Package 'tsExpKit'

January 29, 2013

Maintainer Christoph Bergmeir <c.bergmeir@decsai.ugr.es>

License GPL (>= 2)

Title Experimentation Kit for Time Series Analysis.

Type Package

LazyLoad yes

Author Christoph Bergmeir <c.bergmeir@decsai.ugr.es>

Description Implements a lot of tasks common in the analysis of time series predictors, such as embedding and preprocessing, time series simulation, partitioning into training, test, and validation set. Also present are some functions for error calculation and analysis/visualization.

Version 0.1

Depends R (>= 2.10.0), tseriesChaos, zoo, tseries, tools,

Date 2013-01-29

Suggests scatterplot3d, R.matlab, multicore, RSNNS, combinat, TSA,fNonlinear, Rsge, e1071, lars, nnet, snow, tsDyn, class, dtw,forecast, mgcv, pastecs, entropy, wavelets, fractaldim

Encoding UTF-8

Collate 'aTS-methods.R' 'aTS.R' 'atsDirectoryProcessing.R' 'atsList.R''atsListIO.R' 'computePartitionIndices.R' 'dataComplexity methods.R' 'eTS.R' 'experimentRunning.R' 'measureError.R''normalizeData.R' 'parallelProcessing.R' 'postEvaluatio package.R' 'util.R' 'rKeelAtsIO.R''rKeelIO.R'

R topics documented:

`tsExpKit-package` *Getting started with the tsExpKit package*

Description

The tsExpKit package is a package that brings its own time series classes, based on the zoo classes, which are similar to the `xts` class. It has functions to apply preprocessing such as partitioning for training and testing, normalization, and embedding, as well as functions to build time series repositories, containing various time series packages, and to store characteristics of the series in the repository. Furthermore, it also has abilities to run well-documented experiments, in sequential or parallelized ways.

Details

The package is mainly used internally in our department for experimentation. However, as it may be useful also to other people we decided to release this version publicly on CRAN. Some parts are still work in progress, and some (small) parts may not make much sense without our infrastructure. If you have any problems or suggestions, please contact the package maintainer directly by email. Don't write to the R mailing lists or other internet forums, as it is probable that we won't read your comments there.

To get started with the package, you may want to have a look at the demo's. Don't start them directly, but access them in the package source in the demo subfolder. You may need to adjust some paths etc. in order to get the demo's running.

The attributed time series class `aTS` is basically a zoo object with its own mechanism for attributes, similar to `xts`. The reason for the custom implementation is that `xts` is very strict on the time

stamps it allows, and it turned out that in many situations less strict time stamps, e.g., when having artificial time series or when performing preliminary experiments, are desirable.

Based on the aTS class is its subclass eTS, which is the embedded time series class. You can construct an eTS object from an aTS object simply by defining the embedding to use. Both the aTS and the eTS classes have possibilities to perform normalization on them, to generate training and test partitions from them, etc. They also have a lot of convenience functions to get and set information the attributes that they store.

Based on these classes is the concept of the ATSList, which is the basis for the time series repository structure. It is simply a list of aTS objects. However, there exist many functions for batch processing that work directly on ATSLists, and to work with several series instead of single series. Namely, there are the functions setNamesATSList, summaryTableATSList, computePartitionsATSList, groupATSListByPartitions, and embedATSList. Furthermore, for I/O, there are: loadDifToATSList, loadDatDirToATSList, loadDataDirToATSList, loadATSDir, loadATSDir, saveATSList, savePlotsATSList, and saveAsDataPackageATSList.

Using the ATSList, there are functions that work directly on aTS directories, which are ATSLists exported to disk. Currently, there are the functions embedATSDir, partitionATSDir, and normalizeATSDir to work directly on aTS directories.

A time series data package is a hierarchical structure of ATS directories. Here, a data package, i.e., a package of time series, is stored in different versions (preprocessing). Besides the ATS directory functions mentioned above, there is the function saveAsDataPackageATSList, which saves an ATSList as an ATS directory, and generates summary tables and summary plots. Furthermore, there is the function importATSListIntoDataRepository, which is recommended when first importing data into a data repository. The summary tables generated by these functions contain characteristics of the time series, such as descriptive statistical measures, results of stationarity and linearity tests, and data complexity measures. In concrete, the import functions call one-line-summary functions, i.e., they call oneLineSummary which calls oneLineSummaryAll. This function then calls the specific functions for the different kinds of summaries. Currently implemented are oneLineSummaryDescStat, oneLineSummaryOtherTests, and oneLineSummaryStatNonlin. The data complexity measures should be implemented here. Either, by creating a new one-line-summary function so that an additional table is generated, or by simply adding them to the existing tables.

Yet another important part of the package is the experimentation framework around the function runExperiments. The functions genUsedData, initParallelMode, and assembleExperimentsPath assist in setting up the experiments, and afterwards, the functions evaluateModels, evaluatePredictions, plotPredictions perform the evaluation by generating various csv-files. Afterwards, in order to assist in the analysis of these csv-files, there is currently the function assembleMeasureTable, but probably somewhen there will be more functionality assisting in this also.

Also, there are some functions present to artificially generate time series with certain characteristics, see simulateLinearTS, and simulateNonlinearTS.

A lot of helpful utility functions that solve unspecific common tasks as well as tasks that often have to be solved in relation to the use of the package are present. Examples are: joinCSVFiles, rBindByColNames, cBindByRowNames, multiplyIntervalBoundaries, getFileNameFromPath, etc.

Author(s)

Christoph Bergmeir <c.bergmeir@decsai.ugr.es>

with José M. Benítez <j.m.benitez@decsai.ugr.es>

DiCITS Lab, Sci2s group, DECSAI, University of Granada.

http://dicits.ugr.es, http://sci2s.ugr.es

References

C. Bergmeir, J.M. Benítez, On the Use of Cross-validation for Time Series Predictor Evaluation. Information Sciences 191 (2012) 192-213.

`assembleExperimentsPath` *Assemble a path name for an experiment run*

Description

This function helps you giving your experiment runs meaningful names. It composes a name from the current data, the name of the machine the experiments were run on, your user name, and a given comment

Usage

```
assembleExperimentsPath(resultsComment,
  resultsPath = "/home/ts/results/")
```

Arguments

`resultsComment`	a string that will be part of the directory name, naming the experiments
`resultsPath`	the path where to generate the subdirectory for the results

Value

a string which is a path of to the directory to be created for the experiments

---

`assembleMeasureTable` *Create a table for evaluation of various experiment runs*

Description

This function can be used to unite evaluations of different experiment runs, both accross datasets and methods, for later evaluation.

Usage

```
assembleMeasureTable(resultCollection, measure,
  na.rm = TRUE)
```

Arguments

`resultCollection`	a list containing the paths to the different experiment runs
`measure`	the filename of the measure, e.g. `"predEval_RMSE.csv"`.
`na.rm`	if `TRUE`, cases where NAs are present are removed

Details

TODO: Currently no examples available. Consult with the maintainer if you can't figure out how to use this function.

assemblePathName *Utility function to assemble a path name from two path strings*

Description

OBSOLETE, use system function `file.path` instead.

Usage

```
assemblePathName(path1, path2)
```

Arguments

path1	string that represents first path
path2	string that represents second path

Details

Assembles two path names, e.g. a path and a filename. Automatically determines, if a backslash is needed.

Value

the assembled path name

aTS *Constructor for objects of the attributed time series class*

Description

The attributed time series class (`aTS`) extends the `zooreg` class from the zoo package. It implements its own mechanism for attributes, similar to xts, see `atsAttributes`. Furthermore, there are the following convenience methods for getting and setting some special attributes directly: `getName`, `setName`, `getComment`, `setComment`. Also, there is a method `getTimeStamps`, which in directly returns the result of the `index` function from `zoo`. The method `nSamples` returns the number of samples, i.e., the number of rows of the data matrix. Furthermore, there is the index operator `[]` defined on the class.

Usage

```
aTS(x, ..., name = deparse(substitute(x)),
  comment = NULL)

getTimeStamps(obj, ...)

## S3 method for class 'aTS'
getTimeStamps(obj, ...)

aTS(x, ..., name=deparse(substitute(x)), comment=NULL)
```

```
 getName(obj, ...)

 ## S3 method for class 'aTS'
getName(obj, ...)

 setName(obj, ...)

 ## S3 method for class 'aTS'
setName(obj, name, ...)

 getComment(obj, ...)

 ## S3 method for class 'aTS'
getComment(obj, ...)

 setComment(obj, ...)

 ## S3 method for class 'aTS'
setComment(obj, comment, ...)

 nSamples(obj, ...)

 ## S3 method for class 'aTS'
nSamples(obj, ...)
```

Arguments

x	data to construct the time series from. This is directly passed to as.zooreg from the zoo package.
...	parameters passed to the as.zooreg function
name	a name for the time series
comment	a commentary describing the contents of the series
obj	the aTS object

Value

a new aTS object

Examples

```
ats <- aTS(lynx)

getTimeStamps(ats)
nSamples(ats)


aTSattr(ats, "test") <- 5
atsAttributes(ats)
aTSattr(ats, "test")
```

aTSattr *The aTS attribute mechanism*

Description

The attributes mechanism of the aTS class works as follows: All attributes are stored in a conventional R attribute named aTSattr, which is a list. To get all attributes, i.e., this list, call atsAttributes on the time series. You can replace the whole list using the atsAttributes<- method. To get and set single attributes, use the methods aTSattr, and aTSattr<-.

Set all ats attributes

Usage

```
aTSattr(aTS, name)

aTSattr(aTS, name) <- value

aTSattr(aTS, name) <- value

atsAttributes(aTS)

atsAttributes(aTS) <- value

atsAttributes(aTS) <- value
```

Arguments

aTS	the aTS object
name,	the name of the attribute
value	the value of the atsAttribute

Value

the value of the attribute

Examples

```
ats <- aTS(lynx)

getTimeStamps(ats)
nSamples(ats)


aTSattr(ats, "test") <- 5
atsAttributes(ats)
aTSattr(ats, "test")
```

ATSList *A list of aTS objects*

Description

An ATSList is simply a `list` of `aTS` objects. However, there are a lot of useful functions that can be used together with such a list, namely `setNamesATSList`, `summaryTableATSList`, `computePartitionsATSList`, `groupATSListByPartitions`, and `embedATSList`. Furthermore, for I/O, there are: `loadDifToATSList`, `loadDatDirToATSList`, `loadDataDirToATSList`, `loadATSDir`, `loadATSDir`, `saveATSList`, `savePlotsATSList`, and `saveAsDataPackageATSList`.

cBindByRowNames *Column-bind together two tables, taking into account the row names*

Description

The function is the pendant to `rBindByColNames`. See there for more detailed explanations.

Usage

```
cBindByRowNames(x, y)
```

Arguments

x	first table
y	second table

Value

the two tables cbind-ed together

See Also

`rBindByColNames`

computeIndicesLastBlockEval *Function to split data into training and test set for last block evaluation.*

Description

This function is very similar to `computeTrainingAndTestIndices`, but instead of a vector of indices, it takes a length and an embedding order.

Usage

```
computeIndicesLastBlockEval(len, order, ratio = 0.15)
```

Arguments

len	the length of the time series to be used
order	the order of the embedding to be used
ratio	the ratio of training and test set

Value

a named list with the following elements:

trainIndices	a matrix containing the training indices
testIndices	a matrix containing the test indices

---

computeOrder *Compute the order from a lags vector*

Description

This function computes simply `max(lags)-min(lags)`.

Usage

```
computeOrder(lags)
```

Arguments

lags	a vector containing the lags, e.g. `lags=-4:0`.

---

computePartitionIndices *Compute the indices for the partitioning scheme*

Description

This function computes the indices that can later be used to partition the time series into training and test sets, for different validation schemes. Currently supported are: "lastBlock", "CV", "noDepCV", "blockedCV".

Usage

```
computePartitionIndices(len, type = "lastBlock",
  order = 0, ratioLB = 0.15, numPartitions = 0,
  ratioValSet = 0, seed = 1, notEmbedded = FALSE)
```

Arguments

len	the length of the target series
type	one of c("lastBlock", "CV", "noDepCV", "blockedCV")
order	the embedding dimension, if the indices are computed for a series that is not embedded. If the indices are used with an embedded series, this has to be set to zero.
ratioLB	if between zero and one, defines the ratio of data that are used as test set. If greater one, defines the absolute value of instances in the test set.
numPartitions	the numer of partitions to generate, for the cross-validation schemes
ratioValSet	if an additional validation set is generated, its ratio (zero to not generate additional validation set)
seed	the seed to use for randomly partitioning the data during cross-validation
notEmbedded	if this flag is true, the indices are shifted by "order" into the future, so that the results are the indices in the not embedded series, of the values which are the target values in the embedded series.

---

computePartitions *Compute partitions of a time series*

Description

Compute partitions of a time series

This function calls computePartitionIndices to calculate the partition indices, and then finds the values to the partition indices.

Usage

```
computePartitions(obj, ...)

## S3 method for class 'aTS'
computePartitions(obj, ...)

## S3 method for class 'eTS'
computePartitions(obj, order, ...)
```

Arguments

obj	the aTS object
...	function parameters passed to computePartitionIndices
order	For eTS objects: If the order is missing, it is extracted for the attributes of the series.

Value

the partitioned time series

computePartitionsATSList *Compute partitions for an* ATSList

Description

This method calls for every aTS object in the ATSList the function computePartitions.

Usage

```
computePartitionsATSList(atsList, ...)
```

Arguments

atsList	the ATSList to which to apply the function to
...	parameters passed to computePartitions

computeTrainingAndTestIndices *Function to split data indices into indices for training and test set*

Description

This function is very similar to computeTrainingAndTestIndices. It splits the given indices to indices for a training and a test set. Test set is taken from the end of the data. If the data is to be shuffled, this should be done before calling this function.

Usage

```
computeTrainingAndTestIndices(indices, ratio = 0.15)
```

Arguments

indices	the indices to be used
ratio	ratio of training and test sets (default: 15% of the data is used for testing)

Value

a named list with the following elements:

trainIndices	a matrix containing the training indices
testIndices	a matrix containing the test indices

denormalizeData *Revert data normalization.*

Description

Column-wise normalization of the input matrix is reverted.

Usage

```
denormalizeData(x, normParams)
```

Arguments

x	input data
normParams	the parameters generated by an earlier call to normalizeData that will be used for reverting normalization

Value

column-wise reverse-normalized input

embed.eTS *Embed a time series.*

Description

Generate a zoo object that contains a matrix with lagged versions of the time series. This function is copied from tseriesChaos and adapted to work for zooreg objects.

Usage

```
embed.eTS(x, m, d, lags)
```

Arguments

x	the times series
m	embedding parameter (same as in tseriesChaos)
d	embedding parameter (same as in tseriesChaos)
lags	embedding parameter (same as in tseriesChaos)

embedATSDir *Embed all time series given in an ATS directory*

Description

The function loads all files in an ATS directory into an ATSList, applies embedATSList, and saves the result to an ATS directory.

Usage

```
embedATSDir(inPath,
  embeddingParameters = list(lags = c(-4, -3, -2, -1, 0)),
  outPath = NULL)
```

Arguments

inPath	the input path, i.e., the ATS directory
embeddingParameters	a list giving all parameters that are passed to embedATSList
outPath	the output path. If NULL, it is generated automatically as a subdirectory of the inPath

See Also

partitionATSDir, normalizeATSDir

embedATSList *Embed an ATSList*

Description

This method constructs from every aTS object in the list an eTS object.

Usage

```
embedATSList(atsList, ...)
```

Arguments

atsList	the ATSList to which to apply the function to
...	parameters passed to the eTS constructor

endsWith *Utility function to find ending of a string*

Description

Determines, of the string myString ends with the string mySubString. The function is useful, e.g., when determining the file ending of a filename.

Usage

```
endsWith(myString, mySubString)
```

Arguments

myString	string to determine ending
mySubString	string to compare to

Value

TRUE, if myString ends with mySubString, FALSE otherwise

eTS *Constructor for objects of the embedded time series class*

Description

The eTS time series class can hold a time series in its embedded form and provides methods for extracting and information from it and for manipulating it.

Usage

```
eTS(x, lags, name, targets)

getDims(obj, ...)

getOrder(obj, ...)

getLags(obj, ...)

getnValues(obj, ...)

getSelTargets(obj, ...)

## S3 method for class 'eTS'
getDims(obj, ...)

## S3 method for class 'eTS'
getOrder(obj, ...)
```

```
 ## S3 method for class 'eTS'
getLags(obj, ...)

 ## S3 method for class 'eTS'
getnValues(obj, ...)

 ## S3 method for class 'eTS'
getSelTargets(obj, ...)

 getInvValues(obj, ...)

 ## S3 method for class 'eTS'
getInvValues(obj, ...)

 getValues(obj, ...)

 ## S3 method for class 'eTS'
getValues(obj, ...)

 getTargets(obj, ...)

 ## S3 method for class 'eTS'
getTargets(obj, ...)

 getNaiveForecast(obj, ...)

 ## S3 method for class 'eTS'
getNaiveForecast(obj, ...)
```

Arguments

x	data to construct the time series from. If this is not an aTS object, then the data is directly passed to as.zooreg from the zoo package.
lags	a vector containing all the lags, e.g. lags=c(-4:0)
name	is only used, if input is not an aTS object
targets	columns from the targets that are to be selected
obj	the eTS object
...	additional parameters. Currently not used by any of the methods.

`evaluateModels` *Function to evaluate models built during an experiment run*

Description

This function is used to generate from a bunch of models saved as ".RData" files evaluations in form of tables which contain all training/fitting errors of the models and, depending on the models, some other tables with model characteristics.

Usage

```
evaluateModels(qualifiedModelFileNames,
  calcDenorm = TRUE)
```

Arguments

qualifiedModelFileNames	a list of file names with paths to all the model files to use
calcDenorm	if TRUE, all error measures are also calculated for denormalized model fits, along with the measures for normalized fits that are always calculated

---

evaluatePredictions *Function to evaluate predictions*

Description

This function is used to generate from predictions and the respective "real" reference values (determined by usedData) evaluations in form of tables which contain all errors of these predictions (i.e., errors on the test set).

Usage

```
evaluatePredictions(qualifiedPredictionFileNames,
  usedData, denorm = FALSE)
```

Arguments

qualifiedPredictionFileNames	a list of file names with paths to all the prediction files to use
usedData	the usedData structure which defines the input files. From here, the reference and benchmark values are extracted.
denorm	if TRUE, all error measures are also calculated using denormalized predictions and denormalized reference values and benchmarks.

---

exportToKEELDatasetHomepageDir *Function to export from a time series repository to a format that is used on the department homepage*

Description

Function to export from a time series repository to a format that is used on the department homepage

Usage

```
exportToKEELDatasetHomepageDir(dataPackagePath,
  dataPackageName, wholeEmbeddedSeriesPath,
  partitionsPath, partSuffix = "5-fold",
  targetPath = NULL)
```

Arguments

dataPackagePath	path of the data package
dataPackageName	name of the data package
wholeEmbeddedSeriesPath	path of the embedded series (unpartitioned)
partitionsPath	path of the partitioned series
partSuffix	a string to append to the filenames
targetPath	the output path

---

extractNameFromPath *Get the filename without file ending from a path*

Description

This function calls subsequently removeFileEnding, and getFileNameFromPath

Usage

```
extractNameFromPath(path, suffix)
```

Arguments

path	a string
suffix	a substring to remove from the end of path

Value

a substring of path

---

firstColToColname *Use the first column of a matrix to set its row names*

Description

TODO: This function should be renamed to firstColToRowNames

Usage

```
firstColToColname(x)
```

Arguments

x	a matrix

Value

the matrix, but with the first column removed and set to rownames

genUsedData *Function to generate the* usedData *structure*

Description

Function to generate the usedData structure, which determines the data to be used during the experimentation.

Usage

```
genUsedData(useCases, onlyUseStationarySeries = TRUE,
  removeDuplicatesByDescStatTable = TRUE,
  useSelectionFile = FALSE, selectionFile = "")
```

Arguments

useCases	the use cases to add
onlyUseStationarySeries	should the files be filtered for stationary series?
removeDuplicatesByDescStatTable	should duplicate series be determined by statistical properties and be removed?
useSelectionFile	should a selection file be used to determine, which series actually to use?
selectionFile	if useSelectionFile is TRUE, here the filename of the selection file has to be given

getConfigDefault *Get the default settings of parallelization config*

Description

This function returns default settings for parallelization, to be used with initParallelMode. The default is no parallelization.

Usage

```
getConfigDefault()
```

Value

the default settings

See Also

initParallelMode, getConfigHercules

getConfigHercules *Get the settings of parallelization for hercules*

Description

This function returns the current settings for parallelization for "hercules", our SGE cluster here at DiCITS. This function is intended to be used with `initParallelMode`.

Usage

```
getConfigHercules()
```

Value

the parallelization settings for hercules

See Also

`initParallelMode`, `getConfigDefault`

getFileNameFromPath *Get the filename from a full path*

Description

This function splits the input string along slashes ("/") and returns everything after the last slash it finds.

Usage

```
getFileNameFromPath(path)
```

Arguments

`path`	a string

Value

returns a substring of `path`

getFilenameWithoutDatEnding *Remove the string ".dat" or ".DAT" from a filename*

Description

Remove the string ".dat" or ".DAT" from a filename

Usage

```
getFilenameWithoutDatEnding(filename)
```

Arguments

filename	a string

Value

the filename string, with ".dat" or ".DAT" removed from the end.

getNormParams *Get the normalization parameters of a time series*

Description

This function is a getter function for the normParams atsAttribute.

Usage

```
getNormParams(obj, ...)

## S3 method for class 'aTS'
getNormParams(obj, ...)
```

Arguments

obj	the aTS object
...	additional function parameters (currently not used)

groupATSListByPartitions

Group an ATSList by its partitions

Description

This method sorts the ATSList according to its partitions. It uses the "tsname" attribute of the aTS objects for that.

Usage

```
groupATSListByPartitions(atsList)
```

Arguments

atsList	the ATSList to which to apply the function to

Details

TODO: Where is this used?

Value

a list with two hierarchies

importATSListIntoDataRepository

Function to initially import an ATSList into a time series repository

Description

Function to initially import an ATSList into a time series repository

Usage

```
importATSListIntoDataRepository(atsList, dataPath,
  dataPackageName, partSuffix = "5-fold",
  embeddingParameters = list(lags = c(-4, -3, -2, -1, 0)),
  partitionParametersKeelExp = list(type = c("blockedCV"), order = 0, numPartitions = 5, ratio
  partitionParametersNorm = list(type = c("lastBlock"), order = 0, ratioLB = 0.2, ratioValSet
  normParameters = list(), genDescStatTab = TRUE,
  genStatNonlinTab = TRUE, genStatOtherTestsTab = FALSE,
  minLength = 10)
```

Arguments

atsList	the time series list
dataPath	path of the data package
dataPackageName	name of the data package
partSuffix	a string to append to the filenames
embeddingParameters	parameters for the embedding
partitionParametersKeelExp	parameters for partitioning in the export to KEEL
partitionParametersNorm	parameters for partitioning in the versions of the series that are normalized
normParameters	parameters of the normalization
genDescStatTab	Should the descriptive statistics table be generated?
genStatNonlinTab	Should the table of the results of the linearity tests be generated? (This may be slow)
genStatOtherTestsTab	Should the table with the other tests be generated? (This also may be slow)
minLength	minimal length of the series to be considered

---

initParallelMode *This function determines and initializes automatically the adequate apply function for parallel processing*

Description

This function can be used to initialize and/or auto-determine the apply function to be used, in order to run parallel executions. The function checks the hostname of the machine it is running on, and then chooses from the config the apropriate apply function, and initializes it if necessary. E.g., if running on hercules, the apply function to be used is one from the `Rsge` package, to send parallel jobs to hercules' SGE queue. If run on a multicore linux system, a version of apply from the `multicore` package is used. For debugging, it is often necessary to turn off parallel execution.

Usage

```
initParallelMode(config, packagesToLoad = c("tsExpKit"),
  sgeQueueName = "muylarga", sgeRemoveFiles = TRUE)
```

Arguments

config	the known configurations. See `getConfigDefault` and `getConfigHercules` for examples.
packagesToLoad	the packages that have to be loaded for versions of applies that launch new R processes.
sgeQueueName	for SGE: the name of the queue to use.
sgeRemoveFiles	for SGE: should the temporary files be deleted?

Details

TODO: Parallel execution using the snow package is implemented but probably currently not working.

`joinCSVFiles` *Join all ".csv" files in a directory into one file*

Description

This function finds all ".csv" files in a directory and joins them into one big file, using `rbind`.

Usage

```
joinCSVFiles(path, pattern="\\.csv", targetFilename)
```

Arguments

`path`	the directory where the ".csv" files are
`pattern`	the file ending. This will usually be "\.csv"
`targetFilename`	the target file which contains all files joined together.

`linearityTest` *LM linearity testing against 2 regime genericStar. Performs a 3rd order Taylor expansion LM test.*

Description

TODO: This function is from tsDyn

Usage

```
linearityTest(obj, ...)

## S3 method for class 'eTS'
linearityTest(obj, rob = FALSE,
   sig = 0.95, trace = TRUE, ...)
```

Arguments

`obj`	the time series object
`...`	additional arguments (not used)
`rob`	TODO: What is this parameter?
`sig`	the significance level
`trace`	generate debug output?

list.dirs *find all subdirectories in a directory*

Description

This function calls `list.files`, and filters the result for directories

Usage

```
list.dirs(path = ".", pattern = NULL, all.dirs = FALSE,
  ignore.case = FALSE, recursive = FALSE)
```

Arguments

`path`	the path to explore
`pattern`	passed to `list.files`
`all.dirs`	passed to `list.files`
`ignore.case`	passed to `list.files`
`recursive`	explore the directory recursively, passed to `list.files`

Author(s)

The code is taken from a post on stackoverflow:

http://stackoverflow.com/questions/4749783/how-to-obtain-a-list-of-directories-within-a-directory-like-list-files-but-in

loadATSDir *Load an ATS directory into an* `ATSList`

Description

This function loads an ATS directory into an `ATSList`.

Usage

```
loadATSDir(path, pattern="+\\.RData", ...)
```

Arguments

`path`	the path of the directory
`pattern`	a pattern, to e.g. choose the file ending of the files
`...`	currently not used

Value

an `ATSList` containing all the time series that were read

loadDataDirToATSList *Load a directory of files into an* ATSList

Description

This function compiles a list of filenames according to `path` and `pattern`, and then uses the `fileReadFunction` to read the contents of the files. Then, `aTS` objects are constructed from the file contents and added to an `ATSList`.

Usage

```
loadDataDirToATSList(path, pattern="+\\.dat",
  fileReadFunction=scan)
```

Arguments

`path`	the path of the directory
`pattern`	a pattern, to e.g. choose the file ending of the files
`fileReadFunction`	the function that is to be used to read single files

Value

an `ATSList` containing all the time series that were read

loadDatDirToATSList *Load a directory of ".dat" files into an* ATSList

Description

This function calls `loadDataDirToATSList` with the parameters `pattern="+\.dat"` and `fileReadFunction=scan`.

Usage

```
loadDatDirToATSList(path)
```

Arguments

`path`	the path of the directory

`loadDifToATSList` *Load xls files into an aTS list*

Description

Use OpenOffice to generate a dif file (without any header, ASCII encoding). Then, use this function to load it to an ATSList

Usage

```
loadDifToATSList(path, base)
```

Arguments

`path`	the path to use
`base`	the filename (without extension)

Details

TODO: Before using this function, it should be revised and updated (e.g. to use `assemblePathName`).

Value

an `ATSList`

`loadKeelDatDirToATSList`
Load a directory of KEEL dat files into an ATSList

Description

Load a directory of KEEL dat files into an ATSList

Usage

```
loadKeelDatDirToATSList(path)
```

Arguments

`path`	the path of the directory with the dat files

loadKeelFile *Rudimentary parser for KEEL files.*

Description

Comments are only allowed in the header

Usage

```
loadKeelFile(filename, noColumns = 1)
```

Arguments

filename	the name of the KEEL inputfile
noColumns	the number of columns

loadMatToATSList *Load mat files into an aTS list*

Description

Load mat files into an aTS list

Usage

```
loadMatToATSList(filename, filenameIndices = NULL)
```

Arguments

filename	the file to read in
filenameIndices	of the positive instances

Value

an ATSList

Author(s)

Mabel

measureError — *Calculate an error measure for a predictions*

Description

This function can be used to calculate a lot of different error measures for predictions.

Usage

```
measureError(measure, prediction, reference, benchmark)
```

Arguments

measure	currently implementad are the error measures: "MSE", "RMSE", "SSE", "MAE", "MdAE", "MAPE", "MdAPE", "sMAPE", "sMdAPE", "MRAE", "MdRAE", "GMRAE", "RelMAE", "RelRMSE", "LMR", "PB", "PBMAE", "PBMSE", "DF", "DF_unnorm", "DF_percent", "DF_sig"
prediction	a vector containing the predictions
reference	a vector containing the "true" reference values
benchmark	if needed by the error measure, a benchmark forecast, such as the naive forecast (always take the last known value as forecast).

measureErrorAll — *This function calculates all error measures that are available*

Description

This function calls measureError for all error measures available.

Usage

```
measureErrorAll(prediction, reference, benchmark)
```

Arguments

prediction	a vector containing the predictions
reference	a vector containing the "true" reference values
benchmark	if needed by the error measure, a benchmark forecast, such as the naive forecast (always take the last known value as forecast).

Value

a data.frame containing all the measures

multiplyIntervalBoundaries *Determine new boundaries of a multiplied interval*

Description

This function is a helper function to solve the following problem: You have two variables that you multiply, and you know for each variable the interval of values it can take. Then, this function calculates the interval of values the product can take.

Usage

```
multiplyIntervalBoundaries(minX, maxX, minY, maxY)
```

Arguments

minX	the lower bound of the first interval
maxX	the upper bound of the first interval
minY	the lower bound of the second interval
maxY	the upper bound of the second interval

Examples

```
multiplyIntervalBoundaries(0, 25, -2, 4.5)
```

normalize *normalize an aTS object*

Description

This function calls the function normalize on the coredata of the aTS object

Usage

```
normalize(obj, ...)

## S3 method for class 'aTS'
normalize(obj, type = "norm",
  normParams = type, ...)
```

Arguments

obj	the aTS object
...	additional function parameters (currently not used)
type	the type of the normalization
normParams	normalization parameters to be used. When omitted, the type is used and the parameters are calculated from the data.

normalizeATSDir *Normalize all time series in an ATS directory*

Description

The function loads all files in an ATS directory into an ATSList, applies normalize, and saves the result to an ATS directory. If the ATS directory contains training and test sets, the test sets are normalized using the parameters of the corresponding training sets.

Usage

```
normalizeATSDir(inPath, normParameters = list(),
  outPath = NULL)
```

Arguments

inPath	the input path, i.e., the ATS directory
normParameters	a list giving all parameters that are passed to normalize
outPath	the output path. If NULL, it is generated automatically as a subdirectory of the inPath

See Also

partitionATSDir, embedATSDir

normalizeData *Data normalization.*

Description

The input matrix is column-wise normalized. The parameter type specifies how:

0_1 values are normalized to the [0,1]-interval. The minimum in the data is mapped to zero, the maximum to one.

center the data is centered, i.e. the mean is substracted

norm the data is normalized to mean zero, variance one

Usage

```
normalizeData(x, type = "norm")
```

Arguments

x	input data
type	**either** type string specifying the type of normalization. Implemented are "0_1", "center", and "norm"
	or attribute list of a former call to this method to apply e.g. normalization of the training data to the test data

Value

column-wise normalized input

`oneLineSummary` *Generate a "one-line-summary" for an aTS object*

Description

This function calls `oneLineSummaryAll` on the `coredata` of the `aTS` object.

Usage

```
oneLineSummary(aTS, ...)
```

Arguments

`aTS`	the `aTS` object
`...`	parameters passed to `oneLineSummaryAll`

`oneLineSummaryAll` *Generate a "one-line-summary" for a time series*

Description

This function calls, depending on the parameter `type`, one of the following functions, and returns their result: `oneLineSummaryStatNonlin`, `oneLineSummaryOtherTests`, or `oneLineSummaryDescStat`.

Usage

```
oneLineSummaryAll(ts, name, type=c("StatNonlin",
  "DescStat", "OtherTests"), ...)
```

Arguments

`ts`	coredata of a time series
`name`	the name of the time series
`type`	the type of the "one-line-summary" to calculate. Currently implemented are `"StatNonlin"`, `"DescStat"`, and `"OtherTests"`
`...`	parameters passed to the respective "one-line-summary" function

oneLineSummaryDescStat

Generate a "one-line-summary" of type "DescStat"

Description

This function computes a bunch of descriptive statistics on the time series, such as the mean, the standard deviation, the kurtosis, the skewness, and different versions of permutation entropy.

Usage

```
oneLineSummaryDescStat(ts, name, order = 1)
```

Arguments

ts	coredata of a time series
name	the name of the time series
order	the order to be assumed by tests which need it

oneLineSummaryOtherTests

Generate a "one-line-summary" of type "OtherTests"

Description

This function computes a bunch of measures of the time series, like a runs test, the Bds test, the McLeod.Li.test, etc.

Usage

```
oneLineSummaryOtherTests(ts, name, order = 1,
  alpha = 0.01)
```

Arguments

ts	coredata of a time series
name	the name of the time series
order	the order to be assumed by tests which need it
alpha	significance level for the tests

oneLineSummaryStatNonlin *Generate a "one-line-summary" of type "StatNonlin"*

Description

This function computes a bunch of measures of the time series, mainly statistical tests on the stationarity and linearity of the series.

Usage

```
oneLineSummaryStatNonlin(ts, name,
  alphaStationary = 0.01, alphaNonlinear = 0.01,
  order = 1, ...)
```

Arguments

ts	coredata of a time series
name	the name of the time series
alphaStationary	significance level for the stationarity tests
alphaNonlinear	significance level for the linearity tests
order	the order to be assumed by tests which need it
...	currently not used

partitionATSDir *Partition (to training and test sets) all time series in an ATS directory*

Description

The function loads all files in an ATS directory into an ATSList, applies computePartitions, and saves the result to an ATS directory.

Usage

```
partitionATSDir(inPath,
  partitionParameters = list(type = c("blockedCV"), order = 0, numPartitions = 5, ratioValSet
  outPath = NULL, savePlots = TRUE)
```

Arguments

inPath	the input path, i.e., the ATS directory
partitionParameters	a list giving all parameters that are passed to computePartitions
outPath	the output path. If NULL, it is generated automatically as a subdirectory of the inPath
savePlots	if TRUE, a directory containing plots of all partitions is generated

See Also

embedATSDir, normalizeATSDir

---

permutationEntropy *calculate the permutation entropy of a given time series*

Description

Calculates the permutation entropy of the given order for the given series.

Usage

```
permutationEntropy(ts, ord = 4)
```

Arguments

ts	the time series to which to apply the function
ord	the order of the permutation entropy

---

plot.eTS *Plot function for eTS objects*

Description

Function to plot eTS objects. It just calls the plot function of the zoo package.

Usage

```
## S3 method for class 'eTS'
plot(x, ...)
```

Arguments

x	the eTS object
...	parameters passed to plot.zoo

Details

TODO: Does this work in the multivariate case?

plotPartitionsATS *Save plots of data partitions to files*

Description

This function is used to plot the partitions of a time series as overlay over the original series.

Usage

```
plotPartitionsATS(aTS, partitions, partDir)
```

Arguments

aTS	the original time series as an aTS object
partitions	a partitioned time series. Usually, this is the output of computePartitions
partDir	the destination directory

plotPredictions *Function to plot predictions*

Description

This function is used to plot predictions as overlay on the original time series.

Usage

```
plotPredictions(qualifiedPredictionFileNames,
  referenceDataDir, originalDataDir, targetDir)
```

Arguments

qualifiedPredictionFileNames	a list of file names with paths to all the prediction files to use
referenceDataDir	the directory where the reference data is located (test sets)
originalDataDir	the directory where the original data is located (full time series)
targetDir	directory to which to save to the plots

Details

TODO: Why does this function take directory names, and evaluatePredictions takes a usedData structure?

points.eTS *Points function for eTS objects*

Description

Function to plot eTS objects into an existing plot. It just calls the points function of the zoo package.

Usage

```
## S3 method for class 'eTS'
points(x, ...)
```

Arguments

x	the eTS object
...	parameters passed to points.zoo

Details

TODO: Does this work in the multivariate case?

rBindByColNames *Row-bind together two tables, taking into account the column names*

Description

This function is very powerful and an important utility function of the package. This helper function implements a special case of rbind, where columns in both tables that have the same name are joined, and columns that only exist in one of the two source tables result in "NA" values for the rows in which they don't exist.

Usage

```
rBindByColNames(x, y)
```

Arguments

x	first table
y	second table

Value

the two tables rbind-ed together

See Also

cBindByRowNames

removeFileEnding *Remove file ending from a path string*

Description

This function is a convenience function that simply does: `substr(file, 0, nchar(file)-nchar(ending))`

Usage

```
removeFileEnding(file, ending)
```

Arguments

`file`	a string
`ending`	a substring

Value

the `file` string without the last characters. The amount of characters removed is determined by the length of `ending`.

runExperiments *Start a defined set of experiments.*

Description

This is one of the most important functions of the package. It uses the data defined by `usedData`, a structure which you can generate using the `genUsedData` function, takes the defined methods in the `methodsDefinitions` structure, and applies all methods defined to all data defined. When giving a parallel apply function in `tsApply`, the experiments are run in parallel. All outputs of the experiments are written to the output directory defined in `expPath`.

Usage

```
runExperiments(usedData, expPath, methodsDefinitions,
  tsApply = lapply, seed = 5, predictFunc = predict)
```

Arguments

`usedData`	a structure which defines the data to use for the experiments. Usually generated with the `genUsedData` function.
`expPath`	the path to write the experiments to. A reasonable path can be generated using the function `assembleExperimentsPath`.
`methodsDefinitions`	a structure that defines all the prediction methods you want to use. See the demo code for how to do it.
`tsApply`	the apply function to be used. If using the script on different machines with different parallelization capabilities, you probably want to automatically choose this function using `initParallelMode`.

seed	this seed value is set for every execution of a predictor.
predictFunc	the predict function to be used. Here, you can add special processing of some of the predictors don't use the standard format of predict functions.

Details

TODO: Describe in more detail how to create the methodsDefinitions structure.

Value

the results of all experiments are returned in a list, but – more important – they are saved to disk. The structure in the newly created directory expPath is:

usedData	The usedData structure is saved both in plain text and in the ".RData" format. This structure contains information on all input data used, including md5 hashes of the input data files.
models	a subdirectory with an ".RData" file for every model trained. Take into account that this model may or may not be used for further predictions. This depends on the implementation of the model (if it is implemented in pure R or in C/C++)
predictions	an ".RData" file for every prediction of every model, containing the predictions on the respective test set for this model
results	tables calculated using the predictions and the respective reference and benchmark values: All available error measures are calculated both for the training set (fit, "modEval") and the test set ("predEval"). For the test set, there are also denormalized versions calculated "Denorm". Furthermore, there may be some special tables as the value containing the R package and version for each of the models (or the hash of the git commit, if available).

---

runPrediction *Calculates predictions from a list of models and input data, models are in memory*

Description

This function uses a bunch of models, created with runExperiments, and input data defined in usedData, together with a prediction function to calculate predictions for all data (in the test sets) applying all the trained models. The difference between this function and the function runPredictions is, that the models are not saved on disk and then reloaded. Depending on the implementation of the models, this may cause problems.

Usage

```
runPrediction(usedData, model, expPath,
  predictFunc = predict)
```

Arguments

usedData	the usedData structure to use (see genUsedData)
model	a list of all the models (usually, the output of runExperiments)
expPath	the output path, see assembleExperimentsPath and runExperiments
predictFunc	the function that is applied to the model to get the prediction

Details

TODO: I think this function is not needed/used any more, as the predictions are calculated directly in runExperiments after model building.

See Also

genUsedData, runExperiments, runExperiments, runPredictions

`runPredictions` *Calculates predictions from a list of models and input data, models are on disk*

Description

This function uses a bunch of models, created with runExperiments, and input data defined in usedData, together with a prediction function to calculate predictions for all data (in the test sets) applying all the trained models. The difference between this function and the function runPrediction is, that the models are loaded from disk. Thus, it can be ran indepentdently of the experiments, and e.g., can also be used if the experiments haven't terminated yet, were aborted, etc. The drawback is, that not all models survive a save/load cycle without problems.

Usage

```
runPredictions(usedData, expPath, predictFunc = predict)
```

Arguments

usedData	the usedData structure to use (see genUsedData)
expPath	both the output path, and the path where the models are loaded from. See assembleExperimentsPath and runExperiments
predictFunc	the function that is applied to the model to get the prediction

Details

TODO: This function is not needed/used any more, as the predictions are calculated directly in runExperiments after model building. Maybe there are still some cases, when for some reason prediction failed, but models are there, so that this function can be used to calculate them.

See Also

genUsedData, runExperiments, runExperiments, runPrediction

save.aTS *Save an aTS object to an ".RData" file*

Description

This function saves an aTS object to an ".RData" file.

Usage

```
## S3 method for class 'aTS'
save(aTS, dataPath, name, ...)
```

Arguments

aTS	the aTS object of the time series
dataPath	the path to which to save the series to
name	the (file-)name to which to save to
...	currently not used

saveAsDataPackageATSList *Save an ATSList as a (new) data package*

Description

This function automatizes the import of an ATSList to a data repository as a new data package, by calling sequentially saveATSList with single-file and all-in-one-file mode, savePlotsATSList, and summaryTableATSList with the according types.

Usage

```
saveAsDataPackageATSList(atsList, dataPath,
  dataPackageName, genSummaryPlot = TRUE,
  genDescStatTab = TRUE, genStatNonlinTab = FALSE,
  genStatOtherTestsTab = FALSE, genSingeFilesDir = TRUE,
  minLength = 10)
```

Arguments

atsList	the ATSList to use
dataPath	the path to which to save the data package to
dataPackageName	the name for the data package
genSummaryPlot	if TRUE, savePlotsATSList is used to generate one pdf containing all series.
genDescStatTab	if TRUE, the "DescStat" table is generated
genStatNonlinTab	if TRUE, the "StatNonlin" table is generated

genStatOtherTestsTab
: if TRUE, the "OtherTests" table is generated

genSingeFilesDir
: if TRUE, the series are also saved in single-file mode to disk

minLength
: series below this length will be omitted. For some of the measures calculated for the characteristics tables, a certain minimal length of the series might be necessary.

---

saveATSList *Save an ATSList to an ATS directory*

Description

This function saves an ATSList to an ATS directory.

Usage

```
saveATSList(atsList, dataPath, name, singleFiles = FALSE)
```

Arguments

atsList
: the ATSList to save to disk

dataPath
: the path to which to save to

name
: the name under which to save the list

singleFiles
: if TRUE, every series is saved as a single file, using save.aTS. Otherwise, all series are saved in one file, using the normal save function.

---

savePartitions *Save partitioned time series to ATS directory*

Description

This function is used to save a partitioned time series to an ATS directory.

Usage

```
savePartitions(partitions, path)
```

Arguments

partitions
: a partitioned time series. Usually, this is the output of computePartitions

path
: the directory to which to save the series

savePlotsATSList *Save plots of every time series in an* ATSList *to files*

Description

This function saves plots of all time series in an ATSList to disk.

Usage

```
savePlotsATSList(atsList, filename, acf = TRUE,
  toSeparateJpegs = FALSE)
```

Arguments

atsList	the ATSList to use
filename	the filename or the prefix for the filename
acf	if TRUE, the acf plot is generated and saved along with the plot of the series
toSeparateJpegs	if TRUE, every time series plot is saved as a separate jpeg file. Otherwise, a single ".pdf" file is generated.

setNamesATSList *Set the names of the* ATSList *correctly*

Description

This method calls for every aTS object in the list its getName method, and the list element's names accordingly. E.g., if you have an aTS object of the lynx series, after applying this function to the ATSList, you can access it with atsList[[lynx]], instead of atsList[[1]].

Usage

```
setNamesATSList(atsList)
```

Arguments

atsList	the ATSList for which the names to set

simulateLinearTS *Generate AR, MA, and ARMA series*

Description

This function can be used to generate pure AR, pure MA, and ARMA series. The AR part will be stationary and the MA part invertible. Therefore, the coefficients are not given directly, but a value `maxRoot` which controls the interval from which the roots for the characteristic polynomials are chosen. So, all the roots of the characteristic polynomials are real-valued. For a detailed explanation see the referenced literature.

Usage

```
simulateLinearTS(length, ar = TRUE, ma = TRUE, lags = 4,
  seed = 1, maxRoot = 5, n.start = NA, ...)
```

Arguments

`length`	the length of the series to be generated
`ar`	if `TRUE`, series has an AR part
`ma`	if `TRUE`, series has an MA part
`lags`	the number of lags
`seed`	a seed used for random number generation
`maxRoot`	the roots of the characteristic polynomials are chosen between 1.1 and `maxRoot`
`n.start`	burn-in period. if NA, calculated automatically by arima.sim
`...`	parameters passed to arima.sim

Value

a list containing all the parameters, and a member `ts` with the generated series

References

C. Bergmeir, J.M. Benítez, On the Use of Cross-validation for Time Series Predictor Evaluation. Information Sciences 191 (2012) 192-213.

simulateNonlinearTS *Generate nonlinear time series*

Description

This function can be used to generate nonlinear time series. It is similar to the function `simulateLinearTS`, but applies nonlinear functions to certain lags. The nonlinear functions currently used are: cos, sin, tanh, atan, and exp(-x/10000). For a detailed explanation see the referenced literature.

Usage

```
simulateNonlinearTS(length, lags = 4, seed = 1,
  maxRoot = 5)
```

Arguments

length	the length of the series to be generated
lags	the number of lags
seed	a seed used for random number generation
maxRoot	the roots of the characteristic polynomials are chosen between 1.1 and maxRoot

Value

a list containing all the parameters, and a member ts with the generated series

References

C. Bergmeir, J.M. Benítez, On the Use of Cross-validation for Time Series Predictor Evaluation. Information Sciences 191 (2012) 192-213.

---

simulateStableVarProcess

Simulate data from stable VAR process

Description

This function can be used to simulate data from a random, stable VAR process.

Usage

```
simulateStableVarProcess(dim = 3, order = 2, sd = 1,
  length = 1000)
```

Arguments

dim	the dimension of the VAR (bivariate, trivariate,...)
order	the order of the VAR (how many lags)
sd	the standard deviation of the noise to be included
length	the length of the series to be generated

Value

a matrix containing the simulated data

References

G.N. Boshnakov, B.M. Iqelan (2009). Generation of time series models with given spectral properties. Journal of Time Series Analysis 30(3):349-368.

summary.aTS *Generic summary function for aTS objects*

Description

This function generates a list with some statistical information about the time series in the aTS object.

Usage

```
## S3 method for class 'aTS'
summary(object, ...)
```

Arguments

object	the aTS object
...	additional function parameters (currently not used)

summaryTableATSList *Calculate a summary table of an ATSList*

Description

This method calls for every aTS object in the list the function oneLineSummary, and returns a table that consists of all these summaries.

Usage

```
summaryTableATSList(atsList, ...)
```

Arguments

atsList	the ATSList to which to apply the function to
...	parameters passed to oneLineSummary

toKeelATSDir *Load all files in an ATSDir and save it to a directory of KEEL dat files*

Description

Load all files in an ATSDir and save it to a directory of KEEL dat files

Usage

```
toKeelATSDir(inPath, outPath = NULL)
```

Arguments

inPath	the input path (ATSDir)
outPath	the output path, where to save the dat files

writeATSListToKeelDirectory *Write a list of aTS objects to a directory, in KEEL format*

Description

Write a list of aTS objects to a directory, in KEEL format

Usage

```
writeATSListToKeelDirectory(atsList, path,
  useColNames = FALSE, compression = "none",
  compFilename = "series.zip")
```

Arguments

atsList	the atsList to write out
path	the path to write the data to
useColNames	should the column names be used?
compression	the compression to use. Possible values are zipSingle, zipPackage, and none
compFilename	the file name of the compressed output file

writeATSToKeelFile *Save an aTS time series object to a file in KEEL format*

Description

Save an aTS time series object to a file in KEEL format

Usage

```
writeATSToKeelFile(aTS, filename, useColNames = FALSE)
```

Arguments

aTS	the aTS object to save
filename	complete path to the file to write
useColNames	should colnames present in the aTS object be used for attribute naming?

`writeKeelXmlDescription` *Write out an XML description for use with KEEL*

Description

Write out an XML description for use with KEEL

Usage

```
writeKeelXmlDescription(atsList, path, filename)
```

Arguments

`atsList`	the atsList to use in KEEL
`path`	the directory where to save the xml file
`filename`	the filename of the xml file

Index