Cassandra Spath

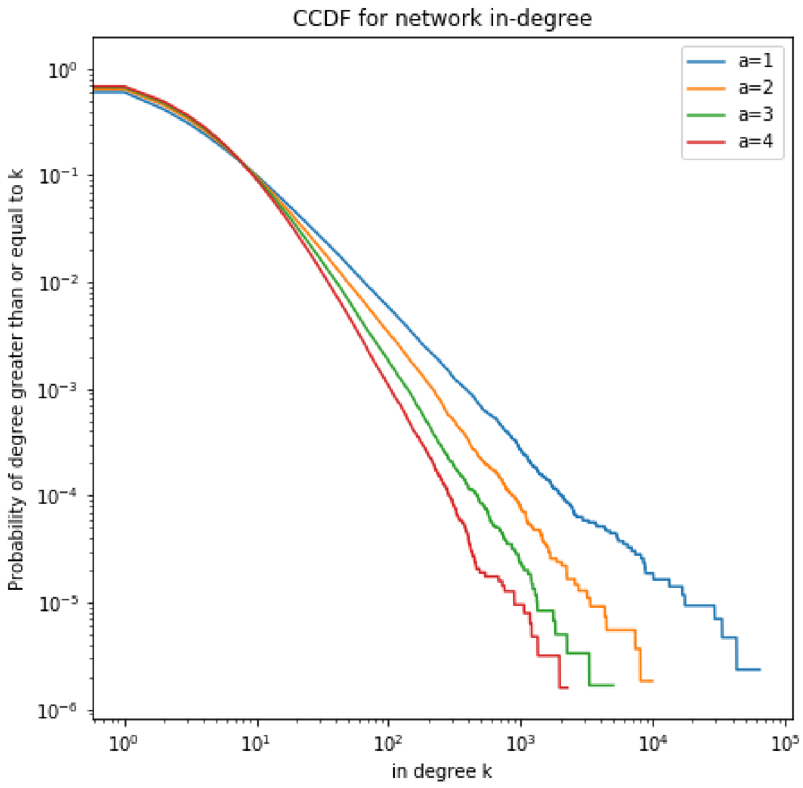
Problem Set 6

CSCI 5352
November 20, 2019

1. (100 pts total) Consider Price's model of a citation network, applied to publications in a single field.

   (a) (35 pts) Implement the simulation algorithm described in Chapter 13.1 of Networks.

   - For choices of $c = 3$ and $n = 10^6$, make a single figure showing the four complementary cumulative distribution functions $Pr(K \geq k_{in})$ (the ccdf) for network in-degree k_{in}, one for each choice of $r = 1, 2, 3, 4$.
   - Briefly discuss the impact of the uniform attachment mechanism on the distribution's shape and comment about the fraction of vertices with $k_{in} = 0$.



As seen in the graph above, the lower the value of α, larger the maximum in-degree k. This is due to the fact, that a lower value causes a greater probability of choosing from the list of targets. Thus, causing the earlier nodes that have higher in-degree to continue getting more edges pointed to it. Thus, if the preferential attachment is more probable than the uniform attachment, there will be a higher maximum degree. Thus, the uniform attachment mechanism results in fewer vertices with higher degrees. However, regardless, most of the elements have in-degree 0 or near 0. This means that while some of the nodes have very high degree. A majority of the nodes have no citations.

(b) (30 pts) Reasonable values of the model parameters for real citation networks are $c = 12$ and $r = 5$.

- For these choices, use your numerical simulation to calculate (i) the average number of citations to a paper (in-degree) in the first 10% of published papers (vertices) and (ii) the average number for a paper in the last 10%.
- Briefly discuss the implications of your results with respect to the "first-mover advantage," and the corresponding bias in citation counts for the first papers published in a field.

The first 10% of papers have an average of 81.36 citations. Whereas the last 10% of papers have an average of 0.19 citations.

This demonstrates the first-mover advantage. The begin nodes have approximately 400 times as many citations are the last papers in this model. Since these papers are more prevalent in the target list, they keep getting cited by other papers later on. Whereas the later papers haven't been cited unless an other later paper chooses it uniformly at random. This is highly unlike due to the number of nodes in the graph.

(c) (15 pts) Under the ICON entry for "arXiv citation networks (1993-2003)," obtain both the network and dates files for the hep-ph citation network.

- For (i) the first 10% and (ii) the last 10% of papers with submission dates, compute their average in-degree. Discuss any steps you took to convert the input data into a form on which you could perform these calculations.
- Briefly discuss how well, and why, these empirical values agree or disagree with your model estimates from question (1b).

After doing some pre-processing on the data, the first 10% of papers have an average of 21.36 citations. The last 10% of papers have an average of 1.92 citations. To do this calculation, I loaded the node data, then the edge data. I only considered edges between nodes that existed in the node data (to ensure they all had timestamps). Additionally, any nodes that didn't have any edges either to or from them were discarded. This data was used to create a network with NetworkX. The nodes were sorted by their corresponding timestamps. I looked at the in degree of the first 10% of nodes from the sorted list to calculate the average number of citations. Then I did the same thing for the last 10% of nodes.

These empirical values don't correspond with the model estimates. The beginning papers have fewer citations empirically than they do in the model, whereas the later papers have more citations. This is likely because over time, new work which uses the previous work will be referenced more than the original work. Thus, the start work is still used but not necessarily cited. Additionally, the later papers likely have new ideas that build on each other causing them to have more citations than predicted.

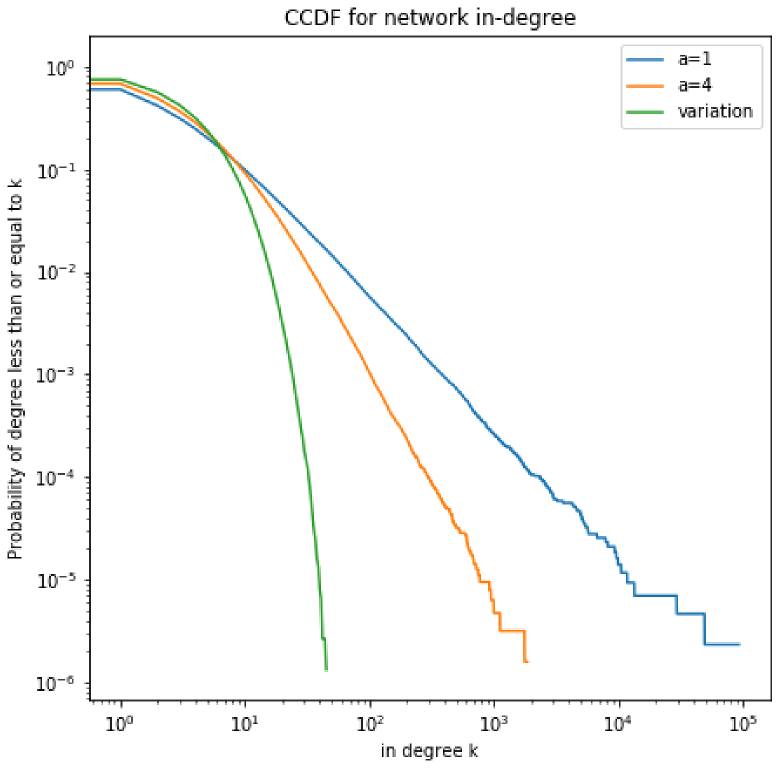
(d) (20 pts) Recall that Price's model is a dramatically simplified view of how nodes in a citation network accumulate new connections. Describe at least three ways that the "preferential attachment" mechanism is unrealistic in this context, and for each, suggest a way that you could analyze a real citation network to demonstrate the difference between what the model predicts and what the real world shows.

The early papers are cited in many of the later papers which are then used by the final papers. Thus, over time these papers might be cited less since the most recent work builds of the work done after the initial papers. This causes the initial papers to have fewer citations than predicted.

In the papers new areas of study may emerge based on the initial papers. These papers then become the base for a new set of papers studying the new topic. Thus, these later papers may have a high number of citations not predicted in the model.

Finally, authors that publish papers are likely to cite their older papers since they are likely continuing to do research on similar topics that pertain to previous papers.

(e) (20 pts extra credit) Now consider a variation of Price's model in which we remove the preferential attachment part. That is, each time a new vertex joins the network, each of its c edges attaches to an existing vertex with equal probability. Using the same parameter choices as in question (1a), produce a figure showing the ccdfs for both this model and Price's model, for r = 1, 4. Briefly discuss the differences in terms of how citations (edges) are distributed across papers (vertices).



As seen above, the effect of the preferential attachments is to bias the new connections towards those with higher in-degree. Thus, there are more vertices with higher degree. Thus, there is more of an even distributions for the variation since the edges are close to evenly distributed around the earlier vertices which were there for more iterations. This seems like a more similar distribution to the empirical data from ICON.

Algorithm 1: Python Code for Problems

```
import numpy as np
import matplotlib.pyplot as plt
import os
import networkx as nx
import random
from mpl_toolkits import mplot3d
import math
import copy
from statistics import mean
from datetime import datetime

#simulation algorithm from text
def simulationAlgorithm(n, c, a, start, targets):
    for i in range(start,n):
        previous = []
        for j in range(c):
            #generate random number
            r = random.random()
            #choose next target
            if r < c/(c+a):
                #pick random target
                index = random.randint(0,len(targets)-1)
                vertex = targets[index]
                while vertex in previous:
                    index = random.randint(0,len(targets)-1)
                    vertex = targets[index]
            else:
                #pick random vertex
                vertex = random.randint(0,i-1)
                while vertex in previous:
                    vertex = random.randint(0,i-1)
            #create a link and add to the target list
            targets.append(vertex)
    return targets

#graph CCDF for graph degrees
def graphCCDF(targets, n, a):
    counts = [0 for i in range(n)]
    for i in targets:
        counts[i] += 1
    maxC = max(counts)
    bins = [0 for i in range(maxC)]
    for c in counts:
        for i in range(c):
            bins[i] += 1
    s = max(bins)
    bins[:] = [x / s for x in bins]
    plt.plot(range(maxC),bins,label="a="+str(a))

#PROBLEM 1a
c = 3
n = 10**6
r = [1,2,3,4]
plt.figure(figsize=(7,7))
for a in r:
    targets = [0,0,1,0,2,1,3,0,2,4]
    simulationAlgorithm(n,c,a,6,targets)
    graphCCDF(targets, n, a)
plt.xscale('log')
plt.yscale('log')
plt.title("CCDF for network in-degree")
plt.ylabel("Probability of degree greater than or equal to k")
plt.xlabel("in degree k")
plt.legend()
plt.savefig("CCDF_overlay.png")

#PROBLEM 1b
c = 12
n = 10**6
a = 5
iterations = 100
firstAve = []
lastAve = []
for it in range(iterations):
    targets = [0,0,1,0,2,1,3,0,2,4,
               1,4,5,6,0,1,3,4,2,4,5,6,7,0,1,4,6,8,
               0,1,3,4,6,8,9,0,2,3,6,7,2,3,6,8,9,10,
               1,3,5,6,9,11,0,2,5,6,8,10,13]
    simulationAlgorithm(n,c,a,15,targets)

    #calculate counts
    counts = [0 for i in range(n)]
    for i in targets:
        counts[i] += 1
    #calculate average of first 10%
    per = int(0.1*n)
    ave = 0
    for i in range(per):
        ave += counts[i]
    firstAve.append(ave/per)
    #calculate average of last 10%
    ave = 0
    for i in range(n-per,n):
        ave += counts[i]
    lastAve.append(ave/per)
print(sum(firstAve)/iterations)
print(sum(lastAve)/iterations)
```

```
#PROBLEM 1c
#load ICON data nodes
nodes = []
nodes_timed = {}
with open("cit-HepPh-dates.txt", "r") as f:
    data = f.readline()
    data = f.readline()
    while data:
        vals = data.split("\n")[0].split("\t")
        date = datetime.fromisoformat(vals[1]).timestamp()
        nodes_timed[int(vals[0])] = date
        nodes.append(int(vals[0]))
        data = f.readline()
f.close()
print(len(nodes))
print(len(nodes_timed))

#load ICON data edges
edge_list = []
node_list = []
with open("Cit-HepPh.txt", "r") as f:
    data = f.readline()
    data = f.readline()
    data = f.readline()
    data = f.readline()
    data = f.readline()
    while data:
        vals = data.split("\n")[0].split("\t")
        v1 = int(vals[0])
        v2 = int(vals[1])
        if v1 in nodes and v2 in nodes:
            node_list.append((v1,nodes_timed[v1]))
            node_list.append((v2,nodes_timed[v2]))
            edge_list.append((v1,v2))
        data = f.readline()
f.close()
#remove duplicates from node list
#print(node_list)
print(len(node_list))
node_list = list(set(node_list))
print(len(node_list))
#sort nodes by timestamps and discard timestamps
node_list = sorted(node_list, key = lambda x: x[1])
nodes = [i[0] for i in node_list]

#create graph
G=nx.DiGraph()
G.add_edges_from(edge_list)
G.add_nodes_from(nodes)
```

```
#calculate average of first 10%
n = len(nodes)
per = int(0.1*n)
ave = 0
for i in range(per):
    index = nodes[i]
    ave += G.in_degree(index)
print(ave/per)
#calculate average of last 10%
ave = 0
for i in range(n-per,n):
    index = nodes[i]
    ave += G.in_degree(index)
print(ave/per)

#PROBLEM 1e
def graphCCDF(targets, n):
    counts = [0 for i in range(n)]
    for i in targets:
        counts[i] += 1
    maxC = max(counts)
    bins = [0 for i in range(maxC)]
    for c in counts:
        for i in range(c):
            bins[i] += 1
    s = max(bins)
    bins[:] = [x / s for x in bins]
    plt.plot(range(maxC),bins, label="variation")


c = 3
n = 10**6
r = [1,4]
plt.figure(figsize=(7,7))
for a in r:
    targets = [0,0,1,0,2,1,3,0,2,4]
    simulationAlgorithm(n,c,a,6,targets)
    graphCCDF(targets, n, a)
plt.xscale('log')
plt.yscale('log')
plt.title("CCDF for network in-degree")
plt.ylabel("Probability of degree greater than or equal to k")
plt.xlabel("in degree k")
plt.legend()
plt.savefig("CCDF_overlay.png")
```