Package ‘NlinTS’

September 3, 2020

Type Package

Title Models for Non Linear Causality Detection in Time Series

Version 1.4.3

Date 2020-09-03

Maintainer Youssef Hmamouche <hmamoucheyussef@gmail.com>

Description Models for non-linear time series analysis and causality detection. The main functionalities of this package consist of an implementation of the classical causality test (C.W.J.Granger 1980) <doi:10.1016/0165-1889(80)90069-X>, and a non-linear version of it based on feed-forward neural networks. This package contains also an implementation of the Transfer Entropy <doi:10.1103/PhysRevLett.85.461>, and the continuous Transfer Entropy using an approximation based on the k-nearest neighbors <doi:10.1103/PhysRevE.69.066138>. There are also some other useful tools, like the VARNN (Vector Auto-Regressive Neural Network) prediction model, the Augmented test of stationarity, and the discrete and continuous entropy and mutual information.

License GNU General Public License

Depends Rcpp

Imports methods, timeSeries, Rdpack

RdMacros Rdpack

LinkingTo Rcpp

SystemRequirements C++11

NeedsCompilation yes

RoxygenNote 7.1.0

Author Youssef Hmamouche [aut, cre]

Repository CRAN

Date/Publication 2020-09-03 15:32:10 UTC

R topics documented:

NlinTS-package 2
causality.test 2

df.test 3
entropy_cont 4
entropy_disc 5
mi_cont 5
mi_disc 6
mi_disc_bi 7
nlin_causality.test 8
te_cont 9
te_disc 10
varmlp 11

Index **13**

`NlinTS-package` *Models for non-linear causality detection in time series.*

Description

Globally, this package focuses on non-linear time series analysis, especially on causality detection. To deal with non-linear dependencies between time series, we propose an extension of the Granger causality test using feed-forward neural networks. This package includes also an implementation of the Transfer Entropy, which can be also seen as a causality measure based on information theory. To do that, the package includes discrete and continuous Transfer entropy using the Kraskov approximation. The NlinTS package includes also some other useful tools, like the VARNN (Vector Auto-Regressive Neural Network) model, the Augmented Dickey-Fuller test of stationarity, and the discrete and continuous entropy and mutual information.

`causality.test` *The Granger causality test*

Description

The Granger causality test

Usage

```
causality.test(ts1, ts2, lag, diff = FALSE)
```

Arguments

`ts1`	Numerical dataframe containing one variable.
`ts2`	Numerical dataframe containing one variable.
`lag`	The lag parameter.
`diff`	Logical argument for the option of making data stationary before making the test.

Details

This is the classical Granger test of causality. The null hypothesis is that the second time series does not cause the first one

Value

gci: the Granger causality index.

Ftest: the statistic of the test.

pvalue: the p-value of the test.

summary (): shows the test results.

References

Granger CWJ (1980). "Testing for Causality." *Journal of Economic Dynamics and Control*, **2**, 329–352. ISSN 0165-1889, doi: 10.1016/01651889(80)90069X.

Examples

```
library (timeSeries) # to extract time series
library (NlinTS)
data = LPP2005REC
model = causality.test (data[,1], data[,2], 2)
model$summary ()
```

`df.test` *Augmented Dickey_Fuller test*

Description

Augmented Dickey_Fuller test

Usage

```
df.test(ts, lag)
```

Arguments

`ts`	Numerical dataframe.
`lag`	The lag parameter.

Details

Computes the stationarity test for a given univariate time series.

Value

df: returns the value of the test.

summary (): shows the test results.

References

Elliott G, Rothenberg TJ, Stock JH (1992). "Efficient tests for an autoregressive unit root."

Examples

```
library (timeSeries)
library (NlinTS)
#load data
data = LPP2005REC
model = df.test (data[,1], 1)
model$summary ()
```

---

entropy_cont *Continuous entropy*

Description

Continuous entropy

Usage

```
entropy_cont(V, k = 3, log = "loge")
```

Arguments

V	Interger vector.
k	Integer argument, the number of neighbors.
log	String argument in the set ("log2", "loge","log10"), which indicates the log function to use. The loge is used by default.

Details

Computes the continuous entropy of a numerical vector using the Kozachenko approximation.

References

Kraskov A, Stogbauer H, Grassberger P (2004). "Estimating mutual information." *Phys. Rev. E*, **69**, 066138. doi: 10.1103/PhysRevE.69.066138.

Examples

```
library (timeSeries)
library (NlinTS)
#load data
data = LPP2005REC
print (entropy_cont (data[,1], 3))
```

---

entropy_disc *Discrete Entropy*

Description

Discrete Entropy

Usage

```
entropy_disc(V, log = "log2")
```

Arguments

V	Integer vector.
log	String argument in the set ("log2", "loge","log10"), which indicates the log function to use. The log2 is used by default.

Details

Computes the Shanon entropy of an integer vector.

Examples

```
library (NlinTS)
print (entropy_disc (c(3,2,4,4,3)))
```

---

mi_cont *Continuous Mutual Information*

Description

Continuous Mutual Information

Usage

```
mi_cont(X, Y, k = 3, algo = "ksg1", normalize = FALSE)
```

Arguments

- `X` — Integer vector, first time series.
- `Y` — Integer vector, the second time series.
- `k` — Integer argument, the number of neighbors.
- `algo` — String argument specifies the algorithm use ("ksg1", "ksg2"), as tow propositions of Kraskov estimation are provided. The first one ("ksg1") is used by default.
- `normalize` — Logical argument (FALSE by default) for the option of normalizing the mutual information by dividing it by the joint entropy.

Details

Computes the Mutual Information between two vectors using the Kraskov estimator.

References

Kraskov A, Stogbauer H, Grassberger P (2004). "Estimating mutual information." *Phys. Rev. E*, **69**, 066138. doi: 10.1103/PhysRevE.69.066138.

Examples

```
library (timeSeries)
library (NlinTS)
#load data
data = LPP2005REC
print (mi_cont (data[,1], data[,2], 3, 'ksg1'))
print (mi_cont (data[,1], data[,2], 3, 'ksg2'))
```

---

mi_disc — *Discrete multivariate Mutual Information*

Description

Discrete multivariate Mutual Information

Usage

```
mi_disc(df, log = "log2", normalize = FALSE)
```

Arguments

- `df` — Datafame of type Integer.
- `log` — String argument in the set ("log2", "loge","log10"), which indicates the log function to use. The log2 is used by default.
- `normalize` — Logical argument (FALSE by default) for the option of normalizing the mutual information by dividing it by the joint entropy.

Details

Computes the Mutual Information between columns of a dataframe.

Examples

```
library (NlinTS)
df = data.frame (c(3,2,4,4,3), c(1,4,4,3,3))
mi = mi_disc (df)
print (mi)
```

---

mi_disc_bi *Discrete bivariate Mutual Information*

Description

Discrete bivariate Mutual Information

Usage

```
mi_disc_bi(X, Y, log = "log2", normalize = FALSE)
```

Arguments

`X`	Integer vector.
`Y`	Integer vector.
`log`	String argument in the set ("log2", "loge","log10"), which indicates the log function to use. The log2 is used by default.
`normalize`	Logical argument (FALSE by default) for the option of normalizing the mutual information by dividing it by the joint entropy.

Details

Computes the Mutual Information between two integer vectors.

Examples

```
library (NlinTS)
mi = mi_disc_bi (c(3,2,4,4,3), c(1,4,4,3,3))
print (mi)
```

`nlin_causality.test` *A non linear Granger causality test*

Description

A non linear Granger causality test

Usage

```
nlin_causality.test(
  ts1,
  ts2,
  lag,
  LayersUniv,
  LayersBiv,
  iters = 100,
  learningRate = 0.1,
  algo = "sgd",
  bias = TRUE,
  activationsUniv = vector(),
  activationsBiv = vector()
)
```

Arguments

`ts1`	Numerical series.
`ts2`	Numerical series.
`lag`	The lag parameter
`LayersUniv`	Integer vector that contains the size of hidden layers of the univariate model. The length of this vector is the number of hidden layers, and the i-th element is the number of neurons in the i-th hidden layer.
`LayersBiv`	Integer vector that contains the size of hidden layers of the bivariate model. The length of this vector is the number of hidden layers, and the i-th element is the number of neurons in the i-th hidden layer.
`iters`	The number of iterations.
`learningRate`	The learning rate to use, O.1 by default, and if Adam algorithm is used, then it is the initial learning rate.
`algo`	String argument, for the optimisation algorithm to use, in choice ["sgd", "adam"]. By default "sgd" (stochastic gradient descent) is used. The algorithm 'adam' is to adapt the learning rate while using "sgd".
`bias`	Logical argument for the option of using the bias in the networks.
`activationsUniv`	String vector for the activations functions to use (in choice ["sigmoid", "relu", "tanh"]) for the univariate model. The length of this vector is the number of hidden layers plus one (the output layer). By default, the relu activation function is used in hidden layers, and the sigmoid in the last layer.

activationsBiv String vector for the activations functions to use (in choice ["sigmoid", "relu", "tanh"]) for the bivariate model. The length of this vector is the number of hidden layers plus one (the output layer). By default, the relu activation function is used in hidden layers, and the sigmoid in the last layer.

Details

A non-linear test of causality using artificial neural networks. Two MLP artificial neural networks are evaluated to perform the test, one using just the target time series (ts1), and the second using both time series. The null hypothesis of this test is that the second time series does not cause the first one.

Value

gci: the Granger causality index.

Ftest: the statistic of the test.

pvalue: the p-value of the test.

summary (): shows the test results.

Examples

```
library (timeSeries) # to extract time series
library (NlinTS)
data = LPP2005REC
model = nlin_causality.test (data[,1], data[,2], 2, c(2), c(4),iters=20)
model$summary ()
```

---

te_cont *Continuous Transfer Entropy*

Description

Continuous Transfer Entropy

Usage

```
te_cont(X, Y, p = 1, q = 1, k = 3, normalize = FALSE)
```

Arguments

X	Integer vector, first time series.
Y	Integer vector, the second time series.
p	Integer, the lag parameter to use for the first vector, (p = 1 by default).
q	Integer the lag parameter to use for the first vector, (q = 1 by default).
k	Integer argument, the number of neighbors.

normalize — Logical argument for the option of normalizing value of TE (transfer entropy) (FALSE by default). This normalization is different from the discrete case, because, here the term H (X(t)| X(t-1), ..., X(t-p)) may be negative. Consequently, we use another technique, we divide TE by H0 - H (X(t)| X(t-1), ..., X(t-p), Yt-1), ..., Y(t-q)), where H0 is the max entropy (of uniform distribution).

Details

Computes the continuous Transfer Entropy from the second time series to the first one using the Kraskov estimation

References

Kraskov A, Stogbauer H, Grassberger P (2004). "Estimating mutual information." *Phys. Rev. E*, **69**, 066138. doi: 10.1103/PhysRevE.69.066138.

Examples

```
library (timeSeries)
library (NlinTS)
#load data
data = LPP2005REC
te = te_cont (data[,1], data[,2], 1, 1, 3)
print (te)
```

---

te_disc — *Discrete Transfer Entropy*

Description

Discrete Transfer Entropy

Usage

```
te_disc(X, Y, p = 1, q = 1, log = "log2", normalize = FALSE)
```

Arguments

X	Integer vector, first time series.
Y	Integer vector, the second time series.
p	Integer, the lag parameter to use for the first vector (p = 1 by default).
q	Integer, the lag parameter to use for the first vector (q = 1 by default)..
log	String argument in the set ("log2", "loge","log10"), which indicates the log function to use. The log2 is used by default.
normalize	Logical argument for the option of normalizing the value of TE (transfer entropy) (FALSE by default). This normalization is done by deviding TE by H (X(t)| X(t-1), ..., X(t-p)), where H is the Shanon entropy.

Details

Computes the Transfer Entropy from the second time series to the first one.

References

Schreiber T (2000). "Measuring Information Transfer." *Physical Review Letters*, **85**(2), 461-464. doi: 10.1103/PhysRevLett.85.461.

Examples

```
library (NlinTS)
te = te_disc (c(3,2,4,4,3), c(1,4,4,3,3), 1, 1)
print (te)
```

---

varmlp — *Artificial Neural Network VAR (Vector Auto-Regressive) model using a MultiLayer Perceptron, with the sigmoid activation function. The optimization algorithm is based on the stochastic gradient descent.*

Description

Artificial Neural Network VAR (Vector Auto-Regressive) model using a MultiLayer Perceptron, with the sigmoid activation function. The optimization algorithm is based on the stochastic gradient descent.

Usage

```
varmlp(
  df,
  lag,
  sizeOfHLayers,
  iters = 100,
  bias = TRUE,
  learningRate = 0.1,
  algo = "sgd",
  activations = vector()
)
```

Arguments

`df`	A numerical dataframe
`lag`	The lag parameter.
`sizeOfHLayers`	Integer vector that contains the size of hidden layers, where the length of this vector is the number of hidden layers, and the i-th element is the number of neurons in the i-th hidden layer.
`iters`	The number of iterations.

bias — Logical, true if the bias have to be used in the network.

learningRate — The learning rate to use, O.1 by default, and if Adam algorithm is used, then it is the initial learning rate.

algo — String argument, for the optimisation algorithm to use, in choice ["sgd", "adam"]. By default "sgd" (stochastic gradient descent) is used. The algorithm 'adam' is to adapt the learning rate while using "sgd".

activations — String vector for the activations functions to use (in choice ["sigmoid", "relu", "tanh"]). The length of this vector is the number of hidden layers plus one (the output layer). By default, the relu activation function is used in hidden layers, and the sigmoid in the last layer.

Details

This function builds the model, and returns an object that can be used to make forecasts and can be updated from new data.

Value

train (df): updates the parameters of the model using the input dataframe.

forecast (df): makes forecasts of an given dataframe. The forecasts include the forecasted row based on each previous "lag" rows, where the last one is the next forecasted row of df.

Examples

```
library (timeSeries) # to extract time series
library (NlinTS)
#load data
data = LPP2005REC
# Predict the last row of the data
train_data = data[1:(nrow (data) - 1), ]
model = varmlp (train_data, 1, c(10), 20, bias = TRUE, learningRate=0.1, algo = "sgd");
predictions = model$forecast (train_data)
print (predictions[nrow (predictions),])
```

Index

causality.test, 2

df.test, 3

entropy_cont, 4
entropy_disc, 5

mi_cont, 5
mi_disc, 6
mi_disc_bi, 7

nlin_causality.test, 8
NlinTS (NlinTS-package), 2
NlinTS-package, 2

te_cont, 9
te_disc, 10

varmlp, 11