Homework 05

STAT 387: Data Science II

Assigned 2017–03–22 12:00
Due 2017–04–04 18:00

Goal: You are going to **create a new community detection method**!

- Please read the **entire instructions** before you begin.
- Start the assignment by copying the `HW05/` directory from the course repository into your repository.
- Contact me as **soon as possible** with any questions.

Part 1. Build a communities extension of your network library

Create a file `comms.py` and import your original `nets.py` file (copy `nets.py` from `HW04/` and do not change it, that code is now your legacy!).

The first thing you will need is a **data structure** to store community information within python. What is the best way to do this? Describe and motivate in your writeup the python data structure you use for storing community information.

- For the purposes of this assignment assume you are working with a partitioning method, meaning each node in the network belongs to exactly one community.
- Hint: This data structure should allow you to retrieve the community of a given node and/or the nodes of a given community.

Using your data structure definition, create functions `read_comms` and `write_comms` to read and write community data, respectively.

- What is the best file format to store this information? Describe this file format in your writeup and use this for your functions.

Next, create a function called `modularity` that takes your network and community data and computes Q as defined in class.

- You can implement modularity as a double sum over all nodes (like in lecture) or as a sum over communities (one of the papers included with the lecture notes may be of use here).

Build inside `test_comms.py` a small test suite for the community code. Aim for 90% or more test coverage and include unit tests and regression tests.

Get some real data: Find a small (< 30 mb) network dataset online that you think has modular structure, download it and put it in directory `HW05/data`. Motivate your choice of these data in your writeup.

Part 2. Build a stochastic block model benchmarking platform

Recall the 4-group, 128-node benchmark graph we discussed briefly in class. That is a special case of a stochastic block model. These models are essentially a collection of Erdos-Renyi random graphs “glued” together with a larger and typically sparser erdos-renyi graph that “sits” on all the nodes.

Specifically, the block model consists of K groups with group k having $n(k)$ nodes. The total number of nodes in the graph is then $N = n(1) + n(2) + \ldots + n(K)$. The probability for two random nodes to be linked is a constant *p_in* if they are within the same group and a constant *p_between* if they are not. (Block model formulations allow *p_in* and *p_between* to vary, but let’s not worry about that.) Therefore the model is defined by a list of group sizes (integers which define the **planted partition***)* and two probabilities which are floats between 0 and 1.

- Create the function `blockmodel(list_group_sizes, p_in, p_btwn)` to build these block models as defined above. (The 4-group benchmark discussed in class has `list_group_sizes = [32, 32, 32, 32]`.)
- Do you think these graphs are a reasonable benchmarking choice? Why or why not? Propose at least **two** other alternative benchmarking graph models. (No need to implement them with code unless you want to!)

Investigate your benchmarking graphs by computing Q as a function of these two probabilities for a number of different group size lists. Put a number of plots into your writeup. Do you see from this investigation any evidence for resolution limits or other problems.

Part 3. Design your own community detection method

Following from the lecture materials, describe a **new community detection method** and motivate why it works. Is it a graph partitioning method or are you ambitious enough to try for overlap? What is your objective function (is it modularity?) and why is it a good choice? Is it similar to or different from any methods discussed in class? Answer these questions in your writeup

Part 4. Implement your community detection method using nets.py and comms.py

Next, implement your method with a function called `find_comms`.

Testing

As per HW04. Be sure to report your test coverage statistics.

Grading

You will be graded (gently) on:

- Conciseness and correctness of your choices on community data structure, community file format, the block models (including your investigation of their properties), and the design and implementation of your community

detection method,
- the quality of the `comms.py` code, including easy-to-understand variable names, clear and useful code comments, and **thorough** documentation strings,
- how well your test suite works, including test coverage,
- your usage of git (number of commits and quality of commit messages will be examined!).

Parts 3 and 4 are potentially quite challenging, but don't worry as I will take this into account when grading. Try your best and come to office hours if you need help!

To submit:

1. Create a `HW05` folder inside your personal repository. This should contain all the files asked for and a **pdf of your writeup**. The writeup should address all of the above questions in a clearly organized way and any plots you have are to be included as figures within the document. Any number of other optional files are allowed.
2. Push your repository to the server. **DO NOT COMMIT LARGE DATA FILES.** (If in doubt, ask me about what to commit!)

If you are having problems, please let me know ASAP. Extra office hours are available if needed.