Lab 1 - Graph Structure & Modelling

COMP-47270 Computational Network Analysis and Modelling

Author : Paula Dwan
Email : paula.dwan@gmail.com
Student ID : 13208660

Table of Contents

PRELIMINARIES

INSTALLING LIBRARIES – PYTHON-NUMPY & PYTHON-SCIPY

```
✔ ~$ sudo apt-get install python-numpy python-scipy
[sudo] password for paula:
Reading package lists...  Done
Building dependency tree
Reading state information...  Done
The following packages were automatically installed and are no longer required:
   linux-headers-3.13.0-32 linux-headers-3.13.0-32-generic
   linux-headers-3.13.0-39 linux-headers-3.13.0-39-generic
   linux-image-3.13.0-32-generic linux-image-3.13.0-39-generic
   linux-image-extra-3.13.0-32-generic linux-image-extra-3.13.0-39-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
   libamd2.3.1 libblas3 libcamd2.3.1 libccolamd2.8.0 libcholmod2.1.2
   libexpat1-dev libgfortran3 liblapack3 libpython-dev libpython2.7-dev
   libumfpack5.6.2 python-decorator python-dev python2.7-dev
Suggested packages:
   gfortran python-nose python-numpy-dbg python-numpy-doc
The following NEW packages will be installed:
   libamd2.3.1 libblas3 libcamd2.3.1 libccolamd2.8.0 libcholmod2.1.2
   libexpat1-dev libgfortran3 liblapack3 libpython-dev libpython2.7-dev
   libumfpack5.6.2 python-decorator python-dev python-numpy python-scipy
   python2.7-dev
0 upgraded, 16 newly installed, 0 to remove and 14 not upgraded.
Need to get 30.8 MB/34.6 MB of archives.
After this operation, 85.9 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libamd2.3.1 amd64 1:4.2.1-3ubuntu1 [23.5 kB]
Get:2 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libcamd2.3.1 amd64 1:4.2.1-3ubuntu1 [21.9 kB]
Get:3 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libccolamd2.8.0 amd64 1:4.2.1-3ubuntu1 [23.3
   kB]
Get:4 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libcholmod2.1.2 amd64 1:4.2.1-3ubuntu1 [281 kB]
Get:5 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libexpat1-dev amd64 2.1.0-4ubuntu1 [115 kB]
Get:6 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libpython2.7-dev amd64 2.7.6-8 [22.0 MB]
Get:7 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libumfpack5.6.2 amd64 1:4.2.1-3ubuntu1 [214 kB]
Get:8 http://ie.archive.ubuntu.com/ubuntu/ trusty/main libpython-dev amd64 2.7.5-5ubuntu3 [7,078 B]
Get:9 http://ie.archive.ubuntu.com/ubuntu/ trusty/main python2.7-dev amd64 2.7.6-8 [269 kB]
Get:10 http://ie.archive.ubuntu.com/ubuntu/ trusty/main python-dev amd64 2.7.5-5ubuntu3 [1,166 B]
Get:11 http://ie.archive.ubuntu.com/ubuntu/ trusty/universe python-scipy amd64 0.13.3-1build1 [7,862
   kB]
Fetched 30.8 MB in 6s (4,681 kB/s)
Selecting previously unselected package libamd2.3.1:amd64.
(Reading database ...  270756 files and directories currently installed.)
Preparing to unpack .../libamd2.3.1_1%3a4.2.1-3ubuntu1_amd64.deb ...
Unpacking libamd2.3.1:amd64 (1:4.2.1-3ubuntu1) ...
Selecting previously unselected package libcamd2.3.1:amd64.
Preparing to unpack .../libcamd2.3.1_1%3a4.2.1-3ubuntu1_amd64.deb ...
Unpacking libcamd2.3.1:amd64 (1:4.2.1-3ubuntu1) ...
Selecting previously unselected package libccolamd2.8.0:amd64.
Preparing to unpack .../libccolamd2.8.0_1%3a4.2.1-3ubuntu1_amd64.deb ...
Unpacking libccolamd2.8.0:amd64 (1:4.2.1-3ubuntu1) ...
Selecting previously unselected package libblas3.
Preparing to unpack .../libblas3_1.2.20110419-7_amd64.deb ...
Unpacking libblas3 (1.2.20110419-7) ...
Selecting previously unselected package libgfortran3:amd64.
Preparing to unpack .../libgfortran3_4.8.2-19ubuntu1_amd64.deb ...
Unpacking libgfortran3:amd64 (4.8.2-19ubuntu1) ...
Selecting previously unselected package liblapack3.
Preparing to unpack .../liblapack3_3.5.0-2ubuntu1_amd64.deb ...
Unpacking liblapack3 (3.5.0-2ubuntu1) ...
Selecting previously unselected package libcholmod2.1.2:amd64.
Preparing to unpack .../libcholmod2.1.2_1%3a4.2.1-3ubuntu1_amd64.deb ...
Unpacking libcholmod2.1.2:amd64 (1:4.2.1-3ubuntu1) ...
Selecting previously unselected package libexpat1-dev:amd64.
Preparing to unpack .../libexpat1-dev_2.1.0-4ubuntu1_amd64.deb ...
Unpacking libexpat1-dev:amd64 (2.1.0-4ubuntu1) ...
Selecting previously unselected package libpython2.7-dev:amd64.
Preparing to unpack .../libpython2.7-dev_2.7.6-8_amd64.deb ...
Unpacking libpython2.7-dev:amd64 (2.7.6-8) ...
Selecting previously unselected package libumfpack5.6.2:amd64.
Preparing to unpack .../libumfpack5.6.2_1%3a4.2.1-3ubuntu1_amd64.deb ...
Unpacking libumfpack5.6.2:amd64 (1:4.2.1-3ubuntu1) ...
```

```
Selecting previously unselected package libpython-dev:amd64.
Preparing to unpack .../libpython-dev_2.7.5-5ubuntu3_amd64.deb ...
Unpacking libpython-dev:amd64 (2.7.5-5ubuntu3) ...
Selecting previously unselected package python-decorator.
Preparing to unpack .../python-decorator_3.4.0-2build1_all.deb ...
Unpacking python-decorator (3.4.0-2build1) ...
Selecting previously unselected package python2.7-dev.
Preparing to unpack .../python2.7-dev_2.7.6-8_amd64.deb ...
Unpacking python2.7-dev (2.7.6-8) ...
Selecting previously unselected package python-dev.
Preparing to unpack .../python-dev_2.7.5-5ubuntu3_amd64.deb ...
Unpacking python-dev (2.7.5-5ubuntu3) ...
Selecting previously unselected package python-numpy.
Preparing to unpack .../python-numpy_1%3a1.8.2-0ubuntu0.1_amd64.deb ...
Unpacking python-numpy (1:1.8.2-0ubuntu0.1) ...
Selecting previously unselected package python-scipy.
Preparing to unpack .../python-scipy_0.13.3-1build1_amd64.deb ...
Unpacking python-scipy (0.13.3-1build1) ...
Processing triggers for man-db (2.6.7.1-1ubuntu1) ...
Setting up libamd2.3.1:amd64 (1:4.2.1-3ubuntu1) ...
Setting up libcamd2.3.1:amd64 (1:4.2.1-3ubuntu1) ...
Setting up libccolamd2.8.0:amd64 (1:4.2.1-3ubuntu1) ...
Setting up libblas3 (1.2.20110419-7) ...
update-alternatives: using /usr/lib/libblas/libblas.so.3 to provide /usr/lib/libblas.so.3
   (libblas.so.3) in auto mode
Setting up libgfortran3:amd64 (4.8.2-19ubuntu1) ...
Setting up liblapack3 (3.5.0-2ubuntu1) ...
update-alternatives: using /usr/lib/lapack/liblapack.so.3 to provide /usr/lib/liblapack.so.3
   (liblapack.so.3) in auto mode
Setting up libcholmod2.1.2:amd64 (1:4.2.1-3ubuntu1) ...
Setting up libexpat1-dev:amd64 (2.1.0-4ubuntu1) ...
Setting up libpython2.7-dev:amd64 (2.7.6-8) ...
Setting up libumfpack5.6.2:amd64 (1:4.2.1-3ubuntu1) ...
Setting up libpython-dev:amd64 (2.7.5-5ubuntu3) ...
Setting up python-decorator (3.4.0-2build1) ...
Setting up python2.7-dev (2.7.6-8) ...
Setting up python-dev (2.7.5-5ubuntu3) ...
Setting up python-numpy (1:1.8.2-0ubuntu0.1) ...
Setting up python-scipy (0.13.3-1build1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.4) ...
```

Installing Libraries – python-matplotlib

```
✔ ~$ sudo apt-get install python-matplotlib
Reading package lists...  Done
Building dependency tree
Reading state information...  Done
The following packages were automatically installed and are no longer required:
   linux-headers-3.13.0-32 linux-headers-3.13.0-32-generic
   linux-headers-3.13.0-39 linux-headers-3.13.0-39-generic
   linux-image-3.13.0-32-generic linux-image-3.13.0-39-generic
   linux-image-extra-3.13.0-32-generic linux-image-extra-3.13.0-39-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
   fonts-lyx python-dateutil python-matplotlib-data python-pyparsing python-tk
   python-tz
Suggested packages:
   dvipng inkscape ipython python-excelerator python-matplotlib-doc python-nose
   python-qt4 python-sip python-tornado python-traits python-wxgtk2.8
   texlive-latex-extra ttf-staypuft tix python-tk-dbg
The following NEW packages will be installed:
   fonts-lyx python-dateutil python-matplotlib python-matplotlib-data
   python-pyparsing python-tk python-tz
0 upgraded, 7 newly installed, 0 to remove and 14 not upgraded.
Need to get 0 B/4,101 kB of archives.
After this operation, 14.5 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Selecting previously unselected package fonts-lyx.
(Reading database ...  272461 files and directories currently installed.)
Preparing to unpack .../fonts-lyx_2.0.6-1build1_all.deb ...
Unpacking fonts-lyx (2.0.6-1build1) ...
Selecting previously unselected package python-dateutil.
Preparing to unpack .../python-dateutil_1.5+dfsg-1ubuntu1_all.deb ...
Unpacking python-dateutil (1.5+dfsg-1ubuntu1) ...
Selecting previously unselected package python-matplotlib-data.
Preparing to unpack .../python-matplotlib-data_1.3.1-1ubuntu5_all.deb ...
Unpacking python-matplotlib-data (1.3.1-1ubuntu5) ...
```

```
Selecting previously unselected package python-pyparsing.
Preparing to unpack .../python-pyparsing_2.0.1+dfsg1-1build1_all.deb ...
Unpacking python-pyparsing (2.0.1+dfsg1-1build1) ...
Selecting previously unselected package python-tz.
Preparing to unpack .../python-tz_2012c-1build1_all.deb ...
Unpacking python-tz (2012c-1build1) ...
Selecting previously unselected package python-matplotlib.
Preparing to unpack .../python-matplotlib_1.3.1-1ubuntu5_amd64.deb ...
Unpacking python-matplotlib (1.3.1-1ubuntu5) ...
Selecting previously unselected package python-tk.
Preparing to unpack .../python-tk_2.7.5-1ubuntu1_amd64.deb ...
Unpacking python-tk (2.7.5-1ubuntu1) ...
Processing triggers for fontconfig (2.11.0-0ubuntu4.1) ...
Setting up fonts-lyx (2.0.6-1build1) ...
Setting up python-dateutil (1.5+dfsg-1ubuntu1) ...
Setting up python-matplotlib-data (1.3.1-1ubuntu5) ...
Setting up python-pyparsing (2.0.1+dfsg1-1build1) ...
Setting up python-tz (2012c-1build1) ...
Setting up python-matplotlib (1.3.1-1ubuntu5) ...
Setting up python-tk (2.7.5-1ubuntu1) ...
Processing triggers for python-support (1.0.15) …
```

CODE PROVIDED

GRAPHPROPS.PY

```python
import networkx as nx
import matplotlib.pyplot as plt
import scipy as sp

#G = nx.gnp_random_graph(1000,0.02)

G = nx.read_edgelist("../../Data/SNAP/facebook_combined.txt")

A = nx.to_scipy_sparse_matrix(G);
fig=plt.figure()
plt.spy(A)  # display matrix
fig.show()

r = nx.degree_assortativity_coefficient(G)

print("Degree Assortativity Coefficient = ", r)

n = nx.average_neighbor_degree(G)
k = nx.k_nearest_neighbors(G)
```

DEGSEQ.PY

Calculate and display degree distribution and display using matlab using Random graph with probability of 0.02, sort from highest to lowest.

```python
import networkx as nx
import matplotlib.pyplot as plt

G = nx.gnp_random_graph(100,0.02)

degree_sequence=sorted(nx.degree(G).values(),reverse=True) # degree sequence
#print "Degree sequence", degree_sequence
dmax=max(degree_sequence)

plt.loglog(degree_sequence,'b-',marker='o')
plt.title("Degree rank plot")
plt.ylabel("degree")
plt.xlabel("rank")

# draw graph in inset
plt.axes([0.45,0.45,0.45,0.45])
Gcc=sorted(nx.connected_component_subgraphs(G), key = len, reverse=True)[0]
pos=nx.spring_layout(Gcc)
plt.axis('off')
nx.draw_networkx_nodes(Gcc,pos,node_size=20)
nx.draw_networkx_edges(Gcc,pos,alpha=0.4)

plt.savefig("degree_histogram.png")
plt.show()
```

Laboratory 1 – Basic Requirements

In this lab we should try to achieve the following :

1. Get up and running with NetworkX and Python
2. Download some networks from SNAP repository {http://snap.stanford.edu}

   Choose some directed and undirected networks from different application domains {social, technological, biological networks.}
3. Use NetworkX to compute
   - The degree distribution;
   - The assortativity co-efficient;
   - The clustering co-efficient.

   of the chosen networks

Using NetworkX to compute

1 : Degree Distribution

Details – MultiGraph.degree

	Graph.degree(nbunch=None, weight=None)¶
Explanation	Return the degree of a node or nodes. The node degree is the number of edges adjacent to that node.
Parameters	nbunch : iterable container, optional (default=all nodes) A container of nodes. The container will be iterated through once. weight : string or None, optional (default=None) The edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node.
Returns	nd : dictionary, or number A dictionary with nodes as keys and degree as values or a number if a single node is specified.

Examples	`>>> G = nx.Graph()   # or DiGraph, MultiGraph, MultiDiGraph, etc`
	`>>> G.add_path([0,1,2,3])`
	`>>> G.degree(0)`
	`1`
	`>>> G.degree([0,1])`
	`{0: 1, 1: 2}`
	`>>> list(G.degree([0,1]).values())`
	`[1, 2]`

Source	http://networkx.lanl.gov/reference/generated/networkx.Graph.degree.html

Source Code – nx_degree.py

Source : http://networkx.github.io/documentation/networkx-1.9.1/examples/drawing/degree_histogram.html

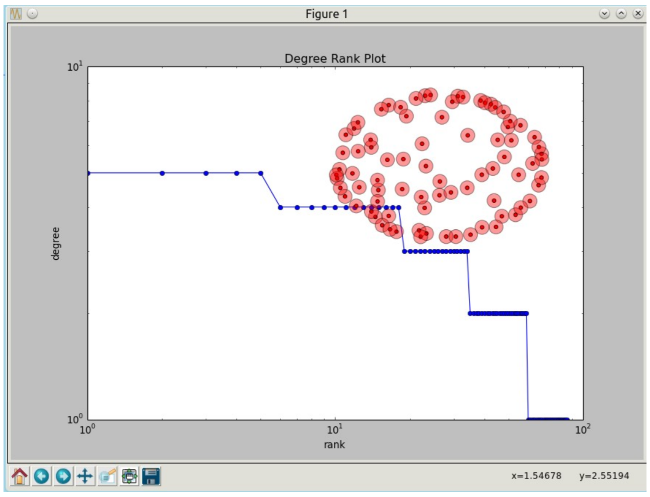
```
#!/usr/bin/env python

"""
Author      Paula Dwan
Email       paula.dwan@gmail.com
Student ID  13208660
Subject     COMP47270 (Computational Network Analysis and Modeling)
Date        12-Jan-2015
Lecturer    Dr.  Neil Hurley
```

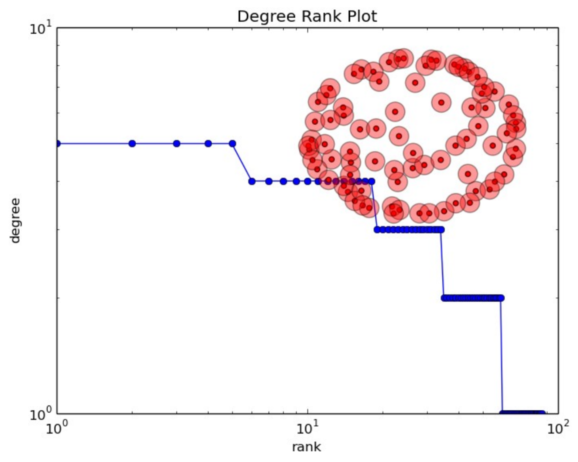
```
LABORATORY 1

Use NetworkX to compute the following for the chosen networks
    (1) degree distribution
    (2) assortativity cooefficient
    (3) clustering cooefficient
"""

import networkx as nx
import matplotlib.pyplot as plt

G = nx.gnp_random_graph(100,0.02)

degree_sequence=sorted(nx.degree(G).values(),reverse=True) # degree sequence
print "Degree Sequence = \n", degree_sequence
dmax=max(degree_sequence)

plt.loglog(degree_sequence,'b-',marker='o')
plt.title("Degree rank plot")
plt.ylabel("degree")
plt.xlabel("rank")

# draw graph in inset
plt.axes([0.45,0.45,0.45,0.45])
Gcc=sorted(nx.connected_component_subgraphs(G), key = len, reverse=True)[0]
pos=nx.spring_layout(Gcc)
plt.axis('off')
nx.draw_networkx_nodes(Gcc,pos,node_size=20)
nx.draw_networkx_edges(Gcc,pos,alpha=0.4)

plt.savefig("degree_histogram.png")
plt.show()
```

Output : Console – nx_degree.py

```
Degree Sequence =
      [8, 5, 5, 5, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
      3, 3, 3, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1,
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
      0, 0, 0, 0]
```

Output : plotted graph – nx_degree.py





2 : Assortativity Coefficient

Details – attribute_assortativity_coefficient

	attribute_assortativity_coefficient(G, attribute, nodes=None)
Explanation	Compute assortativity for node attributes. Assortativity measures the similarity of connections in the graph with respect to the given attribute.
Parameters	G : NetworkX graph Attribute : string Node attribute key nodes: list or iterable (optional) Compute attribute assortativity for nodes in container. The default is all nodes.
Returns	R: float Assortativity of graph for given attribute
Notes	This computes Eq. (2) in Ref. [R148] , trace(M)-sum(M))/(1-sum(M), where M is the joint probability distribution (mixing matrix) of the specified attribute.
References	(1, 2) M. E. J. Newman, Mixing patterns in networks, Physical Review E, 67 026126, 2003

Examples	`>>> G=nx.Graph()`
	`>>> G.add_nodes_from([0,1],color='red')`
	`>>> G.add_nodes_from([2,3],color='blue')`
	`>>> G.add_edges_from([(0,1),(2,3)])`
	`>>> print(nx.attribute_assortativity_coefficient(G,'color'))`
	`1.0`

Source	http://networkx.github.io/documentation/networkx-1.9.1/_modules/networkx/algorithms/assortativity/correlation.html#attribute_assortativity_coefficient

Details – degree_pearson_correlation_coefficient

	degree_pearson_correlation_coefficient(G, x='out', y='in', weight=None, nodes=None)
Details	Compute degree assortativity of graph. Assortativity measures the similarity of connections in the graph with respect to the node degree. This is the same as degree_assortativity_coefficient but uses the potentially faster ***scipy.stats.pearsonr*** function.
Parameters	G : NetworkX graph x: string ('in','out') : The degree type for source node (directed graphs only). y: string ('in','out') : The degree type for target node (directed graphs only). weight: string or None, optional (default=None) : The edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1. The degree is the sum of the edge weights adjacent to the node. nodes: list or iterable (optional) : Compute pearson correlation of degrees only for specified nodes. The default is all nodes.
Returns	r : float Assortativity of graph by degree.
Notes	This calls **scipy.stats.pearsonr.**

References	M. E. J. Newman, Mixing patterns in networks Physical Review E, 67 026126, 2003 Foster, J.G., Foster, D.V., Grassberger, P. & Paczuski, M. Edge direction and the structure of networks, PNAS 107, 10815-20 (2010).

Examples	`>>> G=nx.path_graph(4)`
	`>>> r=nx.degree_pearson_correlation_coefficient(G)`
	`>>> r`
	`-0.5`

Source	http://networkx.lanl.gov/reference/generated/networkx.algorithms.assortativity.degree_pearson_correlation_coefficient.html

Source Code – nx_assortativity.py

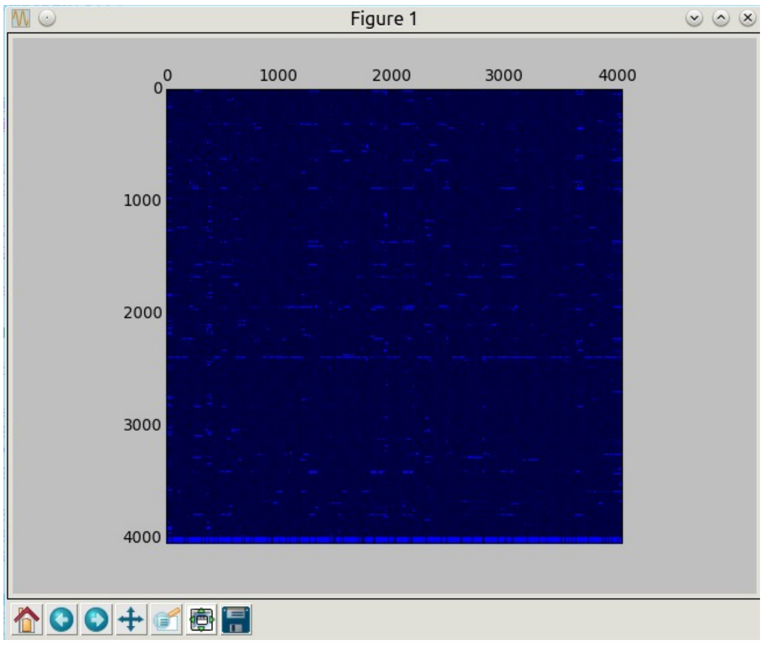
```
#!/usr/bin/env python
"""
Author      Paula Dwan
Email       paula.dwan@gmail.com
Student ID  13208660
Subject     COMP47270 (Computational Network Analysis and Modeling)
Date        12-Jan-2015
Lecturer    Dr.  Neil Hurley

LABORATORY 1

Use NetworkX to compute the following for the chosen networks
    (1) degree distribution
    (2) assortativity cooefficient
    (3) clustering cooefficient
"""

import networkx as nx
import matplotlib.pyplot as plt
import scipy as sp

G = nx.read_edgelist("../../DataSets/social_facebook_combined.txt")
A = nx.to_scipy_sparse_matrix(G);
fig = plt.figure()
plt.spy(A)
fig.show()

"""
degree_assortativity_coefficient
     Compute degree assortativity of graph.
     Assortativity measures the similarity of connections in the graph with respect
          to the node degree.
"""
r = nx.degree_assortativity_coefficient(G)
print "Degree assortativity Co-efficient = ", r

"""
degree_pearson_correlation_coefficient
     Compute degree assortativity of graph.
     Assortativity measures the similarity of connections in the graph with respect
          to the node degree.
     This is the same as degree_assortativity_coefficient but uses the potentially
          faster scipy.stats.pearsonr function.
"""
rp = nx.degree_pearson_correlation_coefficient(G)
print "Pearson's Degree assortativity Co-efficient = \n", rp

"""
average_neighbor_degree
     Returns the average degree of the neighborhood of each node.
"""
n = nx.average_neighbor_degree(G)
print "\nAverage Neighbor Degree = \n", n

"""
k_nearest_neighbors
     Compute the average degree connectivity of graph.
     The average degree connectivity is the average nearest neighbor degree of nodes
          with degree k.
"""
```

```
k = nx.k_nearest_neighbors(G)
print "\nK Nearest Neighbors = \n", k

raw_input("\nPress Enter to Continue ...\n")
```

Output : Console – nx_assortativity.py

```
Degree assortativity Co-efficient =  0.0635772291855

Pearson's Degree assortativity Co-efficient =  0.0635772291857

Average Neighbor Degree =
   {u'2031':   84.67924528301887,   u'4026':   16.333333333333332,   u'3724':   109.33333333333333,
   u'4024': 59.0, u'4025': 17.75, u'4022': 59.0, u'4023': 12.444444444444445, u'4020': 15.75, u'4021':
   15.636363636363637,      u'643':      27.625,      u'4028':      31.5,      u'4029':      30.5,
   u'344': 71.77777777777777, u'345': 63.4375,
   ...                                               ...                                               ...
   u'1364':     128.16666666666666,     u'1365':     154.5952380952381,     u'1362':     383.0,
   u'1363':   80.52380952380952,   u'1360':   53.027027027027025,   u'1361':   129.97080291970804,
   u'3728':   44.75,   u'478':   50.25,   u'479':   64.89189189189189,   u'2916':   81.14942528735632,
   u'3699': 189.33333333333334, u'1368': 114.2, u'1369': 101.24528301886792}

K Nearest Neighbors =
   {1:  478.02666666666664,  2:  260.5408163265306,  3:  203.87813620071685,  4:  142.31060606060606,
   5:  147.1290322580645,  6:  120.46088435374149,  7:  96.3862973760933,  8:  93.57882882882883,
   9:  89.59777777777778,  10:  86.40842105263158,  11:  85.54882154882155,  12:  85.67581300813008,
   13:  75.83739045764362,  14:  74.16420361247947,  15:  71.15345911949686,  16:  72.2736280487805,
   17:  73.00773993808049,  18:  65.89421613394217,  19:  69.64473684210526,  20:  68.70555555555555,
   1045: 54.985645933014354, 22: 73.17388167388168,
   ...  ...  ...
   231:     59.62337662337662,     234:     135.26709401709402,     235:     120.95744680851064,
   211:     123.09004739336493,     245:     118.56326530612245,     127:     125.74803149606299,
   254:     117.35433070866142,     792:     37.946969696969695,     291:     145.62199312714776,
   294: 140.2721088435374, 347: 18.959654178674352, 755: 80.93245033112582}

Press Enter to Continue ...
```

Output : Plotted Graph – nx_assortativity.py



3 : Clustering coefficient

Details – clustering

	clustering(G, nodes=None, mode='dot')
Explanation	Compute the clustering coefficient for nodes. For **unweighted** graphs, the clustering of a node u is the fraction of possible triangles through that node that exist, $c_u = \frac{2T(u)}{deg(u)(deg(u)-1)}$, where ***T(u)*** is the number of triangles through node ***u*** and ***deg(u)*** is the degree of ***u.*** For **weighted** graphs, the clustering is defined as the geometric average of the subgraph edge weights, $c_u = \frac{1}{deg(u)(deg(u)-1))}\sum_{uv}(\hat{w}_{uv}\hat{w}_{uw}\hat{w}_{vw})^{1/3}$ The edge weight $\hat{w}_{uv}$ are normalized by the maximum weight in the network $\hat{w}_{uv} = w_{uv}/\max(w)$. The value of ***cu*** is assigned to 0 if ***deg(u)<2***.
Parameters	**G :** graph **nodes :** container of nodes, optional (default=all nodes in G) Compute clustering for nodes in this container. **weight** string or None, optional (default=None) The edge attribute that holds the numerical value used as a weight. If None, then each edge has weight 1.
Returns	**out :** float, or dictionary Clustering coefficient at specified nodes
Notes	Self loops are ignored.
References	Generalizations of the clustering coefficient to weighted complex networks by J. Saramäki, M. Kivelä, J.-P. Onnela, K. Kaski, and J. Kertész, Physical Review E, 75 027105 (2007). http://jponnela.com/web_documents/a9.pdf

Examples	`>>> G=nx.complete_graph(5)`
	 `>>> print(nx.clustering(G,0))`
	 `1.0`
	 `>>> print(nx.clustering(G))`
	 `{0: 1.0, 1: 1.0, 2: 1.0, 3: 1.0, 4: 1.0}`

Source	https://networkx.github.io/documentation/latest/reference/generated/networkx.algorithms.cluster.clustering.html

Details – average_clustering

	`average_clustering`(*G, trials=1000*)
Explanation	Estimates the average clustering coefficient of G. The local clustering of each node in G is the fraction of triangles that actually exist over all possible triangles in its neighborhood. The average clustering coefficient of a graph G is the mean of local clusterings. This function finds an approximate average clustering coefficient for G by repeating n times (defined in trials) the following experiment: choose a node at random, choose two of its neighbors at random, and check if they are connected. The approximate coefficient is the fraction of triangles found over the number of trials.
Parameters	**G** : NetworkX graph **trials** : integer Number of trials to perform (default 1000).
Returns	**c** : float Approximated average clustering coefficient.
References	Schank, Thomas, and Dorothea Wagner. Approximating clustering coefficient and transitivity. Universität Karlsruhe, Fakultät für Informatik, 2004. http://www.emis.ams.org/journals/JGAA/accepted/2005/SchankWagner2005.9.2.pdf

Examples	`>>> G=nx.complete_graph(5)`
	`>>> print(nx.average_clustering(G))`
	`1.0`

Source	https://networkx.github.io/documentation/latest/reference/generated/networkx.algorithms.approximation.clustering_coefficient.average_clustering.html

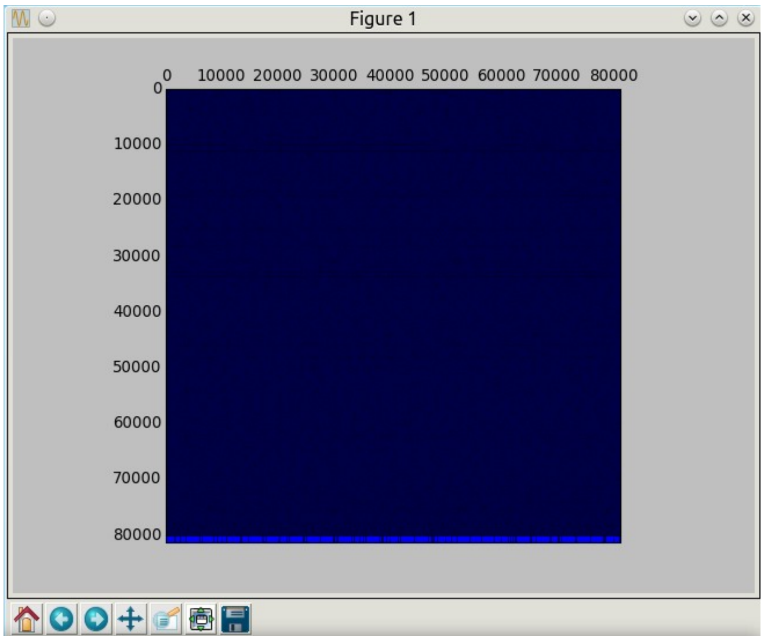
Source Code – nx_clustering.py

```
#!/usr/bin/python

"""
Author        Paula Dwan
Email         paula.dwan@gmail.com
Student ID    13208660
Subject       COMP47270 (Computational Network Analysis and Modeling)
Date          12-Jan-2015
Lecturer      Dr.  Neil Hurley

LABORATORY 1

Use NetworkX to compute the following for the chosen networks
    (1) degree distribution
    (2) assortativity cooefficient
    (3) clustering cooefficient
"""

import networkx as nx
import matplotlib.pyplot as plt
import scipy as sp

src_file = "../../DataSets/social_twitter_combined.txt"
# src_file = "../../DataSets/social_facebook_combined.txt"
# src_file = "../../DataSets/social_gplus_combined.txt" not working also
# src_file = "../../DataSets/amazon_finefoods.txt"

"""
ERROR :
   Traceback (most recent call last):
      File "nx_clustering.py", line 27, in <module>
         G = nx.read_edgelist(src_file)
      File "<string>", line 2, in read_edgelist
      File "/usr/local/lib/python2.7/dist-packages/networkx-1.9.1-
         py2.7.egg/networkx/utils/decorators.py", line 220, in _open_file
         result = func(*new_args, **kwargs)
      File "/usr/local/lib/python2.7/dist-packages/networkx-1.9.1-
         py2.7.egg/networkx/readwrite/edgelist.py", line 369, in read_data=data)
      File "/usr/local/lib/python2.7/dist-packages/networkx-1.9.1-
         py2.7.egg/networkx/readwrite/edgelist.py", line 280, in parse_edgelist
         "Failed to convert edge data (%s) to dictionary."%(d))
   TypeError: Failed to convert edge data ([u'Quality', u'Dog', u'Food']) to dictionary.
"""

G = nx.read_edgelist(src_file)

print "\nGraph Source file = \n", src_file

cce = nx.clustering(G)
print "\nClustering Coefficients for graph = \n", cce

average_cce = nx.average_clustering(G)
print "\nAverage Clustering  Coefficient  for graph = ", average_cce

A = nx.to_scipy_sparse_matrix(G)
fig = plt.figure()
plt.spy(A)
fig.show()

raw_input("\nPress Enter to Continue ...\n")
```

Output : Console – nx_clustering.py

```
Graph Source file =
      ../../DataSets/social_twitter_combined.txt

Clustering Coefficients for graph =
      {u'19609952':    1.0,    u'75607671':    0.2762128325508607,    u'37722200':    0.42610837438423643,
      u'84200495':         0.34233814625058223,         u'164021756':         0.5757575757575758,
      u'14536491':  0.1114757120471933,  u'8021722':  0.6401480111008325,  u'11541':  0.6583333333333333,
      u'18014014':    1.0,    u'26198655':    0.42431972789115646,    u'243370113':    0.5721997300944669,
      u'170922633': 0.6761363636363636, u'19401051': 0.7333333333333333,
      ...   ...     ...
      u'41146726':         0.43859649122807015,         u'1018921':         0.4927536231884058,
      u'14130368':         0.7472527472527473,         u'257372581':         0.4240376138701146,
      u'115072199':    0.7435897435897436,    u'20495174':    0.0,    u'18470775':    0.34254881808838644,
      u'81982975': 1.0, u'242894622': 0.5357142857142857, u'78419512': 0.6666666666666666, u'41741418':
      0.6, u'332531469': 0.6785714285714286, u'32812397': 0.37777777777777777}

Average Clustering Coefficient for graph = 0.565311468612

Press Enter to Continue ...
```

Output : Plotted Graph – nx_clustering.py

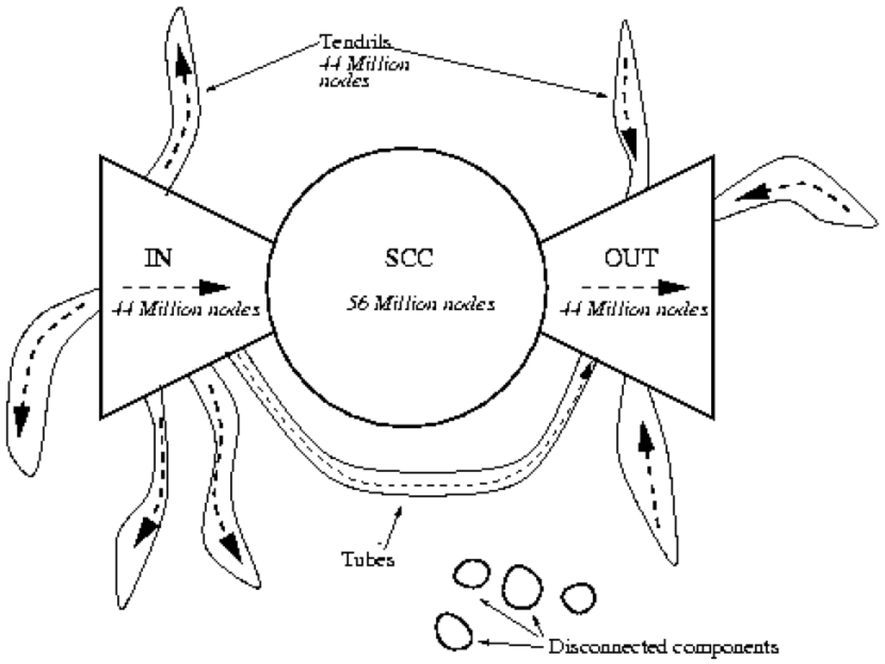


LABORATORY 1 – CASE STUDY

Determine the qualitative nature of the networks you are studying and write up a report.

Do some of the following:

1. For the directed networks, sketch the **Broder et al** picture of the network – number of nodes in strongest connected components and In the In and Out and other sections of the network.



2. Fit a line to a log log plot of the degree distribution – compute the slope of this line to determine the parameter of the power-law degree distribution model.
3. Simulate the Price model to generate a network of n nodes for a particular power-law parameter.
4. Compute all of the above parameters for the Price model. Which networks does the Price model best represent

1 : BRODER ET AL DIAGRAM

DETAILS : SCC (STRONGLY_CONNECTED_COMPONENTS)

	strongly_connected_components(*G*)
Information	Generate nodes in strongly connected components of graph
Parameters:	G : NetworkX Graph An directed graph.
Returns:	comp : generator of lists A list of nodes for each strongly connected component of G.
Raises:	NetworkXNotImplemented: If G is undirected.
Notes	Uses Tarjan's algorithm with Nuutila's modifications. Nonrecursive version of algorithm.
References	Depth-first search and linear graph algorithms, R. Tarjan SIAM Journal of Computing 1(2):146-160, (1972). On finding the strongly connected components in a directed graph. E. Nuutila and E. Soisalon-Soinen Information Processing Letters 49(1): 9-14, (1994)..
Source	https://networkx.github.io/documentation/latest/reference/algorithms.component.html https://networkx.github.io/documentation/latest/reference/generated/networkx.algorithms.components.strongly_connected.strongly_connected_components.html

Details : number_strongly_connected_components

	number_strongly_connected_components(G)
Information	Return number of strongly connected components in graph.
Parameters	G : NetworkX graph A directed graph.
Returns	n : integer Number of strongly connected components
Source	https://networkx.github.io/documentation/latest/reference/generated/networkx.algorithms.components.strongly_connected.number_strongly_connected_components.html

2 : *Log plot :*

Details – degree_histogram

	degree_histogram(*G*)
Information	Return a list of the frequency of each degree value.
Parameters	G : NetworkX graph A graph.
Returns	**hist** : list A list of frequencies of degrees. The degree values are the index in the list.
Notes	The bins are width one, hence len(list) can be large (Order(number_of_edges))
Source	https://networkx.github.io/documentation/latest/reference/generated/networkx.classes.function.degree_histogram.html

In- degree distributions

Out- degree distributions

3 : *Price model for power-law parameters*

4 : *Price model for all parameters*

References

http://arxiv.org/pdf/1011.1533.pdf

```
import numpy as np
def drop_zeros(a_list):
    return [i for i in a_list if i>0]

def log_binning(counter_dict,bin_count=35):

    max_x = log10(max(counter_dict.keys()))
    max_y = log10(max(counter_dict.values()))
    max_base = max([max_x,max_y])
    min_x = log10(min(drop_zeros(counter_dict.keys())))

    bins = np.logspace(min_x,max_base,num=bin_count)

    # Based on : http://stackoverflow.com/questions/6163334/binning-data-in-python-with-scipy-numpy
    bin_means_y =
        (np.histogram(counter_dict.keys(),bins,weights=counter_dict.values())[0] /
        np.histogram(counter_dict.keys(),bins)[0])
    bin_means_x =
        (np.histogram(counter_dict.keys(),bins,weights=counter_dict.keys())[0] /
         np.histogram(counter_dict.keys(),bins)[0])
    return bin_means_x,bin_means_y
```

```
import networkx as nx
ba_g = nx.barabasi_albert_graph(10000,2)
ba_c = nx.degree_centrality(ba_g)
# To convert normalized degrees to raw degrees
# ba_c = {k:int(v*(len(ba_g)-1)) for k,v in ba_c.iteritems()}
ba_c2 = dict(Counter(ba_c.values()))

ba_x,ba_y = log_binning(ba_c2,50)

plt.xscale('log')
plt.yscale('log')
plt.scatter(ba_x,ba_y,c='r',marker='s',s=50)
plt.scatter(ba_c2.keys(),ba_c2.values(),c='b',marker='x')
plt.xlim((1e-4,1e-1))
plt.ylim((.9,1e4))
plt.xlabel('Connections (normalized)')
plt.ylabel('Frequency')
plt.show()
```

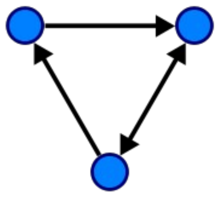
REFERENCES

OVERVIEW

Datasets : http://snap.stanford.edu/data/index.html

NETWORK TYPES

- **Directed** : directed network (http://en.wikipedia.org/wiki/Directed_graph)



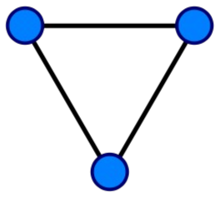
In mathematics, and more specifically in graph theory, a directed graph (or digraph) is a graph, or set of nodes connected by edges, where the edges have a direction associated with them. In formal terms, a digraph is a pair G=(V,A) (sometimes G=(V,E)) of:[1]

  - a set V, whose elements are called vertices or nodes,
  - a set A of ordered pairs of vertices, called arcs, directed edges, or arrows (and sometimes simply edges with the corresponding set named E instead of A).

It differs from an ordinary or undirected graph, in that the latter is defined in terms of unordered pairs of vertices, which are usually called edges.

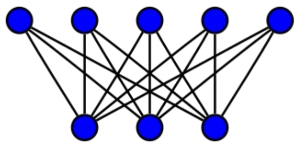
A digraph is called "simple" if it has no loops, and no multiple arcs (arcs with same starting and ending nodes). A directed multigraph, in which the arcs constitute a multiset, rather than a set, of ordered pairs of vertices may have loops (that is, "self-loops" with same starting and ending node) and multiple arcs. Some, but not all, texts allow a digraph, without the qualification simple, to have self loops, multiple arcs, or both.

- **Undirected** : undirected network (http://en.wikipedia.org/wiki/Graph_(mathematics)#Undirected_graph)



An undirected graph is one in which edges have no orientation. The edge (a, b) is identical to the edge (b, a), i.e., they are not ordered pairs, but sets {u, v} (or 2-multisets) of vertices. The maximum number of edges in an undirected graph without a self-loop is n(n - 1)/2.

- **Bipartite** : bipartite network (http://en.wikipedia.org/wiki/Bipartite_graph)



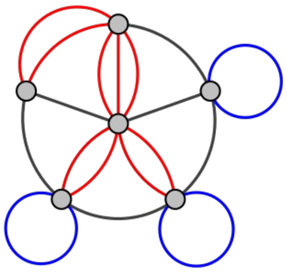
A complete bipartite graph with m = 5 and n = 3

In the mathematical field of graph theory, a bipartite graph (or bigraph) is a graph whose vertices can be divided into two disjoint sets U and V (that is, U and V are each independent sets) such that every edge connects a vertex in U to one in V. Vertex set U and V are often denoted as partite sets. Equivalently, a bipartite graph is a graph that does not contain any odd-length cycles.[1][2]

The two sets U and V may be thought of as a coloring of the graph with two colors: if one colors all nodes in U blue, and all nodes in V green, each edge has endpoints of differing colors, as is required in the graph coloring problem.[3][4] In contrast, such a coloring is impossible in the case of a non-bipartite graph, such as a triangle: after one node is colored blue and another green, the third vertex of the triangle is connected to vertices of both colors, preventing it from being assigned either color.

One often writes G=(U,V,E) to denote a bipartite graph whose partition has the parts U and V, with E denoting the edges of the graph. If a bipartite graph is not connected, it may have more than one bipartition;[5] in this case, the (U,V,E) notation is helpful in specifying one particular bipartition that may be of importance in an application. If |U|=|V|, that is, if the two subsets have equal cardinality, then G is called a balanced bipartite graph.[3] If all vertices on the same side of the bipartition have the same degree, then G is called biregular.

- **Multigraph** : network has multiple edges between a pair of nodes (http://en.wikipedia.org/wiki/Multigraph)



A multigraph with multiple edges (red) and several loops (blue).

Not all authors allow multigraphs to have loops.

In mathematics, and more specifically in graph theory, a multigraph is a graph which is permitted to have multiple edges (also called parallel edges[1]), that is, edges that have the same end nodes. Thus two vertices may be connected by more than one edge.

There are two distinct notions of multiple edges:

- Edges without own identity: The identity of an edge is defined solely by the two nodes it connects. In this case, the term "multiple edges" means that the same edge can occur several times between these two nodes.
- Edges with own identity: Edges are primitive entities just like nodes. When multiple edges connect two nodes, these are different edges.

A multigraph is different from a hypergraph, which is a graph in which an edge can connect any number of nodes, not just two.

For some authors, the terms pseudograph and multigraph are synonymous. For others, a pseudograph is a multigraph with loops.

- **Temporal** : for each node/edge we know the time when it appeared in the network

- **Labeled** : network contains labels (weights, attributes) on nodes and/or edges

Network statistics

Dataset statistics	
Nodes	Number of nodes in the network
Edges	Number of edges in the network
Nodes in largest WCC	Number of nodes in the largest weakly connected component
Edges in largest WCC	Number of edges in the largest weakly connected component
Nodes in largest SCC	Number of nodes in the largest strongly connected component
Edges in largest SCC	Number of edges in the largest strongly connected component
Average clustering coefficient	Average clustering coefficient In graph theory, a clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. Evidence suggests that in most real-world networks, and in particular social networks, nodes tend to create tightly knit groups characterised by a relatively high density of ties; this likelihood tends to be greater than the average probability of a tie randomly established between two nodes (Holland and Leinhardt, 1971;[1] Watts and Strogatz, 1998[2]). Two versions of this measure exist: the global and the local. The global version was designed to give an overall indication of the clustering in the network, whereas the local gives an indication of the embeddedness of single nodes.
Number of triangles	Number of triples of connected nodes (considering the network as undirected)
Fraction of closed triangles	Number of connected triples of nodes / number of (undirected) length 2 paths
Diameter (longest shortest path)	Maximum undirected shortest path length (sampled over 1,000 random nodes)
90-percentile effective diameter	90th percentile of undirected shortest path length distribution (sampled over 1,000 random nodes)

Social circles: Facebook (Social / Undirected)

Dataset information

http://snap.stanford.edu/data/egonets-Facebook.html

This dataset consists of 'circles' (or 'friends lists') from Facebook. Facebook data was collected from survey participants using this Facebook app. The dataset includes node features (profiles), circles, and ego networks.

Facebook data has been anonymized by replacing the Facebook-internal ids for each user with a new value. Also, while feature vectors from this dataset have been provided, the interpretation of those features has been obscured. For instance, where the original dataset may have contained a feature "political=Democratic Party", the new data would simply contain "political=anonymized feature 1". Thus, using the anonymized data it is possible to determine whether two users have the same political affiliations, but not what their individual political affiliations represent.

Data is also available from Google+ and Twitter.

Dataset statistics	
Nodes	4039
Edges	88234
Nodes in largest WCC	4039 (1.000)
Edges in largest WCC	88234 (1.000)
Nodes in largest SCC	4039 (1.000)
Edges in largest SCC	88234 (1.000)
Average clustering coefficient	0.6055
Number of triangles	1612010
Fraction of closed triangles	0.2647
Diameter (longest shortest path)	8
90-percentile effective diameter	4.7

Note that these statistics were compiled by combining the ego-networks, including the ego nodes themselves (along with an edge to each of their friends).

Source (citation)

J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

Files

File	Description
facebook.tar.gz	Facebook data (10 networks, anonymized)
facebook_combined.txt.gz	Edges from all egonets combined
readme-Ego.txt	Description of files

Social circles: Google+ (Social / Directed)

Dataset information

http://snap.stanford.edu/data/egonets-Gplus.html

This dataset consists of 'circles' from Google+. Google+ data was collected from users who had manually shared their circles using the 'share circle' feature. The dataset includes node features (profiles), circles, and ego networks.

Data is also available from Facebook and Twitter.

Dataset statistics	
Nodes	107614
Edges	13673453
Nodes in largest WCC	107614 (1.000)
Edges in largest WCC	13673453 (1.000)
Nodes in largest SCC	69501 (0.646)
Edges in largest SCC	9168660 (0.671)

Average clustering coefficient	0.4901
Number of triangles	1073677742
Fraction of closed triangles	0.6552
Diameter (longest shortest path)	6
90-percentile effective diameter	3

Source (citation)

J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

Files

File	Description
gplus.tar.gz	Google+ (132 networks)
gplus_combined.txt.gz	Edges from all egonets combined
readme-Ego.txt	Description of files

Social circles: Twitter (Social / Directed)

Dataset information

http://snap.stanford.edu/data/egonets-Twitter.html

This dataset consists of 'circles' (or 'lists') from Twitter. Twitter data was crawled from public sources. The dataset includes node features (profiles), circles, and ego networks.

Data is also available from Facebook and Google+.

Dataset statistics	
Nodes	81306
Edges	1768149
Nodes in largest WCC	81306 (1.000)
Edges in largest WCC	1768149 (1.000)
Nodes in largest SCC	68413 (0.841)
Edges in largest SCC	1685163 (0.953)
Average clustering coefficient	0.5653
Number of triangles	13082506
Fraction of closed triangles	0.06415
Diameter (longest shortest path)	7
90-percentile effective diameter	4.5

Source (citation)

J. McAuley and J. Leskovec. Learning to Discover Social Circles in Ego Networks. NIPS, 2012.

Files

File	Description
twitter.tar.gz	Twitter data (973 networks)
twitter_combined.txt.gz	Edges from all egonets combined
readme-Ego.txt	Description of files

Web data: Amazon Fine Foods reviews (Informational / online)

Dataset information

http://snap.stanford.edu/data/web-FineFoods.html

This dataset consists of reviews of fine foods from amazon. The data span a period of more than 10 years, including all ~500,000 reviews up to October 2012. Reviews include product and user information, ratings, and a plaintext review. We also have reviews from all other Amazon categories.

Dataset statistics	
Number of reviews	568,454
Number of users	256,059
Number of products	74,258
Users with > 50 reviews	260
Median no. of words per review	56
Timespan	Oct 1999 - Oct 2012

Source (citation)

J. McAuley and J. Leskovec. From amateurs to connoisseurs: modelling# the evolution of user expertise through online reviews. WWW, 2013.

Files

File	Description
finefoods.txt.gz	Amazon Fine Foods data (~500,000 reviews)

Data format

```
product/productId:   B001E4KFG0
review/userId:       A3SGXH7AUHU8GW
review/profileName:  delmartian
review/helpfulness:  1/1
review/score:        5.0
review/time:         1303862400
review/summary:      Good Quality Dog Food review/text: I have bought several of the Vitality canned
                     dog food products and have found them all to be of good quality.  The product
                     looks more like a stew than a processed meat and it smells better.  My Labrador
                     is finicky and she appreciates this product better than most.
```

where

- product/productId: asin, e.g. amazon.com/dp/B001E4KFG0
- review/userId: id of the user, e.g. A3SGXH7AUHU8GW
- review/profileName: name of the user
- review/helpfulness: fraction of users who found the review helpful
- review/score: rating of the product
- review/time: time of the review (unix time)
- review/summary: review summary
- review/text: text of the review

Web data: Amazon movies reviews (Informational / online)

Dataset information

http://snap.stanford.edu/data/web-Movies.html

This dataset consists of movie reviews from amazon. The data span a period of more than 10 years, including all ~8 million reviews up to October 2012. Reviews include product and user information, ratings, and a plaintext review. We also have reviews from all other Amazon categories.

Dataset statistics	
Number of reviews	7,911,684
Number of users	889,176
Number of products	253,059
Users with > 50 reviews	16,341
Median no. of words per review	101
Timespan	Aug 1997 - Oct 2012

Source (citation)

J. McAuley and J. Leskovec. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. WWW, 2013.

Files

File	Description
movies.txt.gz	Amazon movie data (~8 million reviews)

Data format

```
product/productId:   B00006HAXW
review/userId:       A1RSDE90N6RSZF
review/profileName:  Joseph M.  Kotow
review/helpfulness:  9/9
review/score:        5.0
review/time:         1042502400
review/summary:      Pittsburgh - Home of the OLDIES review/text: I have all of the doo wop DVD's
                     and this one is as good or better than the 1st ones.  Remember once these
                     performers are gone, we'll never get to see them again.  Rhino did an excellent
                     job and if you like or love doo wop and Rock n Roll you'll LOVE this DVD !!
```

where

- •product/productId: asin, e.g. amazon.com/dp/B00006HAXW
- •review/userId: id of the user, e.g. A1RSDE90N6RSZF
- •review/profileName: name of the user
- •review/helpfulness: fraction of users who found the review helpful
- •review/score: rating of the product
- •review/time: time of the review (unix time)
- •review/summary: review summary
- •review/text: text of the review

California road network (Technological / Undirected)

Dataset information

http://snap.stanford.edu/data/roadNet-CA.html

A road network of California. Intersections and endpoints are represented by nodes and the roads connecting these intersections or road endpoints are represented by undirected edges.

Dataset statistics	
Nodes	1965206
Edges	2766607
Nodes in largest WCC	1957027 (0.996)
Edges in largest WCC	2760388 (0.998)
Nodes in largest SCC	1957027 (0.996)
Edges in largest SCC	2760388 (0.998)
Average clustering coefficient	0.0464
Number of triangles	120676
Fraction of closed triangles	0.02097
Diameter (longest shortest path)	849
90-percentile effective diameter	5e+02

Source (citation)

J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29--123, 2009.

Files

File	Description
roadNet-CA.txt.gz	California road network

Pennsylvania road network (Technological / Undirected)

Dataset information

http://snap.stanford.edu/data/roadNet-PA.html

This is a road network of Pennsylvania. Intersections and endpoints are represented by nodes, and the roads connecting these intersections or endpoints are represented by undirected edges.

Dataset statistics	
Nodes	1088092
Edges	1541898
Nodes in largest WCC	1087562 (1.000)
Edges in largest WCC	1541514 (1.000)
Nodes in largest SCC	1087562 (1.000)
Edges in largest SCC	1541514 (1.000)
Average clustering coefficient	0.0465
Number of triangles	67150
Fraction of closed triangles	0.02062
Diameter (longest shortest path)	786
90-percentile effective diameter	5.3e+02

Source (citation)

J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29--123, 2009.

Files

File	Description
roadNet-PA.txt.gz	Pennsylvania road network

Texas road network (Technological / Undirected)

Dataset information

http://snap.stanford.edu/data/roadNet-TX.html

This is a road network of Texas. Intersections and endpoints are represented by nodes, and the roads connecting these intersections or endpoints are represented by undirected edges.

Dataset statistics	
Nodes	1379917
Edges	1921660
Nodes in largest WCC	1351137 (0.979)
Edges in largest WCC	1879201 (0.978)
Nodes in largest SCC	1351137 (0.979)
Edges in largest SCC	1879201 (0.978)
Average clustering coefficient	0.0470
Number of triangles	82869
Fraction of closed triangles	0.02091
Diameter (longest shortest path)	1054
90-percentile effective diameter	6.7e+02

Source (citation)

J. Leskovec, K. Lang, A. Dasgupta, M. Mahoney. Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters. Internet Mathematics 6(1) 29--123, 2009.

Files

File	Description
roadNet-TX.txt.gz	Texas road network

Citing SNAP

We encourage you to cite our datasets if you have used them in your work.

You can use the following BibTeX citation:

```
@misc{snapnets,
  author       = {Jure Leskovec and Andrej Krevl},
  title        = {{SNAP Datasets}: {Stanford} Large Network Dataset Collection},
  howpublished = {\url{http://snap.stanford.edu/data}},
  month        = jun,
  year         = 2014
}
```