Introduction to Web Science/595: Assignment #6

Dr. Nelson

Francis Pruter

Thursday, 06 October, 2014

Contents

Problem 1

```
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.
```

SOLUTION

Below is the code used to make the graphML and the matrix to JSON format that D3 will use. [2] Orginially, I just used the graphML without the weights provided by the vlado.fmf.uni-li.si website. [3] I predict the reason is because the karate.graphML is is undirected while the d3.layout.force is a directed graph. The issue I encountered was when I split the graph on the mouse click, some nodes would completely detach from the graph.

Listing 1: graphML2JSON.py

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

"""
Data file from:
http://igraph.org/python/doc/tutorial/tutorial.html

ref:
http://nbviewer.ipython.org/github/davidrpugh/cookbook-code/
   blob/master/notebooks/chapter06_viz/04_d3.ipynb
"""

from igraph import *
import numpy

karate = Graph.Read_GraphML("karate.GraphML")

layout=karate.layout('kk')
karate.vs["label"] = karate.vs["name"]

with open("graph.json", "w") as f:
  f.write('{\n')
  f.write('  "nodes": [\n')

  for x in range(0,33):
    f.write('   {\n')
    f.write('    "Faction": ' + str(karate.vs['Faction'][x]) + ', ')
    f.write('    "id": ' + str(karate.vs['id'][x]).lstrip('n') + ', ')
    f.write('    "name": "' + str(karate.vs['name'][x]) + '"\n')
    f.write('   },\n')

  f.write('   {\n')
  f.write('    "Faction": ' + str(karate.vs['Faction'][33]) + ', ')
  f.write('    "id": ' + str(karate.vs['id'][33]).lstrip('n') + ', ')
  f.write('    "name": "' + str(karate.vs['name'][33]) + '"\n')
```

Problem 1 continued on next page. . .

```
  f.write('  }\n')
  f.write(' ],\n')
  f.write(' "links": [\n')

  weight = numpy.loadtxt("karatematrix.dat")

  xloc = 0
  for x in weight:
    yloc = 0
    for y in x:
      if (y == 0):
        yloc = yloc+1
        continue

      f.write('  {\n')
      f.write('   "source": ' + str(xloc) + ',')
      f.write('   "target": ' + str(yloc) + ', ')
      f.write('   "weight": ' + str(y) + '\n')
      f.write('  },\n')
      yloc = yloc+1
    xloc = xloc+1
  f.write(' ]\n}')
```

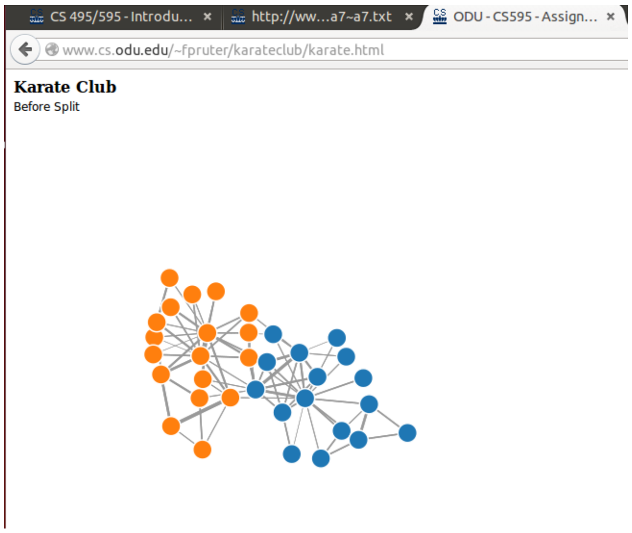
Below is the HTML code. It is based off of David R. Pugh example. URI: http://www.cs.odu.edu/~fpruter/karateclub/karate.html [2] [1]

Listing 2: karate.html

```
<!DOCTYPE html>
<html>
<style>
.node {stroke: #fff; stroke-width: 1.5px;}
.link {stroke: #999; stroke-width}
text { stroke: #fff; stroke-width: 1px; font: 12px sans-serif; pointer-events: none;}
</style>
<head>
     <title>ODU - CS595 - Assign7 - Karate Club Graph</title>
</head>
<body>
    <b> Karate Club </b> <br>
    <text class=split>Before Split</text>
<!--      <script src="d3.min.js"></script> -->
    <!DOCTYPE html>
<meta charset="utf-8">
<div id="karate"></div>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script>

    var width = 500,
        height = 500;

    // We create a color scale.
    var color = d3.scale.category10();

    // We create a force-directed dynamic graph layout.
    var force = d3.layout.force()
        .charge(-320)
        .linkDistance(40)
        .gravity(.2)
        .size([width, height]);

    var toggle = false;

    // In the <div> element, we create a <svg> graphic
    // that will contain our interactive visualization.
    var svg = d3.select("#karate").select("svg")
    if (svg.empty()) {
        svg = d3.select("#karate").append("svg")
                    .attr("width", width)
                    .attr("height", height);
    }

    // We load the JSON file.
    d3.json("graph.json", function(error, graph) {
        // In this block, the file has been loaded
        // and the 'graph' object contains our graph.

        // We load the nodes and links in the force-directed
        // graph.
        graphRec=JSON.parse(JSON.stringify(graph));
        force.nodes(graph.nodes)
            .links(graph.links)
            .start();

        // We create a <line> SVG element for each link
        // in the graph.
        var link = svg.selectAll(".link")
            .data(graph.links)
            .enter().append("line")
            .attr("class", "link")
            //modified to change width based on weight
            .attr("stroke-width", function(d) {
               return d.weight/2
            });

        // We create a <circle> SVG element for each node
        // in the graph, and we specify a few attributes.
        var node = svg.selectAll(".node")
            .data(graph.nodes)
            .enter().append("circle")
            .attr("class", "node")
            .attr("r", 10)  // radius
            .style("fill",  function(d){
                    return color(d.Faction);})
            .call(force.drag);

        // The name of each node is the node number.
        node.append("title")
```

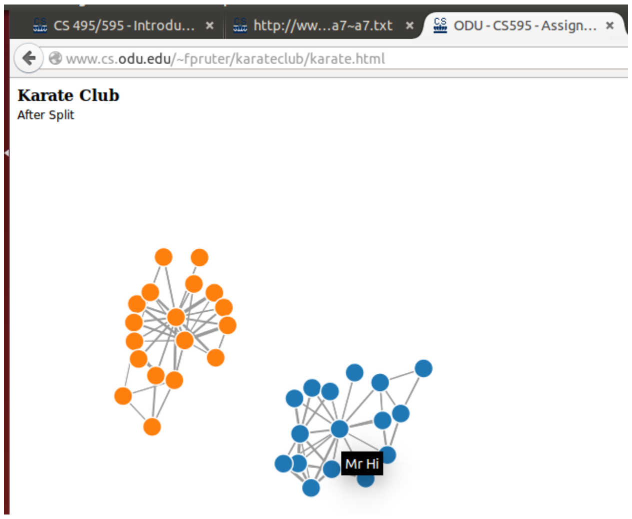
```
            .text(function(d) { return d.name; });

        // We bind the positions of the SVG elements
        // to the positions of the dynamic force-directed graph,
        // at each time step.
        force.on("tick", function() {
            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("cx", function(d) { return d.x; })
                .attr("cy", function(d) { return d.y; });
        });

        svg.on("click", function(d){
          if (toggle == false) {
               toggle = true;
               split();
               var text = d3.select(".split").text("After Split")
               svg.selectAll(".node").style("fill", function(d){
                    return color(d.Faction);
               });
          }
        });

        //adjust threshold
          function split() {
               for (var i = graphRec.links.length -1; i > -1; i--){
                    var src = (graphRec.links[i].source)
                    var tar = (graphRec.links[i].target)

                    var tarfac = 0
                    var srcfac = 0

                    for (x = 0; x < graphRec.nodes.length; x++){
                          if (graphRec.nodes[x].id == src) { srcfac = graphRec.nodes[x
                              ].Faction;}
                          if (graphRec.nodes[x].id == tar) { tarfac = graphRec.nodes[x
                              ].Faction;}
                    }

                    if (srcfac != tarfac) {
                         graph.links.splice(i, 1);
                    }
               }
              restart();
          };
          //Restart the visualisation after any node and link changes
          function restart() {
               link = link.data(graph.links);
               link.exit().remove();
               link.enter().insert("line", ".node").attr("class", "link");
```

```
                force.start();
            };
    });
</script>
</body>
</html>
```



Karate Club
Before Split

Figure 1: Karate Club Before Split



Karate Club
After Split

Figure 2: Karate Club After Split

References

[1] python-igraph Manual. http://nbviewer.ipython.org/github/davidrpugh/cookbook-code/blob/master/notebooks/chapter06$_viz/04_d3.ipynb$.

[2] StackOverFlow: load csv into 2D matrix with numpy for plotting. http://stackoverflow.com/questions/4315506/load-csv-into-2d-matrix-with-numpy-for-plotting, 2010.

[3] David R. Pugh. Method: Data visualization with d3.js and python - part 3. http://blog.nextgenetics.net/?e=32, 2012.