CS 5683: Big Data Analytics

Project-5: Community Detection using BigCLAM

Total Points: 100 (15% toward final)

Due date: Nov 29, 2020 at 11:59pm

In this project, we will explore network community detection using the BigCLAM algorithm. Specifically, we will develop an algorithm to find overlapping communities in the given network – which means a node can present in multiple communities in the given network. We formulate the community detection as a non-negative matrix factorization with maximum log-likelihood estimation problem by parametrizing node strength in communities.

This project is a simple version of experiments done in https://dl.acm.org/doi/10.1145/2433396.2433471

Project Requirements: ***This is an independent project. Spark implementation is not required***

Dataset: We have provided a well curated data sample of the *YouTube social network*[1]. The given data contains **7675 nodes**, **35,622 edges**, and **29 ground-truth communities**. Node-ids are neatly named from 0 to 7674. The given network is disconnected on-purpose – it has one very big component and a very small component.

Project data files:

1. YouTube.edgelist – networkx edgelist file. This can be loaded directly with networkx
2. groundtruth_communities.txt – space separated file, where each line, with a set of node ids, represent a ground-truth community
3. 20percent_seed_communities.txt, neighborhood_seeds – space separated files, where each line, with a set of node ids, represent a community. These data can be used to initialize all communities

BigCLAM algorithm:

Please refer to the lecture powerpoint slides and the above paper to get a detailed description and mathematical derivations for the BigCLAM algorithm.

In overview, BigCLAM is a better version of AGM to identify overlapping communities in networks. To avoid multiple parametrizations in AGM, BigCLAM introduces an approach which has only one parameter (*Factor Matrix*) to optimize.

For simplicity of this project, we are given the inputs: network $G=(V, E)$ with |V| nodes and |E| edges, number of communities K, and the ground-truth communities C^* for evaluation. In real-world application, the number of communities should be parametrized with the algorithm. Given the inputs, BigCLAM algorithm goes as follows:

[1] http://snap.stanford.edu/data/com-Youtube.html

1. **Factor matrix (*F*) initialization**: F is a non-negative matrix, where the element F_{uc} represent the strength of node **u** in the community **c**
    a. Construct a matrix of size **|V| x K**
    b. Community initialization is the important step to determine the performance of community detection algorithms. We initialize communities by assigning strength values in ***F***, where a value F_{uc} represents the node **u's** level of membership in the community ***c.*** There are several ways to initialize the factor matrix ***F***. For simplicity, we will start with seed communities to initialize available values of ***F***. Seed communities contains a fraction of nodes from the ground-truth communities. With the details from seed nodes in each community, we use some heuristics to decide the value of other nodes in each community.
        i. We particularly use ***Conductance:*** Conductance is the measure of fraction of edges that are connected to nodes outside the given community[2]
        ii. Assume we need to find the strength(F_{uc}) of node ***u*** in a community ***c. Note:*** As we are provided the seed set, each community (***c***) will always have some nodes to represent them. Measure the *conductance* κ_{uc} of ***c*** after adding ***u***
        iii. Measure conductance (κ_{uc}) of ***u*** in each community C_i: $C_i \in C^*$. We call the node ***u*** is *locally minimal* to the community C_i if κ_{uC_i} is lower than that of other communities $\kappa_{uC'}$: $C' \in C^* - C\_i$
        iv. Assign $F_{uc} = \begin{cases} 1 & if\ u\ is\ locally\ minimal\ to\ c \\ 0 & otherwise \end{cases}$
        v. **NOTE-1:** Python's NetworkX package has function to measure conductance of ***C*** after adding ***u*** to it. *You do not need to calculate fraction of edges in the network all the time*
        vi. **NOTE-2:** Measuring conductance of each node in the network across all communities would need more computational power to finish efficiently. For the sake of execution time efficiency, measure the conductance of the node-neighborhood *N* (node and its neighbors) across all communities. Set $F_{uc} = 1$ for all nodes in *N* if the conductance of *N* in *c* is lower than that of other communities
        vii. **NOTE-3:** Measuring conductance will still take between 20-60 minutes without parallel processing
    c. We have provided 2 seed communities to start the initialization and conduct experiment with. Along with this, students will create one simple initialization for ***F***. Thus, students should execute the given algorithm for the following 3 types of initializations to ***F:***
        i. *20percent_seed_communities:* ***20%*** percent random sample from each ground-truth communities. If a seed node ***v*** is present in a community ***c***, then set $F_{vc} = 1$. Determine the remaining values of ***F*** using conductance
        ii. *neighborhood_seeds:* Randomly selected node from a community and its neighbors are selected as seed communities. If a seed node ***v*** is present in a community ***c***, then set $F_{vc} = 1$. Determine the remaining values of ***F*** using conductance
        iii. Initialize all values in ***F*** with random numbers **[0,1]**

2. **Matrix factorization:** Use the efficient version of the BigCLAM algorithm – *BigCLAMV2.0* - to optimize all values of ***F***

[2] Yang, Jaewon, and Jure Leskovec. "Defining and evaluating network communities based on ground-truth." *Knowledge and Information Systems* 42.1 (2015): 181-213.

Algorithm steps:

Begin ***I*** iterations:

Iterate over the rows of ***F***:

a. Compute gradient $\nabla l(F_u)$ of row $\boldsymbol{u}$ (while keeping others fixed) [**NOTE:** Use BigCLAMV2.0 approach to find the gradient]

b. Update the row $F_u \leftarrow F_u + \eta \nabla l(F_u)$ [**NOTE:** $\boldsymbol{\eta}$ is the learning rate. Set $\eta = 0.001$. *Feel free to change this number*]

c. Project $\boldsymbol{F_u}$ back to a non-negative vector: If $\boldsymbol{F_{uc} < 0}$: $\boldsymbol{F_{uc} = 0}$

Some tips on in matrix factorization:

a. Set the number of iterations (***I***) between 300 and 500. Stop iterations either you reach ***I*** iterations or if the increase of $\boldsymbol{F_u}$ is less than 0.001% for all $\boldsymbol{F_u}$

b. Since we are using BigCLAMV2.0, we need to find the vector sum of all nodes: $\sum_w F_w$, where $w \in V$. Update this at the beginning of each iteration and use it in the gradient calculation

3. **Community assignment:** Once we complete the factorization, we need to decide the membership of each node in each community, based on the values in ***F***. We will use the assumption described in the paper by using a threshold **δ**. Set $\delta = \sqrt{-\log(1-\varepsilon)}$, where $\varepsilon = 10^{-8}$. Thus the node $\boldsymbol{u}$ belong to the community $\boldsymbol{c}$, if $F_{uc} \geq \delta$. Otherwise, we ignore the membership

4. **Evaluation:** Given the ground-truth communities $\boldsymbol{C^*}$ and detected communities $\boldsymbol{C^\wedge}$, where each ground-truth community $\boldsymbol{C_i \in C^*}$ and each detected community $\boldsymbol{C_i^\wedge \in C^\wedge}$ is identified by a set of its member nodes. To quantify the level of correspondence between $\boldsymbol{C^*}$ and $\boldsymbol{C^\wedge}$, we use *Recall(C^*, $C^\wedge$)*. We use the following average recall metric in this project:

$$Recall(C^*, C^\wedge) = \frac{1}{|C^*|} \sum_{C_i \in C^*} R_C(C_i, C_j^\wedge)$$

where $C_j^\wedge$ is the best matching detected community to the ground-truth community C_i and R_C is the recall. [**NOTE:** *As you do the optimization, initial communities may shift positions in **F**. So, you may need to find the best matching detected community for each ground-truth community.*]

What to output? (i) Your program should write detected communities in a file, (ii) Print the recall score for each initialization method, and (iii) Come up with an idea for two plots to compare methods and results based on your experience from previous projects – compare and contrast results using the plots [**NOTE:** Two plots should not mean the same – **Example:** *Representing the recall of different initializations in horizontal bar plot and vertical bar plot is the same*]

Submission requirements: Submit only one programming (**.ipynb**/**.py**) file – do not write a separate program for multiple initializations. Each initialization should be handled in corresponding functions.

Grading Rubric:

1. Factor matrix initialization: 20 points
2. Matrix factorization: 30 points
3. Community assignment: 10 points
4. Evaluation: 10 points
5. Results: 20 points
6. Documentation: 10 points