深度学习与自然语言处理作业2

院（系）名称 自动化科学与电气工程学院

学 生 姓 名 潘翔

学 生 学 号 ZY2103707

2022 年 4 月

一、作业内容及问题描述

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。截至日期：4 月 22 日晚 12 点前

二、原理

2.1 EM 算法解决问题方法简介

假设初始变量如一中问题描述所示，并且参数分别为s_1, s_2, π, p, q，依次为选择第一个硬币的概率，选择第二个硬币的概率，第一个硬币，第二个硬币，第三个硬币抛掷时出现正面的概率。该描述问题的参数和代码中的参数保持一致。

对于单词抛掷硬币的结果，可以将该硬币抛掷问题的模型写作：

$$\begin{aligned} p(y_i \mid \theta) &= \sum_z p(y_i, z \mid \theta) \\ &= \sum_z p(z \mid \theta) p(y_j \mid z, \theta) \\ &= s_1 \pi^{y_j} (1-\pi)^{1-y_j} + s_2 p^{y_j} (1-p)^{1-y_j} + (1-s_1-s_2) q^{y_j} (1-q)^{1-y_j} \end{aligned}$$

其中y_j是第j个观测结果 1 或 0；随机变量 z 是隐变量，表示未观测到的掷硬币 A 的结果；$\theta = (s_1, s_2, \pi, p, q)$是模型参数。

为了方便起见，观测数据可以表示为$y = (y_1, y_2, \dots, y_n)^T$，隐变量可以表示为$z = (z_1, z_2, \dots, z_n)^T$。观测数据的似然函数可以表示为

$$p(y \mid \theta) = \sum_z p(z \mid \theta) p(y \mid z, \theta)$$

即

$$p(y \mid \theta) = \prod_{j=1}^{n} \left[s_1 \pi^{y_j} (1-\pi)^{1-y_j} + s_2 p^{y_j} (1-p)^{1-y_j} + (1-s_1-s_2) q^{y_j} (1-q)^{1-y_j} \right]$$

而这个问题没法直接解决，所以使用迭代的方法解决该问题。

对此，我们采用 EM 算法对该问题进行求解

EM 算法的理论原理和内容不再过多赘述，针对该问题，使用 EM 算法解决该问题的原理如下：

我们已知：

$$p(y\mid\theta)=\prod_{j=1}^{n}\left[s_1\pi^{y_j}(1-\pi)^{1-y_j}+s_2p^{y_j}(1-p)^{1-y_j}+(1-s_1-s_2)q^{y_j}(1-q)^{1-y_j}\right]$$

设 y_i 来自硬币 A 的概率为 u_{1j}，来自硬币 B 的概率为 u_{2j}，则来自 C 的概率为 $1-u_{1j}-u_{2j}$，且 $u_{1j},u_{2j}\in\{0,1\},j=1,2,\dots n$，即参数 u 为模型的隐变量。

于是完全数据的似然函数可以表示为：

$$p(y\mid\theta)=\prod_{j=1}^{n}\left\{\left[s_1\pi^{y_j}(1-\pi)^{1-y_j}\right]^{u^{1j}}+\left[s_2p^{y_j}(1-p)^{1-y_j}\right]^{u^{2j}}+\left[(1-s_1-s_2)q^{y_j}(1-q)^{1-y_j}\right]^{(1-u_{1j}-u_{2j})}\right\}$$

相应的对数似然函数为

$$\begin{aligned}\log p(y,u\mid\theta)=\sum_{j=1}^{n}u_{1j}&\left[\log s_1+y_j\log\pi+(1-y_j)\log(1-\pi)\right]\\&+u_{2j}\left[\log s_2+y_j\log p+(1-y_j)\log(1-p)\right]\\&+(1-u_{1j}-u_{2j})\left[\log(1-s_1-s_2)+y_j\log q+(1-y_j)\log(1-q)\right]\end{aligned}$$

2.2 EM 算法的 E-Step:确定 Q 函数

因为 EM 算法是迭代算法，设第 i 次迭代的参数的估计值是 $\theta^{(i)}=\left(s_1^{(i)},s_2^{(i)},\pi^{(i)},p^{(i)},q^{(i)}\right)$，又因为隐变量 u_1 代表观测数据来自 A 的概率，隐变量 u_2 代表观测数据来自 B 的概率，所以第 $(i+1)$ 次隐变量：

$$u_{1j}^{(i+1)}=\frac{s_1\pi^{y_j}(1-\pi)^{1-y_j}}{s_1\pi^{y_j}(1-\pi)^{1-y_j}+s_2p^{y_j}(1-p)^{1-y_j}+(1-s_1-s_2)q^{y_j}(1-q)^{1-y_j}}$$

$$u_{2j}^{(i+1)}=\frac{s_2p^{y_j}(1-p)^{1-y_j}}{s_1\pi^{y_j}(1-\pi)^{1-y_j}+s_2p^{y_j}(1-p)^{1-y_j}+(1-s_1-s_2)q^{y_j}(1-q)^{1-y_j}}$$

显然，来自 C 的概率为

$$u_{3j}^{(i+1)} = 1 - u_{1j}^{(i+1)} - u_{2j}^{(i+1)}$$

我们直接将其结果写入公式中即可

求取 Q：

$$Q(\theta,\theta_i) = \sum_z p(z \mid y,\theta_i) \log p(y,z \mid \theta) = E_z\left[\log\left(y,z \mid \theta,\theta^{(i)}\right)\right]$$

将$u_{1j}^{(i+1)}, u_{2j}^{(i+1)}$带入则可以得到

$$\begin{aligned} Q(\theta,\theta_i) = \sum_{j=1}^{n} \{ & u_{1j}^{(i+1)}[\log s_1 + y_j \log \pi + (1-y_j)\log(1-\pi)] \\ & + u_{2j}^{(i+1)}[\log s_2 + y_j \log p + (1-y_j)\log(1-p)] \\ & + (1-u_{1j}^{(i+1)} - u_{2j}^{(i+1)})[\log(1-s_1-s_2) + y_j \log q + (1-y_j)\log(1-q)]\} \end{aligned}$$

2.3 EM 算法的 M-Step:极大化参数

得到了 Q 函数，接下来就是极大化参数，实际上就是求解一个极大化问题的解析解如下:

$$\theta^{(i+1)} = \arg\max_\theta Q\left(\theta,\theta^i\right)$$

使用Q分别对不同的参数求偏导，并令其为 0，省略求偏导解析计算过程，得到结果如下

$$s_1{}^{(i+1)} = \frac{\sum_{j=1}^{n} u_{1j}^{(i+1)}}{n}$$

$$s_2{}^{(i+1)} = \frac{\sum_{j=1}^{n} u_{2j}^{(i+1)}}{n}$$

$$\pi^{(i+1)} = \frac{\sum_{j=1}^{n} u_{1j}^{(i+1)} y_j}{\sum_{j=1}^{n} u_{1j}{}^{(i+1)}}$$

$$p^{(i+1)} = \frac{\sum_{j=1}^{n} u_{2j}^{(i+1)} y_j}{\sum_{j=1}^{n} u_{2j}{}^{(i+1)}}$$

$$q^{(i+1)} = \frac{\sum_{j=1}^{n}\left(1-u_{1j}^{(i+1)} - u_{2j}^{(i+1)}\right) y_j}{\sum_{j=1}^{n}\left(1-u_{1j}^{(i+1)} - u_{2j}^{(i+1)}\right)}$$

然后，不断重复 E-Step 和 M-Step，最终算法收敛。

算法的收敛性在这里不予证明。

三、代码实现过程

产生样本过程如下：

分别设置五个参数的既定值，然后设置样本个数，直接产生样本，代码如下

```python
def get_relusts(self, n, s1, s2, pi, p, q):
    #产生n个抛硬币的结果
    y = []
    for i in range(n):
        flag = np.random.rand(1)
        if flag < s1:
            if np.random.rand(1) < pi:
                y.append(1)
            else:
                y.append(0)
        if flag > s1 and flag < s2 +s1:
            if np.random.rand(1) < p:
                y.append(1)
            else:
                y.append(0)
        else:
            if np.random.rand(1) < q:
                y.append(1)
            else:
                y.append(0)
    return y
```

E-Step

```python
def E_step(self, y, n):
    """
    更新隐变量u
    y 样本
    n 样本个数

    """
    pi = self.params['pi'][0]  #只有一个数值 所以读[0] 不然读出来是一个列表
    p = self.params['p'][0]
    q = self.params['q'][0]
    s1 = self.params['s1'][0]
    s2 = self.params['s2'][0]

    for i in range(n):
        self.params['mu'][i] = (s1 * pow(pi, y[i]) * pow(1-pi, 1-y[i])) / \
                               (s1 * pow(pi, y[i]) * pow(1-pi, 1-y[i]) + s2 * pow(p, y[i]) * pow(1-p, 1-y[i])
                                + (1-s1-s2) * pow(q,y[i])*pow(1-q,1-y[i]))
        self.params['mu2'][i] = (s2 * pow(p, y[i]) * pow(1-p, 1-y[i])) /\
                                (s1 * pow(pi, y[i]) * pow(1-pi, 1-y[i]) + s2 * pow(p, y[i]) * pow(1-p, 1-y[i])
                                 + (1-s1-s2) * pow(q,y[i])*pow(1-q,1-y[i]))
```

M-Step

```python
def M_step(self, y, n):
    """
    更新要求解的参数
    """
    mu = self.params['mu']
    mu2 = self.params['mu2']

    self.params['s1'][0] = sum(mu) / n
    self.params['s2'][0] = sum(mu2) / n
    self.params['pi'][0] = sum([mu[i] * y[i] for i in range(n)]) / sum(mu)
    self.params['p'][0] = sum([mu2[i] * y[i] for i in range(n)]) / sum(mu2)
    self.params['q'][0] = sum([(1-mu[i]-mu2[i]) * y[i] for i in range(n)]) / \
                          sum([1-mu_i-mu2_i for mu_i,mu2_i in zip(mu,mu2)])
```

总函数，轮流调用 E-Step 和 M-Step

```python
def fit(self, y):
    """
    模型入口
    :param y: 观测样本
    :return:
    """
    n = len(y)
    self.__init_params(n)
    print(0, self.params['s1'], self.params['s2'], self.params['pi'], self.params['p'], self.params['q'])
    flag_begin = self.params['s1'][0] * self.params['pi'][0] + self.params['s2'][0] * self.params['p'][0] + (
            1 - self.params['s1'][0] - self.params['s2'][0]) * self.params['q'][0]
    print(f'begin ---{flag_begin}')
    for i in range(self.n_epoch):
        self.E_step(y, n)
        self.M_step(y, n)
        print(i+1, self.params['s1'], self.params['s2'], self.params['pi'], self.params['p'], self.params['q'])
```

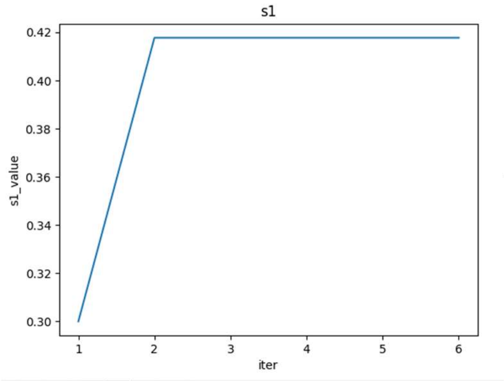
四、结果分析

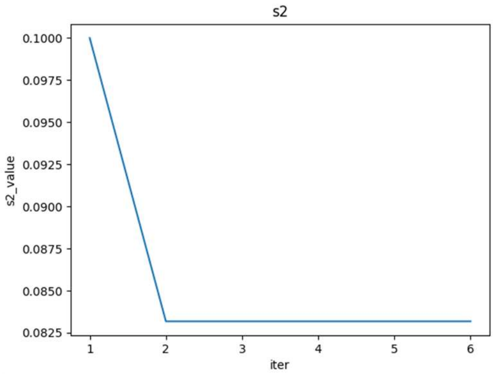
4.1 试验结果记录

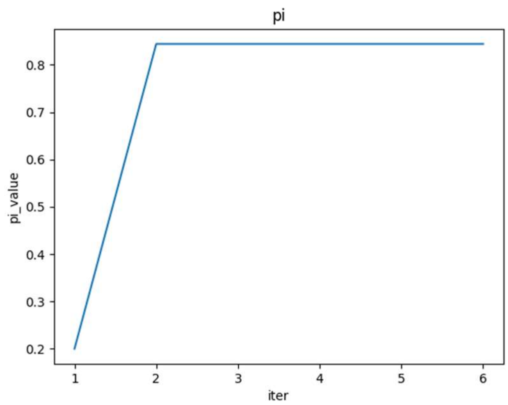
设置要求解的既定参数如下所示，其中样本个数 $n=100$

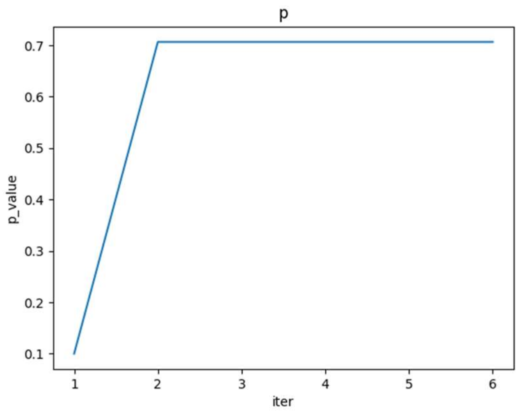
	S1	S2	pi	p	q
既定参数	0.1	0.3	0.7	0.7	0.8
初始参数	0.3	0.1	0.2	0.1	0.1
迭代次数 1	0.4177468595165398	0.08317902006906557	0.844255421109381	0.7066787003610107	0.7066787003610109
迭代次数 2	0.4177468595165393	0.08317902006906548	0.8442554211093808	0.7066787003610101	0.7066787003610118
迭代次数 3	0.41774685951653917	0.08317902006906548	0.8442554211093806	0.7066787003610094	0.7066787003610124
迭代次数 4	0.41774685951653917	0.08317902006906551	0.8442554211093806	0.7066787003610091	0.7066787003610111
迭代次数 5	0.4177468595165392	0.08317902006906551	0.8442554211093808	0.7066787003610093	0.7066787003610107

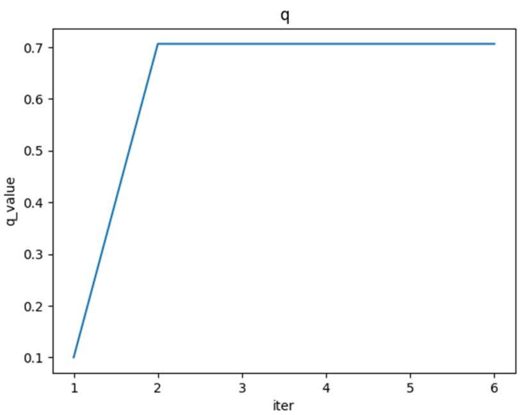
迭代过程图像如下所示











可以看出，能够很快收敛，并且后续迭代结果基本不变。

当修改初始参数时，结果如下

	S1	S2	pi	p	q
既定参数	0.1	0.3	0.7	0.7	0.8
初始参数	0.15	0.35	0.75	0.85	0.85
迭代次数 5	0.15869235554864342	0.3464207947740898	0.6291730474732006	0.7621783539579648	0.7621783539579651

当修改样本个数 $n = 10000$，即投掷硬币的样本量增大 100 倍，结果如下所示

	S1	S2	pi	p	q
既定参数	0.1	0.3	0.7	0.7	0.8
初始参数	0.15	0.35	0.75	0.85	0.85
迭代次数 5	0.15689304	0.3471616867	0.653085256	0.780505976	0.780505976

	639602233	781797	1526291	5962913	5964496

当修改迭代次数，当迭代次数为 1000 次，样本仍然保持 $n=100$，此时结果如下

	S1	S2	pi	p	q
既定参数	0.1	0.3	0.7	0.7	0.8
初始参数	0.3	0.1	0.2	0.1	0.1
迭代次数 1000	0.42271521581866256	0.08246925488304785	0.8635482030804335	0.7377192982456126	0.7377192982456147

4.2 结果分析

通过对上面写出的四组试验进行对比，除此外还测试了多组数据而没有全写在报告中，最终得出的结果分析如下

1. 对比在相同样本，不同初始迭代变量的两组试验结果可以看出，EM 算法对初始值的要求较高，但是这种对初始值的要求并不意味着 EM 算法不具备大范围收敛性，而是其具有多个稳定点，较差，距离最终标准结果较远的初始值会导致算法收敛到局部最优解(当然，这个解一定比初始值更优)
2. 对比在相同既定参数值，相同初始值，不同样本量大小的两组试验可以看出，样本量的大小会影响 EM 算法迭代结果准确性。样本量增大时，最终收敛的结果更接近既定的参数值。当然，试验中选取的样本量 100 和 10000 均可以认为是较大的样本，只是在样本数量级上有所差距，如果样本数量特别小(小样本问题)，可能将无法得出准确的结果。
3. 对比在相同样本，相同初始值，不同迭代次数的两组试验可以看出，迭代次数的增加对最后的结果有一定的影响，在收敛次数增大 200 倍之后，最终结果更接近于目标结果，但是从客观角度来讲，通过增大迭代次数对结果的改善并不显著。

五、参考资料

一篇关于 EM 算法的介绍博客

https://blog.csdn.net/weixin_41566471/article/details/106219019?spm=1001.2101.3001.6650.1&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_default&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-1.pc_relevant_default&utm_relevant_index=2

六、附录

全部代码如下所示，使用 python 实现

```
import numpy as np

import matplotlib.pyplot as plt

#np.random.seed(0)
```

```
class ThreeCoinsMode():
    def __init__(self, n_epoch=1000):
        """
        n_epoch  迭代次数
        """
        self.n_epoch = n_epoch
        self.params = {'s1': None, 's2': None, 'pi': None, 'p':None, 'q':None, 'mu': None, 'mu2': None} #迭代初始参数
        self.ele = {'s1': [1], 's2': [1], 'pi': [1], 'p': [1], 'q': [1]}#作图参数

    def __init_params(self, n):
        """
        对参数初始化操作
        :param n:  观测样本个数
        :return:
        """
        '''
        self.params = {
                        's1': np.random.rand(1),
                        's2': np.random.rand(1),
                        'pi': np.random.rand(1),
                        'p': np.random.rand(1),
                        'q': np.random.rand(1),
                        'mu': np.random.rand(n),
                        'mu2': np.random.rand(n)}
                        #随机生成初始值
        '''
        #  自定义初始值
        self.params = {
                        's1': [0.3],
                        's2': [0.1],
                        'pi': [0.2],
                        'p': [0.1],
                        'q': [0.1],
                        'mu2': np.random.rand(n),    #u 初始值没什么影响，第一个 E-Step 会直接求解 u 值
                        'mu': np.random.rand(n)}

        self.ele = {
            's1':[0 for x in range(0, self.n_epoch+1)],
            's2': [0 for x in range(0, self.n_epoch+1)],
            'pi': [0 for x in range(0, self.n_epoch+1)],
            'p': [0 for x in range(0, self.n_epoch+1)],
            'q': [0 for x in range(0, self.n_epoch+1)],
```

```
        }

def E_step(self, y, n):
        """
        更新隐变量 u
        y  样本
        n  样本个数

        """
        pi = self.params['pi'][0]    #只有一个数值  所以读[0]  不然读出来是一个列表
        p = self.params['p'][0]
        q = self.params['q'][0]
        s1 = self.params['s1'][0]
        s2 = self.params['s2'][0]

        for i in range(n):
                self.params['mu'][i] = (s1 * pow(pi, y[i]) * pow(1-pi, 1-y[i])) / \
                                                (s1 * pow(pi, y[i]) * pow(1-pi, 1-y[i]) + s2 * pow(p, y[i]) * pow(1-p, 1-y[i])
                                                 + (1-s1-s2) * pow(q,y[i])*pow(1-q,1-y[i]))
                self.params['mu2'][i] = (s2 * pow(p, y[i]) * pow(1-p, 1-y[i])) /\
                                                 (s1 * pow(pi, y[i]) * pow(1-pi, 1-y[i]) + s2 * pow(p, y[i]) * pow(1-p, 1-y[i])
                                                  + (1-s1-s2) * pow(q,y[i])*pow(1-q,1-y[i]))

def M_step(self, y, n):
        """
        更新要求解的参数
        """
        mu = self.params['mu']
        mu2 = self.params['mu2']

        self.params['s1'][0] = sum(mu) / n
        self.params['s2'][0] = sum(mu2) / n
        self.params['pi'][0] = sum([mu[i] * y[i] for i in range(n)]) / sum(mu)
        self.params['p'][0] = sum([mu2[i] * y[i] for i in range(n)]) / sum(mu2)
        self.params['q'][0] = sum([(1-mu[i]-mu2[i]) * y[i] for i in range(n)]) / \
                                    sum([1-mu_i-mu2_i for mu_i,mu2_i in zip(mu,mu2)])

def fit(self, y):
        """
        模型入口
        :param y:  观测样本
        :return:
        """
        n = len(y)
```

```
        self.__init_params(n)
        print(0, self.params['s1'], self.params['s2'], self.params['pi'], self.params['p'], self.params['q'])
        flag_begin = self.params['s1'][0] * self.params['pi'][0] + self.params['s2'][0] * self.params['p'][0] + (
                    1 - self.params['s1'][0] - self.params['s2'][0]) * self.params['q'][0]

        self.ele['s1'][0] = self.params['s1'][0]
        self.ele['s2'][0] = self.params['s2'][0]
        self.ele['pi'][0] = self.params['pi'][0]
        self.ele['p'][0] = self.params['p'][0]
        self.ele['q'][0] = self.params['q'][0]

        print(f'begin ---{flag_begin}')
        for i in range(self.n_epoch):
            self.E_step(y, n)
            self.M_step(y, n)
            print(i+1, self.params['s1'], self.params['s2'], self.params['pi'], self.params['p'], self.params['q'])
            self.ele['s1'][i+1] = self.params['s1'][0]
            self.ele['s2'][i+1] = self.params['s2'][0]
            self.ele['pi'][i+1] = self.params['pi'][0]
            self.ele['p'][i+1] = self.params['p'][0]
            self.ele['q'][i+1] = self.params['q'][0]

    def get_relusts(self, n, s1, s2, pi, p, q):
        #产生 n 个抛硬币的结果
        y = []
        for i in range(n):
            flag = np.random.rand(1)
            if flag < s1:
                if np.random.rand(1) < pi:
                    y.append(1)
                else:
                    y.append(0)
            if flag > s1 and flag < s2 +s1:
                if np.random.rand(1) < p:
                    y.append(1)
                else:
                    y.append(0)
            else:
                if np.random.rand(1) < q:
                    y.append(1)
                else:
                    y.append(0)
        return y
```

```
def run_three_coins_model():
    tcm = ThreeCoinsMode()
    s1, s2, pi, p, q = 0.7, 0.7, 0.7, 0.7, 0.8
    y = tcm.get_relusts(3, s1, s2, pi, p, q)
    tcm.fit(y)
    flag1 = s1 * pi + s2 * p + (1-s1-s2) * q
    flag2 = tcm.params['s1'][0]*tcm.params['pi'][0] + tcm.params['s2'][0]*tcm.params['p'][0] + (1 - tcm.params['s1'][0] - tcm.params['s2'][0])*tcm.params['q'][0]
    print(f'flag1 {flag1}\n')
    print(f'flag2 {flag2}')

    x_label = list(range(1, tcm.n_epoch+2))

    fig, ax = plt.subplots()
    ax.plot(x_label, tcm.ele['s1'])
    ax.set_title('s1')
    ax.set_xlabel('iter')
    ax.set_ylabel('s1_value')

    fig, ax = plt.subplots()
    ax.plot(x_label, tcm.ele['s2'])
    ax.set_title('s2')
    ax.set_xlabel('iter')
    ax.set_ylabel('s2_value')

    fig, ax = plt.subplots()
    ax.plot(x_label, tcm.ele['pi'])
    ax.set_title('pi')
    ax.set_xlabel('iter')
    ax.set_ylabel('pi_value')

    fig, ax = plt.subplots()
    ax.plot(x_label, tcm.ele['p'])
    ax.set_title('p')
    ax.set_xlabel('iter')
    ax.set_ylabel('p_value')

    fig, ax = plt.subplots()
    ax.plot(x_label, tcm.ele['q'])
    ax.set_title('q')
    ax.set_xlabel('iter')
    ax.set_ylabel('q_value')

    plt.show()
```

```
if __name__ == '__main__':
    run_three_coins_model()
```