Package ‘plsRglm’

March 15, 2021

Version 1.3.0

Date 2021-03-14

Depends R (>= 2.10)

Imports mvtnorm, boot, bipartite, car, MASS

Enhances pls

Suggests plsdof, R.rsp, chemometrics, plsdepot

Title Partial Least Squares Regression for Generalized Linear Models

Author Frederic Bertrand [cre, aut] (<https://orcid.org/0000-0002-0837-8281>),
Myriam Maumy-Bertrand [aut] (<https://orcid.org/0000-0002-4615-1512>)

Maintainer Frederic Bertrand `<frederic.bertrand@math.unistra.fr>`

Description Provides (weighted) Partial least squares Regression for generalized linear models and repeated k-fold cross-validation of such models using various criteria <arXiv:1810.01005>. It allows for missing data in the explanatory variables. Bootstrap confidence intervals constructions are also available.

License GPL-3

Encoding UTF-8

URL `https://fbertran.github.io/plsRglm/`,
`https://github.com/fbertran/plsRglm/`

BugReports `https://github.com/fbertran/plsRglm/issues/`

VignetteBuilder R.rsp

Classification/MSC 62J12, 62J99

RoxygenNote 7.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2021-03-15 22:40:06 UTC

R topics documented:

`plsRglm-package` *plsRglm-package*

Description

Provides (weighted) Partial least squares Regression for generalized linear models and repeated k-fold cross-validation of such models using various criteria <arXiv:1810.01005>. It allows for missing data in the explanatory variables. Bootstrap confidence intervals constructions are also available.

References

A short paper that sums up some of features of the package is available on `https://arxiv.org/`, Frédéric Bertrand and Myriam Maumy-Bertrand (2018), "plsRglm: Partial least squares linear and generalized linear regression for processing incomplete datasets by cross-validation and bootstrap techniques with R", *arxiv*, `https://arxiv.org/abs/1810.01005`, `https://github.com/fbertran/plsRglm/` et `https://fbertran.github.io/plsRglm/`

Examples

```
set.seed(314)
library(plsRglm)
data(Cornell)
cv.modpls<-cv.plsR(Y~.,data=Cornell,nt=6,K=6)
res.cv.modpls<-cvtable(summary(cv.modpls))
```

---

aic.dof *Akaike and Bayesian Information Criteria and Generalized minimum description length*

Description

This function computes the Akaike and Bayesian Information Criteria and the Generalized minimum description length.

Usage

```
aic.dof(RSS, n, DoF, sigmahat)

bic.dof(RSS, n, DoF, sigmahat)

gmdl.dof(sigmahat, n, DoF, yhat)
```

Arguments

`RSS`	vector of residual sum of squares.
`n`	number of observations.
`DoF`	vector of Degrees of Freedom. The length of `DoF` is the same as the length of `RSS`.
`sigmahat`	Estimated model error. The length of `sigmahat` is the same as the length of `RSS`.
`yhat`	vector of squared norm of Yhat. The length of `yhat` is the same as the length of `sigmahat`.

Details

The gmdl criterion is defined as

$$gmdl = \frac{n}{2}log(S) + \frac{DoF}{2}log(F) + \frac{1}{2}log(n)$$

with

$$S = \hat{\sigma}^2$$

Value

`vector`	numerical values of the requested AIC, BIC or GMDL.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

M. Hansen, B. Yu. (2001). Model Selection and Minimum Descripion Length Principle, *Journal of the American Statistical Association*, 96, 746-774.
N. Kraemer, M. Sugiyama. (2011). The Degrees of Freedom of Partial Least Squares Regression. *Journal of the American Statistical Association*, 106(494), 697-705.
N. Kraemer, M.L. Braun, Kernelizing PLS, Degrees of Freedom, and Efficient Model Selection, *Proceedings of the 24th International Conference on Machine Learning*, Omni Press, (2007) 441-448.

See Also

`plsR.dof` for degrees of freedom computation and `infcrit.dof` for computing information criteria directly from a previously fitted plsR model.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsR(yCornell,XCornell,4)
dof.object <- plsR.dof(modpls)
aic.dof(modpls$RSS,modpls$nr,dof.object$DoF,dof.object$sigmahat)
bic.dof(modpls$RSS,modpls$nr,dof.object$DoF,dof.object$sigmahat)
gmdl.dof(dof.object$sigmahat,modpls$nr,dof.object$DoF,dof.object$yhat)
naive.object <- plsR.dof(modpls,naive=TRUE)
aic.dof(modpls$RSS,modpls$nr,naive.object$DoF,naive.object$sigmahat)
bic.dof(modpls$RSS,modpls$nr,naive.object$DoF,naive.object$sigmahat)
gmdl.dof(naive.object$sigmahat,modpls$nr,naive.object$DoF,naive.object$yhat)
```

`AICpls` *AIC function for plsR models*

Description

This function provides AIC computation for an univariate plsR model.

Usage

```
AICpls(ncomp, residpls, weights = rep.int(1, length(residpls)))
```

Arguments

`ncomp`	Number of components
`residpls`	Residuals of a fitted univariate plsR model
`weights`	Weights of observations

Details

AIC function for plsR models with univariate response.

Value

`real`	AIC value

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Baibing Li, Julian Morris, Elaine B. Martin, Model selection for partial least squares regression, Chemometrics and Intelligent Laboratory Systems 64 (2002) 79-89, doi: 10.1016/S01697439(02)00051-5.

See Also

`loglikpls` for loglikelihood computations for plsR models and `AIC` for AIC computation for a linear models

Examples

```
data(pine)
ypine <- pine[,11]
Xpine <- pine[,1:10]
(Pinscaled <- as.data.frame(cbind(scale(ypine),scale(as.matrix(Xpine)))))
colnames(Pinscaled)[1] <- "yy"

lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)

modpls <- plsR(ypine,Xpine,10)
modpls$Std.Coeffs
lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)

AIC(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled))
print(logLik(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)))

sum(dnorm(modpls$RepY, modpls$Std.ValsPredictY, sqrt(mean(modpls$residY^2)), log=TRUE))
sum(dnorm(Pinscaled$yy,fitted(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)),
sqrt(mean(residuals(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled))^2)), log=TRUE))
loglikpls(modpls$residY)
loglikpls(residuals(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)))
AICpls(10,residuals(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)))
AICpls(10,modpls$residY)
```

`aze` *Microsatellites Dataset*

Description

This database was collected on patients carrying a colon adenocarcinoma. It has 104 observations on 33 binary qualitative explanatory variables and one response variable `y` representing the cancer stage according to the to Astler-Coller classification (Astler and Coller, 1954). This dataset has some missing data due to technical limits. A microsattelite is a non-coding DNA sequence.

Format

A data frame with 104 observations on the following 34 variables.

y the response: a binary vector (Astler-Coller score).

D2S138 a binary vector that indicates whether this microsatellite is altered or not.

D18S61 a binary vector that indicates whether this microsatellite is altered or not.

D16S422 a binary vector that indicates whether this microsatellite is altered or not.

D17S794 a binary vector that indicates whether this microsatellite is altered or not.

D6S264 a binary vector that indicates whether this microsatellite is altered or not.

D14S65 a binary vector that indicates whether this microsatellite is altered or not.

D18S53 a binary vector that indicates whether this microsatellite is altered or not.

D17S790 a binary vector that indicates whether this microsatellite is altered or not.

D1S225 a binary vector that indicates whether this microsatellite is altered or not.

D3S1282 a binary vector that indicates whether this microsatellite is altered or not.

D9S179 a binary vector that indicates whether this microsatellite is altered or not.

D5S430 a binary vector that indicates whether this microsatellite is altered or not.

D8S283 a binary vector that indicates whether this microsatellite is altered or not.

D11S916 a binary vector that indicates whether this microsatellite is altered or not.

D2S159 a binary vector that indicates whether this microsatellite is altered or not.

D16S408 a binary vector that indicates whether this microsatellite is altered or not.

D5S346 a binary vector that indicates whether this microsatellite is altered or not.

D10S191 a binary vector that indicates whether this microsatellite is altered or not.

D13S173 a binary vector that indicates whether this microsatellite is altered or not.

D6S275 a binary vector that indicates whether this microsatellite is altered or not.

D15S127 a binary vector that indicates whether this microsatellite is altered or not.

D1S305 a binary vector that indicates whether this microsatellite is altered or not.

D4S394 a binary vector that indicates whether this microsatellite is altered or not.

D20S107 a binary vector that indicates whether this microsatellite is altered or not.

D1S197 a binary vector that indicates whether this microsatellite is altered or not.

D1S207 a binary vector that indicates whether this microsatellite is altered or not.

D10S192 a binary vector that indicates whether this microsatellite is altered or not.

D3S1283 a binary vector that indicates whether this microsatellite is altered or not.

D4S414 a binary vector that indicates whether this microsatellite is altered or not.

D8S264 a binary vector that indicates whether this microsatellite is altered or not.

D22S928 a binary vector that indicates whether this microsatellite is altered or not.

TP53 a binary vector that indicates whether this microsatellite is altered or not.

D9S171 a binary vector that indicates whether this microsatellite is altered or not.

Source

Weber *et al.* (2007). Allelotyping analyzes of synchronous primary and metastasis CIN colon cancers identified different subtypes. *Int J Cancer*, 120(3), pages 524-32.

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Société Française de Statistique*, 151(2), pages 1-18.

Examples

```
data(aze)
str(aze)
```

`aze_compl` *As aze without missing values*

Description

This is a single imputation of the `aze` dataset which was collected on patients carrying a colon adenocarcinoma. It has 104 observations on 33 binary qualitative explanatory variables and one response variable `y` representing the cancer stage according to the to Astler-Coller classification (Astler and Coller, 1954). A microsattelite is a non-coding DNA sequence.

Format

A data frame with 104 observations on the following 34 variables.

y the response: a binary vector (Astler-Coller score).

D2S138 a binary vector that indicates whether this microsatellite is altered or not.

D18S61 a binary vector that indicates whether this microsatellite is altered or not.

D16S422 a binary vector that indicates whether this microsatellite is altered or not.

D17S794 a binary vector that indicates whether this microsatellite is altered or not.

D6S264 a binary vector that indicates whether this microsatellite is altered or not.

D14S65 a binary vector that indicates whether this microsatellite is altered or not.

D18S53 a binary vector that indicates whether this microsatellite is altered or not.

D17S790 a binary vector that indicates whether this microsatellite is altered or not.

D1S225 a binary vector that indicates whether this microsatellite is altered or not.

D3S1282 a binary vector that indicates whether this microsatellite is altered or not.

D9S179 a binary vector that indicates whether this microsatellite is altered or not.

D5S430 a binary vector that indicates whether this microsatellite is altered or not.

D8S283 a binary vector that indicates whether this microsatellite is altered or not.

D11S916 a binary vector that indicates whether this microsatellite is altered or not.

D2S159 a binary vector that indicates whether this microsatellite is altered or not.

D16S408 a binary vector that indicates whether this microsatellite is altered or not.

D5S346 a binary vector that indicates whether this microsatellite is altered or not.

D10S191 a binary vector that indicates whether this microsatellite is altered or not.

D13S173 a binary vector that indicates whether this microsatellite is altered or not.

D6S275 a binary vector that indicates whether this microsatellite is altered or not.

D15S127 a binary vector that indicates whether this microsatellite is altered or not.

D1S305 a binary vector that indicates whether this microsatellite is altered or not.

D4S394 a binary vector that indicates whether this microsatellite is altered or not.

D20S107 a binary vector that indicates whether this microsatellite is altered or not.

D1S197 a binary vector that indicates whether this microsatellite is altered or not.

D1S207 a binary vector that indicates whether this microsatellite is altered or not.

D10S192 a binary vector that indicates whether this microsatellite is altered or not.

D3S1283 a binary vector that indicates whether this microsatellite is altered or not.

D4S414 a binary vector that indicates whether this microsatellite is altered or not.

D8S264 a binary vector that indicates whether this microsatellite is altered or not.

D22S928 a binary vector that indicates whether this microsatellite is altered or not.

TP53 a binary vector that indicates whether this microsatellite is altered or not.

D9S171 a binary vector that indicates whether this microsatellite is altered or not.

Source

Weber *et al.* (2007). Allelotyping analyzes of synchronous primary and metastasis CIN colon cancers identified different subtypes. *Int J Cancer*, 120(3), pages 524-32.

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Société Française de Statistique*, 151(2), pages 1-18.

Examples

```
data(aze_compl)
str(aze_compl)
```

---

bootpls *Non-parametric Bootstrap for PLS models*

Description

Provides a wrapper for the bootstrap function `boot` from the `boot` R package.
Implements non-parametric bootstraps for PLS Regression models by either (Y,X) or (Y,T) resampling.

Usage

```
bootpls(
  object,
  typeboot = "plsmodel",
  R = 250,
  statistic = coefs.plsR,
  sim = "ordinary",
  stype = "i",
  stabvalue = 1e+06,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	An object of class `plsRmodel` to bootstrap
`typeboot`	The type of bootstrap. Either (Y,X) boostrap (`typeboot="plsmodel"`) or (Y,T) bootstrap (`typeboot="fmodel_np"`). Defaults to (Y,X) resampling.

`R`	The number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case `R` would be a vector of integers where each component gives the number of resamples from each of the rows of weights.
`statistic`	A function which when applied to data returns a vector containing the statistic(s) of interest. `statistic` must take at least two arguments. The first argument passed will always be the original data. The second will be a vector of indices, frequencies or weights which define the bootstrap sample. Further, if predictions are required, then a third argument is required which would be a vector of the random indices used to generate the bootstrap predictions. Any further arguments can be passed to statistic through the `...` argument.
`sim`	A character string indicating the type of simulation required. Possible values are `"ordinary"` (the default), `"balanced"`, `"permutation"`, or `"antithetic"`.
`stype`	A character string indicating what the second argument of `statistic` represents. Possible values of stype are `"i"` (indices - the default), `"f"` (frequencies), or `"w"` (weights).
`stabvalue`	A value to hard threshold bootstrap estimates computed from atypical resamplings. Especially useful for Generalized Linear Models.
`verbose`	should info messages be displayed ?
`...`	Other named arguments for `statistic` which are passed unchanged each time it is called. Any such arguments to `statistic` should follow the arguments which `statistic` is required to have for the simulation. Beware of partial matching to arguments of `boot` listed above.

Details

More details on bootstrap techniques are available in the help of the `boot` function.

Value

An object of class `"boot"`. See the Value part of the help of the function `boot`.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

A. Lazraq, R. Cleroux, and J.-P. Gauchi. (2003). Selecting both latent and explanatory variables in the PLS1 regression model. *Chemometrics and Intelligent Laboratory Systems*, 66(2):117-126.
P. Bastien, V. Esposito-Vinzi, and M. Tenenhaus. (2005). PLS generalised linear regression. *Computational Statistics & Data Analysis*, 48(1):17-46.
A. C. Davison and D. V. Hinkley. (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge.

See Also

boot

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

# Lazraq-Cleroux PLS ordinary bootstrap
set.seed(250)
modpls <- plsR(yCornell,XCornell,3)

#(Y,X) resampling
Cornell.bootYX <- bootpls(modpls, R=250, verbose=FALSE)

#(Y,T) resampling
Cornell.bootYT <- bootpls(modpls, typeboot="fmodel_np", R=250, verbose=FALSE)

# Using the boxplots.bootpls function
boxplots.bootpls(Cornell.bootYX,indices=2:8)
# Confidence intervals plotting
confints.bootpls(Cornell.bootYX,indices=2:8)
plots.confints.bootpls(confints.bootpls(Cornell.bootYX,indices=2:8))
# Graph similar to the one of Bastien et al. in CSDA 2005
boxplot(as.vector(Cornell.bootYX$t[,-1])~factor(rep(1:7,rep(250,7))),
main="Bootstrap distributions of standardised bj (j = 1, ..., 7).")
points(c(1:7),Cornell.bootYX$t0[-1],col="red",pch=19)



library(boot)
boot.ci(Cornell.bootYX, conf = c(0.90,0.95), type = c("norm","basic","perc","bca"), index=2)
plot(Cornell.bootYX,index=2)
jack.after.boot(Cornell.bootYX, index=2, useJ=TRUE, nt=3)
plot(Cornell.bootYX,index=2,jack=TRUE)

car::dataEllipse(Cornell.bootYX$t[,2], Cornell.bootYX$t[,3], cex=.3,
levels=c(.5, .95, .99), robust=TRUE)
rm(Cornell.bootYX)


# PLS balanced bootstrap

set.seed(225)
Cornell.bootYX <- bootpls(modpls, sim="balanced", R=250, verbose=FALSE)
boot.array(Cornell.bootYX, indices=TRUE)

# Using the boxplots.bootpls function
boxplots.bootpls(Cornell.bootYX,indices=2:8)
# Confidence intervals plotting
```

```
confints.bootpls(Cornell.bootYX,indices=2:8)
plots.confints.bootpls(confints.bootpls(Cornell.bootYX,indices=2:8))
# Graph similar to the one of Bastien et al. in CSDA 2005
boxplot(as.vector(Cornell.bootYX$t[,-1])~factor(rep(1:7,rep(250,7))),
main="Bootstrap distributions of standardised bj (j = 1, ..., 7).")
points(c(1:7),Cornell.bootYX$t0[-1],col="red",pch=19)


library(boot)
boot.ci(Cornell.bootYX, conf = c(0.90,0.95), type = c("norm","basic","perc","bca"),
index=2, verbose=FALSE)
plot(Cornell.bootYX,index=2)
jack.after.boot(Cornell.bootYX, index=2, useJ=TRUE, nt=3)
plot(Cornell.bootYX,index=2,jack=TRUE)

rm(Cornell.bootYX)

# PLS permutation bootstrap

set.seed(500)
Cornell.bootYX <- bootpls(modpls, sim="permutation", R=1000, verbose=FALSE)
boot.array(Cornell.bootYX, indices=TRUE)


# Graph of bootstrap distributions
boxplot(as.vector(Cornell.bootYX$t[,-1])~factor(rep(1:7,rep(1000,7))),
main="Bootstrap distributions of standardised bj (j = 1, ..., 7).")
points(c(1:7),Cornell.bootYX$t0[-1],col="red",pch=19)
# Using the boxplots.bootpls function
boxplots.bootpls(Cornell.bootYX,indices=2:8)


library(boot)
plot(Cornell.bootYX,index=2)

qqnorm(Cornell.bootYX$t[,2],ylim=c(-1,1))
abline(h=Cornell.bootYX$t0[2],lty=2)
(sum(abs(Cornell.bootYX$t[,2])>=abs(Cornell.bootYX$t0[2]))+1)/(length(Cornell.bootYX$t[,2])+1)

rm(Cornell.bootYX)
```

---

bootplsglm — *Non-parametric Bootstrap for PLS generalized linear models*

Description

Provides a wrapper for the bootstrap function boot from the boot R package.
Implements non-parametric bootstraps for PLS Generalized Linear Regression models by either (Y,X) or (Y,T) resampling.

Usage

```
bootplsglm(
  object,
  typeboot = "fmodel_np",
  R = 250,
  statistic = coefs.plsRglmnp,
  sim = "ordinary",
  stype = "i",
  stabvalue = 1e+06,
  verbose = TRUE,
  ...
)
```

Arguments

`object`	An object of class `plsRglmmodel` to bootstrap
`typeboot`	The type of bootstrap. Either (Y,X) boostrap (`typeboot="plsmodel"`) or (Y,T) bootstrap (`typeboot="fmodel_np"`). Defaults to (Y,T) resampling.
`R`	The number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case `R` would be a vector of integers where each component gives the number of resamples from each of the rows of weights.
`statistic`	A function which when applied to data returns a vector containing the statistic(s) of interest. `statistic` must take at least two arguments. The first argument passed will always be the original data. The second will be a vector of indices, frequencies or weights which define the bootstrap sample. Further, if predictions are required, then a third argument is required which would be a vector of the random indices used to generate the bootstrap predictions. Any further arguments can be passed to statistic through the `...` argument.
`sim`	A character string indicating the type of simulation required. Possible values are `"ordinary"` (the default), `"balanced"`, `"permutation"`, or `"antithetic"`.
`stype`	A character string indicating what the second argument of `statistic` represents. Possible values of stype are `"i"` (indices - the default), `"f"` (frequencies), or `"w"` (weights).
`stabvalue`	A value to hard threshold bootstrap estimates computed from atypical resamplings. Especially useful for Generalized Linear Models.
`verbose`	should info messages be displayed ?
`...`	Other named arguments for `statistic` which are passed unchanged each time it is called. Any such arguments to `statistic` should follow the arguments which `statistic` is required to have for the simulation. Beware of partial matching to arguments of `boot` listed above.

Details

More details on bootstrap techniques are available in the help of the `boot` function.

Value

An object of class "boot". See the Value part of the help of the function boot.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

A. Lazraq, R. Cleroux, and J.-P. Gauchi. (2003). Selecting both latent and explanatory variables in the PLS1 regression model. *Chemometrics and Intelligent Laboratory Systems*, 66(2):117-126.
P. Bastien, V. Esposito-Vinzi, and M. Tenenhaus. (2005). PLS generalised linear regression. *Computational Statistics & Data Analysis*, 48(1):17-46.
A. C. Davison and D. V. Hinkley. (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press, Cambridge.

See Also

boot

Examples

```
#Imputed aze dataset
data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y

dataset <- cbind(y=yaze_compl,Xaze_compl)
modplsglm <- plsRglm(y~.,data=dataset,3,modele="pls-glm-logistic")

library(boot)
# Bastien (Y,T) PLS bootstrap
aze_compl.bootYT <- bootplsglm(modplsglm, R=250, verbose=FALSE)
boxplots.bootpls(aze_compl.bootYT)
confints.bootpls(aze_compl.bootYT)
plots.confints.bootpls(confints.bootpls(aze_compl.bootYT))


plot(aze_compl.bootYT,index=2)
jack.after.boot(aze_compl.bootYT, index=2, useJ=TRUE, nt=3)
plot(aze_compl.bootYT, index=2,jack=TRUE)
aze_compl.tilt.boot <- tilt.bootplsglm(modplsglm, statistic=coefs.plsRglm,
R=c(499, 100, 100), alpha=c(0.025, 0.975), sim="ordinary", stype="i", index=1)

# PLS bootstrap balanced
aze_compl.bootYT <- bootplsglm(modplsglm, sim="balanced", R=250, verbose=FALSE)
boxplots.bootpls(aze_compl.bootYT)
confints.bootpls(aze_compl.bootYT)
```

```
plots.confints.bootpls(confints.bootpls(aze_compl.bootYT))


plot(aze_compl.bootYT)
jack.after.boot(aze_compl.bootYT, index=1, useJ=TRUE, nt=3)
plot(aze_compl.bootYT,jack=TRUE)
aze_compl.tilt.boot <- tilt.bootplsglm(modplsglm, statistic=coefs.plsR,
R=c(499, 100, 100), alpha=c(0.025, 0.975), sim="balanced", stype="i", index=1)


# PLS permutation bootstrap

aze_compl.bootYT <- bootplsglm(modplsglm, sim="permutation", R=250, verbose=FALSE)
boxplots.bootpls(aze_compl.bootYT)
plot(aze_compl.bootYT)


#Original aze dataset with missing values
data(aze)
Xaze<-aze[,2:34]
yaze<-aze$y

library(boot)
modplsglm2 <- plsRglm(yaze,Xaze,3,modele="pls-glm-logistic")
aze.bootYT <- bootplsglm(modplsglm2, R=250, verbose=FALSE)
boxplots.bootpls(aze.bootYT)
confints.bootpls(aze.bootYT)
plots.confints.bootpls(confints.bootpls(aze.bootYT))




#Ordinal logistic regression
data(bordeaux)
Xbordeaux<-bordeaux[,1:4]
ybordeaux<-factor(bordeaux$Quality,ordered=TRUE)
dataset <- cbind(y=ybordeaux,Xbordeaux)
options(contrasts = c("contr.treatment", "contr.poly"))
modplsglm3 <- plsRglm(ybordeaux,Xbordeaux,1,modele="pls-glm-polr")
bordeaux.bootYT<- bootplsglm(modplsglm3, sim="permutation", R=250, verbose=FALSE)
boxplots.bootpls(bordeaux.bootYT)
boxplots.bootpls(bordeaux.bootYT,ranget0=TRUE)

bordeaux.bootYT2<- bootplsglm(modplsglm3, sim="permutation", R=250,
strata=unclass(ybordeaux), verbose=FALSE)
boxplots.bootpls(bordeaux.bootYT2,ranget0=TRUE)


if(require(chemometrics)){
data(hyptis)
hyptis
yhyptis <- factor(hyptis$Group,ordered=TRUE)
Xhyptis <- as.data.frame(hyptis[,c(1:6)])
dataset <- cbind(y=yhyptis,Xhyptis)
options(contrasts = c("contr.treatment", "contr.poly"))
modplsglm4 <- plsRglm(yhyptis,Xhyptis,3,modele="pls-glm-polr")
hyptis.bootYT3<- bootplsglm(modplsglm4, sim="permutation", R=250, verbose=FALSE)
rownames(hyptis.bootYT3$t0)<-c("Sabi\nnene","Pin\nene",
"Cine\nole","Terpi\nnene","Fenc\nhone","Terpi\nnolene")
boxplots.bootpls(hyptis.bootYT3)
boxplots.bootpls(hyptis.bootYT3,xaxisticks=FALSE)
boxplots.bootpls(hyptis.bootYT3,ranget0=TRUE)
boxplots.bootpls(hyptis.bootYT3,ranget0=TRUE,xaxisticks=FALSE)
}
```

bordeaux *Quality of wine dataset*

Description

Quality of Bordeaux wines (`Quality`) and four potentially predictive variables (`Temperature`, `Sunshine`, `Heat` and `Rain`).

Format

A data frame with 34 observations on the following 5 variables.

Temperature a numeric vector

Sunshine a numeric vector

Heat a numeric vector

Rain a numeric vector

Quality an ordered factor with levels `1` < `2` < `3`

Source

P. Bastien, V. Esposito-Vinzi, and M. Tenenhaus. (2005). PLS generalised linear regression. *Computational Statistics & Data Analysis*, 48(1):17-46.

References

M. Tenenhaus. (2005). La regression logistique PLS. In J.-J. Droesbeke, M. Lejeune, and G. Saporta, editors, Modeles statistiques pour donnees qualitatives. Editions Technip, Paris.

Examples

```
data(bordeaux)
str(bordeaux)
```

`bordeauxNA` *Quality of wine dataset*

Description

Quality of Bordeaux wines (`Quality`) and four potentially predictive variables (`Temperature`, `Sunshine`, `Heat` and `Rain`).

Format

A data frame with 34 observations on the following 5 variables.

Temperature a numeric vector

Sunshine a numeric vector

Heat a numeric vector

Rain a numeric vector

Quality an ordered factor with levels `1` < `2` < `3`

Details

The value of x1 for the first observation was removed from the matrix of predictors on purpose.

The `bordeauxNA` is a dataset with a missing value for testing purpose.

Source

P. Bastien, V. Esposito-Vinzi, and M. Tenenhaus. (2005). PLS generalised linear regression. *Computational Statistics & Data Analysis*, 48(1):17-46.

References

M. Tenenhaus. (2005). La regression logistique PLS. In J.-J. Droesbeke, M. Lejeune, and G. Saporta, editors, Modeles statistiques pour donnees qualitatives. Editions Technip, Paris.

Examples

```
data(bordeauxNA)
str(bordeauxNA)
```

boxplots.bootpls *Boxplot bootstrap distributions*

Description

Boxplots for bootstrap distributions.

Usage

```
boxplots.bootpls(
  bootobject,
  indices = NULL,
  prednames = TRUE,
  articlestyle = TRUE,
  xaxisticks = TRUE,
  ranget0 = FALSE,
  las = par("las"),
  mar,
  mgp,
  ...
)
```

Arguments

`bootobject`	a object of class `"boot"`
`indices`	vector of indices of the variables to plot. Defaults to `NULL`: all the predictors will be used.
`prednames`	do the original names of the predictors shall be plotted ? Defaults to `TRUE`: the names are plotted.
`articlestyle`	do the extra blank zones of the margin shall be removed from the plot ? Defaults to `TRUE`: the margins are removed.
`xaxisticks`	do ticks for the x axis shall be plotted ? Defaults to `TRUE`: the ticks are plotted.
`ranget0`	does the vertival range of the plot shall be computed to include the initial estimates of the coefficients ? Defaults to `FALSE`: the vertical range is calculated only using the bootstrapped values of the statistics. Especially using for permutation bootstrap.
`las`	numeric in 0,1,2,3; the style of axis labels. 0: always parallel to the axis [default], 1: always horizontal, 2: always perpendicular to the axis, 3: always vertical.
`mar`	A numerical vector of the form `c(bottom,left,top,right)` which gives the number of lines of margin to be specified on the four sides of the plot. The default is `c(5,4,4,2) + 0.1`.
`mgp`	The margin line (in mex units) for the axis title, axis labels and axis line. Note that `mgp[1]` affects title whereas `mgp[2:3]` affect axis. The default is `c(3,1,0)`.
`...`	further options to pass to the `boxplot` function.

Value

NULL

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

See Also

bootpls

Examples

```

data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

# Lazraq-Cleroux PLS ordinary bootstrap
set.seed(250)
modpls <- plsR(yCornell,XCornell,3)
Cornell.bootYX <- bootpls(modpls, R=250)

# Graph similar to the one of Bastien et al. in CSDA 2005
boxplots.bootpls(Cornell.bootYX,indices=2:8)


data(aze_compl)
modplsglm<-plsRglm(y~.,data=aze_compl,3,modele="pls-glm-logistic")
aze_compl.boot3 <- bootplsglm(modplsglm, R=250, verbose=FALSE)
boxplots.bootpls(aze_compl.boot3)
boxplots.bootpls(aze_compl.boot3,las=3,mar=c(5,2,1,1))
boxplots.bootpls(aze_compl.boot3,indices=c(2,4,6),prednames=FALSE)
```

---

coef.plsRglmmodel *coef method for plsR models*

Description

This function provides a coef method for the class "plsRglmmodel"

Usage

```
## S3 method for class 'plsRglmmodel'
coef(object, type = c("scaled", "original"), ...)
```

Arguments

`object`	an object of the class `"plsRglmmodel"`
`type`	if `scaled`, the coefficients of the predictors are given for the scaled predictors, if `original` the coefficients are to be used with the predictors on their original scale.
`...`	not used

Value

An object of class `coef.plsRglmmodel`.

`CoeffC`	Coefficients of the components.
`Std.Coeffs`	Coefficients of the scaled predictors in the regression function.
`Coeffs`	Coefficients of the untransformed predictors (on their original scale).

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

See Also

`coef`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsRglm(yCornell,XCornell,3,modele="pls-glm-family",family=gaussian())
class(modpls)
coef(modpls)
coef(modpls,type="scaled")
rm(list=c("XCornell","yCornell","modpls"))
```

`coef.plsRmodel` *coef method for plsR models*

Description

This function provides a coef method for the class `"plsRmodel"`

Usage

```
## S3 method for class 'plsRmodel'
coef(object, type = c("scaled", "original"), ...)
```

Arguments

`object`	an object of the class "`plsRmodel`"
`type`	if `scaled`, the coefficients of the predictors are given for the scaled predictors, if `original` the coefficients are to be used with the predictors on their original scale.
`...`	not used

Value

An object of class `coef.plsRmodel`.

`CoeffC`	Coefficients of the components.
`Std.Coeffs`	Coefficients of the scaled predictors.
`Coeffs`	Coefficients of the untransformed predictors (on their original scale).

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
https://fbertran.github.io/homepage/

See Also

`coef`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsRglm(yCornell,XCornell,3,modele="pls")
class(modpls)
coef(modpls)
coef(modpls,type="scaled")
rm(list=c("XCornell","yCornell","modpls"))
```

coefs.plsR *Coefficients for bootstrap computations of PLSR models*

Description

A function passed to `boot` to perform bootstrap.

Usage

```
coefs.plsR(dataset, ind, nt, modele, maxcoefvalues, ifbootfail, verbose)
```

Arguments

`dataset`	dataset to resample
`ind`	indices for resampling
`nt`	number of components to use
`modele`	type of modele to use, see plsR
`maxcoefvalues`	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
`ifbootfail`	value to return if the estimation fails on a bootstrap sample
`verbose`	should info messages be displayed ?

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

See also `bootpls`.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

# Lazraq-Cleroux PLS (Y,X) bootstrap
# statistic=coefs.plsR is the default for (Y,X) resampling of PLSR models.
set.seed(250)
modpls <- plsR(yCornell,XCornell,1)
```

```
Cornell.bootYX <- bootpls(modpls, R=250, statistic=coefs.plsR, verbose=FALSE)
```

coefs.plsRglm *Coefficients for bootstrap computations of PLSGLR models*

Description

A function passed to boot to perform bootstrap.

Usage

```
coefs.plsRglm(
  dataset,
  ind,
  nt,
  modele,
  family = NULL,
  maxcoefvalues,
  ifbootfail,
  verbose
)
```

Arguments

dataset	dataset to resample
ind	indices for resampling
nt	number of components to use
modele	type of modele to use, see plsRglm
family	glm family to use, see plsRglm
maxcoefvalues	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
ifbootfail	value to return if the estimation fails on a bootstrap sample
verbose	should info messages be displayed ?

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

See Also

See also bootplsglm.

Examples

```
data(Cornell)

# (Y,X) bootstrap of a PLSGLR model
# statistic=coefs.plsRglm is the default for (Y,X) bootstrap of a PLSGLR models.
set.seed(250)
modplsglm <- plsRglm(Y~.,data=Cornell,1,modele="pls-glm-family",family=gaussian)
Cornell.bootYX <- bootplsglm(modplsglm, R=250, typeboot="plsmodel",
statistic=coefs.plsRglm, verbose=FALSE)
```

`coefs.plsRglmnp` *Coefficients for bootstrap computations of PLSGLR models*

Description

A function passed to `boot` to perform bootstrap.

Usage

```
coefs.plsRglmnp(
  dataRepYtt,
  ind,
  nt,
  modele,
  family = NULL,
  maxcoefvalues,
  wwetoile,
  ifbootfail
)
```

Arguments

`dataRepYtt`	components' coordinates to bootstrap
`ind`	indices for resampling
`nt`	number of components to use
`modele`	type of modele to use, see plsRglm
`family`	glm family to use, see plsRglm
`maxcoefvalues`	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
`wwetoile`	values of the Wstar matrix in the original fit
`ifbootfail`	value to return if the estimation fails on a bootstrap sample

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Note

~~some notes~~

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

See also `bootplsglm`

Examples

```
data(Cornell)

# (Y,X) bootstrap of a PLSGLR model
# statistic=coefs.plsRglm is the default for (Y,X) bootstrap of a PLSGLR models.
set.seed(250)
modplsglm <- plsRglm(Y~.,data=Cornell,1,modele="pls-glm-family",family=gaussian)
Cornell.bootYT <- bootplsglm(modplsglm, R=250, statistic=coefs.plsRglmnp, verbose=FALSE)
```

`coefs.plsRnp` *Coefficients for bootstrap computations of PLSR models*

Description

A function passed to `boot` to perform bootstrap.

Usage

```
coefs.plsRnp(dataRepYtt, ind, nt, modele, maxcoefvalues, wwetoile, ifbootfail)
```

Arguments

`dataRepYtt`	components' coordinates to bootstrap
`ind`	indices for resampling
`nt`	number of components to use
`modele`	type of modele to use, see plsRglm

`maxcoefvalues`	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
`wwetoile`	values of the Wstar matrix in the original fit
`ifbootfail`	value to return if the estimation fails on a bootstrap sample

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

See also `bootpls`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

# Lazraq-Cleroux PLS (Y,X) bootstrap
# statistic=coefs.plsR is the default for (Y,X) resampling of PLSR models.
set.seed(250)
modpls <- plsR(yCornell,XCornell,1)
Cornell.bootYT <- bootpls(modpls, R=250, typeboot="fmodel_np",
statistic=coefs.plsRnp, verbose=FALSE)
```

`confints.bootpls` *Bootstrap confidence intervals*

Description

This function is a wrapper for `boot.ci` to derive bootstrap-based confidence intervals from a `"boot"` object.

Usage

```
confints.bootpls(bootobject, indices = NULL, typeBCa = TRUE)
```

Arguments

`bootobject`	an object of class `"boot"`
`indices`	the indices of the predictor for which CIs should be calculated. Defaults to `NULL`: all the predictors will be used.
`typeBCa`	shall BCa bootstrap based CI derived ? Defaults to `TRUE`. This is a safety option since sometimes computing BCa bootstrap based CI fails whereas the other types of CI can still be derived.

Value

Matrix with the limits of bootstrap based CI for all (defaults) or only the selected predictors (`indices` option). The limits are given in that order: Normal Lower then Upper Limit, Basic Lower then Upper Limit, Percentile Lower then Upper Limit, BCa Lower then Upper Limit.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

See also `bootpls` and `bootplsglm`.

Examples

```
data(Cornell)

#Lazraq-Cleroux PLS (Y,X) bootstrap
set.seed(250)
modpls <- plsR(Y~.,data=Cornell,3)
Cornell.bootYX <- bootpls(modpls, R=250, verbose=FALSE)
confints.bootpls(Cornell.bootYX,2:8)
confints.bootpls(Cornell.bootYX,2:8,typeBCa=FALSE)
```

`CorMat` *Correlation matrix for simulating plsR datasets*

Description

A correlation matrix to simulate datasets

Format

A data frame with 17 observations on the following 17 variables.

y a numeric vector

x11 a numeric vector

x12 a numeric vector

x13 a numeric vector

x21 a numeric vector

x22 a numeric vector

x31 a numeric vector

x32 a numeric vector

x33 a numeric vector

x34 a numeric vector

x41 a numeric vector

x42 a numeric vector

x51 a numeric vector

x61 a numeric vector

x62 a numeric vector

x63 a numeric vector

x64 a numeric vector

Source

Handmade.

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

Examples

```
data(CorMat)
str(CorMat)
```

`Cornell` *Cornell dataset*

Description

The famous Cornell dataset. A mixture experiment on `X1`, `X2`, `X3`, `X4`, `X5`, `X6` and `X7` to analyse octane degree (`Y`) in gazoline.

Format

A data frame with 12 observations on the following 8 variables.

X1 a numeric vector

X2 a numeric vector

X3 a numeric vector

X4 a numeric vector

X5 a numeric vector

X6 a numeric vector

X7 a numeric vector

Y response value: a numeric vector

Source

M. Tenenhaus. (1998). *La regression PLS, Theorie et pratique*. Editions Technip, Paris.

References

N. Kettaneh-Wold. Analysis of mixture data with partial least squares. (1992). *Chemometrics and Intelligent Laboratory Systems*, 14(1):57-69.

Examples

```
data(Cornell)
str(Cornell)
```

cv.plsR	*Partial least squares regression models with k-fold cross-validation*

Description

This function implements k-fold cross-validation on complete or incomplete datasets for partial least squares regression models

Usage

```
cv.plsR(x, ...)
## Default S3 method:
cv.plsRmodel(dataY,dataX,nt=2,limQ2set=.0975,modele="pls",
K=5, NK=1, grouplist=NULL, random=TRUE, scaleX=TRUE,
scaleY=NULL, keepcoeffs=FALSE, keepfolds=FALSE, keepdataY=TRUE,
keepMclassed=FALSE, tol_Xi=10^(-12), weights, verbose=TRUE)
## S3 method for class 'formula'
cv.plsRmodel(formula,data=NULL,nt=2,limQ2set=.0975,modele="pls",
K=5, NK=1, grouplist=NULL, random=TRUE, scaleX=TRUE,
scaleY=NULL, keepcoeffs=FALSE, keepfolds=FALSE, keepdataY=TRUE,
keepMclassed=FALSE, tol_Xi=10^(-12), weights,subset,contrasts=NULL, verbose=TRUE)
PLS_lm_kfoldcv(dataY, dataX, nt = 2, limQ2set = 0.0975, modele = "pls",
K = 5, NK = 1, grouplist = NULL, random = TRUE, scaleX = TRUE,
scaleY = NULL, keepcoeffs = FALSE, keepfolds = FALSE, keepdataY = TRUE,
keepMclassed=FALSE, tol_Xi = 10^(-12), weights, verbose=TRUE)
PLS_lm_kfoldcv_formula(formula,data=NULL,nt=2,limQ2set=.0975,modele="pls",
K=5, NK=1, grouplist=NULL, random=TRUE, scaleX=TRUE,
scaleY=NULL, keepcoeffs=FALSE, keepfolds=FALSE, keepdataY=TRUE,
keepMclassed=FALSE, tol_Xi=10^(-12), weights,subset,contrasts=NULL,verbose=TRUE)
```

Arguments

`x`	a formula or a response (training) dataset
`dataY`	response (training) dataset
`dataX`	predictor(s) (training) dataset
`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `plsRglm` is called.
`nt`	number of components to be extracted
`limQ2set`	limit value for the Q2
`modele`	name of the PLS model to be fitted, only (`"pls"` available for this fonction.

`K`	number of groups. Defaults to 5.
`NK`	number of times the group division is made
`grouplist`	to specify the members of the `K` groups
`random`	should the `K` groups be made randomly. Defaults to `TRUE`
`scaleX`	scale the predictor(s) : must be set to TRUE for `modele="pls"` and should be for glms pls.
`scaleY`	scale the response : Yes/No. Ignored since non always possible for glm responses.
`keepcoeffs`	shall the coefficients for each model be returned
`keepfolds`	shall the groups' composition be returned
`keepdataY`	shall the observed value of the response for each one of the predicted value be returned
`keepMclassed`	shall the number of miss classed be returned
`tol_Xi`	minimal value for Norm2(Xi) and $\det(pp' \times pp)$ if there is any missing value in the `dataX`. It defaults to 10^{-12}
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`verbose`	should info messages be displayed ?
`...`	arguments to pass to `cv.plsRmodel.default` or to `cv.plsRmodel.formula`

Details

Predicts 1 group with the `K-1` other groups. Leave one out cross validation is thus obtained for `K==nrow(dataX)`.

A typical predictor has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with any duplicates removed.

A specification of the form first:second indicates the the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second.

The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula.

Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations.

Value

An object of class "cv.plsRmodel".

`results_kfolds` list of NK. Each element of the list sums up the results for a group division:

list of K matrices of size about nrow(dataX)/K * nt with the predicted values for a growing number of components

... ...

list of K matrices of size about nrow(dataX)/K * nt with the predicted values for a growing number of components

`folds` list of NK. Each element of the list sums up the results for a group division:

list of K vectors of length about nrow(dataX) with the numbers of the rows of dataX that were used as a training set

... ...

list of K vectors of length about nrow(dataX) with the numbers of the rows of dataX that were used as a training set

`dataY_kfolds` list of NK. Each element of the list sums up the results for a group division:

list of K matrices of size about nrow(dataX)/K * 1 with the observed values of the response

... ...

list of K matrices of size about nrow(dataX)/K * 1 with the observed values of the response

`call` the call of the function

Note

Work for complete and incomplete datasets.

Author(s)

Frederic Bertrand
<frederic.bertrand@math.unistra.fr>
http://www-irma.u-strasbg.fr/~fbertran/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frederic Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

Summary method summary.cv.plsRmodel. kfolds2coeff, kfolds2Pressind, kfolds2Press, kfolds2Mclassedind, kfolds2Mclassed and kfolds2CVinfos_lm to extract and transform results from k-fold cross-validation.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

#Leave one out CV (K=nrow(Cornell)) one time (NK=1)
bbb <- cv.plsR(dataY=yCornell,dataX=XCornell,nt=6,K=nrow(Cornell),NK=1)
bbb2 <- cv.plsR(Y~.,data=Cornell,nt=6,K=12,NK=1,verbose=FALSE)
(sum1<-summary(bbb2))

#6-fold CV (K=6) two times (NK=2)
#use random=TRUE to randomly create folds for repeated CV
bbb3 <- cv.plsR(dataY=yCornell,dataX=XCornell,nt=6,K=6,NK=2)
bbb4 <- cv.plsR(Y~.,data=Cornell,nt=6,K=6,NK=2,verbose=FALSE)
(sum3<-summary(bbb4))

cvtable(sum1)
cvtable(sum3)
rm(list=c("XCornell","yCornell","bbb","bbb2","bbb3","bbb4"))
```

`cv.plsRglm` *Partial least squares regression glm models with k-fold cross validation*

Description

This function implements k-fold cross-validation on complete or incomplete datasets for partial least squares regression generalized linear models

Usage

```
cv.plsRglm(x, ...)
## Default S3 method:
cv.plsRglmmodel(dataY,dataX,nt=2,limQ2set=.0975,
modele="pls", family=NULL, K=5, NK=1, grouplist=NULL, random=TRUE,
scaleX=TRUE, scaleY=NULL, keepcoeffs=FALSE, keepfolds=FALSE,
keepdataY=TRUE, keepMclassed=FALSE, tol_Xi=10^(-12), weights, method,
verbose=TRUE)
## S3 method for class 'formula'
cv.plsRglmmodel(formula,data=NULL,nt=2,limQ2set=.0975,
modele="pls", family=NULL, K=5, NK=1, grouplist=NULL, random=TRUE,
scaleX=TRUE, scaleY=NULL, keepcoeffs=FALSE, keepfolds=FALSE,
keepdataY=TRUE, keepMclassed=FALSE, tol_Xi=10^(-12),weights,subset,
start=NULL,etastart,mustart,offset,method,control= list(),contrasts=NULL,
verbose=TRUE)
PLS_glm_kfoldcv(dataY, dataX, nt = 2, limQ2set = 0.0975, modele = "pls",
family = NULL, K = 5, NK = 1, grouplist = NULL, random = TRUE,
scaleX = TRUE, scaleY = NULL, keepcoeffs = FALSE, keepfolds = FALSE,
keepdataY = TRUE, keepMclassed=FALSE, tol_Xi = 10^(-12), weights, method,
verbose=TRUE)
PLS_glm_kfoldcv_formula(formula,data=NULL,nt=2,limQ2set=.0975,modele="pls",
family=NULL, K=5, NK=1, grouplist=NULL, random=TRUE,
scaleX=TRUE, scaleY=NULL, keepcoeffs=FALSE, keepfolds=FALSE, keepdataY=TRUE,
keepMclassed=FALSE, tol_Xi=10^(-12),weights,subset,start=NULL,etastart,
mustart,offset,method,control= list(),contrasts=NULL, verbose=TRUE)
```

Arguments

`x`	a formula or a response (training) dataset
`dataY`	response (training) dataset
`dataX`	predictor(s) (training) dataset
`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `plsRglm` is called.
`nt`	number of components to be extracted
`limQ2set`	limit value for the Q2
`modele`	name of the PLS glm model to be fitted (`"pls"`, `"pls-glm-Gamma"`, `"pls-glm-gaussian"`, `"pls-glm-inverse.gaussian"`, `"pls-glm-logistic"`, `"pls-glm-poisson"`, `"pls-glm-polr"`). Use `"modele=pls-glm-family"` to enable the `family` option.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.) To use the family option, please set `modele="pls-glm-family"`. User defined families can also be defined. See details.
`K`	number of groups. Defaults to 5.
`NK`	number of times the group division is made
`grouplist`	to specify the members of the `K` groups
`random`	should the `K` groups be made randomly. Defaults to `TRUE`
`scaleX`	scale the predictor(s) : must be set to TRUE for `modele="pls"` and should be for glms pls.
`scaleY`	scale the response : Yes/No. Ignored since non always possible for glm responses.
`keepcoeffs`	shall the coefficients for each model be returned
`keepfolds`	shall the groups' composition be returned
`keepdataY`	shall the observed value of the response for each one of the predicted value be returned

`keepMclassed`	shall the number of miss classed be returned (unavailable)
`tol_Xi`	minimal value for Norm2(Xi) and $\det(pp' \times pp)$ if there is any missing value in the `dataX`. It defaults to 10^{-12}
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`start`	starting values for the parameters in the linear predictor.
`etastart`	starting values for the linear predictor.
`mustart`	starting values for the vector of means.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. One or more `offset` terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See `model.offset`.
`method`	**for fitting glms with glm (**`"pls-glm-Gamma"`**,** `"pls-glm-gaussian"`**,** `"pls-glm-inverse.gaussian"`**,** the method to be used in fitting the model. The default method `"glm.fit"` uses iteratively reweighted least squares (IWLS). User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as `glm.fit`. If "model.frame", the model frame is returned. `pls-glm-polr` logistic, probit, complementary log-log or cauchit (corresponding to a Cauchy latent variable).
`control`	a list of parameters for controlling the fitting process. For `glm.fit` this is passed to `glm.control`.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`verbose`	should info messages be displayed ?
`...`	arguments to pass to `cv.plsRglmmodel.default` or to `cv.plsRglmmodel.formula`

Details

Predicts 1 group with the `K-1` other groups. Leave one out cross validation is thus obtained for `K==nrow(dataX)`.

There are seven different predefined models with predefined link functions available :

`"pls"` ordinary pls models

`"pls-glm-Gamma"` glm gaussian with inverse link pls models

`"pls-glm-gaussian"` glm gaussian with identity link pls models

`"pls-glm-inverse-gamma"` glm binomial with square inverse link pls models

`"pls-glm-logistic"` glm binomial with logit link pls models

`"pls-glm-poisson"` glm poisson with log link pls models

`"pls-glm-polr"` glm polr with logit link pls models

Using the `"family="` option and setting `"modele=pls-glm-family"` allows changing the family and link function the same way as for the `glm` function. As a consequence user-specified families can also be used.

The `gaussian` **family** accepts the links (as names) `identity`, `log` and `inverse`.

The `binomial` **family** accepts the links `logit`, `probit`, `cauchit`, (corresponding to logistic, normal and Cauchy CDFs respectively) `log` and `cloglog` (complementary log-log).

The `Gamma` **family** accepts the links `inverse`, `identity` and `log`.

The `poisson` **family** accepts the links `log`, `identity`, and `sqrt`.

The `inverse.gaussian` **family** accepts the links `1/mu^2`, `inverse`, `identity` and `log`.

The `quasi` **family** accepts the links `logit`, `probit`, `cloglog`, `identity`, `inverse`, `log`, `1/mu^2` and `sqrt`.

The function `power` can be used to create a power link function.

... arguments to pass to `cv.plsRglmmodel.default` or to `cv.plsRglmmodel.formula`

A typical predictor has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with any duplicates removed.

A specification of the form first:second indicates the the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second.

The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula.

Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations.

Value

An object of class `"cv.plsRglmmodel"`.

`results_kfolds`	list of `NK`. Each element of the list sums up the results for a group division:
	list of `K` matrices of size about `nrow(dataX)/K * nt` with the predicted values for a growing number of components
	... ...
	list of `K` matrices of size about `nrow(dataX)/K * nt` with the predicted values for a growing number of components
`folds`	list of `NK`. Each element of the list sums up the informations for a group division:
	list of `K` vectors of length about `nrow(dataX)` with the numbers of the rows of `dataX` that were used as a training set
	... ...

list of K vectors of length about nrow(dataX) with the numbers of the rows of dataX that were used as a training set

dataY_kfolds	list of NK. Each element of the list sums up the results for a group division:

list of K matrices of size about nrow(dataX)/K * 1 with the observed values of the response

... ...

list of K matrices of size about nrow(dataX)/K * 1 with the observed values of the response

call	the call of the function

Note

Work for complete and incomplete datasets.

Author(s)

Frederic Bertrand
<frederic.bertrand@math.unistra.fr>
http://www-irma.u-strasbg.fr/~fbertran/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frederic Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18.

See Also

Summary method summary.cv.plsRglmmodel. kfolds2coeff, kfolds2Pressind, kfolds2Press, kfolds2Mclassedind, kfolds2Mclassed and summary to extract and transform results from k-fold cross validation.

Examples

```
data(Cornell)
bbb <- cv.plsRglm(Y~.,data=Cornell,nt=10)
(sum1<-summary(bbb))
cvtable(sum1)

bbb2 <- cv.plsRglm(Y~.,data=Cornell,nt=3,
modele="pls-glm-family",family=gaussian(),K=12,verbose=FALSE)
(sum2<-summary(bbb2))
cvtable(sum2)


#random=TRUE is the default to randomly create folds for repeated CV
bbb3 <- cv.plsRglm(Y~.,data=Cornell,nt=3,
modele="pls-glm-family",family=gaussian(),K=6,NK=10, verbose=FALSE)
(sum3<-summary(bbb3))
plot(cvtable(sum3))
```

```
data(aze_compl)
bbb <- cv.plsRglm(y~.,data=aze_compl,nt=10,K=10,modele="pls",keepcoeffs=TRUE, verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb)
bbb2 <- cv.plsRglm(y~.,data=aze_compl,nt=10,K=10,modele="pls-glm-family",
family=binomial(probit),keepcoeffs=TRUE, verbose=FALSE)
bbb2 <- cv.plsRglm(y~.,data=aze_compl,nt=10,K=10,
modele="pls-glm-logistic",keepcoeffs=TRUE, verbose=FALSE)
summary(bbb,MClassed=TRUE)
summary(bbb2,MClassed=TRUE)
kfolds2coeff(bbb2)

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
rm(list=c("bbb","bbb2"))



data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
bbb <- cv.plsRglm(round(x11)~.,data=pine,nt=10,modele="pls-glm-family",
family=poisson(log),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb <- cv.plsRglm(round(x11)~.,data=pine,nt=10,
modele="pls-glm-poisson",K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb)
boxplot(kfolds2coeff(bbb)[,1])

kfolds2Chisqind(bbb)
kfolds2Chisq(bbb)
summary(bbb)
PLS_lm(ypine,Xpine,10,typeVC="standard")$InfCrit

data(pineNAX21)
bbb2 <- cv.plsRglm(round(x11)~.,data=pineNAX21,nt=10,
modele="pls-glm-family",family=poisson(log),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb2 <- cv.plsRglm(round(x11)~.,data=pineNAX21,nt=10,
modele="pls-glm-poisson",K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)

data(XpineNAX21)
```

```
PLS_lm(ypine,XpineNAX21,10,typeVC="standard")$InfCrit
rm(list=c("Xpine","XpineNAX21","ypine","bbb","bbb2"))



data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
bbb <- cv.plsRglm(x11~.,data=pine,nt=10,modele="pls-glm-family",
family=Gamma,K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb <- cv.plsRglm(x11~.,data=pine,nt=10,modele="pls-glm-Gamma",
K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb)
boxplot(kfolds2coeff(bbb)[,1])

kfolds2Chisqind(bbb)
kfolds2Chisq(bbb)
summary(bbb)
PLS_lm(ypine,Xpine,10,typeVC="standard")$InfCrit

data(pineNAX21)
bbb2 <- cv.plsRglm(x11~.,data=pineNAX21,nt=10,
modele="pls-glm-family",family=Gamma(),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb2 <- cv.plsRglm(x11~.,data=pineNAX21,nt=10,
modele="pls-glm-Gamma",K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
XpineNAX21 <- Xpine
XpineNAX21[1,2] <- NA
PLS_lm(ypine,XpineNAX21,10,typeVC="standard")$InfCrit
rm(list=c("Xpine","XpineNAX21","ypine","bbb","bbb2"))



data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
bbb <- cv.plsRglm(Y~.,data=Cornell,nt=10,NK=1,modele="pls",verbose=FALSE)
summary(bbb)

cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-inverse.gaussian",K=12,verbose=FALSE)
cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-family",
family=inverse.gaussian,K=12,verbose=FALSE)
cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-inverse.gaussian",K=6,
NK=2,verbose=FALSE)$results_kfolds
```

```
cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-family",family=inverse.gaussian(),
K=6,NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-inverse.gaussian",K=6,
NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-family",
family=inverse.gaussian(link = "1/mu^2"),K=6,NK=2,verbose=FALSE)$results_kfolds

bbb2 <- cv.plsRglm(Y~.,data=Cornell,nt=10,
modele="pls-glm-inverse.gaussian",keepcoeffs=TRUE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
PLS_lm(yCornell,XCornell,10,typeVC="standard")$InfCrit
rm(list=c("XCornell","yCornell","bbb","bbb2"))

data(Cornell)
bbb <- cv.plsRglm(Y~.,data=Cornell,nt=10,NK=1,modele="pls")
summary(bbb)

cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",family=gaussian(),K=12)


cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",family=gaussian(),K=6,
NK=2,random=TRUE,keepfolds=TRUE,verbose=FALSE)$results_kfolds

#Different ways of model specifications
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",family=gaussian(),K=6,
NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",family=gaussian,
K=6,NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",family=gaussian(),
K=6,NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",family=gaussian(link=log),
K=6,NK=2,verbose=FALSE)$results_kfolds

bbb2 <- cv.plsRglm(Y~.,data=Cornell,nt=10,
modele="pls-glm-gaussian",keepcoeffs=TRUE,verbose=FALSE)
bbb2 <- cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",
family=gaussian(link=log),K=6,keepcoeffs=TRUE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
PLS_lm_formula(Y~.,data=Cornell,10,typeVC="standard")$InfCrit
```

```
rm(list=c("bbb","bbb2"))


data(pine)
bbb <- cv.plsRglm(x11~.,data=pine,nt=10,modele="pls-glm-family",
family=gaussian(log),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb <- cv.plsRglm(x11~.,data=pine,nt=10,modele="pls-glm-family",family=gaussian(),
K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb)
boxplot(kfolds2coeff(bbb)[,1])

kfolds2Chisqind(bbb)
kfolds2Chisq(bbb)
summary(bbb)
PLS_lm_formula(x11~.,data=pine,nt=10,typeVC="standard")$InfCrit

data(pineNAX21)
bbb2 <- cv.plsRglm(x11~.,data=pineNAX21,nt=10,
modele="pls-glm-family",family=gaussian(log),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb2 <- cv.plsRglm(x11~.,data=pineNAX21,nt=10,
modele="pls-glm-gaussian",K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
PLS_lm_formula(x11~.,data=pineNAX21,nt=10,typeVC="standard")$InfCrit
rm(list=c("bbb","bbb2"))


data(aze_compl)
bbb <- cv.plsRglm(y~.,data=aze_compl,nt=10,K=10,modele="pls",
keepcoeffs=TRUE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb)
bbb2 <- cv.plsRglm(y~.,data=aze_compl,nt=3,K=10,
modele="pls-glm-family",family=binomial(probit),keepcoeffs=TRUE,verbose=FALSE)
bbb2 <- cv.plsRglm(y~.,data=aze_compl,nt=3,K=10,
modele="pls-glm-logistic",keepcoeffs=TRUE,verbose=FALSE)
summary(bbb,MClassed=TRUE)
summary(bbb2,MClassed=TRUE)
kfolds2coeff(bbb2)

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
rm(list=c("bbb","bbb2"))
```

```
data(pine)
bbb <- cv.plsRglm(round(x11)~.,data=pine,nt=10,
modele="pls-glm-family",family=poisson(log),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb <- cv.plsRglm(round(x11)~.,data=pine,nt=10,
modele="pls-glm-poisson",K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb)
boxplot(kfolds2coeff(bbb)[,1])

kfolds2Chisqind(bbb)
kfolds2Chisq(bbb)
summary(bbb)
PLS_lm_formula(x11~.,data=pine,10,typeVC="standard")$InfCrit

data(pineNAX21)
bbb2 <- cv.plsRglm(round(x11)~.,data=pineNAX21,nt=10,
modele="pls-glm-family",family=poisson(log),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb2 <- cv.plsRglm(round(x11)~.,data=pineNAX21,nt=10,
modele="pls-glm-poisson",K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
PLS_lm_formula(x11~.,data=pineNAX21,10,typeVC="standard")$InfCrit
rm(list=c("bbb","bbb2"))



data(pine)
bbb <- cv.plsRglm(x11~.,data=pine,nt=10,modele="pls-glm-family",
family=Gamma,K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb <- cv.plsRglm(x11~.,data=pine,nt=10,modele="pls-glm-Gamma",
K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb)
boxplot(kfolds2coeff(bbb)[,1])

kfolds2Chisqind(bbb)
kfolds2Chisq(bbb)
summary(bbb)
PLS_lm_formula(x11~.,data=pine,10,typeVC="standard")$InfCrit

data(pineNAX21)
bbb2 <- cv.plsRglm(x11~.,data=pineNAX21,nt=10,
modele="pls-glm-family",family=Gamma(),K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)
bbb2 <- cv.plsRglm(x11~.,data=pineNAX21,nt=10,
modele="pls-glm-Gamma",K=10,keepcoeffs=TRUE,keepfolds=FALSE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
PLS_lm_formula(x11~.,data=pineNAX21,10,typeVC="standard")$InfCrit
rm(list=c("bbb","bbb2"))



data(Cornell)
summary(cv.plsRglm(Y~.,data=Cornell,nt=10,NK=1,modele="pls",verbose=FALSE))

cv.plsRglm(Y~.,data=Cornell,nt=3,
modele="pls-glm-inverse.gaussian",K=12,verbose=FALSE)
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",family=inverse.gaussian,K=12,verbose=FALSE)
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-inverse.gaussian",K=6,
NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",
family=inverse.gaussian(),K=6,NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-inverse.gaussian",K=6,
NK=2,verbose=FALSE)$results_kfolds
cv.plsRglm(Y~.,data=Cornell,nt=3,modele="pls-glm-family",
family=inverse.gaussian(link = "1/mu^2"),K=6,NK=2,verbose=FALSE)$results_kfolds

bbb2 <- cv.plsRglm(Y~.,data=Cornell,nt=10,
modele="pls-glm-inverse.gaussian",keepcoeffs=TRUE,verbose=FALSE)

#For Jackknife computations
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])

kfolds2Chisqind(bbb2)
kfolds2Chisq(bbb2)
summary(bbb2)
PLS_lm_formula(Y~.,data=Cornell,10,typeVC="standard")$InfCrit
rm(list=c("bbb","bbb2"))


data(bordeaux)
summary(cv.plsRglm(Quality~.,data=bordeaux,10,
modele="pls-glm-polr",K=7))

data(bordeauxNA)
summary(cv.plsRglm(Quality~.,data=bordeauxNA,
10,modele="pls-glm-polr",K=10,verbose=FALSE))
```

```
summary(cv.plsRglm(Quality~.,data=bordeaux,nt=2,K=7,
modele="pls-glm-polr",method="logistic",verbose=FALSE))
summary(cv.plsRglm(Quality~.,data=bordeaux,nt=2,K=7,
modele="pls-glm-polr",method="probit",verbose=FALSE))
summary(cv.plsRglm(Quality~.,data=bordeaux,nt=2,K=7,
modele="pls-glm-polr",method="cloglog",verbose=FALSE))
suppressWarnings(summary(cv.plsRglm(Quality~.,data=bordeaux,nt=2,K=7,
modele="pls-glm-polr",method="cauchit",verbose=FALSE)))
```

---

cvtable *Table method for summary of cross validated PLSR and PLSGLR models*

Description

The function `cvtable` is wrapper of `cvtable.plsR` and `cvtable.plsRglm` that provides a table summary for the classes `"summary.cv.plsRmodel"` and `"summary.cv.plsRglmmodel"`

Usage

```
cvtable(x, verbose = TRUE, ...)
```

Arguments

`x`	an object of the class `"summary.cv.plsRmodel"`
`verbose`	should results be displayed ?
`...`	further arguments to be passed to or from methods.

Value

`list`List of Information Criteria computed for each fold.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`summary`

Examples

```
data(Cornell)
cv.modpls <- cv.plsR(Y~.,data=Cornell,nt=6,K=6,NK=5)
res.cv.modpls <- cvtable(summary(cv.modpls))
plot(res.cv.modpls) #defaults to type="CVQ2"
rm(list=c("cv.modpls","res.cv.modpls"))


data(Cornell)
cv.modpls <- cv.plsR(Y~.,data=Cornell,nt=6,K=6,NK=25,verbose=FALSE)
res.cv.modpls <- cvtable(summary(cv.modpls))
plot(res.cv.modpls) #defaults to type="CVQ2"
rm(list=c("cv.modpls","res.cv.modpls"))

data(Cornell)
cv.modpls <- cv.plsR(Y~.,data=Cornell,nt=6,K=6,NK=100,verbose=FALSE)
res.cv.modpls <- cvtable(summary(cv.modpls))
plot(res.cv.modpls) #defaults to type="CVQ2"
rm(list=c("cv.modpls","res.cv.modpls"))

data(Cornell)
cv.modplsglm <- cv.plsRglm(Y~.,data=Cornell,nt=6,K=6,
modele="pls-glm-gaussian",NK=100,verbose=FALSE)
res.cv.modplsglm <- cvtable(summary(cv.modplsglm))
plot(res.cv.modplsglm) #defaults to type="CVQ2Chi2"
rm(list=c("res.cv.modplsglm"))
```

---

dicho *Dichotomization*

Description

This function takes a real value and converts it to 1 if it is positive and else to 0.

Usage

```
dicho(val)
```

Arguments

`val`	A real value

Value

0 or 1.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

`ifelse`

Examples

```
dimX <- 6
Astar <- 4
(dataAstar4 <- t(replicate(10,simul_data_YX(dimX,Astar))))

dicho(dataAstar4)

rm(list=c("dimX","Astar"))
```

`fowlkes` *Fowlkes dataset*

Description

A classic dataset from Fowlkes.

Format

A data frame with 9949 observations on the following 13 variables.

Y binary response

MA a numeric vector

MW a numeric vector

NE a numeric vector

NW a numeric vector

PA a numeric vector

SO a numeric vector

SW a numeric vector

color a numeric vector

age1 a numeric vector

age2 a numeric vector

age3 a numeric vector

sexe a numeric vector

Examples

```
data(fowlkes)
str(fowlkes)
```

---

infcrit.dof *Information criteria*

Description

This function computes information criteria for existing plsR model using Degrees of Freedom estimation.

Usage

```
infcrit.dof(modplsR, naive = FALSE)
```

Arguments

`modplsR`	A plsR model i.e. an object returned by one of the functions `plsR`, `plsRmodel.default`, `plsRmodel.formula`, `PLS_lm` or `PLS_lm_formula`.
`naive`	A boolean.

Details

If `naive=FALSE` returns AIC, BIC and gmdl values for estimated and naive degrees of freedom. If `naive=TRUE` returns `NULL`.

Value

`matrix`	AIC, BIC and gmdl values or `NULL`.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

M. Hansen, B. Yu. (2001). Model Selection and Minimum Descripion Length Principle, *Journal of the American Statistical Association*, 96, 746-774.
N. Kraemer, M. Sugiyama. (2011). The Degrees of Freedom of Partial Least Squares Regression. *Journal of the American Statistical Association*, 106(494), 697-705.
N. Kraemer, M. Sugiyama, M.L. Braun. (2009). Lanczos Approximations for the Speedup of Kernel Partial Least Squares Regression, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 272-279.

See Also

plsR.dof for degrees of freedom computation and infcrit.dof for computing information criteria directly from a previously fitted plsR model.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsR(yCornell,XCornell,4)
infcrit.dof(modpls)
```

`kfolds2Chisq`	*Computes Predicted Chisquare for k-fold cross-validated partial least squares regression models.*

Description

This function computes Predicted Chisquare for k-fold cross validated partial least squares regression models.

Usage

```
kfolds2Chisq(pls_kfolds)
```

Arguments

`pls_kfolds`	a k-fold cross validated partial least squares regression glm model

Value

`list`	Total Predicted Chisquare vs number of components for the first group partition
`list()`	...
`list`	Total Predicted Chisquare vs number of components for the last group partition

Note

Use cv.plsRglm to create k-fold cross validated partial least squares regression glm models.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

kfolds2coeff, kfolds2Press, kfolds2Pressind, kfolds2Chisqind, kfolds2Mclassedind and kfolds2Mclassed to extract and transforms results from k-fold cross validation.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
bbb <- cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-gaussian",K=16,verbose=FALSE)
bbb2 <- cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-gaussian",K=5,verbose=FALSE)
kfolds2Chisq(bbb)
kfolds2Chisq(bbb2)
rm(list=c("XCornell","yCornell","bbb","bbb2"))


data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
bbb <- cv.plsRglm(dataY=ypine,dataX=Xpine,nt=4,modele="pls-glm-gaussian",verbose=FALSE)
bbb2 <- cv.plsRglm(dataY=ypine,dataX=Xpine,nt=10,modele="pls-glm-gaussian",K=10,verbose=FALSE)
kfolds2Chisq(bbb)
kfolds2Chisq(bbb2)

XpineNAX21 <- Xpine
XpineNAX21[1,2] <- NA
bbbNA <- cv.plsRglm(dataY=ypine,dataX=XpineNAX21,nt=10,modele="pls",K=10,verbose=FALSE)
kfolds2Press(bbbNA)
kfolds2Chisq(bbbNA)
bbbNA2 <- cv.plsRglm(dataY=ypine,dataX=XpineNAX21,nt=4,modele="pls-glm-gaussian",verbose=FALSE)
bbbNA3 <- cv.plsRglm(dataY=ypine,dataX=XpineNAX21,nt=10,modele="pls-glm-gaussian",K=10,
verbose=FALSE)
kfolds2Chisq(bbbNA2)
kfolds2Chisq(bbbNA3)
rm(list=c("Xpine","XpineNAX21","ypine","bbb","bbb2","bbbNA","bbbNA2","bbbNA3"))


data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y
kfolds2Chisq(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=4,modele="pls-glm-family",
family="binomial",verbose=FALSE))
kfolds2Chisq(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=4,modele="pls-glm-logistic",
verbose=FALSE))
```

```
kfolds2Chisq(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=10,modele="pls-glm-family",
family=binomial(),K=10,verbose=FALSE))
kfolds2Chisq(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=10,modele="pls-glm-logistic",
K=10,verbose=FALSE))
rm(list=c("Xaze_compl","yaze_compl"))
```

---

kfolds2Chisqind — *Computes individual Predicted Chisquare for k-fold cross validated partial least squares regression models.*

Description

This function computes individual Predicted Chisquare for k-fold cross validated partial least squares regression models.

Usage

```
kfolds2Chisqind(pls_kfolds)
```

Arguments

`pls_kfolds`	a k-fold cross validated partial least squares regression glm model

Value

`list`	Individual PChisq vs number of components for the first group partition
`list()`	...
`list`	Individual PChisq vs number of components for the last group partition

Note

Use `cv.plsRglm` to create k-fold cross validated partial least squares regression glm models.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

kfolds2coeff, kfolds2Press, kfolds2Pressind, kfolds2Chisq, kfolds2Mclassedind and kfolds2Mclassed to extract and transforms results from k-fold cross-validation.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
bbb <- cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-gaussian",K=16,verbose=FALSE)
bbb2 <- cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-gaussian",K=5,verbose=FALSE)
kfolds2Chisqind(bbb)
kfolds2Chisqind(bbb2)
rm(list=c("XCornell","yCornell","bbb","bbb2"))


data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
bbb <- cv.plsRglm(dataY=ypine,dataX=Xpine,nt=4,modele="pls-glm-gaussian",verbose=FALSE)
bbb2 <- cv.plsRglm(dataY=ypine,dataX=Xpine,nt=10,modele="pls-glm-gaussian",K=10,verbose=FALSE)
kfolds2Chisqind(bbb)
kfolds2Chisqind(bbb2)

XpineNAX21 <- Xpine
XpineNAX21[1,2] <- NA
bbbNA <- cv.plsRglm(dataY=ypine,dataX=XpineNAX21,nt=10,modele="pls",K=10,verbose=FALSE)
kfolds2Pressind(bbbNA)
kfolds2Chisqind(bbbNA)
bbbNA2 <- cv.plsRglm(dataY=ypine,dataX=XpineNAX21,nt=4,modele="pls-glm-gaussian",verbose=FALSE)
bbbNA3 <- cv.plsRglm(dataY=ypine,dataX=XpineNAX21,nt=10,modele="pls-glm-gaussian",
K=10,verbose=FALSE)
kfolds2Chisqind(bbbNA2)
kfolds2Chisqind(bbbNA3)
rm(list=c("Xpine","XpineNAX21","ypine","bbb","bbb2","bbbNA","bbbNA2","bbbNA3"))


data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y
kfolds2Chisqind(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=4,modele="pls-glm-family",
family=binomial(),verbose=FALSE))
kfolds2Chisqind(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=4,modele="pls-glm-logistic",
verbose=FALSE))
kfolds2Chisqind(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=10,modele="pls-glm-family",
family=binomial(),K=10,verbose=FALSE))
kfolds2Chisqind(cv.plsRglm(dataY=yaze_compl,dataX=Xaze_compl,nt=10,
modele="pls-glm-logistic",K=10,verbose=FALSE))
rm(list=c("Xaze_compl","yaze_compl"))
```

`kfolds2coeff` *Extracts coefficients from k-fold cross validated partial least squares regression models*

Description

This fonction extracts coefficients from k-fold cross validated partial least squares regression models

Usage

```
kfolds2coeff(pls_kfolds)
```

Arguments

`pls_kfolds`	an object that is a k-fold cross validated partial least squares regression models either lm or glm

Details

This fonctions works for plsR and plsRglm models.

Value

`coef.all`	matrix with the values of the coefficients for each leave one out step or `NULL` if another type of cross validation was used.

Note

Only for `NK=1` and leave one out CV

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`kfolds2Pressind`, `kfolds2Press`, `kfolds2Mclassedind`, `kfolds2Mclassed` and `summary` to extract and transform results from k-fold cross validation.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
bbb <- PLS_lm_kfoldcv(dataY=yCornell,dataX=XCornell,nt=3,K=nrow(XCornell),keepcoeffs=TRUE,
verbose=FALSE)
kfolds2coeff(bbb)
boxplot(kfolds2coeff(bbb)[,2])
rm(list=c("XCornell","yCornell","bbb"))

data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
bbb2 <- cv.plsR(dataY=ypine,dataX=Xpine,nt=4,K=nrow(Xpine),keepcoeffs=TRUE,verbose=FALSE)
kfolds2coeff(bbb2)
boxplot(kfolds2coeff(bbb2)[,1])
rm(list=c("Xpine","ypine","bbb2"))
```

`kfolds2CVinfos_glm` *Extracts and computes information criteria and fits statistics for k-fold cross validated partial least squares glm models*

Description

This function extracts and computes information criteria and fits statistics for k-fold cross validated partial least squares glm models for both formula or classic specifications of the model.

Usage

```
kfolds2CVinfos_glm(pls_kfolds, MClassed = FALSE, verbose = TRUE)
```

Arguments

`pls_kfolds`	an object computed using `cv.plsRglm`
`MClassed`	should number of miss classed be computed ?
`verbose`	should infos be displayed ?

Details

The Mclassed option should only set to `TRUE` if the response is binary.

Value

`list`	table of fit statistics for first group partition
`list()`	...
`list`	table of fit statistics for last group partition

Note

Use summary and cv.plsRglm instead.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

kfolds2coeff, kfolds2Pressind, kfolds2Press, kfolds2Mclassedind and kfolds2Mclassed to extract and transforms results from k-fold cross-validation.

Examples

```
data(Cornell)
summary(cv.plsRglm(Y~.,data=Cornell,
nt=6,K=12,NK=1,keepfolds=FALSE,keepdataY=TRUE,modele="pls",verbose=FALSE),MClassed=TRUE)


data(aze_compl)
summary(cv.plsR(y~.,data=aze_compl,nt=10,K=8,modele="pls",verbose=FALSE),
MClassed=TRUE,verbose=FALSE)
summary(cv.plsRglm(y~.,data=aze_compl,nt=10,K=8,modele="pls",verbose=FALSE),
MClassed=TRUE,verbose=FALSE)
summary(cv.plsRglm(y~.,data=aze_compl,nt=10,K=8,
modele="pls-glm-family",
family=gaussian(),verbose=FALSE),
MClassed=TRUE,verbose=FALSE)
summary(cv.plsRglm(y~.,data=aze_compl,nt=10,K=8,
modele="pls-glm-logistic",
verbose=FALSE),MClassed=TRUE,verbose=FALSE)
summary(cv.plsRglm(y~.,data=aze_compl,nt=10,K=8,
modele="pls-glm-family",
family=binomial(),verbose=FALSE),
MClassed=TRUE,verbose=FALSE)


if(require(chemometrics)){
data(hyptis)
hyptis
yhyptis <- factor(hyptis$Group,ordered=TRUE)
Xhyptis <- as.data.frame(hyptis[,c(1:6)])
options(contrasts = c("contr.treatment", "contr.poly"))
modpls2 <- plsRglm(yhyptis,Xhyptis,6,modele="pls-glm-polr")
modpls2$Coeffsmodel_vals
modpls2$InfCrit
modpls2$Coeffs
modpls2$std.coeffs

table(yhyptis,predict(modpls2$FinalModel,type="class"))

modpls3 <- PLS_glm(yhyptis[-c(1,2,3)],Xhyptis[-c(1,2,3),],3,modele="pls-glm-polr",
dataPredictY=Xhyptis[c(1,2,3),],verbose=FALSE)

summary(cv.plsRglm(factor(Group,ordered=TRUE)~.,data=hyptis[,-c(7,8)],nt=4,K=10,
random=TRUE,modele="pls-glm-polr",keepcoeffs=TRUE,verbose=FALSE),
MClassed=TRUE,verbose=FALSE)
}
```

kfolds2CVinfos_lm *Extracts and computes information criteria and fits statistics for k-fold cross validated partial least squares models*

Description

This function extracts and computes information criteria and fits statistics for k-fold cross validated partial least squares models for both formula or classic specifications of the model.

Usage

```
kfolds2CVinfos_lm(pls_kfolds, MClassed = FALSE, verbose = TRUE)
```

Arguments

`pls_kfolds`	an object computed using `PLS_lm_kfoldcv`
`MClassed`	should number of miss classed be computed
`verbose`	should infos be displayed ?

Details

The Mclassed option should only set to `TRUE` if the response is binary.

Value

`list`	table of fit statistics for first group partition
`list()`	...
`list`	table of fit statistics for last group partition

Note

Use summary and cv.plsR instead.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

kfolds2coeff, kfolds2Pressind, kfolds2Press, kfolds2Mclassedind and kfolds2Mclassed to extract and transforms results from k-fold cross-validation.

Examples

```
data(Cornell)
summary(cv.plsR(Y~.,data=Cornell,nt=10,K=6,verbose=FALSE))


data(pine)
summary(cv.plsR(x11~.,data=pine,nt=10,NK=3,verbose=FALSE),verbose=FALSE)
data(pineNAX21)
summary(cv.plsR(x11~.,data=pineNAX21,nt=10,NK=3,
verbose=FALSE),verbose=FALSE)


data(aze_compl)
summary(cv.plsR(y~.,data=aze_compl,nt=10,K=8,NK=3,
verbose=FALSE),MClassed=TRUE,verbose=FALSE)
```

---

kfolds2Mclassed — *Number of missclassified individuals for k-fold cross validated partial least squares regression models.*

Description

This function indicates the total number of missclassified individuals for k-fold cross validated partial least squares regression models.

Usage

```
kfolds2Mclassed(pls_kfolds)
```

Arguments

`pls_kfolds`	a k-fold cross validated partial least squares regression model used on binary data

Value

`list`	Total number of missclassified individuals vs number of components for the first group partition
`list()`	...
`list`	Total number of missclassified individuals vs number of components for the last group partition

Note

Use `cv.plsR` to create k-fold cross validated partial least squares regression models.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`kfolds2coeff`, `kfolds2Press`, `kfolds2Pressind` and `kfolds2Mclassedind` to extract and transforms results from k-fold cross validation.

Examples

```
data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y
kfolds2Mclassed(cv.plsR(dataY=yaze_compl,dataX=Xaze_compl,nt=10,K=8,NK=1,verbose=FALSE))
kfolds2Mclassed(cv.plsR(dataY=yaze_compl,dataX=Xaze_compl,nt=10,K=8,NK=2,verbose=FALSE))
rm(list=c("Xaze_compl","yaze_compl"))
```

`kfolds2Mclassedind` *Number of missclassified individuals per group for k-fold cross validated partial least squares regression models.*

Description

This function indicates the number of missclassified individuals per group for k-fold cross validated partial least squares regression models.

Usage

```
kfolds2Mclassedind(pls_kfolds)
```

Arguments

`pls_kfolds`	a k-fold cross validated partial least squares regression model used on binary data

Value

`list`	Number of missclassified individuals per group vs number of components for the first group partition
`list()`	...
`list`	Number of missclassified individuals per group vs number of components for the last group partition

Note

Use `cv.plsR` or `cv.plsRglm` to create k-fold cross validated partial least squares regression models or generalized linear ones.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`kfolds2coeff`, `kfolds2Press`, `kfolds2Pressind` and `kfolds2Mclassed` to extract and transforms results from k-fold cross-validation.

Examples

```
data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y
kfolds2Mclassedind(cv.plsR(dataY=yaze_compl,dataX=Xaze_compl,nt=10,K=8,NK=1,verbose=FALSE))
kfolds2Mclassedind(cv.plsR(dataY=yaze_compl,dataX=Xaze_compl,nt=10,K=8,NK=2,verbose=FALSE))
rm(list=c("Xaze_compl","yaze_compl"))
```

`kfolds2Press` *Computes PRESS for k-fold cross validated partial least squares regression models.*

Description

This function computes PRESS for k-fold cross validated partial least squares regression models.

Usage

```
kfolds2Press(pls_kfolds)
```

Arguments

`pls_kfolds`	a k-fold cross validated partial least squares regression model

Value

`list`	Press vs number of components for the first group partition
`list()`	...
`list`	Press vs number of components for the last group partition

Note

Use `cv.plsR` to create k-fold cross validated partial least squares regression models.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

kfolds2coeff, kfolds2Pressind, kfolds2Mclassedind and kfolds2Mclassed to extract and transforms results from k-fold cross validation.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
kfolds2Press(cv.plsR(dataY=yCornell,dataX=data.frame(scale(as.matrix(XCornell))[,]),
nt=6,K=12,NK=1,verbose=FALSE))
kfolds2Press(cv.plsR(dataY=yCornell,dataX=data.frame(scale(as.matrix(XCornell))[,]),
nt=6,K=6,NK=1,verbose=FALSE))
rm(list=c("XCornell","yCornell"))


data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
kfolds2Press(cv.plsR(dataY=ypine,dataX=Xpine,nt=10,NK=1,verbose=FALSE))
kfolds2Press(cv.plsR(dataY=ypine,dataX=Xpine,nt=10,NK=2,verbose=FALSE))

XpineNAX21 <- Xpine
XpineNAX21[1,2] <- NA
kfolds2Press(cv.plsR(dataY=ypine,dataX=XpineNAX21,nt=10,NK=1,verbose=FALSE))
kfolds2Press(cv.plsR(dataY=ypine,dataX=XpineNAX21,nt=10,NK=2,verbose=FALSE))
rm(list=c("Xpine","XpineNAX21","ypine"))
```

---

kfolds2Pressind *Computes individual PRESS for k-fold cross validated partial least squares regression models.*

Description

This function computes individual PRESS for k-fold cross validated partial least squares regression models.

Usage

```
kfolds2Pressind(pls_kfolds)
```

Arguments

`pls_kfolds`	a k-fold cross validated partial least squares regression model

Value

`list`	Individual Press vs number of components for the first group partition
`list()`	...
`list`	Individual Press vs number of components for the last group partition

Note

Use `cv.plsR` to create k-fold cross validated partial least squares regression models.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`kfolds2coeff`, `kfolds2Press`, `kfolds2Mclassedind` and `kfolds2Mclassed` to extract and transforms results from k-fold cross validation.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
kfolds2Pressind(cv.plsR(dataY=yCornell,dataX=data.frame(scale(as.matrix(XCornell))[,]),
nt=6,K=12,NK=1))
kfolds2Pressind(cv.plsR(dataY=yCornell,dataX=data.frame(scale(as.matrix(XCornell))[,]),
nt=6,K=6,NK=1))
rm(list=c("XCornell","yCornell"))


data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
kfolds2Pressind(cv.plsR(dataY=ypine,dataX=Xpine,nt=10,NK=1,verbose=FALSE))
kfolds2Pressind(cv.plsR(dataY=ypine,dataX=Xpine,nt=10,NK=2,verbose=FALSE))
```

```
XpineNAX21 <- Xpine
XpineNAX21[1,2] <- NA
kfolds2Pressind(cv.plsR(dataY=ypine,dataX=XpineNAX21,nt=10,NK=1,verbose=FALSE))
kfolds2Pressind(cv.plsR(dataY=ypine,dataX=XpineNAX21,nt=10,NK=2,verbose=FALSE))
rm(list=c("Xpine","XpineNAX21","ypine"))
```

---

loglikpls *loglikelihood function for plsR models*

Description

This function provides loglikelihood computation for an univariate plsR model.

Usage

```
loglikpls(residpls, weights = rep.int(1, length(residpls)))
```

Arguments

`residpls`	Residuals of a fitted univariate plsR model
`weights`	Weights of observations

Details

Loglikelihood functions for plsR models with univariate response.

Value

`real`	Loglikelihood value

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Baibing Li, Julian Morris, Elaine B. Martin, Model selection for partial least squares regression, Chemometrics and Intelligent Laboratory Systems 64 (2002) 79-89, doi: 10.1016/S01697439(02)00051-5.

See Also

`AICpls` for AIC computation and `logLik` for loglikelihood computations for linear models

Examples

```
data(pine)
ypine <- pine[,11]
Xpine <- pine[,1:10]
(Pinscaled <- as.data.frame(cbind(scale(ypine),scale(as.matrix(Xpine)))))
colnames(Pinscaled)[1] <- "yy"

lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)

modpls <- plsR(ypine,Xpine,10)
modpls$Std.Coeffs
lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)

AIC(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled))
print(logLik(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)))

sum(dnorm(modpls$RepY, modpls$Std.ValsPredictY, sqrt(mean(modpls$residY^2)), log=TRUE))
sum(dnorm(Pinscaled$yy,fitted(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)),
sqrt(mean(residuals(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled))^2)), log=TRUE))
loglikpls(modpls$residY)
loglikpls(residuals(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)))
AICpls(10,residuals(lm(yy~x1+x2+x3+x4+x5+x6+x7+x8+x9+x10,data=Pinscaled)))
AICpls(10,modpls$residY)
```

---

permcoefs.plsR *Coefficients for permutation bootstrap computations of PLSR models*

Description

A function passed to `boot` to perform bootstrap.

Usage

```
permcoefs.plsR(dataset, ind, nt, modele, maxcoefvalues, ifbootfail, verbose)
```

Arguments

`dataset`	dataset to resample
`ind`	indices for resampling
`nt`	number of components to use
`modele`	type of modele to use, see plsR
`maxcoefvalues`	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
`ifbootfail`	value to return if the estimation fails on a bootstrap sample
`verbose`	should info messages be displayed ?

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

See also `bootpls`.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

# Lazraq-Cleroux PLS (Y,X) bootstrap
# statistic=permcoefs.plsR is the default for (Y,X) permutation resampling of PLSR models.
set.seed(250)
modpls <- plsR(yCornell,XCornell,1)
Cornell.bootYX <- bootpls(modpls, sim="permutation", R=250, statistic=permcoefs.plsR, verbose=FALSE)
```

`permcoefs.plsRglm` *Coefficients for permutation bootstrap computations of PLSGLR models*

Description

A function passed to `boot` to perform bootstrap.

Usage

```
permcoefs.plsRglm(
  dataset,
  ind,
  nt,
  modele,
  family = NULL,
  maxcoefvalues,
  ifbootfail,
  verbose
)
```

Arguments

`dataset`	dataset to resample
`ind`	indices for resampling
`nt`	number of components to use
`modele`	type of modele to use, see plsRglm
`family`	glm family to use, see plsRglm
`maxcoefvalues`	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
`ifbootfail`	value to return if the estimation fails on a bootstrap sample
`verbose`	should info messages be displayed ?

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

See also `bootplsglm`.

Examples

```
data(Cornell)

# (Y,X) bootstrap of a PLSGLR model
# statistic=coefs.plsRglm is the default for (Y,X) bootstrap of a PLSGLR models.
set.seed(250)
modplsglm <- plsRglm(Y~.,data=Cornell,1,modele="pls-glm-family",family=gaussian)
Cornell.bootYX <- bootplsglm(modplsglm, R=250, typeboot="plsmodel",
sim="permutation", statistic=permcoefs.plsRglm, verbose=FALSE)
```

`permcoefs.plsRglmnp` *Coefficients for permutation bootstrap computations of PLSGLR models*

Description

A function passed to `boot` to perform bootstrap.

Usage

```
permcoefs.plsRglmnp(
  dataRepYtt,
  ind,
  nt,
  modele,
  family = NULL,
  maxcoefvalues,
  wwetoile,
  ifbootfail
)
```

Arguments

`dataRepYtt`	components' coordinates to bootstrap
`ind`	indices for resampling
`nt`	number of components to use
`modele`	type of modele to use, see plsRglm
`family`	glm family to use, see plsRglm
`maxcoefvalues`	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
`wwetoile`	values of the Wstar matrix in the original fit
`ifbootfail`	value to return if the estimation fails on a bootstrap sample

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Note

~~some notes~~

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

See also `bootplsglm`

Examples

```
data(Cornell)

# (Y,X) bootstrap of a PLSGLR model
# statistic=coefs.plsRglm is the default for (Y,X) bootstrap of a PLSGLR models.
set.seed(250)
modplsglm <- plsRglm(Y~.,data=Cornell,1,modele="pls-glm-family",family=gaussian)
Cornell.bootYT <- bootplsglm(modplsglm, R=250, statistic=permcoefs.plsRglmnp, verbose=FALSE)
```

`permcoefs.plsRnp` *Coefficients computation for permutation bootstrap*

Description

A function passed to `boot` to perform bootstrap.

Usage

```
permcoefs.plsRnp(
  dataRepYtt,
  ind,
  nt,
  modele,
  maxcoefvalues,
  wwetoile,
  ifbootfail
)
```

Arguments

`dataRepYtt`	components' coordinates to bootstrap
`ind`	indices for resampling
`nt`	number of components to use
`modele`	type of modele to use, see plsRglm
`maxcoefvalues`	maximum values allowed for the estimates of the coefficients to discard those coming from singular bootstrap samples
`wwetoile`	values of the Wstar matrix in the original fit
`ifbootfail`	value to return if the estimation fails on a bootstrap sample

Value

estimates on a bootstrap sample or `ifbootfail` value if the bootstrap computation fails.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

See Also

See also bootpls

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

# Lazraq-Cleroux PLS (Y,X) bootstrap
# statistic=coefs.plsR is the default for (Y,X) resampling of PLSR models.
set.seed(250)
modpls <- plsR(yCornell,XCornell,1)
Cornell.bootYT <- bootpls(modpls, R=250, typeboot="fmodel_np", sim="permutation",
statistic=permcoefs.plsRnp, verbose=FALSE)
```

---

pine *Pine dataset*

Description

The caterpillar dataset was extracted from a 1973 study on pine processionary caterpillars. It assesses the influence of some forest settlement characteristics on the development of caterpillar colonies. The response variable is the logarithmic transform of the average number of nests of caterpillars per tree in an area of 500 square meters (x11). There are k=10 potentially explanatory variables defined on n=33 areas.

Format

A data frame with 33 observations on the following 11 variables.

x1 altitude (in meters)

x2 slope (en degrees)

x3 number of pines in the area

x4 height (in meters) of the tree sampled at the center of the area

x5 diameter (in meters) of the tree sampled at the center of the area

x6 index of the settlement density

x7 orientation of the area (from 1 if southbound to 2 otherwise)

x8 height (in meters) of the dominant tree

x9 number of vegetation strata

x10 mix settlement index (from 1 if not mixed to 2 if mixed)

x11 logarithmic transform of the average number of nests of caterpillars per tree

Details

These caterpillars got their names from their habit of moving over the ground in incredibly long head-to-tail processions when leaving their nest to create a new colony.

The `pine_sup` dataset can be used as a test set to assess model prediction error of a model trained on the `pine` dataset.

Source

Tomassone R., Audrain S., Lesquoy-de Turckeim E., Millier C. (1992), "La régression, nouveaux regards sur une ancienne méthode statistique", INRA, *Actualités Scientifiques et Agronomiques*, Masson, Paris.

References

J.-M. Marin, C. Robert. (2007). *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. Springer, New-York, pages 48-49.

Examples

```
data(pine)
str(pine)
```

`pineNAX21` *Incomplete dataset from the pine caterpillars example*

Description

The caterpillar dataset was extracted from a 1973 study on pine processionary caterpillars. It assesses the influence of some forest settlement characteristics on the development of caterpillar colonies. There are k=10 potentially explanatory variables defined on n=33 areas.
The value of x2 for the first observation was removed from the matrix of predictors on purpose.

Format

A data frame with 33 observations on the following 11 variables and one missing value.

x1 altitude (in meters)

x2 slope (en degrees)

x3 number of pines in the area

x4 height (in meters) of the tree sampled at the center of the area

x5 diameter (in meters) of the tree sampled at the center of the area

x6 index of the settlement density

x7 orientation of the area (from 1 if southbound to 2 otherwise)

x8 height (in meters) of the dominant tree

x9 number of vegetation strata

x10 mix settlement index (from 1 if not mixed to 2 if mixed)

x11 logarithmic transform of the average number of nests of caterpillars per tree

Details

These caterpillars got their names from their habit of moving over the ground in incredibly long head-to-tail processions when leaving their nest to create a new colony.
The `pineNAX21` is a dataset with a missing value for testing purpose.

Source

Tomassone R., Audrain S., Lesquoy-de Turckeim E., Millier C. (1992). "La régression, nouveaux regards sur une ancienne méthode statistique", INRA, *Actualités Scientifiques et Agronomiques*, Masson, Paris.

Examples

```
data(pineNAX21)
str(pineNAX21)
```

`pine_full` *Complete Pine dataset*

Description

This is the complete caterpillar dataset from a 1973 study on pine_full processionary caterpillars. It assesses the influence of some forest settlement characteristics on the development of caterpillar colonies. The response variable is the logarithmic transform of the average number of nests of caterpillars per tree in an area of 500 square meters (`x11`). There are k=10 potentially explanatory variables defined on n=55 areas.

Format

A data frame with 55 observations on the following 11 variables.

x1 altitude (in meters)

x2 slope (en degrees)

x3 number of pine_fulls in the area

x4 height (in meters) of the tree sampled at the center of the area

x5 diameter (in meters) of the tree sampled at the center of the area

x6 index of the settlement density

x7 orientation of the area (from 1 if southbound to 2 otherwise)

x8 height (in meters) of the dominant tree

x9 number of vegetation strata

x10 mix settlement index (from 1 if not mixed to 2 if mixed)

x11 logarithmic transform of the average number of nests of caterpillars per tree

Details

These caterpillars got their names from their habit of moving over the ground in incredibly long head-to-tail processions when leaving their nest to create a new colony.

Source

Tomassone R., Audrain S., Lesquoy-de Turckeim E., Millier C. (1992), "La régression, nouveaux regards sur une ancienne méthode statistique", INRA, *Actualités Scientifiques et Agronomiques*, Masson, Paris.

References

J.-M. Marin, C. Robert. (2007). *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. Springer, New-York, pages 48-49.

Examples

```
data(pine_full)
str(pine_full)
```

`pine_sup` *Complete Pine dataset*

Description

This is a supplementary dataset (used as a test set for the `pine` dataset) that was extracted from a 1973 study on pine_sup processionary caterpillars. It assesses the influence of some forest settlement characteristics on the development of caterpillar colonies. The response variable is the logarithmic transform of the average number of nests of caterpillars per tree in an area of 500 square meters (`x11`). There are k=10 potentially explanatory variables defined on n=22 areas.

Format

A data frame with 22 observations on the following 11 variables.

x1 altitude (in meters)

x2 slope (en degrees)

x3 number of pine_sups in the area

x4 height (in meters) of the tree sampled at the center of the area

x5 diameter (in meters) of the tree sampled at the center of the area

x6 index of the settlement density

x7 orientation of the area (from 1 if southbound to 2 otherwise)

x8 height (in meters) of the dominant tree

x9 number of vegetation strata

x10 mix settlement index (from 1 if not mixed to 2 if mixed)

x11 logarithmic transform of the average number of nests of caterpillars per tree

Details

These caterpillars got their names from their habit of moving over the ground in incredibly long head-to-tail processions when leaving their nest to create a new colony.

The `pine_sup` dataset can be used as a test set to assess model prediction error of a model trained on the `pine` dataset.

Source

Tomassone R., Audrain S., Lesquoy-de Turckeim E., Millier C. (1992), “La régression, nouveaux regards sur une ancienne méthode statistique”, INRA, *Actualités Scientifiques et Agronomiques*, Masson, Paris.

References

J.-M. Marin, C. Robert. (2007). *Bayesian Core: A Practical Approach to Computational Bayesian Statistics*. Springer, New-York, pages 48-49.

Examples

```
data(pine_sup)
str(pine_sup)
```

---

plot.table.summary.cv.plsRglmmodel

Plot method for table of summary of cross validated plsRglm models

---

Description

This function provides a table method for the class "summary.cv.plsRglmmodel"

Usage

```
## S3 method for class 'table.summary.cv.plsRglmmodel'
plot(x, type = c("CVMC", "CVQ2Chi2", "CVPreChi2"), ...)
```

Arguments

x	an object of the class "table.summary.cv.plsRglmmodel"
type	the type of cross validation criterion to plot.
...	further arguments to be passed to or from methods.

Value

NULL

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

summary

Examples

```
data(Cornell)
bbb <- cv.plsRglm(Y~.,data=Cornell,nt=10,NK=1,
modele="pls-glm-family",family=gaussian(), verbose=FALSE)
plot(cvtable(summary(bbb,verbose=FALSE)),type="CVQ2Chi2")
rm(list=c("bbb"))


data(Cornell)
plot(cvtable(summary(cv.plsRglm(Y~.,data=Cornell,nt=10,NK=100,
modele="pls-glm-family",family=gaussian(), verbose=FALSE),
verbose=FALSE)),type="CVQ2Chi2")
```

---

plot.table.summary.cv.plsRmodel

Plot method for table of summary of cross validated plsR models

---

Description

This function provides a table method for the class "summary.cv.plsRmodel"

Usage

```
## S3 method for class 'table.summary.cv.plsRmodel'
plot(x, type = c("CVMC", "CVQ2", "CVPress"), ...)
```

Arguments

x	an object of the class "table.summary.cv.plsRmodel"
type	the type of cross validation criterion to plot.
...	further arguments to be passed to or from methods.

Value

NULL

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

summary

Examples

```
data(Cornell)
bbb <- cv.plsR(Y~.,data=Cornell,nt=6,K=6,NK=5, verbose=FALSE)
plot(cvtable(summary(bbb)),type="CVQ2")
rm(list=c("bbb"))


data(Cornell)
plot(cvtable(summary(cv.plsR(Y~.,data=Cornell,nt=6,K=6,NK=100, verbose=FALSE))),type="CVQ2")
```

---

plots.confints.bootpls
Plot bootstrap confidence intervals

Description

This function plots the confidence intervals derived using the function confints.bootpls from from a bootpls based object.

Usage

```
plots.confints.bootpls(
  ic_bootobject,
  indices = NULL,
  legendpos = "topleft",
  prednames = TRUE,
  articlestyle = TRUE,
  xaxisticks = TRUE,
  ltyIC = c(2, 4, 5, 1),
  colIC = c("darkgreen", "blue", "red", "black"),
  typeIC,
  las = par("las"),
  mar,
  mgp,
    ...
  )
```

Arguments

`ic_bootobject`	an object created with the `confints.bootpls` function.
`indices`	vector of indices of the variables to plot. Defaults to `NULL`: all the predictors will be used.
`legendpos`	position of the legend as in `legend`, defaults to `"topleft"`
`prednames`	do the original names of the predictors shall be plotted ? Defaults to `TRUE`: the names are plotted.
`articlestyle`	do the extra blank zones of the margin shall be removed from the plot ? Defaults to `TRUE`: the margins are removed.
`xaxisticks`	do ticks for the x axis shall be plotted ? Defaults to `TRUE`: the ticks are plotted.
`ltyIC`	lty as in `plot`
`colIC`	col as in `plot`
`typeIC`	type of CI to plot. Defaults to `typeIC=c("Normal","Basic","Percentile","BCa")` if BCa intervals limits were computed and to `typeIC=c("Normal","Basic","Percentile")` otherwise.
`las`	numeric in 0,1,2,3; the style of axis labels. 0: always parallel to the axis [default], 1: always horizontal, 2: always perpendicular to the axis, 3: always vertical.
`mar`	A numerical vector of the form `c(bottom,left,top,right)` which gives the number of lines of margin to be specified on the four sides of the plot. The default is `c(5,4,4,2) + 0.1.`
`mgp`	The margin line (in mex units) for the axis title, axis labels and axis line. Note that `mgp[1]` affects title whereas `mgp[2:3]` affect axis. The default is `c(3,1,0)`.
`...`	further options to pass to the `plot` function.

Value

`NULL`

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

`confints.bootpls`

Examples

```
data(Cornell)
modpls <- plsR(Y~.,data=Cornell,3)

# Lazraq-Cleroux PLS (Y,X) bootstrap
set.seed(250)
Cornell.bootYX <- bootpls(modpls, R=250, verbose=FALSE)
temp.ci <- confints.bootpls(Cornell.bootYX,2:8)

plots.confints.bootpls(temp.ci)
plots.confints.bootpls(temp.ci,prednames=FALSE)
plots.confints.bootpls(temp.ci,prednames=FALSE,articlestyle=FALSE,
main="Bootstrap confidence intervals for the bj")
plots.confints.bootpls(temp.ci,indices=1:3,prednames=FALSE)
plots.confints.bootpls(temp.ci,c(2,4,6),"bottomright")
plots.confints.bootpls(temp.ci,c(2,4,6),articlestyle=FALSE,
main="Bootstrap confidence intervals for some of the bj")

temp.ci <- confints.bootpls(Cornell.bootYX,typeBCa=FALSE)
plots.confints.bootpls(temp.ci)
plots.confints.bootpls(temp.ci,2:8)
plots.confints.bootpls(temp.ci,prednames=FALSE)


# Bastien CSDA 2005 (Y,T) bootstrap
Cornell.boot <- bootpls(modpls, typeboot="fmodel_np", R=250, verbose=FALSE)
temp.ci <- confints.bootpls(Cornell.boot,2:8)

plots.confints.bootpls(temp.ci)
plots.confints.bootpls(temp.ci,prednames=FALSE)
plots.confints.bootpls(temp.ci,prednames=FALSE,articlestyle=FALSE,
main="Bootstrap confidence intervals for the bj")
plots.confints.bootpls(temp.ci,indices=1:3,prednames=FALSE)
plots.confints.bootpls(temp.ci,c(2,4,6),"bottomright")
plots.confints.bootpls(temp.ci,c(2,4,6),articlestyle=FALSE,
main="Bootstrap confidence intervals for some of the bj")

temp.ci <- confints.bootpls(Cornell.boot,typeBCa=FALSE)
plots.confints.bootpls(temp.ci)
plots.confints.bootpls(temp.ci,2:8)
plots.confints.bootpls(temp.ci,prednames=FALSE)



data(aze_compl)
modplsglm <- plsRglm(y~.,data=aze_compl,3,modele="pls-glm-logistic")

# Lazraq-Cleroux PLS (Y,X) bootstrap
# should be run with R=1000 but takes much longer time
aze_compl.bootYX3 <- bootplsglm(modplsglm, typeboot="plsmodel", R=250, verbose=FALSE)
temp.ci <- confints.bootpls(aze_compl.bootYX3)
```

```
plots.confints.bootpls(temp.ci)
plots.confints.bootpls(temp.ci,prednames=FALSE)
plots.confints.bootpls(temp.ci,prednames=FALSE,articlestyle=FALSE,
main="Bootstrap confidence intervals for the bj")
plots.confints.bootpls(temp.ci,indices=1:33,prednames=FALSE)
plots.confints.bootpls(temp.ci,c(2,4,6),"bottomleft")
plots.confints.bootpls(temp.ci,c(2,4,6),articlestyle=FALSE,
main="Bootstrap confidence intervals for some of the bj")
plots.confints.bootpls(temp.ci,indices=1:34,prednames=FALSE)
plots.confints.bootpls(temp.ci,indices=1:33,prednames=FALSE,ltyIC=1,colIC=c(1,2))

temp.ci <- confints.bootpls(aze_compl.bootYX3,1:34,typeBCa=FALSE)
plots.confints.bootpls(temp.ci,indices=1:33,prednames=FALSE)


# Bastien CSDA 2005 (Y,T) Bootstrap
# much faster
aze_compl.bootYT3 <- bootplsglm(modplsglm, R=1000, verbose=FALSE)
temp.ci <- confints.bootpls(aze_compl.bootYT3)

plots.confints.bootpls(temp.ci)
plots.confints.bootpls(temp.ci,typeIC="Normal")
plots.confints.bootpls(temp.ci,typeIC=c("Normal","Basic"))
plots.confints.bootpls(temp.ci,typeIC="BCa",legendpos="bottomleft")
plots.confints.bootpls(temp.ci,prednames=FALSE)
plots.confints.bootpls(temp.ci,prednames=FALSE,articlestyle=FALSE,
main="Bootstrap confidence intervals for the bj")
plots.confints.bootpls(temp.ci,indices=1:33,prednames=FALSE)
plots.confints.bootpls(temp.ci,c(2,4,6),"bottomleft")
plots.confints.bootpls(temp.ci,c(2,4,6),articlestyle=FALSE,
main="Bootstrap confidence intervals for some of the bj")
plots.confints.bootpls(temp.ci,prednames=FALSE,ltyIC=c(2,1),colIC=c(1,2))

temp.ci <- confints.bootpls(aze_compl.bootYT3,1:33,typeBCa=FALSE)
plots.confints.bootpls(temp.ci,prednames=FALSE)
```

plsR *Partial least squares Regression models with leave one out cross validation*

Description

This function implements Partial least squares Regression models with leave one out cross validation for complete or incomplete datasets.

Usage

```
plsR(x, ...)
## Default S3 method:
plsRmodel(dataY, dataX, nt = 2, limQ2set = 0.0975,
dataPredictY = dataX, modele = "pls", family = NULL, typeVC = "none",
EstimXNA = FALSE, scaleX = TRUE, scaleY = NULL, pvals.expli = FALSE,
alpha.pvals.expli = 0.05, MClassed = FALSE, tol_Xi = 10^(-12), weights,
sparse = FALSE, sparseStop = TRUE, naive = FALSE,verbose=TRUE)
## S3 method for class 'formula'
plsRmodel(formula, data, nt = 2, limQ2set = 0.0975,
dataPredictY, modele = "pls", family = NULL, typeVC = "none",
EstimXNA = FALSE, scaleX = TRUE, scaleY = NULL, pvals.expli = FALSE,
alpha.pvals.expli = 0.05, MClassed = FALSE, tol_Xi = 10^(-12), weights,
subset, contrasts = NULL, sparse = FALSE, sparseStop = TRUE, naive = FALSE,
verbose=TRUE)
PLS_lm(dataY, dataX, nt = 2, limQ2set = 0.0975, dataPredictY = dataX,
modele = "pls", family = NULL, typeVC = "none", EstimXNA = FALSE,
scaleX = TRUE, scaleY = NULL, pvals.expli = FALSE,
alpha.pvals.expli = 0.05, MClassed = FALSE, tol_Xi = 10^(-12),
weights,sparse=FALSE,sparseStop=FALSE,naive=FALSE,verbose=TRUE)
PLS_lm_formula(formula,data=NULL,nt=2,limQ2set=.0975,dataPredictY=dataX,
modele="pls",family=NULL,typeVC="none",EstimXNA=FALSE,scaleX=TRUE,
scaleY=NULL,pvals.expli=FALSE,alpha.pvals.expli=.05,MClassed=FALSE,
tol_Xi=10^(-12),weights,subset,contrasts=NULL,sparse=FALSE,
sparseStop=FALSE,naive=FALSE,verbose=TRUE)
```

Arguments

`x`	a formula or a response (training) dataset
`dataY`	response (training) dataset
`dataX`	predictor(s) (training) dataset
`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `plsR` is called.
`nt`	number of components to be extracted
`limQ2set`	limit value for the Q2
`dataPredictY`	predictor(s) (testing) dataset
`modele`	name of the PLS model to be fitted, only (`"pls"` available for this fonction.
`family`	for the present moment the family argument is ignored and set thanks to the value of modele.

`typeVC`	type of leave one out cross validation. Several procedures are available. If cross validation is required, one needs to selects the way of predicting the response for left out observations. For complete rows, without any missing value, there are two different ways of computing these predictions. As a consequence, for mixed datasets, with complete and incomplete rows, there are two ways of computing prediction : either predicts any row as if there were missing values in it (`missingdata`) or selects the prediction method accordingly to the completeness of the row (`adaptative`). `none` no cross validation `standard` as in SIMCA for datasets without any missing value. For datasets with any missing value, it is the as using `missingdata` `missingdata` all values predicted as those with missing values for datasets with any missing values `adaptative` predict a response value for an x with any missing value as those with missing values and for an x without any missing value as those without missing values.
`EstimXNA`	only for `modele="pls"`. Set whether the missing X values have to be estimated.
`scaleX`	scale the predictor(s) : must be set to TRUE for `modele="pls"` and should be for glms pls.
`scaleY`	scale the response : Yes/No. Ignored since non always possible for glm responses.
`pvals.expli`	should individual p-values be reported to tune model selection ?
`alpha.pvals.expli`	level of significance for predictors when pvals.expli=TRUE
`MClassed`	number of missclassified cases, should only be used for binary responses
`tol_Xi`	minimal value for Norm2(Xi) and $\det(pp' \times pp)$ if there is any missing value in the `dataX`. It defaults to 10^{-12}
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`sparse`	should the coefficients of non-significant predictors (<`alpha.pvals.expli`) be set to 0
`sparseStop`	should component extraction stop when no significant predictors (<`alpha.pvals.expli`) are found
`naive`	Use the naive estimates for the Degrees of Freedom in plsR? Default is `FALSE`.
`verbose`	should info messages be displayed ?
`...`	arguments to pass to `plsRmodel.default` or to `plsRmodel.formula`

Details

There are several ways to deal with missing values that leads to different computations of leave one out cross validation criteria.

A typical predictor has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with any duplicates removed.

A specification of the form first:second indicates the the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second.

The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula.

Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations.

The default estimator for Degrees of Freedom is the Kramer and Sugiyama's one. Information criteria are computed accordingly to these estimations. Naive Degrees of Freedom and Information Criteria are also provided for comparison purposes. For more details, see N. Kraemer and M. Sugiyama. (2011). The Degrees of Freedom of Partial Least Squares Regression. *Journal of the American Statistical Association*, 106(494), 697-705, 2011.

Value

`nr`	Number of observations
`nc`	Number of predictors
`nt`	Number of requested components
`ww`	raw weights (before L2-normalization)
`wwnorm`	L2 normed weights (to be used with deflated matrices of predictor variables)
`wwetoile`	modified weights (to be used with original matrix of predictor variables)
`tt`	PLS components
`pp`	loadings of the predictor variables
`CoeffC`	coefficients of the PLS components
`uscores`	scores of the response variable
`YChapeau`	predicted response values for the dataX set
`residYChapeau`	residuals of the deflated response on the standardized scale
`RepY`	scaled response vector
`na.miss.Y`	is there any NA value in the response vector
`YNA`	indicatrix vector of missing values in RepY
`residY`	deflated scaled response vector
`ExpliX`	scaled matrix of predictors

`na.miss.X`	is there any NA value in the predictor matrix
`XXNA`	indicator of non-NA values in the predictor matrix
`residXX`	deflated predictor matrix
`PredictY`	response values with NA replaced with 0
`press.ind`	individual PRESS value for each observation (scaled scale)
`press.tot`	total PRESS value for all observations (scaled scale)
`family`	glm family used to fit PLSGLR model
`ttPredictY`	PLS components for the dataset on which prediction was requested
`typeVC`	type of leave one out cross-validation used
`dataX`	predictor values
`dataY`	response values
`computed_nt`	number of components that were computed
`CoeffCFull`	matrix of the coefficients of the predictors
`CoeffConstante`	value of the intercept (scaled scale)
`Std.Coeffs`	Vector of standardized regression coefficients
`press.ind2`	individual PRESS value for each observation (original scale)
`RSSresidY`	residual sum of squares (scaled scale)
`Coeffs`	Vector of regression coefficients (used with the original data scale)
`Yresidus`	residuals of the PLS model
`RSS`	residual sum of squares (original scale)
`residusY`	residuals of the deflated response on the standardized scale
`AIC.std`	AIC.std vs number of components (AIC computed for the standardized model
`AIC`	AIC vs number of components
`optional`	If the response is assumed to be binary: i.e. `MClassed=TRUE`. `MissClassed` Number of miss classed results; `Probs` "Probability" predicted by the model. These are not true probabilities since they may lay outside of [0,1]; `Probs.trc` Probability predicted by the model and constrained to belong to [0,1]
`ttPredictFittedMissingY`	Description of 'comp2'
`optional`	If cross validation was requested: i.e. `typeVC="standard"`, `typeVC="missingdata"` or `typeVC="adaptative"`. `R2residY` R2 coefficient value on the standardized scale; `R2` R2 coefficient value on the original scale; `press.tot2` total PRESS value for all observations (original scale); `Q2` Q2 value (standardized scale); `limQ2` limit of the Q2 value

	`Q2_2` Q2 value (original scale)
	`Q2cum` cumulated Q2 (standardized scale)
	`Q2cum_2` cumulated Q2 (original scale)
`InfCrit`	table of Information Criteria
`Std.ValsPredictY`	predicted response values for supplementary dataset (standardized scale)
`ValsPredictY`	predicted response values for supplementary dataset (original scale)
`Std.XChapeau`	estimated values for missing values in the predictor matrix (standardized scale)
`XXwotNA`	predictor matrix with missing values replaced with 0

Note

Use `cv.plsR` to cross-validate the plsRglm models and `bootpls` to bootstrap them.

Author(s)

Frederic Bertrand
`<frederic.bertrand@math.unistra.fr>`
`http://www-irma.u-strasbg.fr/~fbertran/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frederic Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

See also `plsRglm` to fit PLSGLR models.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

#maximum 6 components could be extracted from this dataset
#trying 10 to trigger automatic stopping criterion
modpls10<-plsR(yCornell,XCornell,10)
modpls10

#With iterated leave one out CV PRESS
modpls6cv<-plsR(Y~.,data=Cornell,6,typeVC="standard")
modpls6cv
cv.modpls<-cv.plsR(Y~.,data=Cornell,6,NK=100, verbose=FALSE)
res.cv.modpls<-cvtable(summary(cv.modpls))
plot(res.cv.modpls)
```

```
rm(list=c("XCornell","yCornell","modpls10","modpls6cv"))


#A binary response example
data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y
modpls.aze <- plsR(yaze_compl,Xaze_compl,10,MClassed=TRUE,typeVC="standard")
modpls.aze

#Direct access to not cross-validated values
modpls.aze$AIC
modpls.aze$AIC.std
modpls.aze$MissClassed

#Raw predicted values (not really probabily since not constrained in [0,1]
modpls.aze$Probs
#Truncated to [0;1] predicted values (true probabilities)
modpls.aze$Probs.trc
modpls.aze$Probs-modpls.aze$Probs.trc

#Repeated cross validation of the model (NK=100 times)
cv.modpls.aze<-cv.plsR(y~.,data=aze_compl,10,NK=100, verbose=FALSE)
res.cv.modpls.aze<-cvtable(summary(cv.modpls.aze,MClassed=TRUE))
#High discrepancy in the number of component choice using repeated cross validation
#and missclassed criterion
plot(res.cv.modpls.aze)

rm(list=c("Xaze_compl","yaze_compl","modpls.aze","cv.modpls.aze","res.cv.modpls.aze"))

#24 predictors
dimX <- 24
#2 components
Astar <- 2
simul_data_UniYX(dimX,Astar)
dataAstar2 <- data.frame(t(replicate(250,simul_data_UniYX(dimX,Astar))))
modpls.A2<- plsR(Y~.,data=dataAstar2,10,typeVC="standard")
modpls.A2
cv.modpls.A2<-cv.plsR(Y~.,data=dataAstar2,10,NK=100, verbose=FALSE)
res.cv.modpls.A2<-cvtable(summary(cv.modpls.A2,verbose=FALSE))
#Perfect choice for the Q2 criterion in PLSR
plot(res.cv.modpls.A2)

#Binarized data.frame
simbin1 <- data.frame(dicho(dataAstar2))
modpls.B2 <- plsR(Y~.,data=simbin1,10,typeVC="standard",MClassed=TRUE, verbose=FALSE)
modpls.B2
modpls.B2$Probs
modpls.B2$Probs.trc
modpls.B2$MissClassed
plsR(simbin1$Y,dataAstar2[,-1],10,typeVC="standard",MClassed=TRUE,verbose=FALSE)$InfCrit
cv.modpls.B2<-cv.plsR(Y~.,data=simbin1,2,NK=100,verbose=FALSE)
res.cv.modpls.B2<-cvtable(summary(cv.modpls.B2,MClassed=TRUE))
```

```
#Only one component found by repeated CV missclassed criterion
plot(res.cv.modpls.B2)

rm(list=c("dimX","Astar","dataAstar2","modpls.A2","cv.modpls.A2",
"res.cv.modpls.A2","simbin1","modpls.B2","cv.modpls.B2","res.cv.modpls.B2"))
```

`plsR.dof` *Computation of the Degrees of Freedom*

Description

This function computes the Degrees of Freedom using the Krylov representation of PLS and other quantities that are used to get information criteria values. For the time present, it only works with complete datasets.

Usage

```
## S3 method for class 'dof'
plsR(modplsR, naive = FALSE)
```

Arguments

`modplsR`	A plsR model i.e. an object returned by one of the functions `plsR`, `plsRmodel.default`, `plsRmodel.formula`, `PLS_lm` or `PLS_lm_formula`.
`naive`	A boolean.

Details

If `naive=FALSE` returns values for estimated degrees of freedom and error dispersion. If `naive=TRUE` returns returns values for naive degrees of freedom and error dispersion. The original code from Nicole Kraemer and Mikio L. Braun was unable to handle models with only one component.

Value

`DoF`	Degrees of Freedom
`sigmahat`	Estimates of dispersion
`Yhat`	Predicted values
`yhat`	Square Euclidean norms of the predicted values
`RSS`	Residual Sums of Squares

Author(s)

Nicole Kraemer, Mikio L. Braun with improvements from Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
https://fbertran.github.io/homepage/

References

N. Kraemer, M. Sugiyama. (2011). The Degrees of Freedom of Partial Least Squares Regression. *Journal of the American Statistical Association*, 106(494), 697-705.
N. Kraemer, M. Sugiyama, M.L. Braun. (2009). Lanczos Approximations for the Speedup of Kernel Partial Least Squares Regression, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 272-279.

See Also

`aic.dof` and `infcrit.dof` for computing information criteria directly from a previously fitted plsR model.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsR(yCornell,XCornell,4)
plsR.dof(modpls)
plsR.dof(modpls,naive=TRUE)
```

`plsRglm` *Partial least squares Regression generalized linear models*

Description

This function implements Partial least squares Regression generalized linear models complete or incomplete datasets.

Usage

```
plsRglm(x, ...)
## Default S3 method:
plsRglmmodel(dataY,dataX,nt=2,limQ2set=.0975,
dataPredictY=dataX,modele="pls",family=NULL,typeVC="none",
EstimXNA=FALSE,scaleX=TRUE,scaleY=NULL,pvals.expli=FALSE,
alpha.pvals.expli=.05,MClassed=FALSE,tol_Xi=10^(-12),weights,
sparse=FALSE,sparseStop=TRUE,naive=FALSE,verbose=TRUE)
## S3 method for class 'formula'
plsRglmmodel(formula,data=NULL,nt=2,limQ2set=.0975,
dataPredictY,modele="pls",family=NULL,typeVC="none",
EstimXNA=FALSE,scaleX=TRUE,scaleY=NULL,pvals.expli=FALSE,
alpha.pvals.expli=.05,MClassed=FALSE,tol_Xi=10^(-12),weights,subset,
start=NULL,etastart,mustart,offset,method="glm.fit",control= list(),
contrasts=NULL,sparse=FALSE,sparseStop=TRUE,naive=FALSE,verbose=TRUE)
```

```
PLS_glm(dataY, dataX, nt = 2, limQ2set = 0.0975, dataPredictY = dataX,
modele = "pls", family = NULL, typeVC = "none", EstimXNA = FALSE,
scaleX = TRUE, scaleY = NULL, pvals.expli = FALSE,
alpha.pvals.expli = 0.05, MClassed = FALSE, tol_Xi = 10^(-12), weights,
method, sparse = FALSE, sparseStop=FALSE, naive=FALSE,verbose=TRUE)
PLS_glm_formula(formula,data=NULL,nt=2,limQ2set=.0975,dataPredictY=dataX,
modele="pls",family=NULL,typeVC="none",EstimXNA=FALSE,scaleX=TRUE,
scaleY=NULL,pvals.expli=FALSE,alpha.pvals.expli=.05,MClassed=FALSE,
tol_Xi=10^(-12),weights,subset,start=NULL,etastart,mustart,offset,method,
control= list(),contrasts=NULL,sparse=FALSE,sparseStop=FALSE,naive=FALSE,verbose=TRUE)
```

Arguments

`x`	a formula or a response (training) dataset
`dataY`	response (training) dataset
`dataX`	predictor(s) (training) dataset
`formula`	an object of class "`formula`" (or one that can be coerced to that class): a symbolic description of the model to be fitted. The details of model specification are given under 'Details'.
`data`	an optional data frame, list or environment (or object coercible by `as.data.frame` to a data frame) containing the variables in the model. If not found in `data`, the variables are taken from `environment(formula)`, typically the environment from which `plsRglm` is called.
`nt`	number of components to be extracted
`limQ2set`	limit value for the Q2
`dataPredictY`	predictor(s) (testing) dataset
`modele`	name of the PLS glm model to be fitted (`"pls"`, `"pls-glm-Gamma"`, `"pls-glm-gaussian"`, `"pls-glm-inverse.gaussian"`, `"pls-glm-logistic"`, `"pls-glm-poisson"`, `"pls-glm-polr"`). Use `"modele=pls-glm-family"` to enable the `family` option.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.) To use the family option, please set `modele="pls-glm-family"`. User defined families can also be defined. See details.
`typeVC`	type of leave one out cross validation. For back compatibility purpose. `none` no cross validation
`EstimXNA`	only for `modele="pls"`. Set whether the missing X values have to be estimated.
`scaleX`	scale the predictor(s) : must be set to TRUE for `modele="pls"` and should be for glms pls.
`scaleY`	scale the response : Yes/No. Ignored since non always possible for glm responses.
`pvals.expli`	should individual p-values be reported to tune model selection ?

`alpha.pvals.expli`	level of significance for predictors when pvals.expli=TRUE
`MClassed`	number of missclassified cases, should only be used for binary responses
`tol_Xi`	minimal value for Norm2(Xi) and $\det(pp' \times pp)$ if there is any missing value in the `dataX`. It defaults to 10^{-12}
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`subset`	an optional vector specifying a subset of observations to be used in the fitting process.
`start`	starting values for the parameters in the linear predictor.
`etastart`	starting values for the linear predictor.
`mustart`	starting values for the vector of means.
`offset`	this can be used to specify an *a priori* known component to be included in the linear predictor during fitting. This should be `NULL` or a numeric vector of length equal to the number of cases. One or more `offset` terms can be included in the formula instead or as well, and if more than one is specified their sum is used. See `model.offset`.
`method`	For a glm model (`modele="pls-glm-family"`), the method to be used in fitting the model. The default method `"glm.fit"` uses iteratively reweighted least squares (IWLS). User-supplied fitting functions can be supplied either as a function or a character string naming a function, with a function which takes the same arguments as `glm.fit`. For a polr model (`modele="pls-glm-polr"`), `logistic` or `probit` or (complementary) log-log (`loglog` or `cloglog`) or `cauchit` (corresponding to a Cauchy latent variable).
`control`	a list of parameters for controlling the fitting process. For `glm.fit` this is passed to `glm.control`.
`contrasts`	an optional list. See the `contrasts.arg` of `model.matrix.default`.
`sparse`	should the coefficients of non-significant predictors (<`alpha.pvals.expli`) be set to 0
`sparseStop`	should component extraction stop when no significant predictors (<`alpha.pvals.expli`) are found
`naive`	Use the naive estimates for the Degrees of Freedom in plsR? Default is `FALSE`.
`verbose`	Should details be displayed ?
`...`	arguments to pass to `plsRmodel.default` or to `plsRmodel.formula`

Details

There are seven different predefined models with predefined link functions available :

`"pls"` ordinary pls models

`"pls-glm-Gamma"` glm gaussian with inverse link pls models

`"pls-glm-gaussian"` glm gaussian with identity link pls models

`"pls-glm-inverse-gamma"` glm binomial with square inverse link pls models `"pls-glm-logistic"` glm binomial with logit link pls models

`"pls-glm-poisson"` glm poisson with log link pls models

`"pls-glm-polr"` glm polr with logit link pls models

Using the `"family="` option and setting `"modele=pls-glm-family"` allows changing the family and link function the same way as for the `glm` function. As a consequence user-specified families can also be used.

The `gaussian` **family** accepts the links (as names) `identity`, `log` and `inverse`.

The `binomial` **family** accepts the links `logit`, `probit`, `cauchit`, (corresponding to logistic, normal and Cauchy CDFs respectively) `log` and `cloglog` (complementary log-log).

The `Gamma` **family** accepts the links `inverse`, `identity` and `log`.

The `poisson` **family** accepts the links `log`, `identity`, and `sqrt`.

The `inverse.gaussian` **family** accepts the links `1/mu^2`, `inverse`, `identity` and `log`.

The `quasi` **family** accepts the links `logit`, `probit`, `cloglog`, `identity`, `inverse`, `log`, `1/mu^2` and `sqrt`.

The function `power` can be used to create a power link function.

A typical predictor has the form response ~ terms where response is the (numeric) response vector and terms is a series of terms which specifies a linear predictor for response. A terms specification of the form first + second indicates all the terms in first together with all the terms in second with any duplicates removed.

A specification of the form first:second indicates the the set of terms obtained by taking the interactions of all terms in first with all terms in second. The specification first*second indicates the cross of first and second. This is the same as first + second + first:second.

The terms in the formula will be re-ordered so that main effects come first, followed by the interactions, all second-order, all third-order and so on: to avoid this pass a terms object as the formula.

Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations.

The default estimator for Degrees of Freedom is the Kramer and Sugiyama's one which only works for classical plsR models. For these models, Information criteria are computed accordingly to these estimations. Naive Degrees of Freedom and Information Criteria are also provided for comparison purposes. For more details, see N. Kraemer and M. Sugiyama. (2011). The Degrees of Freedom of Partial Least Squares Regression. *Journal of the American Statistical Association*, 106(494), 697-705, 2011.

Value

Depends on the model that was used to fit the model. You can generally at least find these items.

`nr`	Number of observations
`nc`	Number of predictors
`nt`	Number of requested components

`ww`	raw weights (before L2-normalization)
`wwnorm`	L2 normed weights (to be used with deflated matrices of predictor variables)
`wwetoile`	modified weights (to be used with original matrix of predictor variables)
`tt`	PLS components
`pp`	loadings of the predictor variables
`CoeffC`	coefficients of the PLS components
`uscores`	scores of the response variable
`YChapeau`	predicted response values for the dataX set
`residYChapeau`	residuals of the deflated response on the standardized scale
`RepY`	scaled response vector
`na.miss.Y`	is there any NA value in the response vector
`YNA`	indicatrix vector of missing values in RepY
`residY`	deflated scaled response vector
`ExpliX`	scaled matrix of predictors
`na.miss.X`	is there any NA value in the predictor matrix
`XXNA`	indicator of non-NA values in the predictor matrix
`residXX`	deflated predictor matrix
`PredictY`	response values with NA replaced with 0
`RSS`	residual sum of squares (original scale)
`RSSresidY`	residual sum of squares (scaled scale)
`R2residY`	R2 coefficient value on the standardized scale
`R2`	R2 coefficient value on the original scale
`press.ind`	individual PRESS value for each observation (scaled scale)
`press.tot`	total PRESS value for all observations (scaled scale)
`Q2cum`	cumulated Q2 (standardized scale)
`family`	glm family used to fit PLSGLR model
`ttPredictY`	PLS components for the dataset on which prediction was requested
`typeVC`	type of leave one out cross-validation used
`dataX`	predictor values
`dataY`	response values
`weights`	weights of the observations
`computed_nt`	number of components that were computed
`AIC`	AIC vs number of components
`BIC`	BIC vs number of components
`Coeffsmodel_vals`	
`ChisqPearson`	

`CoeffCFull`	matrix of the coefficients of the predictors
`CoeffConstante`	value of the intercept (scaled scale)
`Std.Coeffs`	Vector of standardized regression coefficients
`Coeffs`	Vector of regression coefficients (used with the original data scale)
`Yresidus`	residuals of the PLS model
`residusY`	residuals of the deflated response on the standardized scale
`InfCrit`	table of Information Criteria:

`AIC`	AIC vs number of components
`BIC`	BIC vs number of components
`MissClassed`	Number of miss classed results
`Chi2_Pearson_Y`	Q2 value (standardized scale)
`RSS`	residual sum of squares (original scale)
`R2`	R2 coefficient value on the original scale
`R2residY`	R2 coefficient value on the standardized scale
`RSSresidY`	residual sum of squares (scaled scale)

`Std.ValsPredictY`	predicted response values for supplementary dataset (standardized scale)
`ValsPredictY`	predicted response values for supplementary dataset (original scale)
`Std.XChapeau`	estimated values for missing values in the predictor matrix (standardized scale)
`FinalModel`	final GLR model on the PLS components
`XXwotNA`	predictor matrix with missing values replaced with 0
`call`	call
`AIC.std`	AIC.std vs number of components (AIC computed for the standardized model

Note

Use `cv.plsRglm` to cross-validate the plsRglm models and `bootplsglm` to bootstrap them.

Author(s)

Frederic Bertrand
`<frederic.bertrand@math.unistra.fr>`
`http://www-irma.u-strasbg.fr/~fbertran/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frederic Bertrand (2010). Comparaison de la regression PLS et de la regression logistique PLS : application aux donnees d'allelotypage. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

See also `plsR`.

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

modplsglm <- plsRglm(yCornell,XCornell,10,modele="pls-glm-gaussian")

#To retrieve the final GLR model on the PLS components
finalmod <- modplsglm$FinalModel
#It is a glm object.
plot(finalmod)


#Cross validation
cv.modplsglm<-cv.plsRglm(Y~.,data=Cornell,6,NK=100,modele="pls-glm-gaussian", verbose=FALSE)
res.cv.modplsglm<-cvtable(summary(cv.modplsglm))
plot(res.cv.modplsglm)

#If no model specified, classic PLSR model
modpls <- plsRglm(Y~.,data=Cornell,6)
modpls
modpls$tt
modpls$uscores
modpls$pp
modpls$Coeffs

#rm(list=c("XCornell","yCornell",modpls,cv.modplsglm,res.cv.modplsglm))


data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y
plsRglm(yaze_compl,Xaze_compl,nt=10,modele="pls",MClassed=TRUE, verbose=FALSE)$InfCrit
modpls <- plsRglm(yaze_compl,Xaze_compl,nt=10,modele="pls-glm-logistic",
MClassed=TRUE,pvals.expli=TRUE, verbose=FALSE)
modpls
colSums(modpls$pvalstep)
modpls$Coeffsmodel_vals

plot(plsRglm(yaze_compl,Xaze_compl,4,modele="pls-glm-logistic")$FinalModel)
plsRglm(yaze_compl[-c(99,72)],Xaze_compl[-c(99,72),],4,
modele="pls-glm-logistic",pvals.expli=TRUE)$pvalstep
plot(plsRglm(yaze_compl[-c(99,72)],Xaze_compl[-c(99,72),],4,
modele="pls-glm-logistic",pvals.expli=TRUE)$FinalModel)
rm(list=c("Xaze_compl","yaze_compl","modpls"))


data(bordeaux)
Xbordeaux<-bordeaux[,1:4]
ybordeaux<-factor(bordeaux$Quality,ordered=TRUE)
modpls <- plsRglm(ybordeaux,Xbordeaux,10,modele="pls-glm-polr",pvals.expli=TRUE)
modpls
```

```
colSums(modpls$pvalstep)


XbordeauxNA<-Xbordeaux
XbordeauxNA[1,1] <- NA
modplsNA <- plsRglm(ybordeaux,XbordeauxNA,10,modele="pls-glm-polr",pvals.expli=TRUE)
modpls
colSums(modpls$pvalstep)
rm(list=c("Xbordeaux","XbordeauxNA","ybordeaux","modplsNA"))


data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
modpls1 <- plsRglm(ypine,Xpine,1)
modpls1$Std.Coeffs
modpls1$Coeffs
modpls4 <- plsRglm(ypine,Xpine,4)
modpls4$Std.Coeffs
modpls4$Coeffs
modpls4$PredictY[1,]
plsRglm(ypine,Xpine,4,dataPredictY=Xpine[1,])$PredictY[1,]

XpineNAX21 <- Xpine
XpineNAX21[1,2] <- NA
modpls4NA <- plsRglm(ypine,XpineNAX21,4)
modpls4NA$Std.Coeffs
modpls4NA$YChapeau[1,]
modpls4$YChapeau[1,]
modpls4NA$CoeffC
plsRglm(ypine,XpineNAX21,4,EstimXNA=TRUE)$XChapeau
plsRglm(ypine,XpineNAX21,4,EstimXNA=TRUE)$XChapeauNA

# compare pls-glm-gaussian with classic plsR
modplsglm4 <- plsRglm(ypine,Xpine,4,modele="pls-glm-gaussian")
cbind(modpls4$Std.Coeffs,modplsglm4$Std.Coeffs)

# without missing data
cbind(ypine,modpls4$ValsPredictY,modplsglm4$ValsPredictY)

# with missing data
modplsglm4NA <- plsRglm(ypine,XpineNAX21,4,modele="pls-glm-gaussian")
cbind((ypine),modpls4NA$ValsPredictY,modplsglm4NA$ValsPredictY)
rm(list=c("Xpine","ypine","modpls4","modpls4NA","modplsglm4","modplsglm4NA"))

data(fowlkes)
Xfowlkes <- fowlkes[,2:13]
yfowlkes <- fowlkes[,1]
modpls <- plsRglm(yfowlkes,Xfowlkes,4,modele="pls-glm-logistic",pvals.expli=TRUE)
modpls
colSums(modpls$pvalstep)
rm(list=c("Xfowlkes","yfowlkes","modpls"))
```

```
if(require(chemometrics)){
data(hyptis)
yhyptis <- factor(hyptis$Group,ordered=TRUE)
Xhyptis <- as.data.frame(hyptis[,c(1:6)])
options(contrasts = c("contr.treatment", "contr.poly"))
modpls2 <- plsRglm(yhyptis,Xhyptis,6,modele="pls-glm-polr")
modpls2$Coeffsmodel_vals
modpls2$InfCrit
modpls2$Coeffs
modpls2$Std.Coeffs

table(yhyptis,predict(modpls2$FinalModel,type="class"))
rm(list=c("yhyptis","Xhyptis","modpls2"))
}

dimX <- 24
Astar <- 6
dataAstar6 <- t(replicate(250,simul_data_UniYX(dimX,Astar)))
ysimbin1 <- dicho(dataAstar6)[,1]
Xsimbin1 <- dicho(dataAstar6)[,2:(dimX+1)]
modplsglm <- plsRglm(ysimbin1,Xsimbin1,10,modele="pls-glm-logistic")
modplsglm

simbin=data.frame(dicho(dataAstar6))
cv.modplsglm <- suppressWarnings(cv.plsRglm(Y~.,data=simbin,nt=10,
modele="pls-glm-logistic",NK=100, verbose=FALSE))
res.cv.modplsglm <- cvtable(summary(cv.modplsglm,MClassed=TRUE,
verbose=FALSE))
plot(res.cv.modplsglm) #defaults to type="CVMC"

rm(list=c("dimX","Astar","dataAstar6","ysimbin1","Xsimbin1","modplsglm","cv.modplsglm",
"res.cv.modplsglm"))
```

PLS_glm_wvc — *Light version of PLS_glm for cross validation purposes*

Description

Light version of `PLS_glm` for cross validation purposes either on complete or incomplete datasets.

Usage

```
PLS_glm_wvc(
  dataY,
  dataX,
  nt = 2,
  dataPredictY = dataX,
  modele = "pls",
  family = NULL,
  scaleX = TRUE,
  scaleY = NULL,
  keepcoeffs = FALSE,
  keepstd.coeffs = FALSE,
  tol_Xi = 10^(-12),
  weights,
  method = "logistic",
  verbose = TRUE
)
```

Arguments

`dataY`	response (training) dataset
`dataX`	predictor(s) (training) dataset
`nt`	number of components to be extracted
`dataPredictY`	predictor(s) (testing) dataset
`modele`	name of the PLS glm model to be fitted (`"pls"`, `"pls-glm-Gamma"`, `"pls-glm-gaussian"`, `"pls-glm-inverse.gaussian"`, `"pls-glm-logistic"`, `"pls-glm-poisson"`, `"pls-glm-polr"`). Use `"modele=pls-glm-family"` to enable the `family` option.
`family`	a description of the error distribution and link function to be used in the model. This can be a character string naming a family function, a family function or the result of a call to a family function. (See `family` for details of family functions.) To use the family option, please set `modele="pls-glm-family"`. User defined families can also be defined. See details.
`scaleX`	scale the predictor(s) : must be set to TRUE for `modele="pls"` and should be for glms pls.
`scaleY`	scale the response : Yes/No. Ignored since non always possible for glm responses.
`keepcoeffs`	whether the coefficients of the linear fit on link scale of unstandardized eXplanatory variables should be returned or not.
`keepstd.coeffs`	whether the coefficients of the linear fit on link scale of standardized eXplanatory variables should be returned or not.
`tol_Xi`	minimal value for Norm2(Xi) and $\det(pp' \times pp)$ if there is any missing value in the `dataX`. It defaults to 10^{-12}
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`method`	logistic, probit, complementary log-log or cauchit (corresponding to a Cauchy latent variable).
`verbose`	should info messages be displayed ?

Details

This function is called by `PLS_glm_kfoldcv_formula` in order to perform cross-validation either on complete or incomplete datasets.

There are seven different predefined models with predefined link functions available :

list("\"pls\"") ordinary pls models

list("\"pls-glm-Gamma\"") glm gaussian with inverse link pls models

list("\"pls-glm-gaussian\"") glm gaussian with identity link pls models

list("\"pls-glm-inverse-gamma\"") glm binomial with square inverse link pls models

list("\"pls-glm-logistic\"") glm binomial with logit link pls models

list("\"pls-glm-poisson\"") glm poisson with log link pls models

list("\"pls-glm-polr\"") glm polr with logit link pls models

Using the `"family="` option and setting `"modele=pls-glm-family"` allows changing the family and link function the same way as for the `glm` function. As a consequence user-specified families can also be used.

The accepts the links (as names) `identity`, `log` and `inverse`.

list("gaussian") accepts the links (as names) `identity`, `log` and `inverse`.

family accepts the links (as names) `identity`, `log` and `inverse`.

The accepts the links `logit`, `probit`, `cauchit`, (corresponding to logistic, normal and Cauchy CDFs respectively) `log` and `cloglog` (complementary log-log).

list("binomial") accepts the links `logit`, `probit`, `cauchit`, (corresponding to logistic, normal and Cauchy CDFs respectively) `log` and `cloglog` (complementary log-log).

family accepts the links `logit`, `probit`, `cauchit`, (corresponding to logistic, normal and Cauchy CDFs respectively) `log` and `cloglog` (complementary log-log).

The accepts the links `inverse`, `identity` and `log`.

list("Gamma") accepts the links `inverse`, `identity` and `log`.

family accepts the links `inverse`, `identity` and `log`.

The accepts the links `log`, `identity`, and `sqrt`.

list("poisson") accepts the links `log`, `identity`, and `sqrt`.

family accepts the links `log`, `identity`, and `sqrt`.

The accepts the links `1/mu^2`, `inverse`, `identity` and `log`.

list("inverse.gaussian") accepts the links `1/mu^2`, `inverse`, `identity` and `log`.

family accepts the links `1/mu^2`, `inverse`, `identity` and `log`.

The accepts the links `logit`, `probit`, `cloglog`, `identity`, `inverse`, `log`, `1/mu^2` and `sqrt`.

list("quasi") accepts the links `logit`, `probit`, `cloglog`, `identity`, `inverse`, `log`, `1/mu^2` and `sqrt`.

family accepts the links `logit`, `probit`, `cloglog`, `identity`, `inverse`, `log`, `1/mu^2` and `sqrt`.

The function can be used to create a power link function.

list("power") can be used to create a power link function.

Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations.

Value

`valsPredict`	`nrow(dataPredictY) * nt` matrix of the predicted values
`list("coeffs")`	If the coefficients of the eXplanatory variables were requested: i.e. `keepcoeffs=TRUE`. `ncol(dataX) * 1` matrix of the coefficients of the the eXplanatory variables

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`PLS_glm` for more detailed results, `PLS_glm_kfoldcv` for cross-validating models and `PLS_lm_wvc` for the same function dedicated to plsR models

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
PLS_glm_wvc(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-gaussian",
dataPredictY=XCornell[1,])
PLS_glm_wvc(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-family",
family=gaussian(),dataPredictY=XCornell[1,], verbose=FALSE)
PLS_glm_wvc(dataY=yCornell[-1],dataX=XCornell[-1,],nt=3,modele="pls-glm-gaussian",
dataPredictY=XCornell[1,], verbose=FALSE)
PLS_glm_wvc(dataY=yCornell[-1],dataX=XCornell[-1,],nt=3,modele="pls-glm-family",
family=gaussian(),dataPredictY=XCornell[1,], verbose=FALSE)
rm("XCornell","yCornell")


## With an incomplete dataset (X[1,2] is NA)
data(pine)
ypine <- pine[,11]
data(XpineNAX21)
```

```
PLS_glm_wvc(dataY=ypine,dataX=XpineNAX21,nt=10,modele="pls-glm-gaussian")
rm("XpineNAX21","ypine")

data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
PLS_glm_wvc(ypine,Xpine,10,modele="pls", verbose=FALSE)
PLS_glm_wvc(ypine,Xpine,10,modele="pls-glm-Gamma", verbose=FALSE)
PLS_glm_wvc(ypine,Xpine,10,modele="pls-glm-family",family=Gamma(), verbose=FALSE)
PLS_glm_wvc(ypine,Xpine,10,modele="pls-glm-gaussian", verbose=FALSE)
PLS_glm_wvc(ypine,Xpine,10,modele="pls-glm-family",family=gaussian(log), verbose=FALSE)
PLS_glm_wvc(round(ypine),Xpine,10,modele="pls-glm-poisson", verbose=FALSE)
PLS_glm_wvc(round(ypine),Xpine,10,modele="pls-glm-family",family=poisson(log), verbose=FALSE)
rm(list=c("pine","ypine","Xpine"))


data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
PLS_glm_wvc(yCornell,XCornell,10,modele="pls-glm-inverse.gaussian", verbose=FALSE)
PLS_glm_wvc(yCornell,XCornell,10,modele="pls-glm-family",
family=inverse.gaussian(), verbose=FALSE)
rm(list=c("XCornell","yCornell"))


data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
PLS_glm_wvc(dataY=yCornell,dataX=XCornell,nt=3,modele="pls-glm-gaussian",
dataPredictY=XCornell[1,], verbose=FALSE)
PLS_glm_wvc(dataY=yCornell[-1],dataX=XCornell[-1,],nt=3,modele="pls-glm-gaussian",
dataPredictY=XCornell[1,], verbose=FALSE)
rm("XCornell","yCornell")

data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y
PLS_glm(yaze_compl,Xaze_compl,10,modele="pls-glm-logistic",typeVC="none", verbose=FALSE)$InfCrit
PLS_glm_wvc(yaze_compl,Xaze_compl,10,modele="pls-glm-logistic", keepcoeffs=TRUE, verbose=FALSE)
rm("Xaze_compl","yaze_compl")
```

`PLS_lm_wvc` *Light version of PLS_lm for cross validation purposes*

Description

Light version of `PLS_lm` for cross validation purposes either on complete or incomplete datasets.

Usage

```
PLS_lm_wvc(
  dataY,
  dataX,
  nt = 2,
  dataPredictY = dataX,
  modele = "pls",
  scaleX = TRUE,
  scaleY = NULL,
  keepcoeffs = FALSE,
  keepstd.coeffs = FALSE,
  tol_Xi = 10^(-12),
  weights,
  verbose = TRUE
)
```

Arguments

`dataY`	response (training) dataset
`dataX`	predictor(s) (training) dataset
`nt`	number of components to be extracted
`dataPredictY`	predictor(s) (testing) dataset
`modele`	name of the PLS model to be fitted, only (`"pls"` available for this fonction.
`scaleX`	scale the predictor(s) : must be set to TRUE for `modele="pls"` and should be for glms pls.
`scaleY`	scale the response : Yes/No. Ignored since non always possible for glm responses.
`keepcoeffs`	whether the coefficients of unstandardized eXplanatory variables should be returned or not.
`keepstd.coeffs`	whether the coefficients of standardized eXplanatory variables should be returned or not.
`tol_Xi`	minimal value for Norm2(Xi) and $\det(pp' \times pp)$ if there is any missing value in the `dataX`. It defaults to 10^{-12}
`weights`	an optional vector of 'prior weights' to be used in the fitting process. Should be `NULL` or a numeric vector.
`verbose`	should info messages be displayed ?

Details

This function is called by `PLS_lm_kfoldcv` in order to perform cross-validation either on complete or incomplete datasets.

Non-NULL weights can be used to indicate that different observations have different dispersions (with the values in weights being inversely proportional to the dispersions); or equivalently, when the elements of weights are positive integers w_i, that each response y_i is the mean of w_i unit-weight observations.

Value

`valsPredict`	`nrow(dataPredictY) * nt` matrix of the predicted values
`list("coeffs")`	If the coefficients of the eXplanatory variables were requested: i.e. `keepcoeffs=TRUE`. `ncol(dataX) * 1` matrix of the coefficients of the the eXplanatory variables

Note

Use `PLS_lm_kfoldcv` for a wrapper in view of cross-validation.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

`PLS_lm` for more detailed results, `PLS_lm_kfoldcv` for cross-validating models and `PLS_glm_wvc` for the same function dedicated to plsRglm models

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
PLS_lm_wvc(dataY=yCornell,dataX=XCornell,nt=3,dataPredictY=XCornell[1,])
PLS_lm_wvc(dataY=yCornell[-c(1,2)],dataX=XCornell[-c(1,2),],nt=3,dataPredictY=XCornell[c(1,2),],
verbose=FALSE)
PLS_lm_wvc(dataY=yCornell[-c(1,2)],dataX=XCornell[-c(1,2),],nt=3,dataPredictY=XCornell[c(1,2),],
keepcoeffs=TRUE, verbose=FALSE)
rm("XCornell","yCornell")

## With an incomplete dataset (X[1,2] is NA)
data(pine)
ypine <- pine[,11]
data(XpineNAX21)
PLS_lm_wvc(dataY=ypine[-1],dataX=XpineNAX21[-1,],nt=3, verbose=FALSE)
PLS_lm_wvc(dataY=ypine[-1],dataX=XpineNAX21[-1,],nt=3,dataPredictY=XpineNAX21[1,], verbose=FALSE)
PLS_lm_wvc(dataY=ypine[-2],dataX=XpineNAX21[-2,],nt=3,dataPredictY=XpineNAX21[2,], verbose=FALSE)
PLS_lm_wvc(dataY=ypine,dataX=XpineNAX21,nt=3, verbose=FALSE)
rm("ypine")
```

predict.plsRglmmodel *Print method for plsRglm models*

Description

This function provides a predict method for the class "plsRglmmodel"

Usage

```
## S3 method for class 'plsRglmmodel'
predict(
  object,
  newdata,
  comps = object$computed_nt,
  type = c("link", "response", "terms", "scores", "class", "probs"),
  se.fit = FALSE,
  weights,
  dispersion = NULL,
  methodNA = "adaptative",
  verbose = TRUE,
  ...
)
```

Arguments

object	An object of the class "plsRmodel".
newdata	An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
comps	A value with a single value of component to use for prediction.
type	Type of predicted value. Available choices are the glms ones ("link", "response", "terms"), the polr ones ("class", "probs") or the scores ("scores").
se.fit	If TRUE, pointwise standard errors are produced for the predictions using the Cox model.
weights	Vector of case weights. If weights is a vector of integers, then the estimated coefficients are equivalent to estimating the model from data with the individual cases replicated as many times as indicated by weights.
dispersion	the dispersion of the GLM fit to be assumed in computing the standard errors. If omitted, that returned by summary applied to the object is used.
methodNA	Selects the way of predicting the response or the scores of the new data. For complete rows, without any missing value, there are two different ways of computing the prediction. As a consequence, for mixed datasets, with complete and incomplete rows, there are two ways of computing prediction : either predicts any row as if there were missing values in it (missingdata) or selects the prediction method accordingly to the completeness of the row (adaptative).
verbose	should info messages be displayed ?
...	Arguments to be passed on to stats::glm and plsRglm::plsRglm.

Value

When type is "response", a matrix of predicted response values is returned.
When type is "scores", a score matrix is returned.

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

See Also `predict.glm`

Examples

```
data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
data(pine_sup)
Xpine_sup<-pine_sup[,1:10]
Xpine_supNA<-Xpine_sup
Xpine_supNA[1,1]<-NA

modpls=plsRglm(dataY=ypine,dataX=Xpine,nt=6,modele="pls-glm-family",family="gaussian",
verbose=FALSE)
modplsform=plsRglm(x11~.,data=pine,nt=6,modele="pls-glm-family",family="gaussian", verbose=FALSE)
modpls2=plsRglm(dataY=ypine,dataX=Xpine,nt=6,modele="pls-glm-family",
dataPredictY=Xpine_sup,family="gaussian", verbose=FALSE)
modpls2NA=plsRglm(dataY=ypine,dataX=Xpine,nt=6,modele="pls-glm-family",
dataPredictY=Xpine_supNA,family="gaussian", verbose=FALSE)

#Identical to predict(modpls,type="link") or modpls$Std.ValsPredictY
cbind(modpls$Std.ValsPredictY,modplsform$Std.ValsPredictY,
predict(modpls),predict(modplsform))

#Identical to predict(modpls,type="response") or modpls$ValsPredictY
cbind(modpls$ValsPredictY,modplsform$ValsPredictY,
predict(modpls,type="response"),predict(modplsform,type="response"))

#Identical to modpls$ttPredictY
predict(modpls,type="scores")
predict(modplsform,type="scores")
```

```
#Identical to modpls2$ValsPredictY
cbind(predict(modpls,newdata=Xpine_sup,type="response"),
predict(modplsform,newdata=Xpine_sup,type="response"))

#Select the number of components to use to derive the prediction
predict(modpls,newdata=Xpine_sup,type="response",comps=1)
predict(modpls,newdata=Xpine_sup,type="response",comps=3)
predict(modpls,newdata=Xpine_sup,type="response",comps=6)
try(predict(modpls,newdata=Xpine_sup,type="response",comps=8))

#Identical to modpls2$ttValsPredictY
predict(modpls,newdata=Xpine_sup,type="scores")

#Select the number of components in the scores matrix
predict(modpls,newdata=Xpine_sup,type="scores",comps=1)
predict(modpls,newdata=Xpine_sup,type="scores",comps=3)
predict(modpls,newdata=Xpine_sup,type="scores",comps=6)
try(predict(modpls,newdata=Xpine_sup,type="scores",comps=8))

#Identical to modpls2NA$ValsPredictY
predict(modpls,newdata=Xpine_supNA,type="response",methodNA="missingdata")

cbind(predict(modpls,newdata=Xpine_supNA,type="response"),
predict(modplsform,newdata=Xpine_supNA,type="response"))

predict(modpls,newdata=Xpine_supNA,type="response",comps=1)
predict(modpls,newdata=Xpine_supNA,type="response",comps=3)
predict(modpls,newdata=Xpine_supNA,type="response",comps=6)
try(predict(modpls,newdata=Xpine_supNA,type="response",comps=8))

#Identical to modpls2NA$ttPredictY
predict(modpls,newdata=Xpine_supNA,type="scores",methodNA="missingdata")
predict(modplsform,newdata=Xpine_supNA,type="scores",methodNA="missingdata")

predict(modpls,newdata=Xpine_supNA,type="scores")
predict(modplsform,newdata=Xpine_supNA,type="scores")
predict(modpls,newdata=Xpine_supNA,type="scores",comps=1)
predict(modpls,newdata=Xpine_supNA,type="scores",comps=3)
predict(modpls,newdata=Xpine_supNA,type="scores",comps=6)
try(predict(modpls,newdata=Xpine_supNA,type="scores",comps=8))
```

`predict.plsRmodel` *Print method for plsR models*

Description

This function provides a predict method for the class "`plsRmodel`"

Usage

```
## S3 method for class 'plsRmodel'
predict(
  object,
  newdata,
  comps = object$computed_nt,
  type = c("response", "scores"),
  weights,
  methodNA = "adaptative",
  verbose = TRUE,
  ...
)
```

Arguments

`object`	An object of the class `"plsRmodel"`.
`newdata`	An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
`comps`	A value with a single value of component to use for prediction.
`type`	Type of predicted value. Available choices are the response values (`"response"`) or the scores (`"scores"`).
`weights`	Vector of case weights. If `weights` is a vector of integers, then the estimated coefficients are equivalent to estimating the model from data with the individual `cases` replicated as many times as indicated by `weights`.
`methodNA`	Selects the way of predicting the response or the scores of the new data. For complete rows, without any missing value, there are two different ways of computing the prediction. As a consequence, for mixed datasets, with complete and incomplete rows, there are two ways of computing prediction : either predicts any row as if there were missing values in it (`missingdata`) or selects the prediction method accordingly to the completeness of the row (`adaptative`).
`verbose`	should info messages be displayed ?
`...`	Arguments to be passed on to `plsRglm::plsR`.

Value

When type is `"response"`, a matrix of predicted response values is returned.
When type is `"scores"`, a score matrix is returned.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

Examples

```
data(pine)
Xpine<-pine[,1:10]
ypine<-pine[,11]
data(pine_sup)
Xpine_sup<-pine_sup[,1:10]
Xpine_supNA<-Xpine_sup
Xpine_supNA[1,1]<-NA

modpls=plsR(dataY=ypine,dataX=Xpine,nt=6,modele="pls", verbose=FALSE)
modplsform=plsR(x11~.,data=pine,nt=6,modele="pls", verbose=FALSE)
modpls2=plsR(dataY=ypine,dataX=Xpine,nt=6,modele="pls",dataPredictY=Xpine_sup, verbose=FALSE)
modpls2NA=plsR(dataY=ypine,dataX=Xpine,nt=6,modele="pls",dataPredictY=Xpine_supNA, verbose=FALSE)

#Identical to predict(modpls,type="response") or modpls$ValsPredictY
cbind(predict(modpls),predict(modplsform))

#Identical to modpls$ttPredictY
predict(modpls,type="scores")
predict(modplsform,type="scores")


#Identical to modpls2$ValsPredictY
cbind(predict(modpls,newdata=Xpine_sup,type="response"),
predict(modplsform,newdata=Xpine_sup,type="response"))

#Select the number of components to use to derive the prediction
predict(modpls,newdata=Xpine_sup,type="response",comps=1)
predict(modpls,newdata=Xpine_sup,type="response",comps=3)
predict(modpls,newdata=Xpine_sup,type="response",comps=6)
try(predict(modpls,newdata=Xpine_sup,type="response",comps=8))

#Identical to modpls2$ttValsPredictY
predict(modpls,newdata=Xpine_sup,type="scores")

#Select the number of components in the scores matrix
predict(modpls,newdata=Xpine_sup,type="scores",comps=1)
predict(modpls,newdata=Xpine_sup,type="scores",comps=3)
predict(modpls,newdata=Xpine_sup,type="scores",comps=6)
try(predict(modpls,newdata=Xpine_sup,type="scores",comps=8))

#Identical to modpls2NA$ValsPredictY
predict(modpls,newdata=Xpine_supNA,type="response",methodNA="missingdata")
```

```
cbind(predict(modpls,newdata=Xpine_supNA,type="response"),
predict(modplsform,newdata=Xpine_supNA,type="response"))

predict(modpls,newdata=Xpine_supNA,type="response",comps=1)
predict(modpls,newdata=Xpine_supNA,type="response",comps=3)
predict(modpls,newdata=Xpine_supNA,type="response",comps=6)
try(predict(modpls,newdata=Xpine_supNA,type="response",comps=8))

#Identical to modpls2NA$ttPredictY
predict(modpls,newdata=Xpine_supNA,type="scores",methodNA="missingdata")
predict(modplsform,newdata=Xpine_supNA,type="scores",methodNA="missingdata")

predict(modpls,newdata=Xpine_supNA,type="scores")
predict(modplsform,newdata=Xpine_supNA,type="scores")
predict(modpls,newdata=Xpine_supNA,type="scores",comps=1)
predict(modpls,newdata=Xpine_supNA,type="scores",comps=3)
predict(modpls,newdata=Xpine_supNA,type="scores",comps=6)
try(predict(modpls,newdata=Xpine_supNA,type="scores",comps=8))
```

print.coef.plsRglmmodel

Print method for plsRglm models

Description

This function provides a print method for the class "`coef.plsRglmmodel`"

Usage

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

Arguments

`x`	an object of the class "`coef.plsRglmmodel`"
`...`	not used

Value

`NULL`

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`print`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modplsglm <- plsRglm(yCornell,XCornell,3,modele="pls-glm-family",family=gaussian())
class(modplsglm)
print(coef(modplsglm))
rm(list=c("XCornell","yCornell","modplsglm"))
```

`print.coef.plsRmodel` *Print method for plsR models*

Description

This function provides a print method for the class `"coef.plsRmodel"`

Usage

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

Arguments

`x`	an object of the class `"coef.plsRmodel"`
`...`	not used

Value

`NULL`

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

`print`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsRglm(yCornell,XCornell,3,modele="pls")
class(modpls)
print(coef(modpls))
rm(list=c("XCornell","yCornell","modpls"))
```

`print.cv.plsRglmmodel` *Print method for plsRglm models*

Description

This function provides a print method for the class `"cv.plsRglmmodel"`

Usage

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

Arguments

`x`	an object of the class `"cv.plsRglmmodel"`
`...`	not used

Value

`NULL`

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparaison de la régression PLS et de la régression logistique PLS : application aux données d'allélotypage. *Journal de la Société Française de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

`print`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
print(cv.plsRglm(dataY=yCornell,dataX=XCornell,nt=10,NK=1,
modele="pls-glm-family",family=gaussian(), verbose=FALSE))
rm(list=c("XCornell","yCornell","bbb"))
```

`print.cv.plsRmodel` *Print method for plsR models*

Description

This function provides a print method for the class `"cv.plsRmodel"`

Usage

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

Arguments

`x`	an object of the class `"cv.plsRmodel"`
`...`	not used

Value

`NULL`

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

`print`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
print(cv.plsR(dataY=yCornell,dataX=XCornell,nt=10,K=6, verbose=FALSE))
rm(list=c("XCornell","yCornell","bbb"))
```

`print.plsRglmmodel` *Print method for plsRglm models*

Description

This function provides a print method for the class `"plsRglmmodel"`

Usage

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

Arguments

`x`	an object of the class `"plsRglmmodel"`
`...`	not used

Value

`NULL`

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparaison de la régression PLS et de la régression logistique PLS : application aux données d'allélotypage. *Journal de la Société Française de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

print

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modplsglm <- plsRglm(yCornell,XCornell,3,modele="pls-glm-gaussian")
class(modplsglm)
print(modplsglm)
rm(list=c("XCornell","yCornell","modplsglm"))
```

---

print.plsRmodel *Print method for plsR models*

Description

This function provides a print method for the class "plsRmodel"

Usage

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

Arguments

x	an object of the class "plsRmodel"
...	not used

Value

NULL

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparaison de la régression PLS et de la régression logistique PLS : application aux données d'allélotypage. *Journal de la Société Française de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

print

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsRglm(yCornell,XCornell,3,modele="pls")
class(modpls)
print(modpls)
rm(list=c("XCornell","yCornell","modpls"))
```

---

print.summary.plsRglmmodel
Print method for summaries of plsRglm models

Description

This function provides a print method for the class "summary.plsRglmmodel"

Usage

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

Arguments

x	an object of the class "summary.plsRglmmodel"
...	not used

Value

language	call of the model

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparaison de la régression PLS et de la régression logistique PLS : application aux données d'allélotypage. *Journal de la Société Française de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

print and summary

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modplsglm <- plsRglm(yCornell,XCornell,3,modele="pls-glm-gaussian")
class(modplsglm)
print(summary(modplsglm))
rm(list=c("XCornell","yCornell","modplsglm"))
```

---

print.summary.plsRmodel *Print method for summaries of plsR models*

Description

This function provides a print method for the class "summary.plsRmodel"

Usage

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

Arguments

x	an object of the class "summary.plsRmodel"
...	not used

Value

language	call of the model

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparaison de la régression PLS et de la régression logistique PLS : application aux données d'allélotypage. *Journal de la Société Française de Statistique*, 151(2), pages 1-18. http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47

See Also

print and summary

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsRglm(yCornell,XCornell,3,modele="pls")
class(modpls)
print(summary(modpls))
rm(list=c("XCornell","yCornell","modpls"))
```

---

signpred *Graphical assessment of the stability of selected variables*

Description

This fonctions plots, for each of the model, the

Usage

```
signpred(
  matbin,
  pred.lablength = max(sapply(rownames(matbin), nchar)),
  labsize = 1,
  plotsize = 12
)
```

Arguments

matbin	Matrix with 0 or 1 entries. Each row per predictor and a column for every model. 0 means the predictor is not significant in the model and 1 that, on the contrary, it is significant.
pred.lablength	Maximum length of the predictors labels. Defaults to full label length.
labsize	Size of the predictors labels.
plotsize	Global size of the graph.

Details

This function is based on the visweb function from the bipartite package.

Value

A plot window.

Author(s)

Bernd Gruber with minor modifications from Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

Vazquez, P.D., Chacoff, N.,P. and Cagnolo, L. (2009) Evaluating multiple determinants of the structure of plant-animal mutualistic networks. *Ecology*, 90:2039-2046.

See Also

See Also visweb

Examples

```
signpred(matrix(rbinom(160,1,.2),ncol=8,dimnames=list(as.character(1:20),as.character(1:8))))
```

---

simul_data_complete   *Data generating detailed process for multivariate plsR models*

Description

This function generates a single multivariate response value $\boldsymbol{Y}$ and a vector of explinatory variables $(X_1, \ldots, X_{totdim})$ drawn from a model with a given number of latent components.

Usage

```
simul_data_complete(totdim, ncomp)
```

Arguments

totdim	Number of columns of the X vector (from ncomp to hardware limits)
ncomp	Number of latent components in the model (from 2 to 6)

Details

This function should be combined with the replicate function to give rise to a larger dataset. The algorithm used is a port of the one described in the article of Li which is a multivariate generalization of the algorithm of Naes and Martens.

Value

`simX`	Vector of explanatory variables
`HH`	Dimension of the response $\boldsymbol{Y}$
`eta`	See Li et al.
`r`	See Li et al.
`epsilon`	See Li et al.
`ksi`	See Li et al.
`f`	See Li et al.
`z`	See Li et al.
`Y`	See Li et al.

Note

The value of r depends on the value of `ncomp` :

`ncomp`	r
2	3
3	3
4	4

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

T. Naes, H. Martens, Comparison of prediction methods for multicollinear data, Commun. Stat., Simul. 14 (1985) 545-576.

Morris, Elaine B. Martin, Model selection for partial least squares regression, Chemometrics and Intelligent Laboratory Systems 64 (2002) 79-89, doi: 10.1016/S01697439(02)000515.

See Also

`simul_data_YX` for data simulation purpose

Examples

```
simul_data_complete(20,6)

dimX <- 6
Astar <- 2
simul_data_complete(dimX,Astar)


dimX <- 6
Astar <- 3
simul_data_complete(dimX,Astar)


dimX <- 6
Astar <- 4
simul_data_complete(dimX,Astar)

rm(list=c("dimX","Astar"))
```

`simul_data_UniYX` *Data generating function for univariate plsR models*

Description

This function generates a single univariate response value Y and a vector of explanatory variables $(X_1, \ldots, X_{totdim})$ drawn from a model with a given number of latent components.

Usage

```
simul_data_UniYX(totdim, ncomp)
```

Arguments

`totdim`	Number of columns of the X vector (from `ncomp` to hardware limits)
`ncomp`	Number of latent components in the model (from 2 to 6)

Details

This function should be combined with the replicate function to give rise to a larger dataset. The algorithm used is a port of the one described in the article of Li which is a multivariate generalization of the algorithm of Naes and Martens.

Value

`vector`	$(Y, X_1, \ldots, X_{totdim})$

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

T. Naes, H. Martens, Comparison of prediction methods for multicollinear data, Commun. Stat., Simul. 14 (1985) 545-576.

Morris, Elaine B. Martin, Model selection for partial least squares regression, Chemometrics and Intelligent Laboratory Systems 64 (2002) 79-89, doi: 10.1016/S01697439(02)000515.

See Also

simul_data_YX and simul_data_complete for generating multivariate data

Examples

```
simul_data_UniYX(20,6)


dimX <- 6
Astar <- 2
simul_data_UniYX(dimX,Astar)
(dataAstar2 <- data.frame(t(replicate(50,simul_data_UniYX(dimX,Astar)))))
cvtable(summary(cv.plsR(Y~.,data=dataAstar2,5,NK=100, verbose=FALSE)))

dimX <- 6
Astar <- 3
simul_data_UniYX(dimX,Astar)
(dataAstar3 <- data.frame(t(replicate(50,simul_data_UniYX(dimX,Astar)))))
cvtable(summary(cv.plsR(Y~.,data=dataAstar3,5,NK=100, verbose=FALSE)))

dimX <- 6
Astar <- 4
simul_data_UniYX(dimX,Astar)
(dataAstar4 <- data.frame(t(replicate(50,simul_data_UniYX(dimX,Astar)))))
cvtable(summary(cv.plsR(Y~.,data=dataAstar4,5,NK=100, verbose=FALSE)))

rm(list=c("dimX","Astar","dataAstar2","dataAstar3","dataAstar4"))
```

simul_data_UniYX_binom — *Data generating function for univariate binomial plsR models*

Description

This function generates a single univariate binomial response value Y and a vector of explanatory variables $(X_1, \ldots, X_{totdim})$ drawn from a model with a given number of latent components.

Usage

```
simul_data_UniYX_binom(totdim, ncomp, link = "logit", offset = 0)
```

Arguments

`totdim`	Number of columns of the X vector (from `ncomp` to hardware limits)
`ncomp`	Number of latent components in the model (from 2 to 6)
`link`	Character specification of the link function in the mean model (mu). Currently, `"logit"`, `"probit"`, `"cloglog"`, `"cauchit"`, `"log"`, `"loglog"` are supported. Alternatively, an object of class "link-glm" can be supplied.
`offset`	Offset on the linear scale

Details

This function should be combined with the replicate function to give rise to a larger dataset. The algorithm used is a modification of a port of the one described in the article of Li which is a multivariate generalization of the algorithm of Naes and Martens.

Value

`vector`	$(Y, X_1, \ldots, X_{totdim})$

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

T. Naes, H. Martens, Comparison of prediction methods for multicollinear data, Commun. Stat., Simul. 14 (1985) 545-576.

Morris, Elaine B. Martin, Model selection for partial least squares regression, Chemometrics and Intelligent Laboratory Systems 64 (2002), 79-89, doi: 10.1016/S01697439(02)000515.

See Also

simul_data_UniYX

Examples

```
layout(matrix(1:6,nrow=2))
# logit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4)))[,1])
# probit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="probit")))[,1])
# cloglog link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="cloglog")))[,1])
# cauchit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="cauchit")))[,1])
# loglog link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="loglog")))[,1])
# log link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="log")))[,1])
layout(1)


layout(matrix(1:6,nrow=2))
# logit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,offset=5)))[,1])
# probit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="probit",offset=5)))[,1])
# cloglog link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="cloglog",offset=5)))[,1])
# cauchit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="cauchit",offset=5)))[,1])
# loglog link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="loglog",offset=5)))[,1])
# log link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="log",offset=5)))[,1])
layout(1)


layout(matrix(1:6,nrow=2))
# logit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,offset=-5)))[,1])
# probit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="probit",offset=-5)))[,1])
# cloglog link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="cloglog",offset=-5)))[,1])
# cauchit link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="cauchit",offset=-5)))[,1])
# loglog link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="loglog",offset=-5)))[,1])
# log link
hist(t(replicate(100,simul_data_UniYX_binom(4,4,link="log",offset=-5)))[,1])
```

```
layout(1)
```

simul_data_YX *Data generating function for multivariate plsR models*

Description

This function generates a single multivariate response value $\boldsymbol{Y}$ and a vector of explinatory variables $(X_1, \ldots, X_{totdim})$ drawn from a model with a given number of latent components.

Usage

```
simul_data_YX(totdim, ncomp)
```

Arguments

`totdim`	Number of column of the X vector (from `ncomp` to hardware limits)
`ncomp`	Number of latent components in the model (from 2 to 6)

Details

This function should be combined with the replicate function to give rise to a larger dataset. The algorithm used is a port of the one described in the article of Li which is a multivariate generalization of the algorithm of Naes and Martens.

Value

`vector`	$(Y_1, \ldots, Y_r, X_1, \ldots, X_{totdim})$

Note

The value of r depends on the value of `ncomp` :

`ncomp`	r
2	3
3	3
4	4

Author(s)

Frédéric Bertrand
<frederic.bertrand@math.unistra.fr>
https://fbertran.github.io/homepage/

References

T. Naes, H. Martens, Comparison of prediction methods for multicollinear data, Commun. Stat., Simul. 14 (1985) 545-576.

Morris, Elaine B. Martin, Model selection for partial least squares regression, Chemometrics and Intelligent Laboratory Systems 64 (2002) 79-89, doi: 10.1016/S01697439(02)000515.

See Also

`simul_data_complete` for highlighting the simulations parameters

Examples

```
simul_data_YX(20,6)


if(require(plsdepot)){
dimX <- 6
Astar <- 2
(dataAstar2 <- t(replicate(50,simul_data_YX(dimX,Astar))))
library(plsdepot)
resAstar2 <- plsreg2(dataAstar2[,4:9],dataAstar2[,1:3],comps=5)
resAstar2$Q2
resAstar2$Q2[,4]>0.0975

dimX <- 6
Astar <- 3
(dataAstar3 <- t(replicate(50,simul_data_YX(dimX,Astar))))
library(plsdepot)
resAstar3 <- plsreg2(dataAstar3[,4:9],dataAstar3[,1:3],comps=5)
resAstar3$Q2
resAstar3$Q2[,4]>0.0975

dimX <- 6
Astar <- 4
(dataAstar4 <- t(replicate(50,simul_data_YX(dimX,Astar))))
library(plsdepot)
resAstar4 <- plsreg2(dataAstar4[,5:10],dataAstar4[,1:4],comps=5)
resAstar4$Q2
resAstar4$Q2[,5]>0.0975

rm(list=c("dimX","Astar"))
}
```

`summary.cv.plsRglmmodel` *Summary method for plsRglm models*

Description

This function provides a summary method for the class "`cv.plsRglmmodel`"

Usage

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

Arguments

`object`	an object of the class "`cv.plsRglmmodel`"
`...`	further arguments to be passed to or from methods.

Value

An object of class "`summary.cv.plsRmodel`" if `model` is missing or `model="pls"`. Otherwise an object of class "`summary.cv.plsRglmmodel`".

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`summary`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
summary(cv.plsRglm(Y~.,data=Cornell,nt=10,NK=1,
modele="pls-glm-family",family=gaussian(), verbose=FALSE))
```

```
rm(list=c("XCornell","yCornell","bbb"))
```

`summary.cv.plsRmodel` *Summary method for plsR models*

Description

This function provides a summary method for the class `"cv.plsRmodel"`

Usage

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

Arguments

`object`	an object of the class `"cv.plsRmodel"`
`...`	further arguments to be passed to or from methods.

Value

An object of class `"summary.cv.plsRglmmodel"`.

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`summary`

Examples

```
data(Cornell)
summary(cv.plsR(Y~.,data=Cornell,nt=10,K=6, verbose=FALSE), verbose=FALSE)
```

`summary.plsRglmmodel` *Summary method for plsRglm models*

Description

This function provides a summary method for the class "`plsRglmmodel`"

Usage

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

Arguments

`object`	an object of the class "`plsRglmmodel`"
`...`	further arguments to be passed to or from methods.

Value

`call`	function call of plsRglmmodel

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`summary`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modplsglm <- plsRglm(yCornell,XCornell,3,modele="pls-glm-gaussian")
class(modplsglm)
summary(modplsglm)
rm(list=c("XCornell","yCornell","modplsglm"))
```

`summary.plsRmodel` *Summary method for plsR models*

Description

This function provides a summary method for the class "`plsRmodel`"

Usage

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

Arguments

`object`	an object of the class "`plsRmodel`"
`...`	further arguments to be passed to or from methods.

Value

`call`	function call of plsRmodel

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

References

Nicolas Meyer, Myriam Maumy-Bertrand et Frédéric Bertrand (2010). Comparing the linear and the logistic PLS regression with qualitative predictors: application to allelotyping data. *Journal de la Societe Francaise de Statistique*, 151(2), pages 1-18. `http://publications-sfds.math.cnrs.fr/index.php/J-SFdS/article/view/47`

See Also

`summary`

Examples

```
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]
modpls <- plsR(yCornell,XCornell,3,modele="pls")
class(modpls)
summary(modpls)
rm(list=c("XCornell","yCornell","modpls"))
```

tilt.bootpls *Non-parametric tilted bootstrap for PLS regression models*

Description

Provides a wrapper for the bootstrap function `tilt.boot` from the `boot` R package.
Implements non-parametric tilted bootstrap for PLS regression models by case resampling : the `tilt.boot` function will run an initial bootstrap with equal resampling probabilities (if required) and will use the output of the initial run to find resampling probabilities which put the value of the statistic at required values. It then runs an importance resampling bootstrap using the calculated probabilities as the resampling distribution.

Usage

```
tilt.bootpls(
  object,
  typeboot = "plsmodel",
  statistic = coefs.plsR,
  R = c(499, 250, 250),
  alpha = c(0.025, 0.975),
  sim = "ordinary",
  stype = "i",
  index = 1,
  stabvalue = 1e+06,
  ...
)
```

Arguments

`object`	An object of class `plsRmodel` to bootstrap
`typeboot`	The type of bootstrap. Either (Y,X) boostrap (`typeboot="plsmodel"`) or (Y,T) bootstrap (`typeboot="fmodel_np"`). Defaults to (Y,T) resampling.
`statistic`	A function which when applied to data returns a vector containing the statistic(s) of interest. `statistic` must take at least two arguments. The first argument passed will always be the original data. The second will be a vector of indices, frequencies or weights which define the bootstrap sample. Further, if predictions are required, then a third argument is required which would be a vector of the random indices used to generate the bootstrap predictions. Any further arguments can be passed to statistic through the `...` argument.
`R`	The number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case `R` would be a vector of integers where each component gives the number of resamples from each of the rows of weights.

`alpha`	The alpha level to which tilting is required. This parameter is ignored if `R[1]` is 0 or if `theta` is supplied, otherwise it is used to find the values of `theta` as quantiles of the initial uniform bootstrap. In this case `R[1]` should be large enough that `min(c(alpha,1-alpha))*R[1]` > 5, if this is not the case then a warning is generated to the effect that the `theta` are extreme values and so the tilted output may be unreliable.
`sim`	A character string indicating the type of simulation required. Possible values are `"ordinary"` (the default), `"balanced"`, `"permutation"`, or `"antithetic"`.
`stype`	A character string indicating what the second argument of `statistic` represents. Possible values of stype are `"i"` (indices - the default), `"f"` (frequencies), or `"w"` (weights).
`index`	The index of the statistic of interest in the output from `statistic`. By default the first element of the output of `statistic` is used.
`stabvalue`	Upper bound for the absolute value of the coefficients.
`...`	ny further arguments can be passed to `statistic`.

Value

An object of class "boot".

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
`https://fbertran.github.io/homepage/`

See Also

`tilt.boot`

Examples

```
## Not run:
data(Cornell)
XCornell<-Cornell[,1:7]
yCornell<-Cornell[,8]

set.seed(1385)
Cornell.tilt.boot <- tilt.bootpls(plsR(yCornell,XCornell,1), statistic=coefs.plsR,
typeboot="fmodel_np", R=c(499, 100, 100), alpha=c(0.025, 0.975), sim="ordinary",
stype="i", index=1)
Cornell.tilt.boot
str(Cornell.tilt.boot)

boxplots.bootpls(Cornell.tilt.boot,indices=2:7)

rm(Cornell.tilt.boot)
```

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

tilt.bootplsglm — *Non-parametric tilted bootstrap for PLS generalized linear regression models*

Description

Provides a wrapper for the bootstrap function `tilt.boot` from the `boot` R package.
Implements non-parametric tilted bootstrap for PLS generalized linear regression models by case resampling : the `tilt.boot` function will run an initial bootstrap with equal resampling probabilities (if required) and will use the output of the initial run to find resampling probabilities which put the value of the statistic at required values. It then runs an importance resampling bootstrap using the calculated probabilities as the resampling distribution.

Usage

```
tilt.bootplsglm(
  object,
  typeboot = "fmodel_np",
  statistic = coefs.plsRglm,
  R = c(499, 250, 250),
  alpha = c(0.025, 0.975),
  sim = "ordinary",
  stype = "i",
  index = 1,
  stabvalue = 1e+06,
  ...
)
```

Arguments

`object`	An object of class `plsRbetamodel` to bootstrap
`typeboot`	The type of bootstrap. Either (Y,X) boostrap (`typeboot="plsmodel"`) or (Y,T) bootstrap (`typeboot="fmodel_np"`). Defaults to (Y,T) resampling.
`statistic`	A function which when applied to data returns a vector containing the statistic(s) of interest. `statistic` must take at least two arguments. The first argument passed will always be the original data. The second will be a vector of indices, frequencies or weights which define the bootstrap sample. Further, if predictions are required, then a third argument is required which would be a vector of the random indices used to generate the bootstrap predictions. Any further arguments can be passed to statistic through the `...` argument.
`R`	The number of bootstrap replicates. Usually this will be a single positive integer. For importance resampling, some resamples may use one set of weights and others use a different set of weights. In this case `R` would be a vector of integers where each component gives the number of resamples from each of the rows of weights.

`alpha`	The alpha level to which tilting is required. This parameter is ignored if `R[1]` is 0 or if `theta` is supplied, otherwise it is used to find the values of `theta` as quantiles of the initial uniform bootstrap. In this case `R[1]` should be large enough that `min(c(alpha,1-alpha))*R[1]` > 5, if this is not the case then a warning is generated to the effect that the `theta` are extreme values and so the tilted output may be unreliable.
`sim`	A character string indicating the type of simulation required. Possible values are `"ordinary"` (the default), `"balanced"`, `"permutation"`, or `"antithetic"`.
`stype`	A character string indicating what the second argument of `statistic` represents. Possible values of stype are `"i"` (indices - the default), `"f"` (frequencies), or `"w"` (weights).
`index`	The index of the statistic of interest in the output from `statistic`. By default the first element of the output of `statistic` is used.
`stabvalue`	Upper bound for the absolute value of the coefficients.
`...`	ny further arguments can be passed to `statistic`.

Value

An object of class "boot".

Author(s)

Frédéric Bertrand
`<frederic.bertrand@math.unistra.fr>`
https://fbertran.github.io/homepage/

See Also

`tilt.boot`

Examples

```
data(aze_compl)
Xaze_compl<-aze_compl[,2:34]
yaze_compl<-aze_compl$y

dataset <- cbind(y=yaze_compl,Xaze_compl)

# Lazraq-Cleroux PLS bootstrap Classic

aze_compl.tilt.boot <- tilt.bootplsglm(plsRglm(yaze_compl,Xaze_compl,3,
modele="pls-glm-logistic", family=NULL), statistic=coefs.plsRglm, R=c(499, 100, 100),
alpha=c(0.025, 0.975), sim="ordinary", stype="i", index=1)
boxplots.bootpls(aze_compl.tilt.boot,1:2)

aze_compl.tilt.boot2 <- tilt.bootplsglm(plsRglm(yaze_compl,Xaze_compl,3,
modele="pls-glm-logistic"), statistic=coefs.plsRglm, R=c(499, 100, 100),
alpha=c(0.025, 0.975), sim="ordinary", stype="i", index=1)
boxplots.bootpls(aze_compl.tilt.boot2,1:2)

aze_compl.tilt.boot3 <- tilt.bootplsglm(plsRglm(yaze_compl,Xaze_compl,3,
modele="pls-glm-family", family=binomial), statistic=coefs.plsRglm, R=c(499, 100, 100),
alpha=c(0.025, 0.975), sim="ordinary", stype="i", index=1)
boxplots.bootpls(aze_compl.tilt.boot3,1:2)


# PLS bootstrap balanced

aze_compl.tilt.boot4 <- tilt.bootplsglm(plsRglm(yaze_compl,Xaze_compl,3,
modele="pls-glm-logistic"), statistic=coefs.plsRglm, R=c(499, 100, 100),
alpha=c(0.025, 0.975), sim="balanced", stype="i", index=1)
boxplots.bootpls(aze_compl.tilt.boot4,1:2)
```

XbordeauxNA *Incomplete dataset for the quality of wine dataset*

Description

Quality of Bordeaux wines (Quality) and four potentially predictive variables (Temperature, Sunshine, Heat and Rain).
The value of Temperature for the first observation was remove from the matrix of predictors on purpose.

Format

A data frame with 34 observations on the following 4 variables.

Temperature a numeric vector

Sunshine a numeric vector

Heat a numeric vector

Rain a numeric vector

Source

P. Bastien, V. Esposito-Vinzi, and M. Tenenhaus. (2005). PLS generalised linear regression. *Computational Statistics & Data Analysis*, 48(1):17-46.

References

M. Tenenhaus. (2005). La regression logistique PLS. In J.-J. Droesbeke, M. Lejeune, and G. Saporta, editors, Modeles statistiques pour donnees qualitatives. Editions Technip, Paris.

Examples

```
data(XbordeauxNA)
str(XbordeauxNA)
```

XpineNAX21 *Incomplete dataset from the pine caterpillars example*

Description

The caterpillar dataset was extracted from a 1973 study on pine processionary caterpillars. It assesses the influence of some forest settlement characteristics on the development of caterpillar colonies. There are k=10 potentially explanatory variables defined on n=33 areas.
The value of x2 for the first observation was remove from the matrix of predictors on purpose.

Format

A data frame with 33 observations on the following 10 variables and one missing value.

x1 altitude (in meters)

x2 slope (en degrees)

x3 number of pines in the area

x4 height (in meters) of the tree sampled at the center of the area

x5 diameter (in meters) of the tree sampled at the center of the area

x6 index of the settlement density

x7 orientation of the area (from 1 if southbound to 2 otherwise)

x8 height (in meters) of the dominant tree

x9 number of vegetation strata

x10 mix settlement index (from 1 if not mixed to 2 if mixed)

Details

These caterpillars got their names from their habit of moving over the ground in incredibly long head-to-tail processions when leaving their nest to create a new colony.
The `XpineNAX21` is a dataset with a missing value for testing purpose.

Source

Tomassone R., Audrain S., Lesquoy-de Turckeim E., Millier C. (1992). “La régression, nouveaux regards sur une ancienne méthode statistique”, INRA, *Actualités Scientifiques et Agronomiques*, Masson, Paris.

Examples

```
data(XpineNAX21)
str(XpineNAX21)
```

Index