形状, 圆圈

描述已自动生成

学 期 2021-2022（2）

**卡通人物

中度可信度描述已自动生成**

**深度学习与自然语言处理**

**第二次大作业**

EM算法

|  |  |
| --- | --- |
| 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 | 孙茗逸 |
| 学号 | ZY2103113 |
| 指导老师 | 秦曾昌 |

2022年 4月

### 1 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。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 EM算法

EM算法是一种迭代优化策略，由于它的计算方法中每一次迭代都分两步，其中一个为期望步（E步），另一个为极大步（M步），所以算法被称为EM算法（Expectation-Maximization Algorithm）。EM算法受到缺失思想影响，最初是为了解决数据缺失情况下的参数估计问题，其算法基础和收敛有效性等问题在Dempster、Laird和Rubin三人于1977年所做的文章《Maximum likelihood from incomplete data via the EM algorithm》中给出了详细的阐述。其基本思想是：首先根据己经给出的观测数据，估计出模型参数的值；然后再依据上一步估计出的参数值估计缺失数据的值，再根据估计出的缺失数据加上之前己经观测到的数据重新再对参数值进行估计，然后反复迭代，直至最后收敛，迭代结束。

### 3算法流程

输入：观察到的数据x=（x1,x2...xn），联合分布 p(x,z;Θ)，条件分布p(x|z,Θ) ，最大迭代次数J。

(1)随机初始化模型参数Θ的初值，Θ0

(2)j=1,2,...,J 开始EM算法迭代：

E步：计算联合分布的条件概率期望：

M步：极大化，得到

若已经收敛，则算法结束，否则继续迭代E步和M步。

输出：模型参数Θ

### 4 公式推导

设初始参数构成为，共M枚硬币（由这三类硬币构成），每枚硬币投掷N次，则投掷硬币的结果可表示为一个M行N列的矩阵result，可表示为：

根据已知参数，可以推断出：

各个种类的硬币掷出的条件分布概率为：

联合三类硬币出现的概率，可以得到投掷硬币结果的联合分布概率：

由此，可以估算出三种硬币的概率为：

再对结果进行加权处理并进行数学运算，可得：

那么，每类硬币是正面的概率分布为：

### 5 实验过程

5.1 生成投掷数据

投掷数据由M行N列构成，即共有M个硬币，每个硬币投掷N次，并按照给定概率进行数据生成。生成数据的参数如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | s1 | s2 | p | q | r |
| 数值 | 0.4 | 0.4 | 0.3 | 0.3 | 0.8 |

5.2 数据初始化

对数据迭代进行初始化操作。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 参数 | s1 | s2 | p | q | r | Epochs |
| 数值 | 0.1 | 0.1 | 0.5 | 0.5 | 0.5 | 50 |

5.3 EM算法流程

E步：首先计算联合分布概率概率，再计算条件分布概率，最终计算出三类硬币的分布概率

M步：对参数进行极大似然估计，实现对参数的更新。

### 6 实验结果

最终结果如下表所示：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | s1 | s2 | p | q | r |
| 真实数值 | 0.4 | 0.4 | 0.3 | 0.3 | 0.8 |
| 预测数值 | 0.405 | 0.405 | 0.2999 | 0.2999 | 0.8014 |

收敛效果较好，但EM算法高度依赖初值的选择，当将初值选择为：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | s1 | s2 | p | q | r |
| 数值 | 0.1 | 0.7 | 0.1 | 0.1 | 0.1 |

最终结果如下表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 参数 | s1 | s2 | p | q | r |
| 真实数值 | 0.4 | 0.4 | 0.3 | 0.3 | 0.8 |
| 预测数值 | 0.195 | 0.4595 | 0.7974 | 0.3013 | 0.2973 |

分析：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

### 7代码

import random

import numpy as np

import matplotlib.pyplot as plt

def judge\_coin\_kind(rand):

if 0 <= rand < num\_prob[0] \* 100:

return 0

elif num\_prob[0] \* 100 <= rand < (num\_prob[1] + num\_prob[0]) \* 100:

return 1

else:

return 2

def judge\_coin\_pos\_or\_neg(kind, rand):

if 0 <= rand < pos\_pron[kind] \* 100:

return 1

else:

return 0

num\_prob = [0.4, 0.4, 0.2]

pos\_pron = [0.3, 0.3, 0.8]

N = 1000 # 硬币的个数

M = 100 # 每个硬币投掷的次数

results = [] # 记录抛掷的结果，M列N行

epochs = 50 # 迭代轮数

init\_parameters = [0.1, 0.7, 0.1, 0.1, 0.1] # 种类一二比例、种类一二三为正概率

process\_parameters = [init\_parameters]

for coin in range(N):

# 第n个硬币

result = []

rand\_coin = random.randint(0, 99)

coin\_kind = judge\_coin\_kind(rand\_coin) # 返回当前硬币的种类

for throw in range(M):

# 第n个硬币投掷M次

rand\_result = random.randint(0, 99)

pos\_or\_neg = judge\_coin\_pos\_or\_neg(coin\_kind, rand\_result) # 返回硬币的正反面

result.append(pos\_or\_neg)

results.append(result)

# 开始迭代

for epoch in range(epochs):

u1\_num=np.power(init\_parameters[2],np.sum(results,1))\* np.power(1 - init\_parameters[2],M - np.sum(results, 1)) \* init\_parameters[0]

den = np.power(init\_parameters[2], np.sum(results, 1)) \* np.power(1 - init\_parameters[2],M - np.sum(results, 1)) \* init\_parameters[0] + np.power(init\_parameters[3], np.sum(results, 1)) \* np.power(1-init\_parameters[3],M-np.sum(results,1))\*init\_parameters[1]+np.power(init\_parameters[4], np.sum(results, 1)) \* np.power(1 - init\_parameters[4], M - np.sum(results, 1)) \* (1 - init\_parameters[0] - init\_parameters[1])

u2\_num=np.power(init\_parameters[3], np.sum(results, 1)) \* np.power(1 - init\_parameters[3],

M - np.sum(results, 1)) \* init\_parameters[1]

u1 = u1\_num / den

u2 = u2\_num / den

u3 = 1 - u1 - u2

init\_parameters[0] = sum(u1) / N

init\_parameters[1] = sum(u2) / N

init\_parameters[2] = sum(u1 \* np.sum(results, 1)) / sum(u1 \* M)

init\_parameters[3] = sum(u2 \* np.sum(results, 1)) / sum(u2 \* M)

init\_parameters[4] = sum(u3 \* np.sum(results, 1)) / sum(u3 \* M)

process\_parameters.append(init\_parameters)

x = np.linspace(0, epochs, epochs + 1)

process\_parameters = np.array(process\_parameters).T

print(process\_parameters)