

Web Science A0: Preparatory Exercise

Denis Helic Elisabeth Lex Robert Thomann Thomas Wurmitzer

March 12, 2017

Prerequisites

Before you can start solving the tasks for assignment **A0** you'll need to ensure that you've installed Python 3.5+ on your development machine. All tasks have been tested with Python 3.6.0, every version of Python from 3.5+ should work though. However please note that we do not officially support it and thus cannot offer support if something goes awry.

Assuming you've successfully setup your Python environment, you'll also need the following external modules/libraries. To avoid version related problems please try to stick as close to the recommended versions below or use `pip` to resolve them.

Additional Dependencies

Besides a working Python environment you'll need the following external Python libraries:

- `numpy == 1.12.0`
- `networkx == 1.11`
- `matplotlib == 2.0`

If you have `pip` installed you can fetch and install those external dependencies in one go by running `pip install -r requirements.txt` inside the assignment folder or your virtual environment (YMMV).

Introduction

The goal of this exercise is to introduce you to Python, `networkx`, and most importantly some of the concepts, ideas and metrics you've learned or heard about in this year's lectures. Since the workload for this preparatory assignment is not that high, this should be seen as a chance for experimentation and exploration.

The assignment is split into five (small) tasks, each of which comes with a `skeleton` that you *can* use as your implementation's foundation and a set of tests that you *can* use to check against your implementation.

Folder Structure

Below is a simple visualization of the directory structure you should find inside the folder after you extracted the archive from the website.

- `a0/`
 - `Makefile` - See *Makefile*
 - `README.pdf`
 - `clustering.py`
 - `components.py`
 - `degree.py`
 - `distance.py`
 - `scatter.py`
 - `requirements.txt` - required external libraries (see *Prerequisites*).
 - `graph-a.gml.gz` - a *random* random graph.
 - `graph-b.gml.gz` - another *random* random graph.
 - `tests/`
 - * `__init__.py`
 - * `test_clustering.py`
 - * `test_components.py`
 - * `test_degree.py`
 - * `test_distance.py`
 - * `test_scatter.py`

Makefile

The `Makefile` packaged with your assignment skeleton provides you with a few convenience functions for building and testing your implementation.

- Issuing `make` or `make all` inside your assignment folder will execute all tasks one by one. To manually run a specific task say `degree`, issue `make degree.json` or simply throw the source file at the interpreter by using `python degree.py`.
- `make test` will start running all test cases inside the `tests` folder. Alternatively you can run a specific test case manually by issuing e.g. `python -m unittest tests.test_degree` in your assignment folder.

Tasks

The provided skeletons for each task already tuck away most of the work regarding plots and showing the results. This assignment consists of the following five tasks:

1. `degree.py`
2. `distance.py`
3. `components.py`
4. `clustering.py`
5. `scatter.py`

A detailed description and task-related TODOs can be found in the provided task skeletons. For the tests to work *do not modify the **perform** functions signature and make sure you return the required metrics and data in the specified/required order!*

To run any of the tasks a simple `make <taskname>.json` or `python <taskname>.py` in the `a0` folder should suffice. To check your solutions for errors read the **Testing** section in this document.

If you encounter any kind of problem a short search or consulting the `Python` documentation, which usually provides detailed documentation including simple examples on each subject, might help.

In any other case, please post general questions regarding the assignment tasks to the `tu-graz.lv.web-science`. If you have a particular, specific problem, you can contact us per e-mail as well. Answering delays might be longer, though.

Testing

This assignment comes with a set of testcases. The purpose is two help you verify the results of your code against a previously calculated set of results. However, try not to solely rely on these automated means for testing your solution.

Besides running `make test` to discover and run *all* testcases, they can also be fired up individually by passing the name of the wanted testcase inside the `tests` folder/module (and without the file extension) e.g. to verify the implementation of your `degree.py` implementation type:

```
% python -m unittest tests.test_degree
```

```
....
```

```
-----  
Ran 4 tests in 0.010s
```

```
OK
```

You can add an additional `-v` before the test case to make the output a little more verbose.

The tests to check and verify your implementation and deliverables are provided for your convenience only. *Use them at your own risk!*

Resources

- Library Documentation & Tutorials
 - **networkx**
 - * <http://networkx.github.io/documentation/latest/>
 - * <http://networkx.github.io/documentation/latest/tutorial/>
 - **numpy**
 - * <http://docs.scipy.org/doc/numpy/>
 - **matplotlib**
 - * <http://matplotlib.org/contents.html>
- Python Language Reference
 - <https://docs.python.org/2/reference/index.html>
- Python Tutorials
 - Google's Python Class, <https://developers.google.com/edu/python/>
 - Think Python: How to Think Like a Computer Scientist, <http://www.greenteapress.com/thinkpython/>
 - Code Academy's Python Track, <http://www.codecademy.com/en/tracks/python>
 - The Hitchhiker's Guide to Python!, <http://docs.python-guide.org>

Please post general questions regarding the assignment tasks to the `tu-graz.lv.web-science`. If you have a particular, specific problem, you can contact us per e-mail as well. Answering delays might be longer, though.