Redistributor Documentation

Contents

Module `redistributor`

Redistributor is a tool for automatic transformation of empirical data distributions. It is implemented in **Python3** as a **Scikit-learn transformer**.

It allows the user to transform their data from arbitrary distribution into other arbitrary distribution. The source and target distributions can be specified exactly, if known beforehand, or can be inferred from the data. Transformation is **piece-wise smooth, monotonic, and invertible**, and can be **saved for later use** on different data assuming the same source distribution.

The empirical distribution can be inferred from a 1D array of data. To redistribute multiple slices of your data use Redistributor_multi class which has a **low memory footprint** and utilizes **parallel computing** to apply multiple Redistributor objects.

Installation

:warning: | Not yet published on PyPi. Coming soon. :—: | :—

The code is hosted in this GitLab repository[1]. To install the released version from Pypi use:

```
pip install redistributor
```

[1] https://gitlab.com/paloha/redistributor

Or install the bleeding edge directly from git:

```
pip install git+https://gitlab.com/paloha/redistributor
```

For development, install the package in editable mode with extra dependencies for documentation and testing:

```
# Clone the repository
git clone git@gitlab.com:paloha/redistributor.git
cd redistributor

 # Use virtual environment [optional]
python3 -m virtualenv .venv
source .venv/bin/activate

# Install with pip in editable mode
pip install -e .[dev]
```

Compatibility

Dependencies

Required packages for Redistributor are specified in the install_requires list in the setup.py file.

Extra dependencies for running the tests, compiling the documentation, or running the examples are specified in the extras_require dictionary in the same file.

The full version-locked list of dependencies and subdependencies is frozen in requirements.txt. Installing with `pip install -r requirements.txt` in a virtual environment should always lead to a fully functional project.

Mathematical description

Assume we are given data $x \sim S$ distributed according to some source distribution S on $\mathbb{R}$ and our goal is to find a transformation R such that $R(x) \sim T$ for some target distribution T on $\mathbb{R}$.

One can mathematically show that a suitable $R\colon \mathbb{R} \to \mathbb{R}$ is given by

$$R := F_T^{-1} \circ F_S,$$

where F_S and F_T are the cumulative distribution functions of S and T, respectively.

If S and T is unknown, one can use approximations $\tilde{F}_S$ and $\tilde{F}_T$ of the corresponding cumulative distribution functions given by interpolating (partially) sorted data

$$(x_i)_{i=1}^N \text{ with } x_i \sim S$$

$$(y_i)_{i=1}^M \text{ with } y_i \sim T.$$

Defining

$$\tilde{R} := \tilde{F}_T^{-1} \circ \tilde{F}_S,$$

one can, under suitable conditions, show that

$$\tilde{R} \xrightarrow[N,M\to\infty]{} R.$$

How to cite

License

Acknowledgement

This work was supported by the *International Mobility of Researchers* (program call no.: CZ.02.2.69/0.0/0.0/16027/0008371[2]).

EUROPEAN UNION
European Structural and Investment Funds
Operational Programme Research,
Development and Education

To do list

:white_check_mark: use code checker (flake8 + autopep8) :black_square_button: specify compatibility with Python versions in this readme :black_square_button: specify compatibility with operating systems in this readme :black_square_button: add citation bibtex in this readme :black_square_button: update urls in setup.py :black_square_button: update acknowledgement in this readme :black_square_button: use pytest and tox[3] for testing :black_square_button: publish on PyPi guide here[4] :black_square_button: verify integrity of html and pdf outputs + gitlab and gitlab pages display

Functions

Function `load_redistributor`

```
def load_redistributor(
    path
)
```

Loads the Redistributor or Redistributor_multi object from a file.

Function `save_redistributor`

```
def save_redistributor(
    d,
    path
)
```

Saves the Redistributor or Redistributor_multi object to a file.

Classes

Class `Redistributor`

```
class Redistributor(
    target=<scipy.stats._distn_infrastructure.rv_frozen object>,
    known_distribution=None,
    bbox=None,
    closed_interval=True,
    prevent_same=True,
    tinity=1000000,
    validate_input=True,
    bins=0
)
```

An algorithm for automatic transformation of data from arbitrary distribution into arbitrary distribution. Source distribution can be known beforehandand or learned from the data. Transformation is piecewise smooth, monotonic and invertible.

[2] https://opvvv.msmt.cz/vyzva/vyzva-c-02-16-027-mezinarodni-mobilita-vyzkumnych-pracovniku.htm
[3] https://tox.readthedocs.io/en/latest/example/pytest.html
[4] https://packaging.python.org/guides/distributing-packages-using-setuptools

Implemented as a Scikit-learn transformer. Can be fitted on 1D vector (for more dimensions use Redistributor_multi wrapper) and saved to be used later for transforming other data assuming the same source distribution.

Uses source's and target's cdf() and ppf() to infer the transform and inverse transform functions.

transform_function = target_ppf(source_cdf(x)) inverse_transform = source_ppf(target_cdf(x))

Time Complexity

This algorithm can scale to a large number of samples > 1e08. Algorithm can be sped up by: 1. Turning off input data validation by setting validate_input=False 2. Choosing target distribution with fast cdf and ppf functions. 3. Specifying the source distribution if it is known.

Parameters

`target` **: obj, optional, default** `scipy.stats.norm(loc=0, scale=1)` Class specifying the target distribution. Must implement cdf(), ppf() and ideally pdf() methods. Continuous distributions from scipy.stats can be used.

`known_distribution` **: obj, optional, default None** Object specifying the source distribution when known. Must implement cdf(), ppf() and ideally pdf() methods. Continuous distributions from scipy.stats can be used.

`bbox` **: tuple, optional, default None** Tuple (a, b) which represent the lower and upper boundary of the source distribution's domain.

`closed_interval` **: bool, optional, default True** If True, the values set in bounding box are included into the interpolation range.

`prevent_same` **: bool, optional, default True** Flag, specifyig whether to prevent same values in the interpolated vector.

`tinity` **: int, default 1e06** Specifies divisor of noise added to vectors with same elements. Bigger the tinity, smaller the noise.

`validate_input` **: bool, optional, default True** Flag specifying if the input data should be validated. Turn off to save computation time. Output is undefined if the data are not valid.

`bins` **: int, optional, default 0** Number of bins which are used to infer the latice density when the distribution is learned. Irelevant with known_distribution.

If bins = x.size, then latice density = 100%. The interpolation step = x.size // bins = 1. This is most precise. If x.size is big enough, we can lower the latice density and the fit will remain very precise.

If bins = 0, it is set automatically. The latice density = 100% is then used up to x.size = 5000. If x.size > 5000, bins = 5000.

Attributes

self.target_cdf: callable, default scipy.stats.norm(loc=0, scale=1).cdf Cummulative Density Function of target distribution.

self.target_ppf: callable, default scipy.stats.norm(loc=0, scale=1).ppf Percent Point Function (inverse of cdf) of target distribution.

self.source_cdf: callable, default None Either known_distribution.cdf function of source distribution or learned on fit() by interpolation on latice.

self.source_ppf: callable, default None Either known_distribution.ppf function of source distribution or learned on fit() by interpolation on latice.

self.a: float, default None Beginning of the interval of the source distribution domain.

self.b: float, default None End of the interval of the source distribution domain.

self.n: int, default None Size of the training data extened by bounding box borders if necessary.

self.fitted: bool, default False Flag that specifies whether the fit() was already called.

Limitations

- Output is undefined when most of the data points have the exact same value, therefore data should not be sparse - multiple zeros etc. Handles small amounts of the exact same values by adding tiny noise.
- Plotting of learned pdf() is just a smoothed approximation obtained by 1st derivative of the learned piece-wise smooth cdf() and it serves only as a visual aid.

Examples

TODO

Ancestors (in MRO)

- sklearn.base.TransformerMixin

Methods

Method `compute_empirical_cdf_error`

```
def compute_empirical_cdf_error(
    self,
    x,
    error_func='mse'
)
```

Computes error caused by approximation of transform function. Error computation needs to sort all data. It can take a long time for big arrays.

Parameters

`x` : numpy array 1D vector of transformed data.
`error_func` : callable or one of `{'mae', 'mse'}` Callable that computes error on two vectors. 'mae' = Error in L1 norm (Mean Absolute Error) 'mse' = Error in L2 norm (Mean Squared Error)

Returns

Float value of specified error or return value of callable.

Method `fit`

```
def fit(
    self,
    x=None
)
```

Calls all necessary methods to infer cdf and ppf of the source distribution. Source distribution can be either specified directly by its function or learned from the training data on a closed or opened interval. Learning approximates the cdf and ppf by interpolating on a latice which density is infered from bins.

Parameters

`x` : 1D numpy array, optional, default None Training data from which the source distribution is learned. Must be specified if self.known_distribution is None.

Method `inverse_transform`

```
def inverse_transform(
    self,
    x
)
```

Applies learned inverse transform function to the data x.

Parameters

`x` **: numpy array** Data to inverse transform. Must be within the interp. interval. of learned transform function.

Returns

Numpy array of inverse transformed data.

Method `plot_hist`

```
def plot_hist(
    self,
    data,
    nbins=None,
    title='Histogram',
    figsize=(15, 2),
    xlim=None
)
```

Displays matplotlib histogram of the specified data and nbins.

Parameters

`data` **: numpy array** Specifying the position of data points on x axis.
`nbins` **: int, optional, default self.bins** Specifying the number of bins of the histogram. If None, the number will be automatically set by matplotlib.
`title` **: str, optional, default** `'Histogram'` Title of the plot.
`figsize` **: tuple (width, height), optional, default (15,2)** Desired size of the figure.
`xlim` **: float, optional, default None** Limit of x axis.

Method `plot_source_pdf`

```
def plot_source_pdf(
    self,
    smoothed=True,
    figsize=(15, 2)
)
```

Displays matplotlib plot of the learned source probability density function.

Parameters

`smoothed` **: boolean, optional, default True** Applies savgol_filter to smooth the curve that is disturbed by derivatives at bin transitions.
`figsize` **: tuple (width, height), optional, default (15,2)** Desired size of the figure.

Method `plot_transform_function`

```
def plot_transform_function(
    self,
    figsize=(15, 2)
)
```

Displays matplotlib plot of the fitted transform function.

Parameters

`figsize` **: tuple (width, height), optional, default (15,2)** Desired size of the figure.

Method `transform`

```
def transform(
    self,
    x
)
```

Applies learned transformation function to the data x.

Parameters

x : numpy array Data to transform. Must be within self.a, self.b interval.

Returns

Numpy array of transformed data.

Class `Redistributor_multi`

```
class Redistributor_multi(
    redistributors,
    nsub,
    cpus=0
)
```

Multi-dimensional wrapper for Redistributor. Allows to use multiple Redistributors on equal-sized slices (submatrices) of N-dimensional input array. Utilizes parallel processing with low memory footprint.

Parameters:

redistributors: numpy array of Redistributor objects Objects that will be used to operate on the data within each slice defined by nsub. Shape must be the same as nsub.

nsub: tuple of int Tuple specifying how to split multidimensional input into submatrices. Corresponding redistributor object will be applied on each submatrix separately. Each int specifies to how many equal submatrices corresponding axis should be split. There must be exactly one int for each axis of multidimensional input data.

cpus: int >= 0 Number of cpu cores to use in multiprocessing. If 0, all cores will be used.

Static methods

`Method init_global_array`

```
def init_global_array(
    array
)
```

Initializer of shared array for multiprocessing pool.

`Method populate`

```
def populate(
    args
)
```

Static method called by child processes that applies desired function of redistributor on the data from shared matrix on desired location and populates the result back to the shared matrix.

Methods

Method `fit`

```
def fit(
    self,
    x=None,
    size_limit=0
)
```

Fits all Redistributor objects in self.redistributors.

Parameters

`x` **: numpy array, optional, default None** Data to fit on. If None, Redistributors must have source set explicitly.

`size_limit` **: float, optional, default 0** Should be 3x smaller than available memory after loading the data in GB. Based on this value, the appropriate size of chunk is infered. If size_limit == 0, the best value is computed automatically.

Method `inverse_transform`

```
def inverse_transform(
    self,
    x,
    inplace=True,
    size_limit=0
)
```

Inverse transforms the data in x using fitted Redistributors.

Parameters

`x` **: numpy array** Data to transform.

`inplace` **: bool, optional, default True** Flag specifying whether to change the data in place without keeping the original values stored in x or operate on copy.

`size_limit` **: float, optional, default 0** See docstring of self.fit

Returns

Inverse transformed data with same shape as input.

Method `machinery`

```
def machinery(
    self,
    purpose,
    size_limit
)
```

Handles locating subarrays and their parallel processing in chunks.

Parameters

`purpose` **: one of** `{'fit', 'transform', 'inverse'}` Specifies what should be done with the data.

`size_limit` **: float** Ideal size of one chunk. 0 = automatic.

Method `transform`

```
def transform(
    self,
    x,
    inplace=True,
    size_limit=0
)
```

Transforms data in x using fitted Redistributors.

Parameters

`x` **: numpy array** Data to transform.

`inplace` **: bool, optional, default True** Flag specifying whether to change the data in place without keeping the original values stored in x or operate on copy.

`size_limit` **: float, optional, default 0** See docstring of self.fit

Returns

Transformed data with same shape as input.
