Assignment 4

Last updated: April 25, 2022

Name: Adam Bakopolus

Uniqname: abakop

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)](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_engineering'}),
 (2,
  {'label': '2',
   'name': 'Agricultural engineering',
   'Class': 'Applied',
   'WikipediaUrl': 'https://en.wikipedia.org/wiki/Agricultural_engineering'}),
 (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 = G.number_of_nodes()  # stores the number of nodes in this graph
```

```
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]: temp_list = list(G.nodes(data=True))
        domains = []

        for node in temp_list:
            domains.append(node[1]['Class'])

        list_of_communities = set(domains) # set of unique community labels
        N = len(list_of_communities)           # number of communities in total
```

```
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

         node_list = []

         for node in temp_list:
             node_list.append(node[1]['Class'])

         dict_num_community = Counter(node_list)   # a Counter object with the format {community_name: number_of_nodes}
```

```
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.algorithms.community.quality.coverage.html#](https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.quality.coverage.html)
3. [`networkx.algorithms.community.quality.performance(G, partition)` (https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.quality.performance.h](https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.quality.performance.h)

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]:
```python
def avg_measure(G, communities, measure):
    """
    Calculate the average value of a given measure across communities in a graph.

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

    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]:
```python
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.
    """

    intra_edges = []
    total_edges = G.edges(community)

    for edge in total_edges:
        if edge[0] in community and edge[1] in community:
            intra_edges.append(edge)
    if len(total_edges) - len(intra_edges) == 0:
        result = len(intra_edges)
    else:
        result = len(intra_edges)/(len(total_edges) - len(intra_edges))

    return result
```

In [14]:
```python
#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.
    """

    intra_edges = []
    community_nodes = len(community)

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

    else:

        total_possible_edges = (community_nodes * (community_nodes - 1))/2

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

        if total_possible_edges == 0:
            result = len(intra_edges)
        else:
            result = len(intra_edges)/total_possible_edges

        return result
```

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]:

```
applied = []
formal = []
natural = []
social = []

for node in temp_list:
    if node[1]['Class'] == 'Applied':
        applied.append(node[0])
    elif node[1]['Class'] == 'Formal':
        formal.append(node[0])
    elif node[1]['Class'] == 'Natural':
        natural.append(node[0])
    else:
        social.append(node[0])

applied = set(applied)
formal = set(formal)
natural = set(natural)
social = set(social)

communities = [applied, formal, natural, social]

mod = community.modularity(G, communities, weight='weight')   # modularity
cov = community.quality.coverage(G, communities)   # coverage
perf = community.quality.performance(G, communities)  # performance
den = avg_measure(G, communities, density_one_community) # density
sep = avg_measure(G, communities, separability_one_community) # separability

science_domain = [mod, cov, perf, den, sep]
```

In [18]:

```
#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 [19]:

```
with open("assets/answer/max_mod_community", 'rb') as f:
    gn_communities = pickle.load(f)

num_max_mod_communities = len(gn_communities)  # number of communities in the partition with the largest modular
```

In [20]:

```
#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 [21]:

```
with open("assets/answer/max_mod_community", 'rb') as f:
    gn_communities = pickle.load(f)

mod = community.modularity(G, gn_communities, weight='weight')   # modularity
cov = community.quality.coverage(G, gn_communities)   # coverage
perf = community.quality.performance(G, gn_communities)  # performance
den = avg_measure(G, gn_communities, density_one_community) # density
sep = avg_measure(G, gn_communities, separability_one_community) # separability

garvin_newman = [num_max_mod_communities, mod, cov, perf, den, sep]
```

In [22]:
```
#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/generated/networkx.algorithm](https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithm) and compute the number of communities in the partition and its modularity, coverage, performance, denisty, and separability. This function uses semi-synchronous updating.

In [23]:
```
lp_generator = community.label_propagation.label_propagation_communities(G)
lp_partition = list(lp_generator)

num_community = len(lp_partition) # number of communities in the partition
mod = community.modularity(G, lp_partition, weight='weight')   # modularity
cov = community.quality.coverage(G, lp_partition)   # coverage
perf = community.quality.performance(G, lp_partition)  # performance
den = avg_measure(G, lp_partition, density_one_community) # density
sep = avg_measure(G, lp_partition, separability_one_community) # separability

label_prop = [num_community, mod, cov, perf, den, sep]
```

In [24]:
```
#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.algorithms.community.modularity_max.greedy_m algorithms-community-modularity-max-greedy-modularity-communities)](https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.modularity_max.greedy_m) 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 [25]:
```
gred_partition = community.greedy_modularity_communities(G)

num_community = len(gred_partition) # number of communities in the partition
mod = community.modularity(G, gred_partition, weight='weight')   # modularity
cov = community.quality.coverage(G, gred_partition)   # coverage
perf = community.quality.performance(G, gred_partition)  # performance
den = avg_measure(G, gred_partition, density_one_community) # density
sep = avg_measure(G, gred_partition, separability_one_community) # separability

c_n_m = [num_community, mod, cov, perf, den, sep]
```

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/generated/networkx.algorithm](https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithm) 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]:
```
partition_options = []
for k in range(3, 41):
    fluid_partition = list(community.asyn_fluidc(G, k, max_iter=100, seed=233))
    mod_testing = community.modularity(G, fluid_partition, weight='weight')
    partition_options.append((k, mod_testing))

best_k = max(partition_options, key= lambda x: x[1])[0] #optimal value of k
fluid_communities = list(community.asyn_fluidc(G, best_k, max_iter=100, seed=233)) #partition with k = best_k
```

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/generated/networkx.algorithm](https://networkx.github.io/documentation/stable/reference/algorithms/generated/networkx.algorithm) 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]:
```
num_community = len(fluid_communities) # number of communities in the partition
mod = community.modularity(G, fluid_communities, weight='weight')   # modularity
cov = community.quality.coverage(G, fluid_communities)   # coverage
perf = community.quality.performance(G, fluid_communities)  # performance
den = avg_measure(G, fluid_communities, density_one_community) # density
sep = avg_measure(G, fluid_communities, separability_one_community) # separability

fluid = [num_community, mod, cov, perf, den, sep]
```

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]:
```
gn = ['Girvan-Newman', garvin_newman[0], garvin_newman[1], garvin_newman[2], garvin_newman[3], garvin_newman[4],
cnm = ['Greedy modularity maximization', c_n_m[0], c_n_m[1], c_n_m[2], c_n_m[3], c_n_m[4], c_n_m[5]]
f = ['Fluid Communities', fluid[0], fluid[1], fluid[2], fluid[3], fluid[4], fluid[5]]
lp = ['Label propogation', label_prop[0], label_prop[1], label_prop[2], label_prop[3], label_prop[4], label_prop


tmp = [gn, cnm, f, lp]
compare = pd.DataFrame(tmp, columns =['Method name', 'num_community', 'modularity', 'coverage', 'performance', '
compare
```

Out[31]:

	Method name	num_community	modularity	coverage	performance	density	separability
0	Girvan-Newman	35	0.580687	0.815252	0.888706	0.654218	1.079485
1	Greedy modularity maximization	14	0.550356	0.805892	0.775095	0.608520	1.480856
2	Fluid Communities	11	0.590856	0.728249	0.915128	0.211423	1.374656
3	Label propogation	28	0.584188	0.808808	0.857888	0.618999	1.285533

In [32]:
```
#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.

While no algorithm performed the best across ALL 5 quality measures, the Girvan-Nerman algorithm performed consistently better than the others, having the highest modularity, coverage, and density scores. The performance score was also second highest when using this algorithm. Therefore, this algorithm performed consistently better in this notebook's example. This top-down algorithm is very computationally expensive but does result in a partition with high quality scores.

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

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.

When comparing the quality of the science domain partition with the partitions generated by the various algorithms, it's clear that the science domain was not a good way to partition the network into communities. When compared to the algorithms, the modularity, coverage, performance, and density scores were the lowest, and the separability score was only better than the Girvan-Newman algorithm. With poor quality scores across the board, this partition should not be used in favor of the other algorithms.

End