1_CodeNo(1 Main)

April 15, 2021

1 Network architecture of the long-distance pathways in the macaque brain

1.0.1 Juhi Pandey 2018393

1.0.2 Ritik Malik 2018406

1.0.3 Yashraj 2018422

The network consists of 1) 383 hierarchically organized regions (nodes) 2) 6,602 directed long-distance connections (edges)

[1]:
```
import networkx as nx
import operator
from collections import Counter
import numpy as np
import matplotlib.pyplot as plt
import math
from numpy.polynomial.polynomial import polyfit
```

[2]:
```
g = nx.DiGraph()
g1 = nx.DiGraph()
node_label = {}
```

[3]:
```
f = open("NameList.txt")
line = f.readline()
while line:
    n1, n2 = line.split()
    node_label[int(n1)] = n2
    line = f.readline()
f.close()
print("No of nodes:", len(node_label))
```

```
No of nodes: 383
```

[4]:
```
f = open("EdgeList.txt")
line = f.readline()
while line:
    n1, n2 = map(int, line.split())
```

```
    g.add_edge(node_label[n1], node_label[n2])
    g1.add_edge(node_label[n2], node_label[n1])
    line = f.readline()
f.close()
print("No of edges:", len(g.edges))
# print(len(g.nodes))
# print(g.edges)
```

```
No of edges: 6602
```

[5]:
```
x = nx.hits(g)
```

1.1 For integrator characteristics

1.1.1 In-degree

[6]:
```
# in degree ranks
d = (list(g.in_degree(g.nodes)))
d.sort(key=lambda x: x[1], reverse=True)
print("Top 15 nodes")
print(* d[:15])
```

```
Top 15 nodes
('32', 105) ('46', 102) ('12o', 99) ('12l', 91) ('11', 90) ('24', 83) ('F7', 75)
('14', 75) ('8A', 71) ('LIP', 68) ('13a', 65) ('MD', 65) ('13', 63) ('F2', 63)
('PIT', 62)
```

1.1.2 In-closeness

[7]:
```
# in closeness centrality
closeness_centrality_in = nx.closeness_centrality(g)
closeness_centrality_in = dict(sorted(closeness_centrality_in.items(),␣
 ↪key=operator.itemgetter(1), reverse=True))
print("Top 15 nodes")
for i in list(closeness_centrality_in.items())[:15]:
    print(i, end=" ")
```

```
Top 15 nodes
('46', 0.5431154917505893) ('12o', 0.5348864691483076) ('32',
0.5324661683829306) ('11', 0.5324661683829306) ('24', 0.526117838506532) ('12l',
0.5245543382435112) ('MD', 0.5064922089496169) ('8A', 0.5007447796282029) ('Cd',
0.4944444875505311) ('23c', 0.4923641138603668) ('8B', 0.49099453357146455)
('F7', 0.4889543900801704) ('LIP', 0.4876036873451423) ('9', 0.4855915675899354)
('6M', 0.4855915675899354)
```

1.1.3 Authorities

[8]:
```
authorities = dict(x[1])
authorities = dict(sorted(authorities.items(), key=operator.itemgetter(1),␣
↪reverse=True))
print("Top 15 nodes")
for i in list(authorities.items())[:15]:
    print(i, end=" ")
```

```
Top 15 nodes
('32', 0.015831704244758073) ('12o', 0.015600917654495632) ('46',
0.01532500434215252) ('12l', 0.014552905754659786) ('11', 0.014496482783232 43)
('24', 0.013686358059245793) ('14', 0.011681841635970788) ('F7',
0.010758218113501593) ('MD', 0.010548633152332784) ('9', 0.00979042829769803)
('8A', 0.009568966632889978) ('8B', 0.0095424957674249) ('LIP',
0.009485890731054807) ('10', 0.009478458180326724) ('23c', 0.009467843328700268)
```

1.2 For distributor characteristics

1.2.1 Out-degree

[9]:
```
# out degree ranks
out_d = (list(g.out_degree(g.nodes)))
out_d.sort(key=lambda x: x[1], reverse=True)
print("Top 15 nodes")
print(* out_d[:15])
```

```
Top 15 nodes
('46', 109) ('24', 93) ('TF', 74) ('9', 69) ('13', 66) ('13a', 65) ('TH', 61)
('TE', 61) ('LIP', 60) ('PGm', 57) ('V2', 55) ('32', 54) ('L#2', 53) ('36', 51)
('PIT', 50)
```

1.2.2 Out-closeness

[10]:
```
# out closeness centrality
closeness_centrality_out = nx.closeness_centrality(g1)
closeness_centrality_out = dict(sorted(closeness_centrality_out.items(),␣
↪key=operator.itemgetter(1), reverse=True))
print("Top 15 nodes")
for i in list(closeness_centrality_out.items())[:15]:
    print(i, end=" ")
```

```
Top 15 nodes
('46', 0.5474767800497351) ('24', 0.5307343708739022) ('TF', 0.5165182716540656)
('TE', 0.5045062188249012) ('9', 0.499424861225226) ('TH', 0.49585754078790295)
('LIP', 0.4909480601860425) ('PGm', 0.4881860457827455) ('45',
0.4861348439097088) ('23', 0.48477692535130174) ('PM#3', 0.48410080690590246)
('12', 0.4794202742424476) ('Idg', 0.4741807084037323) ('13',
0.47288866287674663) ('L#2', 0.47224527694085994)
```

1.2.3 Hubs

[11]:
```
hubs = dict(x[0])
hubs = dict(sorted(hubs.items(), key=operator.itemgetter(1), reverse=True))
print("Top 15 nodes")
for i in list(hubs.items())[:15]:
    print(i, end=" ")
```

```
Top 15 nodes
('46', 0.015204683627532618) ('24', 0.012043365179723658) ('9',
0.011199623290855698) ('TF', 0.010524136819908747) ('TE', 0.00994074274650209)
('TH', 0.009366945303796165) ('13', 0.009267523639687797) ('32',
0.009190899674239212) ('12', 0.008821951764963759) ('23', 0.00871384241866026)
('45', 0.008643582251765309) ('PM#3', 0.00864271967983785 1) ('10',
0.008576842831566728) ('13a', 0.008421112301364166) ('Idg',
0.008034420981898889)
```

1.3 For intermediary characteristics

1.3.1 Betweenness

[12]:
```
betweenness_centrality = nx.betweenness_centrality(g)
betweenness_centrality = dict(sorted(betweenness_centrality.items(),␣
 ↪key=operator.itemgetter(1), reverse=True))
print("Top 15 nodes")
for i in list(betweenness_centrality.items())[:15]:
    print(i, end=" ")
```

```
Top 15 nodes
('24', 0.08784097100981372) ('46', 0.08443409805406714) ('LIP',
0.04653389124898251) ('13a', 0.044443125077972434) ('MD', 0.04014159818290519)
('32', 0.03842292075067489) ('PIT', 0.034008547967126254) ('TF',
0.033996682900884666) ('13', 0.031533444477577215) ('PS', 0.03063231393974855)
('V2', 0.029902918018483927) ('TE', 0.025157721327871835) ('PGm',
0.022881361615237242) ('7b', 0.022855063273476504) ('9', 0.02221030457789365)
```

1.3.2 PageRank

[13]:
```
pagerank = nx.pagerank(g)
pagerank = dict(sorted(pagerank.items(), key=operator.itemgetter(1),␣
 ↪reverse=True))
print("Top 15 nodes")
for i in list(pagerank.items())[:15]:
    print(i, end=" ")
```

```
Top 15 nodes
('32', 0.01563918209038171) ('MD', 0.014171269432721614) ('36r',
0.012625856703811387) ('46', 0.012609599386608022) ('PIT', 0.011774417716507331)
('12o', 0.011771604226761314) ('24', 0.011612532850469377) ('23c',
```

```
0.010749563858063037) ('12l', 0.010633961976959955) ('11', 0.01052850829640844)
('14', 0.01036563047377749) ('13a', 0.010136295077933063) ('8A',
0.008876843387264051) ('F7', 0.008561698295332718) ('8B', 0.008520880431601258)
```

1.4 Degree Distribution

[14]:
```
# degree distribution
degrees = [g.degree(n) for n in g.nodes()]
degrees = Counter(degrees)
degrees = dict(sorted(degrees.items()))
print(degrees)

no_of_nodes = sum(degrees.values())
for key in degrees.keys():
    degrees[key] /= no_of_nodes
```

```
{1: 2, 2: 15, 3: 6, 4: 5, 5: 12, 6: 7, 7: 12, 8: 7, 9: 5, 10: 8, 11: 4, 12: 7,
13: 4, 14: 7, 15: 5, 16: 1, 17: 8, 18: 4, 19: 5, 20: 9, 21: 10, 22: 6, 23: 7,
24: 7, 25: 7, 26: 6, 27: 4, 28: 5, 29: 5, 30: 4, 31: 5, 32: 5, 33: 6, 34: 2, 35:
5, 36: 4, 37: 3, 38: 8, 39: 5, 40: 1, 41: 2, 43: 7, 44: 3, 45: 5, 46: 2, 47: 3,
48: 3, 49: 6, 50: 6, 51: 2, 52: 4, 53: 2, 54: 6, 55: 1, 56: 1, 57: 4, 58: 3, 61:
1, 63: 2, 65: 1, 66: 1, 67: 2, 68: 2, 69: 1, 70: 2, 72: 1, 74: 2, 75: 1, 77: 2,
78: 1, 79: 1, 81: 1, 82: 4, 83: 1, 85: 1, 86: 3, 88: 3, 89: 2, 90: 1, 92: 2, 93:
1, 95: 1, 96: 2, 97: 1, 100: 1, 106: 1, 107: 1, 110: 1, 111: 1, 112: 1, 115: 1,
123: 1, 124: 1, 125: 1, 127: 2, 128: 1, 129: 1, 130: 1, 135: 1, 159: 1, 176: 1,
211: 1}
```

[15]:
```
core = g.copy()
d_core = [core.degree(n) for n in core.nodes()]
d_core = dict(sorted(Counter(d_core).items()))
# print(d_core)
```

1.4.1 Finding the core of the network by peeling off outer edges with 1 node, then with 2 nodes, and so on upto 28 nodes

All the nodes in the core have a degree of at least 29

[16]:
```
for circle in range(1, 29):
    remove = []
    for node in core.nodes:
        if core.degree(node) <= circle:
            remove.append(node)
    core.remove_nodes_from(remove)

while True:
    remove = []
    for node in core.nodes:
        if core.degree(node) < 29:
```

```
            remove.append(node)
    core.remove_nodes_from(remove)
    if not remove:
        break
```

[17]:
```
d_core = [core.degree(n) for n in core.nodes()]
print("Remaining nodes:", len(d_core))
print("This is consistent with what the authors observed in the original paper")
d_core = dict(sorted(Counter(d_core).items()))
print(d_core)
```

```
Remaining nodes: 122
This is consistent with what the authors observed in the original paper
{29: 4, 30: 6, 31: 6, 32: 8, 33: 9, 34: 4, 35: 4, 36: 5, 37: 3, 38: 5, 39: 8,
40: 1, 41: 3, 42: 4, 43: 2, 44: 1, 45: 2, 47: 1, 48: 1, 50: 1, 51: 2, 52: 2, 53:
2, 54: 4, 55: 1, 56: 4, 57: 1, 59: 2, 60: 3, 61: 2, 63: 1, 64: 3, 69: 2, 70: 1,
72: 2, 73: 2, 75: 1, 77: 1, 78: 1, 80: 1, 82: 1, 87: 1, 90: 1, 98: 1, 111: 1,
129: 1}
```

Comparing with the inbuilt function, we saw that the number of edges and nodes are the same as the core graph we created

[18]:
```
new_b = nx.k_core(g, k=29)
d_new = [new_b.degree(n) for n in new_b.nodes()]
d_new = dict(sorted(Counter(d_new).items()))
print(len(new_b.nodes))
print(d_new)
```

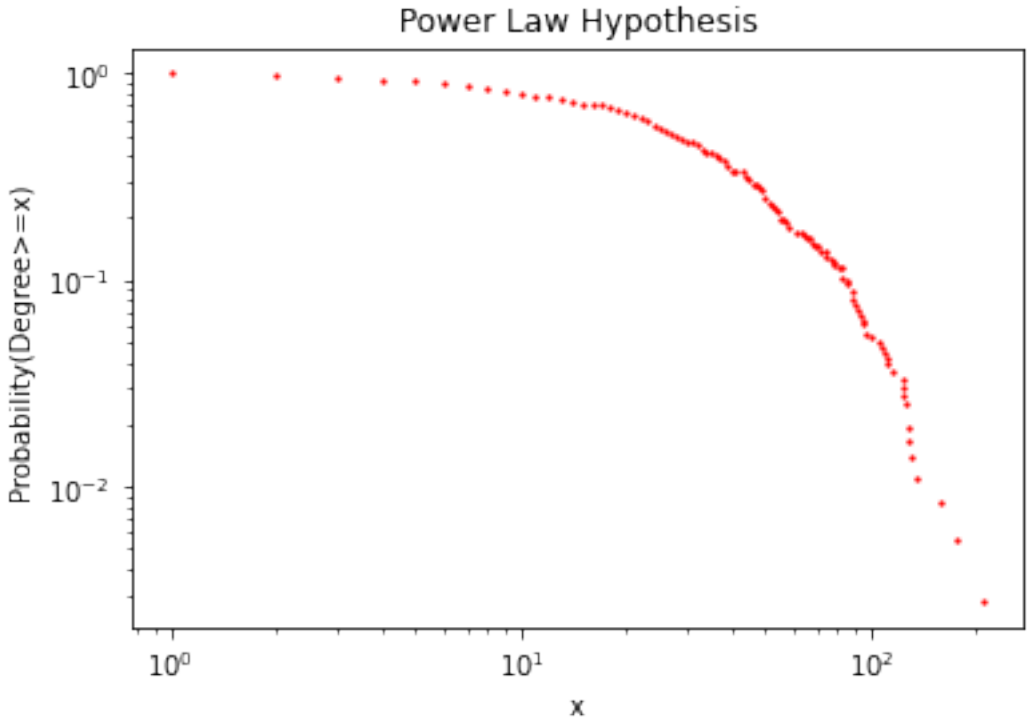
```
122
{29: 4, 30: 6, 31: 6, 32: 8, 33: 9, 34: 4, 35: 4, 36: 5, 37: 3, 38: 5, 39: 8,
40: 1, 41: 3, 42: 4, 43: 2, 44: 1, 45: 2, 47: 1, 48: 1, 50: 1, 51: 2, 52: 2, 53:
2, 54: 4, 55: 1, 56: 4, 57: 1, 59: 2, 60: 3, 61: 2, 63: 1, 64: 3, 69: 2, 70: 1,
72: 2, 73: 2, 75: 1, 77: 1, 78: 1, 80: 1, 82: 1, 87: 1, 90: 1, 98: 1, 111: 1,
129: 1}
```

[19]:
```
cum_degree = {}
temp = 1
for key in degrees:
    cum_degree[key] = temp
    temp -= degrees[key]
# print(cum_degree)
```

[20]:
```
plt.figure()
plt.title("Power Law Hypothesis")
plt.xlabel("x")
plt.ylabel("Probability(Degree>=x)")
plt.xscale("log")
plt.yscale("log")
```

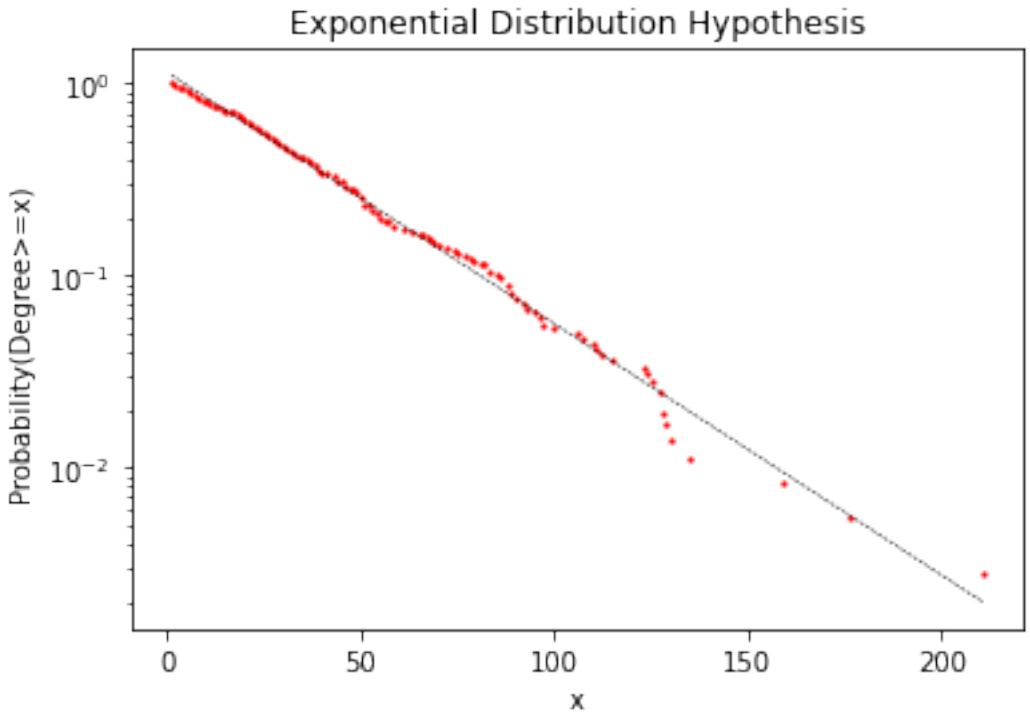
```
plt.scatter(cum_degree.keys(), cum_degree.values(), color="red", s=1.5)
# print(len(cum_degree))

plt.autoscale()
plt.show()
```



1.4.2 The degree distribution is not that of a scale free graph and does not follow power law

[21]:
```
plt.figure()
plt.title("Exponential Distribution Hypothesis")
plt.xlabel("x")
plt.ylabel("Probability(Degree>=x)")
plt.yscale("log")
b, m = polyfit(np.array(list(cum_degree.keys())), np.log10(np.
 ↪array(list(cum_degree.values()))), 1)
y_hat = (b + m * np.array(list(cum_degree.keys())))
y_hat = [10**i for i in y_hat]
plt.plot(np.array(list(cum_degree.keys())), y_hat, color="black", linestyle =␣
 ↪'--', linewidth = 0.5)
plt.scatter(cum_degree.keys(), cum_degree.values(), color="red", s=1.5)
plt.autoscale()
plt.show()
```



1.4.3 This shows strong evidence of Maximum Entropy Exponential Degree Distribution and we can predict the distribution using the regression line

1.5 Vizualising the hierarchy in graph

[22]:
```
hierarchy = nx.DiGraph()
```

[23]:
```
f = open("Mapping.txt")
line = f.readline()
while line:
    parent, child = map(int, line.split())
    hierarchy.add_edge(node_label[parent], node_label[child])
    line = f.readline()
f.close()
```

[24]:
```
print("Hierarchy:", nx.flow_hierarchy(hierarchy))
```

```
Hierarchy: 1.0
```

[25]:
```
nx.write_gml(g, "Graph.gml", stringizer=None)
```

[26]:
```
nx.write_gml(hierarchy, "Hierarchy.gml", stringizer=None)
```

[27]:
```
nx.write_gml(core, "Core.gml", stringizer=None)
```