北京航空航天大學
BEIHANG UNIVERSITY

自然语言处理第二次作业

硬币投掷中的 EM 算法

姓　名：夏秀博

学　号：BY2107030

学　院：机械工程及自动化学院

2022 年 4 月 8 日

投硬币中的 EM 算法

1 问题描述

一个袋子中三种硬币的混合比例为：s1, s2 与 1-s1-s2 (0<=si<=1), 三种硬币掷出正面的概率分别为：p, q, r。 （1）自己指定系数 s1, s2, p, q, r，生成 N 个投掷硬币的结果（由 01 构成的序列，其中 1 为正面，0 为反面），利用 EM 算法来对参数进行估计并与预先假定的参数进行比较。

2 EM 算法

2.1 算法简介

EM 算法即期望最大化算法，非常适用于模型中拥有隐含变量的时候，通过 EM 迭代，使模型参数在期望符合得符合模型特征。

EM 算法的步骤为：

step1 ：初始化分布参数

step2 ：重复 E、M 步骤直到收敛：

a) **E 步骤-期望值计算**：根据参数的假设值，给出未知变量的期望估计，应用于缺失值。

b) **M 步骤-最大化计算**：根据未知变量的估计值，给出当前的参数的极大似然估计。

2.2 EM 算法迭代公式

EM 算法的迭代公式为：

E 步：

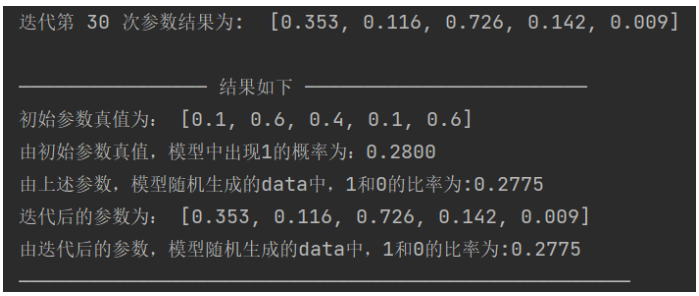
$$Q(\theta,\theta_i) = E[\log P(Y,Z|\theta)|Y,\theta_i] = \sum_Z \log P(Y,Z|\theta)P(Z|Y,\theta_i)$$

M 步：

$$\theta_{i+1} = \arg\max Q(\theta,\theta_i).$$

3 实验结果及说明

EM 算法迭代结果如下：



进行了多次测试，结果如下：

序号	参数	s1	s2	p	q	r	P(xi)
1	真值	0.2	0.4	0.3	0.2	0.4	0.3
	计算值	0.314	0.472	0.247	0.109	0.816	0.3037
2	真值	0.3	0.4	0.8	0.5	0.4	0.56
	计算值	0.444	0.072	0.938	0.02	0.294	0.5605
3	真值	0.1	0.6	0.4	0.1	0.6	0.28
	计算值	0.353	0.116	0.726	0.142	0.009	0.2775

进行了多次试验测试，但各参数始终无法收敛到所设定真值中，对该现象进行了初步的分析。

首先，该问题假设中，投硬币为二项分布，各次硬币投掷间独立，该模型中无论 data 集取多大，可以得到的结果仅仅为 0 与 1 的比例，模型中数据集可展现出的结果太少，导致难以还原模型中的隐含参数。

毕竟，只给你一个 0.3，让你去猜测 s1,s2,p,q,r 是多少…有点强人所难了。

从另一个方向看，假设数据集中 1 的比例为 0.3，那么此时可以有 s1=0.2,s2=0.4,p=0.3,q=0.5,r=0.1.或者，s1=0.5,s2=0.3,p=0.2,q=0.4,r=0.2，等等。拥有无数组可行的参数使模型成立，因此模型在迭代过程中不存在唯一的最大期望值，导致 EM 算法无法收敛。

4 附录

```
import numpy as np

def data_generate(theta0):
    s1, s2, p, q, r = theta0
    n_max = 10000
```

```
    data = []
    for i in range(n_max):
        t1 = np.random.rand()
        t2 = np.random.rand()
        if t1 < s1:   # 仍第一个硬币
            data.append(1) if t2 < p else data.append(0)   # 仍出正面加个 1,背面加个 0
        elif t1 < s1 + s2:   # 仍第二个硬币
            data.append(1) if t2 < q else data.append(0)
        else:   # 仍第三个硬币
            data.append(1) if t2 < r else data.append(0)

    return data


# 通过 xi 的分布计算出 miu1，miu2
def pd_xi(xi, theta):
    s1, s2, p, q, r = theta
    pd = p**xi*(1-p)**(1-xi)*s1 + q**xi*(1-q)**(1-xi)*s2 + r**xi*(1-r)**(1-xi)*(1-s1-s2)
    miu1 = p**xi*(1-p)**(1-xi)*s1/pd
    miu2 = q**xi*(1-q)**(1-xi)*s2/pd
    return miu1, miu2


# EM 算法的 E 步,通过参数 theta 的值计算出各 xi 的 miu1,miu2
def e_bu(data, theta):
    u1 = []
    u2 = []
    for d in data:
        miu1, miu2 = pd_xi(d, theta)
        u1.append(miu1)
        u2.append(miu2)
    return u1, u2


# EM 算法的 M 步，通过 E 步计算出的 u1,u2 来推测最大概率对应的 theta 值.
def m_bu(data, u1, u2):
    temp1 = 0
    temp2 = 0
    temp3 = 0
    s1 = sum(u1)/len(u1)
    s2 = sum(u2)/len(u2)
    for d, miu1, miu2 in zip(data, u1, u2):
        temp1 += miu1*d
        temp2 += miu2*d
```

```
            temp3 += (1-miu1-miu2)*d
        p = temp1/sum(u1)
        q = temp2/sum(u2)
        r = temp3/(len(data)-sum(u1)-sum(u2))
        theta = [s1, s2, p, q, r]
        return theta


# 运行，最大迭代次数为 n_max，超过该次数系统自动停止
def run(data, theta, n_max):
        for i in range(n_max):
            u1, u2 = e_bu(data, theta)
            theta = m_bu(data, u1, u2)
            print('迭代第', i+1, '次参数结果为: ', [float(format(j, '.3f')) for j in theta])
        return theta

if __name__ == '__main__':
        # 所设定的参数 theta =   [s1 s2 p q r]
        theta0 = [0.2, 0.4, 0.3, 0.2, 0.4]
        # 初始化参数，应该是随机的，这里为了测试方便手动输入了，也可以随机,下面四行就是随机生成初始化参数的
        theta_int = [0.3, 0.4, 0.4, 0.2, 0.9]
        # theta_int = np.random.rand(5)
        # theta_rand3 = np.random.rand(3)
        # theta_rand3 = theta_rand3/sum(theta_rand3)
        # theta_int[:2] = theta_rand3[:2]

        # 生成初始化数据：01001001110101111……
        data = data_generate(theta0)
        # 进行 EM 迭代，并返回最终的参数值 th = [s1,s2,p,q,r]
        th = run(data, theta_int, 30)
        print('')
        print('————————'*4, '结果如下', '————————'*6)
        print('初始参数真值为：', theta0)
        print('由初始参数真值，模型中出现 1 的概率为：{:.4f}'.format(theta0[0] * theta0[2] + theta0[1] * theta0[3] +
                                                                    (1 - theta0[0] - theta0[1]) * theta0[4]))
        print('由上述参数，模型随机生成的 data 中，1 和 0 的比率为:{:.4f}'.format(sum(data) / len(data)))

        print('迭代后的参数为：', [float(format(j, '.3f')) for j in th])
        print('由迭代后的参数，模型随机生成的 data 中，1 和 0 的比率为:{:.4f}'.format(th[0]*th[2]+th[1]*th[3]+(1-th[0]-th[1])*th[4]))
        print('————————'*13)
```