INF553 Foundations and Applications of Data Mining

Summer 2019

Assignment 4

Deadline: July. 1st 11:59 PM PST

1. Overview of the Assignment

In this assignment, you will s implement your own Girvan- Newman algorithm using the Spark Framework to detect communities in graphs. You will use the ub_sample_data.csv dataset to find users who have a similar business 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 within a distributed environment.

2. Requirements

2.1 Programming Requirements

a. You must use Python to implement all tasks. There will be 10% bonus for each task if you also submit a Scala implementation and both your Python and Scala implementations are correct.

b. You can ONLY use Spark RDD and standard Python or Scala libraries.

2.2 Programming Environment

Python 3.6, Scala 2.11 and Spark 2.3.3

We will use these library versions to compile and test your code. There will be a 20% penalty if we cannot run your code due to the library version inconsistency.

2.3 Write your own code

Do not share code with other students!!

For this assignment to be an effective learning experience, you must write your own code! We emphasize this point because you will be able to find Python implementations of some of the required functions on the web. Please do not look for or at any such code!

TAs will combine all the code we can find from the web (e.g., Github) as well as other students' code from this and other (previous) sections for plagiarism detection. We will report all detected plagiarism.

2.4 What you need to turn in

Your submission must be a zip file with the name convention: **firstname_lastname_hw4.zip** (all lowercase, e.g., junyu_liu_hw4.zip). You should pack the following required (and optional) files in the zip file (see Figure 1):

a. [REQUIRED] two Python scripts, named: (all lowercase)

firstname_lastname_task1.py

b1. [OPTIONAL] two Scala scripts, named: (all lowercase)

firstname_lastname_task1.scala

b2. [OPTIONAL] one jar package, named: (all lowercase)

firstname_lastname_hw4.jar

c. [OPTIONAL] You can include other scripts to support your programs and also name it with the prefix: "**firstname_lastname_"** (e.g. junyu_liu_Graph.py)

d. You don't need to include your results. We will grade on your code with our testing data (data will be in the same format).

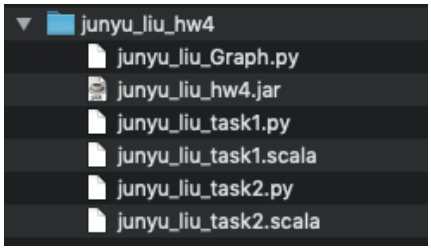

Figure 1: Submission Structure

3. Datasets

You will continue to use Yelp dataset. We have generated a sub-dataset, sample_data.csv, from the Yelp review dataset containing user_id and business_id. You can download it from Blackboard.

4. Tasks

4.1 Graph Construction

To construct the social network graph, each node represents a user and there will be an edge between two nodes if the number of times that two users review the same business is greater than or equivalent to the filter threshold. For example, suppose user1 reviewed [business1, business2, business3] and user2 reviewed [business2, business3, business4, business5]. If the threshold is 2, there will be an edge between user1 and user2.

If the user node has no edge, we will not include that node in the graph.

NOTICE: In this assignment, the filter threshold is 7.

4.2 Task1: Community Detection Based on Girvan-Newman algorithm (12.5 pts)

In, you will implement your own Girvan-Newman algorithm to detect the communities in the network graph. You can refer to the Chapter 10 from the Mining of Massive Datasets book for the algorithm details.

For this homework, you can ONLY use Spark RDD and standard Python or Scala libraries.

4.2.1 Betweenness Calculation (6 pts)

In this part, you will calculate the betweenness of each edge in the original graph.
Then you need to save your result in a **txt** file. The format of each line is

('user_id1', 'user_id2'), betweenness value

```
('-267Yx8RmdP6io2-qI4UcQ', '-TMDrC66dvClx5Z7Hdzrfw'), 8.965476190476188
('-Anyb0vB5LrW273whytNRw', '-TMDrC66dvClx5Z7Hdzrfw'), 7.882142857142856
('OHVhLKw_uIE7saIt_S-udw', 'OVnfDg-KuGWyDocYdKkmPQ'), 7.749999999999999
('-RA9NLalwmRTOX_8UMHnVQ', '0C4ccDiZlnW68ULoDKE3NA'), 6.714583333333334
('-RA9NLalwmRTOX_8UMHnVQ', '-TMDrC66dvClx5Z7Hdzrfw'), 5.738690476190476
('-ShdX4pDKrldKfic9rHhSQ', '0BoO48jZw2kjJjwgwIjbLw'), 4.799107142857144
('-KpEgEen1tj-jdjIS7uVOw', '0BoO48jZw2kjJjwgwIjbLw'), 4.049107142857143
```

Your result should be firstly sorted by the betweenness values in the descending order and then the first user_id in the tuple in **lexicographical** order (the user_id is type of string). The two user_ids in each tuple should also in **lexicographical** order. You do not need to round your result.

Figure 3: betweenness output file format

4.2.2 Community Detection (6.5 pts)

You are required to divide the graph into suitable communities, which reaches the global highest modularity. The formula of modularity is shown below:

Modularity of partitioning S of graph G:

- $\mathbf{Q} = \sum_{s \in S}$ [(# edges within group s) – (expected # edges within group s)]
- $Q(G,S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$

Normalizing cost.: -1<Q<1

A_{ij} = 1 if i connects j, 0 else

According to the Girvan-Newman algorithm, after removing one edge, you should re-compute the betweenness.

The "m" in the formula represents the edge number of the original graph. The "A" in the formula is the adjacent matrix of the original graph. (Hint: In each remove step, "m" and "A" should not be changed).

If the community only has one user node, we still regard it as a valid community.

You need to save your result in a **txt** file. The format is the same with the output file from task1.

4.2.3 Output Result

In this task, you need to save your result of communities in a **txt** file. Each line represents one community and the format is:

'user_id1', 'user_id2', 'user_id3', 'user_id4', …

Your result should be firstly sorted by the size of communities in the ascending order and then the first user_id in the community in **lexicographical** order (the user_id is type of string). The user_ids in each community should also be in the **lexicographical** order.

If there is only one node in the community, we still regard it as a valid community.

```
'0uEqdNTVj-I7zM83bbcgsg', 'B-KS9atQu-gw51QTee-jpg'
'1e0JSCb2aDQakloIzVqiYg', 'Ner9deZ2TUFxdiRijBp0jA'
'4I1FpxJ-HU1iBKo89WV7IQ', 'Wc5L6iuvSNF5WGBlqI08nw'
'5XAXkk6WENi00W_HSSXRWA', '7RCz4Ln_FaTvNrdwe251Dg'
'9Lb4zyV7HbNPx3pz1wGv_A', 'Hv_Sk1QTTBg0Scage3uiqw'
'BGzavA_ddMr-jGmhArv7fg', 'B01A62kTQk4MfwZPOD9sKg'
'w8UNPUs7bCM2Wqg40ewYFQ', 'whINg-cC-FiAv_ATDGMDTg'
'zsJLk34mTDMyuVS_EaBLxw', 'zsZBYWYEmLLs81_f-HHM8w'
'-qVp8jEndaIU20g58ixeaQ', '0kVTI6YwlwU-1fQ-Fvk5yg', '1GbtKqRpDAfv13fUYIBBmA'
'01bmnye5yXRyDZ5Je6tnKg', '1lad9hCBGpHAS2uMSrKCPw', '3QtgI2sJITkbKaXEGsyThw'
'--R1Sfc-QmcHFGHyX6aVjA', '-2kCxY7_aw5h0z7fJnGMbQ', '-br0hLyLDdkvo4UjMCAo6Q', '2k80VAPxlXHsA5X6EIoQpQ'
'-50XWnmQGqBgEI-9ANvL1g', '12iozmUYN_gvAng9kMLs_w', 'TjXqTqY0XAs8VKo_otj3lw', '1QCY0csLfjdDn6cdVq43UA'
'84DM1UZGdfeHmE95SqqqKg', 'IduEi0zDSIM1AGn0MPfSvw', 'RMopAcH2eoi0uPkStfg_YQ', 'WhBwqZeQ7Xte3Tof9DrdPg', '_5HFgadpCIaSVINVyse2Kw'
'088ICGWrFMiVAzD5vS0cRQ', '0FVcoJko1kfZCrJRfssfIA', '1JEXL5K6VTx01tAs6Jskkg', '26WgdHfEjWj4BrN-cUNhVw', '2dLz_y1_rUHj903_lJfT9w'
```

Figure 2: community output file format

4.3 Execution Format

Execution example:

Python:

spark-submit –class firstname_lastname_task1 firstname_lastname_hw4.jar <filter threshold> <input_file_path> <betweenness_output_file_path> <community_output_file_path>

Input parameters:

1. <filter threshold>: the filter threshold to generate edges between user nodes.
2. <input file path>: the path to the input file including path, file name and extension.
3. <betweenness output file path>: the path to the betweenness output file including path, file name and extension.
4. <community output file path>: the path to the community output file including path, file name and extension.

Execution time:

The overall runtime of your task (from reading the input file to finishing writing the community output file) should be less than 180 seconds.

If your runtime is between 180 seconds and 240 seconds, there will be 20% penalty. If your runtime exceeds 300 seconds, there will be 50% penalty for this task.

5. Grading Criteria

(% penalty = % penalty of possible points you get)

1. You can use your free 5-day extension separately or together.
2. There will be 10% bonus if you use both Scala and Python.
3. If you do not apply the Girvan-Newman algorithm, there will be no point for this task.
4. If we cannot run your programs with the command we specified, there will be 80% penalty.
5. If your program cannot run with the required Scala/Python/Spark versions, there will be 20% penalty.
6. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
7. The total runtime of this assignment should not exceed 10 minutes or there will be no point for this assignment.
8. We can regrade on your assignments within seven days once the scores are released. No argue after one week. There will be 20% penalty if our grading is correct.
9. There will be 20% penalty for late submission within a week and no point after a week.
10. Only when your results from Python are correct, the bonus of using Scala will be calculated. There is no partially point for Scala. See the example below:

Example situations

Task	Score for Python	Score for Scala (10% of previous column if correct)	Total
Task1	Correct: 12.5 points	Correct: 12.5 * 10%	13.75
Task1	Wrong: 0 point	Correct: 0 * 10%	0.0
Task1	Partially correct: 6 points	Correct: 6 * 10%	6.6
Task1	Partially correct: 6 points	Wrong: 0	6.6