mightypy Documentation

Release 0.0.10.post1.dev1+gb1239fb

Nishant Baheti

Apr 14, 2024

CONTENTS

This is the documentation of **mightypy**.

CHAPTER

ONE

DESCRIPTION

This package initially started with the idea of expanding mightiness of python with expermenting on data structures and machine learning algorithms.

visit https://mightypy.readthedocs.io/ for detailed documentation

Install from pypi:

```
$ pip install -U mightypy
```

Or, from github:

```
$ pip install git+https://github.com/nishantbaheti/mightypy
```

CHAPTER

TWO

CONTENTS

2.1 mightypy package

2.1.1 Subpackages

2.1.1.1 mightypy.data_structures package

Module contents

mightypy.dsa

class BST

Bases: object

insert(*val*)

traverse(*order='level'*)

class BinaryTree

Bases: object

property height

insert(*val*)

Level order insertion.

https://en.wikipedia.org/wiki/Breadth-first_search

Parameters

val (*Any*) – value of the node in tree.

invert()

traverse(*order='in'*, *method='stack'*)

Tree traversal operation

Parameters

order (*str, optional*) – order in which the tree will be traversed. Defaults to “in”. Options available - “level”,”in”,”pre”,”post”

Raises

ValueError – If wrong order is passed. only “level”,”in”,”pre”,”post” is allowed

Returns
Values of tree nodes in specified order

Return type
values (list)

`class LinkedList`

Bases: `object`

`append`(*value*)

Insert at the end.

Parameters
value (*object*) – Value to insert in linked list

`push`(*value*)

Push at the begining.

Parameters
value (*object*) – Value to push to list

`traverse`()

Traverse the linked list.

Returns
linked list values

Return type
node_values (list)

`binary_search`(*arr: List[int | float | str]*, *ele: int | float | str*) → Tuple[bool, int] | Tuple[bool, None]

Binary search algorithm

Parameters

- **arr** (*List[Union[int,float,str]]*) – array list to search
- **ele** (*Union[int, float, str]*) – element to search

Returns
result of bianry search

Return type
Union[Tuple[bool, int], Tuple[bool, None]]

`linear_search`(*arr: List[int | float | str]*, *ele: int | float | str*) → Tuple[bool, int] | Tuple[bool, None]

Linear search algorithm

Parameters

- **arr** (*List[Union[int,float,str]]*) – array list to search
- **ele** (*Union[int, float, str]*) – element to search

Returns
result of linear search

Return type
Union[Tuple[bool, int], Tuple[bool, None]]

2.1.1.2 mightypy.make package

Module contents

mightypy.make

rotation_matrix_2d(*theta: float*) → ndarray

Create 2D data rotation matrix.

Reference article

https://en.wikipedia.org/wiki/Rotation_matrix

param theta
angle for rotation.

type theta
float

returns
rotation matrix.

rtype
np.ndarray

sine_wave_from_sample(*n_samples: int*, *signal_freq: float*, *n_cycles: int = 10*, *amplitude: int = 1*, *amp_shift: int = 0*, *phase_shift: int = 0*) → Tuple[ndarray, ndarray, ndarray]

Sine wave generation with number of samples and signal frequency

Reference:
https://machinelearningexploration.readthedocs.io/en/latest/MathExploration/Fourier.html#Sine-wave

Parameters

- **n_samples** (*int*) – number of samples.
- **signal_freq** (*float*) – signal frequency.
- **n_cycles** (*int, optional*) – number of cycles. Defaults to 10.
- **amplitude** (*int, optional*) – signal amplitude. Defaults to 1.
- **amp_shift** (*int, optional*) – amplitude shift. Defaults to 0.
- **phase_shift** (*int, optional*) – phase shift. Defaults to 0.

Returns
signal wave, time, freq.

Return type
Tuple[np.ndarray, np.ndarray, np.ndarray]

sine_wave_from_timesteps(*signal_freq: float*, *time_step: float*, *amplitude: int = 1*, *amp_shift: int = 0*, *phase_shift: int = 0*) → Tuple[ndarray, ndarray, ndarray]

Sine wave generation with time steps and signal frequency

Parameters

- **signal_freq** (*float*) – singal frequency.

- **time_step** (*float*) – time step.
- **amplitude** (*int, optional*) – amplitude. Defaults to 1.
- **amp_shift** (*int, optional*) – amplitude shift. Defaults to 0.
- **phase_shift** (*int, optional*) – phase shift. Defaults to 0.

Returns
signal wave, time, freq.

Return type
Tuple[np.ndarray, np.ndarray, np.ndarray]

spiral_data(*data_limit: int = 30*, *n_classes: int = 2*, *n_samples_per_class=300*) → Tuple[ndarray, ndarray]

Generate spiral data for classification problem.

Parameters

- **data_limit** (*int, optional*) – range of data. Defaults to 30.
- **n_classes** (*int, optional*) – number of classes for classification. Defaults to 2.
- **n_samples_per_class** (*int, optional*) – number of samples per classes. Defaults to 300.

Returns
X,y.

Return type
Tuple[np.ndarray, np.ndarray]

2.1.1.3 mightypy.ml package

Module contents

2.1.1.4 mightypy.signal_processing package

Module contents

mightypy.signal_processing

class PSDDenoiser(*threshold: int | float | str | None = 'auto-mean'*)

Bases: object

PSD (Power Spectral Density) Based Denoiser

This method takes the FFT transform of the signal to calculate PSD based on the PSD results and cutoff threshold the signal is filtered and a FFT inverse is applied to regenerate denoised signal.

Parameters
threshold (*Optional[Union[int, float, str]], optional*) – threshold to create cut-off mask, but any threshold can be applied, if it is precalculated by any method chosen by the process, by default auto-mean { auto-mean, auto-max }

Examples

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from mightypy.preprocessing import PSDDenoiser
>>> rng = np.random.default_rng()
>>> fs = 10e3
>>> N = 100
>>> amp = 2 * np.sqrt(2)
>>> freq = 1234.0
>>> noise_power = 0.001 * fs / 2
>>> time = np.arange(N) / fs
>>> X = amp * np.sin(2 * np.pi * freq * time)
>>> X += rng.normal(scale=np.sqrt(noise_power), size=time.shape)
```

```
>>> denoiser = PSDDenoiser()
>>> cleaned_signal = denoiser.transform(X)
>>> plt.plot(X, label="noisy")
>>> plt.plot(cleaned_signal, label="cleaned")
>>> plt.title(f"Threshold : {denoiser.threshold}")
>>> plt.legend(loc="best")
>>> plt.show()
```

```
>>> denoiser = PSDDenoiser(10)
>>> cleaned_signal = denoiser.transform(X)
>>> plt.plot(X, label='noisy')
>>> plt.plot(cleaned_signal, label='cleaned')
>>> plt.title(f"Threshold : {denoiser.threshold}")
>>> plt.legend(loc='best')
>>> plt.show()
```

property f_hat: ndarray | None

FFT of input signal

property filtered_f_hat: ndarray | None

filtered FFT of input signal

psd(*f_hat: ndarray*, *tau: int*) → ndarray

Power Spectral Density

Parameters

- **f_hat** (*np.ndarray*) – Signal in Frequency Domain
- **tau** (*int*) – Interval

Returns

Power spectrum

Return type

np.ndarray

property threshold: str | float | int

Threshold calculated by the process

In Power spectrum after the half length it takes the aggregation of the values and use that as a threshold to cutoff frequencies that are insignificant.

Returns
cutoff threshold value

Return type
Union[str, float, int]

`transform`(*X: ndarray*) → ndarray

Apply PSD

Parameters
X (*np.ndarray*) – Input matrix, signal in IOT Terms.

Returns
Denoised Signal

Return type
np.ndarray

2.1.1.5 mightypy.stats package

Module contents

mightypy.stats

`class WOE_IV`(*event: str*, *non_event: str*, *target_col: str*, *bucket_col: str*, *value_col: str | None = None*, *agg_func: ~typing.Callable = <function count_nonzero>*, *bucket_col_type: str = 'continuous'*, *n_buckets: int = 10*)

Bases: `object`

Weight of Evidence and Information Value.

References

https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html

Parameters

- **event** (*str*) – event name. Generally label true/1.
- **non_event** (*str*) – non event name. Generally label false/0.
- **target_col** (*str*) – Target column name.
- **value_col** (*str*) – Value column name to aggregate(count). Defaults to None.
- **bucket_col** (*str*) – bucketing column name.
- **agg_func** (*Callable, optional*) – Aggregation function name. Defaults to np.count_nonzero.
- **bucket_col_type** (*str, optional*) – Bucketing columns value type. If discrete buckets will not be created else buckets will be created. Defaults to 'continuous'.
- **n_buckets** (*int, optional*) – If bucket column has continuous values then create aritificial buckets. Defaults to 10.

Examples

```
>>> from sklearn.datasets import load_breast_cancer
>>> from mightypy.stats import WOE_IV
```

```
>>> dataset = load_breast_cancer(as_frame=True)
>>> df = dataset.frame[['mean radius', 'target']]
>>> target_map = {0: 'False', 1: 'True'}
>>> df['label'] = df['target'].map(target_map)
```

```
>>> obj = WOE_IV(event='True', non_event='False', target_col='label',
>>>              bucket_col='mean radius')
```

```
>>> cal_df, iv = obj.values(df)
>>> fig = obj.plot()
>>> fig.tight_layout()
>>> fig.show()
```

or directly

```
>>> fig, ax = obj.plot(df)
>>> fig.show()
```

plot(*df: DataFrame | None = None*, *figsize=(10, 5)*) → Figure

Plot weight of evidence and subsequent plots.

Parameters

- **df** (*Optional[pd.DataFrame], optional*) – Input dataframe. Defaults to None.
- **figsize** (*tuple, optional*) – Figure size. Defaults to (10, 5).

Raises

ValueError – If dataframe doesn't exist either in the model or in method args.

Returns

matplotlib figure.

Return type

plt.Figure

values(*df: DataFrame | None = None*) → Tuple[DataFrame, float]

Returns weight of evidence and information value for given dataframe.

Parameters

df (*Optional[pd.DataFrame], optional*) – Input dataframe. Defaults to None.

Raises

ValueError – If input dataframe does not exist either in the model or in method input args.

Returns

calculated dataframe and information value.

Return type

Tuple[pd.DataFrame, float]

population_stability_index(*expected: list | ndarray*, *actual: list | ndarray*, *data_type: str*) → DataFrame

Populaion Stability Index.

References

https://www.listendata.com/2015/05/population-stability-index.html

Parameters

- **expected** (*Union[list, np.ndarray]*) – Expected values.
- **actual** (*Union[list, np.ndarray]*) – Actual values.
- **data_type** (*str*) – Type of data. Helps in bucketing.

Returns
calculated dataframe.

Return type
pd.DataFrame

Examples

```
>>> import numpy as np
>>> from mightypy.stats import population_stability_index
```

continuous data

```
>>> expected_continuous = np.random.normal(size=(500,))
>>> actual_continuous = np.random.normal(size=(500,))
>>> psi_df = population_stability_index(expected_continuous, actual_continuous,␣
↪data_type='continuous')
>>> psi_df.psi.sum()
```

discrete data

```
>>> expected_discrete = np.random.randint(0,10, size=(500,))
>>> actual_discrete = np.random.randint(0,10, size=(500,))
>>> psi_df = population_stability_index(expected_discrete, actual_discrete,␣
↪data_type='discrete')
>>> psi_df.psi.sum()
```

2.1.2 Module contents

2.1.2.1 mightypy

2.2 Contributing

Welcome to `mightypy` contributor's guide.

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, Python Software Foundation's Code of Conduct is a good reference in terms of behavior guidelines.

2.2.1 Issue Reports

If you experience bugs or general issues with `mightypy`, please have a look on the issue tracker. If you don't see anything useful there, please feel free to fire an issue report.

Tip: Please don't forget to include the closed issues in your search. Sometimes a solution was already reported, and the problem is considered **solved**.

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

2.2.2 Documentation Improvements

You can help improve `mightypy` docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

`mightypy` documentation uses Sphinx as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way was a code contribution.

When working on documentation changes in your local machine, you can compile them using `tox`:

```
tox -e docs
```

and use Python's built-in web server for a preview in your web browser (`http://localhost:8000`):

```
python3 -m http.server --directory 'docs/_build/html'
```

2.2.3 Code Contributions

2.2.3.1 Submit an issue

Before you work on any non-trivial code contribution it's best to first create a report in the issue tracker to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

2.2.3.2 Create an environment

Before you start coding, we recommend creating an isolated virtual environment to avoid any problems with your installed Python packages.

python -m venv .venv source .venv/bin/activate

2.2.3.3 Clone the repository

1. Create an user account on GitHub if you do not already have one.
2. Fork the project repository: click on the *Fork* button near the top of the page. This creates a copy of the code under your account on GitHub.
3. Clone this copy to your local disk:

```
git clone git@github.com:YourLogin/mightypy.git
cd mightypy
```

4. You should run:

```
pip install -U pip
pip install -U tox -e .
```

to be able to import the package under development in the Python REPL.

2.2.3.4 Implement your changes

1. Create a branch to hold your changes:

```
git checkout -b my-feature
```

and start making changes. Never work on the main branch!

2. Start your work on this branch. Don't forget to add docstrings to new functions, modules and classes, especially if they are part of public APIs.
3. Add yourself to the list of contributors in `AUTHORS.rst`.
4. When you're done editing, do:

```
.. todo:: if you are not using pre-commit, please remove the following item:
```

Please make sure to see the validation messages from `pre-commit` and fix any eventual issues. This should automatically use flake8/black to check/fix the code style in a way that is compatible with the project.

Important: Don't forget to add unit tests and documentation in case your contribution adds an additional feature and is not just a bugfix.

Moreover, writing a descriptive commit message is highly recommended. In case of doubt, you can check the commit history with:

```
git log --graph --decorate --pretty=oneline --abbrev-commit --all
```

to look for recurring communication patterns.

---

5. Please check that your changes don't break any unit tests with:

```
tox
```

(after having installed `tox` with `pip install tox` or `pipx`).

You can also use `tox` to run several other pre-configured tasks in the repository. Try `tox -av` to see a list of the available checks.

2.2.3.5 Submit your contribution

2.2.3.6 Troubleshooting

2.2.4 Maintainer tasks

2.2.4.1 Releases

2.3 License

2.4 Contributors

- Nishant Baheti <nishantbaheti.it19@gmail.com>

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX