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算法在硬币上的应用

|  |  |
| --- | --- |
| 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 | 熊方书 |
| 学号 | ZY2103309 |
| 指导老师 | 秦曾昌 |

2022年 4月

# 引言

## 问题描述

第 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。

## EM算法简介

**1.2.1 简介**

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）。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。

**1.2.2 问题分析与推导**

假设袋子中有三种硬币分为A，B，C三类，其数量所占袋中比例分别为。并设三种硬币投掷为正面的概率分别为。观察序列为，，则：

设为第k类硬币的后验概率则最大似然估计函数为：

设则：

## 流程



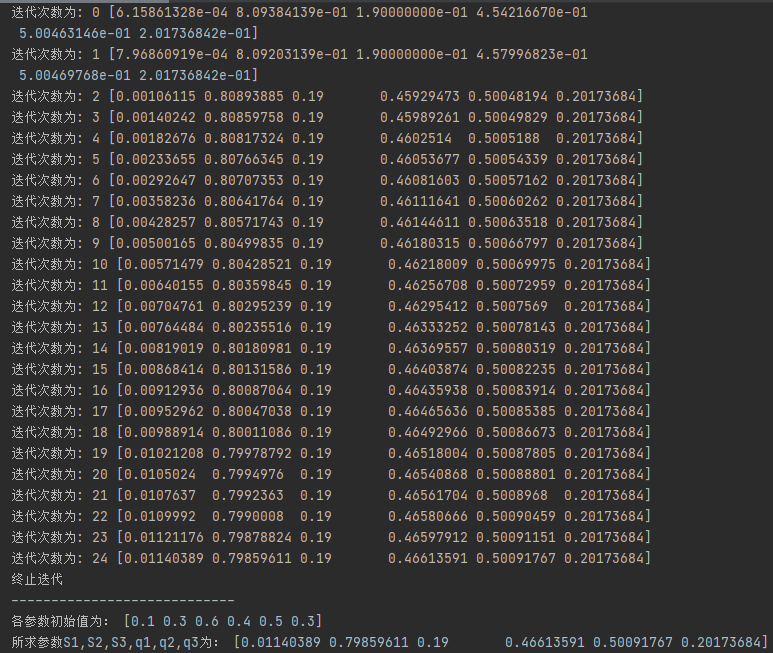
## 代码设计

使用Python语言完成算法和整个实验流程。代码输入输出如下所示：

详细代码见附录。

# 结果分析与总结

## 实验结果



## 实验分析

由上图可见，经过24次迭代后，EM算法收敛，与生成数据的各项参数相当。但是EW算法缺点明显，对初始值较为敏感，且为局部最优算法，并不是全局最优算法。

# 3代码

import numpy as np

def creat\_seq(S1,S2,S3,q1,q2,q3,H,n=300):

x\_seq = np.zeros((n,H))

for i in range(n):

coin\_divide = np.random.random()

if coin\_divide<S1:

coin = 'A'

if S1<coin\_divide<S1+S2:

coin = 'B'

else:

coin = 'C'

for j in range(H):

coin\_filp=np.random.random()

if coin == 'A' and coin\_filp<S1:

x\_seq[i,j]=1

if coin == 'B' and coin\_filp < S2:

x\_seq[i, j] = 1

if coin == 'C' and coin\_filp < S3:

x\_seq[i, j] = 1

return x\_seq

def EM(theta,x\_seq):

S1,S2,S3,q1,q2,q3 = theta[0],theta[1],theta[2],theta[3],theta[4],theta[5]

H,n = np.shape(x\_seq)[1],np.shape(x\_seq)[0]

number=0

theta\_1=np.mat(np.array([S1,S2,S3,q1,q2,q3])).T

while number<300:

h1 = ((q1 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q1) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S1) / ((q1 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q1) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S1 + (q2 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q2) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S2 + (q3 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q3) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S3)

h2 = ((q2 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q2) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S2) / ((q1 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q1) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S1 + (q2 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q2) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S2 + (q3 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q3) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S3)

h3 = ((q3 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q3) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S3) / ((q1 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q1) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S1 + (q2 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q2) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S2 + (q3 \*\* np.sum(x\_seq, axis=1, keepdims=True)) \* ((1 - q3) \*\* (H - np.sum(x\_seq, axis=1, keepdims=True))) \* S3)

S1 = np.sum(h1)/n

S2 = np.sum(h2)/n

S3 = 1-S1-S2

q1 = np.sum(h1 \* np.sum(x\_seq, axis=1, keepdims=True))/np.sum(h1\*H)

q2 = np.sum(h2 \* np.sum(x\_seq, axis=1, keepdims=True))/np.sum(h2\*H)

q3 = np.sum(h3 \* np.sum(x\_seq, axis=1, keepdims=True))/np.sum(h3\*H)

theta\_tmp=np.array([S1,S2,S3,q1,q2,q3])

print('迭代次数为:',number,theta\_tmp)

theta\_2=np.mat(theta\_tmp).T

tmp=theta\_2-theta\_1

if tmp.T\*tmp<0.0000001:

print('终止迭代')

break

else:

theta\_1 = theta\_2

number+=1

return theta\_tmp

S1,S2,S3,q1,q2,q3=0.3,0.2,0.5,0.6,0.5,0.7

x\_seq=creat\_seq(S1,S2,S3,q1,q2,q3,1000,300)

theta\_init=np.array([0.1,0.3,0.6,0.4,0.5,0.3])

theta\_outcome=EM(theta\_init,x\_seq)

print("----------------------------")

print('各参数初始值为：',theta\_init)

print("所求参数S1,S2,S3,q1,q2,q3为：",theta\_outcome)