tprojection Documentation

Release 0.1.0

Gregoire Hornung

Jun 22, 2020

Contents:

CHAPTER 1

tprojection

This library allows you to visually inspect the relation between a dependent variable (the target) and a predictor in a meaningful way. This library is particularly convenient when the target and/or the predictor are categorical, ie when it is difficult to compute a traditionnal correlation coefficient. And by the way, Tprojection stands for target projection.

1.1 Installation

1.2 Basic usage

1.3 Advanced usage

You can find several examples depicting more advanced functionalities in `examples/examples.ipynb`

1.4 Documentation

Please find a light documentation here

1.5 Credits

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

CHAPTER 2

Installation

2.1 Stable release

To install tprojection, run this command in your terminal:

```
$ pip install tprojection
```

This is the preferred method to install tprojection, 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 tprojection can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/greghor/tprojection
```

Or download the tarball:

```
$ curl -OJL https://github.com/greghor/tprojection/tarball/master
```

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

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

CHAPTER 3

Usage

To use tprojection in a project:

```
import tprojection
```

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/greghor/tprojection/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/greghor/tprojection/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/tprojection.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 tprojection
$ cd tprojection/
$ 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 tprojection 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.

4.3 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.org/greghor/tprojection/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ pytest tests.test_tprojection
```

4.5 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 5

Credits

5.1 Development Lead

- Gregoire Hornung <greghor4@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

6.1 tprojection

class `tprojection.core.`**`Tprojection`**(*df*, *target*, *feature*, *target_type=''*, *feature_type=''*, *target_modality=''*, *nb_buckets=0*, *n_estimators=1*, *continuous_threshold=0.05*)

this class allows to study the relation between the target and a single feature, with the specificity to display a chart type adapted to the type of the input variables (categorical or continuous)

Parameters

- **df** (*pandas DataFrame*) –
- **target** (*string*) –
- **feature** (*string*) –
- **target_type** (*string*) – can take the values "categorical" or "continuous"
- **feature_type** (*string*) – can take the values "categorical" or "continuous"
- **target_modality** (*string*) – will be used for multiclass problem (not implemented yet)
- **nb_buckets** (*int (0)*) – if > 0, encode feature on nb_buckets dummy modalities if the cardinality is to high
- **n_estimators** (*int (1)*) – if > 1, use boostrapping to evaluate estimator variance (only relevant for categorical target and features)

`tprojection.utils.`**`get_encoding`**(*df*, *target*, *feature*, *nb_buckets*)

Encode the feature modalities on a maximum of nb_buckets

Parameters

- **`df`** (*`pandas DataFrame`*) –
- **`target`** (*`str`*) –
- **`feature`** (*`str`*) –
- **`nb_buckets`** (*`int`*) –

Returns

Return type Dict()

`tprojection.utils.`**`is_continuous`**(*s*, *thresh*)

Return true if the series is continuous

Parameters

- **`s`** (*`pandas Series`*) –
- **`thresh`** (*`float`*) –

Returns

Return type Boolean

`tprojection.datasets.`**`load_data`**(*dataset*)

load test dataset, possible options are: - titanic

Parameters **`dataset`** (*`str`*) – required dataset

Returns

Return type pandas DataFrame

Python Module Index

Index