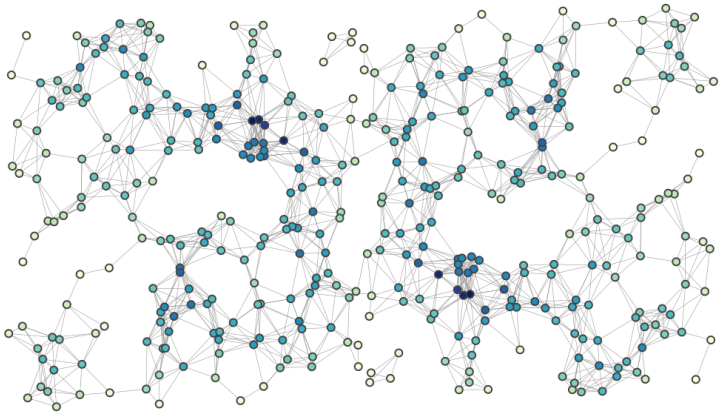


Old Dominion University

CS 432 Web Science

Assignment Six

Derek Goddeau

Professor

Michael L. Nelson

March 27, 2017

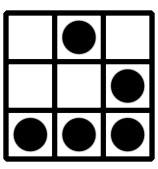
1 Use D3 to visualize Twitter followers

1.1 Getting the data

The data required to create the graph consists of the nodes and the links between them. The desired node data includes username, real name, user ID number and the users profile image. The link data that is needed consists of source nodes and the target nodes they are connected to. The python-twitter package is used to obtain the data, first using the `api.GetFollowers()` method to get a list of all of the target users followers as `User` objects, then creating a Python dictionary to hold the data.

```
data = { 'nodes' : [], 'links' : [] }
for user in users:
    user_data = {   'id': user.id,
                    'name': user.name,
                    'profile_image_url': user.profile_image_url,
                    'screen_name': user.screen_name,
                    'default_profile_image': user.default_profile_image
                }
    data['nodes'].append(dict(user_data))
```

Then to generate the link data a `friends` function to determine if two users follow the other can be wrapped up in a loop over all of the user objects. Because it uses the `api.ShowFriendship()` method it is rate limited to 180 calls per 15 minutes. The `python-twitter` API is instantiated with `sleep_on_rate_limit=True` so that will be the cause if this takes a long time. Use of `itertools.combinations()` ensures no wasted calls eat up more time than needed.

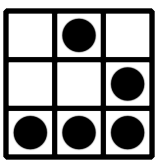


To calculate the time to run on a set of users where n is the number of users:

$$\frac{\left(\frac{\binom{n}{2}}{180}\right) \cdot 15}{60} = \text{Time in Hours}$$

Which works out to about 7 hours to generate links between 100 users and just over two weeks for 700 users. Because of this the data collected using the loop below was cut off after 48 hours of runtime which generated over 350 links between over 600 users. For completeness it is still running on a spare server with a uninterruptible power supply, and wrapped up in a catch-all-exceptions loop and the graph will be updated after completion.

```
for alice, bob in combinations(users, 2):
    friendship = friends(alice.screen_name, bob.screen_name)
    if friendship[0]: # alice following bob
        link = { 'source': alice.screen_name,
                 'target': bob.screen_name
               }
        data['links'].append(dict(link))
    if friendship[1]: # alice followed by bob
        link = { 'source': bob.screen_name,
                 'target': alice.screen_name
               }
        data['links'].append(dict(link))
```

1.2 Graphing the data

To graph the data I looked everywhere for anything to not write JavaScript or D3 and unfortunately while there are some python wrappers they are either old and have not been updated or they only support a minimal and very restrictive portion of D3. Fortunately `NetworkX` is capable of dumping a network graph in a D3 ready JSON format. Since Jupyter Noteboks support the `%%javascript` magic it was possible to dump the Python data to JSON and then move right on to JavaScript in the next block.

Having never used JavaScript for anything beyond standard website usage and the credit for the code goes to the internet in general and Mike Bostock who wrote the script that I modified. The key modification was changing the nodes from circles to the users profile images and resizing on mouseover. This only required adding the `"xlink:href"` attribute and `.on()` events and then updating the positions properly each tick.

The graph is quite large with even less than half of the dataset, when the full dataset is gathered it may be neccisary to choose some random x number of nodes to graph for perfomance purposes. This method of generating interactive graphs for the web, while nontrivial to learn, seems to have much more potential than other methods like Plotly. To view a live version of the graph click the Jupyter Notebook screenshot below.