Assignment 1

Last updated: June 23, 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. This file is submitted automatically when you submit your notebook to be autograded. This is done in Assignment 1 -- Create.
2. An HTML file of the Notebook. Submit this file in Assignment 1 - Submit
3. A PDF file of the Notebook. Submit this file in Assignment 1 - Submit.

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

Please do not remove any code outside of the Not Implemented Error sections. The autograder may need it.

Assignment 1

In this assignment, we are going to practice analyzing networks using the various measurements and metrics that we have learned this week. You will work with three different network datasets. First, you will choose one of the networks and explore its features. Then you will complete a prediction task using a Facebook network.

Dataset descriptions

Below are three data sets that you can work with. For this assignment, you only need to choose one, but you are welcome to explore the others.

1. Star wars interaction graph

- `starwars-full-interactions.json` : characters in Star Wars and their interactions. Each character is treated as a node, and an edge is created if two characters ever appear in the same scene.
- `star_war_label.csv` : labels of the most prominent characters. A character has label 1 if they frequently appear in the Star Wars movies and 0 otherwise. The order of the names and labels in this file is the same as their node number.

> Source: Gabasova, E. (2016). Star Wars social network. DOI: https://doi.org/10.5281/zenodo.1411479 (https://doi.org/10.5281/zenodo.1411479).

2. Facebook friendships

- `slavko.net` : an edge list from a friendship network. Each node represents a user on Facebook.
- `slavko_label.txt` : labels of the nodes indicate their level of influence. A label of 1 indicates the node is "influential," and a label of 0 indicates otherwise. The order of the nodes and labels in this file is the same as their node number. The labels of this graph were synthetically generated, but you can think of them as representing the results from a survey, where users were asked to identify Facebook friends who influence their opinions. A node with a label 1 is a user who was identified as influential by at least one of their friends.

> Blagus, N., Šubelj, L. & Bajec, M. (2012). *Self-similar scaling of density in complex real-world networks* (http://wwwlovre.appspot.com/resources/research/bibs/ssd.bib), Physica A: Statistical Mechanics and its Applications 391(8), 2794-2802, doi:10.1016/j.physa.2011.12.055 (http://dx.doi.org/10.1016/j.physa.2011.12.055), e-print arXiv:1110.5609 (http://arxiv.org/abs/1110.5609), COBISS:8930132 (http://cobiss.izum.si/scripts/cobiss?command=DISPLAY&lani=en&base=COBIB&RID=8930132).

3. Students' Cooperation Social Network

- `multigraph_hashAnonymized.csv` : a cooperation network between students. The nodes are students and an edge between two students indicates that the two students have cooperated in at least one academic activity.
- `multigraph_label.txt` : academic improvement labels. A label of 1 indicates the student has made academic improvement during a period of cooperation and 0 otherwise.

> Fire, M., Katz, G., Elovici, Y., Shapira, B., and Rokach, L. Fire, Michael, et al. "Predicting student exam's scores by analyzing social network data." Active Media Technology. Springer Berlin Heidelberg, 2012. 584-595.

In [1]:
```
import networkx as nx
import pandas as pd
import numpy as np

import urllib
import json
import operator

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

Part 1. Data exploration

For this part, you will choose one of the three datasets described above. We provide the function `get_graph` for you, which extracts a dataset of your choice and creates the corresponding network.

In [2]:
```
def get_graph(dataset):
    if dataset == "starwars":
        f=open("assets/starwars-full-interactions.json")
        data = json.load(f)
        char_map = {entry['name']: i for i, entry in enumerate(data['nodes'])}
        edges = [(edge['source'], edge['target']) for edge in data['links']]
        G=nx.Graph()
        G.add_nodes_from([i for i in range(len(char_map))])
        G.add_edges_from(edges)
    elif dataset == "facebook":
        G = nx.read_edgelist("assets/slavko.net", nodetype=int)
    elif dataset == "student":
        G = nx.read_edgelist("assets/student.txt", nodetype=int)
    else:
        raise ValueError("Not a valid dataset")
    return G
```

Indicate your choice of data in the cell below, using one of the following strings

- `"starwars"`
- `"facebook"`
- `"student"`

In [3]:
```
DATASET = "starwars"  # assign the string of your chosen dataset here
```

In [4]:
```
#hidden tests for Part 1 are within this cell
```

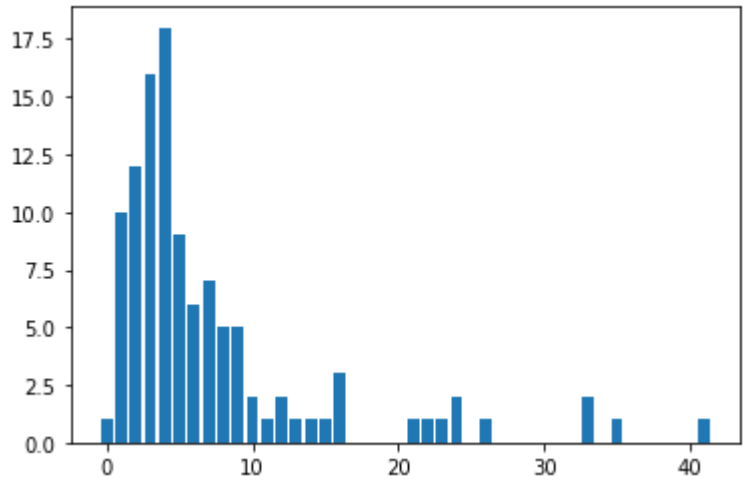
Q1. (2 points, Autograded) Plot the histogram of degree distribution.

Use the `degree_histogram` (https://networkx.github.io/documentation/stable/reference/generated/networkx.classes.function.degree_histogram.html#networkx.classes.f function and then plot the histogram using `plt.bar` (https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.bar.html).

Important: to receive full points from the autograder, please don't modify additional parameters that adjust the appearance of the figure, such as `width`, `bottom`, etc.

In [5]:
```
# get graph
G = get_graph(DATASET)    # assign your graph here

y = nx.degree_histogram(G)
x = range(len(y))
img = plt.bar(x, y)
plt.show()
```



In [6]:
```
#hidden tests for Question 1 are within this cell
```

Q2. (3 points, Autograded) Which node has the largest degree centrality and what is the value?

Hint: you can sort a dictionary `D` by its values using

```
sorted(D.items(), key=lambda item: item[1])
```

In [7]:
```
centrality = nx.degree_centrality(G)
cent_sorted = sorted(centrality.items(), key=lambda item: item[1])
max_value = cent_sorted[-1]

node_max_degree = max_value[0]  # the node which has the largest degree centrality value. This should be an inte
val_deg = max_value[1] # the centrality value of the node with the largest degree centrality. This should be a f
```

In [8]:
```
#hidden tests for Question 2 are within this cell
```

Q3. (5 points, Autograded) List the 5 nodes with the largest and smallest closeness centrality.

In [9]:
```
closeness = nx.closeness_centrality(G)
close_sorted = sorted(closeness.items(), key=lambda item: item[1])

max_5_cc = [n for n, close in close_sorted[-5:]] # a list of 5 nodes with the largest closeness centrality value
min_5_cc = [n for n, close in close_sorted[:5]]  # a list of 5 nodes with the smallest closeness centrality valu
```

In [10]:
```
#hidden tests for Question 3 are within this cell
```

Q4. (5 points, Autograded) List the 5 nodes with the largest and smallest betweenness centrality.

In [11]:
```
between = nx.betweenness_centrality(G)
bet_sorted = sorted(between.items(), key=lambda item: item[1])

max_5_bc = [n for n, bet in bet_sorted[-5:]] # a list of 5 nodes with the largest betweenness centrality values
min_5_bc = [n for n, bet in bet_sorted[:5]]  # a list of 5 nodes with the smallest betweenness centrality values
```

In [12]:
```
#hidden tests for Question 4 are within this cell
```

Q5. (5 points, Autograded) What are the 3 nodes with the largest PageRank values in the graph?

Use damping coefficient $\alpha = 0.9$ and create a list of the three nodes with PageRank values in descending order.

Check whether each pair of nodes among the top 3 nodes with the highest PageRank are connected by an edge.

Hint: See `Graph.has_edge` (https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.Graph.has_edge.html?highlight=has_edge#networkx.Graph.has_edge)

In [13]:
```
page = nx.pagerank(G, alpha = .9)
page_sorted = sorted(page.items(), key=lambda item: item[1])
page_sorted[-3:]
result = [page_sorted[-1][0], page_sorted[-2][0], page_sorted[-3][0]] # list of the 3 nodes with the largest Pag

has_edge12 = G.has_edge(result[0], result[1]) # boolean variable indicating whether 1st and 2nd nodes in `result
has_edge13 = G.has_edge(result[0], result[2]) # boolean variable indicating whether 1st and 3rd nodes in `result
has_edge23 = G.has_edge(result[1], result[2]) # boolean variable indicating whether 2nd and 3rd nodes in `result
```

In [14]:
```
#hidden tests for Question 5 - checking result - are within this cell
```

In [15]:
```
#hidden tests for Question 5 - checking edges - are within this cell
```

Q6 (a). (5 points, Autograded) List the 5 nodes with the largest hub scores and the 5 nodes with the largest authority scores.

In [16]:
```
hubs, auth = nx.hits(G)

hubs_sorted = sorted(hubs.items(), key=lambda item: item[1])
h5 = [n for n, hub in hubs_sorted[-5:]]

auth_sorted = sorted(auth.items(), key=lambda item: item[1])
a5 = [n for n, auth in auth_sorted[-5:]]
```

In [17]:
```
#hidden tests for Question 6a are within this cell
```

Q6 (b). (2 points, Manually graded) In part 6a, you should have found that nodes with the largest hub score are also the nodes with the largest authority score. In fact, the full ranking of nodes by hub score is the same as the ranking by authority score. This is interesting since we do not typically expect that the best hubs are also the best authorities, according to the HITS algorithm. Explain what features of this particular network make these rankings equal.

This finding where the nodes with the largest hub scores are also the nodes with the largest authority scores is reasonable given the Star Wars dataset. The network that is created by the get_graph function is undirected (unweighted, as well) but the main characters of the films will be those with the highest degree centrality and will have strong hub scores. However, given that these main characters will also be the major focus of the story as well, they will frequently have edges to the other main characters as well as many of the ancillary

characters, making them strong authorities, as well. Therefore, given that the main characters of the film have such high degree centrality, they are assigned the same large hub and authority scores. The unweighted and undirected nature of the network as well is driving the unusual situation where the hub and authority scores are the same.

Q7. (5 points, Manually graded) Reflection:

Did you notice any differences between the nodes with the highest scores across the different centrality measures? Is the amount of overlap between the lists of the top nodes expected?

In [18]:
```
print('The nodes with the highest closeness centrality are ' + str(max_5_cc))
print('The nodes with the highest betweenness centrality are ' + str(max_5_bc))
print('The node with the highest degree centrality is ' + str(node_max_degree))
```

```
The nodes with the highest closeness centrality are [70, 64, 17, 21, 4]
The nodes with the highest betweenness centrality are [64, 17, 70, 21, 4]
The node with the highest degree centrality is 17
```

The same 5 nodes have both the highest closeness centrality and the highest betweenness centrality scores across the Star Wars dataset. Additionally, the node with the highest degree centrality was also included in those two other lists. For this particular dataset, this amount of overlap between the lists of the top nodes is expected. These nodes correspond to the main characters of the films who would easily have the highest degree centraility due to their screen time and the amount of interaction the'd have across both main and supporting casts. Additionally, main characters would be expected to have high betweenness scores as well as they will likely always be a part of the shortest path for supporting or main characters to reach other characters due to their high degree centrality and that important nodes typically connect other nodes. Similarly, closeness centraility scores would be high as well following this same logic, with the main character nodes, due to their importance, being close distance-wise to other nodes.

Q8. (3 points, Autograded) What are the average clustering coefficient and graph transitivity of the network?

Transitivity and clustering coefficient are related to the tendency for a network to have triangles. However, they differ in that while clustering coefficient computes clustering on a node-by-node basis first and then averages over the nodes, transitivity computes the fraction of triplets that form a triangle over the entire network.

In [19]:
```
graph_trans = nx.transitivity(G)      # graph transitivity. This should be a float.
avg_clustering = nx.average_clustering(G)   # average clustering coefficient value. This should be a float.
```

In [20]:
```
#hidden tests for Question 8 are within this cell
```

Part 2. Node centrality and connected components

In this part, we are going to remove nodes from a graph and assess the effect that has on the number of connected components in the graph. Generally, removing nodes will tend to **increase** the number of connected components as the graph begins to fragment. However, not all nodes will have the same effect. Removing some highly central nodes is more likely to fragment the graph (and increase the number of connected components) than removing some nodes with low centrality. This is exactly what we want to demonstrate in this part.

Starting with the entire graph, compute the number of connected components. Then remove 150 total nodes, one at a time. After each node removal, compute the number of connected components.

We will compare 3 strategies for choosing which 150 nodes to remove:

1. Remove randomly chosen nodes.
2. Remove the node with the highest degree centrality in the current graph.
3. Remove the node with the highest betweenness centrality in the current graph.

The goal is to test what strategy for node removal will fragment the graph faster.

Note:

1. We will use the **Facebook** network for this part.
2. Use the `random.choice` function for the random selection, and set the random seed to 0.
3. Do not use `np.random.choice` .

In [21]:
```
G = get_graph("facebook")
```

In [22]:
```
import copy
import random
```

9(a). (10 points, Autograded) Let's begin with the first strategy: removing random nodes from the graph.

Using the list `rand_move`, append the number of connected components after removing each of the 150 randomly chosen nodes.

`rand_move` will be initialized with the number of connected components before removing any nodes. You will then append the number of connected components after removing each of the 150 nodes. At the end of the process, `rand_move` will contain 151 integers.

Please use the function `random.choice` 150 times to select each node (instead of using it just once to select all 150 nodes).

In [23]:
```
random.seed(0)          # use a fixed random seed for autograder
G1 = copy.deepcopy(G)  # operate on a copy of the original graph, because node removal is irreversible
rand_move = [nx.number_connected_components(G1)]  # initialize the list of number of connected components
#Append the number of connected components to this list after removing each of the 150 randomly chosen nodes.
for i in range(150):
    remove_me = random.choice(list(G1.nodes))
    G1.remove_node(remove_me)
    rand_move.append(nx.number_connected_components(G1))
```

In [24]:
```
#hidden tests for Question 9a are within this cell
```

9(b). (10 points, Autograded) Now let's change the strategy to choosing the nodes with the highest degree centrality.

Using the list `deg_move`, append the number of connected components after removing each of the 150 nodes. At each step, remove the node with the **currently** highest degree centrality. If multiple nodes have the same centrality values, remove the node which has the **largest node number**. Note that you must recompute the degree centrality of the nodes after removing each node since node removal impacts centrality.

Hint:

To sort by multiply keys, you can use the Python `sorted` function with a specified tuple of keys.

For example, to sort a sequence of pairs `arr = [(x1, y1), (x2, y2), ...(xn, yn)]` first by `x` then by `y`, the `key` is speficied as the following:

```
sorted(arr, key=lambda pair: (pair[0], pair[1]))
```

In [25]:
```
G1 = copy.deepcopy(G)
deg_move = [nx.number_connected_components(G1)]  # initialize the list of number of connected components
#As in Q9(a), append the number of connected components to this list after removing each node.
for i in range(150):
    centrality = nx.degree_centrality(G1)
    cent_sorted = sorted(centrality.items(), key=lambda item: (item[1], item[0]))
    remove_me = cent_sorted[-1][0]

    G1.remove_node(remove_me)
    deg_move.append(nx.number_connected_components(G1))
```

In [26]:
```
#hidden tests for Question 9b are within this cell
```

9(c). (10 points, Autograded) Finally, let's adopt the strategy of choosing the nodes with the highest betweenness centrality.

Using the list `bet_move` , append the number of connected components after removing each of the 150 nodes. At each step, remove the node with the **currently** highest betweenness centrality. If multiple nodes have the same centrality values, remove the node which has the **largest node number**. Note that you must recompute the betweenness centrality of the nodes after removing each node since node removal impacts centrality.

In [27]:
```
G1 = copy.deepcopy(G)
bet_move = [nx.number_connected_components(G1)]  # initialize the list of number of connected components
#As in Q9(a), append the number of connected components to this list after removing each node.
for i in range(150):
    between = nx.betweenness_centrality(G1)
    bet_sorted = sorted(between.items(), key=lambda item: (item[1], item[0]))
    remove_me = bet_sorted[-1][0]

    G1.remove_node(remove_me)
    bet_move.append(nx.number_connected_components(G1))
```

In [28]:
```
#hidden tests for Question 9c are within this cell
```

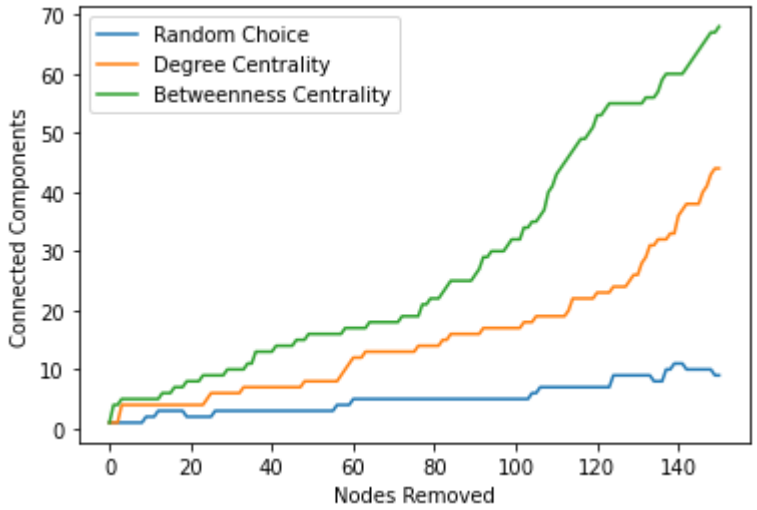
9(d). (2 points, Manually graded) Plot the results.

Now that you have the lists `rand_move`, `deg_move`, and `bet_move`, make a single plot that shows the number of removed nodes on the x-axis and the number of connected components on the y-axis. The plot should include three curves -- one for each node removal strategy.

In [29]:
```
x = range(151)

plt.plot(x, rand_move, label = "Random Choice")
plt.plot(x, deg_move, label = "Degree Centrality")
plt.plot(x, bet_move, label = "Betweenness Centrality")

plt.legend()
plt.xlabel('Nodes Removed')
plt.ylabel('Connected Components')
plt.show()
```



Q9(e). (15 points, Manually graded) Which node removal strategy made the number of connected components increase the fastest? The slowest? Explain your results.

The node removal strategy that made the number of connected components increase the fastest was when the nodes with the highest betweenness centrality were removed. The slowest was the removal of nodes by random choice. The betweenness centrality assumption notes that important nodes connect other nodes. When nodes with the highest score are therefore removed, connected components increase the fastest as "bridges" to other nodes quickly are removed. Random choice has a much slower effect as nodes with low degree

centrality or low betweenness centrality may end up being removed early on in the process, leading to few connected components. Degree centrality removal falls in the middle as removing nodes with a large number of edges will have a far greater impact than random choice, but these nodes may not be critical "bridges" and therefore lead to a smaller impact than the high betweenness centrality nodes.

Part 3. Prediction (18 points, Autograded)

In this part, you are going to complete a supervised prediction task on the **Facebook** network. The task is to predict whether a node is labeled as an "influential" user. You will be given the following two files:

1. `facebook_train.csv` , which contains the training set.
2. `facebook_test.csv` , which contains the test set.

The training set file provides a label for each node, with `True` indicating the node is "influential" and `False` otherwise. The test set file also has a column `label` , but the values are left as `None` for you to predict. The autograder will use another version of the test set file with the true labels to score your predictions.

Use any node attributes (such as the ones we explored in part 1) as features for this prediction and return the result as a dictionary of `<node, label>` , where `node` is a node that appears in the test set file.

Note:

1. Use sklearn Random Forest (https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) as your supervised model, any hyperparameters, and any node attributes as features.
2. Store your predictions in the dictionary `predict_labels` .
3. The autograder will measure the F1 score (`sklearn.metrics.f1_score`) of your precitios. In order to pass, your prediction results must achieve an F1 score higher than or equal to 0.85.

In [30]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
```

In [31]:
```
facebook = get_graph('facebook')
degree = nx.degree_centrality(facebook)
closeness = nx.closeness_centrality(facebook)
betweenness = nx.betweenness_centrality(facebook)
page = nx.pagerank(facebook, alpha = .9)
hub, auth = nx.hits(facebook)

train = pd.read_csv('assets/facebook_train.csv')
train['degree'] = train['node'].apply(lambda x: degree.get(x))
train['closeness'] = train['node'].apply(lambda x: closeness.get(x))
train['betweenness'] = train['node'].apply(lambda x: betweenness.get(x))
train['page'] = train['node'].apply(lambda x: page.get(x))
train['hub'] = train['node'].apply(lambda x: hub.get(x))
train['authority'] = train['node'].apply(lambda x: auth.get(x))

test = pd.read_csv('assets/facebook_test.csv')
test['degree'] = test['node'].apply(lambda x: degree.get(x))
test['closeness'] = test['node'].apply(lambda x: closeness.get(x))
test['betweenness'] = test['node'].apply(lambda x: betweenness.get(x))
test['page'] = test['node'].apply(lambda x: page.get(x))
test['hub'] = test['node'].apply(lambda x: hub.get(x))
test['authority'] = test['node'].apply(lambda x: auth.get(x))

X_train = train.iloc[:, 2:]
y_train = train.iloc[:, 1]
X_test = test.iloc[:, 2:]

forest = RandomForestClassifier(max_depth = 3, n_jobs = 1, random_state = 0).fit(X_train, y_train)
X_test['label'] = forest.predict(X_test)
predict_labels = dict(zip(test.node, X_test.label))
```

In [32]:
```
#hidden tests for Part 3 are within this cell
```

End

In [ ]:
```

```