Project 5: Network Analysis (CS 111 Law)

Date: Nov. 20, 2019 Due: 11:59 pm Monday, November 25
Version 1.1: Minor typo corrected (Chelsea Rider and Chesea. had been reversed.)

Learning Objectives: Doing network analysis of a social network using Python tools

Submit your assignment on Blackboard in a single .zip file called [your NetID]Project5.zip

In this homework you will extend the network analysis you did for this week's lab, and repeat it for a new network of your choice of data.

Overview

This lab involves more data analysis and exploration than software design, so we are going to ask you to turn in a zip file of several files:

- [your NetID]Project5.py that has the Python code you write in a file. You will want to have at least your import statements and the statement that loads the csv file with data into a dataframe, and perhaps more, in a .py file so you don't have to type them over and over at the command line
- [your NetID]Project5.txt that will have the answers to some questions we ask you, such as "Give the names and degrees of the 10 highest degree nodes"
- Several image files (e.g., .png) for several plots that you will create.

You will need the following modules:

- matplotlib.pyplot
- pandas
- networkx
- numpy

numpy, which stands for "numerical Python", is, among other things, a variation of the math module that knows how to work directly with 1 and 2 dimensional collections of numbers (e.g., a pandas series) instead of just the numbers itself. We will use it to take the log base 10 function of an entire pandas series at once, via numpy.log10.

You will want to look carefully at the example from the Lecture 23 lecture notes to see how to do some of these things.

Directions

The first bit of this is the lab, and then you will do some more.

Open the file we've provided. The provided file contains some non-ASCII characters (e.g., some Chinese), and as it happens, the encoding is ISO-8859-1. After you open the file, use the pandas read*csv method to get a dataframe from which we will create a graph data structure. (The file is comma separated, so you won't need a sep argument to read*csv.)

Put what the first 5 rows of the dataframe you get into the text file. (You'll use the pandas dataframe head() method we've used before, which gives 5 rows.)

Create an empty networkx graph called "g". Then create and add all the edges from the dataframe to that graph. If you called the dataframe edge_data, then the line of code that adds in edges from that dataframe is

```
g.add_edges_from(edge_data.values)
```

Note the .values!

Put the number of nodes and edges of g into your text file.

We saw in class how to get the degree data for a graph.

Do so, and put the names and degrees of the 10 highest-degree nodes into your text file.

You will need to use the Pandas sorting method .sort_values(), and by default it sorts from lowest to highest, so you will need to give it the argument ascending=False

And the head() method gives back 5 records by default, but can take an optional argument telling how many records to show. Thus, if you put your degree data into a Pandas series called degree_data, the code you will need is

```
degree_data.sort_values(ascending=False).head(10)
```

(FYI, the .sort_values() method does *not* change the series itself, but gives a new sorted series.)

Make two plots of the degree distribution, both histograms, and include them in your zip file:

1. A plot of the degree distribution, as we did in class. It should have the title "Degree Distribution (no scaling)", and x-axis labeled "Degree". Both this graph and the next one should have the y-axis labeled "Count".
2. You'll notice that the first histogram is very hard to read, because there are so many very low degree

> nodes that we can't see the higher degree nodes at all. This time we'll plot the distribution of log base 10 of the degrees, so we can see better what is going on. This graph should have the title "Distribution of log base 10 of degrees", and x-axis labeled "Log base 10 of degree".

To get a Pandas series of the log base 10 of the degree distribution, use

```
 numpy.log10(degree_data)
```

Notice that for the log 10 graph, an x-axis value of 0 means that the log of the degree is about 0, or the degree is about 1, an x-axis value of 0.5 means that the degree is about the square root of 10, or around 3 or 4, and an x-axis value of 1 means that the degree is about 10. (This graph is a little easier to read, but still not easy.)

Put in your text file the names and centrality values of the 10 most central nodes in the network.

Use the same networkx function degree_centrality() that we used in class; remember it's a dictionary that you might want to convert to another type.

Finally, put a line in your text file that gives us the degree cewntrality value for user James. (Just "James".)

New addtional stuff for Project Starts Here

s you now know, this graph is too large for us to see anything much if we plot it, and most of its vertices have very low degree.

We will get at the "core" of the graph by pruning off all the very low degree nodes. In particular we will look at the *3-core* of our graph (i.e., the k-core for *k* = 3). To obtain the k-core of a graph, we remove all the nodes of degree less than k, and then repeat in the new graph, until eventually we are left with a graph all of whose remaining nodes have degree at least k.

networkx has a k-core function, so we don't have to write this by ourselves.

To obtain our pruned graph:

```
g_cored = networkx.k_core(g, 3)
```

Put the number of nodes and edges of the cored graph into your text file.

Plot the degree distribution of the cored graph, and add that image to your zip file.

For this histogram, title the plot "3-cored graph degree distribution", title the x-axis "Degree" and the y-axis "Count".

*Create a cored graph **with label names for the nodes** as we discuss next and put that image into your zip file.*

The label for the node should be its name from the file, *if that name is at most 8 characters long.* If the name of the node is 9 characters long or longer, then use the first 7 letters followed by a "." character (to indicate an abbreviation).

Thus, the "Chelsea Rider" node should be labeled "Chelsea.", the "Predictive Analytics" node should be labeled "Predict.", but the "James" node should be labeled "James".

You will have to use dictionaries for this—creating an appropriate dictionary. Remember that the .nodes() method for Graphs gives you a list of all the names of the nodes of the graph. That is, for graph `g`, you get the list of all the names of the nodes from `g.nodes()`

Drawing a graph with node labels

The network.draw_networkx() method has a number of optional arguments.

To get labels: give the two optional arguments `with_labels=True` and `label=labels_dict`, where `labels_dict` is a Python dictionary whose keys are the node names of the graph (i.e., the elements of the list g.nodes() for graph g) and whose values are the desired node labels.

We suggest you also give the optional argument `font_size=8` to make the labels small since there are so many of them.

Directions: the Second Analysis

Pick any one graph of your choice from the Stanford Large Network Collection (called SNAP) database at https://snap.stanford.edu/data/ and repeat (almost) the entire analysis above for that graph.

Do not draw the cored version of your new graph if it has over 1024 nodes, because the resulting picture will take a long time to display and then be unreadable.

If the cored graph has at most 1024 nodes, do draw it, but draw it without any node labels at all, or whatever ones it has naturally.

Add the name of and a very short description of that graph and one at least mildly interesting qualitative observation about it to the end of your text file.

Advice about the second analysis

Don't choose a huge graph. On a typical consumer laptop, your analysis should work reasonably quickly on a graph with 90,000 or 200,000 edges. Even 1 million edges if you are willing to wait 10-20 seconds for things to happen and have a fairly new and powerful laptop. If you try a graph with 5 million edges, it will run really slowly (or not at all).

The input file may be a comma separated *or* a space separated csv file. If it's space separated, you will need to use the sep argument when you go to read the csv file.

The core operation will not work if the graph you picked happens to have self loops. ("Edges" that connect a node to itself.) To remove these from g, give the command

```
g.remove_edges_from(g.selfloop_edges())
```

Dealing with multiple plots (Repeated from lab)

In this lab you have to create more than one plot. You have a few options.

1. Assuming you are using Spyder, once you have saved out a plot you need to just kill the plot window with your mouse and keep going with the next plot.
2. Create separate figures, by issuing the statement plt.figure(1) before creating the first, plt.figure(2) before the second, and so on. This will let you have multiple plots up on your screen at once. If in doubt or difficulty, use method 1.