CS 595: Assignment 7

Mallika Kogatam

Fall 2014

Contents

1 Problem 1

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.

1.1 Answer

1. I did a little research on D3 programming, I realized that the input for the d3 program should be JSON format. So I started my assignment by converting the Zachary matrix to JSON text.
2. First I wrote a piece of python code shown in Listing 1 to convert Zachary matrix to JSON text which consists of 'target','source' and 'weight'.
3. The result which is computed from the Listing 1 is used to generate a graph for the karate club with out any split in the group. But JSON text is needed to get the graph of karate club after the split occurred.
4. So to get the graph with two clusters for Zachary karate club data I wrote an other python code shown in Listing 2 which is based on Girvan-Newman Algorithm.
5. I used the same logic which I used in the Assignment 6 to get 2 clusters. But I wrote Assignment 6 in R but now as I need the JSON text I chose python to finish the job.
6. The library "networkx" made my life easy to split the graph into 2 cluseters as it has many predefined function which are required to implement Girvan-Newman Algorithm.
7. Now I used the output generated from both the python codes Listing 1 and 2 in d3 code to get the graphs shown in Figure 1 and Figure 2
8. On loading the page the graph looks like Figure1 and then looks like Figure2 after a single click.
9. 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/~mkogatam/fall14/cs595/A7/karateclubd3.html`.
10. The code for the generating the graphs shown in Figures 1 and 2 is written based on the D3 example at `http://bl.ocks.org/mbostock/950642` and `http://bl.ocks.org/mbostock/2706022`.
11. I have written enough comments in the code Listing 3 which clearly explains the functionality of the code.

```python
#!/usr/local/bin/python
import json

f = open("zachary.dat")
inputlines = f.readlines()
f.close()
karatejson = {"links" : [],"nodes" : []}
counter = 0
for row in inputlines[7 + 34:]:
    columns = row.split()
    node = counter + 1
    #In order draw graph in d3 we need both nodes and links so nodes are computed.
    newNode = { 'id' : str(node) }

    karatejson['nodes'].append( newNode )
    for col in range(len(columns)):
        #compute the links which have edges other wise skip it.
        if columns[col] != "0":
            source = node
            target = col + 1
            weight = int(columns[col])

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

            karatejson['links'].append( newLink )
    counter += 1

print(json.dumps(karatejson))
```

Listing 1: 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

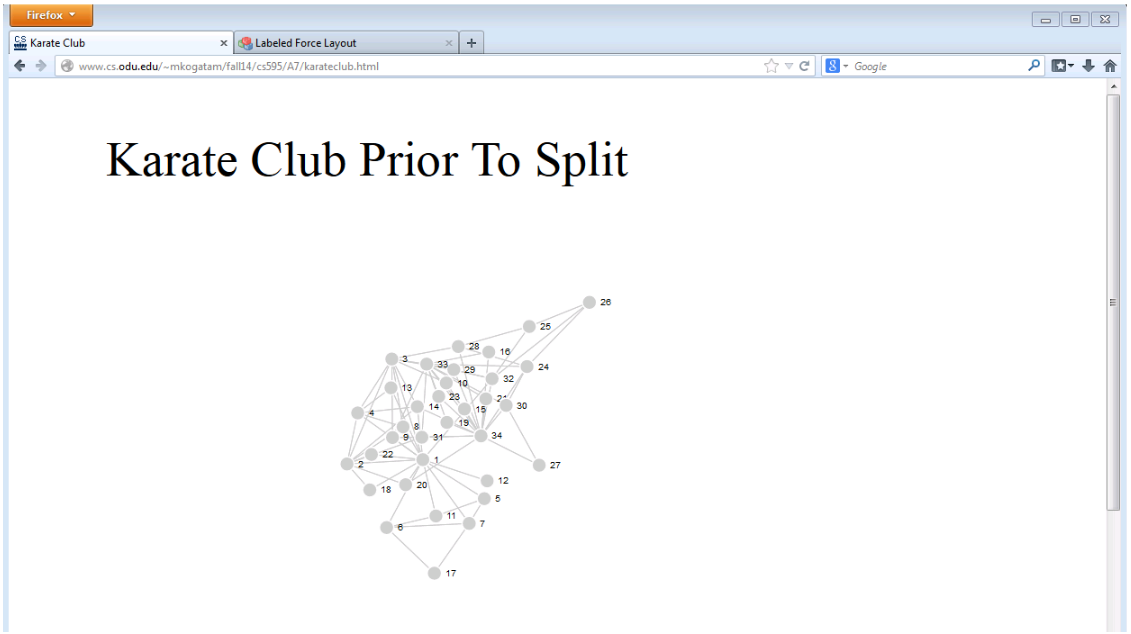
```python
#!/usr/local/bin/python

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

f = open("club.json")
inputlines = json.load(f)
f.close()

club = json_graph.node_link_graph(inputlines)

while (networkx.number_connected_components(club) < 2):
    eb = networkx.edge_betweenness_centrality(club, weight = 'weight')
    edgeRemove = sorted(eb.items(), key=lambda x:x[1], reverse=True)[0][0]
    club.remove_edge(*edgeRemove)

output = json_graph.node_link_data(club)

print(json.dumps(output))
```

Listing 2: Python code that takes in the code produced by Listing 1, 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

```html
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html;charset=utf-8">
    <title>Karate Club</title>
    <style>
        .link {
          stroke: #ccc;
        }
        .node circle {
          fill: #ccc;
          stroke: #fff;
          stroke-width: 1.5px;
        }
        .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">

        function switchgraph(d) {

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

            // toggle state between together and split
            //if the state is 0 then the graph before the split is shown and if
            //its 1 then the graph after the split is shown
            state = 1 - state;

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

        }
        // this function does the whole job.its reads the JSON file with respective to the state
        function loadgraph(filename, label) {
            svg.append("text")
                .attr("font-size", 55)
                .attr("x", 100)
                .attr("y", 100)
                .text(label);
            //reads the data from the JSON file that is passed from the previous funtion
            d3.json(filename, function(error, json) {
              force
                  .nodes(json.nodes)
                  .links(json.links)
                  //The edges are weighted, changing the edge distances using these methods
                  .linkDistance( function(d) { return (d.weight * 20) } )
                  .linkStrength( function(d) { return (d.weight / 7) } )
                  .start();

              var link = svg.selectAll(".link")
                  .data(json.links)
                  .enter()
                  .append("line")
                  .attr("class", "link");

              var node = svg.selectAll(".node")
                  .data(json.nodes)
                  .enter().append("g")
                  .attr("class", "node")
                  .on("mouseover", mouseover)//this highlights the circle when mouse is over the circle
                  .on("mouseout", mouseout)
```

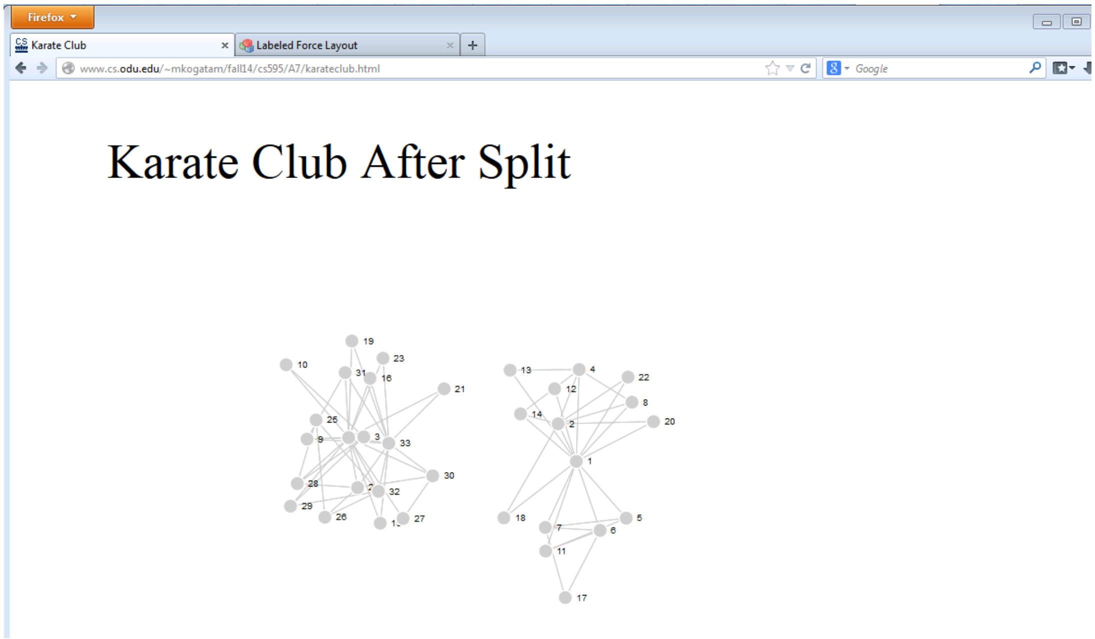
```
                  .call(force.drag);

              node.append(”circle”)//the node is defined to be a circle
                .attr(”r”, 8);

              node.append(”text”)//the text for the node is defined here
                  .attr(”dx”, 12)
                  .attr(”dy”, ”.35em”)
                  .text(function(d) { return d.id });

              // this function draws the graph
              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(”transform”, function(d) { return ”translate(” + d.x + ”,” + d.y + ”)”; });
              });

        });

        links.forEach(function(link) {
          link.source = nodes[link.source] || (nodes[link.source] = {name: link.source});
          link.target = nodes[link.target] || (nodes[link.target] = {name: link.target});
        });

        function mouseover() {
          d3.select(this).select(”circle”).transition()
              .duration(750)
              .attr(”r”, 16);
        }

        function mouseout() {
          d3.select(this).select(”circle”).transition()
              .duration(750)
              .attr(”r”, 8);
        }
        }
        var width = 960,
            height = 800;
        //assigning the files to an array so that they can be called when ever required and depending
        //on the state of the graph
        var datafiles = new Array();
        datafiles[0] = ”karate.json”;
        datafiles[1] = ”karatesplit.json”;
        // Gives the appropriate headings depending on the state of the graph.
        var labels = new Array();
        labels[0] = ”Karate Club Prior To Split”;
        labels[1] = ”Karate Club After Split”;
        var svg = d3.select(”body”).append(”svg”)
            .attr(”width”, width)
            .attr(”height”, height)
            .on(”click”, switchgraph );// this is where the function switchgraph is called on click

        // this line forces the layout to appear.
        var force = d3.layout.force()
            .gravity(.05)
            .charge(-100)
            .size([width, height]);

        // initialize state to ”together”
        var state = 0;
        loadgraph(datafiles[state], labels[state]);
    </script>
  </body>
</html>
```

Listing 3: HTML/JavaScript code that displays the graphs shown in the screenshots from Figures 1 and 2



Karate Club Prior To Split

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



Karate Club After Split

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

Bibliography

[1] Force directed graph with mouseover. http://bl.ocks.org/mbostock/2706022.

[2] Labeled force layout. http://bl.ocks.org/mbostock/950642.

[3] Sort dictionary by value. http://stackoverflow.com/questions/16772071/sort-dict-by-value-python.

[4] Cyrille Rossant. *IPython CookBook*, chapter 6.4, pages 223–238. Packt Publishers, 2014.

[5] Sphinx theme. Networkx documentation, november 2014.