pyFTS Documentation

Release 1.7

Machine Intelligence and Data Science Laboratory - UFMG - Brazi

May 26, 2023

API Documentation:

1 What is pyFTS Library? 1
- 1.1 pyFTS Quick Start . . . 1
  - 1.1.1 How to install pyFTS? . . . 1
  - 1.1.2 What are Fuzzy Time Series (FTS)? . . . 1
  - 1.1.3 Usage examples . . . 2
  - 1.1.4 A short tutorial on Fuzzy Time Series . . . 2
- 1.2 pyFTS . . . 2
  - 1.2.1 pyFTS package . . . 2

2 How to reference pyFTS? 47

3 Indexes 49

Python Module Index 51

Index 53

CHAPTER 1

What is pyFTS Library?

This package is intended for students, researchers, data scientists or whose want to exploit the Fuzzy Time Series methods. These methods provide simple, easy to use, computationally cheap and human-readable models, suitable from statistic laymans to experts.

This tool is developed on MINDS Lab, headed by Prof. Frederico Gadelha Guimarães from Electrical Engineering Departament of Federal University of Minas Gerais (UFMG) at Brazil. Also colaborate with this tool the Brazilian instituitions Federal Institute of North of Minas Gerais (IFNMG) and Federal Institute of Minas Gerais (IFMG) .

1.1 pyFTS Quick Start

1.1.1 How to install pyFTS?

Before of all, pyFTS was developed and tested with Python 3.6. To install pyFTS using pip tool

```
pip install -U pyFTS
```

Ou clone directly from the GitHub repo for the most recent review:

```
pip install -U git+https://github.com/PYFTS/pyFTS
```

1.1.2 What are Fuzzy Time Series (FTS)?

Fuzzy Time Series (FTS) are non parametric methods for time series forecasting based on Fuzzy Theory. The original method was proposed by [1] and improved later by many researchers. The general approach of the FTS methods, based on [2] is listed below:

1. **Data preprocessing**: Data transformation functions contained at pyFTS.common.Transformations, like differentiation, Box-Cox, scaling and normalization.

2. **Universe of Discourse Partitioning**: This is the most important step. Here, the range of values of the numerical time series *Y(t)* will be splited in overlapped intervals and for each interval will be created a Fuzzy Set. This step is performed by pyFTS.partition module and its classes (for instance GridPartitioner, EntropyPartitioner, etc). The main parameters are:

   - the number of intervals
   - which fuzzy membership function (on pyFTS.common.Membership)
   - partition scheme (GridPartitioner, EntropyPartitioner, FCMPartitioner, HuarngPartitioner)

   Check out the jupyter notebook on notebooks/Partitioners.ipynb for sample codes.

3. **Data Fuzzyfication**: Each data point of the numerical time series *Y(t)* will be translated to a fuzzy representation (usually one or more fuzzy sets), and then a fuzzy time series *F(t)* is created.

4. **Generation of Fuzzy Rules**: In this step the temporal transition rules are created. These rules depends on the method and their characteristics: - *order*: the number of time lags used on forecasting - *weights*: the weighted models introduce weights on fuzzy rules for smoothing - *seasonality*: seasonality models - *steps ahead*: the number of steps ahed to predict. Almost all standard methods are based on one-step-ahead forecasting - *forecasting type*: Almost all standard methods are point-based, but pyFTS also provides intervalar and probabilistic forecasting methods.

5. **Forecasting**: The forecasting step takes a sample (with minimum length equal to the model's order) and generate a fuzzy outputs (fuzzy set(s)) for the next time ahead.

6. **Defuzzyfication**: This step transform the fuzzy forecast into a real number.

7. **Data postprocessing**: The inverse operations of step 1.

1.1.3 Usage examples

There is nothing better than good code examples to start. Then check out the demo Jupyter Notebooks of the implemented method os pyFTS!.

A Google Colab example can also be found here.

1.1.4 A short tutorial on Fuzzy Time Series

Part I: Introduction to the Fuzzy Logic, Fuzzy Time Series and the pyFTS library.

Part II: High order, weighted and multivariate methods and a case study of solar energy forecasting..

Part III: Interval and probabilistic forecasting, non-stationary time series, concept drifts and time variant models..

1.2 pyFTS

1.2.1 pyFTS package

Subpackages

pyFTS.benchmarks package

Module contents

pyFTS module for benchmarking the FTS models

Submodules

pyFTS.benchmarks.benchmarks module

pyFTS.benchmarks.Measures module

pyFTS.benchmarks.ResidualAnalysis module

pyFTS.benchmarks.Tests module

pyFTS.benchmarks.Util module

pyFTS.benchmarks.arima module

pyFTS.benchmarks.knn module

pyFTS.benchmarks.naive module

pyFTS.benchmarks.quantreg module

pyFTS.benchmarks.gaussianproc module

pyFTS.benchmarks.BSTS module

pyFTS.common package

Module contents

Submodules

pyFTS.common.Activations module

Activation functions for Time Series Classification

pyFTS.common.Activations.**argmax**(*dist: dict*, *weights: dict*) → str

pyFTS.common.Activations.**scale**(*dist: dict*, *weights: dict*) → dict

pyFTS.common.Activations.**softmax**(*dist: dict*, *weights: dict*) → dict

pyFTS.common.Composite module

Composite Fuzzy Sets

class pyFTS.common.Composite.**FuzzySet**(*name*, *superset=False*, ***kwargs*)

Bases: *pyFTS.common.FuzzySet.FuzzySet*

Composite Fuzzy Set

append(*mf*, *parameters*)

Adds a new function to composition

Parameters

- **mf** –
- **parameters** –

Returns

append_set(*set*)
Adds a new function to composition

Parameters

- **mf** –
- **parameters** –

Returns

membership(*x*)
Calculate the membership value of a given input

Parameters **x** – input value

Returns membership value of x at this fuzzy set

transform(*x*)
Preprocess the data point for non native types

Parameters **x** –

Returns return a native type value for the structured type

pyFTS.common.FLR module

This module implements functions for Fuzzy Logical Relationship generation

class pyFTS.common.FLR.**FLR**(*LHS*, *RHS*)
Bases: object

Fuzzy Logical Relationship

Represents a temporal transition of the fuzzy set LHS on time t for the fuzzy set RHS on time t+1.

LHS = None
Left Hand Side fuzzy set

RHS = None
Right Hand Side fuzzy set

class pyFTS.common.FLR.**IndexedFLR**(*index*, *LHS*, *RHS*)
Bases: *pyFTS.common.FLR.FLR*

Season Indexed Fuzzy Logical Relationship

index = None
seasonal index

pyFTS.common.FLR.**generate_high_order_recurrent_flr**(*fuzzyData*)
Create a ordered FLR set from a list of fuzzy sets with recurrence

Parameters **fuzzyData** – ordered list of fuzzy sets

Returns ordered list of FLR

pyFTS.common.FLR.**generate_indexed_flrs**(*sets*, *indexer*, *data*, *transformation=None*, *alpha_cut=0.0*)

Create a season-indexed ordered FLR set from a list of fuzzy sets with recurrence

Parameters

- **sets** – fuzzy sets
- **indexer** – seasonality indexer
- **data** – original data

Returns ordered list of FLR

pyFTS.common.FLR.**generate_non_recurrent_flrs**(*fuzzyData*, *steps=1*)

Create a ordered FLR set from a list of fuzzy sets without recurrence

Parameters **fuzzyData** – ordered list of fuzzy sets

Returns ordered list of FLR

pyFTS.common.FLR.**generate_recurrent_flrs**(*fuzzyData*, *steps=1*)

Create a ordered FLR set from a list of fuzzy sets with recurrence

Parameters

- **fuzzyData** – ordered list of fuzzy sets
- **steps** – the number of steps ahead on the right side of FLR

Returns ordered list of FLR

pyFTS.common.FuzzySet module

class pyFTS.common.FuzzySet.**FuzzySet**(*name: str*, *mf*, *parameters: list*, *centroid: float*, *alpha: float = 1.0*, ***kwargs*)

Bases: object

Fuzzy Set

Z = None
Partition function in respect to the membership function

alpha = None
The alpha cut value

centroid = None
The fuzzy set center of mass (or midpoint)

membership(*x*)
Calculate the membership value of a given input

Parameters **x** – input value

Returns membership value of x at this fuzzy set

mf = None
The membership function

name = None
The fuzzy set name

parameters = None
The parameters of the membership function

partition_function(*uod=None*, *nbins=100*)
Calculate the partition function over the membership function.

Parameters

- **uod** –
- **nbins** –

Returns

transform(*x*)
Preprocess the data point for non native types

Parameters **x** –

Returns return a native type value for the structured type

type = None
The fuzzy set type (common, composite, nonstationary, etc)

variable = None
In multivariate time series, indicate for which variable this fuzzy set belogs

pyFTS.common.FuzzySet.**check_bounds**(*data*, *fuzzy_sets*, *ordered_sets*)

pyFTS.common.FuzzySet.**check_bounds_index**(*data*, *fuzzy_sets*, *ordered_sets*)

pyFTS.common.FuzzySet.**fuzzyfy**(*data*, *partitioner*, ***kwargs*)
A general method for fuzzyfication.

Parameters

- **data** – input value to be fuzzyfied
- **partitioner** – a trained pyFTS.partitioners.Partitioner object
- **kwargs** – dict, optional arguments
- **alpha_cut** – the minimal membership value to be considered on fuzzyfication (only for mode='sets')
- **method** – the fuzzyfication method (fuzzy: all fuzzy memberships, maximum: only the maximum membership)
- **mode** – the fuzzyfication mode (sets: return the fuzzy sets names, vector: return a vector with the membership

values for all fuzzy sets, both: return a list with tuples (fuzzy set, membership value))

:returns a list with the fuzzyfied values, depending on the mode

pyFTS.common.FuzzySet.**fuzzyfy_instance**(*inst*, *fuzzy_sets: dict*, *ordered_sets: list = None*)
Calculate the membership values for a data point given fuzzy sets

Parameters

- **inst** – data point
- **fuzzy_sets** – a dictionary where the key is the fuzzy set name and the value is the fuzzy set object.
- **ordered_sets** – a list with the fuzzy sets names ordered by their centroids.

Returns array of membership values

pyFTS.common.FuzzySet.**fuzzyfy_instances**(*data: list*, *fuzzy_sets: dict*, *ordered_sets=None*) → list

Calculate the membership values for a data point given fuzzy sets

Parameters

- **inst** – data point
- **fuzzy_sets** – a dictionary where the key is the fuzzy set name and the value is the fuzzy set object.
- **ordered_sets** – a list with the fuzzy sets names ordered by their centroids.

Returns array of membership values

pyFTS.common.FuzzySet.**fuzzyfy_series**(*data*, *fuzzy_sets*, *method='maximum'*, *alpha_cut=0.0*, *ordered_sets=None*)

pyFTS.common.FuzzySet.**fuzzyfy_series_old**(*data*, *fuzzy_sets*, *method='maximum'*)

pyFTS.common.FuzzySet.**get_fuzzysets**(*inst*, *fuzzy_sets: dict*, *ordered_sets: list = None*, *alpha_cut: float = 0.0*) → list

Return the fuzzy sets which membership value for a inst is greater than the alpha_cut

Parameters

- **inst** – data point
- **fuzzy_sets** – a dictionary where the key is the fuzzy set name and the value is the fuzzy set object.
- **ordered_sets** – a list with the fuzzy sets names ordered by their centroids.
- **alpha_cut** – Minimal membership to be considered on fuzzyfication process

Returns array of membership values

pyFTS.common.FuzzySet.**get_maximum_membership_fuzzyset**(*inst*, *fuzzy_sets*, *ordered_sets=None*) → pyFTS.common.FuzzySet.FuzzySet

Fuzzify a data point, returning the fuzzy set with maximum membership value

Parameters

- **inst** – data point
- **fuzzy_sets** – a dictionary where the key is the fuzzy set name and the value is the fuzzy set object.
- **ordered_sets** – a list with the fuzzy sets names ordered by their centroids.

Returns fuzzy set with maximum membership

pyFTS.common.FuzzySet.**get_maximum_membership_fuzzyset_index**(*inst*, *fuzzy_sets*) → int

Fuzzify a data point, returning the fuzzy set with maximum membership value

Parameters

- **inst** – data point
- **fuzzy_sets** – dict of fuzzy sets

Returns fuzzy set with maximum membership

pyFTS.common.FuzzySet.**grant_bounds**(*data*, *fuzzy_sets*, *ordered_sets*)

pyFTS.common.FuzzySet.**set_ordered**(*fuzzy_sets*)

Order a fuzzy set list by their centroids

Parameters **fuzzy_sets** – a dictionary where the key is the fuzzy set name and the value is the fuzzy set object.

Returns a list with the fuzzy sets names ordered by their centroids.

pyFTS.common.Membership module

Membership functions for Fuzzy Sets

pyFTS.common.Membership.**bellmf**(*x*, *parameters*)
Bell shaped membership function

Parameters

- **x** –
- **parameters** –

Returns

pyFTS.common.Membership.**gaussmf**(*x*, *parameters*)
Gaussian fuzzy membership function

Parameters

- **x** – data point
- **parameters** – a list with 2 real values (mean and variance)

Returns the membership value of x given the parameters

pyFTS.common.Membership.**sigmf**(*x*, *parameters*)
Sigmoid / Logistic membership function

Parameters

- **x** –
- **parameters** – an list with 2 real values (smoothness and midpoint)

:return

pyFTS.common.Membership.**singleton**(*x*, *parameters*)
Singleton membership function, a single value fuzzy function

Parameters

- **x** –
- **parameters** – a list with one real value

:returns

pyFTS.common.Membership.**trapmf**(*x*, *parameters*)
Trapezoidal fuzzy membership function

Parameters

- **x** – data point
- **parameters** – a list with 4 real values

Returns the membership value of x given the parameters

pyFTS.common.Membership.**trimf**(*x*, *parameters*)
Triangular fuzzy membership function

Parameters

- **x** – data point
- **parameters** – a list with 3 real values

Returns the membership value of x given the parameters

pyFTS.common.SortedCollection module

class pyFTS.common.SortedCollection.**SortedCollection**(*iterable=()*, *key=None*)

Bases: object

Sequence sorted by a key function.

SortedCollection() is much easier to work with than using bisect() directly. It supports key functions like those use in sorted(), min(), and max(). The result of the key function call is saved so that keys can be searched efficiently.

Instead of returning an insertion-point which can be hard to interpret, the five find-methods return a specific item in the sequence. They can scan for exact matches, the last item less-than-or-equal to a key, or the first item greater-than-or-equal to a key.

Once found, an item's ordinal position can be located with the index() method. New items can be added with the insert() and insert_right() methods. Old items can be deleted with the remove() method.

The usual sequence methods are provided to support indexing, slicing, length lookup, clearing, copying, forward and reverse iteration, contains checking, item counts, item removal, and a nice looking repr.

Finding and indexing are O(log n) operations while iteration and insertion are O(n). The initial sort is O(n log n).

The key function is stored in the 'key' attibute for easy introspection or so that you can assign a new key function (triggering an automatic re-sort).

In short, the class was designed to handle all of the common use cases for bisect but with a simpler API and support for key functions.

```
>>> from pprint import pprint
>>> from operator import itemgetter
```

```
>>> s = SortedCollection(key=itemgetter(2))
>>> for record in [
...         ('roger', 'young', 30),
...         ('angela', 'jones', 28),
...         ('bill', 'smith', 22),
...         ('david', 'thomas', 32)]:
...     s.insert(record)
```

```
>>> pprint(list(s))         # show records sorted by age
[('bill', 'smith', 22),
 ('angela', 'jones', 28),
 ('roger', 'young', 30),
 ('david', 'thomas', 32)]
```

```
>>> s.find_le(29)           # find oldest person aged 29 or younger
('angela', 'jones', 28)
>>> s.find_lt(28)           # find oldest person under 28
```

(continues on next page)

(continued from previous page)

```
('bill', 'smith', 22)
>>> s.find_gt(28)           # find youngest person over 28
('roger', 'young', 30)
```

```
>>> r = s.find_ge(32)       # find youngest person aged 32 or older
>>> s.index(r)              # get the index of their record
3
>>> s[3]                    # fetch the record at that index
('david', 'thomas', 32)
```

```
>>> s.key = itemgetter(0)   # now sort by first name
>>> pprint(list(s))
[('angela', 'jones', 28),
 ('bill', 'smith', 22),
 ('david', 'thomas', 32),
 ('roger', 'young', 30)]
```

around(*k*)

between(*ge*, *le*)

clear()

copy()

count(*item*)
Return number of occurrences of item

find(*k*)
Return first item with a key == k. Raise ValueError if not found.

find_ge(*k*)
Return first item with a key >= equal to k. Raise ValueError if not found

find_gt(*k*)
Return first item with a key > k. Raise ValueError if not found

find_le(*k*)
Return last item with a key <= k. Raise ValueError if not found.

find_lt(*k*)
Return last item with a key < k. Raise ValueError if not found.

index(*item*)
Find the position of an item. Raise ValueError if not found.

insert(*item*)
Insert a new item. If equal keys are found, add to the left

insert_right(*item*)
Insert a new item. If equal keys are found, add to the right

inside(*ge*, *le*)

key
key function

remove(*item*)
Remove first occurence of item. Raise ValueError if not found

pyFTS.common.Util module

pyFTS.common.flrg module

class pyFTS.common.flrg.**FLRG**(*order*, ***kwargs*)
Bases: object

Fuzzy Logical Relationship Group

Group a set of FLR's with the same LHS. Represents the temporal patterns for time t+1 (the RHS fuzzy sets) when the LHS pattern is identified on time t.

LHS = None
Left Hand Side of the rule

RHS = None
Right Hand Side of the rule

append_rhs(*set*, ***kwargs*)

get_key()
Returns a unique identifier for this FLRG

get_lower(*sets*)
Returns the lower bound value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns lower bound value

get_membership(*data*, *sets*)
Returns the membership value of the FLRG for the input data

Parameters

- **data** – input data
- **sets** – fuzzy sets

Returns the membership value

get_midpoint(*sets*)
Returns the midpoint value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns the midpoint value

get_midpoints(*sets*)

get_upper(*sets*)
Returns the upper bound value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns upper bound value

order = None
Number of lags on LHS

reset_calculated_values()

pyFTS.common.fts module

pyFTS.common.tree module

Tree data structure

class pyFTS.common.tree.**FLRGTree**
Bases: object

Represents a FLRG set with a tree structure

class pyFTS.common.tree.**FLRGTreeNode**(*value*)
Bases: object

Tree node for

appendChild(*child*)

getChildren()

getStr(*k*)

paths(*acc=[]*)

pyFTS.common.tree.**build_tree_without_order**(*node*, *lags*, *level*)

pyFTS.common.tree.**flat**(*dados*)

pyFTS.common.transformations package

Module contents

Submodules

pyFTS.common.transformations.adapativeexpectation module

pyFTS.common.transformations.boxcox module

class pyFTS.common.transformations.boxcox.**BoxCox**(*plambda*)
Bases: *pyFTS.common.transformations.transformation.Transformation*

Box-Cox power transformation

y'(t) = log(y(t)) y(t) = exp(y'(t))

apply(*data*, *param=None*, ***kwargs*)
Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param=None*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

parameters

pyFTS.common.transformations.differential module

class pyFTS.common.transformations.differential.**Differential**(*lag*)
Bases: *pyFTS.common.transformations.transformation.Transformation*

Differentiation data transform

y'(t) = y(t) - y(t-1) y(t) = y(t-1) + y'(t)

apply(*data*, *param=None*, ***kwargs*)
Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

parameters

pyFTS.common.transformations.normalization module

class pyFTS.common.transformations.normalization.**Normalization**(***kwargs*)
Bases: *pyFTS.common.transformations.transformation.Transformation*

apply(*data*, *param=None*, ***kwargs*)
Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param=None*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

train(*data*, ***kwargs*)

pyFTS.common.transformations.roi module

class pyFTS.common.transformations.roi.**ROI**(***kwargs*)

Bases: *pyFTS.common.transformations.transformation.Transformation*

Return of Investment (ROI) transformation. Retrieved from Sadaei and Lee (2014) - Multilayer Stock Forecasting Model Using Fuzzy Time Series

y'(t) = (y(t) - y(t-1)) / y(t-1) y(t) = (y(t-1) * y'(t)) + y(t-1)

apply(*data*, *param=None*, ***kwargs*)

Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param=None*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

pyFTS.common.transformations.scale module

class pyFTS.common.transformations.scale.**Scale**(*min=0*, *max=1*)

Bases: *pyFTS.common.transformations.transformation.Transformation*

Scale data inside a interval [min, max]

apply(*data*, *param=None*, ***kwargs*)

Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

parameters

pyFTS.common.transformations.smoothing module

class pyFTS.common.transformations.smoothing.**AveragePooling**(***kwargs*)
Bases: *pyFTS.common.transformations.transformation.Transformation*

apply(*data*)
Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param=None*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

class pyFTS.common.transformations.smoothing.**ExponentialSmoothing**(***kwargs*)
Bases: *pyFTS.common.transformations.transformation.Transformation*

apply(*data*, *param=None*, ***kwargs*)
Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param=None*, ***kwargs*)

Parameters

- **data** – transformed data

- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

class pyFTS.common.transformations.smoothing.**MaxPooling**(***kwargs*)

Bases: *pyFTS.common.transformations.transformation.Transformation*

apply(*data*)

Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param=None*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

class pyFTS.common.transformations.smoothing.**MovingAverage**(***kwargs*)

Bases: *pyFTS.common.transformations.transformation.Transformation*

apply(*data*, *param=None*, ***kwargs*)

Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param=None*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

pyFTS.common.transformations.som module

Kohonen Self Organizing Maps for Fuzzy Time Series

class pyFTS.common.transformations.som.**SOMTransformation**(*grid_dimension: Tuple*, ***kwargs*)

Bases: *pyFTS.common.transformations.transformation.Transformation*

apply(*data: pandas.core.frame.DataFrame*, *param=None*, ***kwargs*)

Transform a M-dimensional dataset into a 3-dimensional dataset, where one dimension is the endogen variable If endogen_variable = None, the last column will be the endogen_variable.

Args: data (pd.DataFrame): M-Dimensional dataset endogen_variable (str): column of dataset names (Tuple): names for new columns created by SOM Transformation. param: **kwargs: params of SOM's train process

percentage_train (float). Percentage of dataset that will be used for train SOM network. default: 0.7 leaning_rate (float): leaning rate of SOM network. default: 0.01 epochs: epochs of SOM network. default: 10000

Returns:

save_net(*filename: str = 'SomNet trained'*)

show_grid(*graph_type: str = 'nodes_graph'*, ***kwargs*)

train(*data: pandas.core.frame.DataFrame*, *percentage_train: float = 0.7*, *leaning_rate: float = 0.01*, *epochs: int = 10000*)

pyFTS.common.transformations.transformation module

class pyFTS.common.transformations.transformation.**Transformation**(***kwargs*)

Bases: object

Data transformation used on pre and post processing of the FTS

apply(*data*, *param*, ***kwargs*)

Apply the transformation on input data

Parameters

- **data** – input data
- **param** –
- **kwargs** –

Returns numpy array with transformed data

inverse(*data*, *param*, ***kwargs*)

Parameters

- **data** – transformed data
- **param** –
- **kwargs** –

Returns numpy array with inverse transformed data

is_multivariate = None

detemine if this transformation can be applied to multivariate data

pyFTS.common.transformations.trend module

pyFTS.data package

Module contents

Module for pyFTS standard datasets facilities

Submodules

pyFTS.data.common module

pyFTS.data.common.**get_dataframe**(*filename: str*, *url: str*, *sep: str = ';'*, *compression: str = 'infer'*)
→ pandas.core.frame.DataFrame

This method check if filename already exists, read the file and return its data. If the file don't already exists, it will be downloaded and decompressed.

Parameters

- **filename** – dataset local filename
- **url** – dataset internet URL
- **sep** – CSV field separator
- **compression** – type of compression

Returns Pandas dataset

Datasets

Artificial and synthetic data generators

Facilities to generate synthetic stochastic processes

class pyFTS.data.artificial.**SignalEmulator**(***kwargs*)

Bases: object

Emulate a complex signal built from several additive and non-additive components

blip(***kwargs*)

Creates an outlier greater than the maximum or lower then the minimum previous values of the signal, and insert it on a random location of the signal.

Returns the current SignalEmulator instance, for method chaining

components = None

Components of the signal

incremental_gaussian(*mu: float*, *sigma: float*, ***kwargs*)

Creates an additive gaussian interference on a previous signal

Parameters

- **mu** – increment on mean
- **sigma** – increment on variance

- **start** – lag index to start this signal, the default value is 0
- **it** – Number of iterations, the default value is 1
- **length** – Number of samples generated on each iteration, the default value is 100
- **vmin** – Lower bound value of generated data, the default value is None
- **vmax** – Upper bound value of generated data, the default value is None

Returns the current SignalEmulator instance, for method chaining

periodic_gaussian(*type: str*, *period: int*, *mu_min: float*, *sigma_min: float*, *mu_max: float*, *sigma_max: float*, ***kwargs*)

Creates an additive periodic gaussian interference on a previous signal

Parameters

- **type** – 'linear' or 'sinoidal'
- **period** – the period of recurrence
- **mu** – increment on mean
- **sigma** – increment on variance
- **start** – lag index to start this signal, the default value is 0
- **it** – Number of iterations, the default value is 1
- **length** – Number of samples generated on each iteration, the default value is 100
- **vmin** – Lower bound value of generated data, the default value is None
- **vmax** – Upper bound value of generated data, the default value is None

Returns the current SignalEmulator instance, for method chaining

run()

Render the signal

Returns a list of float values

stationary_gaussian(*mu: float*, *sigma: float*, ***kwargs*)

Creates a continuous Gaussian signal with mean mu and variance sigma.

Parameters

- **mu** – mean
- **sigma** – variance
- **additive** – If False it cancels the previous signal and start this one, if True this signal is added to the previous one
- **start** – lag index to start this signal, the default value is 0
- **it** – Number of iterations, the default value is 1
- **length** – Number of samples generated on each iteration, the default value is 100
- **vmin** – Lower bound value of generated data, the default value is None
- **vmax** – Upper bound value of generated data, the default value is None

Returns the current SignalEmulator instance, for method chaining

pyFTS.data.artificial.**generate_gaussian_linear**(*mu_ini*, *sigma_ini*, *mu_inc*, *sigma_inc*, *it=100*, *num=10*, *vmin=None*, *vmax=None*)

Generate data sampled from Gaussian distribution, with constant or linear changing parameters

Parameters

- **mu_ini** – Initial mean
- **sigma_ini** – Initial variance
- **mu_inc** – Mean increment after 'num' samples
- **sigma_inc** – Variance increment after 'num' samples
- **it** – Number of iterations
- **num** – Number of samples generated on each iteration
- **vmin** – Lower bound value of generated data
- **vmax** – Upper bound value of generated data

Returns A list of it*num float values

pyFTS.data.artificial.**generate_linear_periodic_gaussian**(*period*, *mu_min*, *sigma_min*, *mu_max*, *sigma_max*, *it=100*, *num=10*, *vmin=None*, *vmax=None*)

Generates a periodic linear variation on mean and variance

Parameters

- **period** – the period of recurrence
- **mu_min** – initial (and minimum) mean of each period
- **sigma_min** – initial (and minimum) variance of each period
- **mu_max** – final (and maximum) mean of each period
- **sigma_max** – final (and maximum) variance of each period
- **it** – Number of iterations
- **num** – Number of samples generated on each iteration
- **vmin** – Lower bound value of generated data
- **vmax** – Upper bound value of generated data

Returns A list of it*num float values

pyFTS.data.artificial.**generate_sinoidal_periodic_gaussian**(*period*, *mu_min*, *sigma_min*, *mu_max*, *sigma_max*, *it=100*, *num=10*, *vmin=None*, *vmax=None*)

Generates a periodic sinoidal variation on mean and variance

Parameters

- **period** – the period of recurrence
- **mu_min** – initial (and minimum) mean of each period
- **sigma_min** – initial (and minimum) variance of each period

- **mu_max** – final (and maximum) mean of each period
- **sigma_max** – final (and maximum) variance of each period
- **it** – Number of iterations
- **num** – Number of samples generated on each iteration
- **vmin** – Lower bound value of generated data
- **vmax** – Upper bound value of generated data

Returns A list of it*num float values

pyFTS.data.artificial.**generate_uniform_linear**(*min_ini*, *max_ini*, *min_inc*, *max_inc*, *it=100*, *num=10*, *vmin=None*, *vmax=None*)

Generate data sampled from Uniform distribution, with constant or linear changing bounds

Parameters

- **mu_ini** – Initial mean
- **sigma_ini** – Initial variance
- **mu_inc** – Mean increment after 'num' samples
- **sigma_inc** – Variance increment after 'num' samples
- **it** – Number of iterations
- **num** – Number of samples generated on each iteration
- **vmin** – Lower bound value of generated data
- **vmax** – Upper bound value of generated data

Returns A list of it*num float values

pyFTS.data.artificial.**random_walk**(*n=500*, *type='gaussian'*)

Simple random walk

Parameters

- **n** – number of samples
- **type** – 'gaussian' or 'uniform'

Returns

pyFTS.data.artificial.**white_noise**(*n=500*)

Simple Gaussian noise signal :param n: number of samples :return:

AirPassengers dataset

Monthly totals of a airline passengers from USA, from January 1949 through December 1960.

Source: Hyndman, R.J., Time Series Data Library, http://www-personal.buseco.monash.edu.au/~hyndman/TSDL/.

pyFTS.data.AirPassengers.**get_data**() → numpy.ndarray

Get a simple univariate time series data.

Returns numpy array

pyFTS.data.AirPassengers.**get_dataframe**() → pandas.core.frame.DataFrame

Get the complete multivariate time series data.

Returns Pandas DataFrame

Bitcoin dataset

Bitcoin to USD quotations

Daily averaged index, by business day, from 2010 to 2018.

Source: https://finance.yahoo.com/quote/BTC-USD?p=BTC-USD

pyFTS.data.Bitcoin.**get_data**(*field: str = 'AVG'*) → numpy.ndarray
Get the univariate time series data.

Parameters **field** – dataset field to load

Returns numpy array

pyFTS.data.Bitcoin.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

DowJones dataset

DJI - Dow Jones

Daily averaged index, by business day, from 1985 to 2017.

Source: https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC

pyFTS.data.DowJones.**get_data**(*field: str = 'AVG'*) → numpy.ndarray
Get the univariate time series data.

Parameters **field** – dataset field to load

Returns numpy array

pyFTS.data.DowJones.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

Enrollments dataset

Yearly University of Alabama enrollments from 1971 to 1992.

pyFTS.data.Enrollments.**get_data**() → numpy.ndarray
Get a simple univariate time series data.

Returns numpy array

pyFTS.data.Enrollments.**get_dataframe**() → pandas.core.frame.DataFrame

Ethereum dataset

Ethereum to USD quotations

Daily averaged index, by business day, from 2016 to 2018.

Source: https://finance.yahoo.com/quote/ETH-USD?p=ETH-USD

pyFTS.data.Ethereum.**get_data**(*field: str = 'AVG'*) → numpy.ndarray
Get the univariate time series data.

Parameters **field** – dataset field to load

Returns numpy array

pyFTS.data.Ethereum.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

EUR-GBP dataset

FOREX market EUR-GBP pair.

Daily averaged quotations, by business day, from 2016 to 2018.

pyFTS.data.EURGBP.**get_data**(*field: str = 'avg'*) → numpy.ndarray
Get the univariate time series data.

Parameters **field** – dataset field to load

Returns numpy array

pyFTS.data.EURGBP.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

EUR-USD dataset

FOREX market EUR-USD pair.

Daily averaged quotations, by business day, from 2016 to 2018.

pyFTS.data.EURUSD.**get_data**(*field: str = 'avg'*) → numpy.ndarray
Get the univariate time series data.

Parameters **field** – dataset field to load

Returns numpy array

pyFTS.data.EURUSD.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

GBP-USD dataset

FOREX market GBP-USD pair.

Daily averaged quotations, by business day, from 2016 to 2018.

pyFTS.data.GBPUSD.**get_data**(*field: str = 'avg'*) → numpy.ndarray
Get the univariate time series data.

Parameters **field** – dataset field to load

Returns numpy array

pyFTS.data.GBPUSD.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

INMET dataset

INMET - Instituto Nacional Meteorologia / Brasil

Belo Horizonte station, from 2000-01-01 to 31/12/2012

Source: http://www.inmet.gov.br

pyFTS.data.INMET.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

Malaysia dataset

Hourly Malaysia eletric load and tempeature

pyFTS.data.Malaysia.**get_data**(*field: str = 'load'*) → numpy.ndarray
Get the univariate time series data.

Parameters **field** – dataset field to load

Returns numpy array

pyFTS.data.Malaysia.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

NASDAQ module

National Association of Securities Dealers Automated Quotations - Composite Index (NASDAQ IXIC)

Daily averaged index by business day, from 2000 to 2016.

Source: http://www.nasdaq.com/aspx/flashquotes.aspx?symbol=IXIC&selected=IXIC

pyFTS.data.NASDAQ.**get_data**(*field: str = 'avg'*) → numpy.ndarray
Get a simple univariate time series data.

Parameters **field** – the dataset field name to extract

Returns numpy array

pyFTS.data.NASDAQ.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

SONDA dataset

SONDA - Sistema de Organização Nacional de Dados Ambientais, from INPE - Instituto Nacional de Pesquisas Espaciais, Brasil.

Brasilia station

Source: http://sonda.ccst.inpe.br/

pyFTS.data.SONDA.**get_data**(*field: str*) → numpy.ndarray
Get a simple univariate time series data.

Parameters **field** – the dataset field name to extract

Returns numpy array

pyFTS.data.SONDA.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

S&P 500 dataset

S&P500 - Standard & Poor's 500

Daily averaged index, by business day, from 1950 to 2017.

Source: https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC

pyFTS.data.SP500.**get_data**() → numpy.ndarray
Get the univariate time series data.

Returns numpy array

pyFTS.data.SP500.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

TAIEX dataset

The Taiwan Stock Exchange Capitalization Weighted Stock Index (TAIEX)

Daily averaged index by business day, from 1995 to 2014.

Source: http://www.twse.com.tw/en/products/indices/Index_Series.php

pyFTS.data.TAIEX.**get_data**() → numpy.ndarray
Get the univariate time series data.

Returns numpy array

pyFTS.data.TAIEX.**get_dataframe**() → pandas.core.frame.DataFrame
Get the complete multivariate time series data.

Returns Pandas DataFrame

Henon chaotic time series

M. Hénon. "A two-dimensional mapping with a strange attractor". Commun. Math. Phys. 50, 69-77 (1976)

dx/dt = a + by(t-1) - x(t-1)^2 dy/dt = x

pyFTS.data.henon.**get_data**(*var: str, a: float = 1.4, b: float = 0.3, initial_values: list = [1, 1], iterations: int = 1000*) → pandas.core.frame.DataFrame

Get a simple univariate time series data.

Parameters **var** – the dataset field name to extract

Returns numpy array

pyFTS.data.henon.**get_dataframe**(*a: float = 1.4, b: float = 0.3, initial_values: list = [1, 1], iterations: int = 1000*) → pandas.core.frame.DataFrame

Return a dataframe with the bivariate Henon Map time series (x, y).

Parameters

- **a** – Equation coefficient
- **b** – Equation coefficient
- **initial_values** – numpy array with the initial values of x and y. Default: [1, 1]
- **iterations** – number of iterations. Default: 1000

Returns Panda dataframe with the x and y values

Logistic_map chaotic time series

May, Robert M. (1976). "Simple mathematical models with very complicated dynamics". Nature. 261 (5560): 459–467. doi:10.1038/261459a0.

x(t) = r * x(t-1) * (1 - x(t -1))

pyFTS.data.logistic_map.**get_data**(*r: float = 4*, *initial_value: float = 0.3*, *iterations: int = 100*) → list

Return a list with the logistic map chaotic time series.

Parameters

- **r** – Equation coefficient
- **initial_value** – Initial value of x. Default: 0.3
- **iterations** – number of iterations. Default: 100

Returns

Lorentz chaotic time series

Lorenz, Edward Norton (1963). "Deterministic nonperiodic flow". Journal of the Atmospheric Sciences. 20 (2): 130–141. https://doi.org/10.1175/1520-0469(1963)020<0130:DNF>2.0.CO;2

dx/dt = a(y -x) dy/dt = x(b - z) - y dz/dt = xy - cz

pyFTS.data.lorentz.**get_data**(*var: str, a: float = 10.0, b: float = 28.0, c: float = 2.6666666666666665, dt: float = 0.01, initial_values: list = [0.1, 0, 0], iterations: int = 1000*) → pandas.core.frame.DataFrame

Get a simple univariate time series data.

Parameters **var** – the dataset field name to extract

Returns numpy array

pyFTS.data.lorentz.**get_dataframe**(*a: float = 10.0, b: float = 28.0, c: float = 2.6666666666666665, dt: float = 0.01, initial_values: list = [0.1, 0, 0], iterations: int = 1000*) → pandas.core.frame.DataFrame

Return a dataframe with the multivariate Lorenz Map time series (x, y, z).

Parameters

- **a** – Equation coefficient. Default value: 10
- **b** – Equation coefficient. Default value: 28
- **c** – Equation coefficient. Default value: 8.0/3.0
- **dt** – Time differential for continuous time integration. Default value: 0.01
- **initial_values** – numpy array with the initial values of x,y and z. Default: [0.1, 0, 0]
- **iterations** – number of iterations. Default: 1000

Returns Panda dataframe with the x, y and z values

Mackey-Glass chaotic time series

Mackey, M. C. and Glass, L. (1977). Oscillation and chaos in physiological control systems. Science, 197(4300):287-289.

dy/dt = -by(t)+ cy(t - tau) / 1+y(t-tau)^10

pyFTS.data.mackey_glass.**get_data**(*b: float = 0.1, c: float = 0.2, tau: float = 17, initial_values: numpy.ndarray = array([0.5, 0.55882353, 0.61764706, 0.67647059, 0.73529412, 0.79411765, 0.85294118, 0.91176471, 0.97058824, 1.02941176, 1.08823529, 1.14705882, 1.20588235, 1.26470588, 1.32352941, 1.38235294, 1.44117647, 1.5]), iterations: int = 1000*) → list

Return a list with the Mackey-Glass chaotic time series.

Parameters

- **b** – Equation coefficient
- **c** – Equation coefficient
- **tau** – Lag parameter, default: 17
- **initial_values** – numpy array with the initial values of y. Default: np.linspace(0.5,1.5,18)
- **iterations** – number of iterations. Default: 1000

Returns

Rossler chaotic time series

O. E. Rössler, Phys. Lett. 57A, 397 (1976).

dx/dt = -z - y dy/dt = x + ay dz/dt = b + z(x - c)

pyFTS.data.rossler.**get_data**(*var: str, a: float = 0.2, b: float = 0.2, c: float = 5.7, dt: float = 0.01, initial_values: numpy.ndarray = [0.001, 0.001, 0.001], iterations: int = 5000*) → numpy.ndarray

Get a simple univariate time series data.

Parameters **var** – the dataset field name to extract

Returns numpy array

pyFTS.data.rossler.**get_dataframe**(*a: float = 0.2, b: float = 0.2, c: float = 5.7, dt: float = 0.01, initial_values: numpy.ndarray = [0.001, 0.001, 0.001], iterations: int = 5000*) → pandas.core.frame.DataFrame

Return a dataframe with the multivariate Rössler Map time series (x, y, z).

Parameters

- **a** – Equation coefficient. Default value: 0.2
- **b** – Equation coefficient. Default value: 0.2
- **c** – Equation coefficient. Default value: 5.7
- **dt** – Time differential for continuous time integration. Default value: 0.01
- **initial_values** – numpy array with the initial values of x,y and z. Default: [0.001, 0.001, 0.001]
- **iterations** – number of iterations. Default: 5000

Returns Panda dataframe with the x, y and z values

Sunspots dataset

Monthly sunspot numbers from 1749 to May 2016

Source: https://www.esrl.noaa.gov/psd/gcos_wgsp/Timeseries/SUNSPOT/

pyFTS.data.sunspots.**get_data**() → numpy.ndarray

Get a simple univariate time series data.

Returns numpy array

pyFTS.data.sunspots.**get_dataframe**() → pandas.core.frame.DataFrame

Get the complete multivariate time series data.

Returns Pandas DataFrame

pyFTS.distributed package

Module contents

Submodules

pyFTS.distributed.dispy module

pyFTS.distributed.spark module

pyFTS.hyperparam package

Module contents

Submodules

pyFTS.hyperparam.Util module

Common facilities for hyperparameter optimization

pyFTS.hyperparam.Util.**create_hyperparam_tables**(*conn*)
Create a sqlite3 table designed to store benchmark results.

Parameters **conn** – a sqlite3 database connection

pyFTS.hyperparam.Util.**insert_hyperparam**(*data*, *conn*)
Insert benchmark data on database

Parameters **data** – a tuple with the benchmark data with format:

Dataset: Identify on which dataset the dataset was performed Tag: a user defined word that indentify a benchmark set Model: FTS model Transformation: The name of data transformation, if one was used mf: membership function Order: the order of the FTS method Partitioner: UoD partitioning scheme Partitions: Number of partitions alpha: alpha cut lags: lags Measure: accuracy measure Value: the measure value

Parameters **conn** – a sqlite3 database connection

Returns

pyFTS.hyperparam.Util.**open_hyperparam_db**(*name*)
Open a connection with a Sqlite database designed to store benchmark results.

Parameters **name** – database filenem

Returns a sqlite3 database connection

pyFTS.hyperparam.GridSearch module

pyFTS.hyperparam.Evolutionary module

pyFTS.models package

Module contents

Fuzzy Time Series methods

Subpackages

pyFTS.models.ensemble package

Submodules

pyFTS.models.ensemble.ensemble module

pyFTS.models.ensemble.multiseasonal module

Module contents

Meta FTS that aggregates other FTS methods

pyFTS.models.incremental package

Module contents

FTS methods with incremental/online learning

Submodules

pyFTS.models.incremental.TimeVariant module

pyFTS.models.incremental.IncrementalEnsemble module

pyFTS.models.multivariate package

Module contents

Multivariate Fuzzy Time Series methods

Submodules

pyFTS.models.multivariate.FLR module

class pyFTS.models.multivariate.FLR.**FLR**
Bases: object

Multivariate Fuzzy Logical Relationship

set_lhs(*var*, *set*)

set_rhs(*set*)

pyFTS.models.multivariate.common module

class pyFTS.models.multivariate.common.**MultivariateFuzzySet**(***kwargs*)
Bases: *pyFTS.common.Composite.FuzzySet*

Multivariate Composite Fuzzy Set

append_set(*variable*, *set*)
Appends a new fuzzy set from a new variable

Parameters

- **variable** – an multivariate.variable instance
- **set** – an common.FuzzySet instance

membership(*x*)
Calculate the membership value of a given input

Parameters **x** – input value

Returns membership value of x at this fuzzy set

set_target_variable(*variable*)

pyFTS.models.multivariate.common.**fuzzyfy_instance**(*data_point*, *var*, *tuples=True*)

pyFTS.models.multivariate.common.**fuzzyfy_instance_clustered**(*data_point*, *cluster*, ***kwargs*)

pyFTS.models.multivariate.variable module

pyFTS.models.multivariate.flrg module

class pyFTS.models.multivariate.flrg.**FLRG**(***kwargs*)
Bases: *pyFTS.common.flrg.FLRG*

Multivariate Fuzzy Logical Rule Group

append_rhs(*fset*, ***kwargs*)

get_lower(*sets*)
Returns the lower bound value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns lower bound value

get_membership(*data*, *variables*)
Returns the membership value of the FLRG for the input data

Parameters

- **data** – input data
- **sets** – fuzzy sets

Returns the membership value

get_upper(*sets*)
Returns the upper bound value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns upper bound value

set_lhs(*var*, *fset*)

pyFTS.models.multivariate.partitioner module

class pyFTS.models.multivariate.partitioner.**MultivariatePartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

Base class for partitioners which use the MultivariateFuzzySet

append(*fset*)

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

build_index()

change_target_variable(*variable*)

format_data(*data*)

fuzzyfy(*data*, ***kwargs*)
Fuzzyfy the input data according to this partitioner fuzzy sets.

Parameters

- **data** – input value to be fuzzyfied
- **alpha_cut** – the minimal membership value to be considered on fuzzyfication (only for mode='sets')
- **method** – the fuzzyfication method (fuzzy: all fuzzy memberships, maximum: only the maximum membership)
- **mode** – the fuzzyfication mode (sets: return the fuzzy sets names, vector: return a vector with the membership

values for all fuzzy sets, both: return a list with tuples (fuzzy set, membership value))

:returns a list with the fuzzyfied values, depending on the mode

prune()

search(*data*, ***kwargs*)
Perform a search for the nearest fuzzy sets of the point 'data'. This function were designed to work with several overlapped fuzzy sets.

Parameters

- **data** – the value to search for the nearest fuzzy sets
- **type** – the return type: 'index' for the fuzzy set indexes or 'name' for fuzzy set names.

Returns a list with the nearest fuzzy sets

pyFTS.models.multivariate.grid module

class pyFTS.models.multivariate.grid.**GridCluster**(***kwargs*)
Bases: *pyFTS.models.multivariate.partitioner.MultivariatePartitioner*

A cartesian product of all fuzzy sets of all variables

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

defuzzyfy(*values*, *mode='both'*)

class pyFTS.models.multivariate.grid.**IncrementalGridCluster**(***kwargs*)
Bases: *pyFTS.models.multivariate.partitioner.MultivariatePartitioner*

Create combinations of fuzzy sets of the variables on demand, incrementally increasing the multivariate fuzzy set base.

fuzzyfy(*data*, ***kwargs*)
Fuzzyfy the input data according to this partitioner fuzzy sets.

Parameters

- **data** – input value to be fuzzyfied
- **alpha_cut** – the minimal membership value to be considered on fuzzyfication (only for mode='sets')
- **method** – the fuzzyfication method (fuzzy: all fuzzy memberships, maximum: only the maximum membership)
- **mode** – the fuzzyfication mode (sets: return the fuzzy sets names, vector: return a vector with the membership

values for all fuzzy sets, both: return a list with tuples (fuzzy set, membership value))

:returns a list with the fuzzyfied values, depending on the mode

incremental_search(*data*, ***kwargs*)

prune()

pyFTS.models.multivariate.mvfts module

pyFTS.models.multivariate.wmvfts module

pyFTS.models.multivariate.cmvfts module

pyFTS.models.multivariate.granular module

pyFTS.models.nonstationary package

Submodules

pyFTS.models.nonstationary.common module

Non Stationary Fuzzy Sets

GARIBALDI, Jonathan M.; JAROSZEWSKI, Marcin; MUSIKASUWAN, Salang. Nonstationary fuzzy sets. IEEE Transactions on Fuzzy Systems, v. 16, n. 4, p. 1072-1086, 2008.

class pyFTS.models.nonstationary.common.**FuzzySet**(*name*, *mf*, *parameters*, ***kwargs*)
Bases: *pyFTS.common.FuzzySet.FuzzySet*

Non Stationary Fuzzy Sets

get_lower(*t*)

get_midpoint(*t*)

get_upper(*t*)

location = None
Pertubation function that affects the location of the membership function

location_params = None
Parameters for location pertubation function

membership(*x*, *t*)
Calculate the membership value of a given input

Parameters

- **x** – input value
- **t** – time displacement or perturbation parameters

Returns membership value of x at this fuzzy set

noise = None
Pertubation function that adds noise on the membership function

noise_params = None
Parameters for noise pertubation function

perform_location(*t*, *param*)

perform_width(*t*, *param*)

perturbate_parameters(*t*)

width = None
Pertubation function that affects the width of the membership function

width_params = None
Parameters for width pertubation function

pyFTS.models.nonstationary.common.**check_bounds**(*data*, *partitioner*, *t*)

pyFTS.models.nonstationary.common.**check_bounds_index**(*data*, *partitioner*, *t*)

pyFTS.models.nonstationary.common.**fuzzify**(*inst*, *t*, *fuzzySets*)
Calculate the membership values for a data point given nonstationary fuzzy sets

Parameters

- **inst** – data points
- **t** – time displacement of the instance
- **fuzzySets** – list of fuzzy sets

Returns array of membership values

pyFTS.models.nonstationary.common.**fuzzySeries**(*data*, *fuzzySets*, *ordered_sets*, *window_size=1*, *method='fuzzy'*, *const_t=None*)

pyFTS.models.nonstationary.common.**window_index**(*t*, *window_size*)

pyFTS.models.nonstationary.cvfts module

pyFTS.models.nonstationary.flrg module

class pyFTS.models.nonstationary.flrg.**NonStationaryFLRG**(*LHS*, ***kwargs*)
Bases: *pyFTS.common.flrg.FLRG*

get_key()
Returns a unique identifier for this FLRG

get_lower(**args*)
Returns the lower bound value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns lower bound value

get_membership(*data*, **args*)
Returns the membership value of the FLRG for the input data

Parameters

- **data** – input data
- **sets** – fuzzy sets

Returns the membership value

get_midpoint(**args*)
Returns the midpoint value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns the midpoint value

get_upper(**args*)
Returns the upper bound value for the RHS fuzzy sets

Parameters **sets** – fuzzy sets

Returns upper bound value

unpack_args(**args*)

pyFTS.models.nonstationary.honsfts module

pyFTS.models.nonstationary.nsfts module

pyFTS.models.nonstationary.partitioners module

class pyFTS.models.nonstationary.partitioners.**PolynomialNonStationaryPartitioner**(*data*, *part*, ***kwargs*)

Bases: *pyFTS.partitioners.partitioner.Partitioner*

Non Stationary Universe of Discourse Partitioner

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

get_polynomial_perturbations(*data*, ***kwargs*)

poly_width(*par1*, *par2*, *rng*, *deg*)

scale_down(*x*, *pct*)

scale_up(*x*, *pct*)

class pyFTS.models.nonstationary.partitioners.**SimpleNonStationaryPartitioner**(*data*, *part*, ***kwargs*)

Bases: *pyFTS.partitioners.partitioner.Partitioner*

Non Stationary Universe of Discourse Partitioner

build(*data*)

Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

pyFTS.models.nonstationary.partitioners.**simplenonstationary_gridpartitioner_builder**(*data*, *npart*, *transformation*)

pyFTS.models.nonstationary.perturbation module

Pertubation functions for Non Stationary Fuzzy Sets

pyFTS.models.nonstationary.perturbation.**exponential**(*x*, *parameters*)

pyFTS.models.nonstationary.perturbation.**linear**(*x*, *parameters*)

pyFTS.models.nonstationary.perturbation.**periodic**(*x*, *parameters*)

pyFTS.models.nonstationary.perturbation.**polynomial**(*x*, *parameters*)

pyFTS.models.nonstationary.util module

Module contents

Fuzzy time series with nonstationary fuzzy sets, for heteroskedastic data

pyFTS.models.seasonal package

Submodules

pyFTS.models.seasonal.SeasonalIndexer module

class pyFTS.models.seasonal.SeasonalIndexer.**DataFrameSeasonalIndexer**(*index_fields*, *index_seasons*, *data_field*, ***kwargs*)

Bases: *pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer*

Use the Pandas.DataFrame index position to index the seasonality

get_data(*data*)

get_data_by_season(*data*, *indexes*)

get_index_by_season(*indexes*)

get_season_by_index(*index*)

get_season_of_data(*data*)

set_data(*data*, *value*)

class pyFTS.models.seasonal.SeasonalIndexer.**DateTimeSeasonalIndexer**(*date_field*, *index_fields*, *index_seasons*, *data_field*, ***kwargs*)

Bases: *pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer*

Use a Pandas.DataFrame date field to index the seasonality

get_data(*data*)

get_data_by_season(*data*, *indexes*)

get_index(*data*)

get_index_by_season(*indexes*)

get_season_by_index(*index*)

get_season_of_data(*data*)

set_data(*data*, *value*)

class pyFTS.models.seasonal.SeasonalIndexer.**LinearSeasonalIndexer**(*seasons*, *units*, *ignore=None*, ***kwargs*)

Bases: *pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer*

Use the data array/list position to index the seasonality

get_data(*data*)

get_index_by_season(*indexes*)

get_season_by_index(*index*)

get_season_of_data(*data*)

class pyFTS.models.seasonal.SeasonalIndexer.**SeasonalIndexer**(*num_seasons*, ***kwargs*)

Bases: `object`

Seasonal Indexer. Responsible to find the seasonal index of a data point inside its data set

get_data(*data*)

get_data_by_season(*data*, *indexes*)

get_index(*data*)

get_index_by_season(*indexes*)

get_season_by_index(*inde*)

get_season_of_data(*data*)

pyFTS.models.seasonal.cmsfts module

pyFTS.models.seasonal.common module

class pyFTS.models.seasonal.common.**DateTime**

Bases: enum.Enum

Data and Time granularity for time granularity and seasonality identification

day_of_month = 30

day_of_week = 7

day_of_year = 364

half = 2

hour = 24

hour_of_day = 24

hour_of_month = 744

hour_of_week = 168

hour_of_year = 8736

minute = 60

minute_of_day = 1440

minute_of_hour = 60

minute_of_month = 44640

minute_of_week = 10080

minute_of_year = 524160

month = 12

quarter = 4

second = 60

second_of_day = 86400

second_of_hour = 3600

second_of_minute = 60.00001

sixth = 6

third = 3

year = 1

class pyFTS.models.seasonal.common.**FuzzySet**(*datepart*, *name*, *mf*, *parameters*, *centroid*, *alpha=1.0*, ***kwargs*)

Bases: *pyFTS.common.FuzzySet.FuzzySet*

Temporal/Seasonal Fuzzy Set

transform(*x*)

Preprocess the data point for non native types

Parameters **x** –

Returns return a native type value for the structured type

pyFTS.models.seasonal.common.**strip_datepart**(*date*, *date_part*, *mask=''*)

pyFTS.models.seasonal.msfts module

pyFTS.models.seasonal.partitioner module

class pyFTS.models.seasonal.partitioner.**TimeGridPartitioner**(***kwargs*)

Bases: *pyFTS.partitioners.partitioner.Partitioner*

Even Length DateTime Grid Partitioner

build(*data*)

Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

build_index()

extractor(*x*)

Extract a single primitive type from an structured instance

mask = None

A string with datetime formating mask

plot(*ax*)

Plot the :param ax: :return:

search(*data*, ***kwargs*)

Perform a search for the nearest fuzzy sets of the point 'data'. This function were designed to work with several overlapped fuzzy sets.

Parameters

- **data** – the value to search for the nearest fuzzy sets
- **type** – the return type: 'index' for the fuzzy set indexes or 'name' for fuzzy set names.
- **results** – the number of nearest fuzzy sets to return

Returns a list with the nearest fuzzy sets

season = None

Seasonality, a pyFTS.models.seasonal.common.DateTime object

pyFTS.models.seasonal.sfts module

Module contents

Submodules

pyFTS.models.song module

pyFTS.models.chen module

pyFTS.models.yu module

pyFTS.models.cheng module

pyFTS.models.hofts module

pyFTS.models.hwang module

pyFTS.models.ifts module

pyFTS.models.ismailefendi module

pyFTS.models.pwfts module

pyFTS.models.sadaei module

pyFTS.partitioners package

Module contents

Module for pyFTS Universe of Discourse partitioners.

Submodules

pyFTS.partitioners.partitioner module

class pyFTS.partitioners.partitioner.**Partitioner**(***kwargs*)

Bases: object

Universe of Discourse partitioner. Split data on several fuzzy sets

build(*data*)

Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

build_index()

check_bounds(*data*)

Check if the input data is outside the known Universe of Discourse and, if it is, round it to the closest fuzzy set.

Parameters **data** – input data to be verified

Returns the index of the closest fuzzy set when data is outside de universe of discourse or None if

the data is inside the UoD.

defuzzyfy(*values*, *mode='both'*)

extractor(*x*)

Extract a single primitive type from an structured instance

fuzzyfy(*data*, ***kwargs*)
Fuzzyfy the input data according to this partitioner fuzzy sets.

Parameters

- **data** – input value to be fuzzyfied
- **alpha_cut** – the minimal membership value to be considered on fuzzyfication (only for mode='sets')
- **method** – the fuzzyfication method (fuzzy: all fuzzy memberships, maximum: only the maximum membership)
- **mode** – the fuzzyfication mode (sets: return the fuzzy sets names, vector: return a vector with the membership

values for all fuzzy sets, both: return a list with tuples (fuzzy set, membership value))

:returns a list with the fuzzyfied values, depending on the mode

get_name(*counter*)
Find the name of the fuzzy set given its counter id.

Parameters **counter** – The number of the fuzzy set

Returns String

kdtree = None
A spatial index to help in fuzzyfication

lower_margin = None
Specific lower exceeding margins for the known UoD. The default value is the self.margin parameter

lower_set()
Return the fuzzy set on lower bound of the universe of discourse.

Returns Fuzzy Set

margin = None
The upper and lower exceeding margins for the known UoD. The default value is .1

membership_function = None
Fuzzy membership function (pyFTS.common.Membership)

name = None
partitioner name

ordered_sets = None
A ordered list of the fuzzy sets names, sorted by their middle point

partitions = None
The number of universe of discourse partitions, i.e., the number of fuzzy sets that will be created

plot(*ax*, *rounding=0*)
Plot the partitioning using the Matplotlib axis ax

Parameters **ax** – Matplotlib axis

plot_set(*ax*, *s*)
Plot an isolate fuzzy set on Matplotlib axis

Parameters

- **ax** – Matplotlib axis
- **s** – Fuzzy Set

prefix = None
prefix of auto generated partition names

search(*data*, ***kwargs*)
Perform a search for the nearest fuzzy sets of the point 'data'. This function were designed to work with several overlapped fuzzy sets.

Parameters

- **data** – the value to search for the nearest fuzzy sets
- **type** – the return type: 'index' for the fuzzy set indexes or 'name' for fuzzy set names.
- **results** – the number of nearest fuzzy sets to return

Returns a list with the nearest fuzzy sets

setnames = None
list of partitions names. If None is given the partitions will be auto named with prefix

sets = None
The fuzzy sets dictionary

transformation = None
data transformation to be applied on data

type = None
The type of fuzzy sets that are generated by this partitioner

upper_margin = None
Specific upper exceeding margins for the known UoD. The default value is the self.margin parameter

upper_set()
Return the fuzzy set on upper bound of the universe of discourse.

Returns Fuzzy Set

variable = None
In a multivariate context, the variable that contains this partitioner

pyFTS.partitioners.Class module

Class Partitioner with Singleton Fuzzy Sets

class pyFTS.partitioners.Class.**ClassPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

Class Partitioner: Given a dictionary with class/values pairs, create singleton fuzzy sets for each class

build(*data: list*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

pyFTS.partitioners.CMeans module

class pyFTS.partitioners.CMeans.**CMeansPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

pyFTS.partitioners.CMeans.**c_means**(*k*, *dados*, *tam*)

pyFTS.partitioners.CMeans.**distance**(*x*, *y*)

pyFTS.partitioners.Entropy module

C. H. Cheng, R. J. Chang, and C. A. Yeh, "Entropy-based and trapezoidal fuzzification-based fuzzy time series approach for forecasting IT project cost," Technol. Forecast. Social Change, vol. 73, no. 5, pp. 524–542, Jun. 2006.

class pyFTS.partitioners.Entropy.**EntropyPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

Huarng Entropy Partitioner

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

pyFTS.partitioners.Entropy.**PMF**(*data*, *threshold*)

pyFTS.partitioners.Entropy.**bestSplit**(*data*, *npart*)

pyFTS.partitioners.Entropy.**entropy**(*data*, *threshold*)

pyFTS.partitioners.Entropy.**informationGain**(*data*, *thres1*, *thres2*)

pyFTS.partitioners.Entropy.**splitAbove**(*data*, *threshold*)

pyFTS.partitioners.Entropy.**splitBelow**(*data*, *threshold*)

pyFTS.partitioners.FCM module

S. T. Li, Y. C. Cheng, and S. Y. Lin, "A FCM-based deterministic forecasting model for fuzzy time series," Comput. Math. Appl., vol. 56, no. 12, pp. 3052–3063, Dec. 2008. DOI: 10.1016/j.camwa.2008.07.033.

class pyFTS.partitioners.FCM.**FCMPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

pyFTS.partitioners.FCM.**fuzzy_cmeans**(*k*, *data*, *size*, *m*, *deltadist=0.001*)

pyFTS.partitioners.FCM.**fuzzy_distance**(*x*, *y*)

pyFTS.partitioners.FCM.**membership**(*val*, *vals*)

pyFTS.partitioners.Grid module

Even Length Grid Partitioner

class pyFTS.partitioners.Grid.**GridPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

Even Length Grid Partitioner

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

class pyFTS.partitioners.Grid.**PreFixedGridPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.Grid.GridPartitioner*

Prefixed UoD with Even Length Grid Partitioner

pyFTS.partitioners.Huarng module

pyFTS.partitioners.Singleton module

Even Length Grid Partitioner

class pyFTS.partitioners.Singleton.**SingletonPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

Singleton Partitioner: Create singleton fuzzy sets for each distinct value in UoD

build(*data: list*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

pyFTS.partitioners.Simple module

Simple Partitioner for manually informed fuzzy sets

class pyFTS.partitioners.Simple.**SimplePartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

Simple Partitioner for manually informed fuzzy sets

append(*name*, *mf*, *parameters*, ***kwargs*)
Append a new partition (fuzzy set) to the partitioner

Parameters

- **name** – Fuzzy set name
- **mf** – One of the pyFTS.common.Membership functions
- **parameters** – A list with the parameters for the membership function
- **kwargs** – Optional arguments for the fuzzy set

append_complex(*fs*)

pyFTS.partitioners.SubClust module

Chiu, Stephen L. "Fuzzy model identification based on cluster estimation." Journal of Intelligent & fuzzy systems 2.3 (1994): 267-278.

class pyFTS.partitioners.SubClust.**SubClustPartitioner**(***kwargs*)
Bases: *pyFTS.partitioners.partitioner.Partitioner*

Subtractive Clustering Partitioner

build(*data*)
Perform the partitioning of the Universe of Discourse

Parameters **data** – training data

Returns

pyFTS.partitioners.SubClust.**imax**(*vec*)

pyFTS.partitioners.SubClust.**subclust**(*data*, *ra*, *rb*, *eps_sup*, *eps_inf*)

pyFTS.partitioners.Util module

pyFTS.partitioners.parallel_util module

pyFTS.probabilistic package

Module contents

Probability Distribution objects

Submodules

pyFTS.probabilistic.ProbabilityDistribution module

pyFTS.probabilistic.kde module

Submodules

pyFTS.conf module

Module contents

pyFTS - A Python library for Fuzzy Time Series models

CHAPTER 2

How to reference pyFTS?

Silva, P. C. L. et al. *pyFTS: Fuzzy Time Series for Python.* Belo Horizonte. 2018. DOI: 10.5281/zenodo.597359. Url: <https://doi.org/10.5281/zenodo.597359>

CHAPTER 3

Indexes

- genindex
- modindex
- search

Python Module Index

p

pyFTS, 45
pyFTS.benchmarks, 2
pyFTS.common, 3
pyFTS.common.Activations, 3
pyFTS.common.Composite, 3
pyFTS.common.FLR, 4
pyFTS.common.flrg, 11
pyFTS.common.FuzzySet, 5
pyFTS.common.Membership, 8
pyFTS.common.SortedCollection, 9
pyFTS.common.transformations, 12
pyFTS.common.transformations.boxcox, 12
pyFTS.common.transformations.differential, 13
pyFTS.common.transformations.normalization, 13
pyFTS.common.transformations.roi, 14
pyFTS.common.transformations.scale, 14
pyFTS.common.transformations.smoothing, 15
pyFTS.common.transformations.som, 16
pyFTS.common.transformations.transformation, 17
pyFTS.common.tree, 12
pyFTS.conf, 45
pyFTS.data, 18
pyFTS.data.AirPassengers, 21
pyFTS.data.artificial, 18
pyFTS.data.Bitcoin, 22
pyFTS.data.common, 18
pyFTS.data.DowJones, 22
pyFTS.data.Enrollments, 22
pyFTS.data.Ethereum, 22
pyFTS.data.EURGBP, 23
pyFTS.data.EURUSD, 23
pyFTS.data.GBPUSD, 23
pyFTS.data.henon, 26
pyFTS.data.INMET, 24
pyFTS.data.logistic_map, 26
pyFTS.data.lorentz, 26
pyFTS.data.mackey_glass, 27
pyFTS.data.Malaysia, 24
pyFTS.data.NASDAQ, 24
pyFTS.data.rossler, 27
pyFTS.data.SONDA, 25
pyFTS.data.SP500, 25
pyFTS.data.sunspots, 28
pyFTS.data.TAIEX, 25
pyFTS.distributed, 28
pyFTS.hyperparam, 29
pyFTS.hyperparam.Util, 29
pyFTS.models, 29
pyFTS.models.ensemble, 30
pyFTS.models.incremental, 30
pyFTS.models.multivariate, 30
pyFTS.models.multivariate.common, 30
pyFTS.models.multivariate.FLR, 30
pyFTS.models.multivariate.flrg, 31
pyFTS.models.multivariate.grid, 32
pyFTS.models.multivariate.partitioner, 31
pyFTS.models.nonstationary, 36
pyFTS.models.nonstationary.common, 33
pyFTS.models.nonstationary.flrg, 34
pyFTS.models.nonstationary.partitioners, 35
pyFTS.models.nonstationary.perturbation, 36
pyFTS.models.seasonal, 39
pyFTS.models.seasonal.common, 38
pyFTS.models.seasonal.partitioner, 39
pyFTS.models.seasonal.SeasonalIndexer, 36
pyFTS.partitioners, 40
pyFTS.partitioners.Class, 42
pyFTS.partitioners.CMeans, 42
pyFTS.partitioners.Entropy, 43
pyFTS.partitioners.FCM, 43

pyFTS.partitioners.Grid, 44
pyFTS.partitioners.partitioner, 40
pyFTS.partitioners.Simple, 44
pyFTS.partitioners.Singleton, 44
pyFTS.partitioners.SubClust, 45
pyFTS.probabilistic, 45

Index

A

alpha (*pyFTS.common.FuzzySet.FuzzySet attribute*), 5
append() (*pyFTS.common.Composite.FuzzySet method*), 3
append() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 31
append() (*pyFTS.partitioners.Simple.SimplePartitioner method*), 44
append_complex() (*pyFTS.partitioners.Simple.SimplePartitioner method*), 44
append_rhs() (*pyFTS.common.flrg.FLRG method*), 11
append_rhs() (*pyFTS.models.multivariate.flrg.FLRG method*), 31
append_set() (*pyFTS.common.Composite.FuzzySet method*), 4
append_set() (*pyFTS.models.multivariate.common.MultivariateFuzzySet method*), 30
appendChild() (*pyFTS.common.tree.FLRGTreeNode method*), 12
apply() (*pyFTS.common.transformations.boxcox.BoxCox method*), 12
apply() (*pyFTS.common.transformations.differential.Differential method*), 13
apply() (*pyFTS.common.transformations.normalization.Normalization method*), 13
apply() (*pyFTS.common.transformations.roi.ROI method*), 14
apply() (*pyFTS.common.transformations.scale.Scale method*), 14
apply() (*pyFTS.common.transformations.smoothing.AveragePooling method*), 15
apply() (*pyFTS.common.transformations.smoothing.ExponentialSmoothing method*), 15
apply() (*pyFTS.common.transformations.smoothing.MaxPooling method*), 16
apply() (*pyFTS.common.transformations.smoothing.MovingAverage method*), 16
apply() (*pyFTS.common.transformations.som.SOMTransformation method*), 17
apply() (*pyFTS.common.transformations.transformation.Transformation method*), 17
argmax() (*in module pyFTS.common.Activations*), 3
around() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
AveragePooling (*class in pyFTS.common.transformations.smoothing*), 15

B

bellmf() (*in module pyFTS.common.Membership*), 8
bestSplit() (*in module pyFTS.partitioners.Entropy*), 43
between() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
blip() (*pyFTS.data.artificial.SignalEmulator method*), 18
BoxCox (*class in pyFTS.common.transformations.boxcox*), 12
build() (*pyFTS.models.multivariate.grid.GridCluster method*), 32
build() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 31
build() (*pyFTS.models.nonstationary.partitioners.PolynomialNonStationary method*), 35
build() (*pyFTS.models.nonstationary.partitioners.SimpleNonStationaryP method*), 36
build() (*pyFTS.models.seasonal.partitioner.TimeGridPartitioner method*), 39
build() (*pyFTS.partitioners.Class.ClassPartitioner method*), 42
build() (*pyFTS.partitioners.CMeans.CMeansPartitioner method*), 42
build() (*pyFTS.partitioners.Entropy.EntropyPartitioner method*), 43
build() (*pyFTS.partitioners.FCM.FCMPartitioner method*), 43
build() (*pyFTS.partitioners.Grid.GridPartitioner method*), 44

build() (*pyFTS.partitioners.partitioner.Partitioner method*), 40
build() (*pyFTS.partitioners.Singleton.SingletonPartitioner method*), 44
build() (*pyFTS.partitioners.SubClust.SubClustPartitioner method*), 45
build_index() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 32
build_index() (*pyFTS.models.seasonal.partitioner.TimeGridPartitioner method*), 39
build_index() (*pyFTS.partitioners.partitioner.Partitioner method*), 40
build_tree_without_order() (*in module pyFTS.common.tree*), 12

C

c_means() (*in module pyFTS.partitioners.CMeans*), 43
centroid (*pyFTS.common.FuzzySet.FuzzySet attribute*), 5
change_target_variable() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 32
check_bounds() (*in module pyFTS.common.FuzzySet*), 6
check_bounds() (*in module pyFTS.models.nonstationary.common*), 34
check_bounds() (*pyFTS.partitioners.partitioner.Partitioner method*), 40
check_bounds_index() (*in module pyFTS.common.FuzzySet*), 6
check_bounds_index() (*in module pyFTS.models.nonstationary.common*), 34
ClassPartitioner (*class in pyFTS.partitioners.Class*), 42
clear() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
CMeansPartitioner (*class in pyFTS.partitioners.CMeans*), 42
components (*pyFTS.data.artificial.SignalEmulator attribute*), 18
copy() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
count() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
create_hyperparam_tables() (*in module pyFTS.hyperparam.Util*), 29

D

DataFrameSeasonalIndexer (*class in pyFTS.models.seasonal.SeasonalIndexer*), 36
DateTime (*class in pyFTS.models.seasonal.common*), 38
DateTimeSeasonalIndexer (*class in pyFTS.models.seasonal.SeasonalIndexer*), 37
day_of_month (*pyFTS.models.seasonal.common.DateTime attribute*), 38
day_of_week (*pyFTS.models.seasonal.common.DateTime attribute*), 38
day_of_year (*pyFTS.models.seasonal.common.DateTime attribute*), 38
defuzzyfy() (*pyFTS.models.multivariate.grid.GridCluster method*), 32
defuzzyfy() (*pyFTS.partitioners.partitioner.Partitioner method*), 40
Differential (*class in pyFTS.common.transformations.differential*), 13
distance() (*in module pyFTS.partitioners.CMeans*), 43

E

entropy() (*in module pyFTS.partitioners.Entropy*), 43
EntropyPartitioner (*class in pyFTS.partitioners.Entropy*), 43
exponential() (*in module pyFTS.models.nonstationary.perturbation*), 36
ExponentialSmoothing (*class in pyFTS.common.transformations.smoothing*), 15
extractor() (*pyFTS.models.seasonal.partitioner.TimeGridPartitioner method*), 39
extractor() (*pyFTS.partitioners.partitioner.Partitioner method*), 40

F

FCMPartitioner (*class in pyFTS.partitioners.FCM*), 43
find() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
find_ge() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
find_gt() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
find_le() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
find_lt() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
flat() (*in module pyFTS.common.tree*), 12
FLR (*class in pyFTS.common.FLR*), 4
FLR (*class in pyFTS.models.multivariate.FLR*), 30
FLRG (*class in pyFTS.common.flrg*), 11
FLRG (*class in pyFTS.models.multivariate.flrg*), 31
FLRGTree (*class in pyFTS.common.tree*), 12
FLRGTreeNode (*class in pyFTS.common.tree*), 12

format_data() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 32
fuzzify() (*in module pyFTS.models.nonstationary.common*), 34
fuzzy_cmeans() (*in module pyFTS.partitioners.FCM*), 43
fuzzy_distance() (*in module pyFTS.partitioners.FCM*), 43
fuzzyfy() (*in module pyFTS.common.FuzzySet*), 6
fuzzyfy() (*pyFTS.models.multivariate.grid.IncrementalGridCluster method*), 32
fuzzyfy() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 32
fuzzyfy() (*pyFTS.partitioners.partitioner.Partitioner method*), 40
fuzzyfy_instance() (*in module pyFTS.common.FuzzySet*), 6
fuzzyfy_instance() (*in module pyFTS.models.multivariate.common*), 31
fuzzyfy_instance_clustered() (*in module pyFTS.models.multivariate.common*), 31
fuzzyfy_instances() (*in module pyFTS.common.FuzzySet*), 6
fuzzyfy_series() (*in module pyFTS.common.FuzzySet*), 7
fuzzyfy_series_old() (*in module pyFTS.common.FuzzySet*), 7
fuzzySeries() (*in module pyFTS.models.nonstationary.common*), 34
FuzzySet (*class in pyFTS.common.Composite*), 3
FuzzySet (*class in pyFTS.common.FuzzySet*), 5
FuzzySet (*class in pyFTS.models.nonstationary.common*), 33
FuzzySet (*class in pyFTS.models.seasonal.common*), 38

G

gaussmf() (*in module pyFTS.common.Membership*), 8
generate_gaussian_linear() (*in module pyFTS.data.artificial*), 19
generate_high_order_recurrent_flr() (*in module pyFTS.common.FLR*), 4
generate_indexed_flrs() (*in module pyFTS.common.FLR*), 4
generate_linear_periodic_gaussian() (*in module pyFTS.data.artificial*), 20
generate_non_recurrent_flrs() (*in module pyFTS.common.FLR*), 5
generate_recurrent_flrs() (*in module pyFTS.common.FLR*), 5
generate_sinoidal_periodic_gaussian() (*in module pyFTS.data.artificial*), 20
generate_uniform_linear() (*in module pyFTS.data.artificial*), 21
get_data() (*in module pyFTS.data.AirPassengers*), 21
get_data() (*in module pyFTS.data.Bitcoin*), 22
get_data() (*in module pyFTS.data.DowJones*), 22
get_data() (*in module pyFTS.data.Enrollments*), 22
get_data() (*in module pyFTS.data.Ethereum*), 22
get_data() (*in module pyFTS.data.EURGBP*), 23
get_data() (*in module pyFTS.data.EURUSD*), 23
get_data() (*in module pyFTS.data.GBPUSD*), 23
get_data() (*in module pyFTS.data.henon*), 26
get_data() (*in module pyFTS.data.logistic_map*), 26
get_data() (*in module pyFTS.data.lorentz*), 26
get_data() (*in module pyFTS.data.mackey_glass*), 27
get_data() (*in module pyFTS.data.Malaysia*), 24
get_data() (*in module pyFTS.data.NASDAQ*), 24
get_data() (*in module pyFTS.data.rossler*), 27
get_data() (*in module pyFTS.data.SONDA*), 25
get_data() (*in module pyFTS.data.SP500*), 25
get_data() (*in module pyFTS.data.sunspots*), 28
get_data() (*in module pyFTS.data.TAIEX*), 25
get_data() (*pyFTS.models.seasonal.SeasonalIndexer.DataFrameSeasonalIndexer method*), 36
get_data() (*pyFTS.models.seasonal.SeasonalIndexer.DateTimeSeasonalIndexer method*), 37
get_data() (*pyFTS.models.seasonal.SeasonalIndexer.LinearSeasonalIndexer method*), 37
get_data() (*pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer method*), 37
get_data_by_season() (*pyFTS.models.seasonal.SeasonalIndexer.DataFrameSeasonalIndexer method*), 36
get_data_by_season() (*pyFTS.models.seasonal.SeasonalIndexer.DateTimeSeasonalIndexer method*), 37
get_data_by_season() (*pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer method*), 37
get_dataframe() (*in module pyFTS.data.AirPassengers*), 21
get_dataframe() (*in module pyFTS.data.Bitcoin*), 22
get_dataframe() (*in module pyFTS.data.common*), 18
get_dataframe() (*in module pyFTS.data.DowJones*), 22
get_dataframe() (*in module pyFTS.data.Enrollments*), 22
get_dataframe() (*in module pyFTS.data.Ethereum*), 23
get_dataframe() (*in module pyFTS.data.EURGBP*), 23
get_dataframe() (*in module pyFTS.data.EURUSD*), 23
get_dataframe() (*in module pyFTS.data.GBPUSD*),

23
get_dataframe() (in module pyFTS.data.henon), 26
get_dataframe() (in module pyFTS.data.INMET), 24
get_dataframe() (in module pyFTS.data.lorentz), 27
get_dataframe() (in module pyFTS.data.Malaysia), 24
get_dataframe() (in module pyFTS.data.NASDAQ), 24
get_dataframe() (in module pyFTS.data.rossler), 28
get_dataframe() (in module pyFTS.data.SONDA), 25
get_dataframe() (in module pyFTS.data.SP500), 25
get_dataframe() (in module pyFTS.data.sunspots), 28
get_dataframe() (in module pyFTS.data.TAIEX), 25
get_fuzzysets() (in module pyFTS.common.FuzzySet), 7
get_index() (pyFTS.models.seasonal.SeasonalIndexer.DateTimeSeasonalIndexer method), 37
get_index() (pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer method), 37
get_index_by_season() (pyFTS.models.seasonal.SeasonalIndexer.DataFrameSeasonalIndexer method), 36
get_index_by_season() (pyFTS.models.seasonal.SeasonalIndexer.DateTimeSeasonalIndexer method), 37
get_index_by_season() (pyFTS.models.seasonal.SeasonalIndexer.LinearSeasonalIndexer method), 37
get_index_by_season() (pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer method), 37
get_key() (pyFTS.common.flrg.FLRG method), 11
get_key() (pyFTS.models.nonstationary.flrg.NonStationaryFLRG method), 34
get_lower() (pyFTS.common.flrg.FLRG method), 11
get_lower() (pyFTS.models.multivariate.flrg.FLRG method), 31
get_lower() (pyFTS.models.nonstationary.common.FuzzySet method), 33
get_lower() (pyFTS.models.nonstationary.flrg.NonStationaryFLRG method), 34
get_maximum_membership_fuzzyset() (in module pyFTS.common.FuzzySet), 7
get_maximum_membership_fuzzyset_index() (in module pyFTS.common.FuzzySet), 7
get_membership() (pyFTS.common.flrg.FLRG method), 11
get_membership() (pyFTS.models.multivariate.flrg.FLRG method), 31
get_membership() (pyFTS.models.nonstationary.flrg.NonStationaryFLRG method), 35
get_midpoint() (pyFTS.common.flrg.FLRG method), 11
get_midpoint() (pyFTS.models.nonstationary.common.FuzzySet method), 33
get_midpoint() (pyFTS.models.nonstationary.flrg.NonStationaryFLRG method), 35
get_midpoints() (pyFTS.common.flrg.FLRG method), 11
get_name() (pyFTS.partitioners.partitioner.Partitioner method), 41
get_polynomial_perturbations() (pyFTS.models.nonstationary.partitioners.PolynomialNonStationa method), 35
get_season_by_index() (pyFTS.models.seasonal.SeasonalIndexer.DataFrameSeasonalIna method), 36
get_season_by_index() (pyFTS.models.seasonal.SeasonalIndexer.DateTimeSeasonalIndexer method), 37
get_season_by_index() (pyFTS.models.seasonal.SeasonalIndexer.LinearSeasonalIndexer method), 37
get_season_by_index() (pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer method), 37
get_season_of_data() (pyFTS.models.seasonal.SeasonalIndexer.DataFrameSeasonalIna method), 36
get_season_of_data() (pyFTS.models.seasonal.SeasonalIndexer.DateTimeSeasonalIndex method), 37
get_season_of_data() (pyFTS.models.seasonal.SeasonalIndexer.LinearSeasonalIndexer method), 37
get_season_of_data() (pyFTS.models.seasonal.SeasonalIndexer.SeasonalIndexer method), 37
get_upper() (pyFTS.common.flrg.FLRG method), 11
get_upper() (pyFTS.models.multivariate.flrg.FLRG method), 31
get_upper() (pyFTS.models.nonstationary.common.FuzzySet method), 33
get_upper() (pyFTS.models.nonstationary.flrg.NonStationaryFLRG method), 35
getChildren() (pyFTS.common.tree.FLRGTreeNode method), 12
getStr() (pyFTS.common.tree.FLRGTreeNode method), 12
grant_bounds() (in module pyFTS.common.FuzzySet), 7
GridCluster (class in

pyFTS.models.multivariate.grid), 32
GridPartitioner (*class in pyFTS.partitioners.Grid*), 44

H

half (*pyFTS.models.seasonal.common.DateTime attribute*), 38
hour (*pyFTS.models.seasonal.common.DateTime attribute*), 38
hour_of_day (*pyFTS.models.seasonal.common.DateTime attribute*), 38
hour_of_month (*pyFTS.models.seasonal.common.DateTime attribute*), 38
hour_of_week (*pyFTS.models.seasonal.common.DateTime attribute*), 38
hour_of_year (*pyFTS.models.seasonal.common.DateTime attribute*), 38

I

imax() (*in module pyFTS.partitioners.SubClust*), 45
incremental_gaussian() (*pyFTS.data.artificial.SignalEmulator method*), 18
incremental_search() (*pyFTS.models.multivariate.grid.IncrementalGridCluster method*), 33
IncrementalGridCluster (*class in pyFTS.models.multivariate.grid*), 32
index (*pyFTS.common.FLR.IndexedFLR attribute*), 4
index() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
IndexedFLR (*class in pyFTS.common.FLR*), 4
informationGain() (*in module pyFTS.partitioners.Entropy*), 43
insert() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
insert_hyperparam() (*in module pyFTS.hyperparam.Util*), 29
insert_right() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
inside() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
inverse() (*pyFTS.common.transformations.boxcox.BoxCox method*), 12
inverse() (*pyFTS.common.transformations.differential.Differential method*), 13
inverse() (*pyFTS.common.transformations.normalization.Normalization method*), 13
inverse() (*pyFTS.common.transformations.roi.ROI method*), 14
inverse() (*pyFTS.common.transformations.scale.Scale method*), 15
inverse() (*pyFTS.common.transformations.smoothing.AveragePooling method*), 15
inverse() (*pyFTS.common.transformations.smoothing.ExponentialSmoo method*), 15
inverse() (*pyFTS.common.transformations.smoothing.MaxPooling method*), 16
inverse() (*pyFTS.common.transformations.smoothing.MovingAverage method*), 16
inverse() (*pyFTS.common.transformations.transformation.Transformat method*), 17
is_multivariate (*pyFTS.common.transformations.transformation.Tra attribute*), 17

K

kdtree (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41
key (*pyFTS.common.SortedCollection.SortedCollection attribute*), 10

L

LHS (*pyFTS.common.FLR.FLR attribute*), 4
LHS (*pyFTS.common.flrg.FLRG attribute*), 11
linear() (*in module pyFTS.models.nonstationary.perturbation*), 36
LinearSeasonalIndexer (*class in pyFTS.models.seasonal.SeasonalIndexer*), 37
location (*pyFTS.models.nonstationary.common.FuzzySet attribute*), 33
location_params (*pyFTS.models.nonstationary.common.FuzzySet attribute*), 33
lower_margin (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41
lower_set() (*pyFTS.partitioners.partitioner.Partitioner method*), 41

M

margin (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41
mask (*pyFTS.models.seasonal.partitioner.TimeGridPartitioner attribute*), 39
MaxPooling (*class in pyFTS.common.transformations.smoothing*), 16
membership() (*in module pyFTS.partitioners.FCM*), 43
membership() (*pyFTS.common.Composite.FuzzySet method*), 4
membership() (*pyFTS.common.FuzzySet.FuzzySet method*), 5
membership() (*pyFTS.models.multivariate.common.MultivariateFuzzyS method*), 30
membership() (*pyFTS.models.nonstationary.common.FuzzySet method*), 33

membership_function (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41
mf (*pyFTS.common.FuzzySet.FuzzySet attribute*), 5
minute (*pyFTS.models.seasonal.common.DateTime attribute*), 38
minute_of_day (*pyFTS.models.seasonal.common.DateTime attribute*), 38
minute_of_hour (*pyFTS.models.seasonal.common.DateTime attribute*), 38
minute_of_month (*pyFTS.models.seasonal.common.DateTime attribute*), 38
minute_of_week (*pyFTS.models.seasonal.common.DateTime attribute*), 38
minute_of_year (*pyFTS.models.seasonal.common.DateTime attribute*), 38
month (*pyFTS.models.seasonal.common.DateTime attribute*), 38
MovingAverage (*class in pyFTS.common.transformations.smoothing*), 16
MultivariateFuzzySet (*class in pyFTS.models.multivariate.common*), 30
MultivariatePartitioner (*class in pyFTS.models.multivariate.partitioner*), 31

N

name (*pyFTS.common.FuzzySet.FuzzySet attribute*), 5
name (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41
noise (*pyFTS.models.nonstationary.common.FuzzySet attribute*), 34
noise_params (*pyFTS.models.nonstationary.common.FuzzySet attribute*), 34
NonStationaryFLRG (*class in pyFTS.models.nonstationary.flrg*), 34
Normalization (*class in pyFTS.common.transformations.normalization*), 13

O

open_hyperparam_db() (*in module pyFTS.hyperparam.Util*), 29
order (*pyFTS.common.flrg.FLRG attribute*), 11
ordered_sets (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41

P

parameters (*pyFTS.common.FuzzySet.FuzzySet attribute*), 5
parameters (*pyFTS.common.transformations.boxcox.BoxCox attribute*), 13
parameters (*pyFTS.common.transformations.differential.Differential attribute*), 13
parameters (*pyFTS.common.transformations.scale.Scale attribute*), 15
partition_function() (*pyFTS.common.FuzzySet.FuzzySet method*), 5
Partitioner (*class in pyFTS.partitioners.partitioner*), 40
partitions (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41
paths() (*pyFTS.common.tree.FLRGTreeNode method*), 12
perform_location() (*pyFTS.models.nonstationary.common.FuzzySet method*), 34
perform_width() (*pyFTS.models.nonstationary.common.FuzzySet method*), 34
periodic() (*in module pyFTS.models.nonstationary.perturbation*), 36
periodic_gaussian() (*pyFTS.data.artificial.SignalEmulator method*), 19
perturbate_parameters() (*pyFTS.models.nonstationary.common.FuzzySet method*), 34
plot() (*pyFTS.models.seasonal.partitioner.TimeGridPartitioner method*), 39
plot() (*pyFTS.partitioners.partitioner.Partitioner method*), 41
plot_set() (*pyFTS.partitioners.partitioner.Partitioner method*), 41
PMF() (*in module pyFTS.partitioners.Entropy*), 43
poly_width() (*pyFTS.models.nonstationary.partitioners.PolynomialNo method*), 35
polynomial() (*in module pyFTS.models.nonstationary.perturbation*), 36
PolynomialNonStationaryPartitioner (*class in pyFTS.models.nonstationary.partitioners*), 35
prefix (*pyFTS.partitioners.partitioner.Partitioner attribute*), 41
PreFixedGridPartitioner (*class in pyFTS.partitioners.Grid*), 44
prune() (*pyFTS.models.multivariate.grid.IncrementalGridCluster method*), 33
prune() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 32
pyFTS (*module*), 45
pyFTS.benchmarks (*module*), 2
pyFTS.common (*module*), 3
pyFTS.common.Activations (*module*), 3
pyFTS.common.Composite (*module*), 3
pyFTS.common.FLR (*module*), 4
pyFTS.common.flrg (*module*), 11

pyFTS.common.FuzzySet (*module*), 5
pyFTS.common.Membership (*module*), 8
pyFTS.common.SortedCollection (*module*), 9
pyFTS.common.transformations (*module*), 12
pyFTS.common.transformations.boxcox (*module*), 12
pyFTS.common.transformations.differential (*module*), 13
pyFTS.common.transformations.normalization (*module*), 13
pyFTS.common.transformations.roi (*module*), 14
pyFTS.common.transformations.scale (*module*), 14
pyFTS.common.transformations.smoothing (*module*), 15
pyFTS.common.transformations.som (*module*), 16
pyFTS.common.transformations.transformation (*module*), 17
pyFTS.common.tree (*module*), 12
pyFTS.conf (*module*), 45
pyFTS.data (*module*), 18
pyFTS.data.AirPassengers (*module*), 21
pyFTS.data.artificial (*module*), 18
pyFTS.data.Bitcoin (*module*), 22
pyFTS.data.common (*module*), 18
pyFTS.data.DowJones (*module*), 22
pyFTS.data.Enrollments (*module*), 22
pyFTS.data.Ethereum (*module*), 22
pyFTS.data.EURGBP (*module*), 23
pyFTS.data.EURUSD (*module*), 23
pyFTS.data.GBPUSD (*module*), 23
pyFTS.data.henon (*module*), 26
pyFTS.data.INMET (*module*), 24
pyFTS.data.logistic_map (*module*), 26
pyFTS.data.lorentz (*module*), 26
pyFTS.data.mackey_glass (*module*), 27
pyFTS.data.Malaysia (*module*), 24
pyFTS.data.NASDAQ (*module*), 24
pyFTS.data.rossler (*module*), 27
pyFTS.data.SONDA (*module*), 25
pyFTS.data.SP500 (*module*), 25
pyFTS.data.sunspots (*module*), 28
pyFTS.data.TAIEX (*module*), 25
pyFTS.distributed (*module*), 28
pyFTS.hyperparam (*module*), 29
pyFTS.hyperparam.Util (*module*), 29
pyFTS.models (*module*), 29
pyFTS.models.ensemble (*module*), 30
pyFTS.models.incremental (*module*), 30
pyFTS.models.multivariate (*module*), 30
pyFTS.models.multivariate.common (*module*), 30
pyFTS.models.multivariate.FLR (*module*), 30
pyFTS.models.multivariate.flrg (*module*), 31
pyFTS.models.multivariate.grid (*module*), 32
pyFTS.models.multivariate.partitioner (*module*), 31
pyFTS.models.nonstationary (*module*), 36
pyFTS.models.nonstationary.common (*module*), 33
pyFTS.models.nonstationary.flrg (*module*), 34
pyFTS.models.nonstationary.partitioners (*module*), 35
pyFTS.models.nonstationary.perturbation (*module*), 36
pyFTS.models.seasonal (*module*), 39
pyFTS.models.seasonal.common (*module*), 38
pyFTS.models.seasonal.partitioner (*module*), 39
pyFTS.models.seasonal.SeasonalIndexer (*module*), 36
pyFTS.partitioners (*module*), 40
pyFTS.partitioners.Class (*module*), 42
pyFTS.partitioners.CMeans (*module*), 42
pyFTS.partitioners.Entropy (*module*), 43
pyFTS.partitioners.FCM (*module*), 43
pyFTS.partitioners.Grid (*module*), 44
pyFTS.partitioners.partitioner (*module*), 40
pyFTS.partitioners.Simple (*module*), 44
pyFTS.partitioners.Singleton (*module*), 44
pyFTS.partitioners.SubClust (*module*), 45
pyFTS.probabilistic (*module*), 45

Q

quarter (*pyFTS.models.seasonal.common.DateTime attribute*), 38

R

random_walk() (*in module pyFTS.data.artificial*), 21
remove() (*pyFTS.common.SortedCollection.SortedCollection method*), 10
reset_calculated_values() (*pyFTS.common.flrg.FLRG method*), 11
RHS (*pyFTS.common.FLR.FLR attribute*), 4
RHS (*pyFTS.common.flrg.FLRG attribute*), 11
ROI (*class in pyFTS.common.transformations.roi*), 14
run() (*pyFTS.data.artificial.SignalEmulator method*), 19

S

save_net() (*pyFTS.common.transformations.som.SOMTransformation method*), 17

Scale (*class in pyFTS.common.transformations.scale*), 14
scale() (*in module pyFTS.common.Activations*), 3
scale_down() (*pyFTS.models.nonstationary.partitioners.PolynomialNonStationaryPartitioner method*), 35
scale_up() (*pyFTS.models.nonstationary.partitioners.PolynomialNonStationaryPartitioner method*), 35
search() (*pyFTS.models.multivariate.partitioner.MultivariatePartitioner method*), 32
search() (*pyFTS.models.seasonal.partitioner.TimeGridPartitioner method*), 39
search() (*pyFTS.partitioners.partitioner.Partitioner method*), 42
season (*pyFTS.models.seasonal.partitioner.TimeGridPartitioner attribute*), 39
SeasonalIndexer (*class in pyFTS.models.seasonal.SeasonalIndexer*), 37
second (*pyFTS.models.seasonal.common.DateTime attribute*), 38
second_of_day (*pyFTS.models.seasonal.common.DateTime attribute*), 38
second_of_hour (*pyFTS.models.seasonal.common.DateTime attribute*), 38
second_of_minute (*pyFTS.models.seasonal.common.DateTime attribute*), 38
set_data() (*pyFTS.models.seasonal.SeasonalIndexer.DataFrameSeasonalIndexer method*), 37
set_data() (*pyFTS.models.seasonal.SeasonalIndexer.DateTimeSeasonalIndexer method*), 37
set_lhs() (*pyFTS.models.multivariate.FLR.FLR method*), 30
set_lhs() (*pyFTS.models.multivariate.flrg.FLRG method*), 31
set_ordered() (*in module pyFTS.common.FuzzySet*), 7
set_rhs() (*pyFTS.models.multivariate.FLR.FLR method*), 30
set_target_variable() (*pyFTS.models.multivariate.common.MultivariateFuzzySet method*), 31
setnames (*pyFTS.partitioners.partitioner.Partitioner attribute*), 42
sets (*pyFTS.partitioners.partitioner.Partitioner attribute*), 42
show_grid() (*pyFTS.common.transformations.som.SOMTransformation method*), 17
sigmf() (*in module pyFTS.common.Membership*), 8
SignalEmulator (*class in pyFTS.data.artificial*), 18
simplenonstationary_gridpartitioner_builder() (*in module pyFTS.models.nonstationary.partitioners*), 36
SimpleNonStationaryPartitioner (*class in pyFTS.models.nonstationary.partitioners*), 35
SimplePartitioner (*class in pyFTS.partitioners.Simple*), 44
singleton() (*in module pyFTS.common.Membership*), 8
SingletonPartitioner (*class in pyFTS.partitioners.Singleton*), 44
sixth (*pyFTS.models.seasonal.common.DateTime attribute*), 38
softmax() (*in module pyFTS.common.Activations*), 3
SOMTransformation (*class in pyFTS.common.transformations.som*), 16
SortedCollection (*class in pyFTS.common.SortedCollection*), 9
splitAbove() (*in module pyFTS.partitioners.Entropy*), 43
splitBelow() (*in module pyFTS.partitioners.Entropy*), 43
stationary_gaussian() (*pyFTS.data.artificial.SignalEmulator method*), 19
strip_datepart() (*in module pyFTS.models.seasonal.common*), 39
subclust() (*in module pyFTS.partitioners.SubClust*), 45
SubClustPartitioner (*class in pyFTS.partitioners.SubClust*), 45

T

third (*pyFTS.models.seasonal.common.DateTime attribute*), 38
TimeGridPartitioner (*class in pyFTS.models.seasonal.partitioner*), 39
train() (*pyFTS.common.transformations.normalization.Normalization method*), 14
train() (*pyFTS.common.transformations.som.SOMTransformation method*), 17
transform() (*pyFTS.common.Composite.FuzzySet method*), 4
transform() (*pyFTS.common.FuzzySet.FuzzySet method*), 6
transform() (*pyFTS.models.seasonal.common.FuzzySet method*), 38
Transformation (*class in pyFTS.common.transformations.transformation*), 17
transformation (*pyFTS.partitioners.partitioner.Partitioner attribute*), 42
trapmf() (*in module pyFTS.common.Membership*), 8
trimf() (*in module pyFTS.common.Membership*), 8
type (*pyFTS.common.FuzzySet.FuzzySet attribute*), 6
type (*pyFTS.partitioners.partitioner.Partitioner attribute*), 42

U

unpack_args() (*pyFTS.models.nonstationary.flrg.NonStationaryFLRG method*), 35
upper_margin (*pyFTS.partitioners.partitioner.Partitioner attribute*), 42
upper_set() (*pyFTS.partitioners.partitioner.Partitioner method*), 42

V

variable (*pyFTS.common.FuzzySet.FuzzySet attribute*), 6
variable (*pyFTS.partitioners.partitioner.Partitioner attribute*), 42

W

white_noise() (*in module pyFTS.data.artificial*), 21
width (*pyFTS.models.nonstationary.common.FuzzySet attribute*), 34
width_params (*pyFTS.models.nonstationary.common.FuzzySet attribute*), 34
window_index() (*in module pyFTS.models.nonstationary.common*), 34

Y

year (*pyFTS.models.seasonal.common.DateTime attribute*), 38

Z

Z (*pyFTS.common.FuzzySet.FuzzySet attribute*), 5