markov Documentation

Release 0.1.0

Xin Bian

Nov 07, 2018

Contents

Contents:

CHAPTER 1

markov

Markov Chain Monte Carlo Project

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

1.1 How to use

- run the code /markov/markov.py
- the input parameters are steps (istep), weights related parameters T and r (T, r), total nodes number (m), the nodes location in the 2-D grid (init.coor([_,_,...], [_,_,...]))
- the code generates a m*m 2D grid, origin is 0, dx=dy=1. For example, if m=3, meaning we have 3 nodes and a 3*3 grid. We can initialize the nodes by init.coor([2,1,1],[1,0,1]). The first, second, and third nodes are located at (2,1), (1,0), (1,1), respectively.

1.2 Features

- This is a Markov Chain Monte Carlo code to estimate the graphs that arise in a distribution network. The Markov Chain is a sequence of graphs. We do not know the transition probability matrix, but we can compute the relative probability of two graphs. See more description in problem_description.
- The code employs Metropolis-Hastings Algorithm
- The proposal probability is based on randomly cutting/adding an edge. There are three cases, 'no cut' case, 'no add' case and normal case.

1. 'no cut' case. If the graph is disconnected by further cutting any edges, it is the so called 'no cut' case. Thus, the probability of adding an edge is 1.

1. P(jli)=1/(total possible edges - edges already exist).

2. We need to cut an edge to go back to previous graph. After adding an edge, if it becomes 'no add' case, P(ilj)=1/(existing edges - edges cannot be cut). After adding an edge, if it is a normal case, P(ilj)=0.5/(existing edges - edges cannot be cut)

2. 'no add' case. If the graph cannot add any more edges, it is 'no add' case. The probability of removing an edge is 1.

1. P(jli)=1/(existing edges - edges cannot be removed)

2. We need to add an edge to go back to previous graph. After cutting, if it becomes cannot cut case, P(ilj)=1/(total possible edges - edges already exist). If it's normal case, P(ilj)=1/(total possible edges - existing edges).

3. Normal case. The probability of adding or removing is 0.5.

   If add an edge, P(jli)=1/(total possible edges - existing edges). If cut an edge, P(jli)=1/(existing edges - edges cannot be removed) The calculation of P(ilj) is similar to previous cases.

1.3 Sample results

- parameters

1. 5 nodes; m=5
2. r=2, T=10
3. total steps; istep=30000
4. initial nodes position (0,0) (1,2) (1,3) (3,2) (4,4); init.coord([0,1,1,3,4],[0,2,3,2,4])

- results

1. 2 most possible graphs: graph1 and graph2
2. expected number of edges connected to vertex 0 is 1.97
3. expected number of edges is 4.96
4. expected maximum distance is 6.64
5. this shows time series of averaged quantities

1.4 Credits

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

CHAPTER 2

Installation

2.1 Stable release

To install markov, run this command in your terminal:

```
$ pip install markov
```

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

You can either clone the public repository:

```
$ git clone git://github.com/xinbian/markov
```

Or download the tarball:

```
$ curl  -OL https://github.com/xinbian/markov/tarball/master
```

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

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

CHAPTER 3

Usage

To use markov in a project:

```
import markov
```

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/xinbian/markov/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

markov could always use more documentation, whether as part of the official markov 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/xinbian/markov/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 *markov* for local development.

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

```
$ git clone git@github.com:your_name_here/markov.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 markov
$ cd markov/
$ 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 markov tests
$ python setup.py test or py.test
$ 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 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/xinbian/markov/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_markov
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search