INF553 Foundations and Applications of Data Mining

Spring 2020

Assignment 4

Deadline: April 7, 3:00 PM PST

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 Network Data Repository for this assignment. 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 are required to only use Spark RDD in order to understand Spark operations more deeply. You will not get any point if you use Spark DataFrame or DataSet.

2.2 Programming Environment

Python 3.6, Scala 2.11 and Spark 2.4.4

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

We will grade all submissions on Vocareum and the submissions on the blackboard will be ignored.

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

task1.py, task2.py

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

task1.scala, task2.scala

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

3. Datasets

We will use a preprocessed version of the Network Data Repository dataset (http://networkrepository.com) for this homework. The processed dataset is available in the file power_input.txt provided along with this assignment. This data is in the form of an edge list, that is in each row, the pair of numbers indicates that there is an edge between the two nodes. There are 906 edges in the dataset provided.

4. Tasks

4.1 Graph Construction

To construct the social network graph, assume that each node is uniquely labeled, and that links are undirected and unweighted. You will form and analyze these networks using the Spark GraphFrames library and Girvan Newman algorithm.

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

In task1, you will use the Spark GraphFrames library to detect communities in the network graph discussed in 4.1. The library 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 provided by the library. 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://docs.databricks.com/spark/latest/graph-analysis/graphframes/user-guide-scala.html

4.2.1 Execution Detail

The version of the GraphFrames should be **0.6.0**.
Install using: pip install graphframes For Python:

- In PyCharm, you need to add the sentence below into your code os.environ["PYSPARK_SUBMIT_ARGS"] = ("--packages graphframes:graphframes:0.6.0-spark2.3-s_2.11")
- In the terminal, you need to assign the parameter "packages" of the spark-submit: --packages graphframes:graphframes:0.6.0-spark2.3-s_2.11

For Scala:

- In Intellij IDEA, you need to add library dependencies to your project “graphframes” % “graphframes” % “0.6.0-spark2.3-s_2.11” “org.apache.spark” %% “spark-graphx” % sparkVersion
- In the terminal, you need to assign the parameter “packages” of the spark-submit: --packages graphframes:graphframes:0.6.0-spark2.3-s_2.11

For the parameter “maxIter” of LPA method, you should set it to 5.

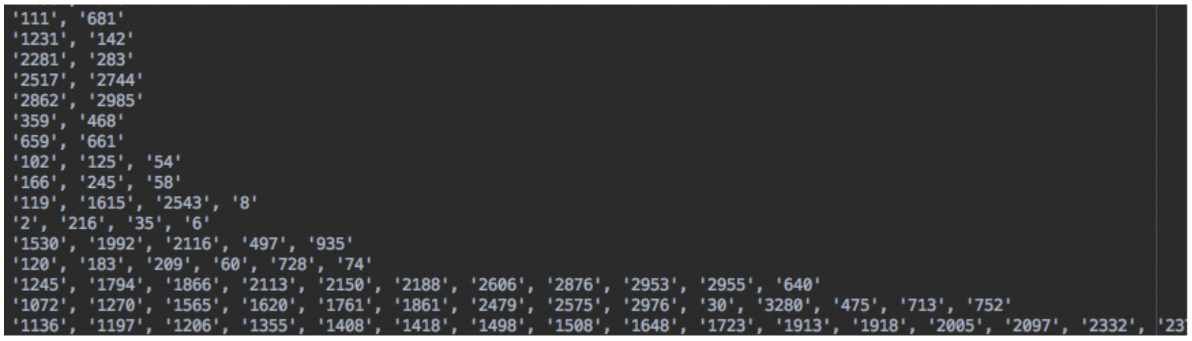
4.2.2 Output Result

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

‘node_id1’, ‘node_id2’, ‘node_id3’, ‘node_id4’, ...

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

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



```
'111', '681'
'1231', '142'
'2281', '283'
'2517', '2744'
'2862', '2985'
'359', '468'
'659', '661'
'102', '125', '54'
'166', '245', '58'
'119', '1615', '2543', '8'
'2', '216', '35', '6'
'1530', '1992', '2116', '497', '935'
'120', '183', '209', '60', '728', '74'
'1245', '1794', '1866', '2113', '2150', '2188', '2606', '2876', '2953', '2955', '640'
'1072', '1270', '1565', '1620', '1761', '1861', '2479', '2575', '2976', '30', '3280', '475', '713', '752'
'1136', '1197', '1206', '1355', '1408', '1418', '1498', '1508', '1648', '1723', '1913', '1918', '2005', '2097', '2332', '23
```

Figure 1: community output file format

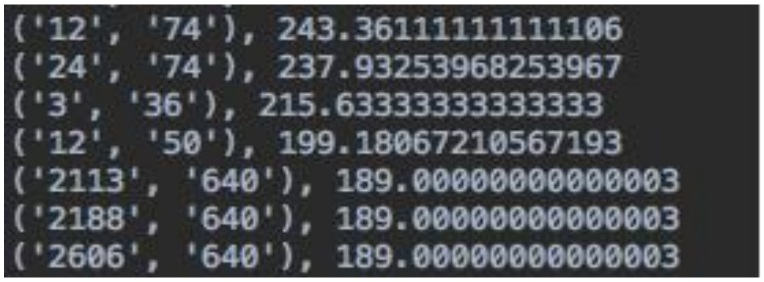
4.3 Task2: Community Detection Based on Girvan-Newman algorithm (5 pts)

In task2, you will implement your own Girvan-Newman algorithm to detect the communities in the network graph discussed in 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 or Scala libraries.

4.3.1 Betweenness Calculation (2 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 **(‘node_id1’, ‘node_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 of type string). The two user_ids in each tuple should also be in **lexicographical** order. You do not need to round your result.

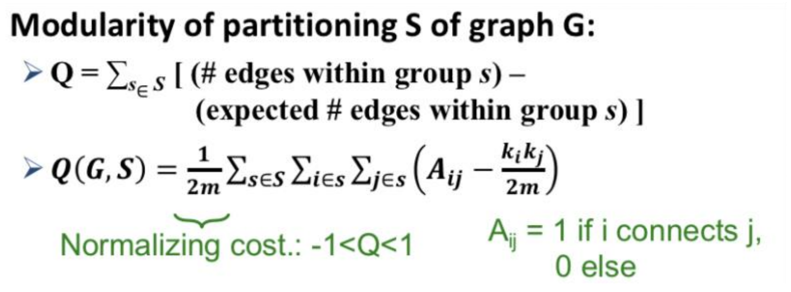


```
('12', '74'), 243.36111111111106
('24', '74'), 237.93253968253967
('3', '36'), 215.63333333333333
('12', '50'), 199.18067210567193
('2113', '640'), 189.00000000000003
('2188', '640'), 189.00000000000003
('2606', '640'), 189.00000000000003
```

Figure 2: betweenness output file format

4.3.2 Community Detection (3 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:

- $Q = \sum_{s \in S}$ [(# edges within group s) – (expected # edges within group s)]
- $Q(G,S) = \underbrace{\frac{1}{2m}}_{\text{Normalizing cost.: } -1<Q<1} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m}\right)$

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 as the output file from task1.

5. Execution Format

Execution example:

spark-submit --packages graphframes:graphframes:0.6.0-spark2.3-s_2.11 task1.py <input_file_path> <community_output_file_path>

spark-submit task2.py <input_file_path> <betweenness_output_file_path> <community_output_file_path>

Input parameters:

1. <input file path>: the path to the input file including path, file name and extension.

2. <betweenness output file path>: the path to the betweenness output file including path, file name and extension.

3. <community output file path>: the path to the community output file including path, file name and extension.

Execution time:

The suggested overall runtime of your task1 (from reading the input file to finishing writing the community output file) is **500** seconds.

The overall runtime of your task2 (from reading the input file to finishing writing the community output file) should be less than **500** seconds.

If your runtime is between 500 seconds and 700 seconds, there will be 50% penalty.
If your runtime exceeds 700 seconds, there will be no point for this task.

6. 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. We 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. If a plagiarism is detected, there will be no point for the entire assignment and we will report all detected plagiarism.
4. All submissions will be graded on the Vocareum. Please strictly follow the format provided, otherwise you can't get the point even though the answer is correct.
5. If the outputs of your program are unsorted or partially sorted, there will be 50% penalty.
6. If you use Spark DataFrame, DataSet, sparksql, there will be 20% penalty.
7. There will be 20% penalty for late submission within a week and no point after a week.
8. 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:

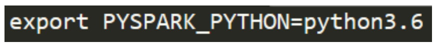
Example situations

Task	Score for Python	Score for Scala (10% of previous column if correct)	Total
Task1	Correct: 2 points	Correct: 2 * 10%	2.2
Task1	Wrong: 0 point	Correct: 0 * 10%	0.0
Task1	Partially correct: 1 points	Correct: 1 * 10%	1.1
Task1	Partially correct: 1 points	Wrong: 0	1

7. Common problems causing fail submission on Vocareum/FAQ

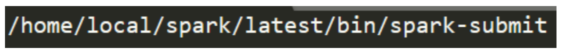
(If your program runs seems successfully on your local machine but fail on Vocareum, please check these)

1. Try your program on Vocareum terminal. Remember to set python version as python3.6,



```
export PYSPARK_PYTHON=python3.6
```

And use the latest Spark



```
/home/local/spark/latest/bin/spark-submit
```

2. Check the input commend line format.

3. Check the output format, for example, the header, tag, typo.

4. Check the requirements of sorting the results.

5. Your program scripts should be named as task1.py task2.py.

6. Check whether your local environment fit the assignment description, i.e. version, configuration.

7. If you implement the core part in python instead of spark, or implement it in a high time complexity way (e.g. search an element in a list instead of a set), your program may be killed on the Vocareum because it runs too slow.

8. You are required to only use Spark RDD in order to understand Spark operations more deeply. You will not get any point if you use Spark DataFrame or DataSet. Don’t import sparksql.