CS 224w: Problem Set 3

Tony Hyun Kim

November 6, 2013

1 Influence maximization

1.1 Example where $f(S_2) < f(T)$

Consider the scenario depicted in Fig. 1: only the nodes x_1, x_2 and x_3 have nontrivial influence sets, indicated by the directed edges.

The optimal set of size $i = 2$ is $T = \{x_1, x_3\}$, which has a combined influence set of 8 elements (counting x_1 and x_3). On the other hand, the greedy algorithm will choose x_2 in the first round, and yield $T_2 = \{x_2, x_1\}$ in the second round, which has an influence set of 7 elements. (Note: x_1 in T_2 could be replaced by x_3).

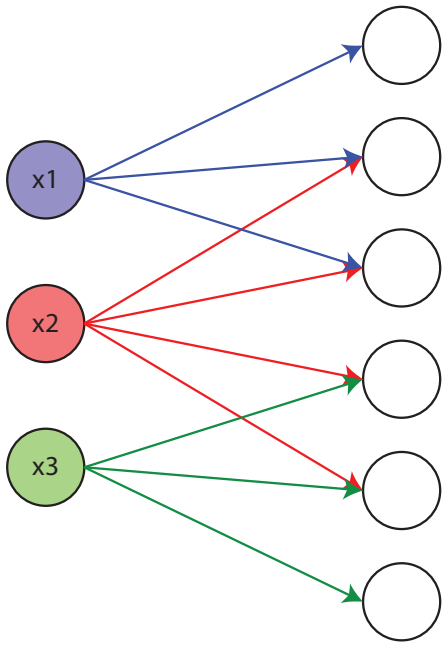

Figure 1: Example where $f(S_2) < f(T)$.

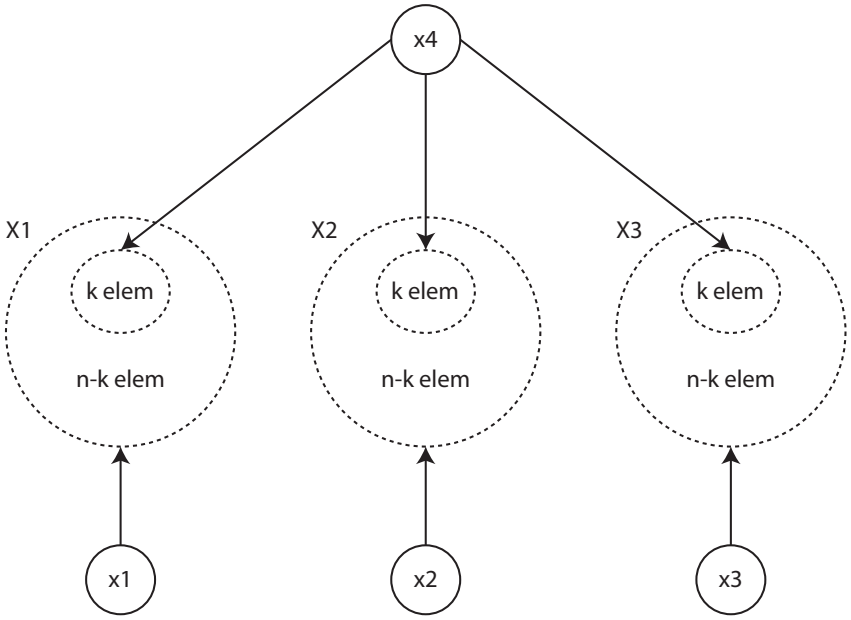

Figure 2: Example where $f(S_3) < 0.8 \cdot f(T)$.

1.2 Example where $f(S_3) \leq 0.8 f(T)$

Consider the scenario depicted in Fig. 2 (which is a generalized form of the case in Section 1.1). Each of the nodes x_1, x_2, x_3 has an influence set of $n+1$ elements (counting itself), whereas node x_4 has an influence set of $3k+1$.

Suppose that $3k > n$. In this case, the first step of the greedy algorithm will choose x_4 since $|X_4| = 3k+1 > |X_{1,2,3}| = n+1$. In the next two steps, the greedy algorithm will choose two of $\{x_1, x_2, x_3\}$ at random. The resulting influence score of the greedy algorithm is $f(S_3) = 2n + k + 3$.

Consider the set $T = \{x_1, x_2, x_3\}$. The influence score will be $f(T) = 3n+3$, and the set T will be optimum as long as $k < n$. So, in the case of Fig. 2, greedy will obtain the suboptimal solution as long as $k < n < 3k$.

Next, we choose n, k such that $f(S_3) \leq 0.8 \cdot f(T)$. The inequality can be stated as follows:

$$0.8 \geq \frac{f(S_3)}{f(T)} = \frac{2n+k+3}{3n+3}, \tag{1}$$

which, after little manipulation, yields:

$$n \geq \frac{5k+3}{2}. \tag{2}$$

The above inequality can be readily satisfied. For example, let $k = 10$, $n = 27$. Since $n < 3k$, the greedy solution will be suboptimal as described above. Furthermore, Eq. 2 becomes $27 \geq 26.5$, so that $f(S_3) \leq 0.8 \cdot f(T)$.

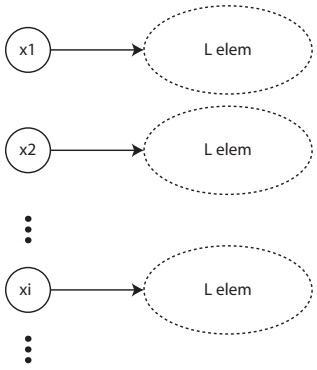

Figure 3: A possible influence set structure that results in a bad data-dependent bound.

1.3 Sufficient property for $f(S_i) = f(T)$

A trivial sufficient condition to achieve $f(S_i) = f(T)$ is to have the top i largest influence sets to be disjoint. In this case, we do not have to worry about possible "interactions" between a node under consideration and the nodes already in our solution set, so the optimal set can be constructed by considering nodes in isolation as in the greedy algorithm.

1.4 Bad data-dependent bound

Here is one pathological example where the data-dependent bound will perform arbitrarily badly. The intuition is that the data-dependent bound relies on the property of diminishing returns; so, I construct an example where the diminishing property is not present.

Consider a case where the influence sets of nodes x_i are disjoint, as depicted in Fig. 3. Each node x_i has an influence set X_i of size $L + 1$. (The downstream nodes in the influence set, *i.e.* $X_i - \{x_i\}$, have trivial influence sets. Furthermore, the influence sets of distinct nodes x_i are disjoint, *i.e.* $X_i \cap X_j = \emptyset$ for $i \neq j$. Assume $|\{x_i\}| \gg k$.

Consider the data-dependent bound $f(S_k) + \sum_{i=1}^{k} \delta_i - f(T)$ for some fixed k. Given disjoint influence sets, S_k is some k-element subset of $\{x_i\}$ and is, in fact, the optimal influence set such that:

$$f(S_k) + \sum_{i=1}^{k} \delta_i - f(T) = \sum_{i=1}^{k} \delta_i. \tag{3}$$

Now, since we assume that $|\{x_i\}| \gg k$, there exists at least k elements of $\{x_i\}$ that are not in S_k. Since the influence sets of x_i are disjoint, we have:

$$f(S_k) + \sum_{i=1}^{k} \delta_i - f(T) = \sum_{i=1}^{k} \delta_i = k \cdot (L + 1), \tag{4}$$

which increases with k without bound.

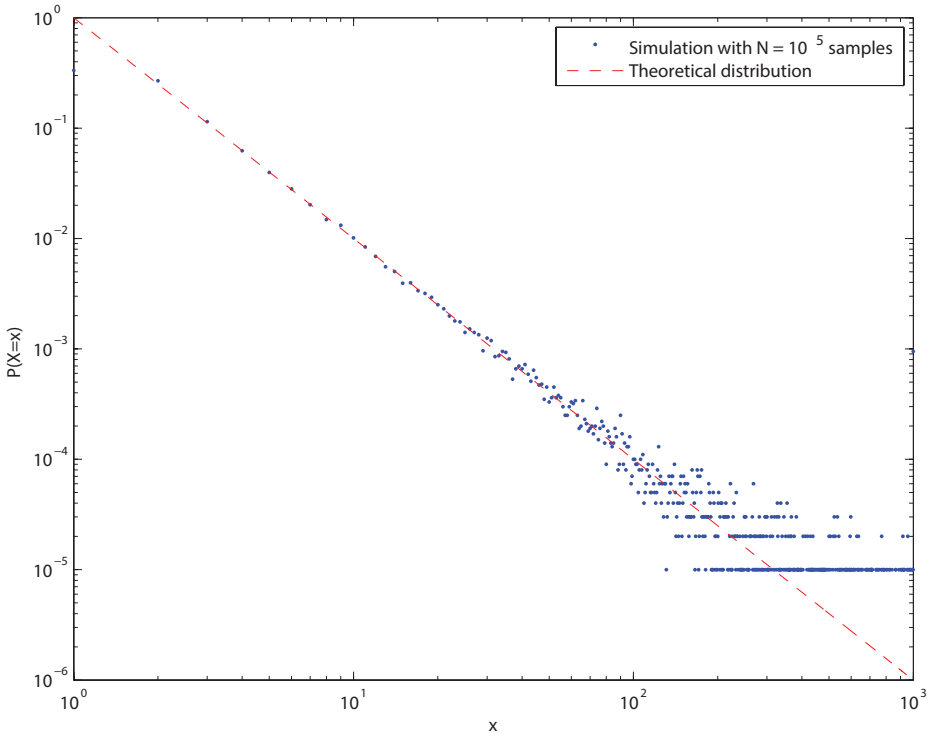

Figure 4: Simulation (blue dots) of the probability distribution $P(x) = \frac{\alpha-1}{x_{\min}} \left(\frac{x}{x_{\min}}\right)^{-\alpha}$ (red dashed line).

2 Empirical power laws

2.1 Complementary CDF

First, we find the CDF $C(x)$:

$$C(x) = \int_{x_{\min}}^{x} P(x')dx' = \int_{x_{\min}}^{x} \frac{\alpha-1}{x_{\min}} \left(\frac{x'}{x_{\min}}\right)^{-\alpha} dx', \tag{5}$$

$$= 1 - \left(\frac{x}{x_{\min}}\right)^{-(\alpha-1)}, \tag{6}$$

when $x \geq x_{\min}$, otherwise $C(x) = 0$. (We assume $\alpha \neq 1$. In that case, the stated pdf is identically zero for all x.)

The complementary CDF (CCDF) is given by:

$$\bar{C}(x) = 1 - C(x) = \left(\frac{x}{x_{\min}}\right)^{-(\alpha-1)}, \tag{7}$$

when $x \geq x_{\min}$, otherwise $\bar{C}(x) = 1$.

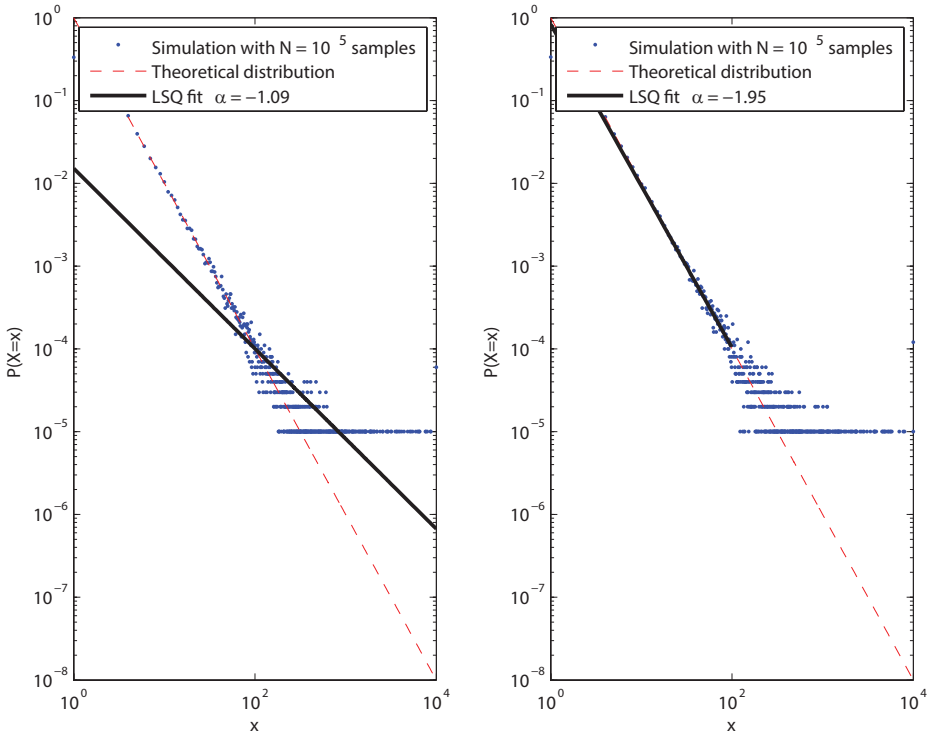

Figure 5: Simulation (blue dots) of the probability distribution $P(x) = \frac{\alpha-1}{x_{\min}} \left(\frac{x}{x_{\min}}\right)^{-\alpha}$ (red dashed line).

2.2 Simulated dataset with $\alpha = 2$ and $x_{\min} = 1$

Since we have the explicit form of the CDF, we can generate samples from $P(x)$ by applying the inverse transform sampling method: the random variable $X = C^{-1}(U)$ will be distributed according to $F(x)$, when C^{-1} is the inverse CDF and U is a uniform random variable.

Using this procedure, I simulated 10^5 draws from $P(x)$ as shown in Fig. 4. I simply omitted the points x for which the estimate of $P(X = x)$ is zero. Practically, including such points in the log-log plot would mess up the vertical scale of the plot.

2.3 Least squares fit for α on PDF

When I use all (nonzero) histogram data to find the exponent, I find $\hat{\alpha}_{\text{hist}} \approx -1.1$. However, as shown in Fig. 5(a), this is a terrible fit for the data. The least squares routine is being fooled by the noisy data points in the tail of the data.

We can get a better visual fit through the data by omitting the noisy heavy-tail portion of the simulation ($x > 10^2$) in the least squares fit. (I chose the cutoff to be $x = 10^2$ arbitrarily.) The result is shown in Fig. 5(b), which yields $\hat{\alpha}_{\text{hist}} = -1.95$, which is much closer to the simulation parameter.

2.4 Least squares fit for α on CCDF

Based on the loglog fit to the simulated CDF, I find $\hat{\alpha}_{\text{CDF}} = 1.999$.

2.5 MLE estimate of α

The MLE estimate of α is given by:

$$\hat{\alpha}_{\text{MLE}} = 1 + n \left[\sum_{i=1}^{n} \log \left(\frac{d_i}{x_{\min}} \right) \right]^{-1} \tag{8}$$

which, when applied to my dataset, revealed $\hat{\alpha}_{\text{MLE}} = 2.002$.

2.6 Comparison of estimation methods

I ran the simulation and fitting procedures 100 times. The results are as follows. Note that the standard deviation, rather than the variance, is cited:

- Fitting the histogram (Section 2.3, with no arbitrary cutoff): $\hat{\alpha} \approx 1.0884 \pm 0.0287$.
- Fitting the CCDF (Section 2.4): $\hat{\alpha} \approx 1.9902 \pm 0.0296$.
- Computing the MLE (Section 2.5): $\hat{\alpha} \approx 1.9994 \pm 0.0028$.

Clearly, the fitting of the entire histogram is highly susceptible to the noise in the long tail (Fig 5), and yields a wildly inaccurate result.

On the other hand, the means of the CCDF and the MLE method both yield good estimates of the true α. However, it appears that the MLE is slightly more accurate, and is more consistent (smaller standard deviation). The performance of the MLE is intuitively sensible, since the MLE model presumes a correct model of the distribution, whereas the fitting methods are "not aware" of the underlying distribution model.

```
% Tony Hyun Kim
% CS 224w: PS3, Problem 2
% clear all; close all;
function [alphac, alphad, alphae] = p2b()

% Part b: Generate a dataset of 100,000 values
%-------------------------------------------------------------
N = 1e5;
U = rand(N,1);

xmin = 1;
alpha = 2;

X = xmin*(1-U).^(-1/(alpha-1));

x = xmin:1:1e4;
[h, x] = hist(X,x);
sieve = h>0;
p = h(sieve)/N;
x = x(sieve);

% subplot(121);
% loglog(x,p,'.');
% % grid on;
% xlabel('x');
% ylabel('P(X=x)');
% hold on;

% Theoretical distribution
% p_cont = (alpha-1)/xmin*(x/xmin).^-alpha;
% plot(x,p_cont,'r--');

% Part c: Fit the histogram
%-------------------------------------------------------------
xc = x;
pc = p;

Pc = polyfit(log(xc),log(pc),1);
pc_cont = exp(polyval(Pc,log(xc)));
% plot(xc,pc_cont,'k-','Linewidth',2);

alphac = abs(Pc(1)); % Alpha estimated by linear fit to histogram

% Part d: Fit the CCDF
%-------------------------------------------------------------
ccdf = 1-cumsum(p);

% The last entry point is spurious
xd = x(1:end-1);
ccdf = ccdf(1:end-1);

% subplot(122);
% loglog(xd,ccdf,'.');
% xlabel('x');
% ylabel('CCDF');
% hold on;
```



```
Pd = polyfit(log(xd),log(ccdf),1);
pd_cont = exp(polyval(Pd,log(xd)));
% plot(xd,pd_cont,'k-','Linewidth',2);

alphad = abs(Pd(1)-1); % Alpha estimated by linear fit to CCDF

% Part e: Use the MLE formula
%-------------------------------------------------------------
alphae = 1 + N/sum(log(X/xmin));
```

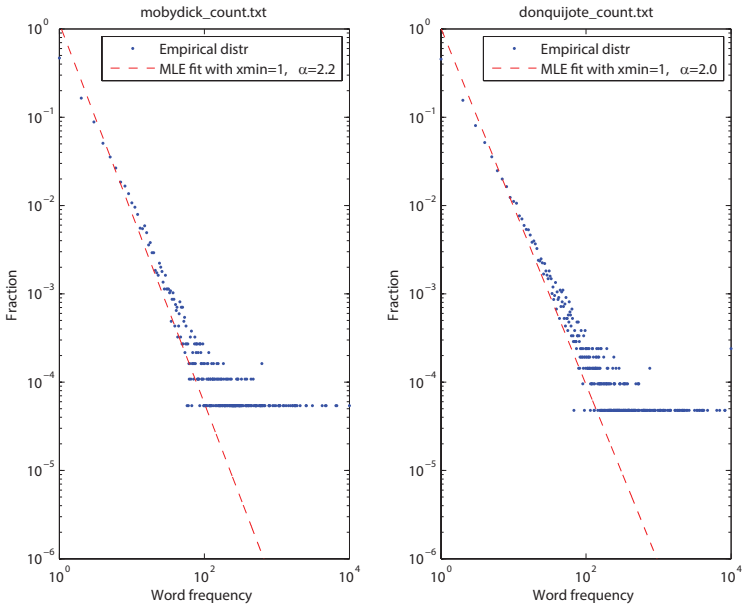

Figure 6: Word frequency distribution from Moby Dick and Don Quijote sources. The MLE fit (red dashed line) uses the formulas from the continuous power-law distribution.

3 Combination of exponentials for generating power laws

I wrote a simple Python program to generate the word counts of the two source files. I then used Matlab to plot and analyze the data.

3.1 Empirical power laws in word-frequency distributions

The empirical distributions are shown in Fig. 6. I also plotted the MLE power-law distribution (dashed lines) assuming $x_{\min} = 1$, since words with this minimum frequency are present in the empirical distribution. Note that MLE estimate of α (Eq. 8) was derived from a continuous power-law distribution whereas we are currently dealing with a discrete distribution. This results in the relatively poor performance of the MLE fit in describing the data.

On the other hand, in Fig. 7, I show the results of the MLE fit assuming a *discrete* power-law distribution. I continued to use $x_{\min} = 1$ since such frequencies arise in the real data. On the other hand, I numerically calculated the necessary normalization coefficient at each value of α assuming a discrete distribution. It can be readily observed that the MLE fits assuming a discrete model performs significantly better than the continuous model.

The fitted parameters (using the discrete model) are:

- Moby Dick: $x_{\min} = 1$, and $\hat{\alpha} = 1.74$.
- Don Quijote: $x_{\min} = 1$, and $\hat{\alpha} = 1.69$.

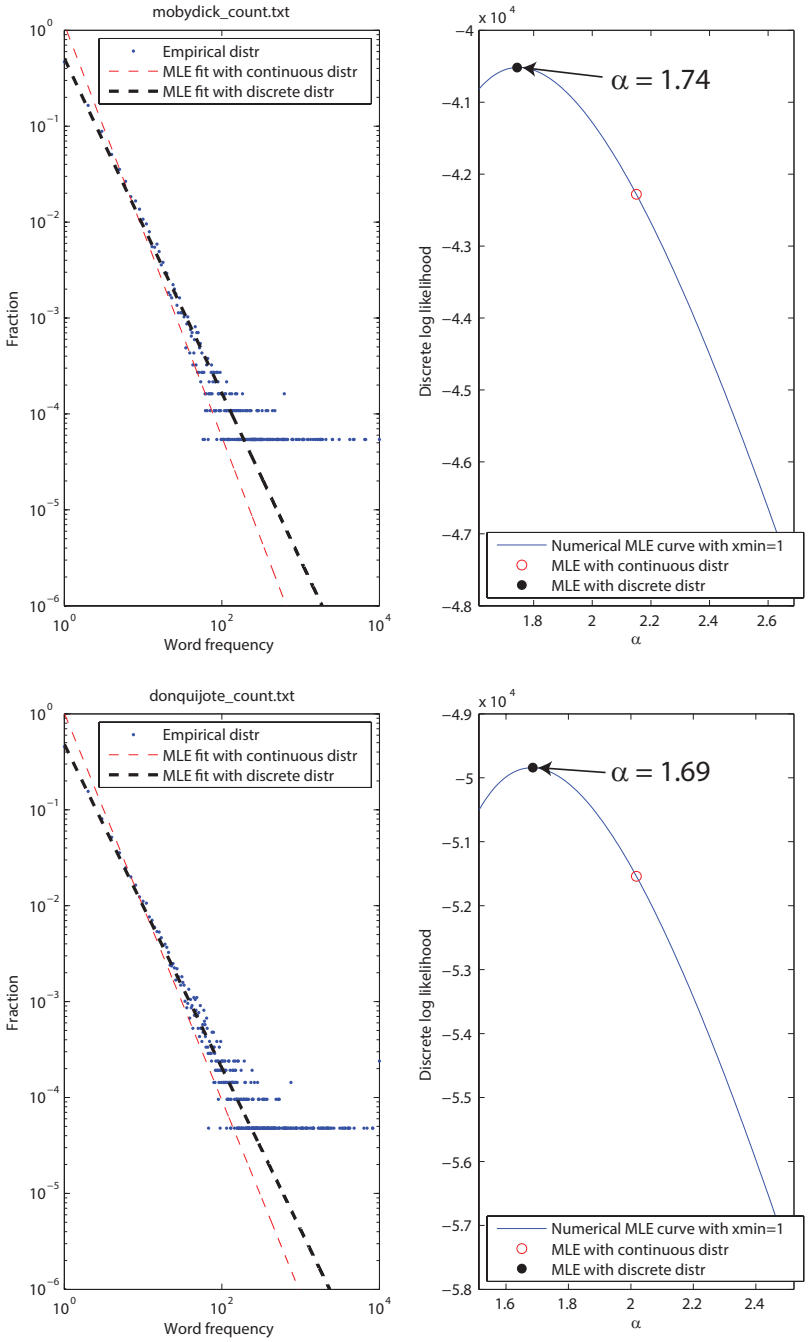

Figure 7: Word frequency distribution from Moby Dick and Don Quijote sources. The parameter $x_{\min}$ was fixed at 1, while the MLE curve was computed as a function of α with the normalization constant appropriate for a discrete power-law distribution. It can be seen that the MLE fit on the discrete model (black curve) performs better than the one based on the continuous model (red curve).

```
% Tony Hyun Kim
% CS 224w, Problem 3.1

%============================================================
% Python code to compute the word frequencies:
%------------------------------------------------------------
% # Tony Hyun Kim
% # CS 224w, Problem 3(a):
% # Perform word count on given corpus
%
% from collections import defaultdict
%
% source = "donquijote"
% fin = open(source+".txt",'r')
%
% wordcount = defaultdict(int)
% for line in fin:
%     word = line.strip()
%     wordcount[word] += 1
%     #print word
%
% fout = open(source+"_count.txt",'w')
% for word, count in wordcount.items():
%     fout.write(str(count)+'\n')
%     #print word + " - " + str(count)
%============================================================

clear all; close all;

source = 'mobydick_count.txt';
% source = 'donquijote_count.txt';
data = load(source); % Frequency of each word
N = length(data);

% Plot the empirical distribution
%------------------------------------------------------------
x = 1:1e4;
h = hist(data,x);
p = h/N;
subplot(121);
loglog(x,p,'.');
title(strrep(source,'_','\_'));
xlabel('Word frequency');
ylabel('Fraction');
hold on;

% MLE fit
%------------------------------------------------------------
xmin  = 1; % Since we observe words with this frequency

% Formulas from the continuous distribution
alpha = 1 + N/sum(log(data/xmin));
p_cont = (alpha-1)/xmin*(x/xmin).^-alpha;

% Perform numerical MLE fit based on discrete distribution
disc_alphas = linspace(0.75*alpha, 1.25*alpha);
```



```
betas = zeros(size(disc_alphas)); % Normalization factors for the discrete distr
LLs   = zeros(size(disc_alphas)); % Log likelihood
for i = 1:length(disc_alphas)
    disc_alpha = disc_alphas(i);
    betas(i) = 1/sum((x/xmin).^-disc_alpha);
    LLs(i)   = sum(log(betas(i).*(data/xmin).^-disc_alpha));
end

subplot(122);
plot(disc_alphas,LLs);
xlim([disc_alphas(1) disc_alphas(end)]);
xlabel('\alpha');
ylabel('Discrete log likelihood');

hold on;
% Show the MLE estimate assuming continuous distribution
plot(alpha,interp1(disc_alphas,LLs,alpha,'linear'),'ro');

% Show the MLE estimate assuming discrete distribution
[~, ind] = max(LLs);
disc_alpha = disc_alphas(ind);
beta       = betas(ind);
mll        = LLs(ind);
plot(disc_alpha, mll, 'k.', 'MarkerSize', 18);

legend('Numerical MLE curve with xmin=1',...
       'MLE with continuous distr',...
       'MLE with discrete distr',...
       'Location','SouthWest');

p_disc = beta*(x/xmin).^-disc_alpha;

% Show the fits on the original graph
subplot(121);
plot(x,p_cont,'r--');
plot(x,p_disc,'k--','LineWidth',2);
ylim([10^-6 10^0]);
legend('Empirical distr',...
       'MLE fit with continuous distr',...
       'MLE fit with discrete distr',...
       'Location','NorthEast');
```

3.2 Theoretical power laws in monkey novels

3.2.1 Distinct words of length y

Given m letters, there are $m^y = e^{\log(m)y}$ distinct words of length y. (This is a permutation with replacement.)

3.3 Relative frequency of a particular word of length y

Given a particular word of length y, the monkey needs to hit the y-long sequence of letters and then terminate with a space. So, the relative frequency x is given by:

$$x = \left(\frac{1-q_s}{m}\right)^y \cdot q_s = q_s \cdot e^{\left(\log \frac{1-q_s}{m}\right)y}. \tag{9}$$

3.4 Probability of a word having relative frequency x

4 Exploring robustness of networks

Initially, I considered using `SNAP.py` for generating the preferential attachment (PA) graph. However, since the `TUNGraph` does not have a method to return a random edge in the graph, I would have to maintain a separate edgelist outside of the graph object, which just does not feel right. So, I just used Matlab to generate the PA graph.

4.1 Diameter robustness

Fig. 8 shows the diameter as a function of fraction of nodes removed under the failure (triangle) and attack (circle) deletion policies. Note that the graph diameter initially increases as nodes are removed (because potential paths between nodes are eliminated), but that the diameter decreases again once large amounts of nodes have been removed (as the graph becomes fragmented).

Analysis of failure vs. attack scenarios for each network:

- G_{nm} graph: In Fig. 8(a), it can be seen that the variation in graph diameter is essentially identical under the failure and attack scenarios. This makes sense since, in the random graph, the nodes are mostly homogeneous (*i.e.* nodes possess roughly the same degree). In Fig. 8(b), increasing failure is shown to slowly increase the diameter of the graph.
- AS graph: In contrast to G_{nm}, the AS graph is highly susceptible to the attack scenario. In Fig. 8(a), it can be seen that the diameter reaches its maximum when only 2% of the nodes are attacked (afterwards, the AS graph becomes fragmented). On the other hand, the AS graph has excellent random failure tolerance. In Fig. 8(b), it is shown that the diameter of the AS graph remains constant (at about 10) even when 50% of the graph is randomly removed!
- PA graph: Like the AS graph, the PA graph is susceptible to the attack policy. On the other hand, the PA graph does not appear as inhomogeneous as the AS graph, as 10% of the graph needs to be attacked in order to fragment the AS graph (compared to only 2% for the AS case).

Comparison of the behavior of the different networks under the two deletion policies. We can conceptually rank the three graphs according to homogeneity $\leftrightarrow$ inhomgeneity:

- The random graph G_{nm} is the most homogeneous, and is relatively robust against targeted attacks. On the other hand, the diameter slowly suffers as more and more nodes fail (Fig. 8(b)).
- Based on the results of the attack simulation, the AS graph is the most inhomogeneous; that is, the AS graph has a few nodes that are crucially important for the connectivity. The AS graph is highly susceptible to attack (on the few important nodes) but very resilient against random failures (since most nodes do not contribute significantly to the connectivity of the graph).
- The PA graph is intermediate between the G_{nm} and AS graphs.

4.2 |LCC| robustness

Fig. 9 shows the size of the largest connected component (|LCC|) as a function of fraction of nodes removed under failure (triangle) and attack (circle) deletion policies.

For each graph, the attack policy is more successful at decreasing the |LCC| than the failure policy. This makes intuitive sense, since nodes with large degree are likely to be part of the LCC.

Under the attack policy, it is seen that the |LCC| of the AS graph is diminished first, then the PA graph, and then the G_{nm} graph. This ordering is consistent with the idea that the AS graph is the most inhomogeneous network of the three. In an inhomogeneous network, it is relatively easy to fragment the LCC by targeting the nodes with high degrees.

Interestingly, even under the random failure policy, the G_{nm} graph appears to be the most robust according to the |LCC| metric. This is contrary to the diameter measure of Fig. 8(b) where the diameter of the AS graph is robust against random failures. This suggests that it is more difficult to disconnect nodes from the LCC in a homogeneous graph than in an inhomogeneous graph.

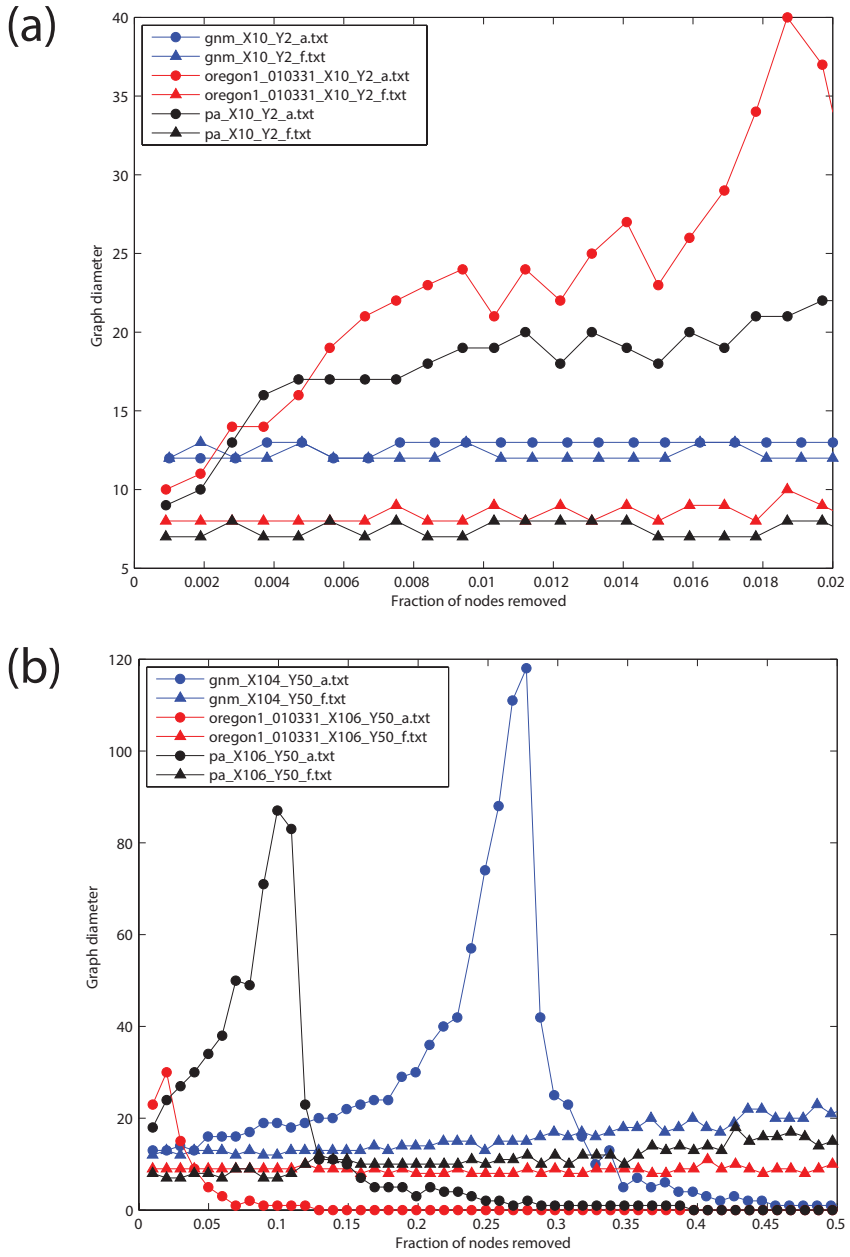

Figure 8: The graph diameter as a function of the number of nodes removed. Two deletion policies are considered: "Attack"-mode (circles) where nodes with highest degree are targeted first; "Failure"-mode (triangles) where nodes are removed at random. [a] The variation in graph diameter in the initial 2% removal. [b] The variation in graph diameter as up to 50% of the nodes are removed.

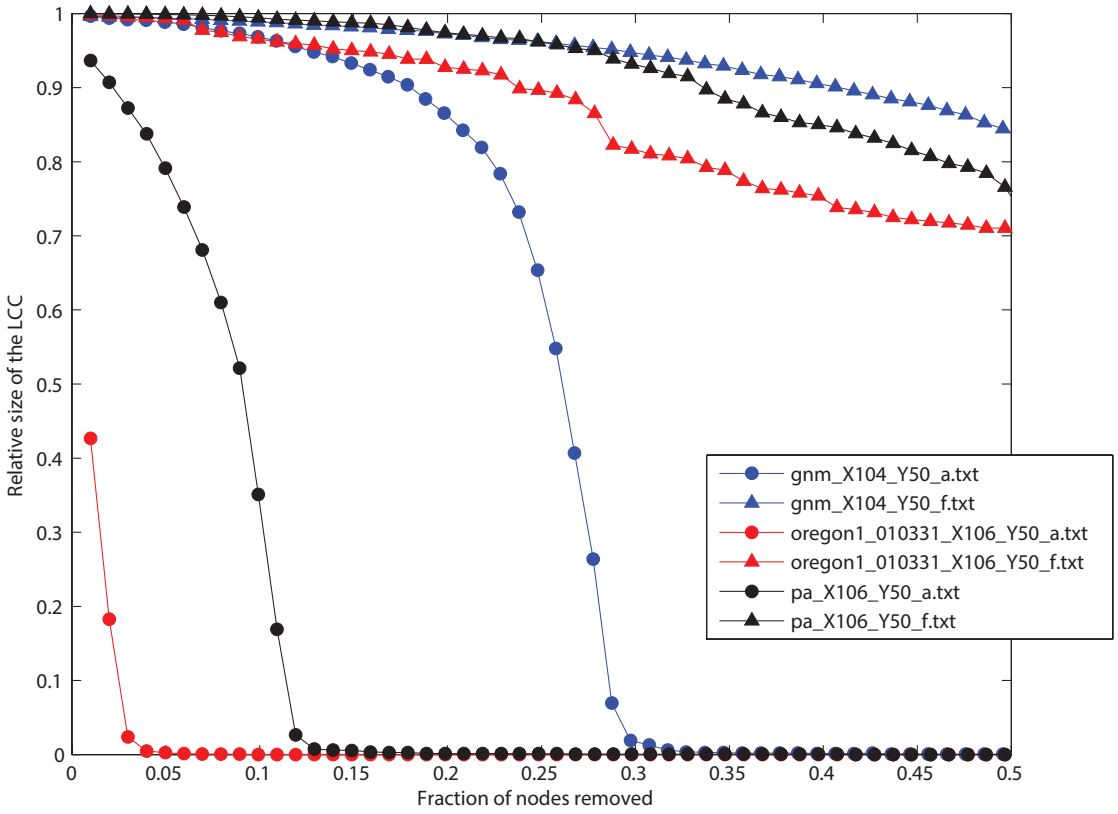

Figure 9: The size of the largest connected component as a function of the number of nodes removed. Two deletion policies are considered: "Attack"-mode (circles) where nodes with highest degree are targeted first; "Failure"-mode (triangles) where nodes are removed at random.

p4_gen.py

```python
# Tony Hyun Kim
# CS 224w, PS 3, Problem 4
# Generate the graphs and save to file
import random
import snap

# Target parameters
n = 10670 # Number of nodes
m = 22002 # Number of edges

# Generate the random network
#-------------------------------------------------------------
G1 = snap.TUNGraph.New()

for nid in range(n):
    G1.AddNode(nid)

x = 0 # Counter for number of random edges added
while (x < m):
    n1 = random.randint(0, n-1)
    n2 = random.randint(0, n-1)
    if (not G1.IsEdge(n1,n2) and (n1 != n2)):
        G1.AddEdge(n1,n2)
        x += 1

snap.SaveEdgeList(G1, "gnm.txt")
```



```
% Tony Hyun Kim
% CS 224w, PS 3, Problem 4
% Generate the preferential attachment graph
clear all; close all;

n = 10670; % Number of nodes
m = 22040; % (Maximum) number of edges

edges = zeros(m,2);
ei = 0; % Number of edges currently in list

% Begin with a complete graph of 40 nodes
%-------------------------------------------------------------
n_seed = 40;
for ni = 0:(n_seed-2)
    for nj = (ni+1):(n_seed-1)
        ei = ei + 1;
        edges(ei,:) = [ni nj];
    end
end

% Next, we add new nodes that make two random, preferential edges
%   to the existing network
%-------------------------------------------------------------
for ni = n_seed:(n-1)            % Each incoming node
    for k = 1:2                  %     makes two edges by
        re = randi(ei);          %     choosing a random prior edge, and
        nj = edges(re,randi(2)); %     choosing one of the endpoints

        ei = ei + 1;
        edges(ei,:) = [ni nj];
    end
end

dlmwrite('pa.txt', edges, 'delimiter', ' ');
```

```
# Tony Hyun Kim
# CS 224w, PS 3, Problem 4a
from __future__ import division # Non-truncating division
import snap

# Load the graph
#source = "gnm"
#source = "oregon1_010331"
source = "pa"

G  = snap.LoadEdgeList(snap.PUNGraph, source+".txt", 0, 1)
n0 = G.GetNodes() # Number of nodes in graph

# Deletion policy
scenario = 'f' # f: failure, a: attack

X = n0//100
Y = 50

# Output file
filename = source+"_X{}_Y{}_".format(X,Y)+scenario+".txt"
f = open(filename, 'w')

num_nodes_deleted = 0
while (num_nodes_deleted/n0 < Y/100):
    # Delete nodes in batches of X
    for _ in range(X):
        # Scenarios:
        #   In 'Attack's, delete nodes with maximum degree
        #   In 'Failure's, delete nodes at random
        if (scenario == 'a'):
            nid = snap.GetMxDegNId(G)
        else:
            nid = G.GetRndNId()
        G.DelNode(nid)
    num_nodes_deleted += X

    # Part a: Compute the diameter
    '''
    diam = snap.GetBfsFullDiam(G, 20)
    f.write("{0:.4f} {1}\n".format(num_nodes_deleted/n0, diam))
    '''

    # Part b: Fraction of nodes in the largest (strongly)
    #           connected component of a graph
    lcc_frac = snap.GetMxWccSz(G)
    f.write("{0:.4f} {1:.4f}\n".format(num_nodes_deleted/n0, lcc_frac))

print "Saved result to {}".format(filename)
f.close()
```

```
% Tony Hyun Kim
% CS 224w, PS 3, Problem 4
% Script to plot the graph parameters under attack/failure
clear all; close all;

graphs = {'gnm',...
          'oregon1_010331',...
          'pa'};
scenarios = ['a', 'f'];
Y = 50;

Ng = length(graphs);
Ns = length(scenarios);

labels = cell(Ng*Ns,1);
li = 1;

colors  = 'brk';
markers = 'o^';
for gi = 1:Ng
    for si = 1:Ns
        file = dir(sprintf('%s_*_Y%d_%s.txt',...
                           graphs{gi}, Y, scenarios(si)));
        filename = file.name;
        data = load(filename);
        plot(data(:,1),data(:,2),...
             strcat(colors(gi),markers(si),'-'),...
             'MarkerFaceColor',colors(gi));
        hold on;
        labels{li} = strrep(filename,'_','\_');
        li = li + 1;
    end
end

xlim([0 Y/100]);
xlabel('Fraction of nodes removed');
ylabel('Relative size of the LCC');
legend(labels,'Location','SouthEast');
```