Package ‘ChemometricsWithR’

September 1, 2017

Type Package

Title Chemometrics with R - Multivariate Data Analysis in the Natural Sciences and Life Sciences

Version 0.1.11

Author Ron Wehrens

Maintainer Ron Wehrens <ron.wehrens@gmail.com>

Description Functions and scripts used in the book "Chemometrics with R - Multivariate Data Analysis in the Natural Sciences and Life Sciences" by Ron Wehrens, Springer (2011). Data used in the package are available from github.

URL https://github.com/rwehrens/CWR

BugReports https://github.com/rwehrens/CWR/issues

Imports MASS, pls, kohonen, devtools

Suggests nnet, randomForest, ada, rrcov, sfsmisc, ipred, fastICA, rda, TIMP, class, e1071, rpart, cluster, ALS, ptw, dtw, boot, leaps, lars, elasticnet, subselect, signal, mclust

License GPL (>= 2)

LazyLoad yes

NeedsCompilation no

Repository CRAN

Date/Publication 2017-09-01 13:09:14 UTC

R topics documented:

  ChemometricsWithR-package ........................................... 2
  AdjRkl ................................................................. 2
  Error ................................................................. 3
  Evaluation .......................................................... 4
  GA ................................................................. 5
  gini ................................................................. 7
  installChemometricsWithRData ..................................... 8
AdjRkl

Description

The Adjusted Rand Index is a measure of similarity for two groupings or clusterings. A value of 1 indicates total agreement.

Usage

AdjRkl(part1, part2)
Arguments

part1  First partitioning.
part2  Second partitioning.

Value

Number.

Author(s)

Ron Wehrens

References


Examples

```r
if (require("kohonen")) {
  data(wines, package = "kohonen")
  wines.dist <- dist(scale(wines))
  wines.sl <- hclust(wines.dist, method = "single")
  wines.cl <- hclust(wines.dist, method = "complete")

  AdjRkl(cutree(wines.sl, 4), cutree(wines.cl, 4))
} else {
  cat("Package kohonen not available.\nInstall it by typing 'install.packages("kohonen")'\n")
}
```

---

**Error**

*Often-used error functions*

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Error functions for classification and regression</td>
</tr>
</tbody>
</table>

**Usage**

```
rmse(x, y)
err.rate(x, y)
```

**Arguments**

```
x, y       True or predicted values, either numbers or factors.
```
Evaluation

Value
Function rms returns the root-mean-square error for real-valued x and y vectors. Function err.rate returns the fraction of non-matching cases in x and y (real numbers or factors).

Author(s)
Ron Wehrens

References

Evaluation function examples for SA- or GA-based variable selection in classification applications.

Description
Two examples of functions that can be used in variable selection for classification. The outcome of these functions should be maximized by the optimization.

Usage
lda.loofun(x, grouping, subset, ...)
pls.cvfun(x, response, subset, ...)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Data matrix: independent variables used by eval.fun</td>
</tr>
<tr>
<td>grouping</td>
<td>Class vector, possibly a factor</td>
</tr>
<tr>
<td>response</td>
<td>Dependent variable, typically a real number</td>
</tr>
<tr>
<td>subset</td>
<td>A vector containing the indices of the variables to be included</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments, such as the number of latent variables to use in plscvfun</td>
</tr>
</tbody>
</table>

Details
The evaluation function should give high values for good subsets, and low values for bad subsets. The lda.loofun function simply counts the number of correct predictions in LOO crossvalidation, and subtracts the number of variables in the subset. Function pls.cvfun returns the mean squared error of cross-validation.

Value
One value indicating the quality of the subset
Author(s)
Ron Wehrens

References

See Also
GA, SA

---

Description
A set of functions implementing simple variable selection in classification applications using genetic algorithms.

Usage

```r
GAfun(X, C, eval.fun, kmin, kmax, popsize = 20, niter = 50,
       mut.prob = 0.05, ...)
GAfun2(X, C, eval.fun, kmin, kmax, popsize = 20, niter = 50,
       mut.prob = 0.05, ...) 

GA.init.pop(popsize, nvar, kmin, kmax)
GA.select(pop, number, qlts, min.qlt = 0.4, qlt.exp = 1)
GA.mut(subset, maxvar, mut.prob = 0.01)
GA.XO(subset1, subset2)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Data matrix: independent variables used by eval.fun</td>
</tr>
<tr>
<td>C</td>
<td>Class vector, used by eval.fun</td>
</tr>
<tr>
<td>eval.fun</td>
<td>evaluation function. Should take a data matrix, a class vector (or factor), and a subset argument</td>
</tr>
<tr>
<td>kmin</td>
<td>Minimal number of variables to retain</td>
</tr>
<tr>
<td>kmax</td>
<td>Maximal number of variables to retain</td>
</tr>
<tr>
<td>popsize</td>
<td>Size of the GA population</td>
</tr>
<tr>
<td>niter</td>
<td>Number of iterations</td>
</tr>
<tr>
<td>mut.prob</td>
<td>Mutation probability</td>
</tr>
<tr>
<td>...</td>
<td>Further arguments to the evaluation function</td>
</tr>
</tbody>
</table>
nvar         The total number of variables to choose from
pop, subset, subset1, subset2         A (part of a) population of trial solutions
number         The number of trial solutions that may produce offspring
qlts         Vector of quality measures for members in a population
min.qlt         Minimal quality of a trial solution to be considered as a future parent
qlt.exp         Quality scaling parameter: the larger this number, the more discrimination between good and bad solutions, and the more greedy the search characteristics
maxvar         Number of variables to choose from

Details

The function generates a population of trial solutions, each containing a number of variables to be retained. For every member of the population, the evaluation function calculates a quality measure, which determines the chance of that member to create offspring. In a process of "survival of the fittest", this leads to subsets for which the evaluation function has a maximal value.

The initialization is done randomly. Selection is simple threshold selection. Mutation swaps variables in or out of the subset; the cross-over type is uniform. Functions GA.init.pop, GA.select, GA.mut and GA.xo are auxiliary functions, not meant to be called directly by the user.

Value

Functions GAfun and GAfun2 both return a list containing the following fields:

best         The best subset
best.q         The quality of the best subset
n.iter         The number of iterations

In addition, the outcome of GAfun2 also contains

qualities         A matrix containing the best, median and worst quality value throughout the optimization

Author(s)

Ron Wehrens

References


See Also

Evaluation, SA
Examples

```r
if (require("pls")) {
  data(gasolineL package = "pls")
  ## Usually more iterations are needed
  GAobj <- GAfun(gasolineNIR, gasoline$octane,
                  eval.fun = pls.cvfun, niter = 20,
                  kmin = 3, kmax = 25, ncomp = 2)
  GAobj
} else {
  cat("Package pls not available.\nInstall it by typing \'install.packages("pls")\'\n")
}
```

---

**gini**  
*Gini impurity index for cart objects*

### Description

A simple implementation of the Gini impurity index for classification and regression trees. Not meant to be called directly - included for demonstration purposes.

### Usage

```r
gini(x, class, splitpoint)
```

### Arguments

- `x`: Numeric vector of length n.
- `class`: Class labels, length n.
- `splitpoint`: Tentative split point.

### Value

The Gini impurity index, given a certain split point, a vector of possible splits, and a vector of class labels. Lower values indicate more pure leaves.

### Author(s)

Ron Wehrens

### References

installChemometricsWithRData

Installation of ChemometricsWithRData

Description

Function to download and install the ChemometricsWithRData package from its github location.

Details

The total size of the data sets in the ChemometricsWithRData package (Prostate2000Raw, prostate, bdata and shootout) is too large for CRAN according to current guidelines. The data package is now available from github only.

Author(s)

Ron Wehrens
Maintainer: Ron Wehrens <ron.wehrens@gmail.com>

References


Examples

## Not run:
installChemometricsWithRData()

## End(Not run)

MCR

Functions for Multivariate Curve Resolution

Description

Multivariate Curve Resolution, or MCR, decomposes a bilinear matrix into its pure components. A classical example is a matrix consisting of a series of spectral measurements on a mixture of chemicals for following the reaction. At every time point, a spectrum is measured that is a linear combination of the pure spectra. The goal of MCR is to resolve the pure spectra and concentration profiles over time.
Usage

```r
mcr(x, init, what = c("row", "col"), convergence = 1e-08,
    maxit = 50)
opax(ncomp)
efax(ncomp)
```

Arguments

- **x**: Data matrix
- **init**: Initial guess for pure compounds
- **what**: Whether the pure compounds are rows or columns of the data matrix
- **convergence**: Convergence criterion
- **maxit**: Maximal number of iterations
- **ncomp**: Number of pure compounds

Details

MCR uses repeated application of least-squares regression to find pure profiles and spectra. The method is iterative; both EFA and OPA are methods to provide initial guesses.

Value

Function `mcr` returns a list containing

- **C**: An estimate of the pure "concentration profiles"
- **S**: An estimate of the pure "spectra"
- **resids**: The residuals of the final decomposition
- **rms**: Root-mean-square values of the individual iterations

Function `opa` returns a list containing

- **pure.compounds**: A matrix containing `ncomp` pure compounds, usually spectra at specific time points
- **selected**: The wavelengths leading to the estimates of the pure concentration profiles

Function `efa` returns a list containing

- **pure.compounds**: A matrix containing `ncomp` pure compounds, usually concentration profiles at specific wavelengths
- **forward**: The development of the singular values of the reduced data matrix when increasing the number of columns in the forward direction
- **backward**: The development of the singular values of the reduced data matrix when increasing the number of columns in the backward direction

Usually, `opa` and `efa` are employed in opposite ways: if `opa` is used to find the "purest" row of a data matrix, one would typically employ `efa` to find the "purest" column, and vice versa.
Author(s)
Ron Wehrens

References

Examples
## Not run:
if (require("ChemometricsWithRData")) {
  data(bdata, package = "ChemometricsWithRData")
  D1.efa <- efa(bdata$d1, 3)
  matplot(D1.efa$forward, type = "l")
  matplot(D1.efa$backward, type = "l")
  matplot(D1.efa$pure.comp, type = "l")
  D1.opa <- opa(bdata$d1, 3)
  matplot(D1.opa$pure.comp, type = "l")
  D1.mcr.efa <- mcr(bdata$d1, D1.efa$pure.comp, what = "col")
  matplot(D1.mcr.efa$c, type = "l", main = "Concentration profiles")
  matplot(t(D1.mcr.efa$), type = "l", main = "Pure spectra")
}
## End(Not run)

PCA

Principal Component Analysis

Description
Functions for PCA: creating a PCA object, extracting variances, scores and loadings for individual PCs, projecting new data in the PC space, and reconstruction using a limited number of PCs.

Usage
PCA(X, warn = TRUE)
## S3 method for class 'PCA'
summary(object, varperc = 90, pc.select = c(1:5,10), ...)
variances(object, npc = maxpc)
## S3 method for class 'PCA'
scores(object, npc = maxpc, ...)
## S3 method for class 'PCA'
loadings(object, npc = maxpc, ...)
reconstruct(object, npc = maxpc)
project(object, npc = maxpc, newdata, ldngs)
Arguments

Arguments

- **x**: a matrix, with each row representing an object.
- **warn**: logical, whether or not to give a warning when the data are not mean-centered.
- **object**: an object of class "PCA" (see below).
- **varperc**: variance threshold in the summary function.
- **...**: extra arguments, e.g., for printing the variance table (digits = ...).
- **pc.select**: PCs to be included in the summary function.
- **npc**: the number of PCs to be returned.
- **newdata**: data (with the same number of variables as the original data) that are to be projected into the space of the first npc PCs.
- **ldngs**: loadings to be used; by default the PCA loadings.

Value

Function **pca** returns an object of class "PCA" with components

- **scores**: object weights per PC.
- **loadings**: variable weights per PC.
- **var**: variance explained per PC.
- **totalvar**: The total variance in the data set.

Function **summary.pca** gives a short summary of the PCA model, stating how many PCs are needed to cover a certain percentage of the total variance, and for selected PCs gives the (cumulative) variance explained.

Function **variances** returns the variances associated with each PC.

Function **scores** returns the scores associated with each PC.

Function **loadings** returns the loadings associated with each PC.

Function **reconstruct** returns the reconstruction of the original data matrix, based on npc PCs.

Function **project** projects the new data into the subspace spanned by the given loadings. If argument ldngs is given, arguments pcamod and npc are not needed.

Author(s)

Ron Wehrens

References


See Also

- plot.PCA
Examples

```r
data(wines, package = "kohonen")
wines.PC <- PCA(scale(wines))
```

---

**PCA.plot**  
Principal Component Analysis plotting functions

**Description**

Plotting functions for PCA: for scores, loadings, scores and loadings simultaneously (a biplot), and variances (a screeplot, where the log of the explained variance is plotted for each PC).

**Usage**

```r
## S3 method for class 'PCA'
scoreplot(object, pc = c(1, 2), pcscores = scores(object),
  show.names = FALSE, xlab, ylab, xlim, ylim, ...)

## S3 method for class 'PCA'
loadingplot(object, pc = c(1, 2), pcloadings = loadings(object),
  scalefactor = 1, add = FALSE, show.names = FALSE,
  xlab, ylab, xlim, ylim, col = "blue", min.length = 0.01, varnames = NULL, ...)

## S3 method for class 'PCA'
biplot(x, pc = c(1,2),
  show.names = c("none", "scores", "loadings", "both"),
  score.col = 1, loading.col = "blue",
  min.length = .01, varnames = NULL, ...)

screeplot(object, type = c("scree", "percentage"), npc, ...)
```

**Arguments**

- **x, object** an object of class "PCA" (see below).
- **pc** which PCs to show.
- **pcscores** matrix of scores, by default the scores of the PCA model object.
- **show.names** show names rather than plotting symbols. For loadingplot and scoreplot a logical (default: FALSE), for biplot one of 'scores', 'loadings', 'both' or 'none' (default).
- **xlab, ylab, xlim, ylim, col** graphical parameters of the plot.
- **pcloadings** matrix of loadings, by default the loadings of the PCA model object.
- **scalefactor** scaling factor for the loadings; used internally, when the loadingplot function is called from within biplot.PCA.
- **add** logical, whether to add to the existing plot (again, useful when loadingplot is called from within biplot.PCA).
pick.peaks

Function to identify local maxima in a vector, typically a spectrum or a chromatogram.

Usage

pick.peaks(x, span)

Description

Function to identify local maxima in a vector, typically a spectrum or a chromatogram.

Details

Score plots and loading plots show the amount of explained variance at the axis labels only when PCA has been performed at mean-centered data.

Author(s)

Ron Wehrens

References


See Also

PCA

Examples

data(wines, package = "kohonen")
wines.PC <- PCA(scale(wines))
wine.classes <- as.integer(vintages)
scoreplot(wines.PC, col = wine.classes, pch = wine.classes)
loadingplot(wines.PC, show.names = TRUE)
biplot(wines.PC, score.col = wine.classes)
screeplot(wines.PC)
Arguments

- **x**: Numerical vector.
- **span**: Neighbourhood, used to define local maxima.

Value

A vector containing positions of local maxima in the input data.

Author(s)

Ron Wehrens

Examples

```r
if (require("ptw")) {
  data(lcms, package = "ptw")
  plot(lcms[,1], type = "l", xlim = c(1000, 1500))
  abline(v = pick.peaks(lcms[,1], 20), col = "blue")
} else {
  cat("Package ptw not available. Install it by typing 'install.packages("ptw")'"
}
```

Description

Simulated Annealing for variable selection in classification

A set of functions implementing simple variable selection in classification applications using simulated annealing

Usage

```
SAfun(x, response, eval.fun, Tinit, niter = 100,
      cooling = 0.05, fraction = 0.3, ...)
SAfun2(x, response, eval.fun, Tinit, niter = 100,
       cooling = 0.05, fraction = 0.3, ...)
SAstep(curr.set, maxvar, fraction = .3, size.dev = 1)
```

Arguments

- **x**: Data matrix: independent variables used by eval.fun
- **response**: Class vector, used by eval.fun
- **eval.fun**: evaluation function. Should take a data matrix, a class vector (or factor), and a subset argument
- **Tinit**: Initial temperature
Simulated Annealing (SA) starts with a random subset, and proceeds by random moves in the solution space. In this implementation, a new solution may deviate in length at most `size.dev` variables: at most two variables may be swapped in or out at each step. If a step is an improvement, it is unconditionally accepted. If not, acceptance is a stochastic process depending on the current temperature - with high temperatures, "bad" moves are more likely to be accepted than with low temperatures. The process stops after a predefined number of iterations.

Functions `SAfun` and `SAfun2` both return a list containing the following fields:

best The best subset
best.q The quality of the best subset

In addition, the outcome of `SAfun2` also contains

qualities A vector containing quality values of solutions seen throughout the optimization
accepts A vector containing logicals indicating which solutions were accepted and which were rejected

Author(s)
Ron Wehrens

References

See Also
Evaluation, GA
Examples

```r
if (require("pls")) {
  data(gasoline, package = "pls")
  ## usually more than 50 iterations are needed
  SAobj <- SAsfun(gasoline$NIR, gasoline$octane,
                   eval.fun = pls.cvfun, Tinit = 3,
                   fraction = 0.02, niter = 50, ncomp = 2)
  SAobj
} else {
  cat("Package pls not available.\nInstall it by typing 'install.packages("pls")'")
}
```

unsigned.range

Unsigned range of the data vector including zero.

Description

Function returning the range of the data where, if necessary, the range is extended to include zero. Not meant to be called directly by the user.

Usage

```r
unsigned.range(x)
```

Arguments

- `x` Numeric vector.

Value

A vector of two numbers.

Note

From the R stats package (see biplot.default).
Index

*Topic **cluster**
  AdjRk1, 2

*Topic **data**
  installChemometricsWithRData, 8

*Topic **hplot**
  PCA.plot, 12

*Topic **manip**
  Error, 3
gini, 7
MCR, 8
pick.peaks, 13

*Topic **misc**
  unsigned.range, 16

*Topic **multivariate**
  PCA, 10

*Topic **optimize**
  Evaluation, 4
  GA, 5
  SA, 14

*Topic **package**
  ChemometricsWithR-package, 2

AdjRk1, 2

biplot.PCA (PCA.plot), 12

ChemometricsWithR
  (ChemometricsWithR-package), 2

ChemometricsWithR-package, 2

efa (MCR), 8
evaluation (Error), 3
Error, 3
Evaluation, 4, 6, 15

GA, 5, 5, 15
GAfun (GA), 5
GAfun2 (GA), 5
gini, 7

installChemometricsWithRData, 8

lda.loofun (Evaluation), 4
loadingplot (PCA.plot), 12
loadings (PCA), 10

MCR, 8
mcr (MCR), 8
opa (MCR), 8

PCA, 10, 13
PCA.plot, 12
pick.peaks, 13
plot.PCA, 11
plot.PCA (PCA.plot), 12
pls.cvfun (Evaluation), 4
project (PCA), 10
reconstruct (PCA), 10
rms (Error), 3

SA, 5, 6, 14
SAfun (SA), 14
SAfun2 (SA), 14
SAstep (SA), 14
scoreplot (PCA.plot), 12
scores (PCA), 10
screeplot (PCA.plot), 12
summary.PCA (PCA), 10
unsigned.range, 16

variances (PCA), 10