rhf Documentation

Release 0.0.1

Andrian Putina

Sep 21, 2020

Contents:

CHAPTER 1

rhf

Python implementation of Random Histogram Forest (RHF)

- Free software: MIT license
- Documentation: https://rhf.readthedocs.io.

1.1 Features

- TODO

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install rhf, run this command in your terminal:

```
$ pip install rhf
```

This is the preferred method to install rhf, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for rhf can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/anrputina/rhf
```

Or download the tarball:

```
$ curl -OJL https://github.com/anrputina/rhf/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use rhf in a project:

```
from rhf import RHF

my_rhf = RHF(num_trees = 100, max_height = 5, split_criterion='kurtosis')
output_scores = my_rhf.fit(data)
```

CHAPTER 4

rhf

4.1 rhf package

4.1.1 Submodules

4.1.2 rhf.rhf module

Main module.

class rhf.rhf.**Node**
Bases: object

Node object

class rhf.rhf.**RHF**(*num_trees=100*, *max_height=5*, *split_criterion='kurtosis'*, *check_duplicates=True*)
Bases: object

Random Histogram Forest. Builds and ensemble of Random Histogram Trees

Parameters

- **num_trees** (*int*) – number of trees
- **max_height** (*int*) – maximum height of each tree
- **split_criterion** (*str*) – split criterion to use - 'kurtosis' or 'random'
- **check_duplicates** (*bool*) – check duplicates in each leaf

check_hash(*data*)
Checks if there are duplicates in the dataset

Parameters **data** – dataset

fit(*data*)
Fit function: builds the ensemble and returns the scores

Parameters **data** – the dataset to fit

Return scores anomaly scores

get_hash(*data*)
Builds hash of data for duplicates identification

Parameters **data** – dataset

class rhf.rhf.**RandomHistogramTree**(*data=None*, *max_height=None*, *split_criterion='kurtosis'*)
Bases: object

Random Histogram Tree object

Parameters

- **max_height** (*int*) – max height of the tree
- **split_criterion** (*bool*) – split criterion to use: 'kurtosis' or 'random'

build(*node*, *data*)
Function which recursively builds the tree

Parameters

- **node** – current node
- **data** – data corresponding to current node

build_tree(*data*)
Build tree function: generates the root node and successively builds the tree recursively

Parameters **data** – the dataset

generate_node(*depth=None*, *parent=None*)
Generates a new new

Parameters

- **depth** (*int*) – depth of the node
- **parent** (Node) – parent node

set_leaf(*node*, *data*)
Transforms generic node into leaf

Parameters

- **node** – generic node to transform into leaf
- **data** – node data used to define node size and data indexes corresponding to node

class rhf.rhf.**Root**
Bases: *rhf.rhf.Node*

Node (Root) object

rhf.rhf.**get_kurtosis_feature_split**(*data*)
Get attribute split according to Kurtosis Split

Parameters **data** – the dataset of the node

Returns

- feature_index: the attribute index to split
- feature_split: the attribute value to split

rhf.rhf.**get_random_feature_split**(*data*)
Get attribute split according to Random Split

Parameters **data** – the dataset of the node

Returns

- feature_index: the attribute index to split
- feature_split: the attribute value to split

4.1.3 Module contents

Top-level package for rhf.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/anrputina/rhf/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

rhf could always use more documentation, whether as part of the official rhf docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/anrputina/rhf/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *rhf* for local development.

1. Fork the *rhf* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/rhf.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv rhf
$ cd rhf/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 rhf tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/anrputina/rhf/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_rhf
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Andrian Putina <anr.putina@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.0.1 (2020-09-16)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index