pycast Documentation

Release 0.0.7-prealpha

Christian Schwarz

November 17, 2012

CONTENTS

CHAPTER

ONE

PYCAST.COMMON

pycast.common.profileme.**profileMe**
alias of _ProfileDecorator

pycast.common.helper.**linear_interpolation**(*first*, *last*, *steps*)
Interpolates all missing values using linear interpolation.

Parameters

- **first** (*Numeric*) – Starting value for the interpolation.
- **last** (*Numeric*) – End Value for the interpolation
- **steps** (*Integer*) – Number of missing values that have to be calculated.

Returns Returns a list of floats containing only the missing values.

Return type List

CHAPTER TWO

TIMESERIES

class `pycast.common.timeseries.`**`TimeSeries`**(*isNormalized=False*, *isSorted=False*)
Represents the base class for all time series data.

Warning TimeSeries instances are NOT threadsafe.

`__add__`(*otherTimeSeries*)
Creates a new TimeSeries instance containing the data of self and otherTimeSeries.

Parameters **otherTimeSeries** (*TimeSeries*) – TimeSeries instance that will be merged with self.

Returns Returns a new TimeSeries instance containing the data entries of self and otherTimeSeries. This TimeSeries will be sorted.

Return type TimeSeries

`__eq__`(*otherTimeSeries*)
Returns if the TimeSeries equals another one.

TimeSeries are equal to each other if:

- they contain the same number of entries
- that each data entry in one TimeSeries is also member of the other one.

The sort order within the TimeSeries datapoints does not matter!

Returns `True` if the TimeSeries objects are equal, `False` otherwise.

Return type Boolean

`__getitem__`(*index*)
Returns the item stored at the TimeSeries index-th position.

Parameters **index** (*Integer*) – Position of the element that should be returned. Starts at 0

Returns Returns a list consisting of [timestamp, data].

Return type List

Raise Raises an `IndexError` if the index is out of range.

`__init__`(*isNormalized=False*, *isSorted=False*)
Initializes the TimeSeries.

Parameters

- **isNormalized** (*Boolean*) – Within a normalized TimeSeries, all data points have the same temporal distance to each other. When this is `True`, the memory consumption of the

TimeSeries might be reduced. Also some algorithms will probably run faster on normalized TimeSeries. This should only be set to `True`, if the TimeSeries is realy normalized! TimeSeries normalization can be forced by executing `TimeSeries.normalize()`.

- **isSorted** (*Boolean*) – If all data points added to the time series are added in their ascending temporal order, this should set to `True`.

`__iter__`()
Returns an iterator to the TimeSeries stored data.

Returns Returns an iterator for the TimeSeries.

Return type Iterator

`__len__`()
Returns the number of data entries that are part of the time series.

Returns Returns an Integer representing the number on data entries stored

within the TimeSeries. :rtype: Integer

`__setitem__`(*index*, *value*)
Sets the item at the index-th position of the TimeSeries.

Parameters

- **index** (*Integer*) – Index of the element that should be set.
- **value** (*List*) – A list of the form [timestamp, data]

Raise Raises an `IndexError` if the index is out of range.

`__str__`()
Returns a string representation of the TimeSeries.

Returns Returns a string representing the TimeSeries in the format: TimeSeries([timestamp, data], [timestamp, data], [timestamp, data]).

Return type String

`__weakref__`
list of weak references to the object (if defined)

`add_entry`(*timestamp*, *data*, *format=None*)
Adds a new data entry to the TimeSeries.

Parameters

- **timestamp** – Time stamp of the datas occurence. This has either to be a float representing the UNIX epochs or a string containing a timestamp in the given format.
- **data** – Data points information. This has to be a numeric value for now.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

`apply`(*method*)
Applies the given ForecastingAlgorithm or SmoothingMethod from the `pycast.methods` module to the TimeSeries.

Parameters **method** (*BaseMethod*) – Method that should be used with the TimeSeries. For more information about the methods take a look into their corresponding documentation.

classmethod **`convert_epoch_to_timestamp`**(*timestamp*, *format*)
Converts the given float representing UNIX-epochs into an actual timestamp.

Parameters

- **timestamp** (*Float*) – Timestamp in the defined format.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

Returns Returns an timestamp as defined by format.

Return type String

classmethod **`convert_timestamp_to_epoch`**(*timestamp*, *format*)

Converts the given timestamp into a float representing UNIX-epochs.

Parameters

- **timestamp** (*Float*) – Timestamp in the defined format.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

Returns Returns an float, representing the UNIX-epochs for the given timestamp.

Return type Float

classmethod **`from_json`**(*jsonBaseString*, *format=None*)

Creates a new TimeSeries instance from the given json string.

Parameters

- **jsonBaseString** (*String*) – JSON string, containing the time series data. This should be a string created by `TimeSeries.to_json()`.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

Returns Returns a TimeSeries instance containing the data.

Return type TimeSeries

Warning This is an unsafe version! Only use it with the original version. All assumtions regarding normalization and sort order will be ignored and set to default.

classmethod **`from_twodim_list`**(*datalist*, *format=None*, *isSorted=False*)

Initializes the TimeSeries's data from the two dimensional list.

Parameters

- **datalist** (*List*) – List containing multiple iterables with at least two values. The first item will always be used as timestamp in the predefined format, the second represents the value. All other items in those sublists will be ignored.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.
- **isSorted** (*Boolean*) – Determines if the datalist is sorted by the timestamps. If this is False, the TimeSeries instance sorts itself after all values are read.

Returns Returns a TimeSeries instance containing the data from datalist.

Return type TimeSeries

`initialize_from_sql_cursor`(*sqlcursor*, *format=None*, *isSorted=False*)
Initializes the TimeSeries's data from the given SQL cursor.

Parameters

- **sqlcursor** (*SQLCursor*) – Cursor that was holds the SQL result for any given "SELECT timestamp, value, ... FROM ..." SQL query. Only the first two attributes of the SQL result will be used.
- **format** (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.
- **isSorted** (*Boolean*) – Determines if the SQL result is already sorted. If this is False, the TimeSeries instance sorts itself after all values are read.

Returns Returns the number of entries added to the TimeSeries.

Return type Integer

Todo This function is not bulletprove, yet.

`is_normalized`()
Returns if the TimeSeries is normalized.

Returns Returns `True` if the TimeSeries is normalized, `False` otherwise.

Return type Boolean

`is_sorted`()
Returns if the TimeSeries is sorted.

Returns Returns `True` if the TimeSeries is sorted ascending, `False` otherwise.

Return type Boolean

`normalize`(*normalizationLevel='minute'*, *fusionMethod='average'*, *interpolationMethod='linear'*)
Normalizes the TimeSeries data points.

If this function is called, the TimeSeries gets ordered ascending automatically. The new timestamps will represent the center of each time bucket.

Parameters

- **normalizationLevel** (*String*) – Level of normalization that has to be applied. The available normalization levels are defined in `timeseries.NormalizationLevels`.
- **fusionMethod** (*String*) – Normalization method that has to be used if multiple data entries exist

within the same normalization bucket. The available methods are defined in `timeseries.FusionMethods`. :param String interpolationMethod: Interpolation method that is used if a data entry at a specific time

is missing. The available interpolation methods are defined in `timeseries.InterpolationMethods`.

Raise Raises a `ValueError` if a parameter has an unknown method.

`sort_timeseries`(*ascending=True*)
Sorts the data points within the TimeSeries according to their occurence inline.

Parameters ascending (*Boolean*) – Determines if the TimeSeries will be ordered ascending or decending. If this is set to decending once, the ordered parameter defined in `TimeSeries.__init__()` will be set to False FOREVER.

Returns Returns self for convenience.

Return type TimeSeries

`sorted_timeseries`(*ascending=True*)

Returns a sorted copy of the TimeSeries, preserving the original one.

As an assumtion this new TimeSeries is not ordered anymore by default.

Parameters ascending (*Boolean*) – Determines if the TimeSeries will be ordered ascending or decending.

Returns Returns a new TimeSeries instance sorted in the requested order.

Return type TimeSeries

`to_gnuplot_datafile`(*datafilepath*, *format=None*)

Dumps the TimeSeries into a gnuplot compatible data file.

Parameters

- **datafilepath** (*String*) – Path used to create the file. If that file already exists, it will be overwritten!
- **format** (*String*) – Format of the timestamp. This is used to convert the timestamp from UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

Returns Returns `True` if the data could be written, `False` otherwise.

Return type Boolean

`to_json`(*format=None*)

Returns a JSON representation of the TimeSeries data.

Parameters format (*String*) – Format of the given timestamp. This is used to convert the timestamp into UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

Returns Returns a basestring, containing the JSON representation of the current

data stored within the TimeSeries. :rtype: String

`to_twodim_list`(*format=None*)

Serializes the TimeSeries data into a two dimensional list of [timestamp, value] pairs.

Parameters format (*String*) – Format of the timestamp. This is used to convert the timestamp from UNIX epochs, if necessary. For valid examples take a look into the `time.strptime()` documentation.

Returns Returns a two dimensional list containing [timestamp, value] pairs.

Return type List

CHAPTER

THREE

SMOOTHING METHODS

class `pycast.methods.`**`SimpleMovingAverage`**(*windowsize=5*)

Implements the simple moving average.

The SMA algorithm will calculate the average value at time t based on the datapoints between [t - floor(windowsize / 2), t + floor(windowsize / 2)].

Explanation: http://en.wikipedia.org/wiki/Moving_average

`__init__`(*windowsize=5*)

Initializes the SimpleMovingAverage.

Parameters **windowsize** (*Integer*) – Size of the SimpleMovingAverages window. This number has to be uneven and positive.

`execute`(*timeSeries*)

Creates a new TimeSeries containing the SMA values for the predefined windowsize.

Returns TimeSeries object containing the smooth moving average.

Return type TimeSeries

Todo This implementation aims to support independent for loop execution.

CHAPTER

FOUR

FORECASTING METHODS

class `pycast.methods.`**`ExponentialSmoothing`**(*smoothingFactor=0.1*, *valuesToForecast=1*)

Implements an exponential smoothing algorithm.

Explanation: http://www.youtube.com/watch?v=J4iODLa9hYw

`__init__`(*smoothingFactor=0.1*, *valuesToForecast=1*)

Initializes the ExponentialSmoothing.

Parameters

- **smoothingFactor** (*Float*) – Defines the alpha for the ExponentialSmoothing. Valid values are (0.0, 1.0).
- **valuesToForecast** (*Integer*) – Number of values that should be forecasted.

Raise Raises a `ValueError` when smoothingFactor has an invalid value.

`execute`(*timeSeries*)

Creates a new TimeSeries containing the smoothed values and one forecasted one.

Returns TimeSeries object containing the exponentially smoothed TimeSeries, including the forecasted value.

Return type TimeSeries

Todo Currently the first normalized value is simply chosen as the starting point.

class `pycast.methods.`**`HoltMethod`**(*smoothingFactor=0.1*, *trendSmoothingFactor=0.5*, *valuesToForecast=1*)

Implements the Holt algorithm.

Explanation: http://en.wikipedia.org/wiki/Exponential_smoothing#Double_exponential_smoothing

`__init__`(*smoothingFactor=0.1*, *trendSmoothingFactor=0.5*, *valuesToForecast=1*)

Initializes the HoltMethod.

Parameters

- **smoothingFactor** (*Float*) – Defines the alpha for the ExponentialSmoothing. Valid values are (0.0, 1.0).
- **trendSmoothingFactor** (*Float*) – Defines the beta for the HoltMethod. Valid values are (0.0, 1.0).
- **valuesToForecast** (*Integer*) – Defines the number of forecasted values that will be part of the result.

Raise Raises a `ValueError` when smoothingFactor or trendSmoothingFactor has an invalid value.

execute(*timeSeries*)
Creates a new TimeSeries containing the smoothed values.

Returns TimeSeries object containing the exponentially smoothed TimeSeries, including the forecasted values.

Return type TimeSeries

Todo Currently the first normalized value is simply chosen as the starting point.

CHAPTER

FIVE

ERROR MEASURES

class `pycast.errors.`**`BaseErrorMeasure`**(*minimalErrorCalculationPercentage=60*)
Baseclass for all error measures.

`__init__`(*minimalErrorCalculationPercentage=60*)
Initializes the error measure.

Parameters **minimalErrorCalculationPercentage** (*Integer*) – The number of entries in an original TimeSeries that have to have corresponding partners in the calculated TimeSeries. Corresponding partners have the same time stamp. Valid values are [0.0, 100.0].

Raise Raises a `ValueError` if minimalErrorCalculationPercentage is not in [0.0, 100.0].

`__weakref__`
list of weak references to the object (if defined)

`calculate`(*startingPercentage*, *endPercentage*)
This is the error calculation function that gets called by get_error().

Both parameters will be correct at this time.

Parameters

- **startingPercentage** (*Float*) – Defines the start of the interval. This has to be a value in [0.0, 100.0]. It represents the value, where the error calculation should be started. 25.0 for example means that the first 25%% of all calculated errors will be ignored.
- **endPercentage** (*Float*) – Defines the end of the interval. This has to be a value in [0.0, 100.0]. It represents the vlaue, after which all error values will be ignored. 90.0 for example means that the last 10%% of all local errors will be ignored.

Returns Returns a float representing the error.

Return type Float

Raise Raises a `NotImplementedError` if the child class does not overwrite this method.

`get_error`(*startingPercentage=0.0*, *endPercentage=100.0*)
Calculates the error for the given interval (startingPercentage, endPercentage) between the TimeSeries given during initialize().

Parameters

- **startingPercentage** (*Float*) – Defines the start of the interval. This has to be a value in [0.0, 100.0]. It represents the value, where the error calculation should be started. 25.0 for example means that the first 25%% of all calculated errors will be ignored.

- **endPercentage** (*Float*) – Defines the end of the interval. This has to be a value in [0.0, 100.0]. It represents the vlaue, after which all error values will be ignored. 90.0 for example means that the last 10%% of all local errors will be ignored.

Returns Returns a float representing the error.

Return type Float

Raise Raises a `ValueError` in one of the following cases: startingPercentage not in [0.0, 100.0] endPercentage not in [0.0, 100.0] endPercentage < startingPercentage

Raise Raises a `StandardError` if `BaseErrorMeasure.initialize()` was not successfull before.

initialize(*originalTimeSeries*, *calculatedTimeSeries*)
Initializes the ErrorMeasure.

During initialization, all local_errors are calculated.

Parameters

- **originalTimeSeries** (*TimeSeries*) – TimeSeries containing the original data.
- **calculatedTimeSeries** (*TimeSeries*) – TimeSeries containing calculated data. Calculated data is smoothed or forecasted data.

Returns Return `True` if the error could be calculated, `False` otherwise based on the minimalErrorCalculationPercentage.

Return type Boolean

local_error(*originalValue*, *calculatedValue*)
Calculates the error between the two given values.

Parameters

- **originalValue** (*Numeric*) – Value of the original data.
- **calculatedValue** (*Numeric*) – Value of the calculated TimeSeries that corresponds to originalValue.

Returns Returns the error measure of the two given values.

Return type Numeric

Raise Raises a `NotImplementedError` if the child class does not overwrite this method.

class pycast.errors.**MeanSquaredError**(*minimalErrorCalculationPercentage=60*)
Implements the mean squared error measure.

Explanation: http://en.wikipedia.org/wiki/Mean_squared_error

calculate(*startingPercentage*, *endPercentage*)
This is the error calculation function that gets called by get_error().

Both parameters will be correct at this time.

Parameters

- **startingPercentage** (*Float*) – Defines the start of the interval. This has to be a value in [0.0, 100.0]. It represents the value, where the error calculation should be started. 25.0 for example means that the first 25%% of all calculated errors will be ignored.
- **endPercentage** (*Float*) – Defines the end of the interval. This has to be a value in [0.0, 100.0]. It represents the vlaue, after which all error values will be ignored. 90.0 for example means that the last 10%% of all local errors will be ignored.

Returns Returns a float representing the error.

Return type Float

Raise Raises a `NotImplementedError` if the child class does not overwrite this method.

local_error (*originalValue*, *calculatedValue*)

Calculates the error between the two given values.

Parameters

- **originalValue** (*Numeric*) – Value of the original data.
- **calculatedValue** (*Numeric*) – Value of the calculated TimeSeries that corresponds to originalValue.

Returns Returns the error measure of the two given values.

Return type Numeric

Raise Raises a `NotImplementedError` if the child class does not overwrite this method.

class `pycast.errors.`**SymmetricMeanAbsolutePercentageError** (*minimalErrorCalculationPercentage=60*)

Implements the symmetric mean absolute percentage error with a boarder of 200%%.

Explanation: http://monashforecasting.com/index.php?title=SMAPE (Formula (3))

If the calculated value and the original value are equal, the error is 0.

calculate (*startingPercentage*, *endPercentage*)

This is the error calculation function that gets called by get_error().

Both parameters will be correct at this time.

Parameters

- **startingPercentage** (*Float*) – Defines the start of the interval. This has to be a value in [0.0, 100.0]. It represents the value, where the error calculation should be started. 25.0 for example means that the first 25%% of all calculated errors will be ignored.
- **endPercentage** (*Float*) – Defines the end of the interval. This has to be a value in [0.0, 100.0]. It represents the vlaue, after which all error values will be ignored. 90.0 for example means that the last 10%% of all local errors will be ignored.

Returns Returns a float representing the error.

Return type Float

Raise Raises a `NotImplementedError` if the child class does not overwrite this method.

local_error (*originalValue*, *calculatedValue*)

Calculates the error between the two given values.

Parameters

- **originalValue** (*Numeric*) – Value of the original data.
- **calculatedValue** (*Numeric*) – Value of the calculated TimeSeries that corresponds to originalValue.

Returns Returns the error measure of the two given values.

Return type Numeric

Raise Raises a `NotImplementedError` if the child class does not overwrite this method.

CHAPTER

SIX

OPTIMIZATION METHODS

class `pycast.optimization.`**`BaseOptimizationMethod`**(*errorMeasureClass*, *precision=-1*)

Baseclass for all optimization methods.

`__init__`(*errorMeasureClass*, *precision=-1*)

Initializes the optimization method.

Parameters

- **errorMeasureClass** (*BaseErrorMeasure*) – Error measure class from pycast.errors
- **precision** (*Integer*) – Defines the accuracy for parameter tuning in 10^precision. This parameter has to be an integer in [-10, 0].

Raise Raises a `TypeError` if errorMeasureClass is not a valid class. Valid classes are derived from `pycast.errors.BaseErrorMeasure`.

Raise Raises a :py:exc:'ValueError if precision is not in [-10, 0].

`__weakref__`

list of weak references to the object (if defined)

`optimize`(*timeSeries*, *forecastingMethods=[ ]*)

Runs the optimization of the given TimeSeries.

Parameters

- **timeSeries** (*TimeSeries*) – TimeSeries instance that requires an optimized forecast. It has to have
- **forecastingMethods** (*List*) – List of forecastingMethods that will be used for optimization. This list cannot be empty!

Returns Returns the optimzed forecasting method with the smallest error.

Return type BaseForecastingMethod, Dictionary

Raise Raises a `ValueError` ValueError if no forecastingMethods are defined.

class `pycast.optimization.`**`GridSearch`**(*errorMeasureClass*, *precision=-1*)

Implements the grid search method for parameter optimization.

GridSearch is the brute force method.

`optimize`(*timeSeries*, *forecastingMethods=[ ]*)

Runs the optimization of the given TimeSeries.

Parameters

- **timeSeries** (*TimeSeries*) – TimeSeries instance that requires an optimized forecast. It has to have
- **forecastingMethods** (*List*) – List of forecastingMethods that will be used for optimization. This list cannot be empty!

Returns Returns the optimzed forecasting method with the smallest error.

Return type BaseForecastingMethod, Dictionary

Raise Raises a `ValueError` ValueError if no forecastingMethods are defined.

`optimize_forecasting_method`(*timeSeries*, *forecastingMethod*)

Optimizes the parameters for the given timeSeries and forecastingMethod.

Parameters

- **timeSeries** (*TimeSeries*) – TimeSeries instance, containing hte original data.
- **forecastingMethod** (*BaseForecastingMethod*) – ForecastingMethod that is used to optimize the parameters.

Todo Errorclass for calculation

Todo percentage for start_error_measure, end_error_measure

Todo Definition of the result that will be returned.

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

p

INDEX