INF553 Foundations and Applications of Data Mining

Spring 2021

Assignment 4

Deadline: April 6, 11:59 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 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.** 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 use the Spark DataFrame and GraphFrames library for task1, but for task2 you can ONLY use Spark RDD and standard Python or Scala libraries. (ps. For Scala, you can try GraphX, but for the assignment, you need to use GraphFrames.)**

2.2 Programming Environment

Python 3.6, Scala 2.11, and Spark 2.3

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

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

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

a. [REQUIRED] two Python scripts, named: **task1.py, task2.py**

b1. [REQUIRED FOR SCALA] two Scala scripts, named: **task1.scala, task2.scala**

b2. [REQUIRED FOR SCALA] one jar package, named: **hw4.jar**

c. [OPTIONAL] You can include other scripts called by your main program

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

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 download it from Vocareum.

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

In this assignment, we use a filter threshold of 7.

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

In task1, you will explore the Spark GraphFrames library to detect communities in the network graph that 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 the underlying edge structure. For the details of the algorithm, you can refer to the paper in the link below. 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

http://www.leonidzhukov.net/hse/2017/networks/papers/raghavan2007.pdf

4.2.1 Execution Detail

The version of the GraphFrames should be **0.6.0**.

For Python:

- In PyCharm, you need to add the sentence below into your code

  pip install graphframes

  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:

'user_id1', 'user_id2', 'user_id3', 'user_id4', ...

Your result should be firstly sorted by the size of communities in ascending order and then the first user id in the community in the **lexicographical** order (the data type of user_id is a 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.

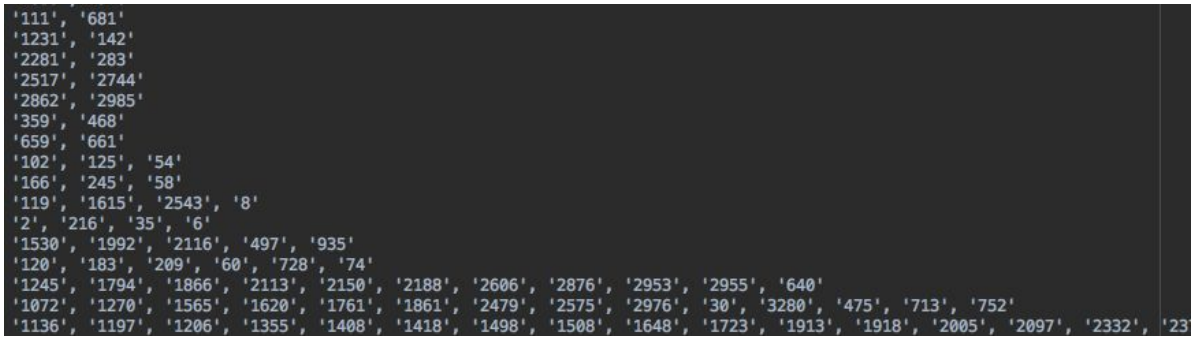

Figure 1: community output file format

4.3 Task2: Community Detection Based on Girvan-Newman algorithm (6 pts)

In task2, you will implement the Girvan-Newman algorithm to detect the communities in the network graph. Because your 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 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. **Remember to delete your code that imports graphframes.**

4.3.1 Betweenness Calculation (3 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 the **lexicographical** order (the user_id is the type of string). The two user_ids in each tuple should also be in the **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 reach 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 as the output file from task1.

4.4 Execution Format

Execution example:

Python:

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

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

Scala:

spark-submit --packages graphframes:graphframes:0.6.0-spark2.3-s_2.11 --class task1 hw4.jar <filter threshold> <input_file_path> <community_output_file_path>

spark-submit --class task2 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 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 250 seconds.

If your runtime exceeds the above limit, there will be no point for this task.

5. About Vocareum

a. You can use the provided datasets under the directory resource: /asnlib/publicdata/
b. You should upload the required files under your workspace: work/
c. You must test your scripts on both the local machine and the Vocareum terminal before submission.
d. During the submission period, the Vocareum will automatically test task1 and task2.
e. During the grading period, the Vocareum will use another dataset that has the same format for testing.
f. We do not test the Scala implementation during the submission period.
g. Vocareum will automatically run both Python and Scala implementations during the grading period.

h. Please start your assignment early! You can resubmit any script on Vocareum. We will only grade on your last submission.

6. Grading Criteria

a. You can use your free **5**-day extension separately or together. You must submit a late-day request via https://forms.gle/6aDASyXAuBeV3LkWA. This form is recording the number of late days you use for each assignment. By default, we will not count the late days if no request submitted.

b. There will be 10% bonus for each task if your Scala implementations are correct. Only when your Python results are correct, the bonus of Scala will be calculated. There is no partial point for Scala.

c. There will be no point if your submission cannot be executed on Vocareum.

d. There is no regrading. Once the grade is posted on the Blackboard, we will only regrade your assignments if there is a grading error. No exceptions.

e. There will be 20% penalty for the late submission within one week and no point after that.