

# INF553 Foundations and Applications of Data Mining

Fall 2019

## Assignment 4

**Deadline: November. 13<sup>th</sup> 3:00 PM PST**

Please add the first line in the code as comments to show your name and USC ID.

### 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.

## 2. Assignment Requirements

### 2.1 Programming Language and Library Requirements

- a. **You must use Python to implement all tasks. You can only use standard Python libraries (i.e., external libraries like Numpy or Pandas are not allowed).** There will be a **10% bonus** for each task (or case) if you also submit a Scala implementation and both your Python and Scala implementations are correct.
- b. **You are required to only use the Spark RDD** to understand Spark operations. You will not receive any point if you use Spark DataFrame or DataSet.

### 2.2 Programming Environment

**Python 3.7, Scala 2.11, JDK 1.8 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 your code with other students!!**

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. We will report all the detected plagiarism.

## Submission

You need to submit following files on **Vocareum** with exactly the same name:

a. Two Python scripts:

hw4\_task1.py

hw4\_task2.py

b. [OPTIONAL] Two Scala scripts:

hw4\_task1.scala

hw4\_task2.scala

hw4.jar

## Datasets

This time we use a subset of Amazon Instant Video category. 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 name of the input file [video\\_small\\_num.csv](#), including path of file. The second argument passed to your program (in the below execution) is the name of the output file, including path of file. Following we present examples of how you can run your program with spark-submit when your application is a Python script.

### Input format: (we will use the following command to execute your code)

```
./spark-submit hw4_task1.py <input_file_name> <output_file_name>
```

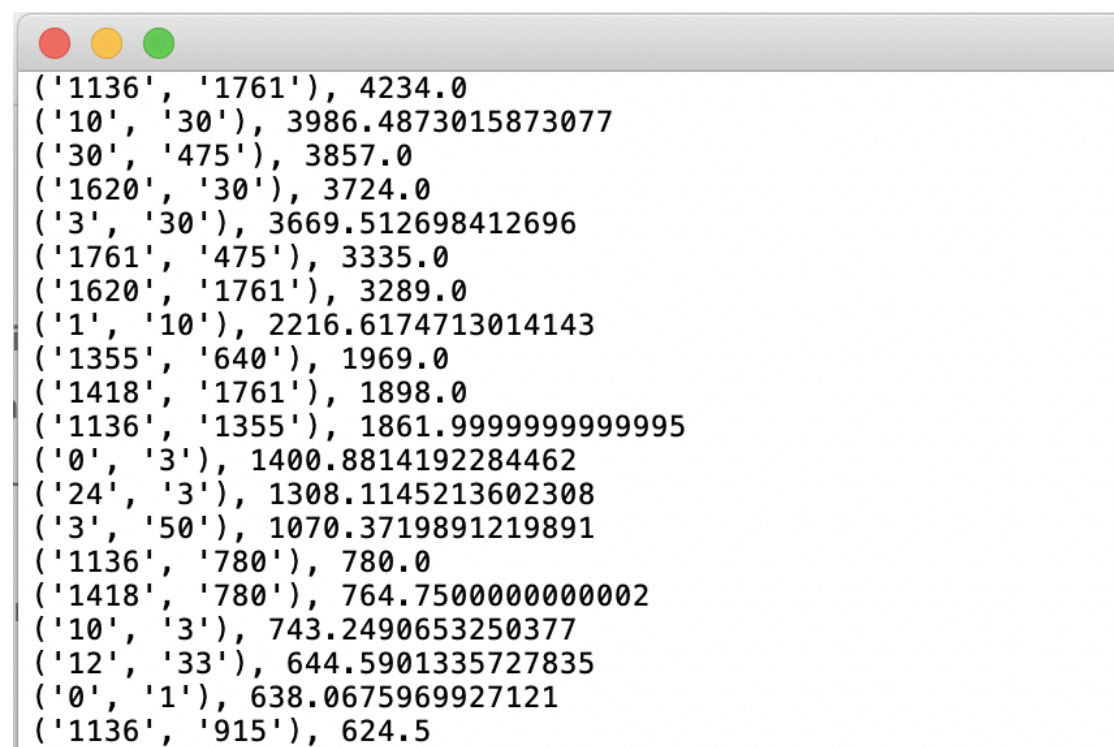
Param: input\_file\_name: the name of the input file (e.g., [video\\_small\\_num.csv](#)), including the file path.

Param: output\_file\_name: the name of the output txt file, including the file path.

**Result format:** In this part, you will calculate the betweenness of each edge in the original graph.

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



```
('1136', '1761'), 4234.0
('10', '30'), 3986.4873015873077
('30', '475'), 3857.0
('1620', '30'), 3724.0
('3', '30'), 3669.512698412696
('1761', '475'), 3335.0
('1620', '1761'), 3289.0
('1', '10'), 2216.6174713014143
('1355', '640'), 1969.0
('1418', '1761'), 1898.0
('1136', '1355'), 1861.9999999999995
('0', '3'), 1400.8814192284462
('24', '3'), 1308.1145213602308
('3', '50'), 1070.3719891219891
('1136', '780'), 780.0
('1418', '780'), 764.7500000000002
('10', '3'), 743.2490653250377
('12', '33'), 644.5901335727835
('0', '1'), 638.0675969927121
('1136', '915'), 624.5
-----
```

**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:

$$\begin{aligned} &\triangleright Q = \sum_{s \in S} [ (\# \text{ edges within group } s) - (\text{expected } \# \text{ edges within group } s) ] \\ &\triangleright Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left( A_{ij} - \frac{k_i k_j}{2m} \right) \end{aligned}$$

Normalizing cost.:  $-1 < Q < 1$        $A_{ij} = 1$  if  $i$  connects  $j$ ,  
0 else

### Execution Example

The first argument passed to your program (in the below execution) is the name of the input file *video\_small\_num.csv*, including path of file. The second argument passed to your program (in the below execution) is the name of the output file, including path of file. Following we present examples of how you can run your program with spark-submit when your application is a Python script.

### Input format: (we will use the following command to execute your code)

`./spark-submit hw4_task2.py <input_file_name> <output_file_name>`

Param: input\_file\_name: the name of the input file (e.g., *video\_small\_num.csv*), including the file path.

Param: output\_file\_name: the name of the output txt file, including the file path.

### 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 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.**

```
'1033', '576'
'111', '681'
'1231', '142'
'2281', '283'
'2517', '2744'
'2862', '2985'
'359', '468'
'659', '661'
.....
```

### Runtime Requirement:

< 60 sec

### **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.