深度学习与自然语言处理第二次作业

SY2106318 孙旭东

代码链接

1.作业内容

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.相关知识

2.1 EM算法

EM算法，即最大期望算法（Expectation-Maximization algorithm, EM），是一类通过迭代进行极大似然估计（Maximum Likelihood Estimation, MLE）的优化算法，通常作为牛顿迭代法的替代用于对包含隐变量（latent variable）或缺失数据（incomplete-data）的概率模型进行参数估计。 EM算法的标准计算框架由E步（Expectation-step）和M步（Maximization step）交替组成，E步主要通过观察数据和现有模型来估计参数，然后用这个估计的参数值来计算似然函数的期望值；而 M步是寻找似然函数最大化时对应的参数。由于算法会保证在每次迭代之后似然函数都会增加，所以函数最终会收敛，算法的收敛性可以确保迭代至少逼近局部极大值 。

由于迭代规则容易实现并可以灵活考虑隐变量， EM算法被广泛应用于处理数据的缺测值， 以及很多机器学习算法，包括高斯混合模型和隐马尔可夫模型的参数估计。

2.2 本题中EM算法推导

课上进行过三硬币模型的推导，本题是对该例的拓展。

隐变量s_1, s_2表示来自各个硬币的概率，$z = \{s_1, s_2, s_3(= 1 - s_1 - s_2)\}$分别表示来自硬币A，B，C的概率，硬币A，B，C正面出现的概率为p, q, r，观察数据为$x = x_1x_2x_3 \ldots x_n$，生成一次观察数据时的概率：

$$P(x_i|\theta) = s_1p^{x_i}(1-p)^{1-x_i} + s_2q^{x_i}(1-q)^{1-x_i} + s_3r^{x_i}(1-r)^{1-x_i}$$

则观察x对应的对数似然函数为：

$$L(\theta|x) = logP(x_1x_2 \ldots x_n|\theta) = log\prod_{i=1}^{n} P(x_i|\theta) = \sum_{i=1}^{n} logP(x_i|\theta)$$

目标即为求θ的最大似然估计$\hat{\theta}$。

更新隐变量时，计算其后验概率，即在已知分布的情况下分别计算硬币来自A，B，C的概率。

- 硬币来自A

$$u_{1i} = P(s_1|x_i,\theta) = \frac{P(x_i,s_1|\theta)}{P(x_i|\theta)} = \frac{s_1p^{x_i}(1-p)^{1-x_i}}{s_1p^{x_i}(1-p)^{1-x_i}+s_2q^{x_i}(1-q)^{1-x_i}+(1-s_1-s_2)r^{x_i}(1-r)^{1-x_i}}$$

- 硬币来自B

$$u_{2i} = P(s_2|x_i,\theta) = \frac{P(x_i,s_2|\theta)}{P(x_i|\theta)} = \frac{s_2q^{x_i}(1-q)^{1-x_i}}{s_1p^{x_i}(1-p)^{1-x_i}+s_2q^{x_i}(1-q)^{1-x_i}+(1-s_1-s_2)r^{x_i}(1-r)^{1-x_i}}$$

- 硬币来自C

$u_{3i}=1-u_{1i}-u_{2i}$

将当前假设的参数代入，可计算得到u_{1i},u_{2i},u_{3i}的具体数值，根据Jensen不等式，可求得对数最大似然函数的一个下界：

$$L(\theta|x)=\sum_{i=1}^{n}logP(x_i|\theta)=\sum_{i=1}^{n}log\sum_{i=1}^{3}[P(s_i|z)P(x_i|s_i,\theta)]\geq\sum_{i=1}^{n}\sum_{i=1}^{3}P(s_i|z)log[P(x_i|s_i,\theta)]$$

因此根据E步求得的值，我们找到下界

$$J=\sum_{i=1}^{n}[u_{1i}log\frac{s_1p^{x_i}(1-p)^{1-x_i}}{u_{1i}}+u_{2i}log\frac{s_2q^{x_i}(1-q)^{1-x_i}}{u_{2i}}+u_{3i}log\frac{s_3r^{x_i}(1-r)^{1-x_i}}{u_{3i}}]$$

分别对各个参数求偏导

$$\frac{\partial J}{\partial s_1}=\sum_{i=1}^{n}(\frac{u_{1i}}{s_1}-\frac{1-u_{1i}-u_{2i}}{1-s_1-s_2})=0\Rightarrow s_1=\frac{1}{n}\sum_{i=1}^{n}u_{1i}$$

$$\frac{\partial J}{\partial s_2}=\sum_{i=1}^{n}(\frac{u_{2i}}{s_2}-\frac{1-u_{1i}-u_{2i}}{1-s_1-s_2})=0\Rightarrow s_2=\frac{1}{n}\sum_{i=1}^{n}u_{2i}$$

$$\frac{\partial J}{\partial p}=\sum_{i=1}^{n}u_{1i}(\frac{x_i}{p}-\frac{1-x_i}{1-p})=0\Rightarrow p=\frac{\sum_{i=1}^{n}u_{1i}x_i}{\sum_{i=1}^{n}u_{1i}}$$

$$\frac{\partial J}{\partial q}=\sum_{i=1}^{n}u_{2i}(\frac{x_i}{q}-\frac{1-x_i}{1-q})=0\Rightarrow q=\frac{\sum_{i=1}^{n}u_{2i}x_i}{\sum_{i=1}^{n}u_{2i}}$$

$$\frac{\partial J}{\partial r}=\sum_{i=1}^{n}u_{3i}(\frac{x_i}{r}-\frac{1-x_i}{1-r})=0\Rightarrow r=\frac{\sum_{i=1}^{n}u_{3i}x_i}{\sum_{i=1}^{n}u_{3i}}$$

用M步求得的新参数值代替原来的参数值，继续迭代。

3.实验过程

3.1数据生成

使用到的数据为依照概率生成的硬币投掷结果序列，首先对使用的参数进行设置：

```
real_param = {'s1': 0.2, 's2': 0.3, 'p': 0.7, 'q': 0.9, 'r': 0.4}
fake_param = {'s1': 0.1, 's2': 0.6, 'p': 0.5, 'q': 0.5, 'r': 0.3}
data_scale = 10000
per_size = 5
epoch = 20
```

real_param为数据生产时依赖的真实参数，其中，s1, s2为第一种和第二种硬币在袋子中所占的比例，p、q、r分别代表三种硬币掷出正面的概率；fake_param为假定的各个参数的初始值；data_scale为要生成的投掷硬币的组数；在实验中我们设置每组投掷的是同一硬币，per_scale为每组投掷结果中硬币投掷的次数；epoch为迭代进行的次数。

生成代码如下：

```python
def make_data():
    """
    根据真实参数(real_param)和数据规模(data_scale)生成数据
    :return: 生成的01序列
    """
    data = []
    for i in range(data_scale):
        which_icon = random.random()
        if which_icon < real_param['s1']:
            data.append([])
            for j in range(per_size):
                if random.random() < real_param['p']:
                    data[i].append(1)
                else:
                    data[i].append(0)
        elif which_icon < real_param['s1'] + real_param['s2']:
            data.append([])
            for j in range(per_size):
                if random.random() < real_param['q']:
                    data[i].append(1)
                else:
                    data[i].append(0)
        else:
            data.append([])
            for j in range(per_size):
                if random.random() < real_param['r']:
                    data[i].append(1)
                else:
                    data[i].append(0)
    return data
```

这一函数用于根据real_param、data_scale和per_size生成数据，即题目中要求的01序列，由于按照组进行生成，最终data的形状为data_scale*per_size。

3.2 EM算法

```python
def em_single(x):
    """
    进行一次EM操作
    :param x: 数据
    :return: 更新后的fake_param
    """
    u1_list = []
    u2_list = []
    s1 = fake_param['s1']
    s2 = fake_param['s2']
    p = fake_param['p']
    q = fake_param['q']
    r = fake_param['r']
    for xi in x:
        num_1 = sum(xi)
        num_0 = len(xi) - num_1
```

```
        u1_i = (s1*math.pow(p, num_1)*math.pow(1-p, num_0))/((s1*math.pow(p,
num_1)*math.pow(1-p, num_0)) +
                                                        (s2 * math.pow(q, num_1) *
math.pow(1 - q, num_0)) +
                                                        ((1-s1-s2) * math.pow(r, num_1) *
math.pow(1 - r, num_0)))
        u2_i = (s2 * math.pow(q, num_1) * math.pow(1 - q, num_0)) / ((s1 * math.pow(p,
num_1) * math.pow(1 - p, num_0)) +
                                                                  (s2 * math.pow(q, num_1)
* math.pow(1 - q, num_0)) +
                                                                  ((1 - s1 - s2) *
math.pow(r, num_1) * math.pow(1 - r,num_0)))
        u1_list.append(u1_i)
        u2_list.append(u2_i)

    new_s1 = sum(u1_list)/len(x)
    new_s2 = sum(u2_list)/len(x)
    new_p = sum(u1_list[i]*sum(x[i])/len(x[i]) for i in range(len(x)))/sum(u1_list)
    new_q = sum(u2_list[i]*sum(x[i])/len(x[i]) for i in range(len(x)))/sum(u2_list)
    new_r = sum((1-u1_list[i]-u2_list[i])*sum(x[i])/len(x[i]) for i in
range(len(x)))/sum((1-u1_list[i]-u2_list[i]) for i in range(len(x)))
    return new_s1, new_s2, new_p, new_q, new_r
```

这一函数为代码的核心部分，其功能为根据当前参数进行后验概率的计算，然后根据推导出的表达式，使用后验概率的值计算新的参数值。

3.3结果展示

为了方便的进行结果展示，除了在控制台打印结果外，还编写了图像绘制代码，代码如下：

```
def draw(z, theta):
    """
    绘制图像
    :param z:参数z历次迭代后取值
    :param theta: 参数theta历次迭代后取值
    :return:
    """
    plt.rcParams["figure.figsize"] = (12, 6)
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.subplot(121)
    plt.plot(z[:, 0], label='s1')
    plt.plot(z[:, 1], label='s2')
    plt.plot(z[:, 2], label='s3')
    real_z = [real_param['s1'], real_param['s2'], 1 - real_param['s1'] - real_param['s2']]
    for t in real_z:
        plt.hlines(t, 0, len(z), linestyles='dashed')
    plt.legend()
    plt.title('隐参数随迭代次数变化')

    plt.subplot(122)
    plt.plot(theta[:, 0], label='p')
    plt.plot(theta[:, 1], label='q')
    plt.plot(theta[:, 2], label='r')
```

```
    real_theta = [real_param['p'], real_param['q'], real_param['r']]
    for t in real_theta:
        plt.hlines(t, 0, len(theta), linestyles='dashed')
    plt.legend()
    plt.title('各硬币掷正面概率随迭代次数变化')

    plt.suptitle(r'$z_{init}$=' + f'{z[0]}'+r'  $\theta_{init}$=' + f'{theta[0]}')

    # plt.show()
    plt.savefig(f'{data_scale}-{per_size}+z-{z[0]}+theta-{theta[0]}.png', dpi=300)
```

最终将图像保存为png格式，命名为投掷组数+每组投掷次数+初始设定的参数.png，真实参数参照图片中虚线部分。

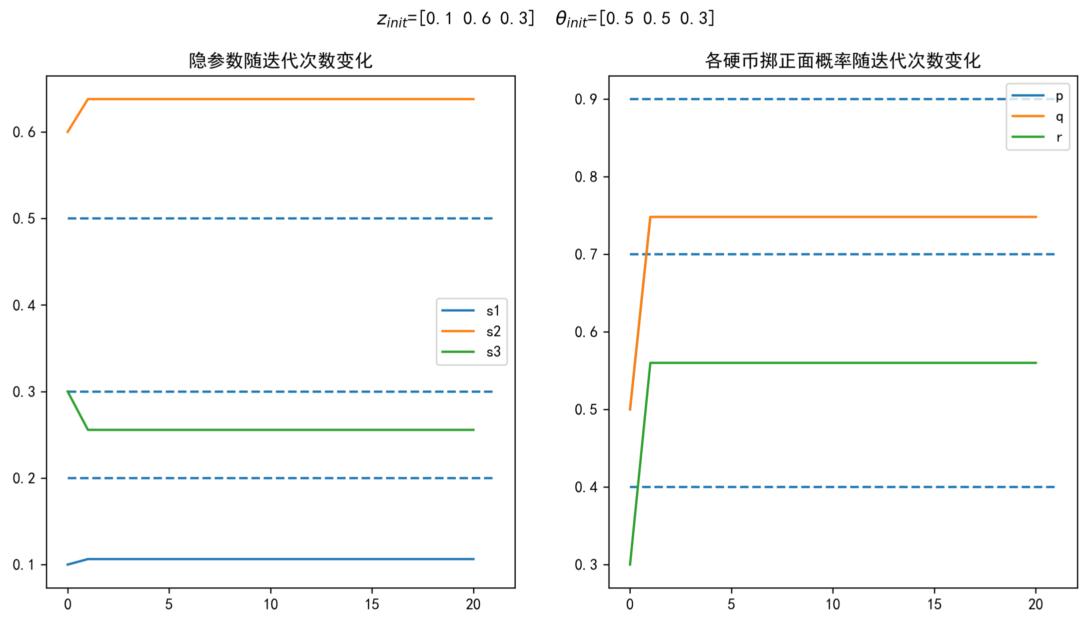
4.实验结果

将真实参数设置为$real_param = \{'s1':0.2,'s2':0.3,'p':0.7,'q':0.9,'r':0.4\}$，初始的假设参数设置为$fake_param = \{'s1':0.1,'s2':0.6,'p':0.5,'q':0.5,'r':0.3\}$，调节data_scale和per_size等参数对比效果。

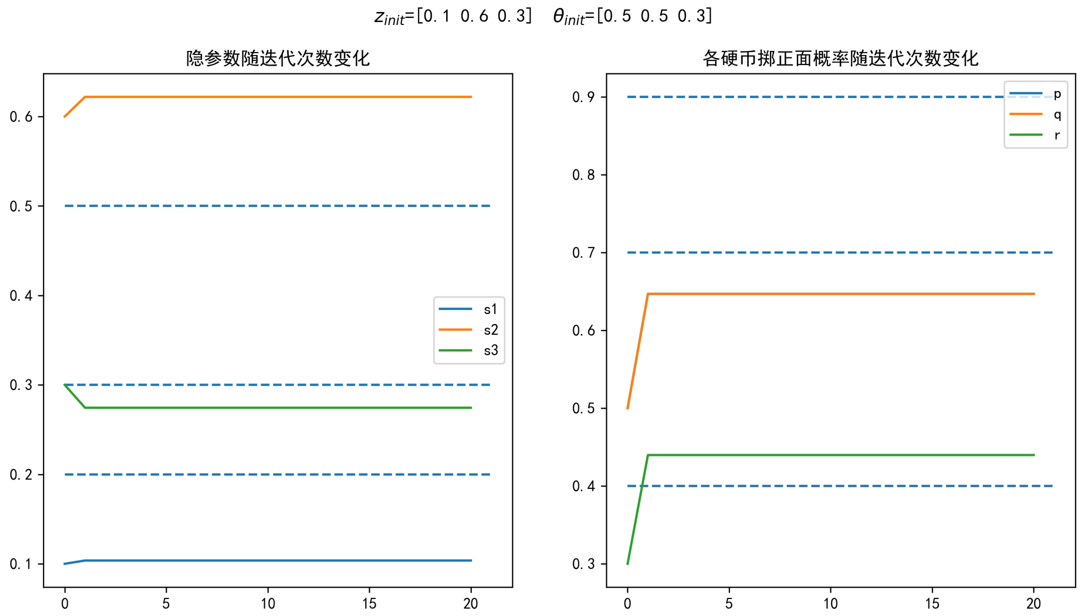
4.1固定per_size为1，调节data_scale

固定per_size为1，即每次仅投掷一枚硬币一次，观察不同data_scale的影响

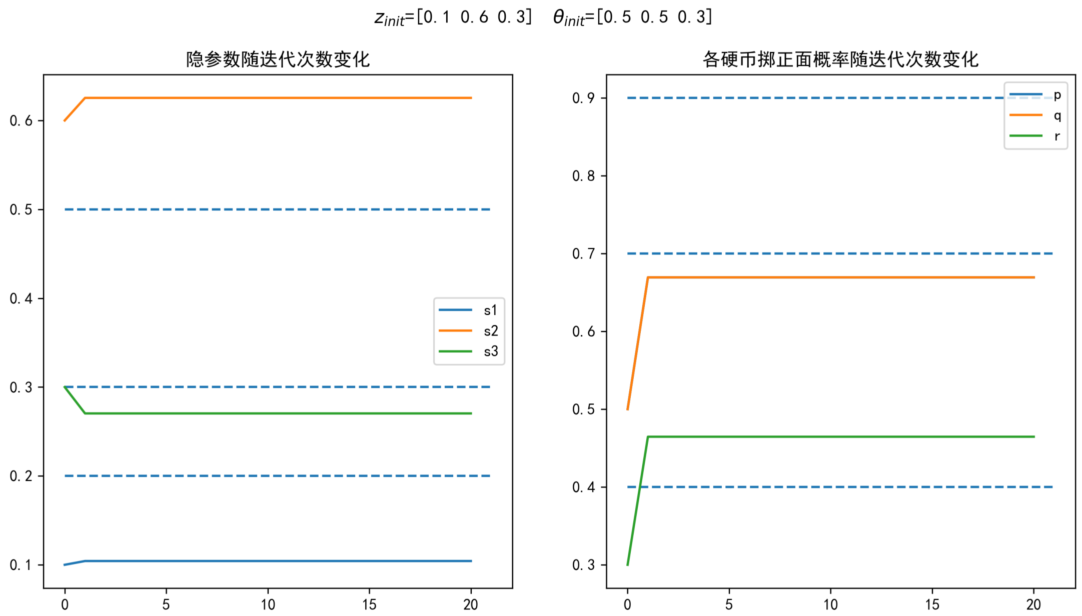
- data_scale = 10



- data_scale = 100



- data_scale = 1000

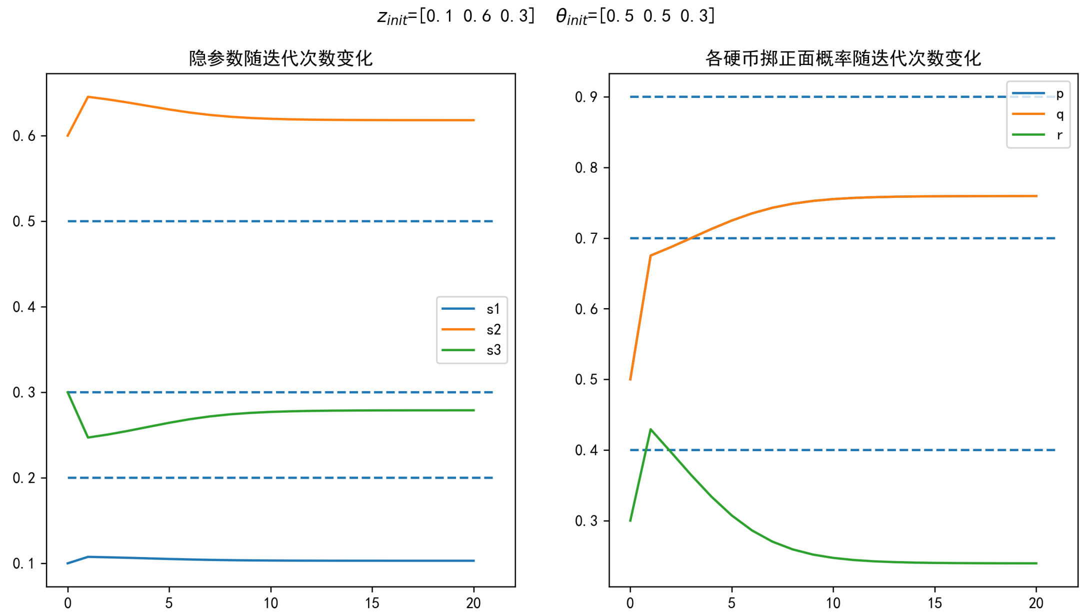


可以看出，在这种设定下，em算法的参数估计能力较差，几乎在第一次参数更新后参数值就会固定下来，分析其原因，在每组每枚硬币仅投掷一次的情况下，算法会很快收敛到局部最优解，虽然与真实参数值相差较大，局部最优解已经能较好地拟合这种情况下的数据。

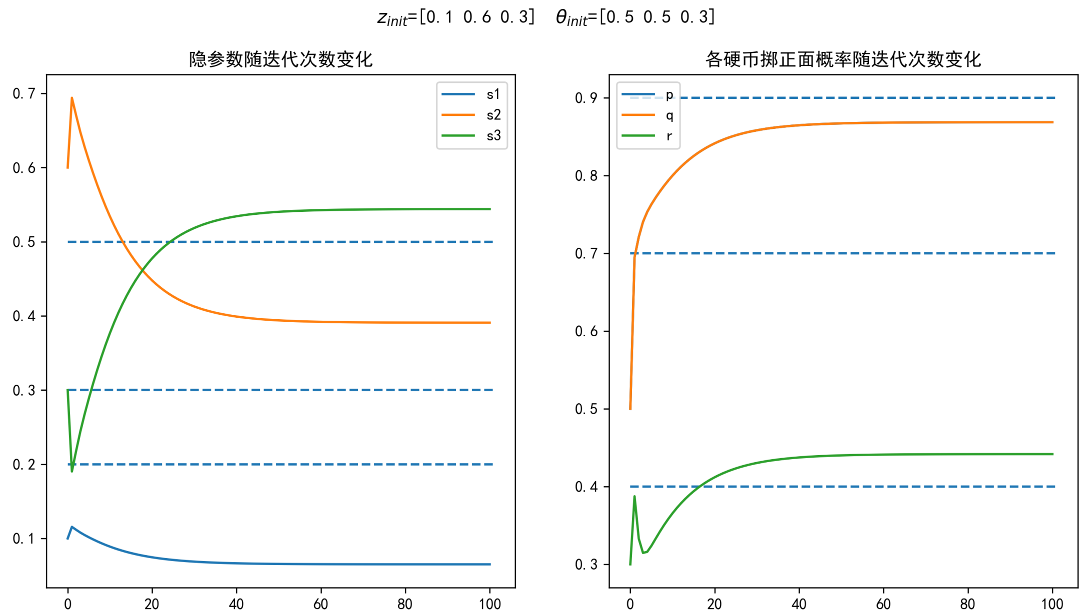
4.2固定data_scale为1000，调节per_size

固定data_size为1000，即每次进行1000组实验，观察不同per_size的影响

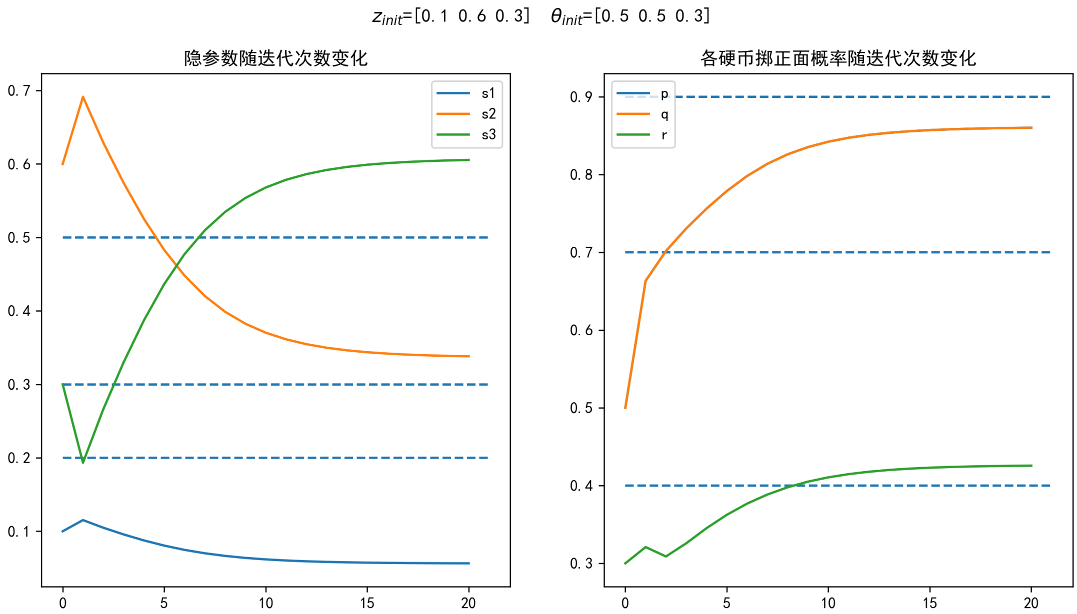
- per_size = 2



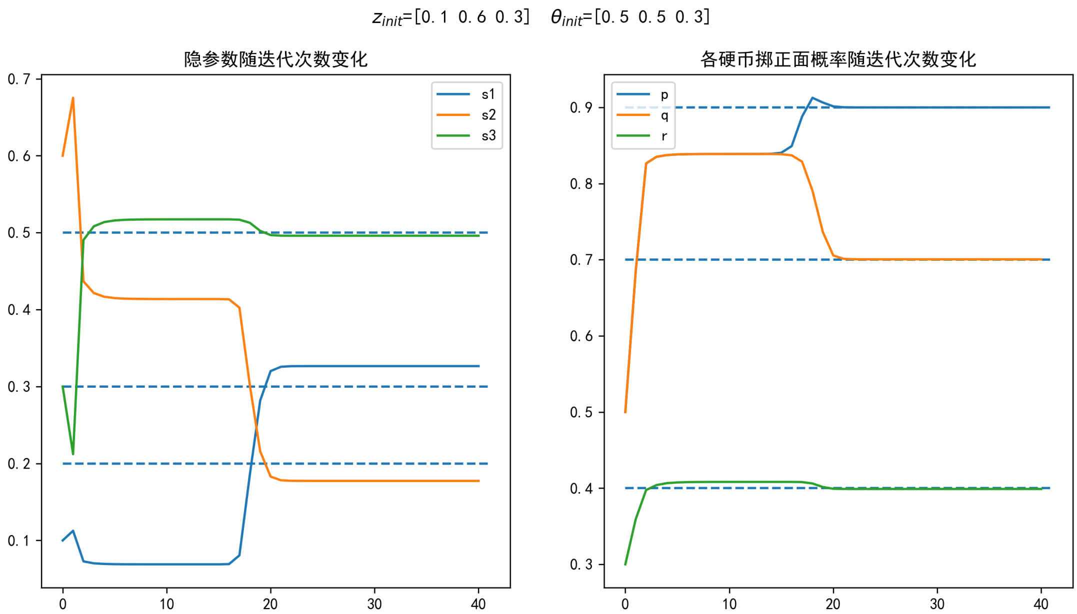
- per_size = 5



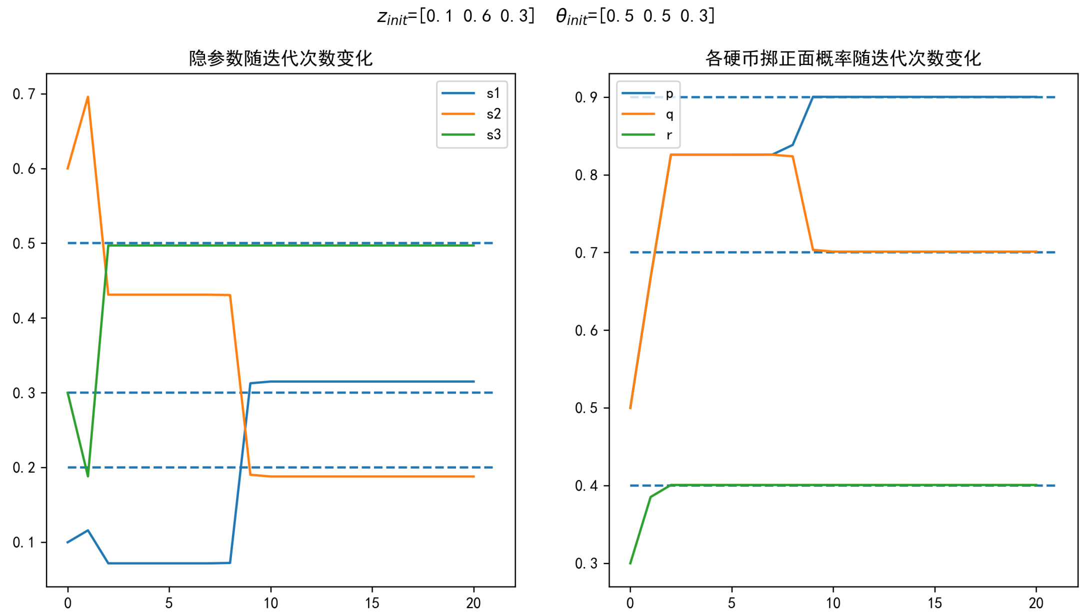
- per_size = 10



- per_size = 100



- per_size = 1000

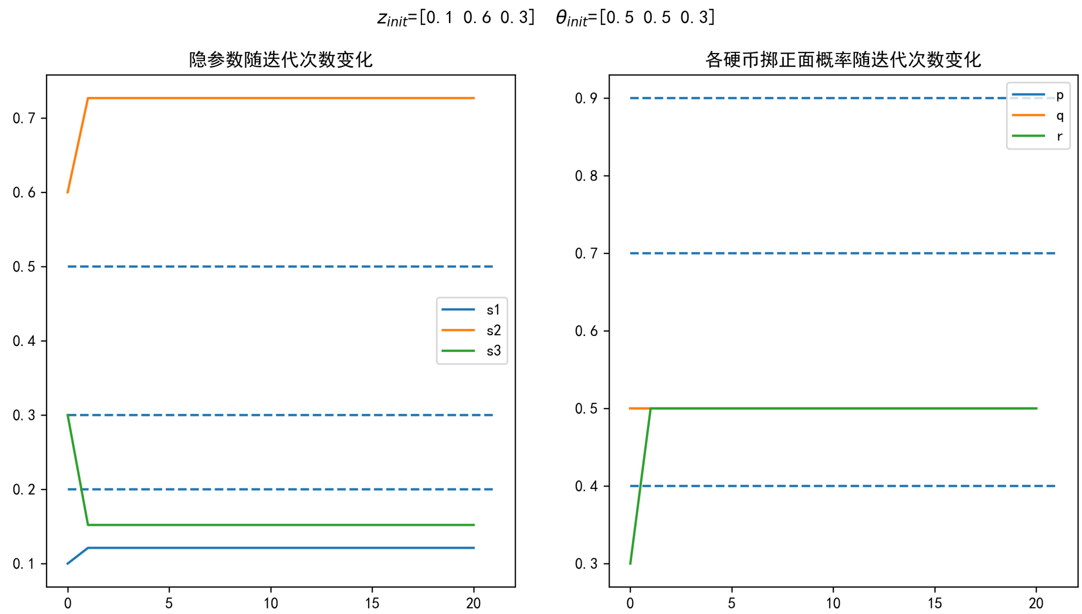


可以看出，per_size这一参数对em算法估计参数效果的影响非常大，当per_size调整为2、5、10时，与上一组实验的最后一个per_size为1相比拟合效果取得了一定的进步，且不会特别快陷入局部最优值，但最终效果离真实值仍有一定距离，当per_size调整为100和1000时，可以看到，最终的拟合效果已经非常逼近真实值了。

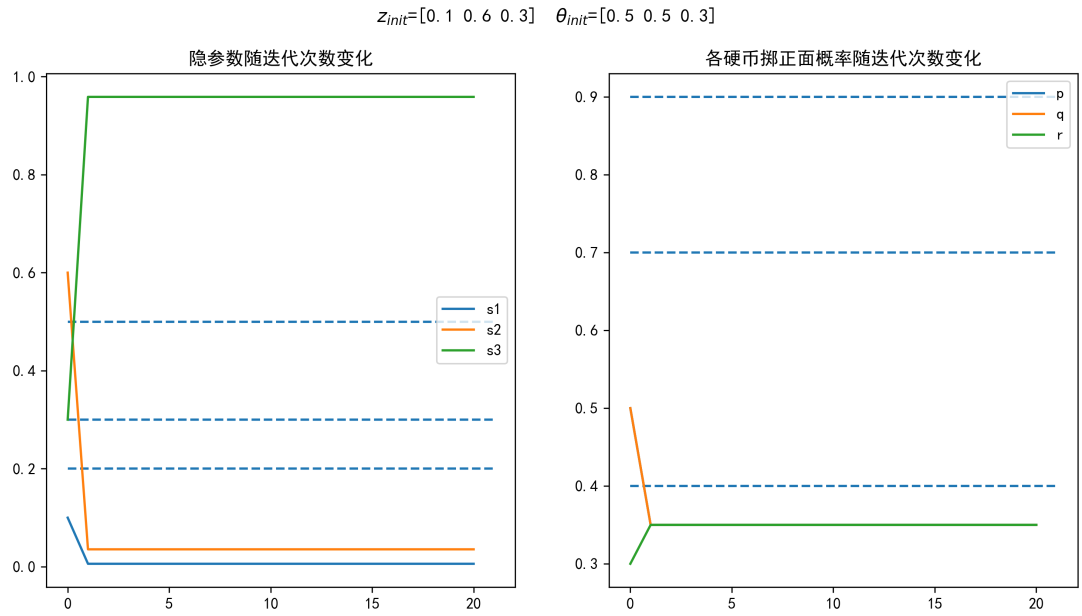
4.3固定data_scale为1，调节per_size

固定data_size为1，即每次只进行1组实验，从三个硬币中选出一个，观察不同per_size的影响

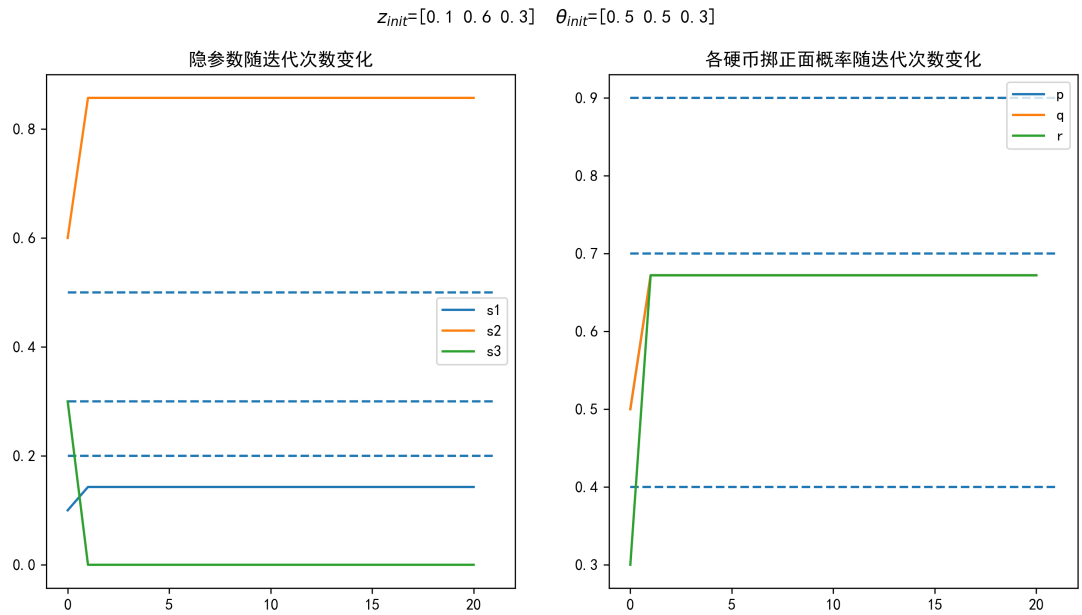
- per_size = 10



- per_size = 100



- per_size = 1000



通过这组实验会发现一个非常有趣的现象，经过一次迭代三个硬币概率的估计值就会趋于相同且不再变化，随着per_size的增大，估计效果会变好。这也容易理解，因为无论选择哪枚硬币，其概率分布趋近数据的真实分布比例时才会使得似然概率最大。从最后一张图中可以看出，该次实验选中的硬币为硬币B。

结论

1. 在固定每组投掷次数为1的情况下，实验组数的增加不会对em算法最终收敛的结果有明显改善；
2. 在固定实验组数为较大值1000的情况下，每组投掷次数的增加会显著改善em算法的效果，当每组投掷增大到1000时，估计值已经相当接近真实值；
3. 在固定实验组数为1的情况下，每组投掷次数的增加会改善em算法的效果，且收敛后结果各硬币为正面概率的估计值相同；

4. 最终收敛的结果可能出现顺序错乱的情况，但硬币占比和硬币为正面的概率会同时错位，因此估计结果的值是准确的。

参考文档

[EM算法 三硬币模型]

[EM算法-硬币实验的理解]