Network Analysis with Python and NetworkX:

The Enron Case Study

Jukka-Pekka “JP” Onnela

Harvard University

ICPSR Summer Workshop

Friday, June 24, 2011

Enron email network

- Enron email communication network covers consists of half a million emails
- This data was originally made public, and posted to the web, by the Federal Energy Regulatory Commission during its investigation
- Nodes of the network are email addresses
- If person i sent at least one email to person j, they are connected by an undirected edge
- Only emails sent within Enron are visible in this data set
- Some basic statistics: Number of nodes: 36,692; Number of edges 367,662 (double-counted)
- Data can be downloaded from `http://snap.stanford.edu/data/email-Enron.html`

Friday, June 24, 2011

Enron email network

(1) Write your own customized routine that reads in the network data as an edge list. Allow for arbitrary comment line characters (e.g. # or $) and arbitrary separators (e.g. “,” or “\t”).

Enron email network

(1) Write your own customized routine that reads in the network data as an edge list. Allow for arbitrary comment line characters (e.g. # or $) and arbitrary separators (e.g. “,” or “\t”).

```
# Routine for reading in a network as an edge list.
def readnet(input_file, sep_char, comment_char):
    G = nx.Graph()
    for line in open(input_file):
        if line[0] != comment_char:
            line = line.rstrip().split(sep_char)
            G.add_edge(int(line[0]), int(line[1]))
    return G
```

Enron email network

(1) Write your own customized routine that reads in the network data as an edge list. Allow for arbitrary comment line characters (e.g. # or $) and arbitrary separators (e.g. “,” or “\t”).

```
# Routine for reading in a network as an edge list.
def readnet(input_file, sep_char, comment_char):
    num_lines = 0
    G = nx.Graph()
    for line in open(input_file):
        num_lines += 1
        if line[0] != comment_char:
            line = line.rstrip().split(sep_char)
            if len(line) == 2:
                G.add_edge(int(line[0]), int(line[1]))
    print 'Read ' + str(num_lines) + ' lines.'
    print 'Network has %d nodes and %d edges.' % (G.number_of_nodes(), G.number_of_edges())
    return G
```


Enron email network

(1) Write your own customized routine that reads in the network data as an edge list. Allow for arbitrary comment line characters (e.g. # or $) and arbitrary separators (e.g. “,” or “\t”).

```
# Routine for reading in a network as an edge list.
def readnet(input_file, sep_char, comment_char):
    num_lines = 0
    G = nx.Graph()
    for line in open(input_file):
        num_lines += 1
        if line[0] != comment_char:
            line = line.rstrip().split(sep_char)
            if len(line) == 2:
                G.add_edge(int(line[0]), int(line[1]))
                if num_lines < 10:
                    print line[0] + " " + line[1]
                if num_lines == 10:
                    print "..."
    print 'Read ' + str(num_lines) + ' lines.'
    print 'Network has %d nodes and %d edges.' % (G.number_of_nodes(), G.number_of_edges())
    return G
```

Enron email network

(2) Write your own customized routine that writes network data as an edge list. Allow for arbitrary separators (e.g. “,” or “\t”).

Enron email network

(2) Write your own customized routine that writes network data as an edge list. Allow for arbitrary separators (e.g. “,” or “\t”).

```
# Routine for writing a network as an edge list.
def writenet(G, output_file, sep_char):
    num_lines = 0
    F = open(output_file, 'w')
    for edge in G.edges():
        F.write(str(edge[0]) + sep_char + str(edge[1]) + '\n')
        num_lines += 1
    F.close()
    print 'Wrote ' + str(num_lines) + ' lines.'
```

Enron email network

(3) Write a function that returns a sociocentric sample on a given list of nodes.

Enron email network

(3) Write a function that returns a sociocentric sample on a given list of nodes.

```
def sociocentric_sample(G, nodes):
    g = nx.subgraph(G, nodes)
    return g
```

Enron email network

(4) Write a function that returns of a set of nodes chosen uniformly at random from the network.

Enron email network

(4) Write a function that returns of a set of nodes chosen uniformly at random from the network.

```
def random_node_selection(G, num_sample_nodes):
    if num_sample_nodes > G.number_of_nodes():
        print 'Sample size exceeds population size.'
        return None
    else:
        sample_nodes = set()
        while len(sample_nodes) < num_sample_nodes:
            sample_nodes.add(random.choice(G.nodes()))
        return sample_nodes
```

PS. You may wish to return a list of nodes by converting the set to list:

```
>>> return list(sample_nodes)
```

Enron email network

(5) Write a function that returns a snowball sample of the nodes of a network (i.e. no edges, just the nodes). The algorithm should start from a given node, and it should stop when it reaches a pre-specified layer, i.e. when it reaches a certain distance from the source node.

Enron email network

(5) Write a function that returns a snowball sample of the nodes of a network (i.e. no edges, just the nodes). The algorithm should start from a given node, and it should stop when it reaches a pre-specified layer, i.e. when it reaches a certain distance from the source node.

```
def snowball_sample(G, node, curr_layer, max_layer):
    global sample_nodes
    sample_nodes.add(node)
    if curr_layer < max_layer:
        for neighbor in G.neighbors(node):
            snowball_sample(G, neighbor, curr_layer + 1, max_layer)
```

Enron email network

(5) Write a function that returns a snowball sample of the nodes of a network (i.e. no edges, just the nodes). The algorithm should start from a given node, and it should stop when it reaches a pre-specified layer, i.e. when it reaches a certain distance from the source node.

```
def snowball_sample_opt(G, node, parent, curr_layer, max_layer):
    global sample_nodes
    sample_nodes.add(node)
    new_neighbors = G.neighbors(node)
    if parent in new_neighbors:
        new_neighbors.remove(parent)
    if curr_layer < max_layer:
        for neighbor in new_neighbors:
            snowball_sample_opt(G, neighbor, node, curr_layer + 1, max_layer)
```

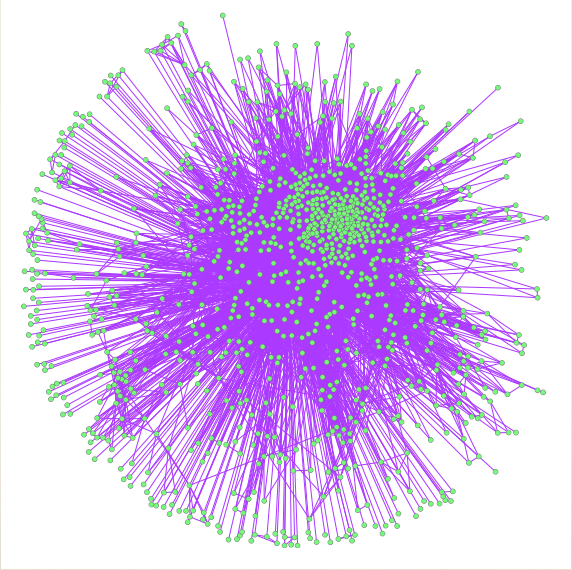

Enron email network

(6) Generate a snowball sample of the network, starting from a randomly chosen source node, and choose the stopping condition such that approximately 1,000 nodes are included in the sample. Write the network to file as an edge list and visualize the sample using Cytoscape.

```
# Visualize a snowball sample of the Enron email network.
# JP Onnela / June 16 2011
#
# Usage: python vis_enron.py

import networkx as nx
import random

def readnet(input_file, sep_char, comment_char):
    G = nx.Graph()
    for line in open(input_file):
        if line[0] != comment_char:
            line = line.rstrip().split(sep_char)
            G.add_edge(int(line[0]), int(line[1]))
    return G

def writenet(G, output_file, sep_char):
    F = open(output_file, 'w')
    for edge in G.edges():
        F.write(str(edge[0]) + sep_char + str(edge[1]) + '\n')
    F.close()

def sociocentric_sample(G, nodes):
    g = nx.subgraph(G, nodes)
    return g

def snowball_sample(G, node, curr_layer, max_layer):
    global sample_nodes
    sample_nodes.add(node)
    if curr_layer < max_layer:
        for neighbor in G.neighbors(node):
            snowball_sample(G, neighbor, curr_layer + 1, max_layer)

G = readnet("email-Enron.txt", "\t", "#")
sample_nodes = set()
snowball_sample(G, random.choice(G.nodes()), 0, 2)
H = sociocentric_sample(G, list(sample_nodes))
print H.number_of_nodes()
print H.number_of_edges()
writenet(H, 'sample_net.txt', ' ')
```


Enron email network

(6) Generate a snowball sample of the network, starting from a randomly chosen source node, and choose the stopping condition such that approximately 1,000 nodes are included in the sample. Write the network to file as an edge list and visualize the sample using Cytoscape.



Enron email network

(7) Compute some basic network characteristics (e.g. average or median degree, clustering coefficient, betweenness centrality, number of components, etc.) for the entire network as well as for samples drawn from the network.

Enron email network

(8a) See what happens to the network, quantified using the above measures, as you delete some fraction of the nodes. The deleted nodes should be chosen uniformly at random, i.e. each node should have the same probability to be chosen for deletion. This process is sometimes called “random failure” of nodes.

(8b) Do the same as above, but this time let the selection probability depend on the degree of the node. More specifically, make that probability linearly dependent on node degree. For example, a node of degree 6 should be twice as likely to be chosen for deletion than a node of degree 3. This is sometimes called “targeted attack” of nodes.

Enron email network

(8a) See what happens to the network, quantified using the above measures, as you delete some fraction of the nodes. The deleted nodes should be chosen uniformly at random, i.e. each node should have the same probability to be chosen for deletion. This process is sometimes called “random failure” of nodes.

```
def random_node_selection(G, num_sample_nodes):
    if num_sample_nodes > G.number_of_nodes():
        print 'Sample size exceeds population size.'
        return None
    else:
        sample_nodes = set()
        while len(sample_nodes) < num_sample_nodes:
            sample_nodes.add(random.choice(G.nodes()))
        return sample_nodes
```


Enron email network

(8b) Do the same as above, but this time let the selection probability depend on the degree of the node. More specifically, make that probability linearly dependent on node degree. For example, a node of degree 6 should be twice as likely to be chosen for deletion than a node of degree 3. This is sometimes called “targeted attack” of nodes.

```
def preferential_node_selection(G, num_sample_nodes):
    if num_sample_nodes > G.number_of_nodes():
        print 'Sample size exceeds population size.'
        return None
    else:
        sample_nodes = set()
        master_list = []
        for (k,v) in G.degree().items():
            master_list.extend([k for x in range(v)])
        while len(sample_nodes) < num_sample_nodes:
            sample_nodes.add(random.choice(master_list))
        return sample_nodes
```

Enron email network

(8b) Do the same as above, but this time let the selection probability depend on the degree of the node. More specifically, make that probability linearly dependent on node degree. For example, a node of degree 6 should be twice as likely to be chosen for deletion than a node of degree 3. This is sometimes called “targeted attack” of nodes.

```
>>> G = nx.barabasi_albert_graph(100000, 2)
>>> numpy.mean(G.degree().values())
--> 3.9999199999999999
>>> sample_nodes = preferential_node_selection(G, 100)
>>> numpy.mean(G.degree(sample_nodes).values())
--> 20.210000000000001
>>> sample_nodes = preferential_node_selection(G, 1000)
>>> numpy.mean(G.degree(sample_nodes).values())
--> 13.425000000000001
>>> sample_nodes = preferential_node_selection(G, 10000)
>>> numpy.mean(G.degree(sample_nodes).values())
--> 8.7766999999999999
```


Enron email network

(9a) Let’s do some random walking on a network. Write two functions, where the first one selects a neighbor of a given node uniformly at random (easy!), and the second one uses the first function to actually perform the walk. The second function will need the starting node of the walk as a parameter, as well as the maximum number of steps to be taken, and it should return the path of the entire walk (including the starting node) as a list.

Enron email network

(9a) Let's do some random walking on a network. Write two functions, where the first one selects a neighbor of a given node uniformly at random (easy!), and the second one uses the first function to actually perform the walk. The second function will need the starting node of the walk as a parameter, as well as the maximum number of steps to be taken, and it should return the path of the entire walk (including the starting node) as a list.

```
def select_neighbor(G, source):
    return random.choice(G[source].keys())

def perform_walk(G, source, num_steps):
    path = [source]
    for step in range(num_steps):
        source = select_neighbor(G, source)
        path.append(source)
    return path
```


Enron email network

(9b) How does the average degree of nodes encountered along the path change as the walk proceeds? In other words, what happens to node average degree as a function of the number of steps taken in the random walk? Use the previous code to perform a large number of random walks (say 10,000), each walk starting from a different source node and consisting of 100 steps, and compute and plot the average degree after zero steps, after one step, after two steps, etc. Note that the average degree after zero steps should be, more or less, equal the overall average degree of the network.

Enron email network

(9b) How does the average degree of nodes encountered along the path change as the walk proceeds? In other words, what happens to node average degree as a function of the number of steps taken in the random walk? Use the previous code to perform a large number of random walks (say 10,000), each walk starting from a different source node and consisting of 100 steps, and compute and plot the average degree after zero steps, after one step, after two steps, etc. Note that the average degree after zero steps should be, more or less, equal the overall average degree of the network.

```
>>> H = nx.barabasi_albert_graph(10000, 3)
... max_walk_length = 100
... degree_counts = {}
...
... for step in range(max_walk_length+1):
...     degree_counts[step] = []
...
... for rep in range(10000):
...      start_node = random.choice(H.nodes())
...      rw_path = perform_walk(H, start_node, max_walk_length)
...      for step in range(max_walk_length+1):
...          degree_counts[step].append(H.degree(rw_path[step]))
...
... for step in range(max_walk_length+1):
...      numpy.mean(degree_counts[step])
```


Enron email network

(9b) How does the average degree of nodes encountered along the path change as the walk proceeds? In other words, what happens to node average degree as a function of the number of steps taken in the random walk? Use the previous code to perform a large number of random walks (say 10,000), each walk starting from a different source node and consisting of 100 steps, and compute and plot the average degree after zero steps, after one step, after two steps, etc. Note that the average degree after zero steps should be, more or less, equal the overall average degree of the network.

For a Barabasi-Albert network with N=10,000 and m = 3 we get the following outcome: