

CSCI 5523: Introduction to Data Mining

Spring 2023

Assignment 4

Deadline: Apr. 13th 11:59 PM CT

1. Overview of the Assignment

In this assignment, you will explore the Spark GraphFrames library as well as 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 and Spark to implement all tasks.**

b. **You can use the Spark DataFrame and GraphFrames library for task1, but for task2 you can ONLY use Spark RDD and standard Python libraries.**

2.2 Submission Platform

We will use Gradescope to automatically run and grade your submission. You must test your scripts on **the local machine** before submission.

2.3 Programming Environment

Python 3.9.12 and Spark 3.2.1

2.4 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.

3. Datasets

You will continue to use the Yelp dataset. We have generated a sub-dataset, `ub_sample_data.csv`, from the Yelp review dataset containing `user_id` and `business_id`. You can access and download the data on [Google Drive](#). TAs will use a different test dataset for grading.

4. Tasks

You need to submit the following files on Gradescope: (all in lowercase)

- a. Python scripts: `task1.py`, `task2.py`
- b. [OPTIONAL] You can include other scripts to support your programs (e.g., callable functions).

You don't need to include your results. TAs will grade the assignment using a separate testing dataset (the data format remains the same).

4.1 Graph Construction

To construct a 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.

In this assignment, we use the filter threshold 7.

4.2 Task1: Community Detection Based on GraphFrames (2 pts)

4.2.1 Task description

In task1, you will explore the Spark GraphFrames library to detect communities in the network graph you constructed in 4.1. In the library, it provides the implementation of the Label Propagation Algorithm (LPA) which was proposed by Raghavan, Albert, and Kumara in 2007. It is an iterative community detection solution whereby information “flows” through the graph based on underlying edge structure. In this task, you do not need to implement the algorithm from scratch, you can call the method for LPA provided by the library. The parameter “`maxIter`” for LPA should be set to 5. The following websites may help you get started with the Spark GraphFrames:

<https://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-python.html>

https://graphframes.github.io/graphframes/docs/_site/user-guide.html

4.2.2 Input Format **(Please make sure you use exactly the same input parameters names!)**

We use the “`argparse`” module to parse the following arguments.

Parameters in task1.py

1. Filter threshold(**--filter_threshold**): the filter threshold to generate edges between user nodes.
2. Input file Path(**--input_file**): the path to the input file including path, file name and extension.
3. community output file path (**--community_output_file**): the path to the community output file including path, file name and extension.

Execution example:

```
$ spark-submit --packages graphframes:graphframes:0.8.2-spark3.2-s_2.12 task1.py  
--filter_threshold <filter_threshold> --input_file <input_file> --community_output_file  
<community_output_file>
```

Example: `spark-submit --packages graphframes:graphframes:0.8.2-spark3.2-s_2.12 task1.py
--filter_threshold 7 --input_file data.csv --community_output_file out1`

4.2.3 Output Result

In this task, you need to save your result of communities in a text file (**.txt**). 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.

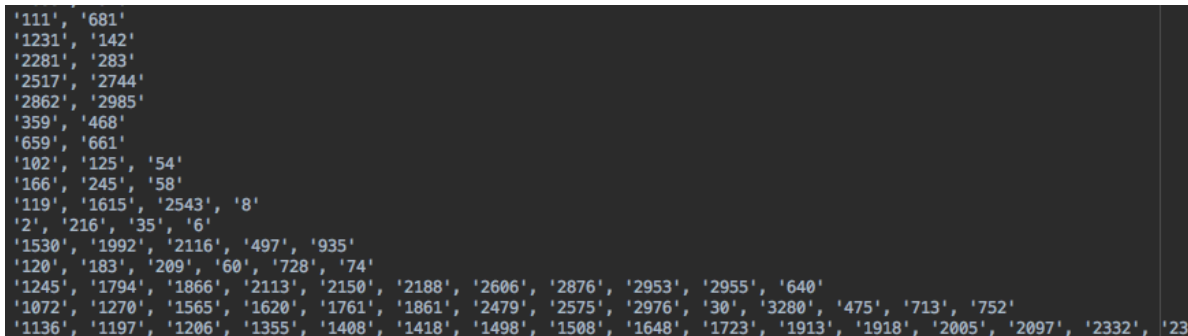


Figure 1: community output file format

4.3 Task2: Community Detection Based on Girvan-Newman algorithm (8 pts)

In task2, you will implement your own Girvan-Newman algorithm to detect the communities in the network graph. Because the task1 and task2 code will be executed separately, you need to construct the graph again in this task following the rules in section 4.1. You can refer to the Chapter 10 from the Mining of Massive Datasets book for the algorithm details.

For task2, you can ONLY use Spark RDD and standard Python libraries. **Remember to delete your code that imports graphframes.**

4.3.1 Betweenness Calculation (4 pts)

In this part, you will calculate the betweenness of each edge in the **original graph** you constructed in 4.1. Then you need to save your result in a **.txt** file. The format of each line is

(‘user_id1’, ‘user_id2’), betweenness value

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 be in lexicographical order. You do not need to round your result.

```
( '12', '74'), 243.36111111111106
( '24', '74'), 237.93253968253967
( '3', '36'), 215.63333333333333
( '12', '50'), 199.18067210567193
( '2113', '640'), 189.00000000000003
( '2188', '640'), 189.00000000000003
( '2606', '640'), 189.00000000000003
```

Figure 2: betweenness output file format

4.3.2 Community Detection (4 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:

$$\text{Q} = \sum_{s \in S} [(\# \text{ edges within group } s) - (\text{expected } \# \text{ edges within group } s)]$$

$$\text{Q}(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

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.

4.3.3 Input Format

We use the “argparse” module to parse the following arguments.

Parameters in task2.py

1. Filter threshold(**--filter_threshold**): the filter threshold to generate edges between user nodes.
2. Input file Path(**--input_file**): the path to the input file including path, file name and extension.
3. Betweenness output file path(**--betweenness_output_file**): the path to the betweenness output file including path, file name and extension.
4. community output file path(**--community_output_file**): the path to the community output file including path, file name and extension.

Execution example:

```
$ python task2.py --filter_threshold <filter_threshold> --input_file <input_file>
--betweenness_output_file <betweenness_output_file> --community_output_file
<community_output_file>
```

Example: `python task2.py --filter_threshold 7 --input_file data.csv --betweenness_output_file out1.txt --community_output_file out2.txt`

4.4 Execution time:

The overall runtime limit of your task1 (from reading the input file to finishing writing the community output file) is **200** seconds.

The overall runtime limit of your task2 (from reading the input file to finishing writing the community output file) is **300** seconds.

If your runtime exceeds the above limit, there will be no point 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. During grading, TAs will count the number of late days **based on the submission time**. For example, if the due date is 2023/09/10 23:59:59 CDT and your submission is 2023/09/11 05:23:54 CDT, the number of late days would be 1. **TAs will record grace days by default (NO separate Emails)**. However, if you want to save it next time and treat your assignment as late submission (with some penalties), **please leave a comment in your submission**.
2. There will be no point if your submission cannot be executed on the grading platform. Please carefully follow the submission instructions in section 6
3. There is no regrading. Once the grade is posted on Canvas, we will only regrade your assignments if there is a grading error. No exceptions.
4. Homework assignments are due at 11:59 pm CT on the due date and should be submitted on Canvas. Late submissions within 24 hours of the due date will receive a 30% penalty. Late submissions after 24 hours of the due date will receive a 70% penalty. Every student has FIVE free

late days for homework assignments. You can use these free late days for any reason, separately or together, to avoid the late penalty. There will be no other extensions for any reason. **You cannot use the free late days after the last day of the class.**

6. Submission Instructions

You will submit your programming assignment via Gradescope. You can access it from Canvas.