final-2

May 5, 2020

0.0.1 (a)Building the PPI graph.

First, you will construct a simple **undirected, unweighted** graph from this protein-protein interaction network. Namely, each graph node corresponds to a protein in the protein interaction network, and an **undirected edge exists between two nodes if there is an interaction between the two proteins.** Show how you construct your graph (code or log of interactive data analysis commands) as part of your solution to part a. Help us out by labeling the submission with the file header FPp2a.

[47]:
```
import matplotlib.pyplot as plt
import networkx as nx
import numpy as np
import os
```

[48]:
```
# read edge list to networkx
# the format of each line: (protein-1, protein-2, confidence of the interaction␣
↪existence )
def createGraph(filename) :
    G = nx.Graph()
    for line in open(filename) :
        strlist = line.split('\t')
        n1 = strlist[0]
        n2 = strlist[1]
        weight = float(strlist[2])
        G.add_weighted_edges_from([(n1, n2, weight)]) #G.add_edges_from([(n1,␣
↪n2)])
    return G
```

[49]:
```
yeast_file = 'yeast.ppi'
G = createGraph(yeast_file)
```

0.0.2 (b)Compute the degree distribution for the 5,001 nodes of the graph

plot it as a bar graph with degree on the x-axis and counts on the y-axis, and submit this image as **FPp2-degreedist.**

[50]:
```
G.number_of_nodes()
```

[50]: 5001

[51]:
```
G.number_of_edges()
```

[51]: 76025

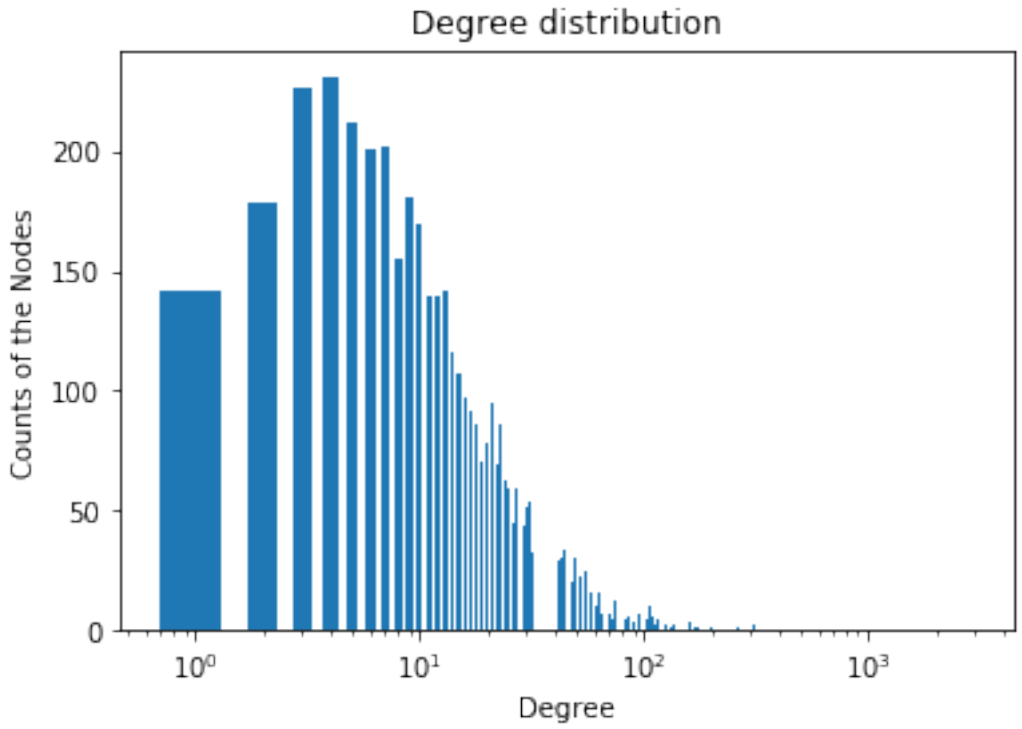
[52]:
```
import collections
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)  # degree␣
↪sequence
#print("Degree sequence", degree_sequence)
degreeCount = collections.Counter(degree_sequence)
print(degreeCount.items())
```

```
dict_items([(2950, 1), (2327, 1), (1093, 1), (1048, 1), (978, 1), (937, 1),
(881, 1), (812, 1), (718, 1), (581, 1), (552, 2), (486, 1), (485, 1), (483, 1),
(453, 1), (450, 1), (431, 1), (420, 1), (406, 1), (404, 1), (383, 1), (377, 1),
(371, 1), (360, 1), (357, 1), (351, 1), (342, 1), (325, 1), (313, 1), (309, 2),
(308, 1), (302, 1), (292, 1), (284, 1), (281, 1), (271, 1), (262, 1), (260, 1),
(258, 1), (254, 1), (246, 1), (234, 1), (232, 1), (211, 1), (208, 1), (204, 2),
(203, 1), (202, 1), (199, 1), (195, 1), (193, 1), (192, 2), (190, 1), (187, 2),
(185, 1), (183, 2), (179, 1), (178, 1), (174, 1), (173, 1), (170, 1), (169, 1),
(168, 2), (167, 1), (165, 1), (164, 1), (161, 1), (160, 3), (156, 1), (155, 1),
(154, 2), (153, 1), (151, 2), (150, 1), (149, 2), (148, 1), (147, 2), (146, 1),
(145, 1), (144, 1), (142, 1), (141, 1), (140, 1), (139, 3), (138, 2), (137, 2),
(136, 2), (135, 3), (134, 2), (132, 1), (131, 4), (130, 1), (129, 4), (128, 2),
(127, 3), (125, 2), (124, 4), (123, 3), (122, 3), (121, 3), (120, 4), (119, 3),
(118, 4), (117, 1), (116, 4), (115, 4), (114, 6), (113, 1), (112, 2), (111, 5),
(110, 4), (109, 6), (108, 5), (107, 3), (106, 10), (105, 4), (104, 4), (103, 5),
(102, 6), (101, 5), (100, 4), (99, 6), (98, 5), (97, 7), (96, 5), (95, 7), (94,
4), (93, 7), (92, 6), (91, 6), (90, 3), (89, 7), (88, 5), (87, 10), (86, 6),
(85, 6), (84, 8), (83, 5), (82, 7), (81, 6), (80, 7), (79, 13), (78, 5), (77,
6), (76, 6), (75, 12), (74, 12), (73, 9), (72, 4), (71, 8), (70, 7), (69, 13),
(68, 12), (67, 8), (66, 16), (65, 7), (64, 7), (63, 16), (62, 25), (61, 10),
(60, 14), (59, 16), (58, 16), (57, 12), (56, 20), (55, 24), (54, 17), (53, 17),
(52, 22), (51, 23), (50, 16), (49, 30), (48, 20), (47, 28), (46, 32), (45, 31),
(44, 33), (43, 30), (42, 29), (41, 33), (40, 42), (39, 35), (38, 38), (37, 45),
(36, 47), (35, 37), (34, 46), (33, 39), (32, 32), (31, 53), (30, 51), (29, 43),
(28, 41), (27, 59), (26, 45), (25, 59), (24, 63), (23, 86), (22, 69), (21, 95),
(20, 78), (19, 70), (18, 86), (17, 92), (16, 97), (15, 107), (14, 116), (13,
142), (12, 139), (11, 139), (10, 169), (9, 181), (8, 155), (7, 202), (6, 201),
(5, 212), (4, 231), (3, 227), (2, 178), (1, 142)])
```

[53]:
```
deg, cnt = zip(*degreeCount.items())

fig, ax = plt.subplots()
plt.bar(deg, cnt, width=0.60)
```

```
#plt.hist(degree_values)
plt.xscale('log')
plt.title("Degree distribution")
plt.ylabel('Counts of the Nodes')
plt.xlabel("Degree")
plt.savefig('FPp2-degreedist.png')
plt.show()
```



Do you believe that this network fits the definition of scale-free? Why or why not? (Put your answer in the file FPp2b-answer.)

I think this PPI Network is scale-free. As the degree sequence shows, there is one node whose degree can be 2950, but there are 142 nodes have degree=1, and 178 have degree = 2. I did some scaling work to my output image, without it the graph will shows really unbalanced.

0.0.3 (c) Compute the local clustering coefficient for each node in the graph.

Submit any **code** you wrote, plus a **tab-delimited file, FPp2c-clustcoef** 1. First column is the protein ID 2. Second column is the clustering coefficient. 3. Both YGR296W and YPL098C have 5 interacting partners; which one has the higher clustering coecient? Explain briefly what the difference means.

[54]:
```
G_test = nx.Graph()
G_test.add_nodes_from(['a','b','c','d','e','f','g'])
```

```
G_test.
 ↪add_edges_from([('a','d'),('b','d'),('b','c'),('c','d'),('d','e'),('d','f'),('d','g'),('e',

G1 = nx.Graph()
G1.add_nodes_from(['a','b','c','d','e','f','g'])
G1.
 ↪add_edges_from([('a','b'),('a','e'),('b','c'),('b','e'),('b','f'),('c','f'),('c','e'),('d',
```

[55]:
```
def local_cluster(G, node):
    #print("degree ", G[node])
    x = sum([1 for u in G[node] for v in G[node] if not u == v and (u, v) in G.
 ↪edges()])
    #print("x = ",x)
    if(len(G[node]) != 0 and len(G[node])!=1):
        y = (len(G[node]) * (len(G[node]) - 1))
        #print("y= ",y)
    else:
        return 0.0
    return  x/y
```

[56]:
```
fp2c = 'FPp2c-clustcoef.txt'
nodes = list(G.nodes())

with open(fp2c,'a') as f:
    for node in nodes:
        f.write(node+'\t'+str(local_cluster(G,node))+'\n')
```

[57]:
```
print("YGR296W: ",local_cluster(G,'YGR296W'))
print("YPL098C: " ,local_cluster(G,'YPL098C'))
```

```
YGR296W:  0.1
YPL098C:  0.8
```

The result of local cluster coefficient is YGR296W = 0.1 and YPL098C = 0.8, so the latter is higher. As the result shows, the difference between then is the Numerator, which means the number of triangles containing v. Consider that they have the same degree = 5, it means the interacting partners of YPL098C also interact with each other much more than YGR296W's partners.

[58]:
```
def triangle(Graph):
    tri_test = list(nx.triangles(Graph).values())
    tmp = 0
    for i in tri_test:
        tmp += i
    return tmp/3.0
```

[59]:
```
print(triangle(G))
```

```
354514.0
```

The number of triangles in G is 354514, I use the existing methods in networkX, which is the nx.triangles(), but this method will count the triangle of every nodes in Graph G, so to get the percise result, we need to divide the result by 3.

0.0.4 (e) A simple way to define the "closeness" is to use shortest path distance.

In order to estimate the path length distribution for this graph, sample 1000 nodes at random from the graph, and compute the distribution of shortest path distances between these 1000 nodes.

[60]:
```
from random import sample
random_nodes = sample(list(G.nodes()),1000)
```

[61]:
```
new_edge = []
for i in range(0,1000):
    for j in range(i+1,1000):
        if G.get_edge_data(random_nodes[i],random_nodes[j]):
            new_edge.append((random_nodes[i],random_nodes[j]))

G_tmp = nx.Graph()
G_tmp.add_nodes_from(random_nodes)
G_tmp.add_edges_from(new_edge)
```

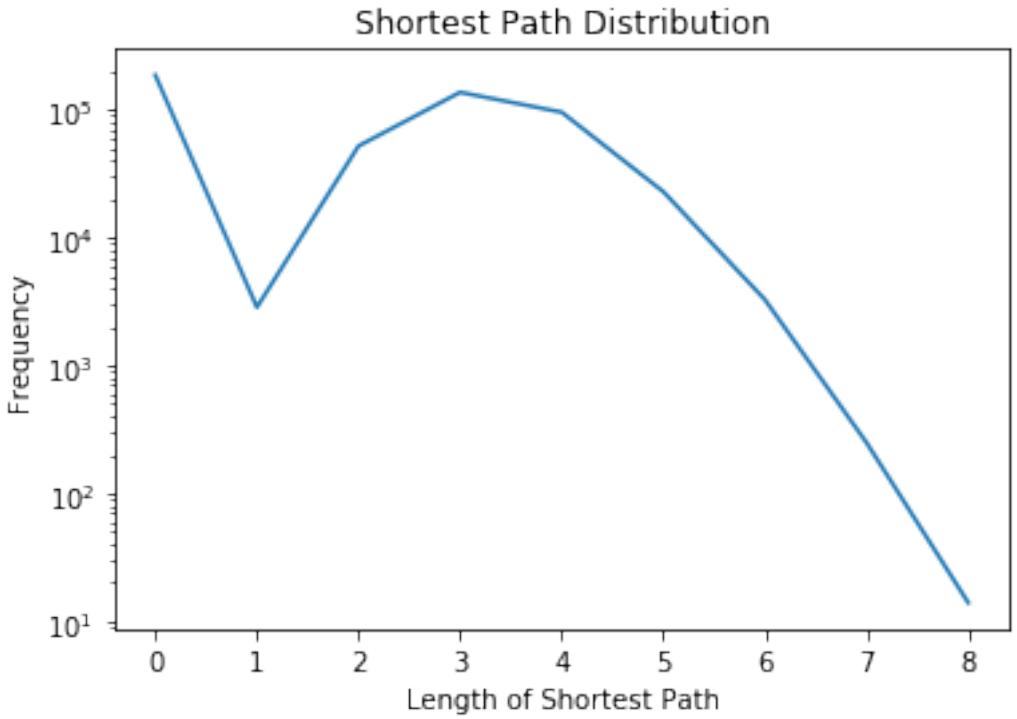
[62]:
```
tmp_nodes = list(G_tmp.nodes())
y = np.zeros((1000,1000),dtype=np.int)
for i in range(0,1000):
    for j in range(i+1,1000):
        if nx.has_path(G_tmp,tmp_nodes[i],tmp_nodes[j]):
            y[i][j] = nx.shortest_path_length(G_tmp, tmp_nodes[i],tmp_nodes[j])
    #print(y[i])
```

[63]:
```
def get_freq(arr):
    unique = np.unique(arr)
    res = [0]*len(unique)
    for i in range(len(arr)):
        for j in range(i+1,len(arr)):
            for k in range(len(unique)):
                if arr[i][j] == unique[k]:
                    res[k] +=1
    return res

xout = np.unique(y)
yout = get_freq(y)
```

[64]:
```
plt.title("Shortest Path Distribution")
plt.plot(xout,yout)
plt.xticks(xout)
plt.yscale('log')
plt.ylabel('Frequency')
```

```
plt.xlabel("Length of Shortest Path")
plt.savefig("FPp2e-spdists.png")
plt.show()
```



0.0.5 Description of methods used.

How does this compare to your expectations for a protein-protein interaction network?

The unexpected thing I found is, the lenght of shortest path have too much 0, which I think it might be more reasonable to be less than len=1. But when I print all the column in y, I realized that there are many proteins do not connect with other proteins, it is because we only find the path between these 1000 nodes. Therefore, the graph with these 1000 nodes and their edges is **disconnected.**

[65]:
```
tmp = list(nx.connected_components(G_tmp))
print(len(tmp))
#print(tmp[1:len(tmp)])
```

204

[66]:
```
diam = []
for c in nx.connected_components(G_tmp):
    diam.append(nx.diameter(G_tmp.subgraph(c)))
print(max(diam))
```

```
8
```

0.0.6 (f) Estimate the diameter of the network based on your answer to the previous question.

How does what you found relate to the results of the previous question?

Since this graph is undirected and disconnected graph, to count the diameter we need to know its connected components. But the result is, there are 204 components,and most of them is an individual node (which menas diameter = 0), so the diam in this network is 8.