NLP-2 三硬币投掷问题

罗志钊 PT2100565

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。

预备知识

极大似然估计

极大似然估计是用来估计模型参数的统计学方法。多数情况下，我们是根据已知调剂来推算结果，而极大似然估计是已知结果，寻求使该结果出现的可能性最大的条件，以此作为估计值。

求解步骤

1. 似然函数公式：

$$L(\theta)=L(x_1,x_2,\ldots,x_n;\theta)=\prod_{i=1}^{n}p(x_i;\theta),\theta\in\Theta$$

2. 对似然函数取对数：

$$l(\theta) = lnL(\theta) = ln\prod_{i=1}^{n} p(x_i;\theta) = \sum_{i=1}^{n} lnp(x_i;\theta)$$

3. 对步骤2式子求导，令导数为0，得到似然方程并求解，得到的参数即为所求。

Jensen不等式

如果f是凸函数，X是随机变量，那么： **E[f(X)] ≥ f[E(X)]**。当且仅当X是常量时，该式取等号。其中，E(X)表示X的数学期望。

Ps：Jensen不等式应用于凹函数时，不等号方向反向。当且仅当x是常量时，该不等式取等号。

EM算法

已知分布模型和随机抽取的样本，用EM算法求每个样本属于哪个分布和模型的参数。

算法流程

给定观察变量**X**、潜在联合分布 $p(\mathbf{X},\mathbf{Z}|\boldsymbol{\theta})$ 和潜在变量**Z**，由参数**θ**控制，目标是最大化似然函数相对于**θ**期望的 $p(\mathbf{X}|\boldsymbol{\theta})$ 。

1. 为参数 $\boldsymbol{\theta}^{\text{old}}$ 选择初值。
2. E step 评估 $p(\mathbf{X},\mathbf{Z}|\boldsymbol{\theta})$
3. M step 评估 由下面公式给出的 $\boldsymbol{\theta}^{\text{new}}$

   $$\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta},\boldsymbol{\theta}^{\text{old}})$$

   同时

   $$\mathcal{Q}(\boldsymbol{\theta},\boldsymbol{\theta}^{\text{old}}) = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X},\boldsymbol{\theta}^{\text{old}}) \ln p(\mathbf{X},\mathbf{Z}|\boldsymbol{\theta}).$$

4. 检查对数似然或参数值的收敛性，如果不满足收敛准则，则令

   $$\boldsymbol{\theta}^{\text{old}} \leftarrow \boldsymbol{\theta}^{\text{new}}$$

   并返回步骤2。

   ps.停止迭代的条件为 $||\boldsymbol{\theta}^{(i+1)} - \boldsymbol{\theta}^{(i)}|| < \boldsymbol{\varepsilon}_1$ 或 $||Q(\boldsymbol{\theta}^{(i+1)},\boldsymbol{\theta}^{(i)}) - Q(\boldsymbol{\theta}^{(i)},\boldsymbol{\theta}^{(i)})|| < \boldsymbol{\varepsilon}_2$ 。

问题分析

对于这个问题，难点主要有两个，其一是生成随机硬币投掷模型，然而如果直接生成，必然导致结果会是三种硬币按照比例和正面向上的均值，是无法得到三种硬币模型正反面向上的参数。其二是根据EM算法对于该问题的处理以及收敛问题的解决。

对于难点一，我选择将硬币抛掷进行分割，每组选取袋中一个硬币进行抛十次，抛n组，每次以组为单位进行处理。总抛掷次数为N=10n。

对于难点二，我按照上面的步骤对算法分解，可得

```
        # 步骤1:初始化似然函数值
        llh_old = -np.infty
        # 步骤2: E步-求隐变量分布
        # 对数似然
        log_ll = np.array([np.sum(data * np.log(theta), axis=1) for theta in esti_p])
        # 似然
        llh = np.exp(log_like)
        # 求隐变量分布
        ws = like/like.sum(0)
        # 概率加权
        vs = np.array([w[:, None] * data for w in ws])
        # 步骤3:M步
        esti_p = np.array([v.sum(0)/v.sum() for v in vs])
        # 更新似然函数
        llh_new = np.sum([w*l for w, l in zip(ws, log_like)])
```

如果满足步骤4的收敛条件则结束迭代

```
        # 步骤4:收敛判断
        if np.abs(llh_new - llh_old) < eps:
            break
        llh_old = llh_new
```

实验过程

1.

参数	赋值
s1	0.3
s2	0.4
p	0.7
q	0.5
r	0.3

对题设参数进行赋值

2. 根据我设置三种硬币的比例和正反面，随机生成50组抛掷样本，每组10次，共计500次抛掷

```python
# 生成随机投掷的函数
def random_pick(some_list, probabilities):
    x = random.uniform(0,1)
    cumulative_probability = 0.0
    for item, item_probability in zip(some_list, probabilities):
         cumulative_probability += item_probability
         if x < cumulative_probability:
               break
    return item
```

`somelist` 有两种，其一是硬币列表，分别对应硬币a，硬币b，硬币c；其二是不同种硬币的正反面。

利用该函数随机生成50组抛掷

```python
    for i in range(0, 50):
        coin.append(random_pick(coin_list, probabilities))
    # print(coin)

    # print(len(coin))
    for i in range(0, len(coin)):
        coin_single = []
        for j in range(0, 10):
            # 硬币的概率coin_sin_pro[coin_list[i]]
            coin_single.append(random_pick(coin_ht, coin_single_pro[coin[i]]))
            # print(coin_single)
        coin_dic.append(coin_single)
    #print(coin_dic)
```

生成的50组抛掷部分结果如下，其余内容放入附录1中。

```
[[1, 0, 0, 1, 1, 0, 1, 1, 1, 0],
 [1, 1, 0, 1, 1, 1, 0, 0, 0, 0],
          ......
 [0, 1, 0, 1, 0, 0, 1, 0, 1, 1]]
```

同时，为了进一步便于对数据的处理，我将每一组的10次抛掷转化成正反面出现次数的键值对（每一对的第一个元素代表正面朝上出现的次数）

```
[(6, 4), (5, 5), (8, 2), (4, 6), (1, 9), (5, 5), (3, 7), (9, 1), (5, 5), (4, 6),
(4, 6), (7, 3), (4, 6), (5, 5), (5, 5), (7, 3), (7, 3), (3, 7), (3, 7), (1, 9), (1,
9), (5, 5), (6, 4), (7, 3), (3, 7), (5, 5), (1, 9), (7, 3), (8, 2), (8, 2), (3, 7),
(8, 2), (4, 6), (3, 7), (9, 1), (7, 3), (8, 2), (1, 9), (5, 5), (8, 2), (8, 2), (6,
4), (1, 9), (3, 7), (5, 5), (4, 6), (5, 5), (3, 7), (8, 2), (5, 5)]
```

3. 设定初始化参数值，预先假定硬币a的正面概率为0.8，硬币B的正面概率为0.5，硬币C的正面概率为0.4（实际上硬币a的正面概率为0.5，硬币B的正面概率为0.5，硬币C的正面概率为0.3）
4. 利用EM算法对数据进行处理，逐次迭代并收敛，可得实验迭代流程：

```
Iteration: 1
A = 0.76,B = 0.47, C = 0.35, likelihood = -319.52
Iteration: 2
A = 0.74,B = 0.47, C = 0.32, likelihood = -315.79
Iteration: 3
A = 0.73,B = 0.47, C = 0.31, likelihood = -314.74
Iteration: 4
A = 0.73,B = 0.47, C = 0.30, likelihood = -314.34
Iteration: 5
A = 0.73,B = 0.48, C = 0.29, likelihood = -314.14
Iteration: 6
A = 0.72,B = 0.48, C = 0.29, likelihood = -314.04
Iteration: 7
A = 0.72,B = 0.48, C = 0.29, likelihood = -313.98
Iteration: 8
A = 0.72,B = 0.49, C = 0.29, likelihood = -313.96
Iteration: 9
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.96
Iteration: 10
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.96
Iteration: 11
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.97
Iteration: 12
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.98
```

实验结果

迭代33次后，likelihood收敛为-314.03。三种硬币的正反面概率分布收敛为：

硬币a正面概率为0.71744366，硬币b正面概率为0.49647873，硬币c正面概率为0.28187633。

```
Iteration: 33
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
[[0.71744366 0.28255634]
 [0.49647873 0.50352127]
 [0.28187633 0.71812367]]

Process finished with exit code 0
```

与之前假设的三个参数对比：

硬币类型	假设正面朝上的概率	实际正面朝上的概率	EM算法收敛的正面朝上的概率
A	0.8	0.7	0.71744366
B	0.5	0.5	0.49647873
C	0.4	0.3	0.28187633

迭代步骤放入附录中。

结论及心得

1. 本实验中，投掷次数极为重要。如果投掷次数选择较少，得出的实验结果波动相当大。原因有两点。其一是每组投掷十次，次数较少，无法逼近正确的硬币正反面概率；其二组数有限，EM算法逼近的值也依赖于此。
2. EM算法对初始值比较敏感。聚类结果随初始值波动大。EM算法很大程度取决于初始参数。例如硬币B的收敛概率就相当逼近正确值，而硬币a和硬币c的收敛概率就和正确值有一定距离。

   部分论文对此也有研究，例如Blömer J, Bujna K. Simple methods for initializing the EM algorithm for Gaussian mixture models[J]. CoRR, 2013.他们提出了基于著名的 K-means++ 算法和 Gonzalez 算法的新初始化方法。这些方法缩小了简单统一初始化技术和复杂方法之间的差距。
3. 关于EM算法收敛，他的收敛结果是基于数据的局部最优的算法，而不是全局极大值点。除本次结果实验外，我也随机生成了其他100次、200次及500次投掷实验，而部分收敛结果与预期相差甚远，结论1中揭示了部分原因。所以并不是只要收敛就一定是正确结果，也是需要看数据量和真实性。

参考内容

1. EM算法详解 - Microstrong的文章 - 知乎 https://zhuanlan.zhihu.com/p/40991784
2. https://github.com/luwill/Machine_Learning_Code_Implementation/tree/master/charpter22_EM

附录

1 抛掷结果

```
[[1, 0, 0, 1, 1, 0, 1, 1, 1, 0],
 [1, 1, 0, 1, 1, 1, 0, 0, 0, 0],
 [1, 1, 1, 1, 0, 0, 1, 1, 1, 1],
 [0, 1, 0, 0, 0, 1, 1, 0, 0, 1],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [1, 0, 1, 0, 1, 0, 0, 1, 1, 0],
 [0, 0, 0, 0, 0, 1, 1, 0, 1, 0],
 [1, 1, 1, 1, 1, 1, 0, 1, 1, 1],
 [0, 0, 1, 1, 0, 0, 1, 1, 1, 0],
 [1, 1, 0, 1, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 1, 1, 1, 1, 0, 0, 0],
 [1, 1, 1, 0, 0, 0, 1, 1, 1, 1],
 [0, 1, 0, 0, 0, 0, 0, 1, 1, 1],
 [0, 1, 0, 0, 1, 1, 1, 0, 0, 1],
 [1, 1, 0, 0, 0, 1, 0, 1, 1, 0],
```

```
 [1, 0, 1, 1, 0, 1, 1, 1, 1, 0],
 [1, 0, 1, 1, 1, 0, 1, 1, 0, 1],
 [0, 0, 0, 0, 0, 1, 0, 1, 0, 1],
 [1, 1, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 1, 0, 1, 0, 1, 0, 1, 0],
 [0, 0, 1, 1, 1, 1, 1, 1, 0, 0],
 [1, 0, 0, 1, 1, 1, 1, 1, 1, 0],
 [0, 0, 0, 0, 1, 0, 0, 1, 0, 1],
 [0, 0, 0, 0, 1, 1, 1, 1, 0, 1],
 [1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 1, 1, 1, 0, 1, 1, 1],
 [1, 1, 1, 1, 1, 0, 1, 0, 1, 1],
 [1, 1, 0, 1, 1, 1, 1, 0, 1, 1],
 [0, 0, 0, 0, 0, 1, 1, 0, 0, 1],
 [1, 1, 1, 1, 1, 1, 0, 1, 0, 1],
 [0, 0, 0, 1, 1, 1, 0, 0, 1, 0],
 [0, 0, 1, 0, 0, 0, 0, 1, 1, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
 [1, 0, 0, 1, 1, 1, 1, 0, 1, 1],
 [1, 1, 1, 1, 1, 1, 0, 1, 1, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
 [0, 1, 1, 1, 0, 0, 0, 0, 1, 1],
 [1, 1, 1, 0, 1, 1, 1, 0, 1, 1],
 [1, 1, 1, 1, 1, 0, 1, 0, 1, 1],
 [1, 0, 0, 0, 1, 1, 1, 1, 0, 1],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 1, 1, 0],
 [0, 1, 1, 0, 1, 0, 0, 1, 0, 1],
 [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
 [0, 0, 0, 1, 1, 1, 1, 0, 0, 1],
 [0, 0, 0, 1, 0, 1, 1, 0, 0, 0],
 [0, 1, 1, 1, 1, 1, 1, 1, 1, 0],
 [0, 1, 0, 1, 0, 0, 1, 0, 1, 1]]
```

2 迭代内容

```
Iteration: 1
A = 0.76,B = 0.47, C = 0.35, likelihood = -319.52
Iteration: 2
A = 0.74,B = 0.47, C = 0.32, likelihood = -315.79
Iteration: 3
A = 0.73,B = 0.47, C = 0.31, likelihood = -314.74
Iteration: 4
A = 0.73,B = 0.47, C = 0.30, likelihood = -314.34
Iteration: 5
A = 0.73,B = 0.48, C = 0.29, likelihood = -314.14
```

```
Iteration: 6
A = 0.72,B = 0.48, C = 0.29, likelihood = -314.04
Iteration: 7
A = 0.72,B = 0.48, C = 0.29, likelihood = -313.98
Iteration: 8
A = 0.72,B = 0.49, C = 0.29, likelihood = -313.96
Iteration: 9
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.96
Iteration: 10
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.96
Iteration: 11
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.97
Iteration: 12
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.98
Iteration: 13
A = 0.72,B = 0.49, C = 0.28, likelihood = -313.99
Iteration: 14
A = 0.72,B = 0.49, C = 0.28, likelihood = -314.00
Iteration: 15
A = 0.72,B = 0.49, C = 0.28, likelihood = -314.00
Iteration: 16
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.01
Iteration: 17
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.01
Iteration: 18
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.02
Iteration: 19
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.02
Iteration: 20
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.02
Iteration: 21
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.02
Iteration: 22
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 23
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 24
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 25
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 26
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 27
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 28
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 29
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 30
```

```
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 31
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 32
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
Iteration: 33
A = 0.72,B = 0.50, C = 0.28, likelihood = -314.03
```

3 实验代码

```python
# 导入numpy库和random库
import numpy as np
import random

# EM算法过程函数定义
def em(data, esti_p, max_iter, eps=1e-3):
    '''
    输入:
    data: 观测数据
    esti_p: 初始化的估计参数值
    max_iter: 最大迭代次数
    eps: 收敛阈值
    输出:
    esti_p: 估计参数
    '''
    # 初始化似然函数值
    llh_old = -np.infty
    for i in range(max_iter):
        # E步: 求隐变量分布
        # 对数似然 [coin_num, exp_num]
        log_like = np.array([np.sum(data * np.log(theta), axis=1) for theta in esti_p])
        # 似然 [coin_num, exp_num]
        like = np.exp(log_like)
        # 求隐变量分布 [coin_num, exp_num]
        ws = like/like.sum(0)
        # 概率加权
        vs = np.array([w[:, None] * data for w in ws])
        # M步: 更新参数值
        esti_p = np.array([v.sum(0)/v.sum() for v in vs])
        # 更新似然函数
        llh_new = np.sum([w*l for w, l in zip(ws, log_like)])
        print("Iteration: %d" % (i+1))
        print("A = %.2f,B = %.2f, C = %.2f, likelihood = %.2f"
              % (esti_p[0,0], esti_p[1,0],esti_p[2,0], llh_new))
        # 满足迭代条件即退出迭代
        if np.abs(llh_new - llh_old) < eps:
            break
        llh_old = llh_new
```

```python
    return esti_p
# 生成随机投掷的函数
def random_pick(some_list, probabilities):
    x = random.uniform(0,1)
    cumulative_probability = 0.0
    for item, item_probability in zip(some_list, probabilities):
         cumulative_probability += item_probability
         if x < cumulative_probability:
               break
    return item

if __name__ == "__main__":
    # coin_list: 硬币列表, 0代表硬币a, 1代表硬币b, 2代表硬币c
    coin_list = [0, 1, 2]
    # probabilities: 代表硬币混合比例, s1=0.3, s2=0.4,1-s1-s2=0.3
    probabilities = [0.3, 0.4, 0.3]
    # coin_ht:代表硬币的正反面, 0代表反面, 1代表正面
    coin_ht = [0, 1]
    # coin_single_pro:代表硬币正反面朝上的概率模型; 硬币a正面朝上的概率0.7, 硬币b正面朝上的概率为
0.5, 硬币c正面朝上的概率为0.3
    coin_single_pro = [[0.3, 0.7], [0.5, 0.5], [0.7, 0.3]]
    # observed:代表n组扔硬币, 每组挑一个一个硬币扔十次, 共计10n次; 这里我选取扔50组, 共计500次投掷
    observed=[]
    # coin:代表每组挑出的硬币类型, 0、1、2分别对应硬币a, 硬币b, 硬币c
    coin = []
    # coin_dic: 代表投掷次数的所有结果, 以01表示, 0代表反面, 1代表正面
    coin_dic = []

    # print(random_pick(coin_list,probabilities))

    for i in range(0, 50):
        coin.append(random_pick(coin_list, probabilities))
    # print(coin)

    # print(len(coin))
    for i in range(0, len(coin)):
        coin_single = []
        for j in range(0, 10):
            # 硬币的概率coin_sin_pro[coin_list[i]]
            coin_single.append(random_pick(coin_ht, coin_single_pro[coin[i]]))
            # print(coin_single)
        coin_dic.append(coin_single)
    #print(coin_dic)

    for i in range(0, 50):
        num = 0
        for j in range(0, 10):
            num = num + coin_dic[i][j]
        observed.append((num, 10 - num))
```

```
        #print(num)
    print("Coin toss list: ")
    print(coin_dic)
    # 观测数据，n次独立试验，每次试验10次抛掷的正反次数
    # 比如第一次试验为5次正面5次反面
    observed_data = np.array(observed)
    # 初始化参数值，这是一个猜测值，在这里我假设硬币a的正面概率为0.8，硬币B的正面概率为0.5，硬币C的正
面概率为0.4
    esti_p = np.array([[0.8, 0.2], [0.5, 0.5],[0.4, 0.6]])
    esti_p = em(observed_data, esti_p, max_iter=500, eps=1e-4)
    print(esti_p)
```