INF 553 – Fall 2017 Assignment 5

Girvan-Newman algorithm

Deadline: 11/29/2017 11:59 PM PST

Assignment Overview (100 points)

In this assignment, you are asked to implement the Girvan-Newman algorithm using the Spark Framework in order to detect communities in the graph. Spark 1.6.1 should be used for this HW. For Scala implementation please use Scala version 2.10. You will use only **ratings.csv** dataset in order to find users who have the similar movie taste. The goal of this assignment is to help you understand how to use the Girvan-Newman algorithm to detect communities in an efficient way by programming it within a distributed environment.

Write your own code!

For this assignment to be an effective learning experience, you must write your own code! Do not share code with other students in the class!!

Submission Details

For this assignment, you will need to turn in a Python or Scala program depending on your language of preference. **There is 10% bonus for Scala implementation.** We will test your code using the same dataset in order to verify its correctness. This assignment will surely need some time to be implemented so please plan accordingly and start early!

- **For Python**, your submission must be a **<Firstname>_<Lastname>_hw5.zip** file containing a python script **<Firstname>_<Lastname>_community.py** which contains both betweenness and modularity functions, a description document and output results (**<firstname>_<lastname>_communities.txt, <firstname>_<lastname>_betweenness.txt)**.

- **For Scala**, your submission must be a **<Firstname>_<Lastname>_hw5.zip** file, in which includes **community.jar** which contains both betweenness and modularity functions and **Scala source codes**, a description document and output results (**<firstname>_<lastname>_communities.txt, <firstname>_<lastname>_betweenness.txt)**.

Execution Example

The program that you will implement should take three parameters. One is the location of input file (i.e. rating.csv), the others are the paths to the output file, including the **firstname_lastname_communities.txt** and **firstname_lastname_betweenness.txt**.
The **main class** name for betweenness should be **Betweenness** (capitalize the first letter). The **main class** name for community should be **Community** (capitalize the first letter). The example of execution example is as follows:

For Scala:

./bin/spark-submit –-master local[*] –class <main_class_name>   <jar_file>   <inputfile_path>/<input_file>
<outputfile_path>/ firstname_lastname_communities.txt   <outputfile_path>/ firstname_lastname_betweenness.txt

For Python:

./bin/spark-submit –-master local[*] <python_script> <inputfile_path> >/<input_file>
<outputfile_path>/firstname_lastname_communities.txt   <outputfile_path>/ firstname_lastname_betweenness.txt

Details

Construct the graph

Each node represents a user. Each edge is generated in following way. In rating.csv, count the number of times that two users rated the same movie. If the number of times is greater or equivalent to **three times**, there is an edge between two users.

Represent the graph

For Python, you can use GraphFrame, you can learn more about GraphFrame from the link:
https://graphframes.github.io/user-guide.html
For Scala, you can use GraphFrame or GraphX, you can learn more about GraphX from the link:
http://spark.apache.org/docs/latest/graphx-programming-guide.html

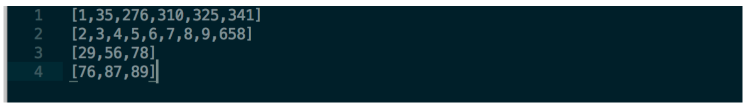
Betweenness and Modularity

You are required to implement betweenness and modularity according to the lecture by yourself. The betweenness function should be calculated on the original graph. You can write a betweenness function and a modularity function, and call two functions several times until finding the max modularity value.

Output Format

- For firstname_lastname_communities.txt

Each list is a community, in which contains userIds. In each list, the userIds should be in ascending order. And all lists should be ordered by the first userId in each list in ascending order. An example is as follows: ***(the example just shows the format, is not a solution***)

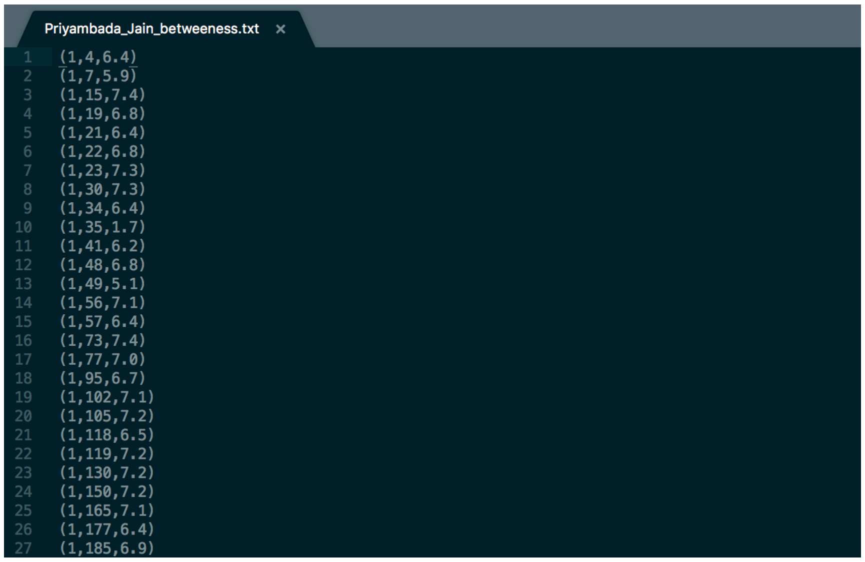


```
[1,35,276,310,325,341]
[2,3,4,5,6,7,8,9,658]
[29,56,78]
[76,87,89]
```

For testing purposes, note that users 1, 35 and 276 are in the same community, 2, 3 and 4 are in the same community, user 29 is an independent community of size 1.

For firstname_lastname_betweenness.txt

Each line is a tuple, the format is like (userId1, userId2, betweenness value). The file is ordered by the first element in ascending order and if the first element is the same, ordered by the second element. The example is as follows:



```
(1,4,6.4)
(1,7,5.9)
(1,15,7.4)
(1,19,6.8)
(1,21,6.4)
(1,22,6.8)
(1,23,7.3)
(1,30,7.3)
(1,34,6.4)
(1,35,1.7)
(1,41,6.2)
(1,48,6.8)
(1,49,5.1)
(1,56,7.1)
(1,57,6.4)
(1,73,7.4)
(1,77,7.0)
(1,95,6.7)
(1,102,7.1)
(1,105,7.2)
(1,118,6.5)
(1,119,7.2)
(1,130,7.2)
(1,150,7.2)
(1,165,7.1)
(1,177,6.4)
(1,185,6.9)
```

Grading

Your codes will be tested on rating.csv file.

The grading guideline is as follows:

- The output does not follow the format, **there will be 20% penalty**.
- The codes cannot be run in the terminal by using the command mentioned above, there will be **20% penalty**.
- **20% penalty is for the late submission**.
- If your execution time **is longer than five minutes**, we will refer to the output files as provided for grading and there will be **20% deduction**.
- **If the result of firstname_lastname_communities.txt is not correct, but firstname_lastname_betweenness.txt is correct, 50% points will be subtracted**.

Appendix:

There are some libraries containing the community detection function. Here is the link to learn more about it: http://sparkling-graph.readthedocs.io/en/latest/comunities.html. The graph is constructed in the same way as mentioned above.