学　　期　2022-2023

北京航空航天大學

BEIHANG UNIVERSITY

深度学习与自然语言处理

基于EM算法的参数估计

院（系）名称	自动化科学与电气工程学院
专业名称	电子信息
学生姓名	苏士鹏
学号	ZY2103306

2022年4月

目录

一，作业要求

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。

二，实验原理及方法

2.1 公式推导

参考《统计学方法》，对该模型的公式进行推导，EM 算法分成 E 步和 M 步，是一种产业用的迭代算法，由于观测数据的极大似然估计求解存在缺失数据的问题，引入 EM 算法可以有效减少这一问题。

E 步，即求隐变量，如式 2.1 所示，给定观测数据和当前的估计参数，求出隐变量的条件概率分布，M 步即将隐变量作为已知量，如式 2.2 所示，求极大化的参数，E 步和 M 步重复执行，直至收敛，

$$Q(\theta,\theta_i)=E[\log P(Y,Z|\theta)|Y,\theta_i]=\sum_Z \log P(Y,Z|\theta)P(Z|Y,\theta_i) \qquad (2.1)$$

$$\theta_{i+1}=\mathrm{argmax}\,Q(\theta,\theta_i) \qquad (2.2)$$

2.2 三硬币模型

为了得到三硬币模型参数，需要先分别计算各个硬币的概率，首先给定假定参数 m=[s1,s2,s3,p,q,r]，硬币为 1，2，3，结果储存在 x 当中，首先需计算各个硬币产生 x 的概率，已知条件为硬币种类。

$$\mathrm{P}(x_i|1,m)=\mathrm{p}^{\mathrm{x_i}}(1-\mathrm{p})^{(1-\mathrm{x_i})}$$
$$\mathrm{P}(x_i|2,m)=\mathrm{q}^{\mathrm{x_i}}(1-\mathrm{q})^{(1-\mathrm{x_i})}$$
$$\mathrm{P}(x_i|3,m)=\mathrm{r}^{\mathrm{x_i}}(1-\mathrm{r})^{(1-\mathrm{x_i})}$$

在此基础上计算来自各个硬币的概率：

$$\mathrm{P}(x_i,1|m)=\mathrm{s_1}\mathrm{p}^{\mathrm{x_i}}(1-\mathrm{p})^{(1-\mathrm{x_i})}$$
$$\mathrm{P}(x_i,2|m)=\mathrm{s_2}\mathrm{q}^{\mathrm{x_i}}(1-\mathrm{q})^{(1-\mathrm{x_i})}$$
$$\mathrm{P}(x_i,3|m)=\mathrm{s_3}\mathrm{r}^{\mathrm{x_i}}(1-\mathrm{r})^{(1-\mathrm{x_i})}$$

最后得到 Q，即已知投掷结果，来自不同硬币的概率：

$$\mu_1(x)=Q_1(1\mid x_i,\mathrm{m})=\frac{\mathrm{P}(x_i,1\mid m)}{\mathrm{P}(x_i,1\mid m)+\mathrm{P}(x_i,2\mid m)+\mathrm{P}(x_i,3\mid m)}$$

$$\mu_2(x)=Q_1(2\mid x_i,\mathrm{m})=\frac{\mathrm{P}(x_i,2\mid m)}{\mathrm{P}(x_i,1\mid m)+\mathrm{P}(x_i,2\mid m)+\mathrm{P}(x_i,3\mid m)}$$

$$\mu_3(x)=Q_1(3\mid x_i,\mathrm{m})=\frac{\mathrm{P}(x_i,3\mid m)}{\mathrm{P}(x_i,1\mid m)+\mathrm{P}(x_i,2\mid m)+\mathrm{P}(x_i,3\mid m)}$$

对结果做参数估计如下：

$$\hat{S}_1=\frac{\sum_{i=1}^{N}\mu_1\left(x^{(i)}\right)}{N}\qquad \hat{p}=\frac{\sum_{i=1}^{N}\mu_1\left(\mathrm{x}^{(i)}\right)\mathrm{x}^{(i)}}{\sum_{i=1}^{N}\mu_1\left(\mathrm{x}^{(i)}\right)}$$

$$\hat{S}_2=\frac{\sum_{i=1}^{N}\mu_2\left(x^{(i)}\right)}{N}\qquad \hat{q}=\frac{\sum_{i=1}^{N}\mu_2\left(\mathrm{x}^{(i)}\right)\mathrm{x}^{(i)}}{\sum_{i=1}^{N}\mu_2\left(\mathrm{x}^{(i)}\right)}$$

$$\hat{S}_3=\frac{\sum_{i=1}^{N}\mu_3\left(x^{(i)}\right)}{N}\qquad \hat{r}=\frac{\sum_{i=1}^{N}\mu_3\left(\mathrm{x}^{(i)}\right)\mathrm{x}^{(i)}}{\sum_{i=1}^{N}\mu_3\left(\mathrm{x}^{(i)}\right)}$$

根据上述公式不断迭代，直至结果不变。

三，实验结果

数组长度(N)	实际值(s1,s2,p,q,r)	初始设定值 (s1,s2,p,q,r)	迭代次数	最终迭代结果
1000	[0.5,0.3,0.3,0.5,0.5]	[0.53,0.27,0.2,0.4,0.4]	2	[0.5052, 0.2842, 0.2832, 0.5131, 0.5131
10000	[0.5,0.3,0.3,0.5,0.5]	[0.53,0.27,0.2,0.4,0.4]	2	[0.5059, 0.2837, 0.2807, 0.5100, 0.5100]
1000	[0.5,0.3,0.3,0.5,0.5]	[0.3,0.3,0.5,0.5,0.2]	2	[0.3065, 0.3065, 0.5446, 0.5446, 0.2301]
1000	[0.5,0.3,0.3,0.5,0.5]	[0.3,0.3,0.2,0.4,0.4]	2	[0.2945, 0.3023, 0.2210, 0.4307, 0.4307]
1000	[0.5,0.3,0.3,0.5,0.5]	[0.6,0.2,0.2,0.4,0.4]	2	[0.5704, 0.2147, 0.3035, 0.5374, 0.5374]

从上表可以看出，EM 迭代算法对该模型的求解能力较差，迭代两次就陷入局部最优，

同时结果与初始值的选择关系十分紧密，这里我选用的收敛值为，10^{-6}设置的较为严格，但模型还是在 2 次迭代到达收敛，这种极快的到达收敛的特性可能是 EM 算法本身的特性，同时变化参数可以看到结果有明显的变化，可见 EM 算法只保证了收敛性，并没有保证全局最优。

附件

```
import math
import random

##数据生成函数，按照要求投掷 N 次生成结果，方便后面调用
def get_data(s,q,N):

    data = []
    for i in range(N):       ##先选择是哪个球
        b_which = random.randint(1,100)
        if 1 <= b_which <= 100 * s[0]:
            ball = 1
        elif b_which <= 100 * (s[0] + s[1]):
            ball = 2
        else:
            ball = 3

        b_direction = random.randint(1,100)
        if ball == 1:        ##计算选定球正面或者反面
            if 1 <= b_direction <= 100 * q[0]:
                data.append(1)
            else:
                data.append(0)
        elif ball == 2:
            if 1 <= b_direction <= 100 * q[1]:
                data.append(1)
            else:
                data.append(0)
        else:
            if 1 <= b_direction <= 100 * q[2]:
                data.append(1)
            else:
                data.append(0)
    return data

def piay_e(s,q,x):      ##做 e 步计算，根据公式进行计算即可

    u1_x, u2_x, u3_x = [], [], []

    for i in range(len(x)):
        if x[i] == 1:
            s0 = s[0]*q[0]
            s1 = s[1]*q[1]
```

```
            s2 = s[2]*q[2]
        else:
            s0 = s[0]*(1-q[0])
            s1 = s[1]*(1-q[1])
            s2 = s[2]*(1-q[2])
        s_all = s0 + s1 + s2
        u1_x.append(s0 / s_all)
        u2_x.append(s1 / s_all)
        u3_x.append(s2 / s_all)

    return [u1_x, u2_x, u3_x]

def play_m(u,x):        ##做 m 步计算，根据公式进行计算

    s = []
    q = []
    for j in range(3):

        s.append(sum(u[j])/len(u[j]))
        q.append(sum([u[j][i] * x[i] for i in range(len(x))])/sum(u[j]))

    return s, q

def play_em(s, q, x, N):        ##将上面两个函数结合，做迭代计算

    s_pre = s
    q_pre = q
    for i in range(N):
        u = piay_e(s_pre,q_pre,x)
        s_exp,q_exp = play_m(u,x)

        #迭代结束条件
        if sum([abs(q_pre[j] - q_exp[j]) for j in range(3)]) +
sum([abs(s_pre[k] - s_exp[k]) for k in range(3)]) < 0.000001:
            break

        s_pre = s_exp
        q_pre = q_exp

    return s_exp,q_exp,i+1



if __name__ == "__main__":
```

```
s = [0.5,0.3,0.2]        #设定正确值，即为真值，
q = [0.3,0.5,0.5]
N = 1000                 #投掷 1000 次硬币
x = get_data(s,q,N)      #生成结果

s_start = [0.6,0.2,0.2] #迭代开始数据
q_start = [0.2,0.4,0.4] #迭代开始数据

s_pre,q_pre,i = play_em(s_start, q_start, x, N) #利用 em 算法做迭代

print("迭代次数",i)
print("迭代得到的各个硬币占比：",s_pre[0], s_pre[1], s_pre[2])
print("初始的的各个硬币占比：",s[0], s[1], s[2])
print("迭代得到的各个硬币正面比例：",q_pre[0], q_pre[1], q_pre[2])
print("初始设定得到的各个硬币正面比例：",q[0], q[1], q[2])
```