STAT542 Homework 7

Yu-Ching Liao

April 25, 2023

1 Problem 1

1.1 Q1

Let X_n be the random variable representing the number of occurrences of the word "popular" after adding n new random words. At the initialization, the string is "popular language", so the number of occurrences of "popular" is 1. We can recursively define the probability mass function of X_n as follows:

$$P(X_n = k) = \frac{k-1}{n+1} P(X_{n-1} = k-1) + \frac{n+2-k}{n+1} P(X_{n-1} = k) \tag{1}$$

for $n \geq 1$, $k = 1, 2, \ldots, n+2$, and with the initial condition $P(X_0 = 1) = 1$.
The recursion formula can be explained as follows:

1. With probability $\frac{k-1}{n+1}$, the $(n+1)$-th word is "popular", and there were already $k-1$ occurrences of "popular" in the first n words.
2. With probability $\frac{n+2-k}{n+1}$, the $(n+1)$-th word is not "popular", and there were already k occurrences of "popular" in the first n words.

By applying this recursion formula for n steps, you can compute the distribution of X_n, the number of occurrences of "popular" after adding n new random words.

1.2 Q2

Let X_n be the number of occurrences of the word "popular" after n new words are added, and let $X := \lim_{n\to\infty} X_n$. We want to find the distribution of X.

Observe that, in each iteration, the probability of adding the word "popular" is equal to the fraction of "popular" occurrences in the current string. Thus, the probability of adding "popular" at step $n+1$ is $\frac{X_n}{n+4}$. Therefore, the process can be modeled as a Polya's urn scheme with two types of balls: "popular" and "others", with initial counts of 1 and 3, respectively.

The limiting distribution of the fraction of "popular" balls in the urn converges to a Beta distribution. In this case, the parameters of the Beta distribution are $\alpha = 1 + 1 = 2$ and $\beta = 3 + 1 = 4$, as we add 1 to each initial count. Thus, the limiting distribution of the fraction of "popular" words, say Y, is $Y \sim Beta(2, 4)$.

Since $X = \lim_{n\to\infty} X_n = \lim_{n\to\infty}(n+4)Y$, the limiting distribution of X is obtained by multiplying the limiting distribution of Y by the total number of words. However, the distribution of X will not have a closed-form expression, as the total number of words is not fixed.

In conclusion, the limiting distribution of the fraction of "popular" words is $Y \sim Beta(2, 4)$. The limiting distribution of X will not have a closed-form expression, but it is obtained by scaling the Beta distribution by the total number of words in the sequence.

2 Problem 2

2.1 Q1

Independent Set Size	Number of Independent Sets
1	5
2	5
3	0
4	0
5	0

2.2 Q2

To show that $P(x) = \prod_{(i,j)\in E} \phi_{i,j}(x_i, x_j)$, we will define the functions $\phi_{i,j}(x_i, x_j)$ using λ. Let us consider the edge set E of the pentagon graph. We know that $x_i = 1$ if the i-th vertex is in the subset, and $x_i = 0$ otherwise. If two vertices i and j are adjacent, we know that they cannot both be in the independent set. Hence, for adjacent vertices i and j, we must have $x_i x_j = 0$.

$$P(x) = \frac{\lambda^{\|x\|_1}}{Z} \cdot 1\{x \text{ is an independent set}\}$$

Define the functions $\phi_{i,j}(x_i, x_j)$ as follows:

$$\phi_{i,j}(x_i, x_j) = \begin{cases} \lambda^{x_i + x_j}, & \text{if } x_i x_j = 0 \\ 0, & \text{if } x_i x_j = 1 \end{cases}$$

Now, rewrite the probability distribution $P(x)$ as a product of these functions:

$$P(x) = \frac{\lambda^{\|x\|_1}}{Z} \cdot 1\{x \text{ is an independent set}\} = \frac{1}{Z} \prod_{(i,j)\in E} \phi_{i,j}(x_i, x_j)$$

Here, the product over all edges $(i, j) \in E$ ensures that the condition $x_i x_j = 0$ is satisfied for all adjacent vertices, making x an independent set. The normalization constant Z ensures that the distribution sums to 1.

Thus, we have shown that $P(x) = \prod_{(i,j)\in E} \phi_{i,j}(x_i, x_j)$ with the defined functions $\phi_{i,j}(x_i, x_j)$ using λ.

3 Problem 3

3.1 Q1

To show that the sum of the PageRanks p_i is N, the number of web pages, we can start by summing both sides of equation 14.108:

$$p = (1 - d)e + d \cdot LD_c^{-1} * p$$

Let's denote the sum of the PageRanks as S, i.e., $S = \sum_{i=1}^{N}(p_i)$. We can then sum both sides of the equation:

$$\sum_{i=1}^{N}(p_i) = \sum_{i=1}^{N}((1 - d)e_i + d \cdot \sum_{j=1}^{N}(\frac{L_{ij}}{c_j} * p_j))$$

Since e is a vector of N ones, summing e_i over i results in N:

$$\sum_{i=1}^{N}(e_i) = N$$

Now let's focus on the right-hand side of the equation. We can split the sum into two parts:

$$S = \sum_{i=1}^{N}((1-d)e_i) + d \cdot \sum_{i=1}^{N}(\sum_{j=1}^{N}(\frac{L_{ij}}{c_j} * p_j))$$

Using the previous result, we can simplify the first term:

$$\sum_{i=1}^{N}((1-d)e_i) = (1-d) \cdot \sum_{i=1}^{N}(e_i) = (1-d) \cdot N$$

Now let's consider the second term:

$$d \cdot \sum_{i=1}^{N}(\sum_{j=1}^{N}(\frac{L_{ij}}{c_j} * p_j))$$

We can interchange the order of summation:

$$d \cdot \sum_{j=1}^{N}(\sum_{i=1}^{N}(\frac{L_{ij}}{c_j} * p_j))$$

Now, observe that $\sum_{i=1}^{N}(L_{ij}) = c_j$, as it counts the number of outgoing links from page j. Therefore, we can simplify the inner sum:

$$\sum_{i=1}^{N}(\frac{L_{ij}}{c_j} * p_j) = \sum_{i=1}^{N}(\frac{L_{ij} * p_j}{c_j}) = \frac{1}{c_j} \cdot \sum_{i=1}^{N}(L_{ij} * p_j) = \frac{1}{c_j} \cdot c_j * p_j = p_j$$

So, the second term becomes:

$$d \cdot \sum_{j=1}^{N}(p_j)$$

Now, we substitute the two terms back into the equation for S:

$$S = (1-d) \cdot N + d \cdot \sum_{j=1}^{N}(p_j)$$

Since the sum of the PageRanks is S, we have:

$$S = (1-d) \cdot N + d \cdot S$$

Rearranging the terms, we get:

$$S - d \cdot S = (1-d) \cdot N$$

Factoring out S, we have:

$$S \cdot (1-d) = (1-d) \cdot N$$

Since d is a positive constant (less than 1), we can safely divide both sides by $(1-d)$:

$$S = N$$

Thus, we have shown that the sum of the PageRanks p_i is equal to N, the number of web pages.

3.2 Q2

```python
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

def create_transition_matrix(adjacency_matrix, d=0.85):
    N = adjacency_matrix.shape[0]
    c = np.sum(adjacency_matrix, axis=0)
    D_c_inv = np.diag(1 / c)
    transition_matrix = d * adjacency_matrix.dot(D_c_inv) + (1 - d) / N
    return transition_matrix

def power_method(transition_matrix, max_iter=1000, tolerance=1e-6):
    N = transition_matrix.shape[0]
    p = np.ones(N) / N
    for _ in range(max_iter):
        p_next = transition_matrix.dot(p)
        if np.linalg.norm(p_next - p, ord=1) < tolerance:
            break
        p = p_next
    return p

def draw_network(adjacency_matrix, node_labels=None):
    G = nx.from_numpy_array(adjacency_matrix.T, create_using=nx.DiGraph)
    pos = nx.shell_layout(G)

    plt.figure(figsize=(8, 8))
    nx.draw_networkx_nodes(G, pos, node_shape='s', node_size=3000, node_color="
        lightblue", edgecolors="black", linewidths=2)
    nx.draw_networkx_edges(G, pos, arrowsize=20, node_size=3000, arrowstyle='->',
        width=2)

    if node_labels:
        nx.draw_networkx_labels(G, pos, labels=node_labels, font_size=14)

    plt.axis('off')
    plt.margins(x=0.1, y=0.1)
    plt.show()

# Define the adjacency_matrix variable according to the network from Figure 14.47
adjacency_matrix = np.array([
    [0, 0, 1, 1, 0, 0],
    [1, 0, 0, 0, 0, 1],
    [1, 1, 0, 1, 1, 0],
    [0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 1],
    [0, 0, 0, 0, 0, 0]
], dtype=float)

transition_matrix = create_transition_matrix(adjacency_matrix)
page_ranks = power_method(transition_matrix)

print("PageRanks:", page_ranks)

node_labels = {i: f"Page {i+1}" for i in range(adjacency_matrix.shape[0])}
draw_network(adjacency_matrix, node_labels=node_labels)
```

Here is the output of my Python code:

```
PageRanks: [0.35390083 0.18603247 0.3744417  0.025      0.035625   0.025     ]
```

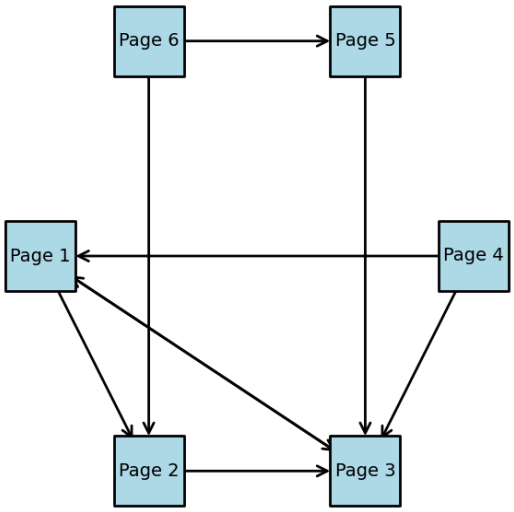

Figure 1: My image