Web Science & Web Technology: Assignment I

Denis Helic Elisabeth Lex Fiona Draxler Thomas Wurmitzer

14.03.2016, `#1114658`

Deadlines

- The deadline is 2016-04-18 23:59:59.
- The hard deadline is 2016-04-19 23:59:59. Committing files to the repository after the deadline will result in deduction of points (see Introduction slides)!

Prerequisites

Before you can start solving the tasks for assignment **A1** you'll need to ensure that you've installed Python 2.7.x, Python 3.x on your development machine. Solving the tasks using Python 3 *should* probably work, 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` >= 1.10
- `networkx` >= 1.11
- `matplotlib` >= 1.5

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 (`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 first assignment is not that high, this should be seen as a chance for experimentation and exploration.

The first assignment is split into five (small) tasks which can be solved and submitted individually. Each of the those tasks comes with a `skeleton` that you *should* use as your implementation's foundation and a set of tests that you *can* use to either verify or check your deliverables and prepare your solutions for submission.

Folder Structure

Below is a simple visualization of the directory structure you should find inside the working copy of your repository after **A1** has been handed out.

- **a1/**
  - `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.
  - **submission/**
  - **tests/**
    - `__init__.py`
    - `check_clustering_submission.py`
    - `check_components_submission.py`
    - `check_degree_submission.py`
    - `check_distance_submission.py`
    - `check_scatter_submission.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, verifying and testing your implementation and submission.

- 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 check` will tell you if the results written by a certain or all tasks comply with the assignment requirements i.e. that the files inside the `submission` folder are valid `JSON` files containing the proper `n`-tuple and that `non-zero` plots exist as well. This might help you prepare and verify your deliverables and make your repository ready for submission.
- `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. The same thing also works for the submission checks (see above).
- `make clean` will delete the contents of the `submission` folder as well as *clutter* like `*.pyc` files from your repository.

Tasks

The provided skeletons for each task already tuck away must of the work regarding plots and persisting the results to disk. You **must not modify** the function signature or `return` statement in any of the provided task skeletons. If you do, we might not be able to automatically verify your solutions and you'll receive **no points** for this task!

In this assignment you have to solve the following five tasks.

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

A detailed description and task-related `TODOs` can be found in the provided task skeletons. As already mentioned above: *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 `a1` folder should suffice. To check your solutions for errors and prepare your repository for submission read the **Testing** and **Submission** 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 two sets of test cases. The purpose of the first is two help you verify the results of your code against a previously calculated set of results. The second one only checks if the submission folder contains the corresponding `*.png` and (valid) `*.json` files. However, try not to solely rely on these automated means for testing your solution.

Besides running `make test` and `make check` 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
```

And to check that the required files for the `degree.py` task in the `submission` folder exist:

```
% python -m unittest tests.check_degree_submission
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

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

Compliance with the required structure of your submission directory is an absolute must, and testing your submission can give you more confidence that you fulfilled those requirements. However the tests to check and verify your implementation and deliverables are provided for your convenience only. *Use them at your own risk!* Even submissions that pass all the provided tests might get points deducted or no points, given a proper reason!

Submission

Assuming you've implemented all tasks, did modify the task templates where you were supposed to and ran `make` or `make all` the `submission` folder should contain the following files:

- **submission**
  - `clustering.json`
  - `clustering.png`
  - `components.json`
  - `components.png`
  - `degree.json`
  - `degree.png`
  - `distance.json`
  - `distance.png`
  - `scatter.json`
  - `scatter.png`

Running `make check` or the underlying command by hand might help you verify that the data inside the `*.json` files is valid and properly formatted (see **Testing**).

To submit your solutions simply `add` the `submission` folder **and** its content to your repository and commit them. Furthermore, **don't forget to commit the code** from each of the tasks! Also *only* committing the code without the results in the submission folder will result in zero points for that task!

Policies

- No other external Python libraries are allowed.
- Your Python programs must be executable by invoking `python <taskname>.py` in the assignment folder. No extra parameters must be needed. Furthermore, the `perform` function's `return` statement and number of parameters needed to call the function shall not be changed!
- Your scripts shall not produce any output to the standard output on the console. Use an internal variable to enable debug output, if you want. Furthermore, plots shall not be `shown` (in a window), they shall just be written to the corresponding PNG files.
- You must not use any platform-specific functions.
- All of your programs must not consume more than 4GB of main memory (RAM) each.
- Your code will be checked for plagiarism using automated and manual means.

Resources

- Library Documentation & Tutorials
  - `networkx`
    - http://networkx.github.io/documentation/networkx-1.9/
    - 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.