sstudentt Documentation

Release 0.1.1

Jonathan Berrisch

Oct 12, 2021

CONTENTS:

CHAPTER

ONE

A PYTHON IMPLEMENTATION OF THE SKEWED STUDENT-T DISTRIBUTION.

This package implements the skewed student-t distribution in python. Parameterized as described in Wurtz et. al (2006)[1]. An implementation in R is already existent[2].

1.1 Features

- Evaluate the density function
- Evaluate the cumulative distribution function
- Evaluate the quantile function
- Generate random numbers

1.2 References

1.3 Licence

Free software: GNU General Public License v3

1.4 Documentation

Documentation: https://sstudentt.readthedocs.io.

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

[1] Wurtz, Y. Chalabi, and L. Luksan. Parameter estimation of arma models with garch/aparch errors. an r and splus software implementation. Journal of Statistical Software, 2006.

[2] R Implementation: https://www.gamlss.com/wp-content/uploads/2018/01/DistributionsForModellingLocationScaleandShape.pdf

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install sstudentt, run this command in your terminal:

```
$ pip install sstudentt
```

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

You can either clone the public repository:

```
$ git clone git://github.com/berrij/sstudentt
```

Or download the tarball:

```
$ curl -OJL https://github.com/berrij/sstudentt/tarball/master
```

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

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

CHAPTER

THREE

USAGE

This page demonstrates the usage of the *sstudentt.SST* Class.

3.1 Importing the Class

```
>>> from sstudentt import SST
```

3.2 Initialize a Class Instance

Now, create an instance of the *sstudentt.SST* class as follows:

```
>>> dist = SST(mu = 1, sigma = 1, nu = 1, tau = 5)
```

Note: This distribution is only defined for tau > 2 it will return NaN if you set tau to <= 2.

3.3 Calculate Densities

You can evaluate the density of your distribution using the .d method:

```
>>> dist.d(5)
array(0.00192913)
```

3.4 Calculate Probabilities

To evaluate the cumulative distribution function use .p:

```
>>> dist.p(5)
array(0.99821359)
```

3.5 Calculate quantiles

Calculate quantiles with the .q method as follows:

```
# Calculate the Median
>>> dist.q(0.5)
array(1.)
```

Note: Since dist.nu equals 1 we have defined a symmetric distribution. That is, the median equals the mean (dist.mu).

3.6 Draw Random Numbers

```
# Draw 5 random realizations
>>> dist.r(5)
array([3.05375391, 1.34209471, 1.01463769, 1.87961664, 1.58893329])
```

Note: You can also define the shape of the return array to draw multiple random numbers as follows. Note that this only works when all class parameters (mu, sigma, nu tau) are defined as scalars. If (some of them) are arrays .r will always return an array of random values that matches the respective input shape

```
# Draw 5 random realizations
>>> dist.r((4,5))
array([[ 1.92072641,  0.60935071,  2.13692281,  0.66015911,  3.11887499],
       [ 2.08452098, -0.3657303 ,  0.95636288,  2.67946154,  0.89610456],
       [ 1.13357025, -0.26609876,  2.32864548,  0.79109498,  2.00020994],
       [ 0.64556586,  1.32889601, -0.49943665, -0.14925501,  1.11598305]])
```

3.7 Use an array of parameter values

It's possible to intialize the distribution using arrays for the parameters.

For demonstration purposes we will define 2 arrays:

```
>>> arr_1 = np.array([[1, 3], [3, 7]])
>>> arr_2 = np.array([[7, 3], [3, 1]])
```

You can use these arrays to instantiate a distribution as follows:

```
>>> dist2 = SST(mu = arr_1, sigma = arr_2, nu = 2, tau = 4)
```

As you can see, it's possible to mix arrays (of equal size) with scalars.

The methods will now return an array of the same shape:

```
>>> dist2.p(2)
array([[6.63755107e-01, 4.35802430e-01],
       [4.35802430e-01, 1.21990298e-05]])
```

Its even possible to use an array (of the same shape) as method input:

```
>>> dist2.p(arr_2)
array([[8.57842312e-01, 6.04032453e-01],
       [6.04032453e-01, 5.29846717e-06]])
```

This does not work with the .r method.

Warning: The functions are relatively robust against arrays of different sizes because it uses the numpy broadcasting for casting arrays together. This can, however, create results which might be hard to interpret. Therefore, I strongly recommend sticking to one of the following for parameter definition:

- Scalars for all parameters
- Arrays of the same shape for all parameters
- A mixture of scalars and same shaped arrays

CHAPTER

FOUR

SST CLASS

class sstudentt.**SST**(*mu*, *sigma*, *nu*, *tau*)

Creates an Instance of the Skewed Student T Distribution. In this parameterization the expectation equals mu and standard deviation equals sigma.

Parameters

- **mu** (*scalar or array_like*) – mu parameter
- **sigma** (*scalar or array_like*) – sigma parameter
- **nu** (*scalar or array_like*) – nu parameter
- **tau** (*scalar or array_like*) – tau parameter

CHAPTER

FIVE

SST METHODS

SST.**d**(*y*)
Density Function

Parameters **y** (*scalar or array_like*) – distribution values

Returns density at the specified y values

Return type array

SST.**p**(*q*)
Distribution Function

Parameters **q** (*scalar or array_like*) – value

Returns The probability that the SST distributed variable will take

a value less than or equal to q. :rtype: array

SST.**q**(*p*)
Quantile Function / Inverse CDF / Percent Point Function

Parameters **p** (*scalar or array_like*) – probabilities

Returns Quantile values corresponding to the specified probabilities.

Return type array

SST.**r**(*n=1*)
Draws Random Numbers which Follow the SST Distribution

Parameters **n** (*int or tuple of return shape, optional*) – sample size

Returns random sample drawn from the SST distribution

Return type array

Note: n is ignored if the distribution parameters are provided as arrays. In that case, a sample with the shape of the provided arrays will be drawn. i.e. n = 1.

CHAPTER

SIX

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:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/berrij/sstudentt/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.

6.1.2 Fix Bugs

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

6.1.3 Implement Features

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

6.1.4 Write Documentation

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

6.1.5 Submit Feedback

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

6.2 Get Started!

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

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

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

6.3 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/berrij/sstudentt/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ pytest tests.test_sstudentt
```

6.5 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

SEVEN

CREDITS

7.1 Development Lead

- Jonathan Berrisch <jonathan@berrisch.biz>

7.2 Contributors

None yet. Why not be the first?

CHAPTER

EIGHT

HISTORY

8.1 0.1.1 (2021-06-05)

- Fix malformed README file

8.2 0.1.0 (2021-06-05)

- Moving to beta state
- Use rtd-sphinx-theme for the documentation
- Update dev requirements

8.3 0.0.5 (2020-04-19)

- First release on PyPi
- Use pydata-sphinx-theme for the documentation

8.4 0.0.3 (2020-04-20)

- Update Documentation

8.5 0.0.3 (2020-04-19)

- Automatic deployment on Test-PyPi via travis

8.6 0.0.2 (2020-04-19)

- Import SST class directly

8.7 0.0.1 (2020-04-19)

- First release on Test-PyPI.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX