Redistributor Documentation

Contents

Module `redistributor`

:warning: | Still under development :—: | :—

This repository introduces two main classes, namely **Redistributor** and **LearnedDistribution**.

Redistributor is a tool for automatic transformation of empirical data distributions. It is implemented as a **Scikit-learn transformer**. It allows users to transform their data from arbitrary distribution into another arbitrary distribution. The source and target distributions, if known beforehand, can be specified exactly (e.g. as a Continuous Scipy distribution[1] or any other class which has cdf and pdf methods implemented), or can be inferred from the data using LearnedDistribution class. Transformation is **piece-wise-linear, monotonic, invertible**, and can be **saved for later use** on different data assuming the same source distribution.

LearnedDistribution is a subclass of Scipy.stats.rv_continous[2] class. It is a continuous random variable obtained by estimating the empirical distribution of a user provided array of numeric data x. It can be used to sample new random points from the learned distribution.

[1]https://docs.scipy.org/doc/scipy/reference/tutorial/stats/continuous.html#continuous-distributions-in-scipy-stats
[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.rv_continuous.html#scipy-stats-rv-continuous

Installation

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

The code is hosted in this GitLab repository[3]. To install the released version from Pypi use:

```
pip install 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

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

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

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[4]).

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

MINISTRY OF EDUCATION,
YOUTH AND SPORTS

Functions

Function `load_redistributor`

```
def load_redistributor(
    path
)
```

Loads the Redistributor object from a file.

Function `make_unique`

```
def make_unique(
    array,
    random_state,
    mode='spread',
    duplicates=None
)
```

Takes a sorted array and adjusts the duplicate values such that all elements of the array are unique. The adjustment is done by linearly separating the duplicates. Read more in docsting of _get_intervals.

In case `mode='keep'` this function does nothing and returns the array.

Supports two deterministic modes 'spread' and 'cluster'. These two define onto how large interval the valueas are spread. If 'cluster' is not possible 'spread' is used implicitly.

In case there are too many duplicates (>5e3), first uses addition of random noise to non-min and non-max values and then continues with the deterministic method.

Keeps the min, max, and unique values unchanged. If the first iteration did not make all elements unique, repeats until failure and warns the user (should be rare).

Parameters

`array` : 1D numpy array Sorted array with potential of having non-unique elements.
`random_state` : RandomState
`mode` : str, one of `{'keep', 'spread', 'cluster', 'noise'}`

'keep' produces discontinuous cdf (cdf with vertical jumps) because it just simply keeps the non unique values 'spread' is deterministic but slow to compute, it separates the non unique values equidistantly and tries to use all the available space between consecutive values. 'cluster' is deterministic and also slow to compute, it separates

[4] https://opvvv.msmt.cz/vyzva/vyzva-c-02-16-027-mezinarodni-mobilita-vyzkumnych-pracovniku.htm

the non unique values equidistantly but it does only use a small space around the value. ‘noise’ is fast, very similar to ‘cluster’, but nondeterministic because it involves randomness and it handles min and max values separately to avoid jumping out of the a,b interval.

duplicates: int, number of duplicates in previous iteration. Do not use, used only for recursion.

Returns

Sorted array of unique elements on the orignal interval.

Function `plot_cdf_ppf_pdf`

```
def plot_cdf_ppf_pdf(
    dist,
    a=None,
    b=None,
    bins=None,
    v=None,
    w=None,
    rows=1,
    cols=3,
    figsize=(16, 5)
)
```

Just a convinience function for visualizing the dist cdf, ppf and pdf functions.

Parameters

`a` **: float** Start of the cdf support
`b` **: float** End of the cdf support
`v` **: float** Start of the ppf support
`w` **: float** End of the ppf support
`rows` **: int,** Number of rows in the figure
`cols` **: int,** Number of cols in the figure
`figsize` **: None or tuple** If None, no new figure is created.

Function `save_redistributor`

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

Saves the Redistributor object to a file.

Classes

Class `KernelDensity`

```
class KernelDensity(
    x,
    ravel_x=True,
    grid_density=10000,
    cdf_method='fast',
    name='KDE',
    **kwargs
)
```

Wrapper around KernelDensity for ease of use as a source or target distribution of Redistributor. It extends the KDE by providing cdf and ppf functions.

Only supports 1D input because Redistributor also works only in 1D. Only supports gaussian kernel. CDF supports two methods, precise and fast. CDF precise is computed using a formula. CDF fast is a linear interpolation of the

CDF precise on a grid of specified density. There is no explicit formula for PPF of gaussian mixutre, so here it is approximated using linear interpolation of the CDF precise on a grid of specified density.

Parameters

`x` : numeric or 1D numpy array 1D vector of which the distribution will be estimated.
`ravel_x` : bool, default True KDE requires 1D arrays. So the x is by default flattened to 1D using np.ravel().
`grid_density` : int User specified number of grid points on which the CDF is computed precisely in order to build the interpolants for fast CDF and PPF. The same grid is used for CDF fast and PPF. The user specified value of grid_density is not it’s final value. It is updated during initialization of this object on call of self._get_ppf().
`cdf_method` : str, one of {`'precise', 'fast'`} Specifies the default method to be used when self.cdf() is called.
`name` : str, default `'LearnedDistribution'` The name of the instance.

kwargs : all other keyword arguments accepted by sklearn.neighbors.KernelDensity.

Methods

pdf : Probability Density Function of a Gaussian Mixture cdf : Cumulative Density Function of a Gaussian Mixture (or its approximation) ppf : Approximation of a Percent Point Function of a Gaussian Mixture rvs : Random sample generator

Methods

Method `cdf`

```
def cdf(
    self,
    k,
    method=None
)
```

Cummulative density function of the estimated distribution.

Method `pdf`

```
def pdf(
    self,
    x
)
```

Probability density function of the estimated distribution.

Method `ppf`

```
def ppf(
    self,
    q
)
```

This method approximates the ppf based on linear interpolation of the cdf on self.grid_density many points. There is no formula for precise computation of gaussian mixture ppf. Therefore, if we wanted a precise function, we would need to bisect the cdf. Bisecting is very slow in comparison to just computing the cdf on a grid and using the interpolant to approximate the ppf.

Method `rvs`

```
def rvs(
    self,
    size=1,
    random_state=None
)
```

Random sample from the estimated distribution.

Class `LearnedDistribution`

```
class LearnedDistribution(
    x,
    a=None,
    b=None,
    bins=None,
    keep_x_unchanged=True,
    subsample_x=None,
    ravel_x=True,
    assume_sorted=False,
    fill_value='auto',
    bounds_error='warn',
    dupl_method='spread',
    seed=None,
    name='LearnedDistribution',
    **kwargs
)
```

A continuous random variable obtained by estimating the empirical distribution of a user provided 1D array of numeric data x. It can be used to sample new random points from the learned distribution.

It approximates the Cumulative Distribution Function (cdf) and Percent Point Function (ppf) of the underlying distribution of x using linear interpolation on a lattice.

An approximation of the Probability Density Function (pdf) is computed as an interpolation of the numerical derivative of the cdf function. Please note it can oscilate a lot if bins is high.

The distribution is defined on a closed finite interval [a, b] or [xmin, xmax] or combination thereof, depending on which bound/s were specified by the user.

WARNING: It can not be used to learn discrete distributions.

Parameters

`x` : 1D numpy array 1D vector of which the distribution will be estimated.

`a` : numeric or None Left boundary of the distribution support if known. If specified, must be smaller than x.min().

`b` : numeric or None Right boundary of the distribution support if known. If specified, must be bigger than x.max().

`bins` : int or None User specified value of bins. Min is 3, max is x.size. If None or 0, bins are set automatically. Upper bound is set to 1000 to prevent unnecessary computation. Used to specify the density of the lattice. More bins means higher precision but also more computation.

`keep_x_unchanged` : bool, default True If True, the x array will be copied before partial sorting. This will result in increased memory usage. But it will not reorder the user provided array.

If False, there will not be any additional memory consumption. But the user provided array x might change its order. This might be very useful if x is a large array and there is not enough available memory.

`subsample_x` : int, default None Sacrifice precision for speed by first subsampling array x with a defined integer step. Not doing random.choice() but rather simple slice(None, None, subsample_x) because it is faster and we assume the array is randomly ordered. Can lead to significant speedups.

`ravel_x` : bool, default True LearnedDistribution requires 1D arrays. So the x is by default flattened to 1D using np.ravel().

`assume_sorted` : bool, default False If the user knows that x is sorted, setting this to True will save a most of time by ommiting partial sorting the array. Especially useful if the array x is big. E.g. 1GB of data takes approx. 10s to partial sort on 5000 positions. If False and x is almost sorted, it will still be faster than if x is randomly ordered.

`fill_value` **:** `None, array-like, float, 2-tuple` **or** `'auto'`**, default=**`'auto'` Specifies where to map the values out of the cdf support. See the docstring of scipy.interpolate.interp1d to learn more about the valid options. Additionally, this class enables the user to use the default auto option, which sets reasonable fill_value automatically.

`bounds_error` **: bool or** `'warn'`**, default** `'warn'` See the docstring of class interp1d_with_warning.

`dupl_method` **: str, one of** `{'keep', 'spread', 'cluster', 'noise'}` default 'spread' Method of solving duplicate lattice values. Read more in docstring of make_unique().

`name` **: str, default** `'LearnedDistribution'` The name of the instance.

`seed` **:** `{None, int`**,numpy.random.Generator,** numpy.random.RandomState}, default None See the docstring of scipy.stats.rv_continuous. Used in _prevent_same() and rvs().

kwargs : all other keyword arguments accepted by rv_continous.

Methods - TODO finish this documentation

cdf ppf pdf rvs entropy … fill in the rest which is implemented … handle the rest which does not make sense

Ancestors (in MRO)

- scipy.stats._distn_infrastructure.rv_continuous
- scipy.stats._distn_infrastructure.rv_generic

Methods

Method `cdf`

```
def cdf(
    self,
    k
)
```

Cumulative distribution function of the given RV.

Parameters

`x` **: array_like** quantiles

arg1, arg2, arg3,... : array_like The shape parameter(s) for the distribution (see docstring of the instance object for more information) `loc` : array_like, optional : location parameter (default=0)

`scale` **: array_like, optional** scale parameter (default=1)

Returns

`cdf` **: ndarray** Cumulative distribution function evaluated at x

Method `ppf`

```
def ppf(
    self,
    q
)
```

Percent point function (inverse of cdf) at q of the given RV.

Parameters

`q` **: array_like** lower tail probability

arg1, arg2, arg3,... : array_like The shape parameter(s) for the distribution (see docstring of the instance object for more information) `loc` : array_like, optional : location parameter (default=0)

`scale` **: array_like, optional** scale parameter (default=1)

Returns

`x` **: array_like** quantile corresponding to the lower tail probability q.

Method `rvs`

```
def rvs(
    self,
    size,
    random_state=None
)
```

Random sample from the learned distribution.

Class `Redistributor`

```
class Redistributor(
    source,
    target
)
```

An algorithm for automatic transformation of data from arbitrary distribution into arbitrary distribution. Source and target distributions can be known beforehandand or learned from the data using LearnedDistribution class. Transformation is piecewise linear, monotonic and invertible.

Implemented as a Scikit-learn transformer. Can be fitted on 1D vector 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))`

Ancestors (in MRO)

- sklearn.base.TransformerMixin

Methods

Method `fit`

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

Redistributor does not need to be fitted.

Method `inverse_transform`

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

Inverse transform the data from target to source distribution.

Method `kstest`

```
def kstest(
    self,
    n=20
)
```

Performs the (one-sample or two-sample) Kolmogorov-Smirnov test.

Method `plot_transform_function`

```
def plot_transform_function(
    self,
    bins=1000,
    newfig=True,
    figsize=(16, 5)
)
```

Plotting the learned transformation from source to target.

Method `transform`

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

Transform the data from source to target distribution.

Class `interp1d_with_warning`

```
class interp1d_with_warning(
    *args,
    **kwargs
)
```

By default behaves exactly as scipy.interpolate.interp1d but allows the user to specify `bounds_error = 'warn'` which overrides the behaviour of _check_bunds to warn instead of raising an error.

Parameters

Accepts all the args and kwargs as scipy.interpolate.interp1d. Additionally,

Initialize a 1-D linear interpolation class.

Ancestors (in MRO)

- scipy.interpolate.interpolate.interp1d
- scipy.interpolate.polyint._Interpolator1D

Generated by *pdoc* 0.9.2 (https://pdoc3.github.io).