COSC 2671 Social Media and Network Analytics

Lab 8

Graph Introduction

Learning outcomes:

- Learn how to construct graphs in Python using networkx
- Visualise graphs

Requirements:

- A PC with Internet connection and Python installed (preference is version 3.X, but 2.7 is also okay, but unfortunately we don't have resources to provide support if there are version incompatibility issues)

Resources:

- This lab worksheet (available on Canvas)
- Associated code lab8Code.zip (available on Canvas)

Python Packages Required:

- networkx
- tweepy

Lab Introduction

In this lab, we will construct a graph, get familiar with networkx and visualise the graph.

Graphs & Networks

To familarise ourselves with networkx, one of the popular Python network analysing tools, we will construct graphs for a dataset we are familiar with – Twitter. In subsequent weeks we will introduce other graphs. For Twitter, we will move away from the tweet textual content and focus on the follower and followed by links between users. We will construct an egonet (this is social network analysis term for an ego-centric network). An egonet allows one to look at the friends/followers/associates a person has and can be visualised to provide a quick summary.

Twitter Egonet

To practise how to extract data, in this part of the lab you'll construct an egonet of yourself.

Open the provided file twitterGraph.ipynb. Part of it has been written for you, including obtaining the follower and followed links – please study it, as we'll use it in subsequent weeks as well you might need it for assignment 2.

You'll also need to copy your twitterClient.py (with your tokens and secrets) to your working directory.

First, we will use networkx to construct a follower+followed egonet, where the ego (you) is in the centre of the graph/network, and there is an inward directed edge from each follower to the ego (Twitter accounts who follow you), and an outward directed edge from the ego to each followed (accounts you follow). In addition, we store the follower count of each of the neighbours, as an indication of their popularity/importance. First, let's study how to construct a networkx graph.

Install networkx, with your favourite Python package manager.

Next, let's construct the egonet. From this week on, there will be less hand holding about where to include the code, as part of learning to code the notebooks. Here is a link to the documentation about networkx that you likely need to consult (https://networkx.github.io/documentation/stable/index.html).

First, we need to import the networkx package. We typically use the following, but you can name networkx with any abbreviation/alias apart from 'nx'. Enter it into a location you feel is appropriate:

```
import networkx as nx
```

Next, networkx has a few builtin graph types that should cover most graphs we want to construct. As following and follower relationships are directional, we will use the directed graph type, 'DiGraph'. We can construct an object representing this graph as follows:

```
egoGraph = nx.DiGraph()
```

The constructed graph is initially empty. We need to add some nodes and edges to it.

To add a node, we can use the following (remember, to explore other options and parameter settings, read the documentation):

```
egoGraph.add_node('A')
```

Which adds a node with id 'A'. Note the id could be an integer as well, but we use strings as it corresponds to Twitter usernames and easier to reference and remember.

To add a node with attributes, say 'followerCount', we can use the following:

```
egoGraph.add_node('B', followerCount=10)
```

Note we can associate with each node any number of attributes and names for those, e.g., weight, popularity etc. We just name it as an attribute within the add_node, e.g, add_node('C', weight=10, popular=True).

To add a directed edge form 'A' to 'B', use the following:

```
egoGraph.add_edge('A', 'B')
```

Exercise 1:
Using this introduction, construct an ego graph of yourself, of your followers and accounts you follow. Open twitterGraph.ipynb and then figure out where to add the necessary code with the supplied file (discuss with your lab demonstrator or neighbour if unsure, but there are some pointers in code to get you started). Please add the code to the appropriate locations.

After constructing the graph, we want to compute some basic statistics about the egonet, including:

- in and out degree of the ego user in our egoGraph
- in and out neighbours

Go to https://networkx.github.io/documentation/stable/reference/index.html. Find out from the documentation how to do the above (Hint: look up documentation about DiGraph). In addition, observe the extensive (and growing) functionality available. Have a read of the different functions and methods if interested, and in subsequent weeks we will use some of them to do further analysis.

Visualisation:
Networkx has a built-in visualiser, but it is relatively simple. Lets first try it.

A graph can be visualised using matplotlib, see
https://networkx.github.io/documentation/stable/reference/drawing.html

To draw your egonet, we can use the following:

```
nx.draw_networkx(egoGraph, arrows=True, with_labels=True,
pos=nx.kamada_kawai_layout(egoGraph, scale=10))
plt.axis('off')
plt.show()
```

The main thing to note is the pos=nx…. This is the graph layout algorithm, which specifies how the nodes should be positioned (if you remember in the graph introduction lecture, a question was asked about this). This is a whole field by itself and we won't go into too much details about it – for those interested, have a look at
https://en.wikipedia.org/wiki/Graph_drawing and some links I put up in canvas for the graph introduction module.

Remember to import matplotlib and pyplot.

```
import matplotlib.pyplot as plt
```

Visualise your egonet!

Optional:
The visualisation in networkx is limited, and typical practitioner would combine networkx with software dedicated to graph visualisation. There are several possibilities, but I would recommend gephi (https://gephi.org/) then igraph (http://igraph.org/redirect.html). Gephi is a Java based program, and among other things it can do visualisation. It can also do some graph analysis, but it cannot construct the graphs, nor in my opinion, have as extensive

functionality as networkx. Note, it requires separate installation, and although it should be relatively straight forward, but there may be some issues hence the tag optional. Follow the installation instructions on the website, it is relatively simple, but may require a Java JRE 7 or 8 installed, which may not be available on everyone's machine. Go to http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html to download and install if gephi install complains about JRE not been installed.

The other possibility is igraph, which has a direct Python interface. It is written in C++, and needs compiling when installing. Usually pip/anaconda can handle it (for Windows, can directly use an MSI installer), but with Macs sometimes the installation process cannot compile the C++ code, hence * am recommending gephi first. But if you can get it working, igraph is a decent library, and can do things directly in Python.

To use either, we first need to write out the file, in a format that either graph software can understand. Add the following code in appropriate location to save the file in graphml format. Graphml is one of the standard graph specification format, and can be directly loaded into gephi.

```
with open(graphFile, 'wb') as fOut:
    nx.write_graphml(egoGraph, fOut)
```

This will output your egonet and write it to graphFile – remember to define this variable in your notebooks. Open up gephi (or igraph) and load ego.graphml or the filename you saved it to. Try the following:

- try different layouts (for a small graph, shouldn't take too long, but might take a while for larger graphs)
- change the size or colour of nodes according to the number of followers