Inf553 – Foundations and Applications of Data Mining

Summer 2018 Assignment 4
Community Detection

Deadline: 07/09 2018 11:59 PM PST

1 Overview of the Assignment

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 ratings.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.

1.1 Environment Requirements

Python: 2.7 Scala: 2.11 Spark: 2.2.1

Student must use python to complete both Task1 and Task2.

There will be 10% bonus if you also use Scala for both Task1 and Task2 (i.e. 10 - 11; 9 - 10).

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.

In order for you to understand more deeply of the Spark, use RDD only, you won't get any point if you use Dataframe or Dataset.

1.2 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!

TA will combine some python code on Github which can be searched by keyword "INF553" and every students' code, using some software tool for detecting Plagiarism.

Do not share code with other students in the class!!

Submission Details

For this assignment you will need to turn in a Python or Scala program depending on your language of preference. We will test your code using the same datasets but with different support thresholds values.

This assignment will surely need some time to be implemented so please plan accordingly and start early!

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.

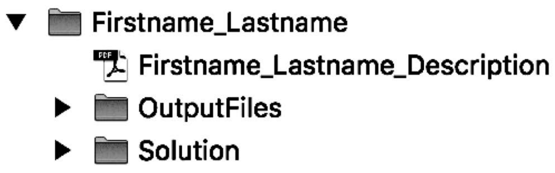


Figure 1: Submission Structure

Details Of This Assignment

Dataset

We are continually using MovieLens dataset. This time we use the **ratings.csv** of the small dataset. You can download one file from Blackboard.

ratings.csv

Construct Graph

Each node represents a user. Each edge is generated in following way:

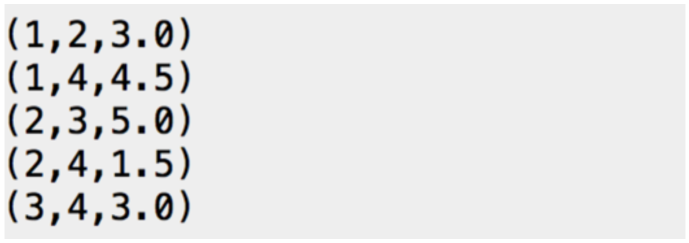
In ratings.csv, count the number of times that two users rated the same movie. If the number of times is greater or equivalent to **9** times, there is an edge between two users.

2 Task 1: Betweenness (50 Points)

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.

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)
```

Figure 2: Result Format of the Betweenness

Runtime Requirement

Less than 100 seconds

Execution Requirements

The program that you will implement should take only one parameters. The parameter is the location of the ratings.csv file. The name of the output file must be Firstname_Lastname_Betweenness.txt. The content of the file must follow the Result Format section above. If your program does not generate this file or it does not follow the specifications as described in the following section, there would be a penalty of 20%.

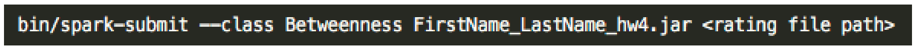
Execution Example

The parameter passed to our program is the the location of the ratings.csv file. 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.

Example of running application with spark-submit

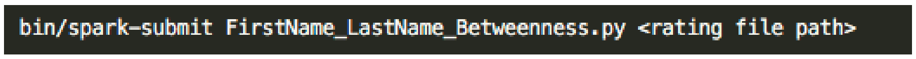
Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

Please use Betweenness as class name



```
bin/spark-submit —-class Betweenness FirstName_LastName_hw4.jar <rating file path>
```

Figure 3: Betweenness: Command Line Format for Scala



```
bin/spark-submit FirstName_LastName_Betweenness.py <rating file path>
```

Figure 4: Betweenness: Command Line Format for python

You don't need to specify the path of the output file in the commandline, you only need to save the file with the format Firstname_Lastname_Betweenness.txt. in the same path your program run.

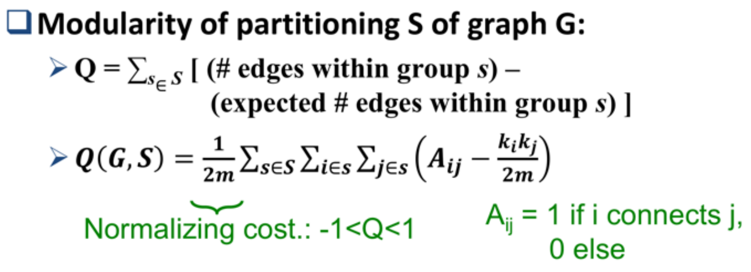
3 Task2: Detect Community (50 Points + 10 Points Bonus)

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} and k_i k_j should change while you delete edges in the graph.

You only need to calculate the modularity when a new Community appear. (When you remove edges in the graph, if no Community be divided, you don't need to calculate the modularity).

The modularity of the graph will first increase and then decrease, when you found that the modularity decreasing, the highest value before is the highest modularity point.



❑ **Modularity of partitioning S of graph G:**

➢ $\mathbf{Q} = \sum_{s \in S}$ **[(# edges within group *s*) – (expected # edges within group *s*)]**

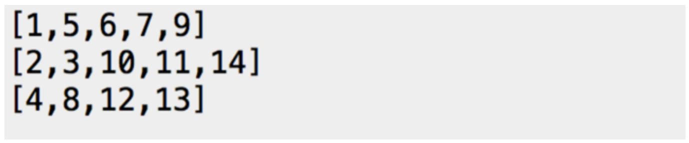
➢ $$Q(G,S) = \underbrace{\frac{1}{2m}}_{\text{Normalizing cost.: } -1<Q<1} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$$

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

Figure 5: Community: Formula of Modularity

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]
```

Figure 6: Result Format of the Betweenness

Runtime Requirement

Less than 500 seconds

Execution Requirements

The program that you will implement should take only one parameters. The parameter is the location of the ratings.csv file. The name of the output file must be Firstname_Lastname_Community.txt. The content of the file must follow the Result Format section above. If your program does not generate this file or it does not follow the specifications as described in the following section, there would be a penalty of 20%.

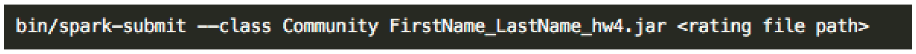
Execution Example

The parameter passed to our program is the the location of the ratings.csv file. 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.

Example of running application with spark-submit

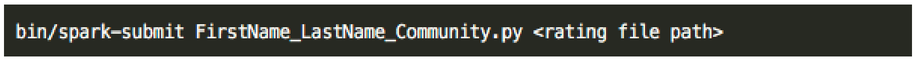
Notice that the argument class of the spark-submit specifies the main class of your application and it is followed by the jar file of the application.

Please use Community as class name



```
bin/spark-submit --class Community FirstName_LastName_hw4.jar <rating file path>
```

Figure 7: Community: Command Line Format for Scala



```
bin/spark-submit FirstName_LastName_Community.py <rating file path>
```

Figure 8: Community: Command Line Format for python

You don't need to specify the path of the output file in the commandline, you only need to save the file with the format Firstname_Lastname_Community.txt. in the same path your program run.

Bonus (10 Points)

There are some libraries containing the community detection function.
You can detect communities by using community detection function in SparklingGraph. Here is the link to learn more about it:
SparklingGraph
The graph is constructed in the same way as mentioned above.
The parameter and result are same as mentioned above.

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. If you use Python, then all python scripts:
Firstname_Lastname_Betweenness.py
Firstname_Lastname_Community.py
3. All Scala scripts:
Firstname_Lastname_Betweenness.scala
Firstname_Lastname_Community.scala
4. A jar package for all Scala file:
Firstname_Lastname_hw4.jar
If you use Scala, 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.
5. Required result files for task1 & 2:
Firstname_Lastname_Betweenness.txt
Firstname_Lastname_Community.txt

Grading Criteria

1. If your programs cannot be run with the commands you provide, your submission will be graded based on the result files you submit and 80% penalty for it.
2. If the files generated by your programs 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 you don't provide the source code and just the .jar file in case of a Java/Scala application there will be 100% penalty.
4. if runtime of your program exceeds the runtime requirement, there will be 20% penalty.
5. There will be 20% penalty for late submission within a week and 0 grade after a week.
6. You can use your free 5-day extension.
7. There will be 10% bonus if you use both Scala and python for the entire assignment.