TSKS33 Hands-On Session 5

Fall 2024

Version of this document: November 1, 2024

Preparation Work (to be Done Before Coming to the Lab)

- Study the course material on partitioning and community detection.

Objective

The task in this lab exercise is to implement spectral modularity maximization for bisection of a network into two communities of a priori unknown size.
We will work on four networks:

1. A small test network for which we provide a reference solution (for debugging purposes)
2. The karate club network
3. A small "stochastic block model" (synthetic data with community structure) with two communities
4. A large "stochastic block model" (synthetic data with community structure) with two communities, for which computational aspects are crucial

All networks can be loaded via the function `load_data.py` (select one of the four lines at the end to get the desired network).

Task 1

Implement spectral modularity maximization using the power method. For networks 1–3, "normal" linear algebra routines are sufficient. For the larger network 4, you will need to use the Python package `scipy.sparse` for numerical linear algebra with sparse matrices. Therefore, we recommend that you implement a solution that uses sparse matrices directly and use it for all networks.

To debug your solution you can compare to the reference solution output below for the test network.

Task 2

Run the community detection algorithm on the Zachary karate club network. Layout the network in Gephi with ForceAtlas and comment on the result. Make sure to import the community labels as well. This can be done using the function `save_csrmatrix_Gephi_gexf_twocolors()` that can be found in `save_Gephi_gexf.py` This function makes sure that Gephi colors the nodes according to the community labels identified by your algorithm. Does the community partitioning look intuitive?

Task 3

Run the community detection algorithm on the small synthetic stochastic block model network. Layout the network in Gephi with ForceAtlas and using the community labels computed by the algorithm, and comment on the result. Does the community partitioning look intuitive?

Then, partition the same network using Gephi's built in community partitioning algorithm. Compare the results and in particular the modularity score of your solution with the one given by Gephi (Gephi uses a different algorithm!). Note that Gephi can't compute the modularity of a given community partitioning (but it shows the modularity of its own). Thus, you have to compute the modularity of your solution in the code.

Task 4

Run your community detection algorithm on the large synthetic stochastic block model network. How long did it take to run?

For comparison, construct the modularity matrix ($\boldsymbol{Z}$ in the lecture) *explicitly*, and try to use numpy to find its eigenvalues. (The computer may hang or take a very long time to finish. Why?)

Implementing the Power Method

Finding the Dominant Eigenvector

Given an $N \times N$ symmetric matrix $\boldsymbol{Z}$, with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$, the power method estimates the eigenvalue with the largest magnitude, that is, λ_1 if $|\lambda_1| > |\lambda_N|$ and λ_N otherwise. It also estimates the corresponding eigenvector (known as the dominant eigenvector). The method is simple:

1. Select a non-zero N-vector $\boldsymbol{x}$ randomly.
2. Assign $\boldsymbol{x} := \boldsymbol{Z}\boldsymbol{x}$ and subsequently $\boldsymbol{x} := \boldsymbol{x}/\|\boldsymbol{x}\|$
3. Repeat step #2 until convergence.

At any time (after the normalization in step #2), the current estimate of the eigenvalue is simply $\boldsymbol{x}^T\boldsymbol{Z}\boldsymbol{x}$. For the method to work, the initial $\boldsymbol{x}$ must not be orthogonal to the dominant eigenvector. If the initial $\boldsymbol{x}$ is selected randomly, this would happen with negligible probability, and even if it happens, numerical rounding errors in the iteration typically anyway ensures that after some time $\boldsymbol{x}$ has a component along the dominant eigenvector, and the algorithm converges.

Finding the Eigenvector Corresponding to the Largest Positive Eigenvalue

In the spectral modularity maximization algorithm we want to find the eigenvector associated with the *largest positive* eigenvalue, λ_1. This is not necessarily the dominant eigenvalue (which could be negative). To use the power method to find the largest positive eigenvalue we proceed as follows:

1. Use the power method to find the dominant eigenvalue. Call this eigenvalue $\hat{\lambda}$ and let $\boldsymbol{x}$ be the corresponding eigenvector.
2. If $\hat{\lambda} > 0$ we are done. $\boldsymbol{x}$ is the eigenvector associated with the largest positive eigenvalue.
3. If $\hat{\lambda} < 0$, run the power method again on $\boldsymbol{Z} - \hat{\lambda}\boldsymbol{I}$. The resulting $\boldsymbol{x}$ is the eigenvector associated with the largest positive eigenvalue. (The resulting eigenvalue is an estimate of $\lambda_1 - \lambda_N$.)

Hints

- Remember that: $\boldsymbol{Zx} = \boldsymbol{Ax} - \frac{1}{2M}\boldsymbol{k}\left(\boldsymbol{k}^T\boldsymbol{x}\right)$. Hence, the matrix $\boldsymbol{Z}$ does not need to be explicitly constructed (except for in task 4). Why does this help us?
- The reference solution output for the test network, after 250 iterations, is

  ```
  largest-magnitude eigenvalue: -1.9192868848312763
  largest-positive eigenvalue: 1.3549980049551595
  largest-positive eigenvector:
  [[ 0.37200009]
   [ 0.45877098]
   [ 0.37200009]
   [-0.25823163]
   [-0.47226977]
   [-0.47226977]]
  ```

  (Numerically the values may differ slightly from one run to the next or from one computer to another.)
- The functions in `common-functions/save_Gephi_gexf` are useful to save networks in a format readable by Gephi.
- The function `to_sparse_mat` in `common-functions/snap_scipy.py` can be used to construct a sparse matrix from a SNAP network.

Examination

- Individual oral examination takes place in class (computer lab). Students are also expected to be able to answer questions relating to the course material. All students must study the pertinent course material before coming to the lab.
- Before asking for oral examination, you have to collect all generated plots/figures and answers into a document, for example, in PowerPoint or LibreOffice. Additionally, the code should be open and ready to be run upon request. It is suggested that you have already ran the code once so that the output is in the terminal.
- If a reference/sample solution is provided, you should format your solution the same way.
- Collaboration on this homework in small groups is encouraged, but each student should perform programming work individually, and individually demonstrate understanding of all tasks.

- Library functions from the Python standard library, numpy, scipy, matplotlib, and SNAP may be used freely. Copying of code from the Internet or from other students is prohibited.
- The program code you have written should be uploaded to Lisam (go to "submissions" and then select the lab session number) for an anti-plagiarism check.

  Plagiarism (copying of code from other students, from the Internet, or other sources) is a serious offense at LiU and normally leads to the filing of a disciplinary case.