DSCI-553 Foundations and Applications of Data Mining

Fall 2025

Assignment 4

Deadline: November 13 - 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 similar business tastes. 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. **For Task 1, you can use the Spark DataFrame and GraphFrames library. For task 2 you can ONLY use Spark RDD and standard Python or Scala libraries.** There will be a **10% bonus** for each task if you also submit a Scala implementation and both your Python and Scala implementations are correct.

2.2 Programming Environment

Python 3.6, JDK 1.8, Scala 2.12, and Spark 3.1.2

We will use these library versions to compile and test your code. There will be no point if we cannot run your code on Vocareum.

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:

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

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

b2. [OPTIONAL, 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

We have generated a sub-dataset, ub_sample_data.csv, from the Yelp review dataset containing user_id and business_id. You can find the data on Vocareum under resource/asnlib/publicdata/.

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**.

Each node represents a user. There should be an edge between two nodes if the number of common businesses reviewed by two users is **greater than or equivalent to** the filter threshold. For example, suppose user1 reviewed *set*{business1, business2, business3} and user2 reviewed *set*{business2, business3, business4, business5}. If the threshold is 2, there will be an edge between user1 and user2.

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

The filter threshold will be given as an input parameter when running your code.

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 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 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**.

(For your convenience, graphframes0.6.0 is already installed for python on Vocareum. The corresponding jar package can also be found under the *$ASNLIB/public* folder.)

For Python (in local machine):

- [Approach 1] Run "python3.6 -m pip install graphframes" in the terminal to install the package.

- [Approach 2] In PyCharm, you add the sentence below into your code to use the jar package

  os.environ["PYSPARK_SUBMIT_ARGS"] = "--packages
  graphframes:graphframes:0.8.2-spark3.1-s_2.12 pyspark-shell"

- In the terminal, you need to assign the parameter "packages" of the spark-submit:

  --packages graphframes:graphframes:0.8.2-spark3.1-s_2.12

For Scala (in local machine):

- In Intellij IDEA, you need to add library dependencies to your project

  "graphframes" % "graphframes" % "0.8.2-spark3.1-s_2.12"

  "org.apache.spark" %% "spark-graphx" % sparkVersion

- In the terminal, you need to assign the parameter "packages" of the spark-submit:

  --packages graphframes:graphframes:0.8.2-spark3.1-s_2.12

For the parameter "maxIter" of the 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 **lexicographical** order (the user_id is of type 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.

```
'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 Task 2: Community Detection Based on Girvan-Newman algorithm (5 pts)

In task 2, you will implement your own Girvan-Newman algorithm to detect the communities in the network graph. You can refer to Chapter 10 from the Mining of Massive Datasets book for the algorithm details.

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.

For task 2, you can ONLY use **Spark RDD** and standard Python or Scala libraries. **Remember to delete your code that imports graphframes.** Usage of **Spark DataFrame** is **NOT** allowed in this task.

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

<div align="center">

('user_id1', 'user_id2'), betweenness value

</div>

Your result should be firstly sorted by the betweenness values in 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 be in lexicographical order.

For output, you should use the python built-in <u>round()</u> function to round the betweenness value to five digits after the decimal point. (**Rounding is for output only, please do not use the rounded numbers for further calculation**)

IMPORTANT: Please strictly follow the output format since your code will be graded automatically. We will not regrade because of formatting issues.

```
('FaAb-CoKW4xDvLTSPtEbgw', 'W4ZuWztx47aiksyYPUMYDw'),1077.7
('7HmXaZ1C8--Mt4MyItJiqg', 'W4ZuWztx47aiksyYPUMYDw'),746.06905
('SqjP1cV8JwDdN0K9QULzog', 'fgkJ6KUlrmbKyLDQervxsw'),715.87146
('HVa84WqQT5Klz4knWtGBqw', 'pxzs-Dy2hXTis-PuNCV37Q'),704.94308
('AkIZlaQe7GUXttx_epYcfg', 'm-BZLIIh5PCAKnzH0qj_0Q'),697.48723
('2pVj1Hid6_Iq3Th9xI0aOQ', 'h9y55WNNg7SYg3kQwzTMmQ'),689.01026
('eh9uc8xHYR9-rki7fWggQQ', 'fgkJ6KUlrmbKyLDQervxsw'),648.23209
('AkIZlaQe7GUXttx_epYcfg', 'hGeXKYWkqJgFC1u_Sh6kCg'),639.0
('PDAw26zhNQU-HhA0acpCWQ', 'XbiKsujS_qxU3xsr0xUqmQ'),639.0
('2pVj1Hid6_Iq3Th9xI0aOQ', 'iIe1gtgyLqV0ZBbArDQkvg'),598.21349
('jWNooyYAuHWUpxnZGTRpeA', 'm-BZLIIh5PCAKnzH0qj_0Q'),585.75
```

<div align="center">

Figure 2: betweenness output file format

</div>

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) $]$

➤ $Q(G, S) = \frac{1}{2m} \sum_{s \in S} \sum_{i \in s} \sum_{j \in s} \left(A_{ij} - \frac{k_i k_j}{2m} \right)$

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**. (Hint: In each remove step, "m", "k_i" and "k_j" should not be changed, while 'A' is calculated based on the updated graph.). In the step of removing the edges with the highest betweenness, if two or more edges have the same (highest) betweenness, you should remove all those edges.

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 task 1.

Hints:

1. For task 2.2, you should take into account the precision. For example: stop the modularity calculation only if there is a significant reduction in the new modularity.
2. A=1 when BOTH i in j and j in i. Not just i in j or j in i.
3. For task 2.2 the stopping criteria plays an important role. Again, avoid the temptation to stop your search at the first decrease in modularity. Instead, continue exploring all potential partitions to find the global maximum. This comprehensive approach ensures that you don't miss the optimal solution.
4. If you want to do a thorough checking of the answer, you can always calculate the modularity for all possible communities (stop until no edges remain).
5. In modularity calculation, For A, using current graph; for kikj, using original graph.

IMPORTANT: Please strictly follow the hints as your code will be graded on a different dataset. Passing the submission dataset does not guarantee passing the grading dataset unless you strictly follow all the hints above and you may lose points because of that. We will not regrade for any points lost due to this. PLEASE DO FOLLOW ALL THE HINTS ABOVE.

4.4 Execution Format

Execution example:

Python:

spark-submit --packages graphframes:graphframes:0.8.2-spark3.1-s_2.12 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:0.8.2-spark3.1-s_2.12 –-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 **400** seconds.

The overall runtime limit of your task 2 (from reading the input file to finishing writing the community output file) is **400** seconds.

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

5. About Vocareum

a. Dataset is under the directory $ASNLIB/publicdata/, jar package is under $ASNLIB/public/

b. You should upload the required files under your workspace: work/, and click submit

c. You should 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

(% penalty = % penalty of possible points you get)

1. You can use your **free 5-day extension** separately or together

a.
   b. This form will record the number of late days you use for each assignment. We will not count late days if no request is submitted. Remember to submit the request **BEFORE the deadline**.

2. There will be a 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.
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 a **50% penalty**.
6. We can regrade your assignments within seven days once the scores are released. **No argument after one week**.
7. There will be a **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 partial point for Scala.

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
```

Use the latest Spark

/opt/spark/spark-3.1.2-bin-hadoop3.2/bin/spark-submit

Select JDK 8 by running the command

"**export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-amd64**"

2. Check the input command line formats.

3. Check the output formats, for example, the headers, tags, typos.

4. Check the requirements of sorting the results.

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

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

7. If you implement the core part in python instead of spark, or implement it with a high time complexity (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 points if you use Spark DataFrame or DataSet. Don't import sparksql.

9. Do not use Vocareum for debugging purposes, please debug on your local machine. Vocareum can be very slow if you use it for debugging.

10. Vocareum is reliable in helping you to check the input and output formats, but its function on

checking the code correctness is limited. It can not guarantee the correctness of the code even with a full score in the submission report.

11. Some students encounter an error like: **the output rate …. has exceeded the allowed value ….bytes/s; attempting to kill the process.**

To resolve this, please remove **all print statements** and set the Spark logging level such that it limits the logs generated - that can be done using sc.setLogLevel . Preferably, set the log level to either **WARN** or **ERROR** when submitting your code.