Maximization of the distribution entropy of possible pathes

Yiliang Zhang

July 18, 2015

School of Mathematical Sciences
Peking University
address: 315, 28#
email: 1300010719@pku.edu.cn

Abstract

We define the discrete state space as $\{0, 1, 2, 3, \dots, N-1, N\}$ and we study $X_s (s = 0, 1, 2, \dots, T)$, which takes value in $\{0, 1, 2, 3, \dots, N-1, N\}$. Also, there are some constraints for the distribution. In this article, we find the distribution of the path with the largest entropy under the constraints by technique of chain rule and lagrange.

Keywords: Markov chain, entropy, chain rule

1 A Lemma

N and m are given integers. There are two sequences $\{a_i\}$ and $\{b_i\}$ with $a_i, b_i \geq 0$ $(0 \leq i \leq N)$. p_i is a probability distribution subject to the constraints $\sum_{i=0}^N ip_i = m$ and $\sum_{i=0}^N p_i = 1 (0 \leq i \leq N)$. Set the probability distribution which maximizes $\sum_{i=0}^N a_i p_i - \sum_{i=0}^N p_i \log p_i$ as $p_i(a)$ and the probability distribution which maximizes $\sum_{i=0}^N b_i p_i - \sum_{i=0}^N p_i \log p_i$ as $p_i(b)$. The value of maximized $\sum_{i=0}^N (\cdot)_i p_i - \sum_{i=0}^N p_i \log p_i$ is set to be $\varepsilon(\cdot)$. So we have a lemma.

Lemma. For $\forall i$ subject to $0 \leq i \leq N$, if $a_i \leq b_i$, then $\varepsilon(a) \leq \varepsilon(b)$.

Proof. We can gain from $a_i \leq b_i$ that

$$\varepsilon(a) = \sum_{i=0}^N a_i p_i(a) - \sum_{i=0}^N p_i(a) \log p_i(a) \leq \sum_{i=0}^N b_i p_i(a) - \sum_{i=0}^N p_i(a) \log p_i(a). \tag{1}$$

However, we know $p_i(b)$ maximizes $\sum_{i=0}^N b_i p_i - \sum_{i=0}^N p_i \log p_i$, so

$$\sum_{i=0}^N b_i p_i(a) - \sum_{i=0}^N p_i(a) \log p_i(a) \leq \sum_{i=0}^N b_i p_i(b) - \sum_{i=0}^N p_i(b) \log p_i(b) = \varepsilon(b). \tag{2}$$

That is

$$\varepsilon(a) \leq \varepsilon(b). \tag{3}$$

2 Chain Rule

Now we want to find the distribution of possible pathes that maximizes the entropy. We know from chain rule that

$$H(X, Y) = H(X) + H(Y|X). \tag{4}$$

So, now we set X to be the random variable on $\{1, 2, 3, \dots, N-1, N\}$ at step s, which means the distribution of X is $p_s(i, \cdot)$. And Y is the possible pathes after step $s+1$. Define $e_s(i)$ is the entropy of possible pathes to end start from i at step s. We have

$$e_s(i) = -\sum_j p_s(i,j)\log p_s(i,j) + \sum_j p_s(i,j)e_{s+1}(j). \tag{5}$$

Also, we have

$$e_{T-1}(i) = -\frac{i}{N}\log\frac{i}{N} - \frac{N-i}{N}\log\frac{N-i}{N}. \tag{6}$$

By the method of Lagrange Multipliers, we can maximize every $e_s(i)$. In that way, we can calculate $p_s(i,j)$ by backwards induction. From the lemma we mentioned in section 1, we can proof that the $p_s(i,j)$ we solved by this method maximizes the entropy of possible pathes.

3 Computer Code

In this section, We provide the computer code we've written to solve the problem.

```
__author__ = 'Administrator'

from demo import *

from math import *


def get_distribution(m = 5, N = 10, b = [i*(10-i)/5 for i in range(11)]):
    lam = sol_lambda(m, N, b)
    c = sol_c(lam, N, b)
    distribution = []
    for j in range(len(b)):
        distribution.append(c * e ** (b[j] - lam * j))
    return distribution
#print(get_distribution())


def get_entropy(distribution, b):
    value = 0
    for i in range(len(b)):
        value += b[i] * distribution[i] - distribution[i] * \
                                         log(distribution[i])
    return value
```

```
class Markov:
    def __init__(self, matrix, t, entropy):
        self.jie = len(matrix)
        self.matrix = matrix
        self.time = t
        self.entropy = entropy

    def backwards_gen_distribution(self, n):
        return get_distribution(n, self.jie - 1, self.entropy)

    def backwards_gen(self):
        t = self.time - 1
        matrix = []
        entropy = []
        for k in range(self.jie):
            distribution = get_distribution(k, self.jie - 1,
                                            self.entropy)
            value  = get_entropy(distribution, self.entropy)
            matrix.append(distribution)
            entropy.append(value)
        return Markov(matrix, t, entropy)


def compound(m1, m2):
    matrix = []
    for i in range(len(m1)):
        row = []
        for j in range(len(m2[0])):
            s = 0
            for k in range(len(m1[0])):
                s += m1[i][k] * m2[k][j]
            row.append(s)
        matrix.append(row)
    return matrix


def gen_last_matrix(N = 10, T = 10):
    t = T - 1
    matrix = []
    entropy = []
    for i in range(N + 1):
        distribution = []
        distribution.append((N - i) / N)
        for j in range(1, N):
            distribution.append(0)
        distribution.append(i / N)
        matrix.append(distribution)
        if 1 in distribution:
            value = 0
        else:
            value = - (i / N) * log(i / N) - ((N - i) / N) * \
                                                log((N - i) / N)
        entropy.append(value)
    return Markov(matrix, t, entropy)


def gen_Markov_chain(T = 10, N = 10):
    Markov_chain = []
    Markov_chain.append(gen_last_matrix(N, T))
    for i in range(T - 1):
        Markov_chain.append(Markov_chain[i].backwards_gen())
    return Markov_chain


def gen_distribution(T = 10, N = 10, start = 5):
    markov_chain = gen_Markov_chain(T, N)
    markov_chain.reverse()
    pre_distribution = []
    pre_distribution.append([markov_chain[0].matrix[start]])
    for i in range(len(markov_chain) - 1):
        new_distribution = compound(pre_distribution[i],
                                    markov_chain[i + 1].matrix)
        pre_distribution.append(new_distribution)
    return pre_distribution


#test
#example = gen_Markov_chain()
#for i in range(len(example)):
    #print(example[i].matrix)

#print(gen_distribution())
```

4 Test

We take the value that $T = 10$, $N = 10$ and start from $N/2$. So we can solve all the $p_s(i,j)$. Set the initial distribution starting from $N/2$ is $\mathbf{q}(0)$ and transition matrix of step s is $\mathbf{T}_s(0 \le s \le 9)$. So we can get:

$$\mathbf{q}(t) = \mathbf{q}(0)\prod_{s=0}^{t-1}\mathbf{T}_s. \tag{7}$$

And obviously, we can easily get $\mathbf{T}_s$ from $p_s(i,j)$, which is

$$\mathbf{T}_s^{i,j} = p_s(i-1, j-1). \tag{8}$$

So, finally we get $q_j(s)(0 \leq j \leq 10, 0 \leq s \leq 9)$. Table 1 shows the output of $q_j(s)$.

Table 1: The value of $q_j(s)$

	j = 0	1	2	3	4	5	6	7	8	9	10
s = 0	.0000011	.00042	.011	.083	.24	.33	.24	.083	.011	.00042	.0000011
1	.00037	.011	.054	.13	.2	.23	.2	.13	.054	.011	.00037
2	.0038	.033	.083	.13	.16	.17	.16	.13	.083	.033	.0038
3	.014	.058	.1	.13	.14	.14	.14	.13	.1	.058	.014
4	.032	.08	.1	.11	.12	.12	.12	.12	.1	.08	.032
5	.06	.094	.1	.1	.096	.095	.096	.1	.1	.094	.06
6	.099	.1	.096	.086	.08	.078	.08	.086	.096	.1	.099
7	.16	.098	.083	.07	.063	.061	.063	.07	.083	.098	.16
8	.25	.076	.06	.05	.045	.044	.045	.05	.06	.076	.25
9	.5	0	0	0	0	0	0	0	0	0	.5