Package 'TPLSr'

April 8, 2021

Type Package

Title Thresholded Partial Least Squares Model for Neuroimaging Data

Version 1.0.3

Description Uses thresholded partial least squares algorithm to create a regression or classification model. For more information, see Lee, Bradlow, and Kable <doi:10.1101/2021.02.09.430524>.

License GPL-3

Depends R (>= 3.5), plotly (>= 4.9.2.1)

Encoding UTF-8

LazyData true

NeedsCompilation no

RoxygenNote 7.1.1

LazyDataCompression xz

Author Sangil Lee [aut, cre]

Maintainer Sangil Lee <sangillee3rd@gmail.com>

Repository CRAN

Date/Publication 2021-04-08 21:40:11 UTC

R topics documented:

evalTuningParam *Evaluate TPLS tuning parameters using cross validation*

Description

Evaluate TPLS tuning parameters using cross validation

Usage

```
evalTuningParam(
  TPLScvmdl,
  type = c("pearson", "spearman", "AUC"),
  X,
  Y,
  compvec,
  threshvec,
  subfold = NULL
)
```

Arguments

`TPLScvmdl`	TPLS_cv model created from `TPLS_cv`
`type`	Cross validation performance measure type. One of 'pearson', 'spearman', or 'AUC'
`X`	The SAME X that was used to create the `TPLScvmdl`. If it's not the same, the function may not work or the results will be completely off
`Y`	The SAME Y that was used to create the `TPLScvmdl`.
`compvec`	Vector containing the number of components you want to assess CV performance for (e.g., c(3,4,5) will provide CV performance of 3, 4, and 5 component TPLS model at various thresholds)
`threshvec`	Vector containing the thresholding level betweeon 0 and 1 you want to assess CV performance for (e.g., seq(0,1,0.1) will provide CV performance of TPLS models at thresholds of 0, 0.1, 0.2, ... ,1)
`subfold`	Optional vector containing smaller data division within folds. For example, if the cross-validation was done at the subject level, with each testing fold being a subject, subfold can be the run number of the scan of each person. This allows for calculation of average CV metric at the run level instead of at the subject level.

Value

A evalTuningParam object that contains the following attributes.

- `type`: Cross validation performance measure type, as specified in the input
- `threshval`: Same as the input threshvec

- `compval`: Same as the input compvec
- `perfmat`: Performance measure 3D matrix: length(compvec)-by-length(threshvec)-by-numfold
- `perf_best`: Best CV performance out of all combinations of compvec and threshvec
- `compval_best`: Number of components that gave the best performance (i.e., perf_best)
- `threshval_best`: Threshold level that gave the best performance (i.e., perf_best)
- `perf_1se` : Performance of the most parsimonious model (least number of coefficients) that is within 1 standard error of perf_best.
- `compval_1se` : Number of components that gave perf_1se
- `threshval_1se` : Threshold level that gave perf_1se

Examples

```
# see examples under TPLS_cv as you'd need a TPLS_cv object to run this function
```

`makePredictor` *Extracts a predictor (betamap and intercept) from a TPLS model at a given number of components and given threshold value*

Description

Extracts a predictor (betamap and intercept) from a TPLS model at a given number of components and given threshold value

Usage

```
makePredictor(TPLSmdl, compval, threshval)
```

Arguments

`TPLSmdl`	A TPLS object created from using function `TPLS`
`compval`	The number of components you want in your model. Providing a vector will provide multiple betamaps (e.g., c(3,4,5) will provide three betamaps each with 3, 4, and 5 PLS components)
`threshval`	Threshold number between 0 and 1 (inclusive) for thresholding the betamap. This must be a scalar.

Value

- `bias`: The intercept of the extracted model. Vector of intercepts if compval is a vector.
- `betamap`: Column vector of betamap. Matrix of betamaps if compval is a vector.

Examples

```
# See examples for TPLS
```

`plotTuningSurface` *Plots the tuning surface of TPLS*

Description

Plots the tuning surface of TPLS

Usage

```
plotTuningSurface(object)
```

Arguments

`object`	: evalTuningParam object

Examples

```
# See examples for TPLS_cv
```

`TPLS` *Fit a TPLS model to data*

Description

Fit a TPLS model to data

Usage

```
TPLS(X, Y, NComp = 50, W = 0, nmc = 0)
```

Arguments

`X`	n-by-v data matrix of real numbers. Rows correspond to observations (trials) and columns to variables (e.g., fMRI voxels).
`Y`	n-by-1 Vector of real numbers. Can be binary (0/1) for classification model, or can be continuous.
`NComp`	Maximum number of partial least squares component you want to use. Default is 50, and this is on the safe side for fMRI.
`W`	n-by-1 vector of positive observation weights.
`nmc`	A switch to skip mean-centering. Default is off (0). Only turn it on (1) when the data is already mean-centered and you want to save memory space by not creating another copy of the data for mean-centering.

Value

A TPLS object that contains the following attributes. Most of the time, you won't need to access the attributes.

- `NComp`: The number of components you specified in the input
- `W`: Normalized version of the observation weights (i.e., they sum to 1)
- `MtrainX`: Column mean of X. Weighted mean if W is given.
- `MtrainY`: Mean of Y. Weighted mean if W is given.
- `scoreCorr`: Correlation between Y and each PLS component. Weighted correlation if W is given.
- `pctVar`: Proportion of variance of Y that each component explains.
- `betamap`: v-by-NComp matrix of TPLS coefficients for each of the v variables, provided at each model with NComp components.
- `threshmap` : v-by-NComp matrix of TPLS threshold values (0~1) for each of the v variables, provided at each model with NComp components.

Examples

```
# Fit example TPLS data with a TPLS model
# Load example data (included with package).
X = TPLSdat$X
Y = TPLSdat$Y

# Fit the model, with default options (50 components, no observation weights)
TPLSmdl <- TPLS(X,Y)

# Make in-sample prediction at threshold of 0.5 and at all possible components
pred <- TPLSpredict(TPLSmdl,1:50,0.5,X)

# Look at the correlation between prediction and Y.
# This is in-sample prediction. Ergo, the model with most components will have the highest
# predictive correlation. In practice, you should choose the number of components and
# threshold using cross-validation. See example for TPLS_cv
cor(Y,pred)

# Extract the predictor for a model with 25 PLS components and threshold at 0.7 (just cuz)
betamap <- makePredictor(TPLSmdl,25,0.5)

# This is the intercept
betamap$bias

# These are the coefficients for the original variables
betamap$betamap
```

`TPLSdat` *Sample participant data from a left-right button press task*

Description

A dataset containing five sample participant's binary button presses inside the scanner (left/right).

Usage

```
TPLSdat
```

Format

A data frame with following variables

X Brain image single trial coefficients. N-by-v matrix

Y Left = 0, Right = 1, binary indicator of participant choice

subj Subject number (i.e., 1, 2, 3)

run Run number (i.e., 1, 2, 3, 4, 5, 6, 7, 8)

mask Binary 3D brain image that indexes where the variables in X came from.

Source

Kable, J. W., Caulfield, M. K., Falcone, M., McConnell, M., Bernardo, L., Parthasarathi, T., ... & Diefenbach, P. (2017). No effect of commercial cognitive training on brain activity, choice behavior, or cognitive performance. Journal of Neuroscience, 37(31), 7390-7402.

`TPLSpredict` *Make predictions about given data `testX` by using an extracted TPLSmdl with `compval` components and `threshval` threshold.*

Description

Make predictions about given data `testX` by using an extracted TPLSmdl with `compval` components and `threshval` threshold.

Usage

```
TPLSpredict(TPLSmdl, compval, threshval, testX)
```

Arguments

`TPLSmdl`	A TPLS object created from using function `TPLS`
`compval`	The number of components you want in your model. Providing a vector will provide multiple predictions (e.g., c(3,4,5) will provide three prediction columns each with 3, 4, and 5 PLS components)
`threshval`	Threshold number between 0 and 1 (inclusive) for thresholding the betamap. This must be a scalar.
`testX`	Data that you want to predict the Y of

Value

- `score`: Column vector of prediction scores. Matrix of scores if compval is a vector.

Examples

```
# See examples for TPLS
```

`TPLS_cv` *Fit a TPLS model to data with cross validation*

Description

Fit a TPLS model to data with cross validation

Usage

```
TPLS_cv(X, Y, foldid, NComp = 50, W = 0)
```

Arguments

`X`	n-by-v data matrix of real numbers. Rows correspond to observations (trials) and columns to variables (e.g., fMRI voxels).
`Y`	n-by-1 Vector of real numbers. Can be binary (0/1) for classification model, or can be continuous.
`foldid`	A vector of values between 1 and number of folds identifying what fold each observation is in.
`NComp`	Maximum number of partial least squares component you want to use. Default is 50, and this is on the safe side for fMRI.
`W`	n-by-1 vector of positive observation weights.

Value

A TPLS_cv object that contains the following attributes. Most of the time, you won't need to access the attributes.

- `NComp`: The number of components you specified in the input
- `numfold`: Total number of cross-validation folds
- `testfold`: A vector indice that should be the same as foldid, if it was provided accurately.
- `cvMdls` : A vector of TPLS models, one for each fold.

Examples

```
# Fit example TPLS data with a TPLS model using cross-validation
# Load example data (included with package).
X = TPLSdat$X # single trial brain image of subjects pressing left/right buttons
Y = TPLSdat$Y # binary variable that is 1 if right button is pushed, 0 if left button is pushed
subj = TPLSdat$subj # 1, 2, or 3, depending on who the subject is
run = TPLSdat$run # 1, 2, ..., 8, depending on the scan run of each subject

# Fit the model, using 3-fold cross-validation at the subject level
# (i.e., train on two subjects, test on 1, repeat three times)
TPLScvmdl <- TPLS_cv(X,Y,subj)

# Evaluate the tuning parameters via cross-validation.
# We'll test 1~50 components and thresholding from 0 to 1 in 0.05 increments.
# Also include subfold information.
# This allows for calculation of correlation at the run-level instead of at the subject level.
cvstats <- evalTuningParam(TPLScvmdl,"pearson",X,Y,1:50,seq(0,1,0.05),subfold=run)

# plot the tuning parameter surface.
# It'll show the point of best performance (and also point of 1SE performance).
# The plot is interactive, so spin it around
plotTuningSurface(cvstats)

# These are the tuning parameters of best performance
cvstats$compval_best # 8 components
cvstats$threshval_best # 0.1 thresholding (leave only 10% of all voxels)

# Now build a new TPLS model, using all the data, using the best tuning parameters
TPLSmdl <- TPLS(X,Y,NComp=cvstats$compval_best)

# Extract the prediction betamap that gave the best CV performance
betamap <- makePredictor(TPLSmdl,cvstats$compval_best,cvstats$threshval_best)

# This is the intercept
betamap$bias

# These are the coefficients for the original variables
betamap$betamap

# Project the betamap into brain-space so that we can look at it.
mask = TPLSdat$mask # mask 3D image of the brain from which X was extracted from
```

```
brainimg = mask*1 # make a copy
brainimg[mask] = betamap$betamap # put the betamap into the brain image
fig1 <- plot_ly(z = brainimg[,15,], type = "heatmap") # looking at a slice of the brain image
fig2 <- plot_ly(z = 1*mask[,15,], type = "heatmap") # a slice of the brain mask for reference
fig <- subplot(fig1, fig2)
fig

# Figures show a coronal section of the brain (but flipped right 90 degrees).
# on the left, you should see the bilateral motor cortex coefficients with opposing signs.
# This is just a simple visual demonstration. You should use other packages to output
# coefficients into a nifti file and view them in a separate viewer.
```

Index