Networks: structure, evolution & processes

Internet Analytics - Lab 2

2.4 PageRank

2.4.1 Random Surfer Model

Exercise 2.12

In [1]:
```
import networkx as nx
import random as rd
import numpy as np
import pandas as pd
```

In [2]:
```
components = nx.read_adjlist("../data/components.graph",create_using=nx.DiGraph)
absorbing = nx.read_adjlist("../data/absorbing.graph",create_using=nx.DiGraph)
```

In [3]:
```
def randomSurferWithRestart(graph):
    node = rd.sample(graph.nodes(),1)
    count = [0]* len(graph.nodes())
    n = 1000
    i = 0
    while(i<n):
        i+=1
        out_edges = list(graph.out_edges(node))
        if(len(out_edges)==0):
            node = rd.sample(graph.nodes(),1)[0]
        else:
            node = rd.sample(list(graph.out_edges(node)),1)[0][1]
        count[int(node)]+= 1
    total = sum(count)
    return [float(i)/total for i in count]
```

In [4]:
```
print(randomSurferWithRestart(components))
print(randomSurferWithRestart(absorbing))
```

```
[0.0, 0.0, 0.0, 0.0, 0.288, 0.135, 0.289, 0.288]
[0.14, 0.338, 0.152, 0.228, 0.142]
```

To make sure that the algorithm would run properly a major bug had to be taken care of in the case of the absorbing graph. This is because the absorbing graph has a node that doesn't have any outgoing edges stopping the random surfing process. To resolve this I had the algorithm choose a new starting node uniformly at random. The second problem is with disconnected subgraphs as in the case of components. In that case the page ranking depends completely on what node we start as that will determine what nodes are accessible.

Exercise 2.13

In [5]:
```
def randomSurferImproved(graph):
    node = rd.sample(graph.nodes(),1)
    count = [0]* len(graph.nodes())
    n = 1000
    i = 0
    df = 0.15
    while(i<n):
        i+=1
        if(rd.random()< df):
            node= rd.sample(graph.nodes(),1)[0]
        out_edges = list(graph.out_edges(node))
        if(len(out_edges)==0):
            node = rd.sample(graph.nodes(),1)[0]
        else:
            node = rd.sample(list(graph.out_edges(node)),1)[0][1]
        count[int(node)]+= 1
    total = sum(count)
    return [float(i)/total for i in count]
```

In [6]:

```
print(randomSurferImproved(components))
print(randomSurferImproved(absorbing))
```

```
[0.157, 0.163, 0.158, 0.074, 0.129, 0.052, 0.134, 0.133]
[0.139, 0.394, 0.118, 0.197, 0.152]
```

I think the page rank score makes intuitive sense, as linking to a web page shows a certain level of trust or "vouching" for another webpage. It therefore makes sense to rank webpages based on how many times you end up on that website from other sites. This is seen in our normalised scores too. For components, the node 5 has the worst score as only one node points to it but not only it. On the other hand the node 2 has the best score as not only do 2 nodes point to it but it also points to nodes that soon enough return to the node 2 (2->0->1->2).

2.4.2 Power Iteration Method

Exercise 2.14: Power Iteration method

In [7]:

```
wiki = nx.read_adjlist("../data/wikipedia.graph", create_using=nx.DiGraph)
wikiTitles = pd.read_csv("../data/wikipedia_titles.tsv", sep="\t")
```

In [8]:

```
def constructMatrix(graph):
    node= None
    theta = 0.99
    n=1000
    i=0
    size = len(graph.nodes())
    #construct H u,v
    H = np.empty((size,size))
    nodes_with_edges = set()
    for edge in graph.edges():
        H[int(edge[0]),int(edge[1])] = 1/graph.degree[edge[0]]
        nodes_with_edges.add(edge[0])
        nodes_with_edges.add(edge[1])

    for node in graph.nodes():
        if(node not in nodes_with_edges):
            H[int(node)][:] += 1/size
    G = theta*H + (1-theta)*(np.ones((size,size))/size)
    return G
```

```
def solve(graph):
    size =graph.shape[0]
    sol = np.ones((size))/size
    maxIter = 10000
    iterN = 0
    while(iterN< maxIter):
        iterN+=1
        solNew = graph @ sol
        sol = solNew/np.linalg.norm(solNew, ord=2)
    return sol
```

In [9]:

```
G = constructMatrix(wiki)
sol = solve(G)
```

In [10]:

```
ids = np.argsort(sol)[-10:]
values = [sol[i] for i in ids]
names = [wikiTitles.values[i][1] for i in ids]
print(ids)
print(values)
print(names)
```

```
[3015 3953 4082  865 4079 3066 1676 3480 4055 5489]
[0.031449115154348985, 0.03174701196379255, 0.03189343920763847, 0.03251371324699419
6, 0.033602843332839506, 0.03449060733654383, 0.034880229667211084, 0.03490314378319
5784, 0.03654710633462896, 0.03660087217466944]
['Least squares', 'Physical paradox', 'Portal:Physics', 'Bessel function', 'Portal:M
athematics', 'Linear regression', 'e (mathematical constant)', 'Monty Hall problem',
'Portal:Algebra', 'X Window System core protocol']
```

The top 10 pages with the highest PageRank score are : 'Least squares', 'Physical paradox', 'Portal:Physics', 'Bessel function', 'Portal:Mathematics', 'Linear regression', 'e (mathematical constant)', 'Monty Hall problem', 'Portal:Algebra', 'X Window System core protocol'

2.4.3 Gaming the system *(Bonus)*

Exercise 2.15 *(Bonus)*

In [ ]: