Package 'graphclass'

September 21, 2018

Title Network classification

Version 1.0

Date 2017-20-02

Author Jesus Arroyo Relion, Daniel Kessler, Elizaveta Levina and Stephan F. Taylor

Maintainer Jesus Arroyo Relion <jarroyor@umich.edu>

Description Network classification with applications to brain connectomics.

Depends R (>= 3.2.2)

Imports Rdpack, Matrix, lattice

License GPL-2

Encoding UTF-8

LazyData true

RdMacros Rdpack

RoxygenNote 6.0.1

R topics documented:

COBRE.data *fMRI brain networks of the COBRE dataset*

Description

This dataset contains fMRI brain networks of 124 subjects and corresponding class labels indicating schizophrenia status (70 healthy controls and 54 schizophrenic subjects).

Usage

```
COBRE.data
```

Format

`COBRE.data` is a list with two elements

X.cobre A matrix containing the upper triangular part of the networks. Each row represents a subject, and the columns represent edges. The edge weights represent the Fisher-transformed correlation between the fMRI time series of the nodes. The networks are composed of 263 nodes and 34453 different edges. For a description of the preprocessing steps to obtain the network edge weights, see Arroyo-Relion et al. (2017). .

Y.cobre Class labels of the subjects in the dataset. `Y=1` represents schizophrenia status.

Source

http://fcon_1000.projects.nitrc.org/indi/retro/cobre.html

Examples

```
data(COBRE.data)
```

construct_D *Constructor for penalty matrix*

Description

This function constructs an auxiliary matrix to compute the node penalty in the regularized classifier, which is internally used by the optimization algorithm. This matrix can be passed to the function `graphclass` in order to avoid creating it every time this function is called. Use it for efficiency when running the classifier multiple times (for example, in a cross-validation routine).

Usage

```
construct_D(nodes = 264)
```

Arguments

`nodes`	Number of nodes in the network, by default is 264 (Power parcellation).

Value

A sparse matrix

Examples

```
D263 = construct_D(263)

# Load COBRE data
data(COBRE.data)
X <- COBRE.data$X.cobre
Y <- COBRE.data$Y.cobre

# 5-fold cross validation of the subgraph selection penalty
fold_index <- (1:length(Y) %% 5) + 1

gclist <- list()
for(fold in 1:5) {
    foldout <- which(fold_index == fold)
    gclist[[fold]] <- graphclass(X = X[-foldout,], Y = factor(Y[-foldout]),
                      Xtest = X[foldout,], Ytest = factor(Y[foldout]),
                      type = "intersection",
                      lambda = 1e-4, rho = 1, gamma = 1e-5,
                      D = D263)
}
# test error on each fold
lapply(gclist, function(gc) gc$test_error)
```

---

get_matrix *Converts a vector into an adjacency matrix*

Description

Given a vector that encodes an adjacency matrix, returns the matrix representation.

Usage

```
get_matrix(beta, type = c("undirected", "directed"))
```

Arguments

beta	Vectorized adjacency matrix. If the network is undirected, the vector is assumed to represent the upper triangular part of the adjacency matrix in column major order. For undirected network, the vector contains the entries ordered by column and excluding the diagonal elements.
type	Specifies whether the vector represents an `undirected` or `directed` network. Default is `undirected`.

Value

Adjacency matrix.

Examples

```
# Obtain the adjacency matrix of a COBRE data subject
data(COBRE.data)
A <- get_matrix(COBRE.data$X.cobre[1,], type = "undirected")
plot_adjmatrix(A)
```

---

graphclass *Regularized logistic regression classifier for networks.*

Description

graphclass fits a regularized logistic regression to a set of network adjacency matrices with responses, and returns an object with the classifier.

Usage

```
graphclass(X = NULL, Y = NULL, type = c("intersection", "union", "groups",
  "fusion"), ...)

## Default S3 method:
graphclass(X = NULL, Y = NULL, Xtest = NULL,
  Ytest = NULL, Adj_list = NULL, type = c("intersection", "union",
  "groups", "fusion"), lambda = 0, rho = 0, gamma = 1e-05,
  params = NULL, id = "", verbose = F, D = NULL, Groups = NULL,
  G_penalty_factors = NULL, ...)
```

Arguments

X	A matrix with the training samples, in wich each row represents the vectorized (by column order) upper triangular part of a network adjacency matrix.
Y	A vector containing the class labels of the training samples (only 2 classes are supported for now).
type	Type of penalty function. Default is "intersection". See details.
Xtest	Optional argument for providing a matrix containing the test samples, with each row representing an upper-triangular vectorized adjacency matrix.
Ytest	Optional argument containing the labels of test samples.
Adj_list	A training list of of symmetric adjacency matrices with zeros in the diagonal
lambda	penalty parameter $lambda$, by default is set to 0.
rho	penalty parameter rho controlling sparsity, by default is set to 0.
gamma	ridge parameter (for numerical purposes). Default is gamma = 1e-5.
params	A list containing internal parameters for the optimization algorithm. See details.
verbose	whether output is printed
D	matrix D used by the penalty to define the groups. This optional argument can be used to pass a precomputed matrix D, which can be time saving if the method is fitted multiple times. See the function construct_D.
Groups	list of lists, where each list correspond to a grouping and each sublist to sets of indexes in X. Each sublist should be a non-overlapping group.
G_penalty_factors	For type "groups", each group is penalized by this factor. Should sum to 1.

Details

The function `graphclass` fits a regularized logistic regression to classify a set of network adjacency matrices with N labeled nodes and corresponding responses. The classifier fits a matrix of coefficients $B \in R^{N \times N}$, in which B_{ij} indicates the coefficient corresponding to the edge (i, j).

The argument `type` provides options to choose the penalty function. If `type = "intersection"` or `"union"`, the penalty corresponds to the node selection penalty defined as

$$\Omega(B) = \lambda \left(\sum_{i=1}^{N} \sqrt{\sum_{j=1}^{N} B_{ij}^2} + \rho \sum_{i=1}^{N} \sum_{j=1}^{N} |B_{ij}| \right).$$

When `type = "intersection"`, a symmetric restriction on B is enforced, and the penalty promotes subgraph selection. If `type = "union"`, the penalty promotes individual node selection. See Arroyo-Relion et al. (2017) for more details.

The value `type = "groups"` corresponds to a generic group lasso penalty. The groups of edges have to be specified using the argument `Groups` with a list of arrays, in which each element of the list corresponds to a group, and the array indicates the indexes of the variables in that group. The optional argument `G_penalty_factors` is an array of size equal to the number of groups, and can be used to specify different weights for each group on the penalty (for example, when groups have different sizes).

The optional argument `params` is a list that allows to control some internal parameters of the optimization algorithm. The elements `beta_start` and `b_start` are initial values for the optimization algorithm. The value of `beta_start` is a vector that indicates the weights of the upper triangular part of B, and `b_start` is the initial value of the threshold in the logistic regression. By default, these parameters are set to zero. The elements `MAX_ITER` and `CONV_CRIT` can be used to change the maximum number of iterations and the convergence criterion in the proximal algorithm for fitting the node selection penalty (see Arroyo-Relion et al. (2017)). By default, these values are set to `MAX_ITER=300` and `CONV_CRIT = 1e-5`.

Value

An object containing the trained graph classifier.

`beta`	Edge coefficients vector of the regularized logistic regression solution.
`b`	Intercept value.
`Yfit`	Fitted logistic regression probabilities in the train data.
`Ypred`	Predicted class for the test samples (if available).
`train_error`	Percentage of train samples that are misclassified.
`test_error`	Percentage of test samples that are misclassified (if available).

References

J.D. Arroyo-Relion, D. Kessler, E. Levina, S.F. Taylor (2017). "Network classification with applications to brain connectomics." https://arxiv.org/pdf/1708.07852.pdf.

@seealso `plot.graphclass`, `predict.graphclass`

Examples

```
# Load COBRE data
data(COBRE.data)
```

```
X <- COBRE.data$X.cobre
Y <- COBRE.data$Y.cobre

# An example of the subgraph selection penalty
gc = graphclass(X = X, Y = factor(Y), type = "intersection",
               lambda = 1e-4, rho = 1, gamma = 1e-5)
plot(gc)


# 5-fold cross validation
fold_index <- (1:length(Y) %% 5) + 1

gclist <- list()
for(fold in 1:5) {
    foldout <- which(fold_index == fold)
    gclist[[fold]] <- graphclass(X = X[-foldout,], Y = factor(Y[-foldout]),
                     Xtest = X[foldout,], Ytest = factor(Y[foldout]),
                     type = "intersection",
                     lambda = 1e-4, rho = 1, gamma = 1e-5,
                     D = D263)
}
# test error on each fold
lapply(gclist, function(gc) gc$test_error)
```

`matrix_to_vec` *Convert matrix to vector.*

Description

The function encodes an adjacency matrix into a vector.

Usage

```
matrix_to_vec(A, type = c("undirected", "directed"))
```

Arguments

`A`	Adjacency matrix of a network.
`type`	Parameter to specify whether the adjacency matrix represents an `undirected` or `directed` network. Default is `undirected`.

Value

A vector containing the upper triangular part of an adjacency matrix (if the graph is undirected) or adjacency entry values ordered by column and excluding the diagonal entries (if the graph is directed).

node_sparsity *Percentage of inactive nodes in a network*

Description

Calculates the node sparsity of a vectorized adjacency matrix, defined as the average number of nodes with no edges.

Usage

```
node_sparsity(beta, type = c("undirected", "directed"))
```

Arguments

beta	Vectorized adjacency matrix.
type	Specifies whether the vector represents an undirected or directed network. Default is undirected.

Value

Percentage of inactive nodes in the graph

Examples

```
A <- matrix(0, ncol = 4, nrow = 4)
A[2, 1] <- 1
A[1, 2] <- 1
A[2, 3] <- 1
A[3, 2] <- 1
vec <- matrix_to_vec(A)
node_sparsity(vec)
```

plot.graphclass *Plot function for graph classifier.*

Description

Plots the coefficients matrix obtained with the function graphclass.

Usage

```
## S3 method for class 'graphclass'
plot(object, ...)
```

Arguments

object	trained graphclass object

Examples

```
data(COBRE.data)
X <- COBRE.data$X.cobre
Y <- COBRE.data$Y.cobre

# An example of the subgraph selection penalty
gc = graphclass(X, Y = factor(Y), lambda = 1e-5, rho = 1)

plot(gc)
```

`plot_adjmatrix` *Plots a vectorized adjacency matrix.*

Description

Draws a plot of the adjacency matrix represented by a vector using the function `levelplot` of the `lattice` package.

Usage

```
plot_adjmatrix(edgevalues, type = c("undirected", "directed"),
  edgetype = c("real", "prob", "binary"), communities = NULL,
  community_labels = NULL, main = "", axislabel = "Nodes")
```

Arguments

`edgevalues`	Edge values of the adjacency matrix. The argument can be either a square adjacency matrix, or a vectorized adjacency matrix. For undirected networks, the vector should contain the upper triangular part in column-major order. For directed networks use both
`type`	If `edgevalues` is a vector, this parameter specifies whether the vector represents an `undirected` or `directed` network. Default is `undirected`.
`edgetype`	Specify the type of edge values. For real-valued edges, `type = "real"`. If the edges are between 0 and 1, `tipe = "prob"`. For binary edges, `type = "binary"`.
`communities`	Optional argument to specify a list in which each element contains an array indicating the indexes of the nodes on each community.
`community_labels`	Labels for each community. The array should have the same length than `communities`.
`main`	Title of the plot
`axislabel`	Label for the axes.

Value

An objected returned by the function `levelplot` of the `lattice` package.

Examples

```
# Plot the adjacency matrix of a COBRE data subject
data(COBRE.data)
X1 <- COBRE.data$X.cobre[1,]

# Plot adjacency matrix with nodes in default order
plot_adjmatrix(X1)

# Plot adj. matrix divided by communities
data(power.parcellation)
# Node assignments (note that node 75 is missing on COBRE)
node.assignments <- power.parcellation$Master.Assignment[-75]
communities = lapply(c(1:13, -1), function(x) which(node.assignments==x))
plot_adjmatrix(X1, communities = communities,
      community_labels = c(1:13, -1), axislabel = "Brain systems")
```

`power.parcellation` *Node assignments of the Power parcellation*

Description

Table containing the node assignments to brain systems of the 264 regions of interest from Power et al. (2011). These nodes were used to construct COBRE.data and UMich.data. Note that node 75 is missing in the COBRE data.

Usage

```
power.parcellation
```

Format

`power.parcellation` is a data frame in which the nodes represent nodes. The data frame has three columns:

Master.Assignment Brain system number

Color Color representing the system.

Suggested.System Suggested brain system.

#' @examples data(power.parcellation)

Source

http://www.nil.wustl.edu/labs/petersen/Resources_files/Consensus264.xls

predict.graphclass *Prediction function for graph classifier*

Description

Given an object of type graphclass, this function obtains the fitted classes for a new data.

Usage

```
## S3 method for class 'graphclass'
predict(object, newdata, type = c("class", "prob",
  "error"), Ytest, ...)
```

Arguments

type	Indicates the type of response: class for class predictions, prob for predicted probabilities, error for misclassification error (if Ytest is provided).
Ytest	If type = "error", true classes to compare.
gc	A trained graph classifier object
X	A matrix containing rows with vectorized upper triangular adjacency matrices (column-major order)

Value

A vector containing the predicted classes.

Examples

```
data(COBRE.data)
X <- COBRE.data$X.cobre
Y <- COBRE.data$Y.cobre

#Split into train and test
test <- runif(length(Y)) <= 0.1
gc <- graphclass(X = X[!test, ], Y = factor(Y[!test]), type = "intersection",
               lambda = 1e-4, rho = 1, gamma = 1e-5)

gc.test <- predict(gc, X[test, ])
predict(gc, X[test, ], "error", Ytest = Y[test])
```

UMich.data *fMRI brain networks of the UMich dataset*

Description

The UMich dataset contains the fMRI brain networks of 78 subjects and class labels of schizophrenia disease status.

Usage

```
UMich.data
```

Format

`UMich.data` is a list with two elements

X.cobre A matrix containing the upper triangular part of networks. Each row represents a subject, and the columns represent edges. The edge weights represent the Fisher-transformed correlation between the fMRI time series of the nodes. Each network has 264 labeled nodes and 34716 different edges. For a description of the preprocessing steps to obtain the network edge weights, see Arroyo-Relion et al. (2017). .

Y.cobre Class labels of the subjects in the dataset. `Y = 1` represents schizophrenia status.

#' @examples data(COBRE.data)

Index