fields are available if parameter ci.coeffs was provided.
Check also confint.regcoeffs, summary.regcoeffs and plot.regcoeffs.

regcoeffs.getStats

Distribution statistics for regression coefficients

Description

calculates standard error, t-values and p-values for regression coefficients based on Jack-Knifing method.

Usage

regcoeffs.getStats(coeffs, ci.coeffs = NULL, use.mean = TRUE)

Arguments

coeffs
array (npred x ncomp x nresp) with regression coefficients
ci.coeffs
array (npred x ncomp x nresp x cv) with regression coefficients for computing confidence intervals (e.g. from cross-validation) using Jack-Knifing method
use.mean
logical, tells how to compute standard error for regression coefficients. If TRUE mean values for ci.coeffs is computed first. If FALSE, values (coefficients computed for global model) are used as mean.
Value

a list with statistics three arrays: standard error, t-values and p-values computed for each regression coefficient.

---

**regres**

*Regression results*

---

**Description**

Class for storing and visualisation of regression predictions

**Usage**

```r
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:

```r
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(err)
```
Arguments

err vector with difference between reference and predicted y-values

regres.err Error of prediction

Description

Calculates array of differences between predicted and reference values.

Usage

regres.err(y.pred, y.ref)

Arguments

y.pred matrix with predicted values
y.ref vector with reference values

regres.r2 Determination coefficient

Description

Calculates matrix with coefficient of determination for every response and components

Usage

regres.r2(err, ytot)

Arguments

err vector with difference between reference and predicted y-values
yttot total variance for y-values
**regres.rmse**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculates matrix with root mean squared error of prediction for every response and components.</td>
</tr>
</tbody>
</table>

**Usage**

`regres.rmse(err)`

**Arguments**

- `err` vector with difference between reference and predicted y-values

---

**regres.slope**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculates matrix with slope of predicted and measured values for every response and components.</td>
</tr>
</tbody>
</table>

**Usage**

`regres.slope(y.pred, y.ref)`

**Arguments**

- `y.pred` matrix with predicted values
- `y.ref` vector with reference values

---

**regress.addattrs**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add names and attributes to matrix with statistics</td>
</tr>
</tbody>
</table>

**Usage**

`regress.addattrs(stat, attrs, name)`

**Arguments**

- `stat` matrix with statistics
- `attrs` attributes from error matrix
- `name` name of statistic
repmat  
*Replicate* matrix $x$

**Description**
Replicate matrix $x$

**Usage**
`repmat(x, nrows, ncols = nrows)`

**Arguments**
- $x$: original matrix
- $nrows$: number of times replicate matrix row wise
- $ncols$: number of times replicate matrix columns wise

---

rotationMatrixToX1  
*Creates a rotation matrix to map a vector* $x$ to $[1 \ 0 \ 0 \ ... \ 0]$

**Description**
Creates a rotation matrix to map a vector $x$ to $[1 \ 0 \ 0 \ ... \ 0]$

**Usage**
`rotationMatrixToX1(x)`

**Arguments**
- $x$: Vector (sequence with $J$ coordinates)

**Value**
Rotation matrix ($J \times J$)
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(obj, ncomp = NULL, ...)

Arguments

obj 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 a PCA model

Usage

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

Arguments

obj 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
selectCompNum.pls  

Select optimal number of components for PLS model

Description

Allow user to select optimal number of components for PLS model

Usage

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

Arguments

- `obj`: 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

The method sets `ncomp.selected` parameter for the model and return it back. The parameter points out to the optimal number of components in the model. You can either specify it manually, as argument `ncomp`, or use one of the algorithms for automatic selection.

Automatic selection by default based on cross-validation statistics. If no cross-validation results are found in the model, the method will use test set validation results. If they are not available as well, the model will use calibration results and give a warning as in this case the selected number of components will lead to overfitted model.

There are two algorithms for automatic selection you can chose between: either first local minimum of RMSE ('selcrit="min"') or Wold's rule ('selcrit="wold"').

The first local minimum criterion finds at which component, A, error of prediction starts raising and selects (A - 1) as the optimal number. The Wold's criterion finds which component A does not make error smaller at least by 5 as the optimal number.

If model is PLS2 model (has several response variables) the method computes optimal number of components for each response and returns the smallest value. For example, if for the first response 2 components give the smallest error and for the second response this number is 3, A = 2 will be selected as a final result.

It is not recommended to use automatic selection for real applications, always investigate your model (via RMSE, Y-variance plot, regression coefficients) to make correct decision.

See examples in help for `pls` function.

Value

the same model with selected number of components
**selratio**

*Selectivity ratio calculation*

**Description**

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

**Usage**

```r
selratio(obj, ncomp = obj$ncomp.selected)
```

**Arguments**

- `obj`: a PLS model (object of class `pls`)
- `ncomp`: number of components to count

**Value**

array `nvar x ncomp x ny` with selectivity ratio values

**References**


---

**setDistanceLimits**

*Set residual distance limits*

**Description**

Calculates and set critical limits for residuals of PCA model

**Usage**

```r
setDistanceLimits(obj, ...)
```

**Arguments**

- `obj`: a model object
- `...`: other parameters
setDistanceLimits.pca  Compute and set statistical limits for Q and T2 residual distances.

Description

Computes statistical limits for orthogonal and score distances (based on calibration set) and assign
the calculated values as model properties.

Usage

```r
## S3 method for class 'pca'
setDistanceLimits(
  obj,
  lim.type = obj$lim.type,
  alpha = obj$alpha,
  gamma = obj$gamma,
  ...
)
```

Arguments

- `obj` object with PCA model
- `lim.type` type of limits ("jm", "chisq", "ddmoments", "ddrobust")
- `alpha` significance level for detection of extreme objects
- `gamma` significance level for detection of outliers (for data driven approach)
- `...` other arguments

Details

The limits can be accessed as fields of model objects: `$Qlim` and `$T2lim`. Each is a matrix with
four rows and `ncomp` columns. First row contains critical limits for extremes, second row - for
outliers, third row contains mean value for corresponding distance (or its robust estimate in case of
`lim.type = "ddrobust"`) and last row contains the degrees of freedom.

Value

Object models with the two fields updated.
**setDistanceLimits.pls**  
Compute and set statistical limits for residual distances.

**Description**
Computes statistical limits for orthogonal and score distances (x-decomposition) and orthogonal distance (y-decomposition) based on calibration set and assign the calculated values as model properties.

**Usage**
```r
## S3 method for class 'pls'
setDistanceLimits(
  obj,
  lim.type = obj$lim.type,
  alpha = obj$alpha,
  gamma = obj$gamma,
  ...
)
```

**Arguments**
- **obj** object with PLS model
- **lim.type** type of limits ("jm", "chisq", "ddmoments", "ddrobust")
- **alpha** significance level for detection of extreme objects
- **gamma** significance level for detection of outliers (for data driven approach)
- **...** other arguments

**Details**
The limits can be accessed as fields of model objects: $Qlim$, $T2lim$, and $Zlim$. Each is a matrix with four rows and ncomp columns. In case of limits for x-decomposition, first row contains critical limits for extremes, second row - for outliers, third row contains mean value for corresponding distances (or its robust estimate in case of lim.type = "ddrobust") and last row contains the degrees of freedom.

**Value**
Object models with the three fields updated.
### showDistanceLimits

*Show residual distance limits*

**Description**

Calculates and set critical limits for residuals of PCA model

**Usage**

```r
showDistanceLimits(obj, ...)
```

**Arguments**

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

### showLabels

*Show labels on plot*

**Description**

Show labels on plot

**Usage**

```r
showLabels(
  ps, show.excluded = FALSE, pos = 3, cex = 0.65, col = "darkgray", force.x.values = NULL, bwd = 0.8
)
```

**Arguments**

- `ps` 'plotseries' object
- `show.excluded` logical, are excluded rows also shown on the plot
- `pos` position of the labels relative to the data points
- `cex` size of the labels text
- `col` color of the labels text
- `force.x.values` vector with forced x-values (or NULL)
- `bwd` bar width in case of bar plot
showPredictions

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

showPredictions.classres

Show predicted class values

Description
Shows a table with predicted class values for classification result.

Usage
## S3 method for class 'classres'
showPredictions(obj, ncomp = obj$ncomp.selected, ...)

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.
simca

Description

simca is used to make SIMCA (Soft Independent Modelling of Class Analogies) model for one-class classification.

Usage

simca(
  x,
  classname,
  ncomp = min(nrow(x) - 1, ncol(x) - 1, 20),
  x.test = NULL,
  c.test = NULL,
  cv = NULL,
  ...
)

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.
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).
cv cross-validation settings (see details).
... any other parameters suitable for pca method.

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.

Cross-validation settings, cv, 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**

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

- **classname**
  a short text with class name.

- **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.

**Author(s)**

Sergey Kucheryavskiy (svkucheryavski@gmail.com)

**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.

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 \texttt{pca} class:

\begin{itemize}
\item \texttt{selectCompNum.pca}  set number of optimal components in the model
\item \texttt{plotScores.pca}  shows scores plot.
\item \texttt{plotLoadings.pca}  shows loadings plot.
\item \texttt{plotVariance.pca}  shows explained variance plot.
\item \texttt{plotCumVariance.pca}  shows cumulative explained variance plot.
\item \texttt{plotResiduals.pca}  shows \textit{Q} vs. \textit{T2} residuals plot.
\end{itemize}

Examples

\begin{verbatim}
## make a SIMCA model for Iris setosa class with full cross-validation
library(mdatools)
data = iris[, 1:4]
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 = "cal", 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)
plotLoadings(model, comp = c(1, 2), show.labels = TRUE)
plotResiduals(model, ncomp = 2)
par(mfrow = c(1, 1))
\end{verbatim}
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`: optional 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 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.
plotCooman.simcam shows Cooman’s plot 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]
x.se = caldata[1:25, ]
x.ve = caldata[26:50, ]
x.vi = caldata[51:75, ]

x.test = iris[seq(2, nrow(iris), 2), 1:4]
c.test = iris[seq(2, nrow(iris), 2), 5]

# create individual models
m.se = simca(x.se, classname = "setosa")
m.se = selectCompNum(m.se, 1)

m.vi = simca(x.vi, classname = "virginica")
m.vi = selectCompNum(m.vi, 2)

m.ve = simca(x.ve, classname = "versicolor")
m.ve = selectCompNum(m.ve, 1)

# combine models into SIMCAM objects, show statistics and plots
m = simcam(list(m.se, m.vi, m.ve), info = "simcam model for Iris data")
summary(m)

# show predictions and residuals for calibration data
par(mfrow = c(2, 2))
plotPredictions(m)
plotCooman(m, nc = c(1, 2))
plotModelDistance(m, nc = 1)
plotDiscriminationPower(m, nc = c(1, 2))
```
par(mfrow = c(1, 1))

# apply the SIMCAM model to test set and show statistics and plots
res = predict(m, x.test, c.test)
summary(res)
plotPredictions(res)

---

**simcam.getPerformanceStats**

*Performance statistics for SIMCAM model*

**Description**

Calculates discrimination power and distance between individual SIMCA models.

**Usage**

`simcam.getPerformanceStats(models, classnames)`

**Arguments**

- `models` list with SIMCA models (as provided to simcam class)
- `classnames` names of the classes for each model

---

**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` list with prediction results from each model (pcares objects)
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.
- `plotCooman.simcamres` makes Cooman’s plot.

Methods, inherited from classres class:

- `showPredictions.classres` show table with predicted values.
- `plotPredictions.classres` makes plot with predicted values.

Check also simcam.
### Description

@description simcares is used to store results for SIMCA one-class classification.

### Usage

```r
simcares(class.res, pca.res = NULL)
```

### Arguments

- `class.res`: results of classification (class `classres`).
- `pca.res`: results of PCA decomposition of data (class `pcares`).

### 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 predicted classes plot.
- plotSensitivity.classres sensitivity plot.
- plotSpecificity.classres specificity plot.
- plotPerformance.classres performance plot.

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
```
showPredictions(model$calres)

simdata

Spectral data of polyaromatic hydrocarbons mixing

Description

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.

splitExcludedData

Split the excluded part of data

Description

Split the excluded part of data

Usage

splitExcludedData(data, type)
Arguments

data matrix with hidden data values
type type of plot

splitPlotData Split dataset to x and y values depending on plot type

Description

Split dataset to x and y values depending on plot type

Usage

splitPlotData(data, type)

Arguments

data matrix with data values (visible or hidden)
type type of plot

summary.classres Summary statistics about classification result object

Description

Generic summary function for classification results. Prints performance values for the results.

Usage

## S3 method for class 'classres'
summary(
  object,
  ncomp = object$ncomp.selected,
  nc = seq_len(object$nclasses),
  ...
)

Arguments

object classification results (object of class plsdares, simcamres, etc.).
ncomp which number of components to make the plot for (use NULL to show results for all available).
nc vector with class numbers to show the summary for.
... other arguments
### summary.ipls

**Summary for iPLS results**

**Description**

Shows statistics and algorithm parameters for iPLS results.

**Usage**

```r
## S3 method for class 'ipls'
summary(object, glob.ncomp = object$gm$ncomp.selected, ...)
```

**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.mcrals**

*Summary method for mcrals object*

**Description**

Shows some statistics (explained variance, etc) for the case.

**Usage**

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

**Arguments**

- `object` (mcrals object)
- `...` (other arguments)

**summary.mcrpure**

*Summary method for mcrpure object*

**Description**

Shows some statistics (explained variance, etc) for the case.

**Usage**

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

**Arguments**

- `object` (mcrpure 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 = object$ncomp.selected,
  ny = seq_len(nrow(object$yloadings)),
  ...
)
```

**Arguments**

- `object`  
a PLS model (object of class `pls`)
- `ncomp`  
how many components to count.
- `ny`  
which y variables to show the summary for (can be a vector)
- `...`  
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 = object$ncomp.selected,
  nc = seq_len(object$nclasses),
  ...
)
```
**summary.plsdares**

**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

**Description**

Shows performance statistics for the results.

**Usage**

```r
## S3 method for class 'plsdares'
summary(object, nc = seq_len(object$nclasses), ...)
```

**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 = seq_len(object$nresp), 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.regcoeffs

*Summary method for regcoeffs object*

**Description**

Shows estimated coefficients and statistics (if available).

**Usage**

```r
## S3 method for class 'regcoeffs'
summary(object, ncomp = 1, ny = 1, alpha = 0.05, ...)
```

**Arguments**

- `object` : object of class `regcoeffs`
- `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.regmodel**

*Summary method for regression model object*

**Description**

Shows performance statistics for the model.

**Usage**

```r
## S3 method for class 'regmodel'
summary(
  object,
  ncomp = object$ncomp.selected,
  ny = seq_len(object$res$cal$nresp),
  res = object$res,
  ...)
```

**Arguments**

- `object`: a regression model (object of class `regmodel`)
- `ncomp`: number of components to show summary for
- `ny`: which y variables to show the summary for (can be a vector)
- `res`: list of results to show summary for
- `...`: other arguments

**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 = object$ncomp.selected, ny = seq_len(object$nresp), ...)```

**Arguments**

- `object`: regression results (object of class `regres`)
- `ncomp`: model complexity to show the summary for (if NULL - shows for all available values)
- `ny`: for which response variable show the summary for (one value or a vector)
- `...`: other arguments
summary.simca  Summary method for SIMCA model object

Description

Shows performance statistics for the model.

Usage

## S3 method for class 'simca'
summary(object, ncomp = object$ncomp.selected, res = object$res, ...)

Arguments

- `object`: a SIMCA model (object of class simca)
- `ncomp`: number of components to show summary for
- `res`: list of result objects to show summary for
- `...`: other arguments

summary.simcam  Summary method for SIMCAM model object

Description

Shows performance statistics for the model.

Usage

## S3 method for class 'simcam'
summary(object, nc = seq_len(object$nclasses), ...)

Arguments

- `object`: a SIMCAM model (object of class simcam)
- `nc`: number of class to show summary for (can be vector)
- `...`: other arguments
**summary.simcamres**  
*Summary method for SIMCAM results object*

**Description**

Shows performance statistics for the results.

**Usage**

```r
## S3 method for class 'simcamres'
summary(object, nc = seq_len(object$nclasses), ...)
```

**Arguments**

- **object**: SIMCAM results (object of class `simcamres`)
- **nc**: number of class to show summary for (can be vector)
- **...**: other arguments

---

**summary.simcares**  
*Summary method for SIMCA results object*

**Description**

Shows performance statistics for the results.

**Usage**

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

**Arguments**

- **object**: SIMCA results (object of class `simcares`)
- **...**: other arguments
unmix.mcrpure  

Description
Unmix spectral data using pure variables estimated before

Usage
unmix.mcrpure(obj, x)

Arguments

obj mcrpure object
x matrix with spectra

Value
Returns a list with resolved spectra and contributions (matrices).

vipscores  

VIP scores for PLS model

Description
Calculates VIP (Variable Importance in Projection) scores for predictors for given number of components and response variable.

Usage
vipscores(obj, ncomp = obj$ncomp.selected)

Arguments

obj a PLS model (object of class pls)
ncomp number of components to count

Details
May take some time in case of large number of predictors Returns results as a column-vector, with all necessary attributes inherited (e.g. xaxis.values, excluded variables, etc.). If you want to make a plot use for example: mdaplot(mda.t(v), type = "l"), where v is a vector with computed VIP scores. Or just try plotVIPScores.pls.
Vipscores

Value

matrix nvar x ny with VIP score values (columns correspond to responses).

References

Index

* datasets
  carbs, 14
  pellets, 108
  people, 109
  simdata, 261

as.matrix.classes, 10
as.matrix.ldecomps, 10
as.matrix.plsdaRes, 11
as.matrix.plsres, 11
as.matrix.regcoeffs, 12
as.matrix.res, 11
as.matrix.simcamRes, 13
as.matrix.simcaRes, 13

capitalize, 14

carbs, 14
categorize, 15
categorize.pca, 15, 200
categorize.pls, 16, 200
chisq.crit, 17
chisq.prob, 17
classify.plsda, 18
classify.simca, 18
classmodel.processRefValues, 19
classes, 19, 210, 258–260
classes.getPerformance, 20
confint.regcoeffs, 21, 240
constraint, 22
constraintAngle, 22
constraintClosure, 23
constraintNonNegativity, 23
constraintNorm, 24
constraints.list, 24
constraintUnimod, 24
crossval, 25
crossval.getParams, 25
crossval.regmodel, 26
crossval.simca, 26
crossval.str, 27
dd.crit, 27
ddmoments.param, 28
ddrobust.param, 28
ellipse, 29
employ.constraint, 29
employ.prep, 30
eye, 30
fprintf, 31
getCalibrationData, 31
classRes.getPerformance, 20
getConfusionMatrix, 33
getImportance, 33
getConvexHull, 34
dataLabels, 35
classRes.getPerformance, 20
getImplementedConstraints, 35
getImplementedPrepMethods, 35
getLabelsAsIndices, 36
getLabelsAsValues, 36
getMainTitle, 36
getPlotColors, 37
getProbabilities, 37
getProbabilities.pca, 38
getProbabilities.simca, 38
getPureVariables, 39
getR, 39
getRegcoeffs, 40
getRegcoeffs.regmodel, 40, 201
getRes, 41
getSelectedComponents, 42
gSelectivityRatio, 42
gSelectivityRatio.pls, 43
gVariance.mcr, 43
gVIPscores, 44
gVIPscores.pls, 44
hotelling.crit, 45
INDEX

hotelling.prob, 45
imshow, 46
ipls, 46, 96, 165, 201
ipls.backward, 49
ipls.forward, 49
jm.crit, 50
jm.prob, 50
ldecomp, 51, 105, 106, 260
ldecomp.getDistances, 52
ldecomp.getLimitsCoordinates, 52
ldecomp.getLimParams, 53
ldecomp.getQLimits, 53
ldecomp.getT2Limits, 54
ldecomp.getVariances, 54
ldecomp.plotResiduals, 55
mcr, 56
mcrals, 22, 57, 97
mcrals.cal, 61
mcralsfcnmls, 63
mcrals.nnls, 63
mcrals.ols, 64
mcrpure, 58, 59, 65, 66, 97
mda.cbind, 68
mda.data2im, 68
mda.df2mat, 69, 79
mda.exclcols, 69
mda.exclrows, 70
mda.getattr, 70
mda.getexclind, 71
mda.im2data, 71
mda.inclcols, 72
mda.inclrows, 72
mda.purge, 73
mda.purgeCols, 73
mda.purgeRows, 73
mda.rbind, 74
mda.setattr, 74
mda.setimbg, 75
mda.show, 75
mda.subset, 76, 79
mda.t, 76
mdaplot, 77, 90, 97, 127, 143, 146, 149, 150, 156, 163, 173, 191, 213
mdaplot.areColors, 80
mdaplot.formatValues, 80
mdaplot.getColors, 81
mdaplot.getXAxisLim, 82
mdaplot.getXTickLabels, 82
mdaplot.getYTickLabels, 84
mdaplot.plotAxes, 85
mdaplot.prepareColors, 86
mdaplot.showColorbar, 86
mdaplot.showLines, 87
mdaplotg, 80, 87, 96, 97, 142, 145, 146, 148, 149, 164
mdaplotg.getLegend, 90
mdaplotg.getXLim, 91
mdaplotg.getYLim, 91
mdaplotg.prepareData, 92
mdaplotg.processParam, 93
mdaplotg.showLegend, 93
mdaplotyy, 94
mdatools, 96
pca, 51, 96, 97, 105, 106, 112, 128, 137, 157, 164, 175, 199, 252–254
pca.cal, 101
pca.getB, 102
pca.mvreplace, 100, 102
pca.nipals, 104
pca.run, 104
pca.svd, 105
pcares, 51, 96, 99, 100, 105, 259, 260
pcv, 97, 107
pellets, 108
people, 109
pinv, 109
plot.classres, 110
plot.ipls, 110
plot.mcr, 111
plot.pca, 111
plot.pcares, 112
plot.pls, 112, 200
plot.plsda, 113
plot.plsdares, 113
plot.plsres, 114, 210, 213
plot.randtest, 115, 239
plot.regcoeffs, 115, 240
plot.regres, 116
plot.simca, 117
plot.simcam, 117
plot.simcamres, 118
plotBars, 118, 168
plotBiplot, 119
plotBiplot.pca, 100, 119
plotConfidenceEllipse, 120, 134
plotContributions, 121
plotContributions.mcr, 59, 66, 121
plotConvexHull, 122, 134
plotCooman, 123
plotCooman.simcam, 123, 125, 256
plotCooman.simcamres, 124, 258
plotConvexHull, 122, 134
plotCooman, 123
plotCooman.simcam, 123, 125, 256
plotCooman.simcamres, 124, 258
plotConvexHull, 122, 134
plotCumVariance, 126
plotCumVariance.ldecomp, 106, 127, 260
plotCumVariance.mcr, 59, 66, 127
plotCumVariance.pca, 100, 128, 254
plotDensity, 128, 168
plotDiscriminationPower, 129
plotDiscriminationPower.simcam, 129, 256
plotDistDoF, 100, 130
plotErrorbars, 131, 168
plotExtreme, 131
plotExtreme.pca, 100, 132
plotHist, 133
plotHist.randtest, 133, 239
plotHist, 133
plotHotellingEllipse, 134
plotLines, 135, 168
plotLoadings, 136
plotLoadings.pca, 100, 136, 254
plotMisclassified, 137
plotMisclassified.classmodel, 138, 208, 254, 256
plotMisclassified.classres, 20, 138
plotModelDistance, 139
plotModelDistance.simcam, 139, 256
plotModellingPower, 141
plotPerformance, 141
plotPerformance.classmodel, 138, 142, 167, 170
plotPerformance.classres, 20, 138, 143, 168, 170, 210, 260
plotPointsShape, 144
plotPredictions, 144
plotPredictions.classmodel, 145, 208, 253, 256
plotPredictions.classres, 20, 110, 145, 210, 258, 260
plotPredictions.regmodel, 146, 201
plotPredictions.regres, 116, 147, 210, 213
plotPredictions.simcam, 148
plotPredictions.simcamres, 149
plotProbabilities, 149
plotProbabilities.classres, 150
plotPurity, 150
plotPurity.mcrpure, 66, 151
plotPuritySpectra, 151
plotPuritySpectra.mcrpure, 66, 152
plotQDoF, 100, 153
plotRegressionLine, 154
plotResiduals, 155
plotResiduals.ldecomp, 56, 106, 155, 157, 260
plotResiduals.pca, 100, 156, 183, 254
plotResiduals.regres, 157, 195
plotRMSE, 158
plotRMSE IPLS, 159, 165
plotRMSE.regmodel, 160, 201
plotRMSE.regres, 160, 210, 213
plotScatter, 161, 168
plotScores, 162
plotScores.ldecomp, 106, 134, 163, 164, 260
plotScores.pca, 100, 134, 163, 254
plotScores.pca, 100, 134, 163, 254
plotSelectivityRatio, 166
plotSelectivityRatio.pls, 166, 200
plotSensitivity, 167
plotSensitivity.classmodel, 167, 208, 254, 256
plotSensitivity.classres, 20, 168, 210, 260
plotSensitivity.classres, 20, 168, 210, 260
plotSensitivity.classres, 20, 168, 210, 260
plotSpectra, 171
plotSpectra.mcr, 59, 66, 171
plotT2DoF, 100, 172
plotVariance, 172
plotVariance.ldecomp, 106, 173, 260
plotVariance.mcr, 39, 66, 174
plotVariance.pca, 100, 174, 254
plotVariance.pls, 175
plotVariance.plsres, 176
plotVIPScores, 177
plotVIPScores.pls, 177, 200, 272
plotWeights, 178
plotWeights.pls, 178, 200
plotXCumVariance, 179
plotXCumVariance.pls, 179, 200
plotXCumVariance.plsres, 180, 210, 213
plotXLoadings, 180
plotXLoadings.pls, 181, 200
plotXResiduals, 181
plotXResiduals.pls, 182, 200
plotXResiduals.plsres, 183, 210, 213
plotXScores, 184
plotXScores.pls, 134, 184, 200
plotXScores.plsres, 134, 185, 210, 213
plotXVariance, 186
plotXVariance.pls, 186, 200
plotXVariance.plsres, 187, 210, 213
plotXYLoadings, 187
plotXYLoadings.pls, 188, 200
plotXYResiduals, 188
plotXYResiduals.pls, 189, 200
plotXYResiduals.plsres, 190
plotXYScores, 191
plotXYScores.pls, 191, 200
plotXYScores.plsres, 192, 210, 213
plotYCumVariance, 193
plotYCumVariance.pls, 193, 200
plotYCumVariance.plsres, 194, 210, 213
plotYResiduals, 194
plotYResiduals.plsres, 195, 210, 213
plotYResiduals.regmodel, 195, 201
plotYVariance, 196
plotYVariance.pls, 196, 200
plotYVariance.plsres, 197, 210, 213
pls, 96, 113, 114, 176, 179, 181, 185, 186, 188, 192, 193, 197, 198, 208, 214, 217, 246
pls.cal, 203
pls.getLLimitsCoordinates, 204
pls.getLLimits, 204
pls.run, 205
pls.simpls, 200, 205
plsla, 19, 96, 113, 138, 145, 148, 167, 170, 206, 211, 217
plsla, 96, 138, 143, 146, 149, 168, 171, 207, 209
plsla, 96, 114, 176, 180, 184, 185, 187, 192, 194, 197, 201, 210, 212, 217
predict.mcrals, 59, 215
predict.mcrpure, 66, 215
predict.pca, 100, 105, 216
predict.pls, 200, 212, 216
predict.plsla, 208, 209, 211, 217
predict.simca, 218, 253, 259
predict.simcam, 218, 255, 258
prep, 219
prep.alsbasecorr, 97, 220
prep.autoscale, 97, 221
prep.generic, 221
prep.list, 222
prep.msc, 97, 222
prep.norm, 97, 223
prep.ref2km, 223
prep.savgol, 97, 224
prep.snv, 97, 223, 225
prep.transform, 225
prep.varsel, 226
preparePlotData, 227
prepCalData, 227
print.classres, 228
print.ipls, 228
print.ldecomp, 229
print.mcrals, 229
print.mcrpure, 230
print.pca, 230
print.pcres, 231
print.pls, 231
print.plsda, 232
print.plsdares, 232
print.plsres, 233
print.randtest, 233
print.regcoeffs, 234
print.regmodel, 234
print.regres, 235
print.simca, 235
print.simcam, 236
print.simcamres, 236
print.simcares, 237
randtest, 96, 115, 126, 133, 201, 237
regcoeffs, 199, 207, 239
regcoeffs.getStats, 240
regres, 241
regres.bias, 241
regres.err, 242
regres.r2, 242
regres.rmse, 243
regres.slope, 243
regress.addattrs, 243
repmat, 244
rotationMatrixToX1, 244
selectCompNum, 245
selectCompNum.pca, 100, 245, 254
selectCompNum.pls, 199, 200, 246
selratio, 43, 200, 247
setDistanceLimits, 247
setDistanceLimits.pca, 100, 248
setDistanceLimits.pls, 200, 249
showDistanceLimits, 250
showLabels, 250
showPredictions, 251
showPredictions.classres, 20, 210, 251, 258, 260
simcam.getPerformanceStats, 257
simcamres, 96, 118, 138, 143, 146, 149, 168, 171, 255, 257
simcares, 19, 96, 253, 259
simdata, 261
splitExcludedData, 261
splitPlotData, 262
summary.classres, 262
summary.classres, 20, 262
summary.classres, 262
summary.ipls, 159, 165, 263
summary.ldecomp, 263
summary.mcrals, 264
summary.mcrpure, 264
summary.pca, 265
summary.pcares, 265
summary.pls, 200, 266
summary.plsda, 266
summary.plsdares, 267
summary.plsres, 210, 213, 267
summary.randtest, 239, 268
summary.regcoeffs, 240, 268
summary.regmodel, 269
summary.regres, 269
summary.simca, 270
summary.simcam, 270
summary.simcamres, 271
summary.simcares, 271
unmix.mcrpure, 66, 272
vipscores, 44, 166, 177, 200, 272