Package ‘MIdecoding’

June 18, 2018

Version 1.0-1

Date 2018-06-04

Title Estimating Mutual Information by Decoding

Depends R (>= 3.3.0)

Suggests randomForest, xgboost

Imports e1071, parallel, abind

Description Implements methods for estimating mutual information by decoding as described in Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkacik, G., and Swain, P. S. (2018) Distributed and dynamic intracellular organization of extracellular information. Proc Natl Acad Sci U S A. <https://dx.doi.org/10.1073/pnas.1716659115>.

License LGPL-3 + file LICENCE

LazyLoad yes

R topics documented:

`as.array.MIdecoding` *Cast Bootstrap Estimates of Mutual Information into an Array*

Description

Extract an array of bootstrap estimates of Mutual Information (MI) organised by bootstrap and hyperparameter from the `data.frame` produced by `MIdecoding`.

Usage

```
## S3 method for class 'MIdecoding'
as.array(x, ...)
```

Arguments

`x`	an object of class `"MIdecoding"` as produced by `MIdecoding`.
`...`	additional arguments passed to or from previous methods.

Value

A numeric array of MI estimates (in bits) where the first dimension specifies bootstrap replicate and where remaining dimensions specify estimates for different classifier hyperparameters. Dimensions are named according to hyperparameter name, and elements along each dimension are named according to hyperparameter value. Missing bootstraps are indicated by `NA` (e.g., as produced by non-zero `crossval` arguments in the call to `MIdecoding`).

Note

This function is designed for use with the full output from `MIdecoding`, i.e., by specifying `opt.par.only=FALSE` in the arguments to `MIdecoding`.

References

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkacik, G., Swain, P. S. (2018) Distributed and dynamic intracellular organization of extracellular information. *Proc Natl Acad Sci U S A*. https://dx.doi.org/10.1073/pnas.1716659115

See Also

`MIdecoding` for generating bootstrap estimates of mutual information by decoding.

Examples

```
## Load sample data
data(YeastStressTypeResponse)

## Obtain and summarise bootstrap estimates of MI by decoding
MIest <- MIdecoding(YeastStressTypeResponse,
                    params=list(ncomponents=1:10,
                                cost=c(0.1, 1, 10, 100)),
                    crossval=1, opt.par.only=FALSE)

## Cast to an array
MIest.array <- as.array(MIest)

## Mean MI obtained for each combination of hyperparameters
apply(MIest.array, 2:length(dim(MIest.array)), mean, na.rm=TRUE)
```

`Info` *Calculate Mutual Information from a Joint Probability Matrix*

Description

Calculate Mutual Information (MI) between, for example, the actual and predicted classes output by a machine learning classifier.

Usage

```
Info(M)
```

Arguments

`M`	a unit-normalised square matrix

Value

The mutual information (in bits) between joint discrete probability distributions.

Note

This function expects a unit-normalised joint probability matrix, so unnormalised confusion matrices will not produce valid MI estimates.

References

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkacik, G., Swain, P. S. (2018) Distributed and dynamic intracellular organization of extracellular information. *Proc Natl Acad Sci U S A*. https://dx.doi.org/10.1073/pnas.1716659115

See Also

`MIdecoding` for generating bootstrap estimates of mutual information by decoding.

Examples

```
## MI between uncorrelated two-state distributions
Info(matrix(rep(0.25, 4), 2)) # 0 bits

## MI between perfectly correlated two-state distributions
Info(matrix(c(0.5, 0, 0, 0.5), 2)) # 1 bit
```

`MIdecoding` *Estimate Mutual Information by Decoding*

Description

Generate bootstrap estimates for a lower bound on the Mutual Information (MI) between discrete states and stochastic time series by evaluating the performance of a classifier function (machine learning algorithm).

Usage

```
MIdecoding(tcdata, Nbootstraps=25, classifier=MIdecodingClassifiers(),
           pca=classifier=="svmlinear", params=NULL,
           normalise=c("bootstrap", "none", "raw"),
           softnorm=TRUE, featurenorm=TRUE,
           crossval=0, n.opt.pars=5,
           Nbootstrap.batch=if(crossval>0) 12 else Nbootstraps,
           silent=FALSE, parallel=TRUE, uniform=TRUE,
           opt.par.only=TRUE, ...)
## S3 method for class 'MIdecoding'
print(x, ...)
```

Arguments

`tcdata`	a list of matrices, the stochastic time series for each state, where each row is a sample and columns are time points. All states must have the same number of time points (columns), but may differ in the number of samples.
`Nbootstraps`	the number of bootstrap estimates to make.
`classifier`	the classifier function to use. Defaults to `"svmlinear"`, a linear SVM classifier using the one-vs-one method to support multi-class classification (i.e., when number of classes is more than two). Other possible choices include `"svmrbf"` for a SVM classifier with the radial basis function kernel, `"rforest"` for the random forest classifier, and `"xgboost"` for the eXtreme Gradient Boosting classifier. For more information see the ‘Details’ section in `MIdecodingClassifiers`.
`pca`	whether or not the data should be transformed onto principal components before application of the classifier. This allows reduction of dimensionality for the SVM classifiers by specification of the `Ncomponents` parameter in `params`. Primarily intended for use with the `"svmlinear"` classifier. Components are determined by performing PCA across all training data.
`params`	a named list of hyperparameter values to try. If `NULL`, then a classifier-specific default list of hyperparameters is used. For more information see the ‘Details’ section of `MIdecodingClassifiers`.
`normalise`	whether and when data should be normalised. `"bootstrap"` (the default) renormalises the data set for each bootstrap; `"none"` turns normalisation off; `"raw"` normalises the entire data set before bootstrapping. Normalisation can improve classifier performance and is applied equally across all classes and across training and testing sets.
`softnorm`	whether normalisation should be hard (such that all data lies between -1 and +1) or soft (subtracting the mean and dividing by two times the standard deviation).
`featurenorm`	whether normalisation should be performed feature-wise (i.e., per time point) or according to the aggregate of all features.
`crossval`	the number of rounds of four-fold cross-validation to run before picking optimal hyperparameters. If `crossval==0`, then cross-validation is not performed and the specified number of bootstraps are obtained for each combination of hyperparameters. Results from cross-validation bootstraps form part of the final output.
`n.opt.pars`	the number of optimal hyperparameter combinations for which the full complement of bootstraps should be obtained. Ignored if cross-validation is not performed.
`Nbootstrap.batch`	the number of bootstraps to calculate in each batch. After each batch, the hyperparameters that are optimal will be re-evaluated.
`silent`	whether to turn off output of progress.
`parallel`	whether to run bootstraps in parallel.
`uniform`	whether to train and test using equal numbers of samples in each class. If `FALSE` (not recommended), the number of samples per class depends on availability, and affects training weights and class priors in estimates of MI from test data.

`opt.par.only`	whether to subset final results to just the bootstraps of the highest-ranking combination of hyperparameters.
`...`	additional arguments to be passed on to the classifier. For more information see 'Details'.
`x`	an object of class `"MIdecoding"`

Details

Given small sample sizes, it is difficult to estimate the Mutual Information (MI) between discrete states and stochastic time series due to the high dimensionality (large configuration space) of the time series. The MI by decoding algorithm attempts to overcome this limitation by performing the estimation in state space. A machine learning classifier (the 'decoder') is trained on a subset of the data to transform time series into state space. By then treating confusion matrices derived from test data as providing joint probabilities over the state-space probability distributions, we can calculate MI. The data processing inequality then guarantees that this is a lower bound to the true MI between the states and time series. As such, the higher the performance of the trained classifier function, the tighter the bound becomes.

In order to find the optimal classifier function, `MIdecoding` provides several alternative machine learning algorithms (described in `MIdecodingClassifiers`) and performs a heuristic search over pre-defined values of the various hyperparameters available to each of the algorithms. When values for multiple hyperparameters are specified in the `params` argument (as they are by default), a grid search over all possible combinations of hyperparameters is performed. If this space is large, or a high number of bootstraps is desired, then it is recommended to specify one or more rounds of cross validation (i.e., `crossval > 0`), in which case, bootstraps at all possible hyperparameter combinations are only obtained in these initial rounds. After that, additional bootstraps are obtained only for the `n.opt.pars` hyperparameter combinations with highest MI. Every `Nbootstrap.batch` bootstraps, the optimal hyperparameters are reevaluated, and bootstraps continue to be obtained until one combination of hyperparameters has at least `Nbootstrap` bootstrap replicates and has mean MI greater than any other combination of hyperparameters.

If either the `"xgboost"` or `"rforest"` classifiers are chosen, then by default they additionally return the importance of each feature (e.g., each time point) in the classifier. If this information is not required, then some processing time can be saved by specifying `featrank=FALSE` as an additional argument (which gets passed on to the classifiers).

Value

An object of class `MIdecoding`, which is a `data.frame` with rows for each bootstrap, columns specifying the hyperparameter values used at the given bootstrap, and columns with:

`MutInf`	the mutual information estimated from this bootstrap,
`confM`	the (normalised) confusion matrix for this bootstrap,
`totalerrors`	the total number of errors made by the classifier across all classes for this bootstrap, and
`featrank`	(for `"xgboost"` and `"rforest"` classifiers only) the importance of each feature.

References

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkacik, G., Swain, P. S. (2018) Distributed and dynamic intracellular organization of extracellular information. *Proc Natl Acad Sci U S A*. https://dx.doi.org/10.1073/pnas.1716659115

See Also

Info for calculating mutual information from a confusion matrix, MIdecodingClassifiers for details on the classifiers that can be used with MIdecoding and their hyperparameters, summary.MIdecoding to obtain reformatted results for just the optimal set of hyperparameters, and as.array.MIdecoding for investigating classifier performance as a function of the hyperparameters.

Examples

```
## Load sample data
data(YeastStressTypeResponse)

## Obtain bootstrap estimates of MI by decoding
MIest <- MIdecoding(YeastStressTypeResponse,
                    params=list(ncomponents=1:10))
MIest

## If the data set has high-order structure that can only be
## resolved with non-linear transformation, then it is more
## appropriate to choose a non-linear classifier (in which case
## pca=FALSE by default):
MIest.rbf <- MIdecoding(YeastStressTypeResponse,
                        classifier='svmrbf',
                        params=list(cost=100, gamma=10^(-4:-1)))
## Compare with linear SVM:
mean(MIest.rbf$MutInf)
mean(MIest$MutInf)

## Perform a wider search of parameter space, but use
## cross-validation to limit the number of bootstraps for
## poorly-performing parameter combinations, and include results
## for all parameters in output:
MIest.scan <- MIdecoding(YeastStressTypeResponse,
                         params=list(ncomponents=1:10,
                                     cost=c(0.1, 1, 10)),
                         crossval=1, opt.par.only=FALSE)
## Display the hyperparameters that were searched over
lapply(attr(MIest.scan, 'params'), unique)
```

MIdecodingClassifiers *Decoding Classifiers Available in the MIdecoding Function*

Description

Outputs the machine learning classifiers available for estimation of Mutual Information (MI) by the MIdecoding function. Optionally outputs the default sets of hyperparameters used for each classifier.

Usage

```
MIdecodingClassifiers(withpars=FALSE)
```

Arguments

withpars	whether to return for each available classifier the parameter values that define its default hyperparameter search space.

Details

The best choice of classifier to use with the MIdecoding function is data-dependent, so it is worth performing preliminary tests with multiple classifiers to see which produces the highest estimated mutual information (MI), and thus a tighter lower bound on the true MI.

For many standard data sets, classification by a linear Support Vector Machine (SVM) on data transformed by Principal Component Analysis (PCA) will be sufficient, and can even outperform more advanced machine learning algorithms if the number of available samples is small. However, if a non-linear transformation would be required to successfully partition the data (e.g., if separation of the classes is in the frequency domain), then it will be necessary to use either SVM with the Radial Basis Function (RBF) kernel or one of the ensemble classifiers (Random Forest or eXtreme Gradent Boosting (XGBoost)).

The classifiers currently available with MIdecoding and their associated hyperparameters are briefly introduced in the following.

Linear SVM: The default classifier in the call to MIdecoding (specified as `classifier="svmlinear"`).

SVM classification (with a linear kernel) using the one-vs-one method to support multi-class classification (i.e., when number of classes is more than two). Implementation is via the svm function of the **e1071** package (which should already be installed since it is a requirement of this package). Further details of the algorithm can be found by consulting the documentation for the svm function.

Available hyperparameters:

`cost` cost of constraints violation.

`ncomponents` (only if `pca=TRUE`) the number of principle components to use as input to the classifier.

Nonlinear SVM: Specified as `classifier="svmrbf"` in the call to MIdecoding.

SVM classification with a RBF kernel using the one-vs-one method to support multi-class classification (i.e., when number of classes is more than two). Implementation is via the svm function of the **e1071** package (which should already be installed since it is a requirement of this package). Further details of the algorithm can be found by consulting the documentation for the svm function.

Available hyperparameters:

`cost` cost of constraints violation.

`gamma` γ parameter of the RBF kernel.

`ncomponents` (only if `pca=TRUE`) the number of principle components to use as input to the classifier.

Random Forest: Specified as `classifier="rforest"` in the call to MIdecoding.

The Random Forest method of ensemble classification, with implementation via the randomForest function of the **randomForest** package (which needs to be installed in order to use this classifier). Further details of the algorithm can be found by consulting the documentation for the randomForest function.

Available hyperparameters:

`ntree` number of trees grown.

`mtry` number of predictors sampled for spliting at each node.

XGBoost: Specified as `classifier="xgboost"` in the call to `MIdecoding`.

The eXtreme Gradient Boosting (XGBoost) method of ensemble classification, with implementation via the `xgboost` function of the **xgboost** package (which needs to be installed in order to use this classifier). Further details of the algorithm can be found by consulting the documentation for the `xgboost` function.

Available hyperparameters:

`nrounds`	maximum number of boosting iterations.
`max.depth`	maximum depth of a tree.

Value

a character vector with the names of all available classifiers, or, if `withpars=TRUE`, a named list of lists of numeric vectors specifying, for each available classifier, the hyperparameter values scanned by default.

References

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkacik, G., Swain, P. S. (2018) Distributed and dynamic intracellular organization of extracellular information. *Proc Natl Acad Sci U S A*. `https://dx.doi.org/10.1073/pnas.1716659115`

See Also

`MIdecoding` for generating bootstrap estimates of mutual information by decoding.

Examples

```
## List default hyperparameter values scanned by each classifier
MIdecodingClassifiers(TRUE)
```

---

summary.MIdecoding *Summarise Bootstrap Estimates of Mutual Information*

Description

Extract and print summary statistics for the bootstrap estimates of Mutual Information (MI) produced by `MIdecoding` at the combination of hyperparameters resulting in the maximum MI.

Usage

```
## S3 method for class 'MIdecoding'
summary(object, ...)
## S3 method for class 'summary.MIdecoding'
print(x, ...)
```

Arguments

`object`	an object of class `"MIdecoding"` as produced by `MIdecoding`.
`x`	an object of class `"summary.MIdecoding"` as produced by `summary.MIdecoding`.
`...`	additional arguments passed to or from previous methods.

Details

The print method displays the mean and standard deviation of the bootstrap MI estimates, the hyperparameter values giving the highest MI, the median number of classification errors made by the chosen classifier, and the mean confusion matrix.

If a named list was used in the call to `MIdecoding`, then the dimensions of the confusion matrix array will be appropriately named by class, and, in any case, each dimension is labelled according to whether it corresponds to the actual class (`"actual"`), the predicted class (`"predicted"`) or to the bootstraps (`"bootstraps"`).

Value

An object of class `summary.MIdecoding`, which is a `list` with elements:

`MutInf`	the bootstrap MI estimates,
`confM`	the (normalised) confusion matrices corresponding to the MI estimates as a 3-dimensional array with bootstraps along the third dimension, and
`totalerrors`	the (absolute) number of errors made by the classifier for each bootstrap.

References

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkacik, G., Swain, P. S. (2018) Distributed and dynamic intracellular organization of extracellular information. *Proc Natl Acad Sci U S A*. `https://dx.doi.org/10.1073/pnas.1716659115`

See Also

`MIdecoding` for generating bootstrap estimates of mutual information by decoding.

Examples

```
## Load sample data
data(YeastStressTypeResponse)

## Obtain and summarise bootstrap estimates of MI by decoding
MIest <- MIdecoding(YeastStressTypeResponse,
                    params=list(ncomponents=1:10),
                    opt.par.only=FALSE)
summary(MIest)

## Calculate the standard deviation of elements in the confusion matrix
MIest.summary <- summary(MIest)
apply(MIest.summary$confM, 1:2, sd)
```

`YeastStressTypeResponse`

Yeast Msn2 Localisation in Response to Stress

Description

Single-cell microscopy data from a microfluidics-based screening of the dynamics of nuclear translocation of Msn2, a transcription factor in *S. cerevisiae* that is the target of evolutionarily conserved signalling pathways including the protein kinase A (PKA) and TOR kinase pathways. The data set includes single-cell time series describing the response for environmental transitions from rich media (2% glucose) into three different stresses: carbon stress (low glucose), osmotic stress or oxidative stress.

Usage

```
data(YeastStressTypeResponse)
```

Format

A list of matrices named according to environmental condition (`"gluc"` for carbon stress, `"nacl"` for osmotic stress, `"oxid"` for oxidative stress, and `"rich"` for no stress), with the rows of each matrix corresponding to independent single-cell time series, and the columns corresponding to 21 observations taken at 2.5 minute intervals.

Note

This experimental data set forms a small subset of the complete data available at `https://dx.doi.org/10.7488/ds/2214`.

Source

`https://dx.doi.org/10.7488/ds/2214`

References

Granados, A. A., Pietsch, J. M. J., Cepeda-Humerez, S. A., Farquhar, I. L., Tkacik, G., Swain, P. S. (2018) Distributed and dynamic intracellular organization of extracellular information. *Proc Natl Acad Sci U S A*. `https://dx.doi.org/10.1073/pnas.1716659115`

Index