

1

****

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,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：



其中是第个观测结果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；是模型参数。

为了方便起见，观测数据可以表示为，隐变量可以表示为。观测数据的似然函数可以表示为



即



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：



设来自硬币A的概率为，来自硬币B的概率为，则来自C的概率为，且，即参数为模型的隐变量。

于是完全数据的似然函数可以表示为：



相应的对数似然函数为



## 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，又因为隐变量代表观测数据来自A的概率，隐变量代表观测数据来自B的概率，所以第次隐变量：





显然，来自C的概率为



我们直接将其结果写入公式中即可

求取Q：



将带入则可以得到



## 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:



使用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











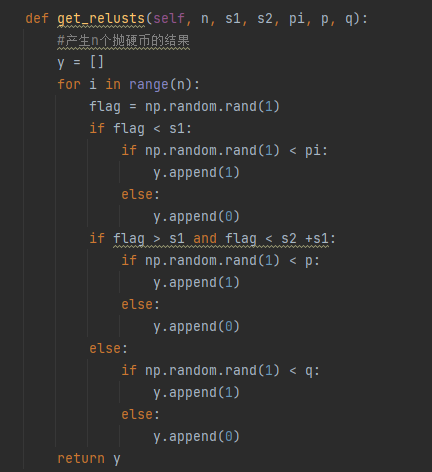
然后，不断重复E-Step和M-Step，最终算法收敛。

算法的收敛性在这里不予证明。

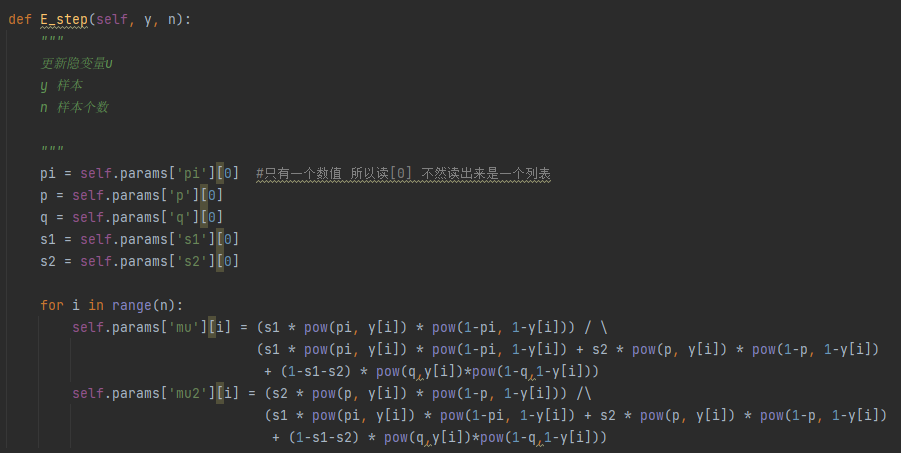
# 三、代码实现过程

产生样本过程如下：

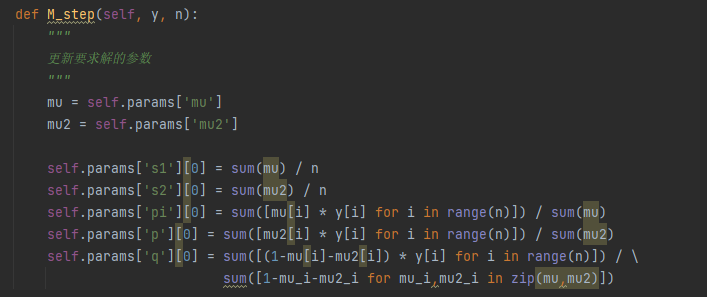
分别设置五个参数的既定值，然后设置样本个数，直接产生样本，代码如下



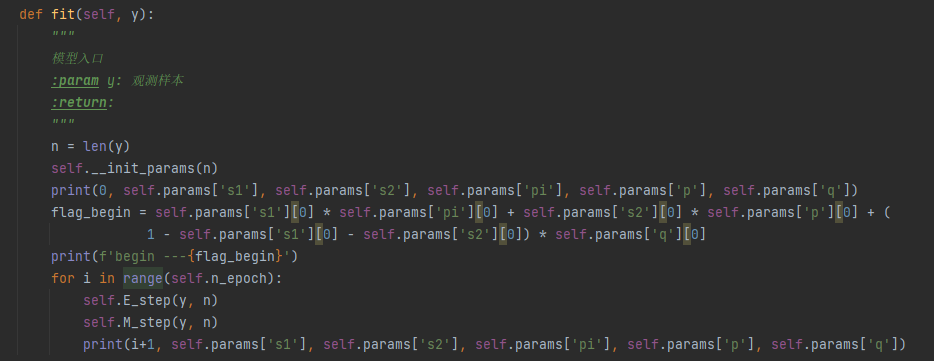
E-Step



M-Step



总函数，轮流调用E-Step和M-Step



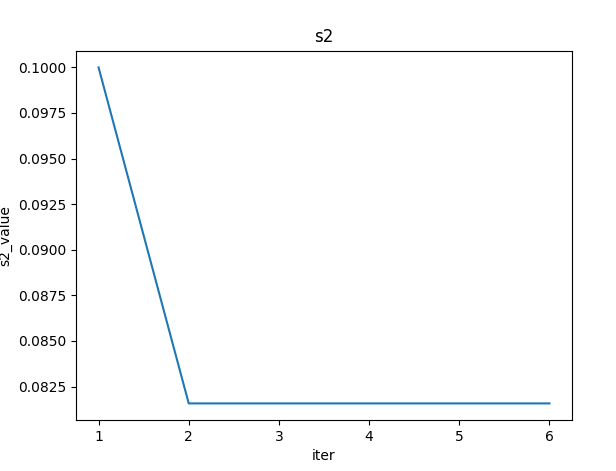
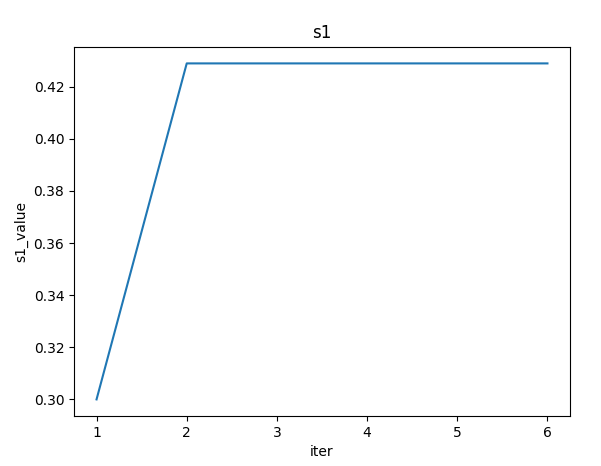
## 四、结果分析

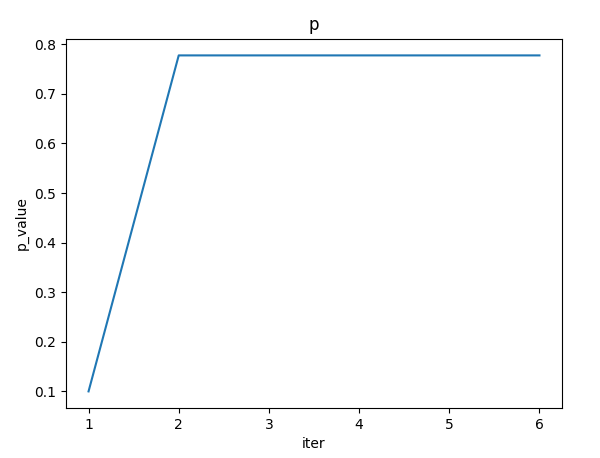
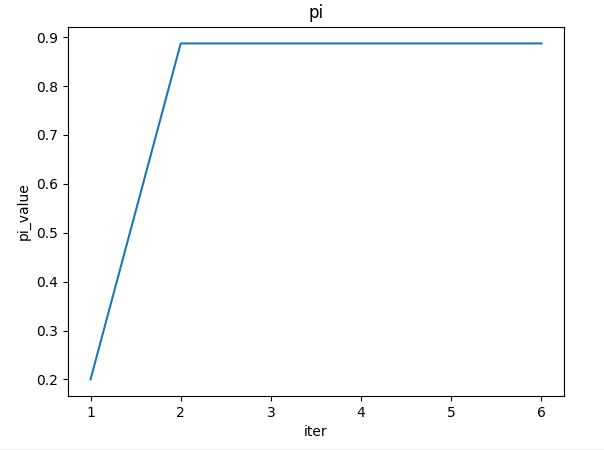
## 4.1试验结果记录

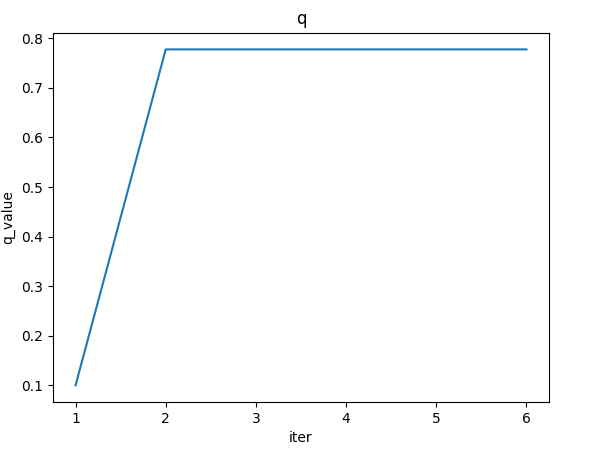
设置要求解的既定参数如下所示，其中样本个数

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | S1 | S2 | pi | p | q |
| 既定参数 | 0.7 | 0.7 | 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 | 0.40770329446109843 | 0.08461381507698582 | 0.8038189097628572 | 0.6455203116304945 | 0.6455203116304936 |
| 迭代次数2 | 0.4077032944610984 | 0.08461381507698573 | 0.8038189097628572 | 0.6455203116304952 | 0.6455203116304942 |
| 迭代次数3 | 0.4077032944610984 | 0.08461381507698547 | 0.8038189097628572 | 0.6455203116304972 | 0.6455203116304944 |
| 迭代次数4 | 0.40770329446109826 | 0.08461381507698529 | 0.8038189097628569 | 0.6455203116304977 | 0.6455203116304943 |
| 迭代次数5 | 0.40770329446109826 | 0.08461381507698525 | 0.8038189097628567 | 0.645520311630498 | 0.6455203116304945 |

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 | 0.7 | 0.7 | 0.7 | 0.7 | 0.8 |
| 初始参数 | 0.6 | 0.6 | 0.6 | 0.6 | 0.8 |
| 迭代次数5 | 0.6142779376312285 | 0.6142779376312285 | 0.7394606912267363 | 0.7394606912267363 | 0.8832936372915975 |

当修改样本个数，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 | 0.7 | 0.7 | 0.7 | 0.7 | 0.8 |
| 初始参数 | 0.6 | 0.6 | 0.6 | 0.6 | 0.8 |
| 迭代次数5 | 0.6175114466285482 | 0.6175114466285482 | 0.7701485540809666 | 0.7701485540809666 | 0.8993460474481269 |

当修改迭代次数，当迭代次数为1000次，样本仍然保持，此时结果如下

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | S1 | S2 | pi | p | q |
| 既定参数 | 0.7 | 0.7 | 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 | 0.41002131215406046 | 0.08428266969227588 | 0.8133273502355824 | 0.6594506094233286 | 0.6594506094233209 |

## 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()