CS532 Web Science: Assignment 6

Finished on March 17, 2016

Dr. Michael L. Nelson

Naina Sai Tipparti
ntippart@cs.odu.edu

Contents

Listings

List of Figures

Problem 1

Question

Use D3 to visualize your Twitter followers. Use my twitter account ("@phonedude_mln") if you do not have >= 50 followers. For example, @hvdsomp follows me, as does @mart1nkle1n. They also follow each other, so they would both have links to me and links to each other.

To see if two users follow each other, see:
`https://dev.twitter.com/rest/reference/get/friendships/show`

Attractiveness of the graph counts! Nodes should be labeled (avatar images are even better), and edge types (follows, following) should be marked.

Note: for getting GitHub to serve HTML (and other media types), see:
`http://stackoverflow.com/questions/6551446/can-i-run-html-files-directly-from-github-instead-of-just-viewing-their-source`

Be sure to include the URI(s) for your D3 graph in your report.

Answer

With the use of D3.js (a JavaScript library for manipulating documents based on data) [?], and the Force-Directed Graph template for directed graphs (`http://bl.ocks.org/mbostock/1153292`), following solution was achieved.

The solution for this problem is outlined by the following steps:

1. **Develop an python script to get the followers of followers:** In order to get the followers of followers across a variable degree, I derived the iterative python script as shown in Listing 1

```
import time
import tweepy
import json
import sys
import os
import re

consumer_key = "mvW9Y4uy8xJWMNwqV97qCa2aX"
consumer_secret = "vyCabSoD6CPXeAdL7onyEGO6lBl6YyPeivLpng1MgM2bOBjisU"
access_token = "798668178-bH8DbMpNuWkfhAHxuODgWSHwQE65B1WZnc4Ahtej"
access_token_secret = "FhykPKnQcgKQBE43os2bDZ31ugH9RVSG3HYoOL7QG7RNC"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)
screen_name='9ulovesu'

try:
    user = api.get_user(screen_name)
except tweepy.TweepError as e:
```

```
    if isinstance(e, tweepy.TweepError):
        print('RateLimitError')

    sys.exit(1)

print(user.screen_name,user.id)
ids = [f for f in user.followers_ids()]

ids.append(user.id)
user_friends = {}
print(user.followers_count, ids)

counter = 0
excluded = []

for id_ in ids:
    friend = api.get_user(id_)
    print(counter, friend.name, friend.profile_image_url, friend.id)
    print '-->'
    counter += 1
    user_connected_friends = []

    try:
        friend_followers = [f for f in api.get_user(friend.id).followers_ids()]
        for connected in friend_followers:
            if connected in ids:
                print(connected )
                print(",")
                user_connected_friends.append(connected)
        user_friends[friend.id] = {'name':friend.name, 'avatar':friend.
            profile_image_url, 'screen_name': friend.screen_name, 'followers_count':
            friend.followers_count, 'friends_count':friend.friends_count, '
            connected_to': user_connected_friends}
        print(user_friends[friend.id])
    except tweepy.TweepError as e:
        print('\n This is ---', e)
        if isinstance(e, tweepy.TweepError):
            time.sleep(60 * 15)
            try:
                friend_followers = [f for f in api.get_user(friend.id).followers_ids()
                    ]
                for connected in friend_followers:
                    if connected in ids:
                        print(connected)
                        print(",")
                        user_connected_friends.append(connected)
                user_friends[friend.id] = {'name':friend.name, 'avatar':friend.
                    profile_image_url, 'screen_name': friend.screen_name, '
                    followers_count': friend.followers_count, 'friends_count':friend.
                    friends_count, 'connected_to': user_connected_friends}
                print(user_friends[friend.id])
                continue

            except tweepy.TweepError as e:
                print(e)
                excluded.append(friend.id)
        else:
            print(e)
            excluded.append(friend.id)

with open("9ulovesu.json", 'w') as file:
    data = json.load(file)
```

Listing 1: Get followers of followers

2. **Visualize graph:** With the use of the graph template at `http://bl.ocks.org/`

`mbostock/1153292`, I replaced the default data in the JSON file with the output of followers as outlined in Listing **??**.

3. **Excluded Data** Values shown in Table 1 are followers of Naina Sai Tipparti Twitter's

ID	NAME	Screen-Name
152684352	Anudeep	anuhearthacker
2538627170	ÃĂmitha SophiaD'Souza	DSouzMitha
2433403314	GK Tecvision	GKTecvision

Table 1: Naina Sai Tipparti Twitter Followers Excluded Data

Account, but were not included because these accounts have a private settings and cannot be extracted without authentication.

```
            "id": 2755649263,
            "screen_name": "RithikaR9",
            "friends_count": 58,
            "connected_to": [798668178, 54493821, 118623489, 157985123],
            "followers_count": 5,
            "avatar": "http://abs.twimg.com/sticky/default_profile_images/
                default_profile_3_normal.png",
            "name": "Rithika Reddy"
        }, {
            "id": 54493821,
            "screen_name": "Manoj_Chandra11",
            "friends_count": 157,
            "connected_to": [2755649263, 1875320502, 147860404, 335970153, 798668178,
                709467001421500416, 118623489, 157985123],
            "followers_count": 54,
            "avatar": "http://pbs.twimg.com/profile_images/709387424477155328/
                HTm2wPZN_normal.jpg",
            "name": "manoj Kompalli"
        }, {
            "id": 118623489,
            "screen_name": "dineshpaladhi",
            "friends_count": 207,
            "connected_to": [798668178, 709467001421500416, 54493821, 2755649263,
                157985123],
            "followers_count": 61,
            "avatar": "http://abs.twimg.com/sticky/default_profile_images/
                default_profile_5_normal.png",
            "name": "dinesh kumar paladhi"
        }, {
            "id": 335970153,
            "screen_name": "abhipolavarapu",
            "friends_count": 21,
            "connected_to": [798668178, 54493821],
            "followers_count": 11,
            "avatar": "http://pbs.twimg.com/profile_images/532157901696028672/
                XltjrHmY_normal.jpeg",
            "name": "Abhishek Polavarapu"
        }, {
            "id": 709467001421500416,
```

Listing 2: Who follows who graph file label

The result of running followers.py as in Listing 1 and figure 1 can be seen at `http://www.cs.odu.edu/~ntippart/cs532/a6/followers.html`

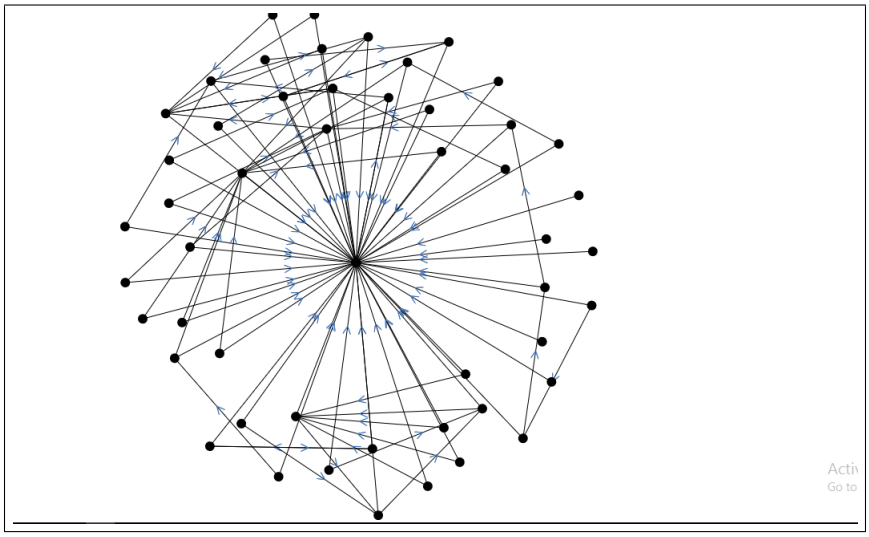

Figure 1: Naina Sai Tipparti Followers Network Graph

Problem 3

Question

Using D3, create a graph of the Karate club before and after the split.

- Weight the edges with the data from:
`http://vlado.fmf.uni-lj.si/pub/networks/data/ucinet/zachary.dat`

- Have the transition from before/after the split occur on a mouse click. This is a toggle, so the graph will go back and forth between connected and disconnected.

Answer

Using an example from the primary author of the D3 JavaScript library, Mike Bostok [?], a graph was created of Zachary's Karate Club using the pickled [?] dataset found at `http://nexus.igraph.org/api/dataset_info?id=1&format=html`. The D3 library provides a force-directed graphing layout [?], which was used to display the graph. A transition from the initial graph, shown in Figure 2, to the graph after the split of the karate club, shown in Figure 3, was created using standard JavaScript.

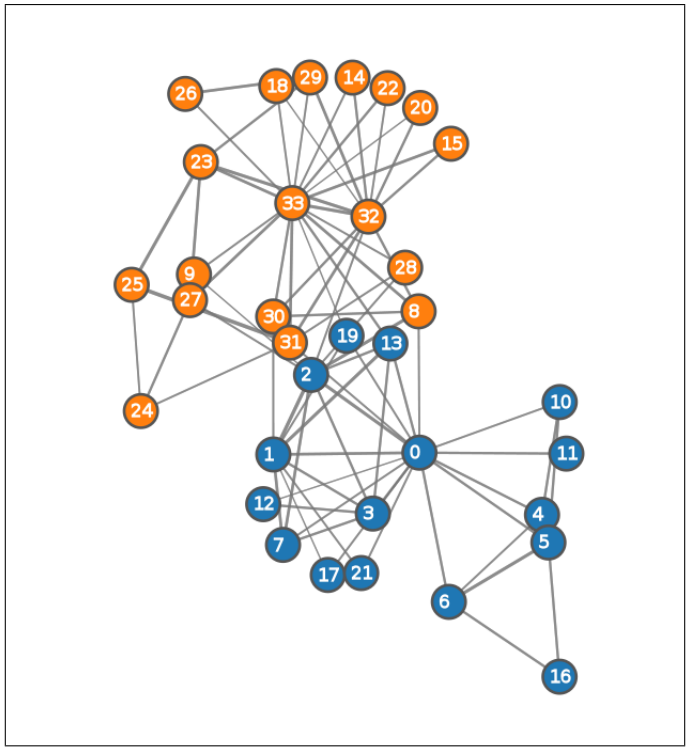

Figure 2: Initial Graph

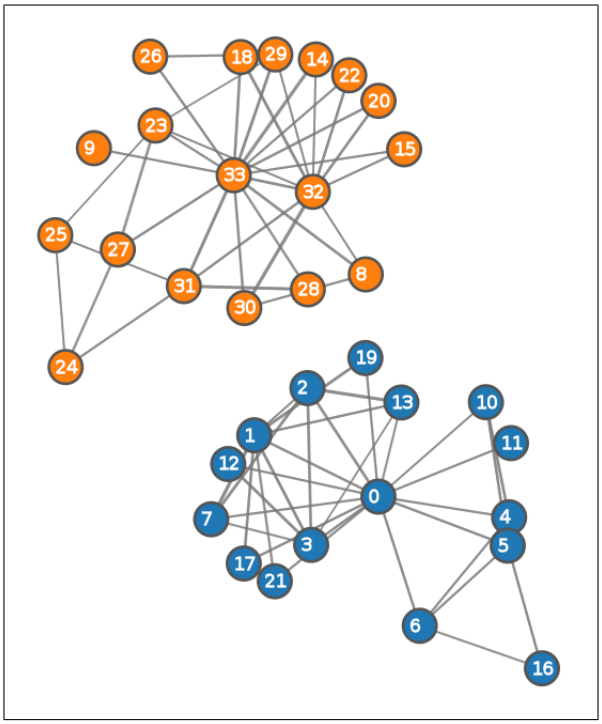

Figure 3: Graph After Split

The dataset was first parsed into matrices using the `build_matrix` function, shown in Listing 3. These matrices were converted into python dictionary objects, which are pickled [?] into the json format [?]. This output was used as the input for the JavaScript code, which uses the D3 library [?] to create the graphs.

The python code to produce the json data is shown in Listing 3.

```
#! /usr/bin/env python

import pickle
import json

def build_matrix(raw, n):
    """Builds a matrix from a list of strings where each string
    is a space-separated row of the matrix"""
    matrix = [[0 for y in range(n)] for x in range(n)]
    for idx, line in enumerate(raw):
        for idy, val in enumerate(line.split()):
            matrix[idx][idy] = int(val)
    return matrix

def build_nodes(clubs, names):
    """Builds a list of nodes represented as a python dict"""
    return [{"id": i, "club": c, "name": names[i]} for i, c in enumerate(clubs)]

def build_links(egraph, cgraph):
    """Builds a list of links represented as a python dict"""
    links = []
    for idx, line in enumerate(egraph):
        for idy, val in enumerate(line):
            if val == 1:
                links.append({"source": idx, "target": idy, "value": cgraph[idx][idy]})
    return links

if __name__ == '__main__':
    data = pickle.loads(open('karate.pickle').read())['karate']
    clubs = data.vs['Faction']
    names = data.vs['name']

    # Read lines from input data file
    lines = [line.strip() for line in open('karate.txt').readlines()]

    # parse size of nxn matrix
    n = int(lines[1].split()[0].replace('N=', ''))

    # build matrices using input data
    egraph = build_matrix(lines[7:41], n)
    cgraph = build_matrix(lines[41:], n)

    m = {}
    m['nodes'] = build_nodes(clubs, names)
    m['links'] = build_links(egraph, cgraph)
    with open('out.json', 'w') as output:
        output.write(json.dumps(m, indent=1, separators=(',', ': ')))
```

Listing 3: Data Converter

The JavaScript code to produce the graph is shown in Listing 4.

```
var width = 960,
    height = 700;

var color = d3.scale.category10();

var svg = d3.select("body").append("svg")
    .attr("width", width)
    .attr("height", height);

var force = d3.layout.force()
    .charge(-400)
    .gravity(0.1)
    .linkDistance(90)
    .size([width, height]);

d3.json("out.json", function(error, graph) {
    var link = svg.selectAll(".link")
        .data(graph.links)
        .enter().append("line")
        .attr("class", "link")
        .style("stroke-width", function(d) { return Math.sqrt(d.value); });

    var node = svg.selectAll(".node")
        .data(graph.nodes)
        .enter().append("g")
        .attr("class", "node")
        .call(force.drag);

    node.append("circle")
        .attr("r", 12)
        .style("fill", function(d) { return color(d.club); });

    node.append("text")
        .attr("dx", -7)
        .attr("dy", ".35em")
        .text(function(d) { return d.id; });

    function update() {
        node.data(graph.nodes)
            .exit().remove();
        link.data(graph.links)
            .exit().remove();
        force.nodes(graph.nodes)
            .links(graph.links)
            .on("tick", tick)
            .start();
    };

    function tick() {
        link.attr("x1", function(d) { return d.source.x; })
            .attr("y1", function(d) { return d.source.y; })
            .attr("x2", function(d) { return d.target.x; })
            .attr("y2", function(d) { return d.target.y; });

        node.attr("transform", function(d) {
            return "translate(" + d.x + "," + d.y + ")";
        });
    };

    var done = false;
    svg.on("click", function(d) {
        if(!done) {
            done = true;
            graph.links = graph.links.filter(function(d) {
                return graph.nodes[d.source.id].club == graph.nodes[d.target.id].club;
            });
```

```
                update();
            }
        });
        update();
    });
```

Listing 4: Building the Graph