Package ‘NlinTS’

January 22, 2019

Type Package
Title Non Linear Time Series Analysis
Version 1.3.7
Date 2019-01-22
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 6.0.1
Author Youssef Hmamouche [aut, cre],
Sylvain Barthelemy [cph]
Repository CRAN
Date/Publication 2019-01-22 22:30:03 UTC

R topics documented:

NlinTS-package
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

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

The test evaluates if the second time series causes the first one using the Granger test of causality.

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


Examples

```r
library(tseries) # 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

```r
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


Examples

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

---

entropy_cont  

Continuous entropy

Description

Continuous entropy

Usage

```r
entropy_cont(V, k = 3)
```

Arguments

- `V`  
  Integer vector.
- `k`  
  Integer argument, the number of neighbors.

Details

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

References


Examples

```r
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")

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.
Details

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

References


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")

Arguments

df Dataframe 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.

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**

```r
mi_disc_bi(X, Y, log = "log2")
```

**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.

**Details**

Computes the Mutual Information between two integer vectors.

**Examples**

```r
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**

```r
nlin_causality.test(ts1, ts2, lag, LayersUniv, LayersBiv, iters, bias = TRUE)
```
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.
- **bias**: Logical argument for the option of using the bias in the networks.

Details

The test evaluates if the second time series causes the first one. 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.

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

```r
library(timeSeries) # to extract time series
library(NlinTS)
data = LPP2005REC
# We construct the model based
model = nlin_causality.test(data[,1], data[,2], 2, c(2, 2), c(4, 4), 500, TRUE)
model$summary()
```

---

**te_cont**

*Continuous Transfer Entropy*

Description

Continuous Transfer Entropy

Usage

```r
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), Y(t-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 one using the Kraskov estimation

References


Examples

```r
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

```r
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 dividing TE by \( H(X(t) | X(t-1), ..., X(t-p)) \), where \( H \) is the Shannon entropy.

Details

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

References


Examples

library (NlinTS)

```r
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.*

Description

Artificial Neural Network VAR (Vector Auto-Regressive) model using a MultiLayer Perceptron.

Usage

```r
varmlp(df, lag, sizeofHLayers, iters, bias = TRUE)
```

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.
**Details**

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

**Value**

- `train (df)`: updates the model using the input dataframe.
- `forecast (df)`: returns the next row forecasts of a given dataframe.

**Examples**

```r
library (timeSeries) # to extract time series
library (NlinTS)
#load data
data = LPP2005REC
# Prepare data to make one forecasts
train_data = head (data, nrow (data) - 1)
test_data = tail (data, 1)
model = varmlp (train_data, 1, c(10,5), 200, TRUE)
predictions = model$forecast (train_data)
print (tail (predictions,1))
# Update the model (learning from new data)
model$train (test_data)
```
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, 7
NlinTS (NlinTS-package), 2
NlinTS-package, 2

te_cont, 8
te_disc, 9

varmlp, 10