Programming-Assignment-1

July 1, 2020

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

1 Assignment 1 - Creating and Manipulating Graphs

Eight employees at a small company were asked to choose 3 movies that they would most enjoy watching for the upcoming company movie night. These choices are stored in the file `Employee_Movie_Choices.txt`.

A second file, `Employee_Relationships.txt`, has data on the relationships between different coworkers.

The relationship score has value of `-100` (Enemies) to `+100` (Best Friends). A value of zero means the two employees haven't interacted or are indifferent.

Both files are tab delimited.

[1]:
```
import networkx as nx
import pandas as pd
import numpy as np
from networkx.algorithms import bipartite


# This is the set of employees
employees = set(['Pablo',
                 'Lee',
                 'Georgia',
                 'Vincent',
                 'Andy',
                 'Frida',
                 'Joan',
                 'Claude'])

# This is the set of movies
movies = set(['The Shawshank Redemption',
              'Forrest Gump',
              'The Matrix',
```

```
              'Anaconda',
              'The Social Network',
              'The Godfather',
              'Monty Python and the Holy Grail',
              'Snakes on a Plane',
              'Kung Fu Panda',
              'The Dark Knight',
              'Mean Girls'])


# you can use the following function to plot graphs
# make sure to comment it out before submitting to the autograder
def plot_graph(G, weight_name=None):
    '''
    G: a networkx G
    weight_name: name of the attribute for plotting edge weights (if G is␣
 ↪weighted)
    '''
    %matplotlib notebook
    import matplotlib.pyplot as plt

    plt.figure()
    pos = nx.spring_layout(G)
    edges = G.edges()
    weights = None

    if weight_name:
        weights = [int(G[u][v][weight_name]) for u,v in edges]
        labels = nx.get_edge_attributes(G,weight_name)
        nx.draw_networkx_edge_labels(G,pos,edge_labels=labels)
        nx.draw_networkx(G, pos, edges=edges, width=weights);
    else:
        nx.draw_networkx(G, pos, edges=edges);
```

1.0.1 Question 1

Using NetworkX, load in the bipartite graph from `Employee_Movie_Choices.txt` and return that graph.

This function should return a networkx graph with 19 nodes and 24 edges

[2]:
```
def answer_one():

    G = nx.read_adjlist('Employee_Movie_Choices.txt', delimiter = '\t')

    return G

G = answer_one()
```

1.0.2 Question 2

Using the graph from the previous question, add nodes attributes named `'type'` where movies have the value `'movie'` and employees have the value `'employee'` and return that graph.

This function should return a networkx graph with node attributes `{'type': 'movie'}` or `{'type': 'employee'}`

[3]:
```
def answer_two():

    G = answer_one()
    for g in G:
        if g in employees:
            G.add_node(g, type = 'employee')
        elif g in movies:
            G.add_node(g, type = 'movie')

    return G
```

1.0.3 Question 3

Find a weighted projection of the graph from `answer_two` which tells us how many movies different pairs of employees have in common.

This function should return a weighted projected graph.

[4]:
```
def answer_three():

    G = answer_two()
    wpg = bipartite.weighted_projected_graph(G, employees)

    return wpg
```

1.0.4 Question 4

Suppose you'd like to find out if people that have a high relationship score also like the same types of movies.

Find the Pearson correlation (using `DataFrame.corr()`) between employee relationship scores and the number of movies they have in common. If two employees have no movies in common it should be treated as a 0, not a missing value, and should be included in the correlation calculation.

This function should return a float.

[5]:
```
def fill_dict(val):
    if val is np.nan:
        return {'weight': 0}
    else:
        return val

def answer_four():
```

```
    G = answer_three()

    relationship = nx.read_edgelist('Employee_Relationships.txt',␣
↪data=[('relationship_score', int)])

    df_G = pd.DataFrame(G.edges(data = True), columns = ['From', 'To',␣
↪'movies_score'])
    df_relationship = pd.DataFrame(relationship.edges(data = True), columns =␣
↪['From', 'To', 'relationship_score'])

    df_G_copy = df_G.copy()
    #df_G_copy.rename({'From': 'To', 'To': 'From'}, inplace = True)
    df_G_copy.rename(columns={"From":"From1", "To":"From"}, inplace=True)
    df_G_copy.rename(columns={"From1":"To"}, inplace=True)

    df_concat = pd.concat([df_G, df_G_copy])


    df_final = pd.merge(df_concat, df_relationship,  on = ['From', 'To'], how =␣
↪'right')


    # get rid of the NaN values and substitute it with a {'weight': 0}
    #df_final['movies_score'] = [row if isinstance(row, dict) else␣
↪dict(weight=0) for row in df_final['movies_score']]
    df_final['movies_score'] = df_final['movies_score'].map(fill_dict)
    df_final['movies_score'] = df_final['movies_score'].map(lambda x:␣
↪x['weight'])

    df_final['relationship_score'] = df_final['relationship_score'].map(lambda␣
↪x: x['relationship_score'])

    outcome = df_final['movies_score'].corr(df_final['relationship_score'],␣
↪method = 'pearson')


    return outcome


answer_four()
```

[5]: 0.78839622217334726