INF 553 – Spring 2018

Assignment 4 Community Detection

Deadline: 04/09 2018 11:59 PM PST

Assignment Overview

In this assignment you are asked to implement the Girvan-Newman algorithm using the Spark Framework in order to detect communities in the graph. You will use only *video_small_num.csv* dataset in order to find users who have the similar product 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.

Environment Requirements

Python: 2.7 Scala: 2.11 Spark: 2.2.1

IMPORTANT: We will use these versions to compile and test your code. If you use other versions, there will be a 20% penalty since we will not be able to grade it automatically.

You can only use Spark RDD.

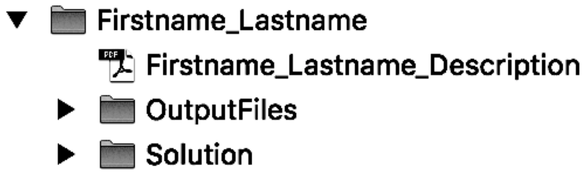
Write your own code!

For this assignment to be an effective learning experience, you must write your own code! I emphasize this point because you will be able to find Python implementations of most or perhaps even all of the required functions on the web. Please do not look for or at any such code! **Do not share code with other students in the class!!**

Submission Details

For this assignment you will need to turn in a Python, Java, or Scala program depending on your language of preference.

Your submission must be a .zip file with name: **<Firstname>_<Lastname>_hw4.zip**. The structure of your submission should be identical as shown below. The Firstname_Lastname_Description.pdf file contains helpful instructions on how to run your code along with other necessary information as described in the following sections. The *OutputFiles* directory contains the deliverable output files for each problem and the *Solution* directory contains your source code.



Datasets

We are continually using Amazon Review data. This time we use a subset of Amazon Instant Video category. We have already transferred the string id of user and product to integers for your convenience. You should download one file from Blackboard:

1. video_small_num.csv

Construct Graph

Each node represents a user. Each edge is generated in following way:
In *video_small_num.csv*, count the number of times that two users rated the same product. If the number of times is greater or equivalent to **7** times, there is an edge between two users.

Task1: Betweenness (50%)

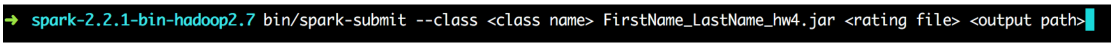
You are required to implement **Girvan-Newman Algorithm** to find betweenness of each edge in the graph. The betweenness function should be calculated only once from the original graph.

Execution Example

The first argument passed to your program (in the below execution) is the path of *video_small_num.csv* file (e.g. *“spark-2.2.1-bin-hadoop2.7/HW4/video_small_num.csv*”). The second input is the output path *(output path is the directory of your output file, not including file name. e.g. “spark-2.2.1-bin-hadoop2.7/HW4/”).* Following we present examples of how you can run your program with spark-submit both when your application is a Java/Scala program or a Python script.

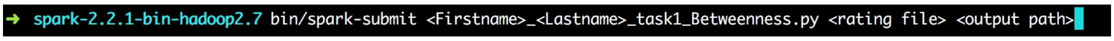
A. **Example of running a Java/Scala application with spark-submit:**
Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

You should use **Betweenness** as your class name for this task.



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit --class <class name> FirstName_LastName_hw4.jar <rating file> <output path>
```

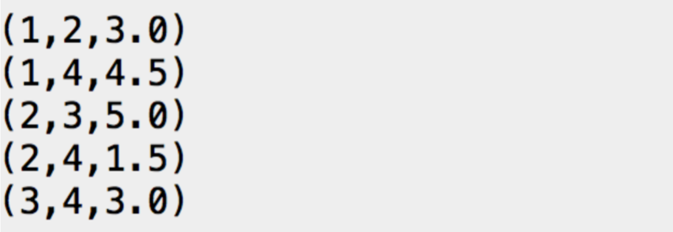
B. **Example of running a Python application with spark-submit:**



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit <Firstname>_<Lastname>_task1_Betweenness.py <rating file> <output path>
```

Result format:

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: (the example just shows the format, is NOT a solution)



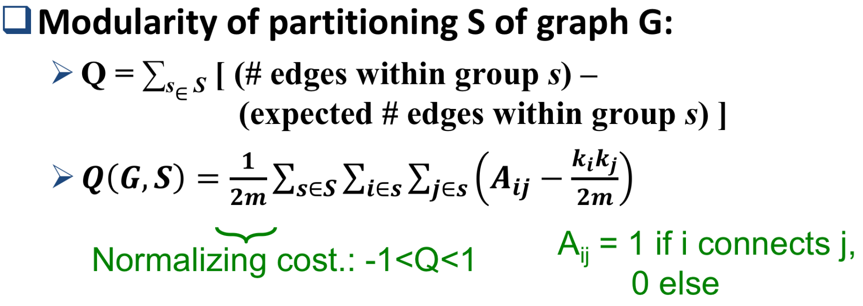
```
(1,2,3.0)
(1,4,4.5)
(2,3,5.0)
(2,4,1.5)
(3,4,3.0)
```

Runtime Requirement:

< 60 sec

Task2: Detect Community (50%)

You are required to implement betweenness and modularity in this task. You also need to divide the graph into suitable communities, which reaches the highest modularity.
When you use the following formula to calculate modularity of partition S of G, you should be aware that A_{ij} should remain the same as original graph (i.e. A_{ij} does not change while you delete any edge)



❑ **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

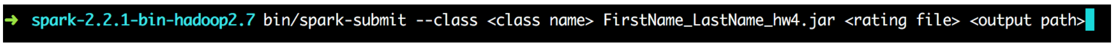
Execution Example

The first argument passed to your program (in the below execution) is the path of *video_small_num.csv* file (e.g. *"spark-2.2.1-bin-hadoop2.7/HW4/video_small_num.csv"*). The second input is the output path *(output path is the directory of your output file, not including file name. e.g. "spark-2.2.1-bin-hadoop2.7/HW4/")*. Following we present examples of how you can run your program with spark-submit both when your application is a Java/Scala program or a Python script.

A. Example of running a Java/Scala application with spark-submit:

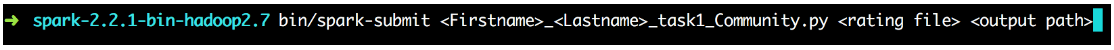
Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

You should use **Community** as your class name for the task.



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit --class <class name> FirstName_LastName_hw4.jar <rating file> <output path>
```

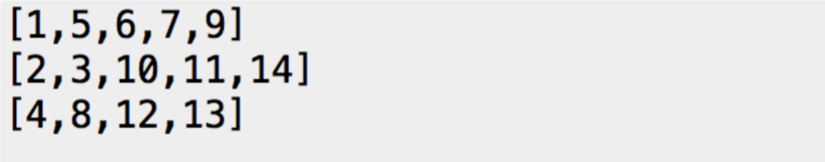
B. Example of running a Python application with spark-submit:



```
➜ spark-2.2.1-bin-hadoop2.7 bin/spark-submit <Firstname>_<Lastname>_task1_Community.py <rating file> <output path>
```

Result format:

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. And example is as follows: (the example just shows the format, is NOT a solution)



```
[1,5,6,7,9]
[2,3,10,11,14]
[4,8,12,13]
```

Runtime Requirement:

< 60 sec

Description File

Please include the following content in your description file:

1. Mention the Spark version and Python version
2. Describe how to run your program for both tasks

Submission Details

Your submission must be a .zip file with name: *<Firstname>_<Lastname>_hw4.zip*

Please include all the files in the right directory as following:

1. A description file: *<Firstname>_<Lastname>_desription.pdf*
2. All Scala scripts:
   <Firstname>_<Lastname>_task1_Betweenness.scala
   <Firstname>_<Lastname>_task1_Community.scala
3. A jar package for all Scala file: *<Firstname>_<Lastname>_hw4.jar*
   If you use Scala for all tasks, please make all *.scala file into ONLY ONE *<Firstname>_<Lastname>_hw4.jar* file and strictly follow the class name mentioned above. And DO NOT include any data or unrelated libraries into your jar.
4. If you use Python, then all python scripts:
   <Firstname>_<Lastname>_task1_Betweenness.py
   <Firstname>_<Lastname>_task2_Community.py
5. Required result files for task1 & 2:
   <Firstname>_<Lastname>_ Betweenness.txt
   <Firstname>_<Lastname>_ Community.txt

Grading Criteria:

1. If your programs cannot run with the commands you provide, your submission will be graded based on the result files you submit, and there will be an 80% penalty
2. If the files generated are not sorted based on the specifications, there will be 20% penalty.
3. If your program generates more than one file, there will be 20% penalty.
4. if runtime of your program exceeds the runtime requirement, there will be 20% penalty.
5. **If you don't provide the source code, especially the Scala scripts, there will be 20% penalty.**
6. You can use your free 5-day extension.

7. There will be 10% bonus if you use Scala for the entire assignment.