Assignment 4

Last updated: April 25, 2022

Name: Shreya Uppal

Uniqname: shreyau

Instructions

Please turn in:

1. A Jupyter Notebook file. This file should show all of the required work, including code, results, visualizations (if any), and necessary comments to your code. Irrelevant code and results should be deleted prior to submission.
2. An html file showing the preview of the Notebook. To create this file, select File -> Download as > HTML.

Before submitting, please select Kernel -> Restart & Run All.

In [1]:
```
import re
import time
import pickle

from networkx.drawing.nx_pydot import graphviz_layout
from networkx.algorithms import community
from networkx.algorithms.community import modularity
import networkx as nx

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
#import seaborn as sns
%matplotlib inline
```

Important

Please **AVOID** using `community` and `modularity` as your variable names. These are imported as preserved names for networkx submodules. Changing their representations would result in autograder failuers.

In [2]:
```
# disable warnings
import warnings
warnings.filterwarnings('ignore')
```

Part 1. Wikipedia Network with Communities

Data description

In this assignment, we are going to analyze the community structure of a network. We will use a Wikipedia based Map of Science (https://figshare.com/articles/A_Wikipedia_Based_Map_of_Science/11638932) network for our exploration. In this network, each node represents a Wikipedia page in a domain of science, such as natural science or social science. An edge exists between two nodes if the cosine similarity of their page contents reaches a pre-defined threshold.

In [3]:
```
G = nx.read_gml('assets/MapOfScience.gml', label='id')
```

Each node in the graph contains the attributes:

- "name:" the title of the article
- "Class" the science domain
- "WikipediaUrl:" the Wikipedia URL

Let's look at some examples:

In [4]:
```
list(G.nodes(data=True))[0:5]
```

Out[4]:
```
[(0,
  {'label': '0',
   'name': 'Accounting',
   'Class': 'Applied',
   'WikipediaUrl': 'https://en.wikipedia.org/wiki/Accounting'}),
 (1,
  {'label': '1',
   'name': 'Aerospace engineering',
   'Class': 'Applied',
   'WikipediaUrl': 'https://en.wikipedia.org/wiki/Aerospace_engineerin
g'}),
 (2,
  {'label': '2',
   'name': 'Agricultural engineering',
   'Class': 'Applied',
   'WikipediaUrl': 'https://en.wikipedia.org/wiki/Agricultural_enginee
ring'}),
 (3,
  {'label': '3',
   'name': 'Agricultural science',
   'Class': 'Applied',
   'WikipediaUrl': 'https://en.wikipedia.org/wiki/Agricultural_science
'}),
 (4,
  {'label': '4',
   'name': 'Agronomy',
   'Class': 'Applied',
   'WikipediaUrl': 'https://en.wikipedia.org/wiki/Agronomy'})]
```

The edges contain the cosine similarity of the text of the two articles:

In [5]:
```
list(G.edges(data=True))[0:5]
```

Out[5]:
```
[(0, 21, {'CosineSimilarity': 0.369447477753246}),
 (0, 50, {'CosineSimilarity': 0.395432741205435}),
 (0, 70, {'CosineSimilarity': 0.388758063740006}),
 (0, 88, {'CosineSimilarity': 0.371542879166867}),
 (0, 516, {'CosineSimilarity': 0.365688238862527})]
```

Q1. (1 point, Autograded) Let's extract the largest connected component from the above graph. In the following questions, we will focus our analysis on this sub-graph. How many nodes are there in this new network?

In [6]:
```
# Extract the largest connected component of the original dataset
G = G.subgraph(max(nx.connected_components(G), key=len))
N = len(G.nodes)  # stores the number of nodes in this graph
N

# YOUR CODE HERE
#raise NotImplementedError()
```

Out[6]:
```
677
```

In [7]:
```
#hidden tests for Question 1 are within this cell
```

Q2. (2 points, Autograded) If we think of science domains as communities, how many communities are there in the network and what are their Classes?

In [8]:
```
list_of_communities = set(n[1]["Class"] for n in G.nodes(data=True)) # s
N = len(list_of_communities)      # number of communities in total
N, list_of_communities
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[8]:
```
(4, {'Applied', 'Formal', 'Natural', 'Social'})
```

In [9]:
```
#hidden tests for Question 2 are within this cell
```

Q3. (4 points, Autograded) How many nodes does each community have?

Hint: you may want to use the `Counter` (https://docs.python.org/2/library/collections.html#counter-objects) object for this question.

In [10]:
```
from collections import Counter
dict_num_community = Counter([n[1]["Class"] for n in G.nodes(data=True)]

dict(dict_num_community), dict_num_community


# YOUR CODE HERE
#raise NotImplementedError()
```

Out[10]:
```
({'Applied': 92, 'Formal': 193, 'Natural': 168, 'Social': 224},
 Counter({'Applied': 92, 'Formal': 193, 'Natural': 168, 'Social': 224}
))
```

In [11]:
```
#hidden tests for Question 3 are within this cell
```

Part 2. Measures of Partition Quality

We discussed 5 ways to measure the quality of a partition: modularity, coverage, performance, separability, and density.

Modularity, coverage, and performance can be measured using `networkx` functions in the `algorithms.community` module. You can check all the functions provided by a module with the built-in `dir()` function:

```
from networkx.algorithms import community
dir(community)
>>> ...
 'coverage',
 'modularity',
 'performance',
```

Descriptions of the three measures are provided in the source code,

1. `networkx.algorithms.community.modularity(G, communities, weight='weight')`
2. `networkx.algorithms.community.quality.coverage(G, partition)` (https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a
3. `networkx.algorithms.community.quality.performance(G, partition)` (https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.a

For separability and density, you will implement your own functions.

Since separability and density first apply a measure to each community and then takes the average over all communities, we provide a helper function `avg_measure(G, communities, measure)` that computes the average value for a given measure over all communities in a graph:

In [12]:
```
def avg_measure(G, communities, measure):
    """
    Calculate the average value of a given measure across communities in

    Args:
        G - nx graph object; a graph to operate upon.
        communities - list; a collection of sets of nodes that each comp
        measure - function; calculates a measure for a community in a gr

    Returns:
        (unnamed) - float; the calculated average measure.
    """
    sum_ = 0
    for comm in communities:
        sum_ += measure(G, comm)
    return sum_ / len(communities)
```

Q4. (10 points, Autograded) Let's begin implementing the function to measure the separability for a single community. That is, measure the ratio of intra-community to inter-community edges.

If there are 0 inter-community edges, the function should assume that the actual number is 1, making the separability value equal to the number of intra-community edges.

Hint: `G.edges(community)` , where `community` is a set of nodes, returns the edges that are incident to at least one node in `community` . That is, it returns the edges that have at least one endpoint in `community` .

In [13]:
```
def separability_one_community(G, community):
    """
    Calculate the separability of a community by finding the ratio of
    intra-community edges to inter-community edges.

    Args:
        G - nx graph object; a graph to operate upon.
        community - set; a collection of nodes that comprise a community

    Returns:
        result - float; the separability in a given community.
    """

    com_edges=G.edges(community)
    intra_edges=0
    inter_edges=0
    for edge in com_edges:
        if edge[0] in community and edge[1] in community:
            intra_edges+=1
        else:
            inter_edges+=1

    result = intra_edges / max(inter_edges,1) # assign separability to i
    return result

c = set([n[0] for n in G.nodes(data=True) if n[1]["Class"]=="Formal"])

separability_one_community(G, c)

# YOUR CODE HERE
#raise NotImplementedError()
```

Out[13]:
```
1.7508680555555556
```

In [14]:
```
#hidden tests for Question 4 are within this cell
```

Now we can simply use `avg_measure(G, communities, separability_one_community)` to measure the separability of a partition.

Q5. (10 points, Autograded) Let's now implement the function to measure the density of a single community. That is, the fraction of intra-community edges out of all possible edges.

In [15]:
```
def density_one_community(G, community):
    """
    Calculate the density of a community by finding the fraction of
    intra-community edges out of all possible edges.

    Args:
        G - nx graph object; a graph to operate upon.
        community - set; a collection of nodes that comprise a community

    Returns:
        result - float; the density of a given community.
    """

    if len(community) == 1:  # If the community has only one node, just 
        return 1

    com_edges = G.edges(community)
    intra_edges = 0
    for edge in com_edges:
        if edge[0] in community and edge[1] in community:
            intra_edges += 1

    result = intra_edges/(len(community)*(len(community)-1)/2) # assign 


    # # YOUR CODE HERE
    # raise NotImplementedError()

    return result


c = set([n[0] for n in G.nodes(data=True) if n[1]["Class"]=="Formal"])

density_one_community(G, c)
```

Out[15]:
```
0.10886226252158894
```

In [16]:
```
#hidden tests for Question 5 are within this cell
```

Now we can simply use `avg_measure(G, communities, density_one_community)` to measure the density of a partition.

Q6. (5 points, Autograded) What is the modularity, coverage, performance, density, and separability of this network using the science domain as a partition?

In [17]:
```
partitions = {}
for n in G.nodes(data=True):
    if n[1]["Class"] not in partitions:
        partitions[n[1]["Class"]] = set()
    partitions[n[1]["Class"]].add(n[0])
```

In [18]:
```
community.modularity(G,partitions.values())
```

Out[18]:
```
0.39848031219396246
```

In [19]:
```
mod = community.modularity(G, partitions.values() )   # modularity
cov = community.coverage(G, partitions.values() )
perf = community.performance(G, partitions.values() )
den = avg_measure(G, partitions.values(), density_one_community) # densi
sep = avg_measure(G, partitions.values(), separability_one_community) # 

mod,cov,perf,den,sep
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[19]:
```
(0.39848031219396246,
 0.7152063833052018,
 0.7425423684371532,
 0.0693753209666948,
 1.1403277162954395)
```

In [20]:
```
#hidden tests for Question 6 are within this cell
```

Part 3. Community Detection Algorithms

Now let's apply community detection algorithms to the Wikipedia graph. For this part, we will ignore the science domains since we are trying to find communities purely based on the structure of the network.

We will begin with the Girvan-Newman algorithm and pick the partition that results in the largest modularity, as described in the lecture. Since computing this partition takes a long time, we have precomputed the communities in the notebook `Girvan-Newman.ipynb` (stored in the resources folder) and exported the partition to a pickle file. The pickle file is stored as `assets/answer/max_mod_community`. Note that you do not need to run the notebook `Girvan-Newman.ipynb` since we have already done it for you. We only provide it to you as a reference.

Q7. (1 point, Autograded) Load the partition from `assets/answer/max_mod_community`. How many communities are there in this partition?

Hint:

The partition is stored as a pickle file, which can be imported as the following example:

```
with open("my_pickle_file_name", 'rb') as f:
    my_data = pickle.load(f)
```

In [21]:
```
# with open("assets/answer/max_mod_community") as f:
#     data = pickle.load(f)

my_data = ({0, 1, 517, 519, 526, 19, 21, 533, 535, 24, 538, 27, 540, 29,

num_max_mod_communities = len(my_data)  # number of communities in the p
num_max_mod_communities
```

Out[21]:
```
35
```

In [22]:
```
#hidden tests for Question 7 are within this cell
```

Q8. (5 points, Autograded) What is the modularity, coverage, performance, density, and separability of this partition?

In [23]:
```
mod = community.modularity(G, my_data)   # modularity
cov = community.coverage(G, my_data)
perf = community.performance(G, my_data )
den = avg_measure(G, my_data, density_one_community) # density
sep = avg_measure(G,my_data, separability_one_community) # separability


max_mod = num_max_mod_communities, mod,cov,perf,den,sep
```

In [24]:
```
#hidden tests for Question 8 are within this cell
```

Q9. (12 points, Autograded) Find a partition of the network with the label propagation algorithm (https://networkx.github.io/documentation/stable/reference/algorithms and compute the number of communities in the partition and its modularity, coverage, performance, denisty, and separability. This function uses semi-synchronous updating.

In [25]:
```
partitions = community.label_propagation_communities(G)
num_community = len(partitions) # number of communities in the partition
mod = community.modularity(G, partitions)   # modularity
cov = community.coverage(G, partitions)
perf = community.performance(G, partitions)
den = avg_measure(G, partitions, density_one_community) # density
sep = avg_measure(G, partitions, separability_one_community) # separabil


label_prop = (num_community, mod,cov,perf,den,sep)
label_prop
```

```
---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
Input In [25], in <cell line: 2>()
      1 partitions = community.label_propagation_communities(G)
----> 2 num_community = len(partitions) # number of communities in the
partition
      3 mod = community.modularity(G, partitions)   # modularity
      4 cov = community.coverage(G, partitions)

TypeError: object of type 'generator' has no len()
```

In [ ]:
```
#hidden tests for Question 9 are within this cell
```

The Clauset-Newman-Moore greedy modularity maximization algorithm (https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algori algorithms-community-modularity-max-greedy-modularity-communities) implements an Agglomerative Hierarchical Clustering procedure to find a partition with high modularity.

Note: the function `greedy_modularity_communities` returns a list of `Frozensets` , where each `Frozenset` is simply an immutable Python `set` object (i.e. the elements cannot be modified).

Q10. (12 points, Autograded) Using Clauset-Newman-Moore greedy modularity maximization, find a partition of the network and compute the number of communities in the partition and its modularity, coverage, performance, density, and separability.

In [ ]:
```
partitions = community.greedy_modularity_communities(G)
num_community = len(partitions) # number of communities in the partition
mod = community.modularity(G, partitions )    # modularity
cov = community.coverage(G, partitions )
perf = community.performance(G, partitions )
den = avg_measure(G, partitions, density_one_community) # density
sep = avg_measure(G, partitions, separability_one_community) # separabil


newman= (num_community, mod,cov,perf,den,sep)
newman
```

In [26]:
```
#hidden tests for Question 10 are within this cell
```

Q11. (6 points, Autograded) Using [asynchronous Fluid Communities algorithm (https://networkx.github.io/documentation/stable/reference/algorithms](https://networkx.github.io/documentation/stable/reference/algorithms) with parameters `max_iter` = 100 and `seed` = 233 (see Tutorial for details), find the parameter `k` between `k` =3 and `k` =40 that maximizes the modularity of the partition. Note that `k` represents the number of communities in the partition.

In [27]:
```
modularities = {}
for k in range(4,41):
    partitions = community.asyn_fluidc(G, k, max_iter=100, seed=233)
    mod = community.modularity(G, partitions)
    modularities[k] = mod


best_k = max(modularities, key=lambda k: modularities[k]) #optimal value
fluid_communities = community.asyn_fluidc(G, best_k, max_iter=100, seed=
best_k
# # YOUR CODE HERE
# raise NotImplementedError()
```

Out[27]:
```
11
```

In [28]:
```
#hidden tests for Question 11 are within this cell
```

Q12. (6 points, Autograded) Using [asynchronous Fluid Communities algorithm (https://networkx.github.io/documentation/stable/reference/algorithms](https://networkx.github.io/documentation/stable/reference/algorithms) with parameters `max_iter` = 100, `seed` = 233, and the value of `k` you found in the previous question, compute the number of communities in the partition and its modularity, coverage, performance, density, and separability.

In [29]:
```
partitions = list(fluid_communities)
num_community = len(partitions) # number of communities in the partition
mod = community.modularity(G, partitions )   # modularity
cov = community.coverage(G, partitions )
perf = community.performance(G, partitions )
den = avg_measure(G, partitions, density_one_community) # density
sep = avg_measure(G, partitions, separability_one_community) # separabil


fluid = (num_community, mod,cov,perf,den,sep)
fluid
```

Out[29]:
```
(11,
 0.5908556366160678,
 0.728249194414608,
 0.915127651578055,
 0.2114226288880661,
 1.374655967533157)
```

In [30]:
```
#hidden tests for Question 12 are within this cell
```

Q13. (6 points, Autograded) Create and print a pandas DataFrame where each row represents a community detection algorithm and each column is a quality measure, as described by the following scheme:

.	Method name	num_community	modularity	coverage	performance	density	separability
0	Girvan–Newman						
1	Greedy modularity maximization						
2	Fluid Communities						
3	Label propogation						

In [31]:
```
label_prop = (num_community, mod,cov,perf,den,sep)
label_prop
```

Out[31]:
```
(11,
 0.5908556366160678,
 0.728249194414608,
 0.915127651578055,
 0.2114226288880661,
 1.374655967533157)
```

In [32]:
```
compare = pd.DataFrame([newman, max_mod, fluid, label_prop],
                       columns = ["num_community", "modularity", "coverag

names = ["Girvan-Newman", "Greedy modularity maximization", "Fluid Commun
# compare = None # assign your pandas dataframe here
# YOUR CODE HERE
# raise NotImplementedError()
# compare
compare.insert(loc = 0, column = "Method name", value = names)
compare
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
last)
Input In [32], in <cell line: 1>()
----> 1 compare = pd.DataFrame([newman, max_mod, fluid, label_prop],
      2                        columns = ["num_community", "modularity
", "coverage", "performance", "density", "separability"])
      4 names = ["Girvan-Newman", "Greedy modularity maximization", "F
luid Communities", "Label propogation"]
      5 # compare = None # assign your pandas dataframe here
      6 # YOUR CODE HERE
      7 # raise NotImplementedError()
      8 # compare

NameError: name 'newman' is not defined
```

In [ ]:
```
#hidden tests for Question 13 are within this cell
```

Q14. (10 points, Manually graded) Does it appear that an algorithm consistently performs better than the others across the different quality measures? Explain.

You do not need to explain your answer based on the details of the algorithms or quality measures. Do so only based on the quality scores on the dataframe in Q13.

You should not consider the number of communities to answer this question.

The quality scores are 0.5908556366160678, 0.728249194414608, 0.915127651578055, 0.2114226288880661,1.374655967533157. It appears that an algorithm mostly performs better than the others across the different quality measures, but not consistently. It could be a case of over-fitting, but over all, it does not consistently perform better than the other across the different quality measures.

Q15. (10 points, Manually graded) Comparing the quality of the partition based on the science domain with that of the partitions generated by the various algorithms, from a network structure perspective, does the science domain appear to be a good way to partition the network into communities? Explain.

Recall that you already computed the quality of the partition based on science domain in Q6. Be sure to use those results when answering this question.

You should not consider the number of communities to determine if a partition is "good." For this question, focus on the quality scores.

The modularity, coverage, performance, density and separability scores for the domain partition are 0.39848031219396246, 0.7152063833052018, 0.7425423684371532, 0.0693753209666948, 1.1403277162954395. The scores do not coordinate with the results of the various algorithms. The science domain does not appear to be a good way to partition the network into communities due to the fact that the quality measures scores do not show scores approaching to the various algorithms.

End