Introduction to Web Science: Assignment #7

Dr. Nelson

Alexander Nwala

Thursday, November 6, 2014

Contents

Problem 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 1

With the use of D3.js (a JavaScript library for manipulating documents based on data) [2], and the Force-Directed Graph template for undirected graphs (http://bl.ocks.org/mbostock/4062045), the following solution was achieved.

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

1. **Convert the Karate club graph to json:** With the use of BeautifulSoup [1], as outlined in Listing 1, the Karate club graph, karate.GraphML was converted to karateClubGraph.json.

Listing 1: Convert karate.GraphML to karateClubGraph.json

```
#convert Karate club GraphML file to a json file
if( len(parentNodes) > 0 ):

     for i in range(0, len(parentNodes)):
          factionAndNodeName = parentNodes[i].findAll('data')
          faction = factionAndNodeName[0].text
          nodeName = factionAndNodeName[1].text

          stringToWrite = '\t {"name": "' + nodeName + '", "faction": ' + faction
              + ', "color": 1 },' + '\n'
          #remove comma
          if( i == len(parentNodes)-1 ):
               stringToWrite = '\t {"name": "' + nodeName + '", "faction": ' +
                   faction + ', "color": 1 }' + '\n'

          outputFile.write(stringToWrite)


     outputFile.write('\t],' + '\n')
     outputFile.write('\t"links":' + '\n')
     outputFile.write('\t[' + '\n')

parentEdges = soup.findAll('edge')
if( len(parentEdges) > 0 ):
     for i in range(0, len(parentEdges)):

          edgeWeight = parentEdges[i].find('data')
          #print edgeWeight.text

          #data = parentEdges[i].find('edge')
```

```
            sourceTargetDate = str(parentEdges[i])

            indexOfStart = sourceTargetDate.find('source="')
            indexOfEnd = sourceTargetDate.find('>')
            sourceTargetDate = sourceTargetDate[indexOfStart:indexOfEnd]

            sourceTargetDate = sourceTargetDate.split('"')
            sourceData = sourceTargetDate[1][1:]
            targetData = sourceTargetDate[3][1:]

            stringToWrite = '\t {"source": ' + sourceData + ', "target": ' +
                targetData + ', "weight": ' + edgeWeight.text + ', "id": "e' + str(i
                +1) + '" },\n'

            if( i == len(parentEdges)-1 ):
                 stringToWrite = '\t {"source": ' + sourceData + ', "target": ' +
                     targetData + ', "weight": ' + edgeWeight.text + ', "id": "e' +
                     str(i+1) + '" }\n'
...
```

2. **Color code graph based on before/after split:** In karateClubGraph.json, every node was given the same "color" attribute, but a different color based on it's "faction" attibute as outlined below

```
...
 {"name": "Mr Hi", "faction": 1, "color": 1 },
 {"name": "2", "faction": 1, "color": 1 },
 {"name": "3", "faction": 1, "color": 1 },
 {"name": "4", "faction": 1, "color": 1 },
 {"name": "5", "faction": 1, "color": 1 },
 {"name": "6", "faction": 1, "color": 1 },
 {"name": "7", "faction": 1, "color": 1 },
 {"name": "8", "faction": 1, "color": 1 },
 {"name": "9", "faction": 2, "color": 1 },
 {"name": "10", "faction": 2, "color": 1 },
 ...
```

3. **Toggle node color on click:** As shown in Listing 2. All nodes have been wired to the on click event. This means when any node is clicked, the function specialNodeClick, is called. And this function simply toggles the fill color of the nodes from the static color to the colors representing the different factions.

Listing 2: Toggle node color

```
//OnClick event block
var node = svg.selectAll(".node")
          .data(graph.nodes)
          .enter()
          .append("circle")
          .attr("class", "node")
          .on("click", specialNodeClick)
          .attr("id", function(d) { return d.name; })
          .attr("r", 5)
```

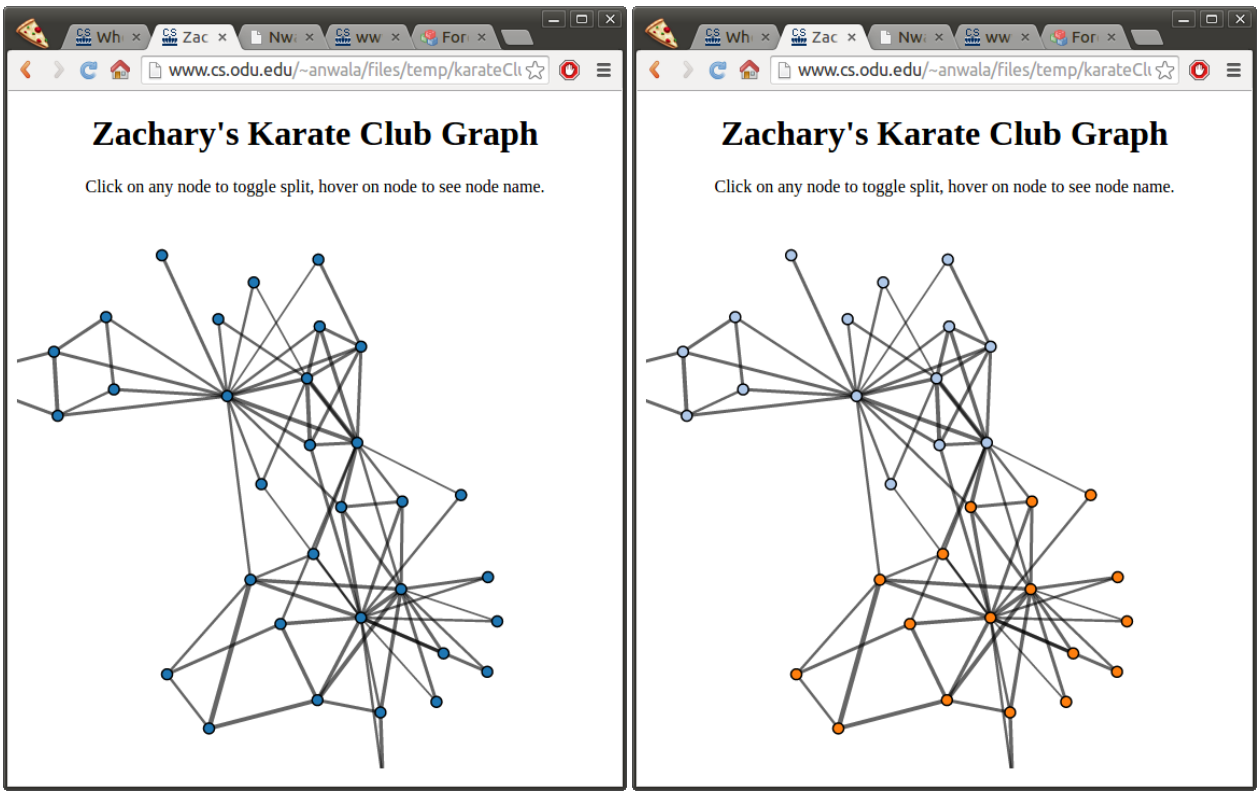
```
            .style("fill", function(d) { return color(d.color); })
            .call(force.drag);

function specialNodeClick(d)
{
  if(globalClickFlag==1)
  {
    d3.selectAll('.node').style('fill', function(d) { return color(d.faction - 10)
        ; });
    globalClickFlag = 0;
  }
  else
  {
    d3.selectAll('.node').style('fill', function(d) { return color(d.color); });
    globalClickFlag = 1;
  }
}
```

CONCLUSION 1

The result (Figure 1) of the foregoing operations can be seen at http://www.cs.odu.edu/~anwala/files/temp/karateClub.html

Figure 1: Karate Club Graph



(a) Before Split

(b) After Split

Problem 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. Attractiveness of the graph counts!

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

SOLUTION 2

With the use of D3.js (a JavaScript library for manipulating documents based on data) [2], 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 algorithm to get the followers of followers:** In order to get the followers of followers across a variable degree, I derived the iterative algorithm (gen) presented in below:

```
Input:
    twitterScreenName (s)
    maximumFollowerCount (m)
    maximumDegreeOfGraph (d)
    graphOutputFile (O)
Output:
    A graph file O, with the following format:

    <1st degree follower 1>+...+
        <2nd degree follower 1>+...+
            <3rd degree follower 1>+...+
                ...<nth degree follower 1>+
                   <nth degree follower 2>+
                   ...
                   <nth degree follower n>+
            <3rd degree follower 2>
            ...
            <3rd degree follower n>
        <2nd degree follower 2>
        ...
        <2nd degree follower n>
    <1st degree follower 2>+
    ...
    <1st degree follower n>+
```

```
procedure gen( s, m, d )
    while( end = False )

        if( O is empty ) then
            followersList = getFollowersFromTwitter(s, m);
            O.write( followersList );
        else:
            follower =
            getFollowerWithMinimumPlusCountAndWithinDegree(followersList, d);

            followersList = getFollowersFromTwitter(follower, d);
            O.write( followersList );

            if( all followers have been explored ) then
                end = True;
            endif
        endif
end

procedure getFollowersFromTwitter( userId, countOfFollowersToReturn )
    n = countOfFollowersToReturn;
    if( operation on userId is authorized ) then
        listOfFollowers = <follower1, follower2,..., follower(n)>;
    else
        listOfFollowers = <>;
    endif

    return listOfFollowers;
end

getFollowerWithMinimumPlusCountAndWithinDegree( listOfFollowers, d )

    follower = None;
    minimumPlus = Infinity;

    foreach follower in listOfFollowers:
        plusCount, tabCount = getPlusCountAndTabCount(follower);

        if( plusCount != 0 and plusCount < d ) then
            if( plusCount < minimumPlus and tabCount < d ) then
                minimumPlus = plusCount;
                item = follower;
            endif
        endif
    endfor
end
```

The implementation of gen(), is outlined in Listing 3.

Listing 3: Get followers of followers: gen()

```
#algorithm gen
...
if( len(lines) == 0 ):
     #base case initialize with 1 degree+
     friends = getXFriendsOfFriendsFromTwitter(screenName, maxFriends)
     for friend in friends:
          inputOutputFile.write(userName + globalDelimeter2 + friend + '+\n' )
else:
     #non-base case expand item with smallest plus within tabcount
     mininumPlus = 1000
     item = ''
     itemTabCount = 0
     indexOfItem = -1

     i = 0
     for line in lines:

          plusCount, tabCount = getPlusCountAndTabCount(line)

          #print line.strip(), plusCount, tabCount
          #get minimum plus count which is within the set degree
          if( plusCount != 0 and plusCount < maxDegree ):
               if( plusCount < mininumPlus and tabCount < maxDegree ):
                    mininumPlus = plusCount
                    item = line
                    itemTabCount = tabCount
                    indexOfItem = i

          i += 1
     #print item.strip(), itemTabCount, indexOfItem
     if( indexOfItem > -1 ):
          inputOutputFile.close()

          #expand
          try:
               inputOutputFile = open(outputFileName, 'w')
          except:
               exc_type, exc_obj, exc_tb = sys.exc_info()
               fname = os.path.split(exc_tb.tb_frame.f_code.co_filename)[1]
               print(fname, exc_tb.tb_lineno, sys.exc_info() )
               return

          for i in range(0, len(lines)):

               if(i == indexOfItem):

                    tabs = str('\t') * itemTabCount
                    inputOutputFile.write(tabs + lines[i].strip() + '+\n')

                    screenName = lines[i].strip()
                    screenName = screenName.split(globalDelimeter)[1]
```

Problem 2 continued on next page...

```
                    screenName = screenName.replace('+','')

                    #get userName
                    userName = lines[i].strip()
                    userName = userName.split(globalDelimeter)[0]
                    userName = userName.split(globalDelimeter2)
                    userName = userName[-1]


                    friends = getXFriendsOfFriendsFromTwitter(screenName,
                        maxFriends)

                    for friend in friends:

                         tabs = str('\t') * (itemTabCount+1)
                         inputOutputFile.write( tabs + userName + globalDelimeter2
                              + friend + '+\n' )
               else:
                    inputOutputFile.write(lines[i])
...
```

2. **Convert gen()'s output O to a links file:** This was achieved by a python script as shown in Listing 4.

Listing 4: Generate graph file

```
#Generate graph file
...
i = 0
for relationship in relationships:
     sourceAndTarget = relationship.split(' ,, ')[0]
     source = sourceAndTarget.split(' -> ')[0]
     target = sourceAndTarget.split(' -> ')[1]

     source = source.strip()
     target = target.strip()

     localType = globalType

     if( i != len(relationships) - 1):
          if( degree0Node.strip().lower() == source.strip().lower() ):
               localType = 'licensing'

          stringToWrite = '\t{source: "' + source + '", target: "' + target + '",
              type: "' + localType + '"},\n'
     else:
          if( degree0Node.strip().lower() == source.strip().lower() ):
               localType = 'licensing'

          stringToWrite = '\t{source: "' + source + '", target: "' + target + '",
              type: "' + localType + '"}\n'

     outputFile.write(stringToWrite)
     i = i + 1
```

```
...
```

3. **Visualize graph:** With the use of the graph template at http://bl.ocks.org/mbostock/1153292, I replaced the default data in the JavaScript file with the output of gen() as outlined in Listing 5.

Listing 5: Who follows who graph file

```
//Who follows who graph links
var links =
[
     {source: "Dr. Nelson", target: "Dragan Espenschied", type: "licensing"},
     {source: "Dragan Espenschied", target: "Brian Mackern", type: "suit"},
     {source: "Brian Mackern", target: "CCD RADIO", type: "suit"},
     {source: "CCD RADIO", target: "Los De La Tarde", type: "suit"},
     {source: "Los De La Tarde", target: "Toni Franois", type: "suit"},
     {source: "Los De La Tarde", target: "Rulo.", type: "suit"},
     {source: "Los De La Tarde", target: "Sopitas", type: "suit"},
     ...
     {source: "Mounia Lalmas", target: "Seth A. Tropper", type: "suit"},
     {source: "Dr. Nelson", target: "Denise Howell", type: "licensing"}
];
```

CONCLUSION 2

The result of running gen() (Figure 2.) can be seen at
http://www.cs.odu.edu/~anwala/files/temp/whosFollowingWho.html

Figure 2: Who follows who

(a) Who follows who

References

[1] BeautifulSoup. http://www.crummy.com/software/BeautifulSoup/bs4/doc/. Accessed: 2014-11-06.

[2] Data Driven Documents. http://d3js.org/. Accessed: 2014-11-06.