Assignment 7

CS 595: Introduction to Web Science

Fall 2013

Shawn M. Jones

Finished on November 7, 2013

1

Question

1. 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.

Answer

The Karate club data has been rendered into graphs using D3 as shown in Figures 1 and 2. On load, the graph looks like Figure 1 and then looks like Figure 2 after a single click. From that point, further clicks will toggle between the "together" Karate Club and the "split" Karate Club. A live version of this can be experienced at: `http://www.cs.odu.edu/~sjone/courses/cs595-ws-f13/graph.html`.

The code for generating the graph was shamelessly stolen from the D3 example at `http://bl.ocks.org/mbostock/950642` and is shown in Listing 1. Of course, modifications were made to suit the needs of the assignment.

From Listing 1, the function `loadgraph` on line 42 does most of the work. Line 43 shows the overall title of the graph being rendered. Line 49 starts the loading of the appropriate JSON file and subsequent actions on that data.

The edges are weighted, changing the edge distances using the `linkDistance` and `linkStrength` methods of the selection on lines 53 and 54.

Line 47 loads all of the edge data (referenced as *links*) and line 63 loads all of the node data, creating `<g>` tags with a class of `node` [2]. Line 69 inserts the image of the stick figures in for each person inside the corresponding `<g>` tags. Line 76 appends the text labels to each node inside the corresponding `<g>` tags. Line 81 actually draws the graph.

Line 104 sets up the canvas for the SVG code to populate, and line 109 starts the *force-directed layout*, which maps nodes to SVN `<image>` elements and links to `<line>` elements[1]. Line 116 runs `loadgraph` for the first time with the "together" club graph JSON file.

All of that gets the graph drawn for the first time. The click response is handled by the method referenced on line 107. On a `click` event, the `switchgraph` function from line 27 is executed, which clears the existing graph and toggles the state, then calls `loadgraph` with the datafile and label associated with the set state. The statement on line 35 allows the state to be toggled to the values defined on lines 96-100.

Of course, this just discusses the HTML/JavaScript/D3 part of the assignment. The data set used was not in the JSON format, and hence had to be converted for use in D3. Not wanting to use JavaScript to parse the file, I felt that this was a job for Python.

Listing 2 shows the code that takes the given "matrix" dataset and converts it into JSON. It is run like so:

```
./convertdata.py | python -mjson.tool > club.json
```

The `python -mjson.tool` pretty-formats the JSON code for human consumption.

Listing 3 shows the code that takes the JSON produced by `convertdata.py` and runs the Girvan-Newman algorithm over it, splitting the graph into two clusters. It is run like so:

```
./createClubSplit.py club.json > split-club.json
```

This time, I natively pretty-formatted the output with line 35.

Thus, the Python programs `convertdata.py` and `createClubSplit.py` create the JSON files that are consumed by the JavaScript in `graph.html`.

Again, a live version of this can be experienced at: `http://www.cs.odu.edu/~sjone/courses/cs595-ws-f13/graph.html`.

Figure 1: Screenshot of Karate Club Graph Before Split Drawn in D3

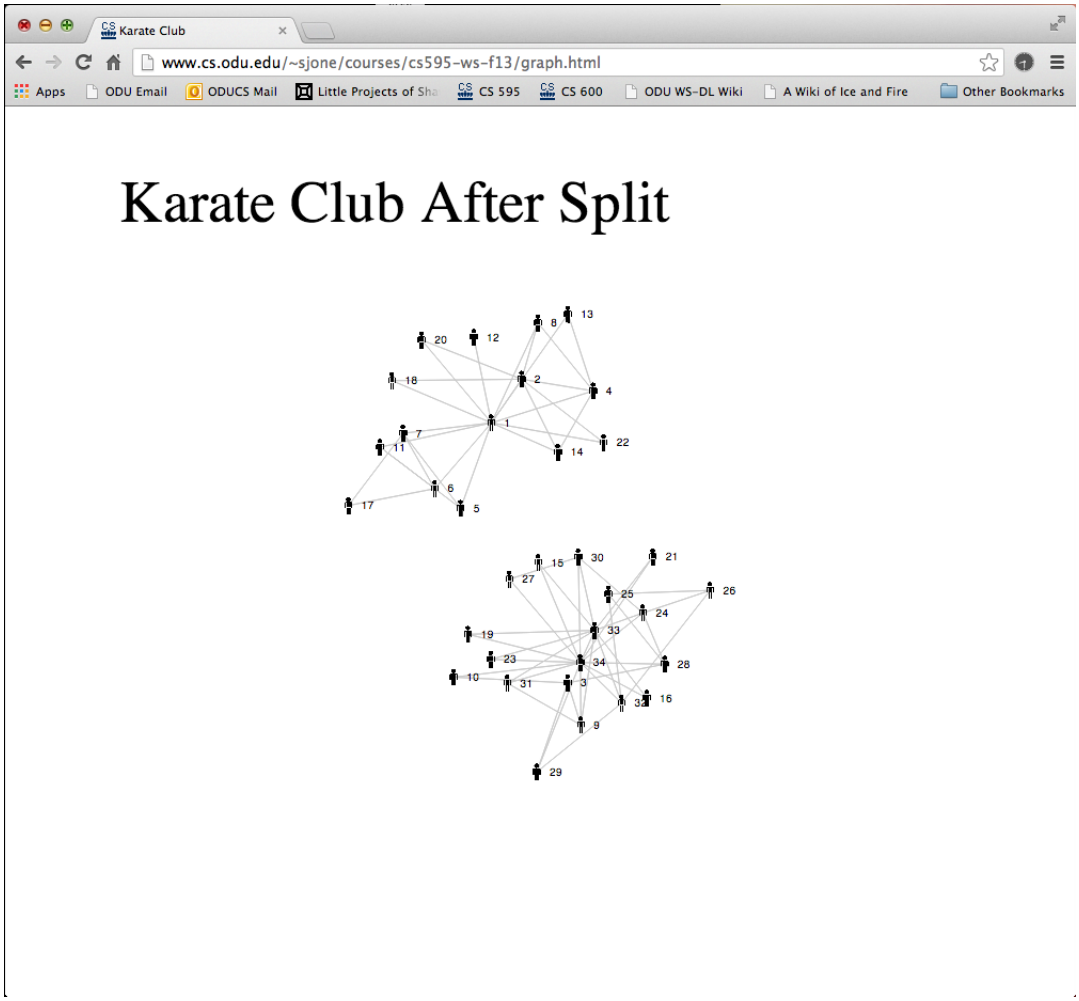

Figure 2: Screenshot of Karate Club Graph Split Drawn in D3

```
<!DOCTYPE html>
<html>
  <head>
    <!-- somehow needed for d3.layout.force to work -->
    <meta http-equiv="Content-Type" content="text/html;charset=
        utf-8">
    <title>Karate Club</title>
    <style>
.link {
  stroke: #ccc;
}

.node text {
  pointer-events: none;
  font: 10px sans-serif;
}

    </style>
  </head>
  <body>
    <script type="text/javascript" src="d3/d3.v3.js"></script>
    <script type="text/javascript">
/*
    Shamelessly stolen/inspired by:
    http://bl.ocks.org/mbostock/950642
*/

function switchgraph(d) {

    // clear the existing graph
    d3.selectAll(".node").remove();
    d3.selectAll(".link").remove();
    d3.selectAll("text").remove();

    // toggle state between together and split
    state = 1 - state;

    // load the split graph
    loadgraph(datafiles[state], labels[state]);

}

function loadgraph(filename, label) {
    svg.append("text")
        .attr("font-size", 55)
        .attr("x", 100)
        .attr("y", 100)
        .text(label);
```

```
48
49      d3.json(filename, function(error, json) {
50        force
51            .nodes(json.nodes)
52            .links(json.links)
53            .linkDistance( function(d) { return (d.weight * 20) }
                  )
54            .linkStrength( function(d) { return (d.weight / 7) } )
55            .start();
56
57        var link = svg.selectAll(".link")
58            .data(json.links)
59            .enter()
60            .append("line")
61            .attr("class", "link");
62
63        var node = svg.selectAll(".node")
64            .data(json.nodes)
65            .enter().append("g")
66            .attr("class", "node")
67            .call(force.drag);
68
69        node.append("image")
70            .attr("xlink:href", "http://www.i2clipart.com/cliparts
                  /b/a/4/c/clipart-stick-figure-male-256x256-ba4c.png
                  ")
71            .attr("x", -8)
72            .attr("y", -8)
73            .attr("width", 16)
74            .attr("height", 16);
75
76        node.append("text")
77            .attr("dx", 12)
78            .attr("dy", ".35em")
79            .text(function(d) { return d.id });
80
81        force.on("tick", function() {
82          link.attr("x1", function(d) { return d.source.x; })
83              .attr("y1", function(d) { return d.source.y; })
84              .attr("x2", function(d) { return d.target.x; })
85              .attr("y2", function(d) { return d.target.y; });
86
87          node.attr("transform", function(d) { return "translate("
                + d.x + "," + d.y + ")"; });
88        });
89
90  });
91  }
92
```

```
93  var width = 960,
94      height = 800;
95
96  var datafiles = new Array();
97  datafiles[0] = "club.json";
98  datafiles[1] = "split-club.json";
99
100 var labels = new Array();
101 labels[0] = "Karate Club Prior To Split";
102 labels[1] = "Karate Club After Split";
103
104 var svg = d3.select("body").append("svg")
105     .attr("width", width)
106     .attr("height", height)
107     .on("click", switchgraph );
108
109 var force = d3.layout.force()
110     .gravity(.05)
111     .charge(-100)
112     .size([width, height]);
113
114 // initialize state to "together"
115 var state = 0;
116 loadgraph(datafiles[state], labels[state]);
117
118     </script>
119   </body>
120
121 </html>
```

Listing 1: HTML/JavaScript code that displays the graphs shown in the screenshots from Figures 1 and 2

```
#!/usr/local/bin/python3

import json

f = open("zachary.dat")
inputlines = f.readlines()
f.close()

outputDict = { "nodes" : [], "links" : [] }

nodeCounter = 0

# skip the unnecessary lines
# the header information (lines 0 - 6) isn't useful to us
# the matrix from lines 7 to 34 is just the mapping of
# connections, with no weights
for row in inputlines[7 + 34:]:
    name = nodeCounter + 1

    newNode = { 'id' : str(name) }

    outputDict['nodes'].append( newNode )

    columns = row.split()

    for j in range(len(columns)):

        if columns[j] != "0":

            weight = int(columns[j])
            source = nodeCounter
            target = j

            newLink = \
                { "source" : source, "target" : target, "weight"
                    : weight }

            outputDict['links'].append( newLink )

    nodeCounter += 1

print(json.dumps(outputDict))
```

Listing 2: Python code that converts the given matrix data file into JSON for the initial "together" view of the Karate Club that is used for the graph shown in Figure 1

```
#!/usr/local/bin/python3

import sys
import json
import networkx
from networkx.readwrite import json_graph


if __name__ == "__main__":
    inputfile = sys.argv[1]

    f = open(inputfile)
    inputdata = json.load(f)
    f.close()

    club = json_graph.node_link_graph(inputdata)

    # 4. repeat until we have 2 clusters
    while (networkx.number_connected_components(club) < 2):
        # 1. calculate edge-betweenness for all edges
        # 3. recalculate betweenness
        eb = networkx.edge_betweenness_centrality(club, weight='
            weight')

        # 2. remove the edge with highest betweenness
        # Thanks Stack Overflow:
        # http://stackoverflow.com/questions/16772071/sort-dict-
            by-value-python
        edge2remove = sorted(eb.items(), key=lambda x:x[1],
            reverse=True)[0][0]

        club.remove_edge(*edge2remove)

    # club should be split, sadly... :(

    outputdata = json_graph.node_link_data(club)

    print(json.dumps(outputdata, indent=4))
```

Listing 3: Python code that takes in the code produced by Listing 2, runs the Girvan-Newman algorithm on it, and then produces a JSON file showing the split Karate Club to be used by the graph shown in Figure 2

2

Question

2. Use D3 to create a who-follows-whom graph of your Twitter account. Use my twitter account ("@phonedude_mln") if you do not have an interesting number of followers.

Answer

Not attempted

References

[1] Bostock, M. Force layout - mbostock/d3 wiki - github. https://github.com/mbostock/d3/wiki/Force-Layout, Oct. 2013.

[2] Bostock, M. Selections - mbostock/d3 wiki - github. https://github.com/mbostock/d3/wiki/Selections, Nov. 2013.