Assignment 6

CS532-s16: Web Sciences

Spring 2016

John Berlin

Generated on March 17, 2016

1

Question

```
1.  D3 graphing (5 points)

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

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

Answer

The twitter account I choose to visualize was the WebSciDL. Doing so proved to be an undertaking. The python file seen in listing 1 shows the process. The first thing I do is to get the followers of the WebSciDl group, this was easy as the rate limit for this was not an issue. After getting that comes the hard part.

In checking if the followers friends follow each other was not easy. The rate limit for "reference/get/friendships/show" only let me get through about 3 a day with sleeping to avoid rate limit hits. So I decided to do it a "longer way". So I simply got the friends the normal way and computed the friendship by code. It must be noted that in the method `getFriends` the json output is simply appended to the file. To correct this you must when finished do so by hand. The past assignments we have been asking twitter and I simply re-used the majority of that code to complete this portion.

Building the graph was simple enough thanks to using the networkx python library as seen in listing 2. As usual for more information please see the comments in the listing for details but in short the process is as such.

1. Get the twitter followers and those followers friends
2. Add the followers as nodes to the directed graph
3. Add WebSciDL edge to all nodes except WebSciDL
4. Add rest of the edges
5. Prepare data for output
6. Write out data

Please not that the image url contains _normal so I by hand well editor find replace removed it for the full image.

The javascript used for the vis is seen in listing 4 and the vis can be seen at

`http://www.cs.odu.edu/~jberlin/WebSciGraphVis/a6.html`

Table 1: Gender Homophilly results

males	females	nodes	edges	p	q	2pq	crossEdges	crossEdgesP	homophily	
120	82	38	120	1181	0.68	0.32	0.43	386	0.33	Yes

2

Question

```
2.  Gender homophily in your Twitter graph (5 points)

Take the Twitter graph you generated in question #1 and test for
male-female homophily.  For the purposes of this question you can
consider the graph as undirected (i.e., no distinction between
"follows" and "following").  Use the twitter name (not "screen
name"; for example "Michael L. Nelson" and not "@phonedude_mln")
and programatically determine if the user is male or female.  Some
sites that might be useful:

https://genderize.io/
https://pypi.python.org/pypi/gender-detector/0.0.4

Create a table of Twitter users and their likely gender.  List any
accounts that can't be determined and remove them from the graph.

Perform the homophily test as described in slides 11-15, Week 7.

Does your Twitter graph exhibit gender homophily?
```

As seen in the table below yes because crossEdge Percent(0.33) is greater than 2pq(0.32). The code used to generate the results is seen in listing 3 I used a method found on github to ask genderize.io for the results. The actual data is too large to put in this report and can be found in the file wsdlGenderResults2.csv.

```
import csv
import json

import networkx as nx
import tweepy

import config


# get the friends for a person
def getFriends(api, screenname):
    print("looking up friends for follower: %s\n" % screenname)
    fl = []
    # get the friends by using a cursor to query the twitter api
    items = {'screenname': screenname}
    try:
        for friend in tweepy.Cursor(api.friends, screen_name=screenname, count=200).items():
            fl.append(friend.screen_name)
    except Exception as e:
        print("There was an exception ", e)
    items['friends'] = fl
    with open("wsdlfollwerFriends.json", "a") as out:
        out.write(json.dumps(items, indent=2) + ",\n")
    return fl



# get the wsdl groups twitter followers
def getWSDL_follwers(tapi):
    fs = []  # type: list[tweepy.User]
    # get the followers by using a cursor to query the twitter api
    for page in tweepy.Cursor(tapi.followers, screen_name="WebSciDL", count=200).pages():
        print(page)
        fs.extend(page)

        # add the followers to out dic
    with open("wsdltwitterfollwers.csv", "w+") as out:
        out.write("name,screenName,imurl\n")
        for pp in fs:
            print(pp)
            out.write("%s,%s,%s\n" % (pp.name, pp.screen_name, pp.profile_image_url))


def get_friends():
    auth = tweepy.OAuthHandler(config.consumer_key, config.consumer_secret)
    auth.set_access_token(config.access_token, config.access_secret)
    # do not want twitter to slap a rate limit exceeded on me so explicitly wait after each request to avoid that
    api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True)  # type: tweepy.API
    with open("wsdlfollwerFriends.json", "r+") as r:
        it = json.load(r)
        print(it)
    gotten = set(map(lambda x: x['screenname'], it['followers']))
    for g in gotten:
        print(g)

    with open('wsdltwitterfollwers.csv', "r") as o:
        reader = csv.DictReader(o)
        out = {}
        for row in reader:
            print(row)
            if row['screenName'] not in gotten:
                print(row['screenName'])
                flist = getFriends(api=api, screenname=row['
screenName'])
                if len(flist) > 0:
                    print(len(flist))



if __name__ == "__main__":
    print("Hi")
    auth = tweepy.OAuthHandler(config.consumer_key, config.
    consumer_secret)
    auth.set_access_token(config.access_token, config.access_secret
    )
    # do not want twitter to slap a rate limit exceeded on me so
    explicitly wait after each request to avoid that
    api = tweepy.API(auth, wait_on_rate_limit=True,
    wait_on_rate_limit_notify=True)  # type: tweepy.API
    # # build_graph()
    # # tweepy.User
    # bg2()

    getWSDL_follwers(api)
    get_friends()
    # set up oauth
```
Listing 1: Get Twitter Data for the WSDL

```python
import csv
import json

import networkx as nx
h
class Node:
    def __init__(self, row):
        # name,screenName,imurl
        self.name = row['name']
        self.screenName = row['screenName']
        self.imurl = row['imurl']
        self.indegree = 0
        self.outdegree = 0
        self.group = 0
        self.eGroup = set()

    def to_jdic(self):
        out = {'name': self.name, 'screenName': self.screenName, 'imurl': self.imurl,
               'indegree': self.indegree, 'outdegree': self.outdegree, 'group': self.group,
               'egroups': list(self.eGroup)}
        return out

    def __str__(self):
        return self.screenName


class Edge:
    def __init__(self, source, sIndex, target, tIndex, edgeToGroup):
        self.source = source
        self.sIndex = sIndex
        self.target = target
        self.tIndex = tIndex
        self.edgeToGroup = edgeToGroup

    def to_jdic(self):
        out = {'source': self.sIndex, 'sname': self.source, 'target': self.tIndex, 'tname': self.target,
               'egroup': self.edgeToGroup}
        return out


class Edge2:
    def __init__(self, source, target):
        self.source = source
        self.target = target

    def to_jdic(self):
        out = {'source': self.source, 'target': self.target}
        return out


'''
These are the groups I got after inspecting the wsdl twitter followers by hand
```

```
wsdlTwitterHandles  members themselves
digLibHandles digital libraries and archival twitter accounts

'''
wsdlTwitterHandles = ['machawk1', 'aalsum', 'justinfbrunelle', '
    phonedude_mln', 'weiglemc', 'Galsondor',
                      'shawnmjones', 'ibnesayeed', 'LulwahMA', '
    yasmina_anwar', 'kaylamarie0110',
                      'maturban1', 'CorrenMcCoy', 'acnwala', '
    hanysalaheldeen', 'simplesimon2013', 'fmccown',
                      'mart1nkle1n', 'joansm1th', 'hvdsomp', '
    johnaberlin', 'WebSciDL', 'DanMilanko']

digLibHandles = ['internetarchive', 'HistWebArchives', 'TPDL2016',
    'UKWebArchive', "NetPreserve", "ijdl",
                 'JCDLConf', 'archiveitorg', "archiveis", "idjl", "
    webrecorder_io", "tpdl2016", "WOSP2014",
                 "WebArch_RT"]

userToGroup = {}

'''
    node groups:
    normal: 0
    wsdl: 1
    diglib: 2
    odu: 3

    edge groups means which node groups
    point to another node groups

    edge groups:
    normal -> normal 0
    normal -> wsdl 1
    normal -> dlib 2
    normal -> odu 3

    wsdl -> normal 4
    wsdl -> wsdl 5
    wsdl -> dlib 7
    wsdl -> odu 6


    dlib -> normal 8
    dlib -> wsdl 9
    dlib -> dlib 10
    dlib -> odu 11

    odu -> normal 12
    odu -> wsdl 13
    odu -> dlib 14
    odu -> odu 15
    '''


def getGroup(test):
    g = 0
```

```python
    if test in wsdlTwitterHandles:
        # print("We have a wsdl person ", test)
        g = 1
    elif test in digLibHandles:
        # print("We have a diglib person",test)
        g = 2
    elif 'odu' in test.lower() or 'monarch' in test.lower() or 'MaceandCrown' in test.lower():
        # print("We have odu",test)
        g = 3
    userToGroup[test] = g
    return g

#simple enumeration of the edge groups possibilities
edgeTGroup = {(0, 0): 0, (0, 1): 1, (0, 2): 2, (0, 3): 3, (1, 0): 4, (1, 1): 5, (1, 2): 6, (1, 3): 7,
              (2, 0): 8, (2, 1): 9, (2, 2): 10, (2, 3): 11,
              (3, 0): 12, (3, 1): 13, (3, 2): 14, (3, 3): 15}


def bg():
    with open("wsdlfollwerFriends.json", "r+") as r:
        it = json.load(r)

    it = it['followers']

    nlist = []
    #get a directed graph object
    graph = nx.DiGraph()

    # add the nodes to the graph
    with open('wsdltwitterfollwers.csv', "r") as o:
        reader = csv.DictReader(o)
        for row in reader:
            nlist.append(row['screenName'])
            n = Node(row)
            sname = row['screenName']

            if sname in "WebSciDL":
                print(sname)
            n.group = getGroup(sname)

            graph.add_node(row['screenName'], attr_dict={'nclass': n})

    # since I know that these nodes were gotten by the followers of the wsdl I add by hand the edge to it
    for sname in nlist:
        if "WebSciDL" not in sname:
            graph.add_edge(sname, "WebSciDL")

    nlist = sorted(nlist)
    '''
        get the friendship for the followers by adding the edges
        and checking if graph has the node we added first from the
        only wsdl follower file
    '''
    for ff in it:
        fflist = []
        screanname = ff['screenname']
        for ffFriend in ff['friends']:
            if graph.has_node(ffFriend) and "WebSciDL" not in ffFriend:
                fflist.append(ffFriend)
                graph.add_edge(screanname, ffFriend)

    nodeList = []
    edgeList = []
    '''
        build our output
        for each node in the graph
        get its python class and determine the in out degree
        for each edge for the node add its edge groups
    '''
    for node, ndata in sorted(graph.nodes(data=True), key=lambda x: x[0]):
        # print(node, ndata['nclass'])
        nodeClass = ndata['nclass']
        # print(nodeClass.screenName)
        nodeClass.indegree = graph.in_degree(node)
        nodeClass.outdegree = graph.out_degree(node)
        nodeList.append(nodeClass)
        for source, target in graph.edges(node):
            nodeClass.eGroup.add(edgeTGroup[(userToGroup[source], userToGroup[target])])
            e = Edge(source, nlist.index(source), target, nlist.index(target),
                     edgeTGroup[(userToGroup[source], userToGroup[target])])
            edgeList.append(e)
            print("%s—>%s" % (source, target))
        print("++++++++++++++++++++++++++++++\n")

    g = {}
    g['nodes'] = nodeList
    g['links'] = edgeList
    print(json.dumps(g, default=lambda c: c.to_jdic(), indent=1))
    with open("wsdlgraphData.json", "w+") as out:
        out.write(json.dumps(g, default=lambda c: c.to_jdic(), indent=1))
```

Listing 2: Build The WSDL Graph

```python
import csv
import json
from collections import Counter

import networkx as nx
import requests

edgeTGroup = {(0, 0): 0, (0, 1): 1, (0, 2): 2, (0, 3): 3, (1, 0): 4, (1, 1): 5, (1, 2): 6, (1, 3): 7,
              (2, 0): 8, (2, 1): 9, (2, 2): 10, (2, 3): 11,
              (3, 0): 12, (3, 1): 13, (3, 2): 14, (3, 3): 15}

wsdlTwitterHandles = ['machawk1', 'aalsum', 'justinfbrunelle', 'phonedude_mln', 'weiglemc', 'Galsondor',
                      'shawnmjones', 'ibnesayeed', 'LulwahMA', 'yasmina_anwar', 'kaylamarie0110',
                      'maturban1', 'CorrenMcCoy', 'acnwala', 'hanysalaheldeen', 'simplesimon2013', 'fmccown',
                      'mart1nkle1n', 'joansm1th', 'hvdsomp', 'johnaberlin', 'WebSciDL', 'DanMilanko']

digLibHandles = ['internetarchive', 'HistWebArchives', 'TPDL2016', 'UKWebArchive', "NetPreserve", "ijdl",
                 'JCDLConf', 'archiveitorg', "archiveis", "idjl", "webrecorder_io", "tpdl2016", "WOSP2014",
                 "WebArch_RT"]

userToGroup = {}


def getGroup(test):
    g = 0
    if test in wsdlTwitterHandles:
        # print("We have a wsdl person ", test)
        g = 1
    elif test in digLibHandles:
        # print("We have a diglib person",test)
        g = 2
    elif 'odu' in test.lower() or 'monarch' in test.lower() or 'MaceandCrown' in test.lower():
        # print("We have odu",test)
        g = 3
    userToGroup[test] = g
    return g


def getGenders(names):
    '''
    Thanks https://github.com/block8437/gender.py
    The MIT License (MIT)

    Copyright (c) 2013 block8437

    Permission is hereby granted, free of charge, to any person obtaining a copy of
    this software and associated documentation files (the "Software"), to deal in
    the Software without restriction, including without limitation the rights to
    use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of
    the Software, and to permit persons to whom the Software is furnished to do so,
    subject to the following conditions:

    The above copyright notice and this permission notice shall be included in all
    copies or substantial portions of the Software.

    THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS
    FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER
    IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
    CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
    '''
    url = ""
    cnt = 0
    for name in names:
        if url == "":
            url = "name[0]=" + name
        else:
            cnt += 1
            url = url + "&name[" + str(cnt) + "]=" + name

    req = requests.get("http://api.genderize.io?" + url)
    results = json.loads(req.text)

    retrn = []
    for result in results:
        if result["gender"] is not None:
            retrn.append((result["gender"], result["probability"], result["count"]))
        else:
            retrn.append((u'None', u'0.0', 0.0))
    return retrn


def splitOrWhole(s):
    # gender only works on first names so split the name
    splitted = s['name'].split(' ')
    # there was a first name
    if len(splitted) > 0:
        return splitted[0]
    else:  # otherwise just give back the original
        return s


```

```python
class GNode:
    def __init__(self, row, g):
        # name,screenName,imurl
        self.name = row['name']
        self.imurl = row['imurl']
        self.screenName = row['screenname']
        self.indegree = 0
        self.outdegree = 0
        self.gender = g
        self.group = 0

    def to_jdic(self):
        out = {'name': self.name, 'screenName': self.screenName, '
    gender': self.gender,
               'indegree': self.indegree, 'outdegree': self.
    outdegree, "group":self.group}
        return out

    def __str__(self):
        return self.screenName


class GEdge:
    def __init__(self, source, sIndex, target, tIndex, edgeToGroup,
     sg, tg,cross):
        self.source = source
        self.sIndex = sIndex
        self.target = target
        self.tIndex = tIndex
        self.sGender = sg
        self.tGender = tg
        self.edgeToGroup = edgeToGroup
        self.isCross = cross

    def to_jdic(self):
        out = {'source': self.sIndex, 'sname': self.source, 'target
    ': self.tIndex, 'tname': self.target,
               'egroup': self.edgeToGroup, "cross":self.isCross}
        return out


def getGender():
    with open('wsdltwitterfollwers.csv', "r") as o:
        reader = csv.DictReader(o)
        rrList = []
        rList = []
        for row in reader:
            rList.append(row)
            # be nice to the api so we only send 9 at a time as max
     is 10
            if len(rList) == 9:
                rrList.append(list(rList))
                rList.clear()
        with(open("wsdlGenderResults2.csv", "w+")) as out:
            out.write("name,screenname,gender,prob\n")
            for rl in rrList:
                result = getGenders(list(map(lambda r: splitOrWhole
    (r), rl)))
                for gdr, rrl in zip(result, rl):
                    print(gdr)
                    out.write("%s,%s,%s,%f\n" % (rrl['name'], rrl['
    screenName'], gdr[0], float(gdr[1])))


def check_gender_homophily():
    with open("wsdlfollwerFriends.json", "r+") as r:
        it = json.load(r)

    it = it['followers']

    nlist = []
    ng = {}
    graph = nx.DiGraph()
    genderCounter = Counter()

    with open('wsdlGenderResults2.csv', "r") as o:
        reader = csv.DictReader(o)
        for row in reader:
            if 'None' not in row['gender']:
                nlist.append(row['screenname'])
                n = GNode(row, row['gender'])
                ng[row['screenname']] = row['gender']
                genderCounter[row['gender']] += 1
                genderCounter['peeps'] += 1
                sname = row['screenname']
                n.group = getGroup(sname)
                graph.add_node(row['screenname'], attr_dict={'
    nclass': n})

    nlist = sorted(nlist)

    for ff in it:
        fflist = []
        screanname = ff['screenname']
        if graph.has_node(screanname):
            for ffFriend in ff['friends']:
                if graph.has_node(ffFriend):
                    fflist.append(ffFriend)
                    cross = 0
                    if  ng[screanname] != ng[ffFriend]:
                        cross = 1
                    graph.add_edge(screanname, ffFriend, attr_dict
    ={'sg': ng[screanname], 'tg': ng[ffFriend], 'cross':cross})

    nGenders = genderCounter['peeps']
    nMale = genderCounter['male']
    nFemale = genderCounter['female']

    p = nMale / nGenders
    q = nFemale / nGenders
    twopq = 2 * p * q
    r= "Total Members of Gender graph: %d\nNumber of males in graph
    : %d\nNumber of Females in graph: %d"%(nGenders, nMale, nFemale
    )
    r2= "P value of: %.2f\nQ value of: %.2f\n2pq value of %2f" % (p, q, twopq)


    print("Checking cross edges")
    numCrossGenderEdges = 0
    nEdges = 0
    for source, target, gender in graph.edges(data=True):
        nEdges += 1
        if gender['sg'] != gender['tg']:
            numCrossGenderEdges += 1


    print(r)
    print(r2)
    crossPercent = numCrossGenderEdges/nEdges
    print("Number of edges: %d, Number of cross-gender edges: %d, Percent: %.2f "%(nEdges,numCrossGenderEdges,crossPercent))
    win="Yes"
    if crossPercent == twopq:
        print("2pq(%.2f) == cross edge precentage(%.2f)"%(twopq,crossPercent))
        print("There is no homophily")
        win="No"
    else:
        print("2pq(%.2f) != cross edge precentage(%.2f)"%(twopq,crossPercent))
        print("There is homophily")

    with open("homophillyTest.csv","w+") as out:
        out.write("males,females,nodes,edges,p,q,2pq,crossEdges,crossEdgesP,homophily\n")
        out.write("%d,%d,%d,%d,%d,%.2f,%.2f,%.2f,%d,%.2f,%s"%(nGenders, nMale, nFemale,nGenders,nEdges,p,q,twopq,numCrossGenderEdges,crossPercent,win))

    nodeList = []
    edgeList = []
    for node, ndata in sorted(graph.nodes(data=True), key=lambda x: x[0]):
        # print(node, ndata['nclass'])
        nodeClass = ndata['nclass']
        # print(nodeClass.screenName)
        nodeClass.indegree = graph.in_degree(node)
        nodeClass.outdegree = graph.out_degree(node)
        nodeList.append(nodeClass)
        for source, target,data in graph.edges(node,data=True):
            e = GEdge(source,nlist.index(source),target,nlist.index(target),edgeTGroup[(userToGroup[source],userToGroup[target])],data['sg'],data['tg'],data['cross'])
            edgeList.append(e)


    g = {}
    g['nodes'] = nodeList
    g['links'] = edgeList
    with open("wsdlgraphGender.json","w+") as out:
        out.write(json.dumps(g,default=lambda c:c.to_jdic(),
    indent=1))

if __name__ == "__main__":
    check_gender_homophily()
```
Listing 3: Check Gender

```
var width, height, color, svg, graph, linkGroups,
   nodefill = ["#8C564B", "#AEC7E8", "#2CA02C", "#1F77B4"], textsG,
    force,
   linksG, nodesG, k, tooltip, inOutExtent, circleRadius, node_drag
    , link;

var curLinksData = [], curNodesData = [], filter,
   layout, linkedByIndex = {},
   node, toggle = 0, text,
   sort, allData,
   padding = 1.5, // separation between circles
   radius = 25, curWhat = 16;


//most things were derived from    //https://flowingdata.com
    /2012/08/02/how-to-make-an-interactive-network-visualization/

//these are the color values and groupings I have mapped
// node groups:
//     normal: 0
// wsdl: 1
// diglib: 2
// odu: 3
//
// edge groups:
//     normal -> normal 0
// normal -> wsdl 1
// normal -> dlib 2
// normal -> odu 3
//
// wsdl -> normal 4
// wsdl -> wsdl 5
// wsdl -> dlib 7
// wsdl -> odu 6
//
//
// dlib -> normal 8
// dlib -> wsdl 9
// dlib -> dlib 10
// dlib -> odu 11
//
// odu -> normal 12
// odu -> wsdl 13
// odu -> dlib 14
// odu -> odu 15


function linkDist(l) {
   var ret;
   //make the link distances more dynamic i Kinda gave up here
   if (l.source.group == 0) {
      ret = 200;
   } else if (l.source.group == 1) {
      ret = 100;
   } else if (l.source.group == 2) {
      ret = 150;
   } else {
      ret = 175;
   }
   if (curWhat == 0) {
      ret = 200;
   }
   if (curWhat == 5) {
      ret = 200;
   }
   if (curWhat == 6) {
      ret = 200;
   }
   if (curWhat == 15) {
      ret = 200;
   }
   if (curWhat == 8) {
      ret = 200;
   }
   return ret;
}

function collide(node) {
   //got this from examples http://bl.ocks.org/mbostock/3231298#index.html
   var r = 2 * node.radius + 8,
      nx1 = node.x - r,
      nx2 = node.x + r,
      ny1 = node.y - r,
      ny2 = node.y + r;
   return function (quad, x1, y1, x2, y2) {
      if (quad.point && (quad.point !== node)) {
         var x = node.x - quad.point.x,
            y = node.y - quad.point.y,
            l = Math.sqrt(x * x + y * y),
            r = node.radius + quad.point.radius + padding;
         if (l < r) {
            l = (l - r) / l * .8;
            node.x -= x *= l;
            node.y -= y *= l;
            quad.point.x += x;
            quad.point.y += y;
         }
      }
      return x1 > nx2
         || x2 < nx1
         || y1 > ny2
         || y2 < ny1;
   };
}

function showDetails(d) {
   //show the detail about a node in the graph

   var content = '<p class="main">User Name: ' + d.name + '</span></p>';
   content += '<hr class="tooltip-hr">';
   content += '<p class="main"> Screen Name: ' + d.screenName + '</span></p>';
```

```
    content += '<hr class="tooltip-hr">';
    content += '<img src=' + d.imurl + ' alt="Stuff" style="width
     :100px;height:100px;">';
    tooltip.showTooltip(content, d3.event);
}


//begin not really used section
function mapNameToNode(nodes) {
    var map = d3.map();
    nodes.forEach(function (node) {
       map.set(node.screenName, node);
    });
    return map;
}


function buildIndex() {
    for (var i = 0; i < allData.nodes.length; i++) {
       linkedByIndex[i + "," + i] = 1;
    }
    allData.links.forEach(function (d) {
       linkedByIndex[d.source.index + "," + d.target.index] = 1;
    });
}

function neighboring(a, b) {
    return linkedByIndex[a.index + "," + b.index] || linkedByIndex[b
     .index + "," + a.index];
}
//end not really used section


//so when we redo things I can have the data already nice and tidy
function prepairData(data) {
    //the node radius is based on the sum of their in and out degree
    inOutExtent = d3.extent(data.nodes, function (node) {
       return node.indegree + node.outdegree;
    });

    circleRadius = d3.scale.sqrt()
       .range([3, 14]).domain(inOutExtent);

    data.nodes.forEach(function (n) {
       //where do we want to place our nodes
       n.x = Math.floor(Math.random() * width);
       n.y = Math.floor(Math.random() * height);
       n.radius = circleRadius(n.indegree + n.outdegree);
    });

    var nameNodeMap = mapNameToNode(data.nodes);
    data.links.forEach(function (l) {
       //point our links to the nodes that have position computed
     already
       var s = nameNodeMap.get(l.sname);
       var t = nameNodeMap.get(l.tname);
       l.sx = s.x;
```

```
      l.sy = s.y;
      l.tx = t.x;
      l.ty = t.y;
   });
   return data;
}

//begin copy pasta from http://www.coppelia.io/2014/07/an-a-to-z-of
    -extra-features-for-the-d3-force-layout/
function dragstart(d, i) {
   force.stop(); // stops the force auto positioning before you
    start dragging
}
function dragmove(d, i) {
   d.px += d3.event.dx;
   d.py += d3.event.dy;
   d.x += d3.event.dx;
   d.y += d3.event.dy;
   tick();
}
function dragend(d, i) {
   // of course set the node to fixed so the force doesn't include
    the node in its auto positioning stuff
   d.fixed = true;
   tick();
   force.resume();
}
function releasenode(d) {
   d.fixed = false; // of course set the node to fixed so the force
     doesn't include the node in its auto positioning stuff
   //force.resume();
}

//end copy pasta from http://www.coppelia.io/2014/07/an-a-to-z-of-
    extra-features-for-the-d3-force-layout/


function tick(e) {
   /*
      Do one iteraction of the force simulation
      consider our div elements size so we do not go outisde of it
    */
   var iw = $("#vis").innerWidth(), ih = $("#vis").innerHeight();
   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 = Math.max(6, Math.min(iw, d.x));
      })
      .attr("cy", function (d) {
        return d.y = Math.max(6, Math.min(ih, d.y));
      });


  node.each(collide);
}

function updateNodes() {
  /*
     when a user choses a new set of links to display we must redo
    the nodes
   */
  node = nodesG.selectAll("node")
      .data(curNodesData);
  node.enter().append("circle")
      .attr("class", "node")
      .attr("r", function (n) {
        return circleRadius(n.indegree + n.outdegree);
      })
      .style("fill", function (d) {
        return d3.rgb(nodefill[d.group]);
      })
      .style("stroke", function (d) {
        return d3.rgb(nodefill[d.group]).brighter().toString();
      })
      .on("mouseover", showDetails).on("mouseout", function () {
      tooltip.hideTooltip();
  }).on('dblclick', releasenode)
      .call(node_drag);
  //nuke them when done
  node.exit().remove();
}


function updateLinks() {
  /*
     when a user choses a new set of links to display we must redo
    the nodes
     this bad boy does the heavy lifting for us
     our nodes are set invisible based on their weight ie links to
    them
   */
  link = linksG.selectAll("link")
      .data(curLinksData);

  link.enter()
      .append("line")
      .attr("class", "link")
      .style("marker-end", "url(#to)");
  svg.selectAll("defs").remove();
  //since this graph is directed add some pointers to indicate the
    direction
  var def = svg.append("defs").selectAll("marker").data(["to"]);

```

```
    def.enter().append("marker")
        .attr("id", function (d) {
            return d;
        })
        .attr("viewBox", "0 -5 10 10")
        .attr("refX", 25)
        .attr("refY", 0)
        .attr("markerWidth", 6)
        .attr("markerHeight", 6)
        .attr("orient", "auto")
        .append("path")
        .attr("d", "M0,-5L10,0L0,5 L10,0 L0, -5")
        .style("stroke", "#080808")
        .style("opacity", "1.0");
    def.exit().remove();
    link.exit().remove();
    node.filter(function (n) {
        return n.weight == 0;
    }).style("visibility", "hidden");

}


function update() {
    //on each change update the nodes
    force.links(curLinksData);
    force.nodes(curNodesData);
    force.start();
    updateNodes();
    updateLinks();

}


function makeVis(data) {
    /*
        called once do all things to make it so number one
        perpare the data
        get width height of the vis div element
     */
    allData = prepairData(data);
    width = $("#vis").width();
    height = $(window).innerHeight();
    //buildIndex();
    //build our link groups for our link displayer
    linkGroups = _.groupBy(allData.links, function (l) {
        return l.egroup
    });
    linkGroups[16] = _.filter(allData.links, function (l) {
        return l.tname == "WebSciDL";
    });
    linkGroups[17] = allData.links;
    curNodesData = allData.nodes;
    curLinksData = linkGroups[16];
    //do d3 things
    force = d3.layout.force();
    tooltip = Tooltip("vis-tooltip", 230);
    node_drag = d3.behavior.drag()
        .on("dragstart", dragstart)
        .on("drag", dragmove)
        .on("dragend", dragend);
    svg = d3.select("#vis").append("svg")
        .attr("width", width).attr("height", height);
    linksG = svg.append("g").attr("id", "links");
    nodesG = svg.append("g").attr("id", "nodes");
    textsG = svg.append("g").attr("id", "texts");
    force.size([width, height])
        .charge(-100).linkDistance(linkDist)
        .on("tick", tick);

    inOutExtent = d3.extent(data.nodes, function (node) {
        return node.indegree + node.outdegree;
    });

    circleRadius = d3.scale.sqrt()
        .range([3, 14]).domain(inOutExtent);
    update();
    buildIndex();
    $("#link_select").on("change", function (e) {
        //when our link_select value has changed update vis
        updateData($(this).val());
    });
}


function updateData(what) {
    //change our visualization to the link group selected
    curWhat = what;
    console.log("current what", what);
    curLinksData = linkGroups[what];
    node.each(function (n) {
        n.fixed = false;
    });
    link.remove();
    node.remove();
    update();
}

function resize() {
    //I wanted to take a stab at dynamic sizing of our vis
    width = $("#vis").width();
    height = $(window).innerHeight();
    svg.attr("width", width).attr("height", height);
    force.size([width, height]).resume();
}

//when the window resizes call resize
d3.select(window).on("resize", resize);

//load our data
d3.json("data/wsdlgraphData.json", function (error, data) {
    makeVis(data);
});
```

Listing 4: WebSciDL graph vis